RELATIONAL COST ANALYSIS

A dissertation submitted towards the degree

Doctor of Engineering

of the

Faculty of Mathematics and Computer Science

of

Saarland University

by

EZGI ÇIÇEK

Saarbrücken, January 2018

Day of Colloquium:	22/01/2018
Dean of the Faculty:	Prof. Dr. Frank-Olaf Schreyer
Chair of the Committee:	Prof. Dr. Jan Reinecke
Reporters:	Dr. Deepak Garg
	Prof. Dr. Amal Ahmed
	Prof. Dr. Gert Smolka
Academic Assistant:	Dr. Marco Patrignani

Dedicated to the loving memories of
my grandfather
Ali Ulvi Çiçek
&
my great-grandfather
Hacı Burhan Deniz

ABSTRACT

Programming languages research has made great progress towards statically estimating the execution cost of a program. However, when one is interested in how the execution costs of two programs compare to each other (i.e., relational cost analysis), the use of unary techniques does not work well in many cases. In order to support *a relational cost analysis*, we must ultimately support reasoning about not only the executions of *a single program*, but also the executions of *two programs*, taking into account their similarities. This dissertation makes several contributions to the understanding and development of such a relational cost analysis. It shows how:

- Refinement types and effect systems can express functional and relational quantitative properties of pairs of programs, including the difference in execution costs.
- Relational cost analysis can be adapted to reason about *dynamic stability*, a measure of the update times of incremental programs as their inputs change.
- A sound and complete bidirectional type system can be developed (and implemented) for relational cost analysis.

ZUSAMMENFASSUNG

Die Programmiersprachen-Forschung hat große Fortschritte bei der statischen Einschätzung der Ausführungskosten von Programmen gemacht. Wenn man allerdings wissen möchte, wie die Ausführungskosten zweier Programme sich zueinander verhalten (relationale Kostenanalyse), funktionieren unäre Methoden in vielen Fällen nicht gut. *Eine relationale Analyse* muss insbesondere nicht nur die Ausführung *eines einzelnen Programmes* betrachten, sondern die Ausführung *beider Programme*, um Ähnlichkeiten berücksichtigen zu können. Diese Dissertation liefert mehrere Beiträge zum Verständnis und zur Entwicklung solcher relationalen Kostenanalysen. Sie zeigt:

- Refinement-Typsysteme und Effekt-System können funktional und relational qualitative Eigenschaften von Programmpaaren ausdrücken, insbesondere die Differenz der Ausführungskosten.
- Relationale Kostenanalyse kann angepasst werden, um *dynamische Stabilität* zu analysieren. Diese misst die Update-Zeit inkrementeller Programme, wenn deren Eingaben sich ändern.
- Ein korrektes und vollständiges bidirektionales Typsystem für die relationale Kostenanalyse kann entwickelt und implementiert werden.

ACKNOWLEDGMENTS

> The first rule of discovery is to have brains and good luck. The second rule of discovery is to sit tight and wait till you get a bright idea.
>
> George Pólya, How to Solve It: A New Aspect of Mathematical Method

Throughout my life, I have often felt like I was a lucky person. Yet doing a PhD, I learned that good luck and hard work are often not enough. One also needs teachers who can help you figure out at which idea to "sit on". In my case, perhaps thanks to my good luck, I met Deepak Garg who has been more than generous in sharing his bright ideas and wisdom with me. All I needed to succeed was to sit tight and wait (ehem ... work) till I graduated.

Still, sitting on ideas can take its toll on you. It is often lonely and can get frustrating. In the past years, my life would have been miserable without the support of my friends and family who were tolerant and supportive of me sitting on some weird ideas.

I would like to extend my heartfelt thanks to

- All my friends and colleagues in Saarbrücken and Kaiserslautern. You were like a family abroad.
- Deepak for being a truly amazing advisor.
- Marco Gaboardi and Gilles Barthe who were both encouraging and helpful over the past two years. Marco also read initial draft of this thesis for which I am grateful.
- Rose Hoberman for teaching me to communicate effectively.
- Umut Acar, for introducing me to MPI-SWS and showing me that what you first perceive as bad luck can sometimes be good for you.
- Soichiro Hidaka, who advised me during my internship at NII, Tokyo. His encouragement and kind words were a great source of support.
- Georg for all the love and space.
- My parents, Necmiye and Orhan, and my brother Mert Can for being there for me and cheering me up with their travel stories.

Finally, I was lucky to be granted with bad luck from time to time so that I could grow and learn to not take things granted and that failure is fuel for growth.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Software systems inevitably contain bugs. Fortunately, recent research in programming languages has produced significant advances that allow automatic detection of bugs that cause a program to crash or to produce an unintended result. With the help of formal verification systems, such as type systems or program logics, errors can be detected at compile time by proving *functional correctness properties* of a program.

However, it is not enough for a program to execute without errors. In safety- and security-critical systems, like flight control systems or cryptographic applications, programs must not only execute correctly but must also finish executing *within specified resource bounds*. Such performance issues are significant because, in addition to wasting resources, they can make programs insecure, e.g. by allowing unintended leakage of secret inputs, or they can even render programs unusable, e.g. by allowing malicious users to create denial-of-service attacks. Even in contexts where the stakes are not so high, the resource usage of programs is still important for the purposes of usability and resource allocation: a user of a mobile phone may want to know that a software update does not slow down the phone significantly or cloud computation providers may want to ensure that their users do not exceed available resource quotas. In all of these scenarios, programming resource-aware systems requires guaranteeing not only functional correctness properties but also *non-functional properties* on the resource usage of programs.

While the programming languages community has already made significant advances towards the former *functional correctness guarantees* through strong typing, good language design and sophisticated program logics, relatively less attention has been paid to expressing and verifying *resource guarantees*. As software becomes more and more vital to human life and operations, clearly, it is critical to address this issue for a more reliable and secure software ecosystem.

The broad goal of this thesis is to guide program construction and verification in such a way that critical resource and safety correctness properties are guaranteed *not only for a single program but also for a pair of programs*. This has the potential for tremendous practical impact not only in safety-critical software but also in software deployed in everyday use.

1.1 APPLICATIONS OF RELATIONAL COST ANALYSIS

One of the traditional approaches for providing guarantees on resource usage of programs is to statically analyze the amount of resources needed to run a program. Formal techniques for performing such static execution cost analysis build on extensions of classical techniques for statically reasoning about functional properties of programs and usually focus on worst-case bounds. However, almost all of these techniques are inherently *unary*, i.e. they only reason about individual executions of a single program. As we demonstrate in this thesis, many important resource properties of programs are often *relational*, i.e. they naturally talk about pairs of executions, of programs that are either identical or closely related.

This *relational* nature of reasoning about resource usage can be observed when programmers want to

a) compare the efficiency of two implementations for the same problem or of two similar problems

b) refactor a program fragment without increasing its resource usage

c) show that the execution cost of a program is independent of the secret values of its inputs, i.e. given arbitrary input values, two executions of the same program have the same execution cost (constant-time analysis in cryptography)

d) show how the execution cost of a program varies depending on changes to its inputs (stability analysis)

In all of these cases, in order to prove interesting quantitative correctness properties, one would need to reason how the cost of one execution compares to the other. To statically reason about this comparison, one would need to prove upper bounds on the *execution cost difference* between two closely-related programs or two executions of the same program with different inputs. We refer to this difference, i.e. $\mathsf{cost}(e_1) - \mathsf{cost}(e_2)$, as the *relative cost* of e_1 with respect to e_2 and we refer to this analysis as *relational cost analysis*. In general, the cost could refer to the number of evaluation steps, abstract units of execution time, or to some consumption measure of another resource.

To see how bounds on *the relative costs* could be useful in relational verification in practice, consider the scenarios discussed above. For instance, scenario a) might arise in the context of compiler optimizations

where a compiler developer may want to prove that the optimized version of a program, e', is no slower than the original program, e, i. e. establish that $\mathtt{cost}(e') \leqslant \mathtt{cost}(e)$, or equivalently, $\mathtt{cost}(e') - \mathtt{cost}(e) \leqslant 0$. A similar scenario might arise in the context of approximate computations where a programmer may need to prove that the approximate version of the program, e_a, which often sacrifices precision for the sake of efficiency, runs much faster than the original program, e, i. e. establish that $\mathtt{cost}(e_a) - \mathtt{cost}(e) \leqslant t$. Unlike scenarios a) and b), where we have slightly different programs, there could also be scenarios like c) and d) where the two programs are identical but their inputs differ. For instance, the scenario c) might arise in the context of cryptographic applications where a programmer may need to prove that a program doesn't have a timing leak, i. e. establish that the relative cost of two of its executions (under arbitrary secret inputs) is always zero, e. g. $\forall v_1, v_2.\ \mathtt{cost}(e[v_1/x]) - \mathtt{cost}(e[v_2/x]) = 0$. Scenario d) might arise in the context of algorithmic stability analysis where a programmer may need to establish how the execution cost of a program e varies as its inputs change, e. g. establish that $\forall v_1, v_2.\ \mathtt{cost}(e[v_1/x]) - \mathtt{cost}(e[v_2]) \leqslant t$. In all of these scenarios, determining static upper bounds on the relative costs of two closely-related expressions would be helpful.

WHY WE NEED A NEW RELATIONAL COST ANALYSIS A natural way to statically establish an upper bound on the *relative cost* of a program e_1 with respect to another program e_2 would be to first establish an upper bound on e_1's cost and a lower bound on e_2's cost, i.e., find t, k such that $\mathtt{cost}(e_1) \leqslant t$ and $k \leqslant \mathtt{cost}(e_2)$. Then, the relative cost of e_1 with respect to e_2 can be upper bounded by the difference between these upper and lower bounds, i.e., $\mathtt{cost}(e_1) - \mathtt{cost}(e_2) \leqslant t - k$.

Although combining worst- and best-case bounds as described above is a sound way of establishing relative costs of two programs, this approach has two major limitations.

First, there could be cases in which naive non-relational cost analysis is difficult or intractable, but where relational cost analysis becomes easier or tractable. For example, consider a developer updating a distributed cloud application which uses almost all available hardware resources such as memory on a single machine. Since every patch to the application potentially increases memory requirements, the developer has to ensure that the updated application does not run out of memory. One solution would be to derive a global memory bound for the updated application. However, this may be difficult or even impossible in many situations. On the other hand, a formal *relational* analysis might be able to show that *the updated application does not use more memory than*

the original one. Such an analysis could be local—if, e.g., changes have been made to the body of only one loop—and may match the intuitive soundness reasoning in the mind of the developer.

Second, even in cases where non-relational cost analysis is possible, often a naive combination of best- and worst-case bounds results in imprecise bounds. To see how imprecise the naive non-relational method can be, consider a simple program

$$\text{if } n \geqslant 0 \text{ then } f(n) \text{ else } 1$$

with a single input n, where f is a closed function with equal maximum and minimum execution cost $c(n)$ that is linear in n. Assuming that evaluation of a conditional takes 1 unit of cost, the program runs slowest with cost $c(n) + 1$ when n is non-negative and it runs fastest with cost 1 when n is negative. What can we say about the relative cost of two runs of this program? Although one may naturally answer that the relative cost is simply bounded by the difference in the worst- and best-case executions costs of the two runs, i.e. $c(n)$, the precise answer depends on the values assigned to n in the two runs of the program. If we know that the two values assigned to n will not differ in the two runs, then the two executions would follow the same path and the difference in their execution cost would be 0, not $c(n)$. A non-relational analysis based on best- and worst-case execution times cannot establish this 0 cost, whereas a *relational* analysis, which takes into account the fact that n is the same in the two runs, may.

Instead of using existing unary analyses, which are not well-suited for relational verification as demonstrated by the aforementioned problems, *a relational analysis* is needed. Before we describe our proposed method of conducting such a relational cost analysis, there are several desired properties that one might expect from such a relational cost analysis, which are worth pointing out upfront:

- **Similarity/dependency tracking** In order to obtain precise bounds, the analysis should track potential similarities/dependencies between the inputs as well as the program code.
- **Size and cost sensitivity** The execution cost of a program often depends on the sizes of its inputs. Therefore, a relational cost analysis should be able to track the sizes of the program's inputs statically.
- **Precision** A static analysis cannot, due to over-approximation, always achieve the same level of precision as a careful manual analysis. However, the underlying language to express the bounds on

the execution costs should be expressive enough to obtain tight bounds whenever possible.

- **Support for local reasoning** The analysis should not require more information than necessary. For instance, to find the relative cost of two programs that differ slightly, it should be possible to only focus on parts of the programs that differ.

- **Reliability and safety** At a bare minimum, we would like to have the assurance that the bounds obtained by our analysis are indeed asymptotically upper bounds on the actual relative costs of the two programs, i. e., the analysis is sound with respect to a cost semantics.

- **Verifiability** It should be fairly easy for programmers to apply the analysis—at least given their hunch about the bound. This goal is vital to the practicality of the approach.

In this thesis, we propose a *relational cost analysis* that meets all of these criteria.

1.2 CONTRIBUTIONS

The notions of refinement types and effect systems—the main tools we use in this thesis—have been around for at least four decades. As one of the main contributions of this thesis, we convincingly demonstrate the use of refinement type and effect systems *in a relational setting*, i. e., to reason about functional and *quantitative* properties of *a pair of programs*, and we show that this approach is practical and can be applied to non-trivial domains like incremental computations.[1]

In particular, this dissertation makes the following contributions:

- We present *relational cost analysis*, a new type-based verification of how the execution cost of one program compares to another, possibly similar program. We show how *refinement types* and *type and effect systems* can be combined to statically verify precise bounds on the differences on the execution costs of a pair of programs.

- We demonstrate how the *relational cost analysis* can be adapted to reason about *dynamic stability*—a measure of the *update times of incremental programs* as their inputs change. Apart from showing the applicability of relational cost analysis to a seemingly unrelated setting with a different, more complex evaluation semantics, our incremental complexity analysis provides a high-level type-based

[1] *For an overview of incremental computations, see Section 7.1.*

verification mechanism for reasoning about dynamic stability of incremental programs. Prior to our approach, reasoning about dynamic stability required tedious direct analysis of cost semantics.

- We design and implement *a bidirectional typechecking* technique for relational cost analysis. Through a series of benchmarks, we evaluate our analysis to demonstrate that relational cost analysis can be used to verify a variety of functional and quantitative safety and correctness properties of programs from different areas such as cryptography, incremental computations and algorithm analysis, while imposing a low annotation burden on the programmer.

1.3 THESIS OUTLINE

The rest of the thesis is broken into five parts.

PART I (RELCOST) Chapter 3 starts with an informal, example-driven overview of relational cost analysis in the context of Cost^{ML}, a high-level, functional programming language which is a subset of ML. The language is compact enough to keep proofs and definitions readable but expressive enough to type non-trivial programs from various domains. This chapter introduces RelCost, a relational type and effect system through examples.

Chapter 4 presents RelCost's type and effect system. For RelCost's soundness, the execution cost of RelCost programs are formalized in a parametric way that allows for a wide range of cost metrics. To this end, Section 4.1 defines a big-step operational semantics that is parametrized with execution cost metrics.

Chapter 5 proves RelCost sound with respect to this cost semantics by developing an abstract semantic model combining step-indexed binary and unary logical relations for relational and non-relational reasoning about cost.

Chapter 6 discusses related work.

PART II(DUCOSTIT) Chapter 7 demonstrates how we can adapt the relational cost analysis technique to the setting of incremental computation in order to reason about update times of incremental programs. It starts with a review of incremental computation. Then it provides an informal, example-driven overview of dynamic stability analysis in the context of Cost^{ML} by introducing a type and effect system called DuCostIt that is similar to RelCost, but it has a different underlying semantic model that is geared towards incremental computation.

Chapter 8 presents DuCostIt's type and effect system. For DuCostIt's soundness, in addition to the standard evaluation semantics, Section 8.1 defines an abstract change-propagation semantics, modeling incremental evaluation under arbitrary changes to inputs of a program.

Chapter 9 proves DuCostIt sound with respect to the from-scratch and change propagation cost semantics by developing an abstract semantic model combining step-indexed binary and unary logical relations for relational and non-relational reasoning about cost.

Chapter 10 discusses related work.

PART III (BIRELCOST) To demonstrate that all the power that comes with RelCost's rich type system can be used in practice, Chapter 11 presents BiRelCost, a bidirectional algorithmic type system for RelCost.

To show BiRelCost sound *and* complete with respect to RelCost, we follow a two-step approach: 1) embedding of RelCost into an intermediate language, RelCost Core, and 2) algorithmic type checking of RelCost Core.

Chapter 12 first discusses several aspects of RelCost's type system—such as non-syntax directed rules and relational subtyping—that make it hard to algorithmize. Then, the chapter describes how these obstacles can be circumvented by describing an embedding from RelCost to RelCost Core, an intermediate language that has only type-directed rules and no relational subtyping. We use the embedding to argue that our bidirectional type checking is complete up to non-determinism.

Chapter 13 describe an algorithmic type system for RelCost Core. We rely on bidirectionality, which allows us to type check with very few type annotations. We call our bidirectional system BiRelCost and we discuss aspects of BiRelCost that differs from existing bidirectional typecheckers. Section 13.2 proves that the algorithmic type system is sound and complete w.r.t. the type system of RelCost Core.

Chapter 14 presents a prototype implementation of BiRelCost which combines the two steps from RelCost to RelCost Core and from RelCost Core to BiRelCost. The implementation reduces the problem of typechecking to constraint satisfaction in SMT and makes use of a few heuristics that eliminate the non-determinism inherent in RelCost's typing rules. Section 14.4 uses this implementation to demonstrate the precision and generality of the approach by typechecking several example programs ranging over compiler optimizations, security and algorithmic stability.

Chapter 15 discusses related work.

PART IV (EPILOGUE) The thesis concludes with Chapter 16, which summarizes the contributions and discusses several directions for future work.

PART V (APPENDIX) Appendices A to C contain the proofs of the necessary lemmas and theorems for RelCost, DuCostIt and BiRelCost, respectively. Appendix D contains additional lemmas and example programs considered throughout the thesis.

1.4 ASPECTS OF RELATIONAL COST ANALYSIS NOT COVERED IN THE THESIS

Although the broad theme of this thesis is relational cost analysis and its applications, certain aspects of relational cost analysis are not covered in the thesis. In order to establish the scope of the thesis, we list these out-of-scope topics below.

- *Inference of relational cost bounds:* The thesis does not cover how the relational cost bounds can be *inferred* automatically. Instead, we focus on typechecking.
- *Realizability of incremental update times:* We do not describe how the algorithmic change propagation technique, described in Section 8.1, can be implemented. Zoe Paraskevopoulou's Master's thesis [85] describes a concrete change propagation technique that can be implemented.

1.5 PREVIOUSLY PUBLISHED MATERIAL

This thesis is partly based on the work and the writing in the following papers:

[1] Ezgi Çiçek, Deepak Garg, and Umut A. Acar. "Refinement Types for Incremental Computational Complexity." In: *Programming Languages and Systems - 24th European Symposium on Programming, ESOP 2015, London, UK, April 11-18, 2015. Proceedings*. 2015, pp. 406–431.

[2] Ezgi Çiçek, Zoe Paraskevopoulou, and Deepak Garg. "A Type Theory for Incremental Computational Complexity With Control Flow Changes." In: *Proceedings of the 21st International Conference on Functional Programming*. ICFP 2016. Nara, Japan, 2016.

[3] Ezgi Çiçek, Gilles Barthe, Marco Gaboardi, Deepak Garg, and Jan Hoffmann. "Relational Cost Analysis." In: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*. POPL 2017. ACM, 2017, pp. 316–329.

2

METHODOLOGY

▶ SYNOPSIS In this chapter, we discuss two necessary ingredients for specifying and enforcing relational cost bounds: *type and effect systems* and *refinement types*. These two techniques are particularly well-suited for establishing quantitative properties of programs and form the basis of our work. Before we explain how these techniques can be adapted to reason about relational properties such as relational cost and dynamic stability bounds, we first provide a background for readers that are unfamiliar with them.

2.1 CAPTURING COMPUTATION VIA TYPE AND EFFECT SYSTEMS

Type systems are one of the most lightweight static analysis techniques: by capturing properties of sets of values, types enable programmers to reason about the functional correctness of their programs. However, most often one is interested in not only what the program computes but also *how* the program computes. That is where type and effect systems come into play: by capturing the side effects that occur during the computation, type and effect systems can enable programmers to reason about sophisticated functional *and* quantitative correctness properties of programs.

Type and effect systems are usually formalized by the judgment $\Gamma \vdash e : \tau, \epsilon$ where ϵ is an effect term (hence it is simply called *effect*). Informally, the judgment can be read as: under the context Γ, which contains the type declarations of the program's free variables, the program e yields a value of type τ *and* during the computation the program may have the effect ϵ.[2] The meaning of the effect ϵ changes depending on the kind of the computational property the effect captures. For instance, in the context of memory management, where data is allocated on the heap per region, one may want to statically track the set of regions that are allocated during the evaluation of the program. Then, such a type and effect system can be used to ensure that no memory accesses, apart from the ones specified in ϵ (e. g., to unallocated or deallocated regions), occur at runtime. Similarly, in the context of a language with exceptions, one may want to statically track the set of exceptions raised during the execution of the program. Then, such

[2] *We assume a call-by-value language here. For a call-by-name language, the context Γ contains type and effect assumptions of the form $x :_{\epsilon_i} \tau_i$ since variables may be substituted by expressions that may themselves have effects.*

a type and effect system can be used to ensure that a program typed with $\epsilon = \emptyset$ would have no uncaught exceptions. In these two examples, the effect can be modeled by a set of events observed during the computation. However, the effect doesn't necessarily have to be a set: e. g., in the context of cost analysis, the effect could capture the number of evaluation steps or in the context of incremental computations, the effect could capture the size of the evaluation trace.

Techniques based on type and effect systems have several strengths compared to other static analysis techniques for verification (e. g. based on program logics or abstract interpretation). First, type and effect systems work well with higher-order functions by means of effect annotations on function types. (Consequently, the approach naturally supports separate compilation) Second, type and effect systems are a lightweight method for verification, i. e. they are more amenable to adoption in practice by programmers. Compared to program logics that often require domain-specific experts to prove the desired properties, type and effect systems are usually equipped with type inference and typechecking mechanisms that reduce the burden on the programmer. Finally, by supporting a rigorous analysis early in the development process, the approach enables the development of correct-by-construction programs, reducing the need for extensive testing at runtime.

Even though type and effect systems are well-suited for modeling and verifying many interesting computational behaviors, in order to model relational cost analysis, existing type and effect systems do not suffice due to two reasons.

First of all, existing type and effect systems are often unary, i. e., they reason about a single program, whereas relational cost analysis requires us to reason about a pair of programs. In this thesis, we demonstrate how effect systems can be generalized to the relational setting.

Secondly, the bounds on relative costs, as well as execution costs, usually depend on some properties of input values of programs which cannot be captured by simple types. These properties themselves could be unary, e. g., capturing the sizes of the program's inputs, or they could be relational, e. g., capturing how two inputs of a program differ from each other. Next, to describe such unary and relational dependencies to program values, we discuss a complementary type-theoretic approach to increase expressivity of type and effect systems, namely *refinement types*.

2.2 MIDDLE GROUND ON DEPENDENCY: REFINEMENT TYPES

One of the fundamental ways of modeling dependency on program values is *dependent types*. Dependent types allow enriching the type information so that types can refer not only to other types but also to *values*. For instance, the usual function type $A \to B$ is generalized to $\Pi x : A.B$ so that type B of the return value may vary depending on its argument $x : A$.[3]. For a short introduction to dependent types, see [22].

[3] *If x doesn't occur in B, we have the usual simple function type $A \to B$*

Dependent types enable programmers to express more program properties than what is possible with conventional type systems like ML and Haskell. Properties that can be expressed by dependent types could provide functional guarantees (e. g. showing that quicksort produces a sorted list) as well as resource guarantees (e. g. showing that a program doesn't have memory leaks). Moreover, such properties can be relational. In fact, full-fledged dependent types can express most properties which we know how to define mathematically.

However, this tremendous expressivity comes at a steep price: typechecking dependent types is extremely complicated and undecidable in general. This is because dependent types remove the isolation between values and types that exists in simply-typed languages, hence lifting the general halting problem to typechecking.

Instead, researchers have designed restricted forms of dependent types by sacrificing some of the expressivity of dependent types for the sake of decidable and low-complexity typechecking. Such systems are often called "lightweight dependent types" or "refinement types'.[4].

[4] *There is no consensus on the actual nomenclature: see [1] for a detailed discussion on the difference between refinement and indexed types.*

A prominent example of refinement types is DML, which extends ML with indexed types [106]. The main idea behind DML is to bring back the isolation between values and types: by allowing types to only refer to index terms which are separate from DML expressions, one can create a phase distinction between typechecking and execution of a program. Hence, while retaining additional expressivity, typechecking is reduced to constraint satisfaction over the index terms, which is often decidable.

DML's type refinements are mostly in the form of indexed types where list and tree types are indexed with the number of elements, although they also consider richer refinements like the height of trees. Crucially, the type system ensures that constructors and destructors preserve the size information and pattern matching is index-dependent. DML has demonstrated that–even using only lightweight refinements, dependent types are powerful to statically prove the absence of notorious bugs such as array index out of bounds errors.

Motivated by the success of DML and recent use of refinement types in mainstream languages (like LiquidHaskell [104]), in this thesis, we build on ideas from refinement types. In particular, for relational cost analysis, we employ DML-style unary index refinements to express not only unary dependencies, such as input sizes, but also relational dependencies, such as the number of elements that differ between the two lists.

Part I

RELCOST

3

RELCOST BY EXAMPLES

▶ SYNOPSIS Recall that the goal of relational cost analysis is to derive an upper bound on the *difference* in the execution costs of two programs, say e_1 and e_2. This difference, i.e. $\mathtt{cost}(e_1) - \mathtt{cost}(e_2)$, is also called the *relative cost* of e_1 with respect to e_2. When e_1 and e_2 have the forms $\mathtt{f}\ e_1'$ and $\mathtt{f}\ e_2'$, the same analysis can be used to determine how the cost of a function $\mathtt{f}$ varies with the argument.

In this chapter, we introduce a relational refinement type and effect system for relational cost analysis. We first give an overview of RelCost's type system and then present some of its features through examples.

TWO-LAYERED TYPING Precise relational cost analysis requires understanding which expressions and values may be related. RelCost's types (shown in Figure 1) make this explicit by syntactically (as well as semantically) separating the relational types τ from the non-relational ones A: the former represent a pair of *related* values (expressions), capturing the similarities between them, whereas the latter represent individual values (expressions). For instance, the unary type int represents integer values whereas the relational type int_r represents pairs of identical integer values. In general, any non-relational type can be trivially made relational by encapsulating it within the weakest relation using the $\mathtt{U}\,\cdot$ modality. For instance, the type $\mathtt{U}\,\mathrm{int}$ represents a pair of unrelated integers whereas the type int_r represents a pair of identical integers.[5]

Corresponding to these two layers of types, there are two typing judgments in RelCost. The *unary* typing judgment has the form $\Omega \vdash^{t}_{k} e : A$, where k and t are lower and upper bounds on the execution cost of e under the unary (non-relational) typing environment Ω. The *relational* typing judgment has the form $\Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau$, where t is an upper bound on the relative cost of e_1 with respect to e_2 under the relational typing environment Γ. Relational typing aims to benefit from the similarities between the inputs and the programs, whereas unary typing considers a single program and a single input in isolation.

[5] $\mathtt{U}\,\cdot$ *is generalized to* $\mathtt{U}\,(A_1, A_2)$*, which relates pairs of arbitrary values of different types* A_1 *and* A_2*.*

SIZE AND COST REFINEMENTS RelCost makes use of two kinds of type refinements: size refinements and cost refinements.

Unary types	A	$::=$	$\mathsf{int} \mid A_1 \times A_2 \mid A_1 + A_2 \mid \mathsf{list}[n]\, A \mid$ $A_1 \xrightarrow{\mathsf{exec}(k,t)} A_2 \mid \forall i \overset{\mathsf{exec}(k,t)}{::} S.\, A \mid$ $\exists i{::}S.\, A \mid C \,\&\, A \mid C \supset A \mid \mathsf{unit}$
Relational types	τ	$::=$	$\mathsf{int}_r \mid \tau_1 \times \tau_2 \mid \tau_1 + \tau_2 \mid \mathsf{list}[n]^{\alpha}\, \tau \mid$ $\tau_1 \xrightarrow{\mathsf{diff}(t)} \tau_2 \mid \forall i \overset{\mathsf{diff}(t)}{::} S.\, \tau \mid \exists i{::}S.\, \tau \mid$ $C \,\&\, \tau \mid C \supset \tau \mid \mathsf{unit}_r \mid U\, A \mid \Box \tau$
Sorts	S	$::=$	$\mathbb{N} \mid \mathbb{R}$
Index terms	$I, k,$ t, α	$::=$	$i \mid 0 \mid \infty \mid I + 1 \mid I_1 + I_2 \mid I_1 - I_2 \mid$ $\frac{I_1}{I_2} \mid I_1 \cdot I_2 \mid \lceil I \rceil \mid \lfloor I \rfloor \mid \mathtt{log}_2(I) \mid$ $I_1^{I_2} \mid \sum_{i=I_1}^{I_n} I \mid \mathtt{min}(I_1, I_2) \mid \mathtt{max}(I_1, I_2)$
Constraints	C	$::=$	$I_1 \doteq I_2 \mid I_1 < I_2 \mid \neg\, C$
Constraint env.	Φ	$::=$	$\top \mid C \wedge \Phi$
Sort env.	Δ	$::=$	$\emptyset \mid \Delta, i :: S$
Unary type env.	Ω	$::=$	$\emptyset \mid \Omega, x : A$
Relat. type env.	Γ	$::=$	$\emptyset \mid \Gamma, x : \tau$
Primitive env.	Υ	$::=$	$\emptyset \mid \Upsilon, \zeta : \tau_1 \xrightarrow{\mathsf{diff}(t)} \tau_2 \mid$ $\Upsilon, \zeta : A_1 \xrightarrow{\mathsf{exec}(k,t)} A_2$

Figure 1: Syntax of RelCost's types

First, since the execution cost of a program often depends on the sizes of its inputs, unary list types are refined to the form $\mathsf{list}[n]\, A$, where n is the exact length of the list. Relational list types are refined to the form $\mathsf{list}[n]^{\alpha}\, \tau$, which represents a pair of lists, both of length exactly n and that differ in at most α positions.[6] To statically deal with the remaining $n - \alpha$ elements that are not allowed to differ between the two lists, RelCost introduces the comonadic type $\Box \tau$, representing the diagonal relation that relates only *syntactically equal* values (expressions). The type $\mathsf{list}[n]^{\alpha}\, \tau$ is interpreted such that at most α elements of the two lists are of type τ and at least $n - \alpha$ elements are of type $\Box \tau$, i.e. identical.

[6] *Note that n in $\mathsf{list}[n]\, A$ is a unary refinement whereas the n and α in $\mathsf{list}[n]^{\alpha}\, \tau$ are relational refinements, relating a pair of list values.*

Second, worst-/best-case execution costs and relative costs are treated as unary and relational *effects*, respectively. The standard function type

$A_1 \to A_2$ is refined to the corresponding unary type $A_1 \xrightarrow{\text{exec}(k,t)} A_2$, which carries k and t, the minimum and maximum execution costs of the function body. Similarly, the type $\tau_1 \to \tau_2$ is refined to the relational type $\tau_1 \xrightarrow{\text{diff}(t)} \tau_2$, which carries t, the maximum relative cost of the bodies of the two functions (given related arguments of type τ_1). Similar cost annotations are written on universally quantified types, capturing the costs of their closures.

EXAMPLE 1 (CONDITIONAL RECONSIDERED) Coming back to the example from Chapter 1, let us see how the relative cost of two runs of

$$e = \text{if } n \geqslant 0 \text{ then } f(n) \text{ else } 1$$

can be established in RelCost. If n is not allowed to differ in the two runs, i.e., it has type int_r, then the two runs of e can be typed relationally with relative cost 0:

$$n : \text{int}_r \vdash e \ominus e \lesssim 0 : \text{int}_r \tag{1}$$

The intuition behind this typing is that since the two runs take the same execution path, it suffices to relationally type the two branches $f(n) \ominus f(n)$ and $1 \ominus 1$ component-wise, i.e. *synchronously*. Both of these branches have 0 relative cost and int_r result type, so the two runs of e can be typed as shown above in (eq. (1)).

In contrast, if the value of n may differ between the two runs, i.e. $n : U\,\text{int}$, then these programs can be typed with cost $c(n)$:

$$n : U\,\text{int} \vdash e \ominus e \lesssim c(n) : U\,\text{int} \tag{2}$$

In this case, since the two executions might take different paths, we lose the relational reasoning. In order to establish an upper bound on their relative cost, we need to switch to a worst- and best-case execution cost comparison. In the type system, this is achieved by using the following switch rule:

$$\frac{|\Gamma| \vdash^{t_1}_{\_} e_1 : A \qquad |\Gamma| \vdash^{\_}_{k_2} e_2 : A}{\Gamma \vdash e_1 \ominus e_2 \lesssim t_1 - k_2 : U\,A}\ \textbf{switch}$$

where e_1 and e_2 are two arbitrary programs that are typed independently with maximum execution cost t_1 and minimum execution cost k_2, respectively.[7] (Note that the premises are unary judgments, while the conclusion is relational). Then the relative cost of e_1 with respect

[7] *The generalized version of* ***switch*** *for* $U(A_1, A_2)$ *is shown in Figure 10 (in Chapter 4).*

to e_2 is upper bounded by $t_1 - k_2$. Since the execution costs of e_1 and e_2 are independent of their relation, we can type them with a non-relational environment $|\Gamma|$ obtained from Γ by ignoring the relations for each type, e.g., $|\mathrm{int}_r| = |U\,\mathrm{int}| = \mathrm{int}$.

Using this rule, we can type e independently once with maximum execution cost $c(n) + 1$ and once with minimum execution cost 1 and obtain the typing in eq. (2). Note that because n is unrelated in the two runs, any computation that depends on it must be unrelated as well. Hence, the result type is also unrelated, i.e. $U\,\mathrm{int}$.

EXAMPLE 2 (CONSTANT-TIME COMPARISON) In cryptographic applications, it is often necessary to prove that a program is *constant-time*, i.e., its execution time is independent of secret inputs, to prevent an attacker from inferring the secret inputs by measuring the execution time. Using relational cost analysis, we can prove that a program is constant time without separately proving that its worst- and best-case execution costs are equal (as would be necessary if we used non-relational cost analysis). For example, consider the following constant-time comparison function `comp` that checks the equality of two passwords represented as equal-length lists of bits.

$$
\begin{array}{l}
\mathtt{fix}\ \mathtt{comp}(l_1, l_2).\mathtt{case}\ l_1\ \mathtt{of} \\
\quad \mathrm{nil} \ \to \mathrm{true} \\
\mid h_1 :: tl_1 \to \ \mathtt{case}\ l_2\ \mathtt{of}\ \ \mathrm{nil} \ \to \mathrm{false} \\
\qquad\qquad\qquad\qquad\qquad \mid h_2 :: tl_2 \ \to\ \mathtt{boolAnd}\ \langle \mathtt{comp}\ \langle tl_1, tl_2\rangle, \mathtt{eq}\ \langle h_1, h_2\rangle\rangle
\end{array}
$$

Suppose that the function `boolAnd` returns the conjunction of the two boolean values in constant-time; it has type[8]

$$\mathtt{boolAnd} \ominus \mathtt{boolAnd} \lesssim 0 : (U\,\mathrm{bool} \times U\,\mathrm{bool}) \xrightarrow{\mathrm{diff}(0)} U\,\mathrm{bool}$$

and that the function `eq` checks integer equality in constant-time; it has type

$$\mathtt{eq} \ominus \mathtt{eq} \lesssim 0 : (U\,\mathrm{int} \times U\,\mathrm{int}) \xrightarrow{\mathrm{diff}(0)} U\,\mathrm{bool}\ .$$

We can now show that the function `comp` is *constant-time* by typing it as follows:

$$
\begin{array}{l}
\vdash \mathtt{comp} \ominus \mathtt{comp} \lesssim 0 : \forall n, \alpha, \beta{::}\mathbb{N}. \\
\qquad\qquad (\mathrm{list}[n]^{\alpha}\, U\,\mathrm{int} \times \mathrm{list}[n]^{\beta}\, U\,\mathrm{int}) \xrightarrow{\mathrm{diff}(0)} U\,\mathrm{bool}.
\end{array}
$$

[8] *The function* `boolAnd` *can be defined and typed in our language, but we assume* `eq` *to be a primitive function.*

The annotation on $\xrightarrow{\text{diff}(0)}$ means that the relative cost of two runs of the function is 0 and, here, the universal quantification over α, β means that this relative cost holds no matter how much the lists differ.[9] The proof of this judgment proceeds by induction on the input lists (via a typing rule for fixpoints). The interesting case is when the two lists have at least one element each. Inductively, we know that the relative cost of $\texttt{comp}\langle \texttt{tl}_1, \texttt{tl}_2\rangle$ is 0. Furthermore, we assumed that `eq` and `boolAnd` are constant-time. Therefore, we can easily conclude that `comp` is constant-time.

[9] *The expression-level introduction and elimination forms for universal and existential quantifiers such as those over n, α, β are omitted from all examples for better readability.*

Note that this proof of the relative cost of `comp` is trivial compared to a proof through a non-relational analysis that would have to establish best- and worst-case execution costs (taking into account constant factors carefully) and show that they are equal.

EXAMPLE 3 (SQUARE-AND-MULTIPLY) This example demonstrates how we can combine RelCost's relational and non-relational reasoning principles to obtain precise bounds on the relative cost of programs. Consider the square-and-multiply algorithm, a fast way of computing positive powers of a number based on the observation that $x^m = x \cdot (x^2)^{\frac{m-1}{2}}$ when m is odd, and $x^m = (x^2)^{\frac{m}{2}}$ when m is even. The following function, `sam`, implements this idea, assuming that m is stored in binary form in a list `l` of 0s and 1s, with the least significant bit at the head.

```
fix sam(x).λl.case l of
  nil  → contra
| b :: bs  → case bs of  nil  → if x = 0 then 1 else x
                        | _ :: _  →  let r = sam x bs in if b = 0 then r^2 else x · r^2
```

Assume that multiplication (as in $x \cdot r^2$) has a fixed cost t. Consider two executions of `sam` on the same base ($x : \text{int}_r$) and two exponents that differ in at most α bit positions ($l : \text{list}[n]^{\alpha}\,(U\,\text{int})$). What is the maximum relative cost of one run with respect to the other? Intuitively, the relative cost is in $O(\alpha \cdot t)$ since the two runs may enter the two *different* branches of the **if** in at most α recursive calls and the difference in the cost of the two branches is exactly one multiplication (r^2 vs $x \cdot r^2$). Hence, `sam` can be given the following type for a suitable linear function P:

$$\vdash \texttt{sam} \ominus \texttt{sam} \lesssim 0 : \text{int}_r \xrightarrow{\text{diff}(0)} \forall n > 0, \alpha{::}\mathbb{N}.\ \text{list}[n]^{\alpha}\,U\,\text{int} \xrightarrow{\text{diff}(P(\alpha\cdot t))} U\,\text{int}.$$

We explain how `sam`'s type is derived in RelCost, focusing on the branch of the case analysis that recursively calls `sam`. From `l`'s type, we know

that at most α bits differ in the two runs. However, we do not know whether b is among these α bits. Accordingly, our case analysis rule for lists, rule **r-caseL** in Figure 8, requires two sub-cases for the cons branch: either the head b differs in the two runs or it does not. In the first case, we assume that b may have different values in the two runs and $bs : \text{list}[n-1]^{\alpha-1}\,(U\,\text{int})$. The total cost $P(\alpha \cdot t)$ suffices for the recursive call's cost $P((\alpha-1)\cdot t)$ as well as t, the relative cost of the two branches of the expression $(\text{if } b = 0 \text{ then } r^2 \text{ else } x \cdot r^2)$, which is established through unary analysis of the expression and the rule **switch**. In the second case, we assume that b has the same value in the two runs and $bs : \text{list}[n-1]^{\alpha}\,(U\,\text{int})$. In this case, the two runs can differ only in the recursive call, which has an (inductive) cost of $P(\alpha \cdot t)$. Technically, the assumption that b has the same value in the two runs is represented using the relational type constructor $\Box\tau$, which is the *diagonal* sub-relation of τ, i.e., the subset of τ containing equal (not just related) values in the left and right components. Here, $b : \Box\,(U\,\text{int})$ in the second sub-case.

Note that the relative cost of `sam` obtained by taking the difference of worst- and best-case costs would be linear in n, not in α. Thus, direct relational analysis makes the reasoning more precise in this example.

EXAMPLE 4 (TWO-DIMENSIONAL COUNT) This example also demonstrates that RelCost's relational analysis can establish that one program is faster than another when a unary analysis cannot. Consider the following function `2Dcount` that counts the number of rows of a matrix M (represented as a list of lists in row-major form) that both contain a key x and satisfy a predicate p. The function takes as argument another function `find` that returns 1 when a given row l contains x, else returns 0.

```
fix 2Dcount(find).λx.λM.case M of
  nil  → 0
| l :: M′ → let r = 2Dcount find x M′ in
             if p l then r + (find x l) else r
```

Consider the following two different implementations of `find`.

```
fix find1(x).λl.case l of
  nil  → 0
| h :: tl → if h = x then 1 else find1 x tl
```

```
fix find2(x).λl.case l of
```

$$\begin{array}{l} \quad \mathtt{nil} \ \to 0 \\ \mid h :: tl \ \to \ \text{if } (\mathtt{find2}\ x\ tl) = 1 \text{ then } 1 \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{else if } (h = x) \text{ then } 1 \text{ else } 0 \end{array}$$

The function `find1` scans the row l from head to tail and returns 1 when an element matches x, whereas the function `find2` recurs to the end of l and scans it from tail to head, looking for a match. For simplicity, assume that applications cost a unit and all other operations cost nothing. We can establish that on input lists of length n, the unary cost of `find1` lies in the interval $[1, 3n]$ and that of `find2` lies in the interval $[3n, 4n]$. Hence, `find1` is never slower than `find2` and, so, the relative cost of (`2Dcount find1`) with respect to (`2Dcount find2`) is upper-bounded by 0 (assuming that the same matrix M is given to the two expressions, i.e., M has type $\mathrm{list}[m]^0\ (\mathrm{list}[n]^0\ \mathrm{int})$ for some m and n). In RelCost, this cost can be established in three steps. First, we type `2Dcount`. [10]

[10] *If the arrow $\to$ has no cost annotations, the cost is assumed to be 0, i.e. we have $\xrightarrow{\mathrm{diff}(0)}$.*

$$\vdash \mathtt{2Dcount} \ominus \mathtt{2Dcount} \lesssim 0 : \ (\mathrm{U}\,\mathrm{int} \to \forall n{::}\mathbb{N}.\, \mathrm{U}\,(\mathrm{list}[n]\,\mathrm{int}) \xrightarrow{\mathrm{diff}(0)} \mathrm{U}\,\mathrm{int}) \to \\ \mathrm{int}_r \to \forall m, n{::}\mathbb{N}.\, \mathrm{list}[m]^0\ (\mathrm{list}[n]^0\ \mathrm{int}_r) \xrightarrow{\mathrm{diff}(0)} \mathrm{U}\,\mathrm{int}$$

This type means that, given two `find` functions with relative cost 0 (first $\xrightarrow{\mathrm{diff}(0)}$ in the type above), the relative cost of `2Dcount` with those `find` functions is 0. This type is easily established by induction on M's outer list. Then, we show that the relative cost of `find1` with respect to `find2` is 0, i.e.,

$$\vdash \mathtt{find1} \ominus \mathtt{find2} \lesssim 0 : \ \mathrm{U}\,\mathrm{int} \to \forall n{::}\mathbb{N}.\, \mathrm{U}\,(\mathrm{list}[n]\,\mathrm{int}) \xrightarrow{\mathrm{diff}(0)} \mathrm{U}\,\mathrm{int}$$

This is done by establishing the best- and worst-case costs of `find1` and `find2` as outlined above (we omit the technical details). Using these two types we can immediately prove that for a fixed matrix $M : \mathrm{list}[m]^0\ (\mathrm{list}[n]^0\ \mathrm{int})$, we have

$$\vdash (\mathtt{2Dcount\ find1}\ M) \ominus (\mathtt{2Dcount\ find2}\ M) \lesssim 0 : \mathrm{U}\,\mathrm{int}$$

Importantly, this relational cost cannot be established using a naive best- and worst-case analysis. This is because the cost of the function (`2Dcount find1` M) is as high as $3nm + 7m$ when the predicate p is true on all rows of M and the element x does not appear anywhere, and the cost of (`2Dcount find2` M) is as low as $4m$ when the predicate p is false on every row. Clearly, $3nm + 7m$ is more than $4m$, so a unary cost analysis cannot establish that (`2Dcount find1` M) is always faster than (`2Dcount find2` M).

OTHER EXAMPLES Additional examples, including standard list functions (e.g. append, reverse etc), selection sort, mergesort (a divide and conquer program), an instance of approximate computation and loop unswitching (an optimizing program transformation), can be found in Appendix D.2. In Section 14.4, we describe typing of two example programs, `map` and `msort`, in detail.

4

THE RELCOST TYPE SYSTEM

▶ SYNOPSIS In this chapter, we present the technical ideas behind RelCost. We first describe RelCost's type grammar and expression syntax, then we present the underlying abstract cost semantics (Section 4.1) and explain the typing (Section 4.2) and subtyping rules (section 4.3). The design of the type system reflects the underlying semantic model, explained later in Chapter 5.

TYPES RelCost's type syntax is shown in Figure 1 and it consists of two kinds of types. Unary or non-relational types, denoted A, are ascribed to single expressions, whereas relational types, denoted τ, are ascribed to pairs of expressions. Both types contain familiar type constructors with some refinements. We explain a few salient points here. The relational base types int_r and unit_r are distinguished from their unary counterparts int and unit syntactically; for the remaining type constructors such as sums and products, we rely on the context to make this distinction clear. Similar to function types, universally quantified types are also refined with costs for the bodies of their closures.

Relational types are interpreted as sets of pairs of values whereas unary types are interpreted—as usual—as sets of values (explained in Section 5). Any pairs of unary types A_1 and A_2 can be trivially made relational using the full (weakest) relation $U(A_1, A_2)$ (read "unrelated"), that contains all pairs of values of types A_1 and A_2. When A_1 and A_2 are both equal to some A, we simply write $U\,A$ instead of $U(A, A)$. For instance, the type $U\,\text{int}$ specifies two arbitrary values of type int. In contrast, the relational type int_r ascribes only those pairs of integers where the two components are equal. The relational type $\tau_1 + \tau_2$ represents two values with the same tag: either both inl or both inr. Pairs of values of a sum type with different tags can be typed at $U(A_1 + A_2)$.

The type $\Box\,\tau$ specifies two values of type τ that are *syntactically* equal. The $\Box$ annotation is used mainly in typing list expressions, e.g., in typing related lists of type $\text{list}[n]^{\alpha}\,\tau$, where at most α elements of the two related lists are allowed to differ whereas at least $n - \alpha$ elements are assumed to be identical, i.e., of type $\Box\,\tau$. The need for $\Box$ annotation as a separate type constructor is best understood by looking at sum types. For example, $(\text{int}_r + U\,\text{int})$ contains pairs of tagged values which have the same tag but whose content can differ if the tag is inr. The stronger

type $\Box\,(\mathrm{int}_r\ +\ U\,\mathrm{int})$ forces both values to be syntactically equal and is, in fact, semantically equal to $(\mathrm{int}_r\ +\ \mathrm{int}_r)$. Technically, $\Box$ is a comonadic type: a constructive S4 necessitation modality, but with a relational interpretation.

Additionally, to represent arithmetic relations between parameters like n, α, t and k, RelCost includes two forms of constrained types. The constrained type $C\ \&\ A$ means that the *constraint* C holds and the type is A. Analogously, the constrained type $C \supset A$ means that if C holds then the type is A. The relational counterparts of constrained types are $C\ \&\ \tau$ and $C \supset \tau$ and they are defined similarly. For instance, $((1 \leqslant n)\ \&\ \mathrm{list}[n]\,A)$ specifies non-empty lists. Constraints are drawn from a rich language of predicates, explained below.

INDICES Index terms I, k, t, α are a key ingredient of RelCost's relational cost analysis (shown in Figure 1). They serve two purposes: (i) as refinements on the typing judgments and function types, they specify relative or best- and worst-case costs and (ii) as refinements on the list types, they specify the lengths of lists and the maximum number of differences between related lists. We consider index terms to be *sorted*. Index terms over list types are always interpreted over the domain $\mathbb{N}$ of natural numbers, whereas the cost terms are interpreted over the domain $\mathbb{R}$ of real numbers extended with $\{-\infty, \infty\}$. Most operations over index terms are overloaded for the sorts $\mathbb{N}$ and $\mathbb{R}$ and there is a natural subsorting from $\mathbb{N}$ to $\mathbb{R}$. The index term ∞ is used to mean that there is no guaranteed bound on the (relative) cost. The sorting judgment $\Delta \vdash I :: S$ assigns sort S to the index term I; its rules are shown in Figure 2.

CONSTRAINTS Constraints C represent predicates over index terms. They may appear in (a) constrained types like $C\ \&\ \tau$ and $C \supset \tau$, (b) assumptions Φ in typing judgments (explained below) and (c) constraint entailment in subtyping, denoted $\Delta; \Phi \models C$, and read "for any substitution for the index variables in Δ, the constraint assumptions Φ entail the constraint C".We do not stipulate syntactic rules for constraint entailment, but they are drawn from first-order logic extended with the axioms of arithmetic.

EXPRESSIONS AND VALUES The syntax of $\mathrm{Cost}^{\mathrm{ML}}$'s expressions and values is shown in Figure 3. It includes the standard introduction and elimination forms for RelCost's types. Integer constants are written n. Recursive functions are written $\mathrm{fix}\ f(x).e$. This is also written $\lambda x.e$ when f doesn't occur in e. Index variable quantification and instantiation are

$\boxed{\Delta \vdash I :: S}$

$$\frac{\Delta(i) = S}{\Delta \vdash i :: S}\ \textbf{inVar} \qquad \frac{}{\Delta \vdash 0 :: \mathbb{N}}\ \textbf{zero} \qquad \frac{}{\Delta \vdash \infty :: \mathbb{R}}\ \textbf{infinity}$$

$$\frac{\Delta \vdash I :: \mathbb{N}}{\Delta \vdash (I+1) :: \mathbb{N}}\ \textbf{plus} \qquad \frac{\Delta \vdash I :: \mathbb{R} \qquad \circ \in \{\lfloor\ \rfloor, \lceil\ \rceil\}}{\Delta \vdash (\circ S) :: \mathbb{N}}\ \textbf{op-un-N}$$

$$\frac{\Delta \vdash I_1 :: \mathbb{N} \qquad \Delta \vdash I_2 :: \mathbb{N} \qquad \diamond \in \{\mathtt{min}, \mathtt{max}, +, -, *, \div, \hat{}\}}{\Delta \vdash (I_1 \diamond I_2) :: \mathbb{N}}\ \textbf{op-bin-N}$$

$$\frac{\Delta \vdash t_1 :: \mathbb{R} \qquad \Delta \vdash t_2 :: \mathbb{R} \qquad \star \in \{\mathtt{min}, \mathtt{max}, +, -, *, /, \hat{}\}}{\Delta \vdash (t_1 \star t_2) :: \mathbb{R}}\ \textbf{op-bin-R}$$

$$\frac{\Delta \vdash t :: \mathbb{R}}{\Delta \vdash \mathtt{log}_2(t) :: \mathbb{R}}\ \textbf{op-log} \qquad \frac{\Delta \vdash I :: \mathbb{N}}{\Delta \vdash I :: \mathbb{R}}\ \textbf{i}\sqsubseteq$$

$$\frac{\Delta \vdash I_1 :: \mathbb{N} \qquad \Delta \vdash I_n :: \mathbb{N} \qquad \Delta, i :: \mathbb{N} \vdash I :: S \qquad S \in \{\mathbb{N}, \mathbb{R}\}}{\Delta \vdash \sum_{i=I_1}^{I_n} I :: S}\ \textbf{isum}$$

Figure 2: Sorting rules

denoted $\Lambda.e$ and $e[\,]$, respectively. To simplify programs that case analyze lists, index terms do not appear in expressions. Elimination forms for constrained types $C \,\&\, \tau$ and $C \supset \tau$ are written clet e_1 as x in e_2 and $\mathrm{celim}_{\supset}\ e$, respectively.

Before we explain RelCost's typing rules, we describe its abstract evaluation semantics.

4.1 ABSTRACT COST MODEL

We consider a big-step call-by-value operational cost semantics for RelCost. The evaluation judgment $e \Downarrow^{c,r} v$ states that expression e evaluates to value v. The judgment is instrumented with two cost counters, recording two independent costs: a) *reduction steps* c, which are an artifact of the proof technique we use and interact only with the step-index in our semantic model and b) *execution cost* r (tracking the resource use), which interact only with the execution (relative) cost bounds in the type system. The cost semantics is parametric in the execution costs: symbolic costs for elimination forms described below can be set externally without affecting the analysis. Apart from the costs, the evaluation rules are fairly standard and shown in Figure 4. We briefly discuss the high-level design principles behind our abstract cost semantics.

$$
\begin{array}{lrl}
\text{Expr.} & e & ::= \; x \mid n \mid \text{fix } f(x).e \mid e_1\ e_2 \mid \zeta\ e \mid \langle e_1, e_2\rangle \mid \pi_1(e) \mid \pi_2(e) \mid \\
& & \text{inl } e \mid \text{inr } e \mid \ \text{case } (e, x.e_1, y.e_2) \mid \text{nil } \mid \text{cons}(e_1, e_2) \mid \\
& & (\ \text{case } e \text{ of nil } \to e_1 \mid h :: tl \to e_2) \mid \Lambda.e \mid e[\,] \mid \\
& & \text{pack } e \mid \text{unpack } e_1 \text{ as } x \text{ in } e_2 \mid \text{let } x = e_1 \text{ in } e_2 \mid \\
& & \text{clet } e_1 \text{ as } x \text{ in } e_2 \mid \text{celim}_{\supset}\ e \mid ()
\end{array}
$$

$$
\begin{array}{lrl}
\text{Values} & v & ::= \; n \mid \text{fix } f(x).v \mid \langle v_1, v_2\rangle \mid \text{inl } v \mid \text{inr } \mid \text{nil } \mid \\
& & \text{cons}(v_1, v_2) \mid \Lambda.e \mid \text{pack } v \mid ()
\end{array}
$$

Figure 3: Syntax of terms and values

The total execution cost of an expression is the sum of the costs of its subexpressions, plus a distinct symbolic cost for the following elimination constructs: projections, pattern matches on lists *and* sum types, function applications, and let-bindings. The elimination forms for index variables and constraints do not incur any additional cost since they are usually compiled away before program's execution. All other reduction rules, including the ones for values, are assigned zero additional cost. We use metavariables like c_{app} to denote such construct-dependent elimination costs. The analysis is sound for any values of these cost metavariables as long as they are all real numbers. [11] Our cost model could be generalized by operating on a pre-ordered monoidal structure with an identity and a binary operation as well.

Like execution costs, the reduction steps follow a similar pattern with the exception that instead of symbolic steps, each aforementioned elimination incurs a unit step.

In principle, one could merge the reduction steps with the execution costs and keep track of a single cost like in the original RelCost paper [110]. However, due to the step-indexing that we use to deal with recursive functions in our semantic model, this would require recursive functions to consume some resource every time: i. e. the cost of application (c_{app}) would need to be at least 1.[12] Even though this might seem like a minor constraint, it has two drawbacks. First, such a restriction is unnecessarily prohibitive. If we were to track different resources other than execution costs (e. g., the number of network calls), we would still be forced to always incur 1 cost for applications. Second, this restriction is semantically unsatisfying: step-indexing is a technique to make the proofs of non-well-founded definitions go through, and shouldn't be integral to the semantics of the language.

[11] *Usually, the values for metavariables would be non-negative, accounting for how much resource is consumed by executing the corresponding construct. However, these values could be also negative, meaning that executing the corresponding construct produces some resources.*

[12] *To see why, see the soundness proof of fixpoints in Appendix A.2, in which applications take at least a step.*

$\boxed{e \Downarrow^{c,r} v}$ Expression e evaluates to value v with c steps and cost r.

$$\frac{}{n \Downarrow^{0,0} n}\ \textbf{const} \qquad \frac{e \Downarrow^{c,r} v}{\text{inl } e \Downarrow^{c,r} \text{inl } v}\ \textbf{inl} \qquad \frac{e \Downarrow^{c,r} v}{\text{inr } e \Downarrow^{c,r} \text{inr } v}\ \textbf{inr}$$

$$\frac{e \Downarrow^{c,r} \text{inl } v \qquad e_1[v/x] \Downarrow^{c_r,r_r} v_r}{\text{case } (e, x.e_1, y.e_2) \Downarrow^{c+c_r+1,r+r_r+c_{case}} v_r}\ \textbf{case-inl}$$

$$\frac{e \Downarrow^{c,r} \text{inr } v \qquad e_2[v/y] \Downarrow^{c_r,r_r} v_r}{\text{case } (e, x.e_1, y.e_2) \Downarrow^{c+c_r+1,r+r_r+c_{case}} v_r}\ \textbf{case-inr}$$

$$\frac{}{\text{fix } f(x).e \Downarrow^{0,0} \text{fix } f(x).e}\ \textbf{fix}$$

$$\frac{e_1 \Downarrow^{c_1,r_1} \text{fix } f(x).e \qquad e_2 \Downarrow^{c_2,r_2} v_2 \qquad e[v_2/x, (\text{fix } f(x).e)/f] \Downarrow^{c_r,r_r} v_r}{e_1\ e_2 \Downarrow^{c_1+c_2+c_r+1,r_1+r_2+r_r+c_{app}} v_r}\ \textbf{app}$$

$$\frac{e \Downarrow^{c,r} v \qquad \hat{\zeta}(v) = (c_r, r_r, v_r)}{\zeta\ e \Downarrow^{c+c_r+1,r+r_r+c_{primapp}} v_r}\ \textbf{primapp} \qquad \frac{}{\Lambda.e \Downarrow^{0,0} \Lambda.e}\ \textbf{Lam}$$

$$\frac{e \Downarrow^{c,r} \Lambda.e_b \qquad e_b \Downarrow^{c_r,r_r} v_r}{e[\,] \Downarrow^{c+c_r,r+r_r} v_r}\ \textbf{iApp} \qquad \frac{e \Downarrow^{c,r} v}{\text{pack } e \Downarrow^{c,r} \text{pack } v}\ \textbf{pack}$$

$$\frac{e_1 \Downarrow^{c_1,r_1} \text{pack } v \qquad e_2[v/x] \Downarrow^{c_2,r_2} v_r}{\text{unpack } e_1 \text{ as } x \text{ in } e_2 \Downarrow^{c_1+c_2,r_1+r_2} v_r}\ \textbf{unpack} \qquad \frac{}{\text{nil} \Downarrow^{0,0} \text{nil}}\ \textbf{nil}$$

$$\frac{e_1 \Downarrow^{c_1,r_1} v_1 \qquad e_2 \Downarrow^{c_2,r_2} v_2}{\text{cons}(e_1, e_2) \Downarrow^{c_1+c_2,r_1+r_2} \text{cons}(v_1, v_2)}\ \textbf{cons}$$

$$\frac{e \Downarrow^{c,r} \text{nil} \qquad e_1 \Downarrow^{c_r,r_r} v_r}{\text{case } e \text{ of nil } \to e_1 \mid h :: tl \to e_2 \Downarrow^{c+c_r+1,r+r_r+c_{caseL}} v_r}\ \textbf{caseL-nil}$$

$$\frac{e \Downarrow^{c,r} \text{cons}(v_1, v_2) \qquad e_2[v_1/h, v_2/tl] \Downarrow^{c_r,r_r} v_r}{\text{case } e \text{ of nil } \to e_1 \mid h :: tl \to e_2 \Downarrow^{c+c_r+1,r+r_r+c_{caseL}} v_r}\ \textbf{caseL-cons}$$

$$\frac{e_1 \Downarrow^{c_1,r_1} v_1 \qquad e_2 \Downarrow^{c_2,r_2} v_2}{\langle e_1, e_2 \rangle \Downarrow^{c_1+c_2,r_1+r_2} \langle v_1, v_2 \rangle}\ \textbf{prod} \qquad \frac{e \Downarrow^{c,r} \langle v_1, v_2 \rangle}{\pi_1(e) \Downarrow^{c+1,r+c_{proj}} v_1}\ \textbf{proj1}$$

$$\frac{e \Downarrow^{c,r} \langle v_1, v_2 \rangle}{\pi_2(e) \Downarrow^{c+1,r+c_{proj}} v_2}\ \textbf{proj2} \qquad \frac{e_1 \Downarrow^{c_1,r_1} v_1 \qquad e_2[v_1/x] \Downarrow^{c_r,r_r} v_r}{\text{let } x = e_1 \text{ in } e_2 \Downarrow^{c_1+c_r+1,r_1+r_r+c_{let}} v_r}\ \textbf{let}$$

$$\frac{e_1 \Downarrow^{c_1,r_1} v_1 \qquad e_2[v_1/x] \Downarrow^{c_r,r_r} v_r}{\text{clet } e_1 \text{ as } x \text{ in } e_2 \Downarrow^{c_1+c_r,r_1+r_r} v_r}\ \textbf{clet} \qquad \frac{e \Downarrow^{c,r} v}{\text{celim}_{\supset}\ e \Downarrow^{c,r} v}\ \textbf{celim}$$

Figure 4: RelCost's evaluation semantics

The advantage of the abstract cost metric we presented here is twofold: a) it is easy to understand and reason about and b) it nicely captures asymptotic costs. RelCost's effect system could be extended to more fine-grained metrics, if needed. Alternatively, it can be easily simplified to more coarse-grained metrics by setting the values of some of these metavariables to zero, as in some examples of Chapter 3.

4.2 TYPING JUDGMENTS

RelCost's type system contains two typing judgments. The unary judgment

$$\Delta; \Phi_a; \Omega \vdash^{t}_{k} e : A$$

states that the execution cost of e is lower bounded by k and upper bounded by t, and the expression e has the unary type A. The relational judgment

$$\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau$$

states that the relative cost of e_1 with respect to e_2 is upper bounded by t and the two expressions have the relational type τ. These typing judgments use two kinds of type environments: Ω and Γ are type environments for the unary and relational typing, respectively. Besides these, both typing judgments have two other environments: Δ for index variables and Φ_a for assumed constraints. There is also an additional global environment Υ, containing types of primitive functions, but this environment remains the same across the rules, so we don't write it explicitly. In the presentation of the typing rules, we omit premises concerning well-formedness of types, which clutter the presentation and do not provide any insights.

LOWER BOUNDS ON THE RELATIVE COST RelCost's unary judgment tracks both a lower and an upper bound on the execution cost of a program whereas RelCost's relational judgment only tracks an upper bound on the relative cost. The curious reader may wonder why we do not also track a lower bound k on the relational judgment as follows

$$\Delta; \Phi_a; \Gamma \vdash k \lesssim e_1 \ominus e_2 \lesssim t : \tau \,.$$

However, doing so is redundant because the following **swap** rule is *admissible*.

$$\frac{\Delta; \Phi_a; \Gamma \vdash k \lesssim e_1 \ominus e_2 \lesssim t : \tau}{\Delta; \Phi_a; d(\Gamma) \vdash -t \lesssim e_2 \ominus e_1 \lesssim -k : d(\tau)}\ \textbf{swap}$$

In essence, the rule states that if we can show that the relative cost of e_1 and e_2 is lower bounded by k and upper bounded by t, then we can also show that the relative cost of e_2 and e_1 is lower bounded by $-t$ and upper bounded by $-k$. Semantically, this rule follows trivially from the fact that $k \leqslant c_1 - c_2 \leqslant t$ if and only if $-t \leqslant c_2 - c_1 \leqslant -k$. Note that for this to work, relational function types must must also internalize the lower bounds on the relative cost, as in $\tau_1 \xrightarrow{\text{diff}(k,t)} \tau_2$. In addition, in the conclusion of the **swap** rule, the result type and the environment are also dualized using the type level operation $d(.)$. For instance, $d(\tau_1 \xrightarrow{\text{diff}(k,t)} \tau_2) = d(\tau_1) \xrightarrow{\text{diff}(-k,-t)} d(\tau_2)$.

Since this rule is admissible, adding lower bounds to the relational judgment is redundant: Whenever we are interested in a lower bound on $e_1 \ominus e_2$, we can instead derive an upper bound on $e_2 \ominus e_1$ and flip the sign of the bound. Hence, we do not consider the extended relational judgment with the lower bound any further.

RELCOST'S TYPING PRINCIPLES AND DESIGN CHOICES Before explaining the details of RelCost's type system, we review the general design principles behind the unary and relational typing rules.

- The total cost of an expression is obtained by summing the costs of its subexpressions. Moreover, for the unary typing, elimination constructs mentioned in Section 4.1 incur an additional symbolic cost. For relational typing, since we track the difference in the execution costs, these costs cancel out in all the rules that relate two structurally similar programs.

- In all the *synchronous* typing rules that relate two structurally similar expressions, we only allow eliminating truly related expressions that are *not* of type $U\,A$. For instance, case-elimination on $U\,(A_1 + A_2)$ cannot be typed *relationally*. All such cases are handled *uniformly*: If the eliminated expressions are unrelated, i.e., of type $U\,A$ (or generally $U\,(A_1, A_2)$), the verification can be done only by switching to non-relational typing for the whole expression. Another possibility would be to duplicate all typing rules for elimination forms that have unrelated types so that continuations would switch to non-relational reasoning. This approach is

taken in refinement type systems such as FlowCaml [93] or the published version of DuCostIt [35] but we believe our approach is cleaner (it results in fewer typing rules).

- RelCost's index refinements are a form of lightweight dependent types that enable static reasoning about runtime properties of a program. In RelCost, we choose to keep the complexity of dependencies limited in comparison to full dependent types. Richer dependencies, such as allowing index terms to be different in two related expressions, would increase the number of programs that can be relationally analyzed. However, this would also make the metatheory more difficult.

The typing rules for the unary and relational typing judgments are shown in Figures 5 and 6, and Figures 7 to 10, respectively. Below, we explain selected rules for the two judgments separately.

4.2.1 *Unary Typing*

The unary typing rules treat lower and upper bounds similarly. Values are assumed to evaluate with zero cost. So, variables (rule **var**), as well as all introduction forms including functions and index abstractions incur zero cost. For functions, the minimum and maximum costs of the body, denoted k and t respectively, are internalized into the type $A_1 \xrightarrow{\text{exec}(k,t)} A_2$ (rule **fix**). These internalized costs are technically called latent costs, as they manifest themselves when the function is applied. In the rule **app**, these internalized costs k and t are added to the total minimum and maximum execution costs of the application along with an additional symbolic cost c_{app} for the function application.

Similar to functions, for universally quantified expressions $\Lambda.e$, the minimum and maximum costs of the closure, denoted k and t respectively, are internalized into the type $\forall i \overset{\text{exec}(k,t)}{::} S.\, A$ (rule **iLam**). In the rule **iApp**, these internalized costs k and t are first substituted with a witness I and then added to the total minimum and maximum execution costs of the index term application.

Existentially quantified types are introduced using **pack** rule and eliminated using **unpack** rule.

The rule $\sqsubseteq$ **exec** allows weakening of the result type as well as the costs: An expression with minimum execution cost k and maximum execution cost t can be typed with a lower cost $k' \leqslant k$ and a higher cost $t' \geqslant t$. As usual, weakening is needed when typing a case construct

$\boxed{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e : A}$ Execution cost of e is lower bounded by k and upper bounded by t, and e has the unary type A.

$$\frac{}{\Delta;\Phi_a;\Omega \vdash^{0}_{0} n : \mathsf{int}}\ \textbf{const} \qquad \frac{\Omega(x) = A}{\Delta;\Phi_a;\Omega \vdash^{0}_{0} x : A}\ \textbf{var}$$

$$\frac{}{\Delta;\Phi_a;\Omega \vdash^{0}_{0} () : \mathsf{unit}}\ \textbf{unit} \qquad \frac{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e : A_1 \qquad \Delta \vdash^{A} A_2\ \mathtt{wf}}{\Delta;\Phi_a;\Omega \vdash^{t}_{k} \mathsf{inl}\ e : A_1 + A_2}\ \textbf{inl}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e : A_2 \qquad \Delta \vdash^{A} A_1\ \mathtt{wf}}{\Delta;\Phi_a;\Omega \vdash^{t}_{k} \mathsf{inr}\ e : A_1 + A_2}\ \textbf{inr}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Omega \vdash^{t}_{k} e : A_1 + A_2 \\ \Delta;\Phi_a;x : A_1,\Omega \vdash^{t'}_{k'} e_1 : A \\ \Delta;\Phi_a;y : A_2,\Omega \vdash^{t'}_{k'} e_2 : A\end{array}}{\Delta;\Phi_a;\Omega \vdash^{t+t'+c_{case}}_{k+k'+c_{case}} \mathsf{case}\ (e, x.e_1, y.e_2) : A}\ \textbf{case}$$

$$\frac{\Delta \vdash^{A} A_1 \xrightarrow{\mathsf{exec}(k,t)} A_2\ \mathtt{wf} \qquad \Delta;\Phi_a;x : A_1, f : A_1 \xrightarrow{\mathsf{exec}(k,t)} A_2,\Omega \vdash^{t}_{k} e : A_2}{\Delta;\Phi_a;\Omega \vdash^{0}_{0} \mathsf{fix}\ f(x).e : A_1 \xrightarrow{\mathsf{exec}(k,t)} A_2}\ \textbf{fix}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{t_1}_{k_1} e_1 : A_1 \xrightarrow{\mathsf{exec}(k,t)} A_2 \qquad \Delta;\Phi_a;\Omega \vdash^{t_2}_{k_2} e_2 : A_1}{\Delta;\Phi_a;\Omega \vdash^{t_1+t_2+t+c_{app}}_{k_1+k_2+k+c_{app}} e_1\ e_2 : A_2}\ \textbf{app}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{t_1}_{k_1} e_1 : A_1 \qquad \Delta;\Phi_a;\Omega \vdash^{t_2}_{k_2} e_2 : A_2}{\Delta;\Phi_a;\Omega \vdash^{t_1+t_2}_{k_1+k_2} \langle e_1, e_2\rangle : A_1 \times A_2}\ \textbf{prod}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e : A_1 \times A_2 \qquad i \in \{1,2\}}{\Delta;\Phi_a;\Omega \vdash^{t+c_{proj}}_{k+c_{proj}} \pi_i(e) : A_i}\ \textbf{proj}_i$$

$$\frac{\Delta \vdash^{A} A\ \mathtt{wf}}{\Delta;\Phi_a;\Omega \vdash^{0}_{0} \mathsf{nil} : \mathsf{list}[0]\,A}\ \textbf{nil}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{t_1}_{k_1} e_1 : A \qquad \Delta;\Phi_a;\Omega \vdash^{t_2}_{k_2} e_2 : \mathsf{list}[n]\,A}{\Delta;\Phi_a;\Omega \vdash^{t_1+t_2}_{k_1+k_2} \mathsf{cons}(e_1, e_2) : \mathsf{list}[n+1]\,A}\ \textbf{cons}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Omega \vdash^{t}_{k} e : \mathsf{list}[n]\,A \\ \Delta;\Phi_a \wedge n = 0;\Omega \vdash^{t'}_{k'} e_1 : A' \\ i,\Delta;\Phi_a \wedge n = i+1; h : A, tl : \mathsf{list}[i]\,A,\Omega \vdash^{t'}_{k'} e_2 : A'\end{array}}{\Delta;\Phi_a;\Omega \vdash^{t+t'+c_{caseL}}_{k+k'+c_{caseL}} \mathsf{case}\ e\ \mathsf{of}\ \mathsf{nil} \to e_1 \mid h :: tl \to e_2 : A'}\ \textbf{caseL}$$

$$\frac{i :: S,\Delta;\Phi_a;\Omega \vdash^{t}_{k} e : A \qquad i \notin \mathrm{FIV}(\Phi_a;\Omega)}{\Delta;\Phi_a;\Omega \vdash^{0}_{0} \Lambda.e : \forall i \overset{\mathsf{exec}(k,t)}{::} S.\,A}\ \textbf{iLam}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e : \forall i \overset{\mathsf{exec}(k',t')}{::} S.\,A \qquad \Delta \vdash I : S}{\Delta;\Phi_a;\Omega \vdash^{t+t'[I/i]}_{k+k'[I/i]} e[\,] : A\{I/i\}}\ \textbf{iApp}$$

Figure 5: RelCost unary typing rules (Part 1)

$\boxed{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e : A}$ Execution cost of e is lower bounded by k and upper bounded by t, and e has the unary type A.

$$\frac{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e : A\{I/i\} \qquad \Delta \vdash I :: S}{\Delta;\Phi_a;\Omega \vdash^{t}_{k} \text{pack } e : \exists i{::}S.\, A}\ \textbf{pack}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{t_1}_{k_1} e_1 : \exists i{::}S.\, A_1 \qquad i :: S, \Delta;\Phi_a;x : A_1,\Omega \vdash^{t_2}_{k_2} e_2 : A_2 \qquad i \notin FV(\Phi_a;\Gamma,A_2,k_2,t_2)}{\Delta;\Phi_a;\Omega \vdash^{t_1+t_2}_{k_1+k_2} \text{unpack } e_1 \text{ as } x \text{ in } e_2 : A_2}\ \textbf{unpack}$$

$$\frac{\Upsilon(\zeta) = A_1 \xrightarrow{\text{exec}(k,t)} A_2 \qquad \Delta;\Phi_a;\Omega \vdash^{t'}_{k'} e : A_1}{\Delta;\Phi_a;\Omega \vdash^{t+t'+c_{primapp}}_{k+k'+c_{primapp}} \zeta\ e : A_2}\ \textbf{primapp}$$

$$\frac{\Delta;\Phi_a \models C \qquad \Delta;\Phi_a \wedge C;\Omega \vdash^{t}_{k} e : A}{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e : C \,\&\, A}\ \textbf{c-andI}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{t_1}_{k_1} e_1 : C \,\&\, A_1 \qquad \Delta;\Phi_a \wedge C;x : A_1,\Omega \vdash^{t_2}_{k_2} e_2 : A_2}{\Delta;\Phi_a;\Omega \vdash^{t_1+t_2}_{k_1+k_2} \text{clet } e_1 \text{ as } x \text{ in } e_2 : A_2}\ \textbf{c-andE}$$

$$\frac{\Delta;\Phi_a \wedge C;\Omega \vdash^{t}_{k} e : A \qquad \Delta \vdash C\ \texttt{wf}}{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e : C \supset A}\ \textbf{c-impI}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e : C \supset A \qquad \Delta;\Phi_a \models C}{\Delta;\Phi_a;\Omega \vdash^{t}_{k} \text{celim}_{\supset}\ e : A}\ \textbf{c-implE}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{t_1}_{k_1} e_1 : A_1 \qquad \Delta;\Phi_a;x : A_1,\Omega \vdash^{t_2}_{k_2} e_2 : A_2}{\Delta;\Phi_a;\Omega \vdash^{t_1+t_2+c_{let}}_{k_1+k_2+c_{let}} \text{let } x = e_1 \text{ in } e_2 : A_2}\ \textbf{let}$$

$$\frac{\Delta;\Phi_a \wedge C;\Omega \vdash^{t}_{k} e : A \qquad \Delta;\Phi_a \wedge \neg C;\Omega \vdash^{t}_{k} e : A \qquad \Delta \vdash C\ \texttt{wf}}{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e : A}\ \textbf{split}$$

$$\frac{\Delta;\Phi_a \models \bot \qquad \Delta \vdash^{A} A\ \texttt{wf}}{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e : A}\ \textbf{contra}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e : A \qquad \Delta;\Phi_a \models A \sqsubseteq A' \qquad \Delta;\Phi_a \models k' \leqslant k \qquad \Delta;\Phi_a \models t \leqslant t'}{\Delta;\Phi_a;\Omega \vdash^{t'}_{k'} e : A'}\ \sqsubseteq \text{exec}$$

Figure 6: RelCost unary typing rules (Part 2)

whose branches have different static costs. Subtyping is described later in Section 4.3.

4.2.2 *Relational Typing*

Relational typing establishes the relative cost of a pair of expressions and gives the pair a relational type. Relational typing rules can be divided into two categories: (a) *synchronous rules* that relate two structurally similar expressions and (b) *asynchronous rules* that relate two expressions with different structures but possibly similar subcomputations.

SYNCHRONOUS RULES All synchronous rules (shown in Figures 7 to 9) relate two structurally similar expressions, e.g., a pair of cons constructs or a pair of functions. If the two expressions contain subexpressions, the corresponding subexpressions are related component-wise. The rule **r-var** relates a variable to itself with zero relative cost. Similarly, all other axioms like **r-const** and **r-nil** relate an expression to itself. The rules **r-cons1** and **r-cons2** type non-empty lists of size $n+1$. If the tails have the relational type $\mathrm{list}[n]^{\alpha}\,\tau$, then the two cons'ed lists can be typed at either $\mathrm{list}[n+1]^{\alpha+1}\,\tau$ or $\mathrm{list}[n+1]^{\alpha}\,\tau$ depending on whether the heads may differ or not. The corresponding elimination rule **r-caseL** has four premises. The first premise establishes the type $\mathrm{list}[n]^{\alpha}\,\tau$ for the pair of lists being eliminated. The second premise types the nil branches, which are taken only when the two lists are empty and, hence, the constraint assumption $n = 0$ is added in this premise. If the lists are not empty, then there are two cases corresponding to the two cons rules. In the first case, the heads of the lists are the same and the tails differ in at most α elements (third premise). In this case, we assume that the heads have type $\Box\,\tau$. In the second case, the heads of the lists may differ (they have type τ, without a $\Box$) and the tails differ in at most $\alpha - 1$ elements (fourth premise). The value $\alpha - 1$ is represented by a fresh variable β that satisfies the constraint $\alpha = \beta + 1$.

Like all other values, recursive functions are relationally typed with zero cost. The relative cost t of the two related bodies is internalized into the function type $\tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2$ (rule **r-fix**). In the rule **r-app**, this internalized cost is added to the total cost of the application. The rule **r-inl** introduces a sum type with tag inl on both expressions. The **r-case** rule eliminates a sum type and assumes synchronous execution: The same branch must be taken in the left and right expressions. This is en-

$\boxed{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau}$ Relative cost of e_1 with respect to e_2 is upper bounded by t and the two expressions have relational type τ.

$$\frac{}{\Delta;\Phi_a;\Gamma \vdash n \ominus n \lesssim 0 : \mathrm{int}_r}\ \textbf{r-const} \qquad \frac{\Gamma(x) = \tau}{\Delta;\Phi_a;\Gamma \vdash x \ominus x \lesssim 0 : \tau}\ \textbf{r-var}$$

$$\frac{}{\Delta;\Phi_a;\Gamma \vdash () \ominus () \lesssim 0 : \mathrm{unit}_r}\ \textbf{r-unit}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash e \ominus e' \lesssim t : \tau_1 \qquad \Delta \vdash \tau_2\ \mathtt{wf}}{\Delta;\Phi_a;\Gamma \vdash \mathrm{inl}\ e \ominus \mathrm{inl}\ e' \lesssim t : \tau_1 + \tau_2}\ \textbf{r-inl}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash e \ominus e' \lesssim t : \tau_2 \qquad \Delta \vdash \tau_1\ \mathtt{wf}}{\Delta;\Phi_a;\Gamma \vdash \mathrm{inr}\ e \ominus \mathrm{inr}\ e' \lesssim t : \tau_1 + \tau_2}\ \textbf{r-inr}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash e \ominus e' \lesssim t : \tau_1 + \tau_2 \\ \Delta;\Phi_a;x : \tau_1,\Gamma \vdash e_1 \ominus e_1' \lesssim t' : \tau \qquad \Delta;\Phi_a;y : \tau_2,\Gamma \vdash e_2 \ominus e_2' \lesssim t' : \tau\end{array}}{\Delta;\Phi_a;\Gamma \vdash \ \mathrm{case}\ (e, x.e_1, y.e_2) \ominus\ \mathrm{case}\ (e', x.e_1', y.e_2') \lesssim t + t' : \tau}\ \textbf{r-case}$$

$$\frac{\begin{array}{c}\Delta \vdash \tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2\ \mathtt{wf} \\ \Delta;\Phi_a;x : \tau_1, f : \tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2,\Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau_2\end{array}}{\Delta;\Phi_a;\Gamma \vdash \mathrm{fix}\ f(x).e_1 \ominus \mathrm{fix}\ f(x).e_2 \lesssim 0 : \tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2}\ \textbf{r-fix}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a \vdash \tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2\ \mathtt{wf} \\ \Delta;\Phi_a;x : \tau_1, f : \Box(\tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2),\Gamma \vdash e \ominus e \lesssim t : \tau_2 \\ \forall x \in \mathrm{dom}(\Gamma).\ \ \Delta;\Phi_a \models \Gamma(x) \sqsubseteq \Box\Gamma(x)\end{array}}{\Delta;\Phi_a;\Gamma \vdash \mathrm{fix}\ f(x).e \ominus \mathrm{fix}\ f(x).e \lesssim 0 : \Box(\tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2)}\ \textbf{r-fixNC}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_1' \lesssim t_1 : \tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2 \\ \Delta;\Phi_a;\Gamma \vdash e_2 \ominus e_2' \lesssim t_2 : \tau_1\end{array}}{\Delta;\Phi_a;\Gamma \vdash e_1\ e_2 \ominus e_1'\ e_2' \lesssim t_1 + t_2 + t : \tau_2}\ \textbf{r-app}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_1' \lesssim t_1 : \tau_1 \qquad \Delta;\Phi_a;\Gamma \vdash e_2 \ominus e_2' \lesssim t_2 : \tau_2}{\Delta;\Phi_a;\Gamma \vdash \langle e_1, e_2\rangle \ominus \langle e_1', e_2'\rangle \lesssim t_1 + t_2 : \tau_1 \times \tau_2}\ \textbf{r-prod}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash e \ominus e' \lesssim t : \tau_1 \times \tau_2 \qquad i \in \{1,2\}}{\Delta;\Phi_a;\Gamma \vdash \pi_i(e) \ominus \pi_i(e') \lesssim t : \tau_i}\ \textbf{r-proj}_i$$

$$\frac{\Delta \vdash \tau\ \mathtt{wf} \qquad \Delta \vdash \alpha :: \mathbb{N}}{\Delta;\Phi_a;\Gamma \vdash \mathrm{nil}\ \ominus \mathrm{nil}\ \lesssim 0 : \mathrm{list}[0]^{\alpha}\,\tau}\ \textbf{r-nil}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_1' \lesssim t_1 : \tau \qquad \Delta;\Phi_a;\Gamma \vdash e_2 \ominus e_2' \lesssim t_2 : \mathrm{list}[n]^{\alpha}\,\tau}{\Delta;\Phi_a;\Gamma \vdash \mathrm{cons}(e_1, e_2) \ominus \mathrm{cons}(e_1', e_2') \lesssim t_1 + t_2 : \mathrm{list}[n+1]^{\alpha+1}\,\tau}\ \textbf{r-cons1}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_1' \lesssim t_1 : \Box\tau \qquad \Delta;\Phi_a;\Gamma \vdash e_2 \ominus e_2' \lesssim t_2 : \mathrm{list}[n]^{\alpha}\,\tau}{\Delta;\Phi_a;\Gamma \vdash \mathrm{cons}(e_1, e_2) \ominus \mathrm{cons}(e_1', e_2') \lesssim t_1 + t_2 : \mathrm{list}[n+1]^{\alpha}\,\tau}\ \textbf{r-cons2}$$

Figure 7: RelCost relational typing rules (Part 1)

$\boxed{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau}$ Relative cost of e_1 with respect to e_2 is upper bounded by t and the two expressions have relational type τ.

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash e \ominus e' \lesssim t : \mathrm{list}[n]^{\alpha}\,\tau \\ \Delta;\Phi_a \wedge n = 0;\Gamma \vdash e_1 \ominus e_1' \lesssim t' : \tau' \\ i,\Delta;\Phi_a \wedge n = i+1;h : \square\tau, tl : \mathrm{list}[i]^{\alpha}\,\tau,\Gamma \vdash e_2 \ominus e_2' \lesssim t' : \tau' \\ i,\beta,\Delta;\Phi_a \wedge n = i+1 \wedge \alpha = \beta+1;h : \tau, tl : \mathrm{list}[i]^{\beta}\,\tau,\Gamma \vdash e_2 \ominus e_2' \lesssim t' : \tau'\end{array}}{\Delta;\Phi_a;\Gamma \vdash \begin{array}{l}\mathtt{case}\ e\ \mathtt{of\ nil}\ \to e_1 \\ \mid h :: tl \to e_2\end{array} \ominus \begin{array}{l}\mathtt{case}\ e'\ \mathtt{of\ nil}\ \to e_1' \\ \mid h :: tl \to e_2'\end{array} \lesssim t + t' : \tau'}\ \textbf{r-caseL}$$

$$\frac{i :: S,\Delta;\Phi_a;\Gamma \vdash e \ominus e' \lesssim t : \tau \qquad i \notin \mathrm{FIV}(\Phi_a;\Gamma)}{\Delta;\Phi_a;\Gamma \vdash \Lambda.e \ominus \Lambda.e' \lesssim 0 : \forall i \overset{\mathrm{diff}(t)}{::} S.\,\tau}\ \textbf{r-iLam}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash e \ominus e' \lesssim t : \forall i \overset{\mathrm{diff}(t')}{::} S.\,\tau \qquad \Delta \vdash I : S}{\Delta;\Phi_a;\Gamma \vdash e[] \ominus e'[] \lesssim t + t'[I/i] : \tau\{I/i\}}\ \textbf{r-iApp}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash e \ominus e' \lesssim t : \tau\{I/i\} \qquad \Delta \vdash I :: S}{\Delta;\Phi_a;\Gamma \vdash \mathrm{pack}\ e \ominus \mathrm{pack}\ e' \lesssim t : \exists i{::}S.\,\tau}\ \textbf{r-pack}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_1' \lesssim t_1 : \exists i{::}S.\,\tau_1 \\ i :: S,\Delta;\Phi_a;x : \tau_1,\Gamma \vdash e_2 \ominus e_2' \lesssim t_2 : \tau_2 \qquad i \notin \mathrm{FV}(\Phi_a;\Gamma,\tau_2,t_2)\end{array}}{\Delta;\Phi_a;\Gamma \vdash \mathrm{unpack}\ e_1\ \mathrm{as}\ x\ \mathrm{in}\ e_2 \ominus \mathrm{unpack}\ e_1'\ \mathrm{as}\ x\ \mathrm{in}\ e_2' \lesssim t_1 + t_2 : \tau_2}\ \textbf{r-unpack}$$

$$\frac{\Upsilon(\zeta) = \tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2 \qquad \Delta;\Phi_a;\Gamma \vdash e \ominus e' \lesssim t' : \tau_1}{\Delta;\Phi_a;\Gamma \vdash \zeta\ e \ominus \zeta\ e' \lesssim t + t' : \tau_2}\ \textbf{r-primapp}$$

$$\frac{\Delta;\Phi_a \models C \qquad \Delta;\Phi_a \wedge C;\Gamma \vdash e \ominus e' \lesssim t : \tau}{\Delta;\Phi_a;\Gamma \vdash e \ominus e' \lesssim t : C\ \&\ \tau}\ \textbf{r-c-andI}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_1' \lesssim t_1 : C\ \&\ \tau_1 \\ \Delta;\Phi_a \wedge C;x : \tau_1,\Gamma \vdash e_2 \ominus e_2' \lesssim t_2 : \tau_2\end{array}}{\Delta;\Phi_a;\Gamma \vdash \mathrm{clet}\ e_1\ \mathrm{as}\ x\ \mathrm{in}\ e_2 \ominus \mathrm{clet}\ e_1'\ \mathrm{as}\ x\ \mathrm{in}\ e_2' \lesssim t_1 + t_2 : \tau_2}\ \textbf{r-c-andE}$$

$$\frac{\Delta;\Phi_a \wedge C;\Gamma \vdash e \ominus e' \lesssim t : \tau \qquad \Delta \vdash C\ \mathtt{wf}}{\Delta;\Phi_a;\Gamma \vdash e \ominus e' \lesssim t : C \supset \tau}\ \textbf{r-c-impI}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash e \ominus e' \lesssim t : C \supset \tau \qquad \Delta;\Phi_a \models C}{\Delta;\Phi_a;\Gamma \vdash \mathrm{celim}_{\supset}\ e \ominus \mathrm{celim}_{\supset}\ e' \lesssim t : \tau}\ \textbf{r-c-implE}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_1' \lesssim t_1 : \tau_1 \qquad \Delta;\Phi_a;x : \tau_1,\Gamma \vdash e_2 \ominus e_2' \lesssim t_2 : \tau_2}{\Delta;\Phi_a;\Gamma \vdash \mathrm{let}\ x = e_1\ \mathrm{in}\ e_2 \ominus \mathrm{let}\ x = e_1'\ \mathrm{in}\ e_2' \lesssim t_1 + t_2 : \tau_2}\ \textbf{r-let}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a \wedge C;\Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau \\ \Delta;\Phi_a \wedge \neg C;\Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau \qquad \Delta \vdash C\ \mathtt{wf}\end{array}}{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau}\ \textbf{r-split}$$

Figure 8: RelCost relational typing rules (Part 2)

$\boxed{\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau}$ Relative cost of e_1 with respect to e_2 is upper bounded by t and the two expressions have relational type τ.

$$\frac{\Delta; \Phi_a \models \bot \qquad \Delta \vdash \tau \ \mathtt{wf}}{\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau}\ \textbf{r-contra}$$

$$\frac{\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau \qquad \Delta; \Phi_a \models \tau \sqsubseteq \tau' \qquad \Delta; \Phi_a \models t \leqslant t'}{\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t' : \tau'}\ \textbf{r-}\sqsubseteq$$

$$\frac{\Delta; \Phi_a; \Gamma \vdash e \ominus e \lesssim t : \tau \qquad \forall x \in \mathrm{dom}(\Gamma).\ \Delta; \Phi_a \models \Gamma(x) \sqsubseteq \Box\Gamma(x)}{\Delta; \Phi_a; \Gamma, \Gamma'; \Omega \vdash e \ominus e \lesssim 0 : \Box\tau}\ \textbf{nochange}$$

Figure 9: RelCost relational typing rules (Part 3)

sured by the interpretation of the type $\tau_1 + \tau_2$ that only contains pairs of values with the same tag. If the case analyzed values have different tags, i.e., they are related at type $U(A_1 + A_2)$, then the analysis must switch to unary reasoning via the **switch** rule that is explained below.

The rule **nochange** relates an expression to itself at the (diagonal) type $\Box\tau$ and assigns a relative cost of 0, if the expression depends only on variables that are also labeled $\Box$. The latter condition ensures that at runtime, the two expressions being compared are syntactically equal. Statically, the rule applies when for all variables $x \in \Gamma$, the assumed type of x, i.e. $\Gamma(x)$, is a subtype of the same type annotated with $\Box$, i.e. of $\Box\Gamma(x)$. In addition, the rule **r-fixNC** allows inductively typing a recursive function with $\Box$ annotation. In typing the function's body, the function itself is assumed to be $\Box$-annotated. [13].

[13] *This rule cannot be derived using the rules **nochange** and **r-fix**.*

The rule **r-split** permits a case analysis on the index domain, allowing us to obtain more precise bounds. For example, when typing a divide-and-conquer algorithm that operates on a pair of lists in RelCost, one often needs to analyze the cases $\alpha = 0$ (where the two lists may not differ) and $\alpha > 0$ (where the two lists may differ) separately. This rule allows doing that. [14]

[14] *An example use of **r-split** rule illustrated in the* `msort` *example in Chapter 14.*

Finally, the rule **r-contra** allows us to give a pair of programs any well-typed term whenever we have inconsistent, i. e. contradictory, assumptions in the constraint context Φ_a. For example, when case analyzing a non-empty list, we can use this rule to discharge the `nil` case.

ASYNCHRONOUS RULES In addition to the synchronous rules that require the two related expressions to have the same structure, RelCost has several *asynchronous* rules that allow typing two expressions

$$\frac{\Delta;\Phi_a;|\Gamma|_1 \vdash^{t_1}_{k_1} e_1 : A_1 \qquad \Delta;\Phi_a;|\Gamma|_2 \vdash^{t_2}_{k_2} e_2 : A_2}{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \lesssim t_1 - k_2 : U\,(A_1, A_2)}\ \textbf{switch}$$

$$\frac{\Delta;\Phi_a;|\Gamma|_1 \vdash^{t_1}_{k_1} e_1 : A_1 \qquad \Delta;\Phi_a;x : U\,A_1,\Gamma \vdash e_2 \ominus e \lesssim t_2 : \tau_2}{\Delta;\Phi_a;\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 \ominus e \lesssim t_1 + t_2 + c_{let} : \tau_2}\ \textbf{r-let-e}$$

$$\frac{\Delta;\Phi_a;|\Gamma|_2 \vdash^{t_1}_{k_1} e_1 : A_1 \qquad \Delta;\Phi_a;x : U\,A_1,\Gamma \vdash e \ominus e_2 \lesssim t_2 : \tau_2}{\Delta;\Phi_a;\Gamma \vdash e \ominus \text{let } x = e_1 \text{ in } e_2 \lesssim t_2 - k_1 - c_{let} : \tau_2}\ \textbf{r-e-let}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;|\Gamma|_2 \vdash^{-}_{k'} e' : A_1 + A_2 \\ \Delta;\Phi_a;x : U\,A_1,\Gamma \vdash e \ominus e_1' \lesssim t : \tau \qquad \Delta;\Phi_a;y : U\,A_2,\Gamma \vdash e \ominus e_2' \lesssim t : \tau\end{array}}{\Delta;\Phi_a;\Gamma \vdash e \ominus \text{ case } (e', x.e_1', y.e_2') \lesssim t - k' - c_{case} : \tau}\ \textbf{r-e-case}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;|\Gamma|_1 \vdash^{t}_{-} e : A_1 + A_2 \\ \Delta;\Phi_a;x : U\,A_1,\Gamma \vdash e_1 \ominus e' \lesssim t' : \tau \qquad \Delta;\Phi_a;y : U\,A_2,\Gamma \vdash e_2 \ominus e' \lesssim t' : \tau\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{ case } (e, x.e_1, y.e_2) \ominus e' \lesssim t' + t + c_{case} : \tau}\ \textbf{r-case-e}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;|\Gamma|_2 \vdash^{-}_{k'} e' : \text{list}[n]\,A \qquad \Delta;\Phi_a \wedge n = 0;\Gamma \vdash e \ominus e_1' \lesssim t : \tau \\ i,\Delta;\Phi_a \wedge n = i+1;h : U\,A, tl : U\,\text{list}[i]\,A,\Gamma \vdash e \ominus e_2' \lesssim t : \tau\end{array}}{\Delta;\Phi_a;\Gamma \vdash e \ominus \begin{array}{l}\text{case } e' \text{ of nil } \to e_1' \\ \mid h :: tl \to e_2'\end{array} \lesssim t - k' - c_{caseL} : \tau}\ \textbf{r-e-caseL}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;|\Gamma|_1 \vdash^{t}_{-} e : \text{list}[n]\,A \qquad \Delta;\Phi_a \wedge n = 0;\Gamma \vdash e_1 \ominus e' \lesssim t : \tau \\ i,\Delta;\Phi_a \wedge n = i+1;h : U\,A, tl : U\,\text{list}[i]\,A,\Gamma \vdash e_2 \ominus e' \lesssim t : \tau\end{array}}{\Delta;\Phi_a;\Gamma \vdash \begin{array}{l}\text{case } e \text{ of nil } \to e_1 \\ \mid h :: tl \to e_2\end{array} \ominus e' \lesssim t + t' + c_{caseL} : \tau}\ \textbf{r-caseL-e}$$

Figure 10: Asynchronous typing rules

$$
\begin{array}{lcl}
|\cdot|_j & : & \text{Relational type} \to \text{Unary type} \\
\\
|\mathrm{int}_r|_j & = & \mathrm{int} \\
|\mathrm{unit}_r|_j & = & \mathrm{unit} \\
|\tau_1 \times \tau_2|_j & = & |\tau_1|_j \times |\tau_2|_j \\
|\tau_1 + \tau_2|_j & = & |\tau_1|_j + |\tau_2|_j \\
|\mathrm{list}[n]^{\alpha}\,\tau|_j & = & \mathrm{list}[n]\,|\tau|_j \\
|\tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2|_j & = & |\tau_1|_j \xrightarrow{\mathrm{exec}(0,\infty)} |\tau_2|_j \\
|\forall i \overset{\mathrm{diff}(t)}{::} S.\,\tau|_j & = & \forall i \overset{\mathrm{exec}(0,\infty)}{::} S.\,|\tau|_j \\
|\exists i{::}S.\,\tau|_j & = & \exists i{::}S.\,|\tau|_j \\
|C \,\&\, \tau|_j & = & C \,\&\, |\tau|_j \\
|C \supset \tau|_j & = & C \supset |\tau|_j \\
|U\,(A_1, A_2)|_j & = & A_j \\
|\Box\,\tau|_j & = & |\tau|_j
\end{array}
$$

Figure 11: RelCost refinement removal operation

that may be related partially or arbitrarily. These rules are shown in Figure 10. Our appendix shows an example of an optimizing program transformation–loop unswitching–that heavily relies on these asynchronous rules (Appendix D.2.2).

The most generic asynchronous rule is the **switch** rule that allows two arbitrary expressions e_1 and e_2 of types A_1 and A_2, respectively to be related at the weakest relation with type $U\,(A_1, A_2)$. When read from bottom to top, this rule allows switching from *relational* reasoning to *unary* reasoning where the two expressions are typed independently in their respective erased environments $|\Gamma|_j$ where $j \in \{1, 2\}$. Then, the relative cost is computed by taking the difference of the left expression's maximum cost and the right expression's minimum cost.

The type erasure operation $|.|_j$ is a function from relational types to unary types and it simply forgets the relational refinements. Its definition is shown in Figure 11. Since unrelated types $U\,(A_1, A_2)$ consist of the unary types A_1 and A_2 of the left and the right expressions, respectively, the erasure function is indexed by $j \in \{1, 2\}$ to select one of these expressions: $|U\,(A_1, A_2)|_j = A_j$. For function types $\tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2$, erasure constructs the weakest non-relational type $|\tau_1|_j \xrightarrow{\mathrm{exec}(0,\infty)} |\tau_2|_j$, providing no meaningful guarantees on minimum and maximum cost. The defi-

nition of $|.|_j$ extends pointwise to relational environments: $|\Gamma, x : \tau|_j = |\Gamma|_j, x : |\tau|_j$.

The remaining asynchronous rules apply when the left expression is related to a subexpression of the right expression, or vice-versa. These rules allow us to temporarily break the relational reasoning and regain it again later. Every asynchronous rule has a corresponding inverse/symmetric rule. For instance, the rule **r-let-e** relates $\text{let } x = e_1 \text{ in } e_2$ to an arbitrary expression e by relating e_2 to e. The symmetric rule **r-e-let** dually relates e to $\text{let } x = e_1 \text{ in } e_2$. We explain only the rule **r-let-e** here. In the first premise, we type the subexpression e_1 non-relationally with maximum execution cost t_1 and type A_1. In the second premise, we relate the left subexpression e_2 to the right expression e with relative cost t_2 under the assumption that the variable x is unrelated in the two runs $(x : U\, A_1)$. Since x occurs only in e_2, this is sound. The total relative cost is the sum of the costs t_1 and t_2, plus an additional cost c_{let} for the extra let elimination performed on the left side.[15]

[15] *In the symmetric rule **r-e-let**, this cost c_{let} is subtracted from the total relative cost since the let expression appears on the right side.*

4.3 SUBTYPING

Subtyping is central to both unary and relational typing. There are two subtyping judgments: $\Delta; \Phi_a \models^A A_1 \sqsubseteq A_2$ for unary types and $\Delta; \Phi_a \models \tau_1 \sqsubseteq \tau_2$ for relational types. Unary and relational subtyping rules are shown in Figure 12 and Figures 13 and 14, respectively.

Subtyping is constraint-dependent, because it must, for instance, be able to show that $\text{list}[n]^\alpha \tau \sqsubseteq \text{list}[m]^\alpha \tau$ when $m = n$. In RelCost, $\Box$'s comonadic properties are manifest via subtyping. This results in interactions between $\Box$ and other connectives as, for instance, in the rules **r-$\to$ $\Box$diff**, **r-l2** and **r-l$\Box$**. These interactions pose a nontrivial challenge for algorithmization, which is tackled in Chapter 11. Similar interactions exist between the modality $U\,(A_1, A_2)$ and other connectives.

The rules **u-$\to$** exec and **r-$\to$** diff are subtyping rules for unary and relational function types, respectively. Beyond the usual contravariance for arguments and covariance for results, upper bounds on costs are covariant whereas lower bounds are contavariant. We have two additional subtyping rules for function types. The rule **r-$\to$** execdiff allows converting two unrelated functions—with minimum and maximum execution costs k' and t, respectively—to related functions with execution cost $t - k'$, but with unrelated arguments and results. The rule **r-$\to$ $\Box$diff** captures the idea that syntactically equal functions, when applied to

$\boxed{\Delta;\Phi \models^{A} A_1 \sqsubseteq A_2}$ Unary type A_1 is a subtype of type A_2

$$\frac{\Delta;\Phi \models^{A} A_1' \sqsubseteq A_1 \qquad \Delta;\Phi \models^{A} A_2 \sqsubseteq A_2' \qquad \Delta;\Phi \models k' \leqslant k \qquad \Delta;\Phi \models t \leqslant t'}{\Delta;\Phi \models^{A} A_1 \xrightarrow{\text{exec}(k,t)} A_2 \sqsubseteq A_1' \xrightarrow{\text{exec}(k',t')} A_2'}\ \textbf{u-}\rightarrow_{\textbf{exec}}$$

$$\frac{i :: S, \Delta;\Phi \models^{A} A \sqsubseteq A' \qquad i :: S, \Delta;\Phi \models k' \leqslant k \qquad i :: S, \Delta;\Phi \models t \leqslant t' \qquad i \notin FV(\Phi)}{\Delta;\Phi \models^{A} \forall i \overset{\text{exec}(k,t)}{::} S.\, A \sqsubseteq \forall i \overset{\text{exec}(k',t')}{::} S.\, A'}\ \textbf{u-}\forall_{\textbf{exec}}$$

$$\frac{\Delta;\Phi \models^{A} A_1 \sqsubseteq A_1' \qquad \Delta;\Phi \models^{A} A_2 \sqsubseteq A_2'}{\Delta;\Phi \models^{A} A_1 \times A_2 \sqsubseteq A_1' \times A_2'}\ \textbf{u-}\times$$

$$\frac{\Delta;\Phi \models^{A} A_1 \sqsubseteq A_1' \qquad \Delta;\Phi \models^{A} A_2 \sqsubseteq A_2'}{\Delta;\Phi \models^{A} A_1 + A_2 \sqsubseteq A_1' + A_2'}\ \textbf{u-}+$$

$$\frac{\Delta;\Phi \models n \doteq n' \qquad \Delta;\Phi \models^{A} A \sqsubseteq A'}{\Delta;\Phi \models^{A} \text{list}[n]\, A \sqsubseteq \text{list}[n']\, A'}\ \textbf{u-l}$$

$$\frac{i :: S, \Delta;\Phi \models^{A} A \sqsubseteq A' \qquad i \notin FV(\Phi)}{\Delta;\Phi \models^{A} \exists i{::}S.\, A \sqsubseteq \exists i{::}S.\, A'}\ \textbf{u-}\exists$$

$$\frac{\Delta;\Phi \wedge C \models C' \qquad \Delta;\Phi \models^{A} A \sqsubseteq A'}{\Delta;\Phi \models^{A} C\ \&\ A \sqsubseteq C'\ \&\ A'}\ \textbf{u-c-and}$$

$$\frac{\Delta;\Phi \wedge C' \models C \qquad \Delta;\Phi \models^{A} A \sqsubseteq A'}{\Delta;\Phi \models^{A} C \supset A \sqsubseteq C' \supset A'}\ \textbf{u-c-impl} \qquad \frac{}{\Delta;\Phi \models^{A} A \sqsubseteq A}\ \textbf{u-refl}$$

$$\frac{\Delta;\Phi \models^{A} A_1 \sqsubseteq A_2 \qquad \Delta;\Phi \models^{A} A_2 \sqsubseteq A_3}{\Delta;\Phi \models^{A} A_1 \sqsubseteq A_3}\ \textbf{u-trans}$$

Figure 12: RelCost unary subtyping rules

$\boxed{\Delta;\Phi \models \tau_1 \sqsubseteq \tau_2}$ Relational type τ_1 is a subtype of type τ_2

$\boxed{\Delta;\Phi \models^{A} A_1 \sqsubseteq A_2}$ Unary type A_1 is a subtype of type A_2

$$\frac{}{\Delta;\Phi \models \mathrm{int}_r \sqsubseteq \Box\,\mathrm{int}_r}\ \textbf{r-int-}\Box \qquad \frac{}{\Delta;\Phi \models \Box\,U\,(\mathrm{int},\mathrm{int}) \sqsubseteq \mathrm{int}_r}\ \textbf{r-}\Box\textbf{U-int}$$

$$\frac{}{\Delta;\Phi \models \mathrm{unit}_r \sqsubseteq \Box\,\mathrm{unit}_r}\ \textbf{r-unit}$$

$$\frac{\Delta;\Phi_a \models \tau_1' \sqsubseteq \tau_1 \qquad \Delta;\Phi_a \models \tau_2 \sqsubseteq \tau_2' \qquad \Delta;\Phi_a \models t \leqslant t'}{\Delta;\Phi_a \models \tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2 \sqsubseteq \tau_1' \xrightarrow{\mathrm{diff}(t')} \tau_2'}\ \textbf{r-}\to\ \textbf{diff}$$

$$\frac{}{\Delta;\Phi \models \Box\,(\tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2) \sqsubseteq \Box\,\tau_1 \xrightarrow{\mathrm{diff}(0)} \Box\,\tau_2}\ \textbf{r-}\to\ \Box\textbf{diff}$$

$$\frac{}{\Delta;\Phi \models U\,(A_1 \xrightarrow{\mathrm{exec}(k,t)} A_2, A_1' \xrightarrow{\mathrm{exec}(k',t')} A_2') \sqsubseteq U\,(A_1,A_1') \xrightarrow{\mathrm{diff}(t-k')} U\,(A_2,A_2')}\ \textbf{r-}\to\ \textbf{execdiff}$$

$$\frac{i :: S,\Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad i :: S,\Delta;\Phi_a \models t \leqslant t' \qquad i \notin FV(\Phi_a)}{\Delta;\Phi_a \models \forall i \overset{\mathrm{diff}(t)}{::} S.\,\tau \sqsubseteq \forall i \overset{\mathrm{diff}(t')}{::} S.\,\tau'}\ \textbf{r-}\forall\textbf{diff}$$

$$\frac{}{\Delta;\Phi \models \Box\,(\forall i \overset{\mathrm{diff}(t)}{::} S.\,\tau) \sqsubseteq \forall i \overset{\mathrm{diff}(0)}{::} S.\,\Box\,\tau}\ \textbf{r-}\forall\,\Box$$

$$\frac{}{\Delta;\Phi \models U\,(\forall i \overset{\mathrm{exec}(k,t)}{::} S.\,A, \forall i \overset{\mathrm{exec}(k',t')}{::} S.\,A') \sqsubseteq \forall i \overset{\mathrm{diff}(t-k')}{::} S.\,U\,(A,A')}\ \textbf{r-}\forall\textbf{U}$$

$$\frac{\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_1' \qquad \Delta;\Phi_a \models \tau_2 \sqsubseteq \tau_2'}{\Delta;\Phi_a \models \tau_1 \times \tau_2 \sqsubseteq \tau_1' \times \tau_2'}\ \textbf{r-}\times$$

$$\frac{}{\Delta;\Phi \models \Box\tau_1 \times \Box\tau_2 \equiv \Box\,(\tau_1 \times \tau_2)}\ \textbf{r-}\times\Box$$

$$\frac{}{\Delta;\Phi \models U\,(A_1 \times A_2, A_1' \times A_2') \sqsubseteq U\,(A_1,A_1') \times U\,(A_2,A_2')}\ \textbf{r-}\times\textbf{U}$$

$$\frac{\Delta;\Phi \models \tau_1 \sqsubseteq \tau_1' \qquad \Delta;\Phi \models \tau_2 \sqsubseteq \tau_2'}{\Delta;\Phi \models \tau_1 + \tau_2 \sqsubseteq \tau_1' + \tau_2'}\ \textbf{r-}+$$

$$\frac{}{\Delta;\Phi \models \Box\tau_1 + \Box\tau_2 \sqsubseteq \Box\,(\tau_1 + \tau_2)}\ \textbf{r-}+\Box$$

$$\frac{\Delta;\Phi_a \models n \doteq n' \qquad \Delta;\Phi_a \models \alpha \leqslant \alpha' \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau'}{\Delta;\Phi_a \models \mathrm{list}[n]^{\alpha}\,\tau \sqsubseteq \mathrm{list}[n']^{\alpha'}\,\tau'}\ \textbf{r-l1}$$

$$\frac{\Delta;\Phi \models \alpha \doteq 0}{\Delta;\Phi \models \mathrm{list}[n]^{\alpha}\,\tau \sqsubseteq \mathrm{list}[n]^{\alpha}\,\Box\,\tau}\ \textbf{r-l2}$$

$$\frac{}{\Delta;\Phi \models \mathrm{list}[n]^{\alpha}\,\Box\,\tau \sqsubseteq \Box\,(\mathrm{list}[n]^{\alpha}\,\tau)}\ \textbf{r-l}\Box$$

Figure 13: RelCost relational subtyping rules (Part 1)

$\boxed{\Delta; \Phi \models \tau_1 \sqsubseteq \tau_2}$ Binary type τ_1 is a subtype of type τ_2

$$\frac{i :: S, \Delta; \Phi_a \models \tau \sqsubseteq \tau' \qquad i \notin FV(\Phi_a)}{\Delta; \Phi_a \models \exists i{::}S.\, \tau \sqsubseteq \exists i{::}S.\, \tau'}\ \textbf{r-}\exists$$

$$\frac{}{\Delta; \Phi \models \exists i{::}S.\, \Box\tau \sqsubseteq \Box(\exists i{::}S.\, \tau)}\ \textbf{r-}\exists\Box$$

$$\frac{\Delta; \Phi_a \wedge C \models C' \qquad \Delta; \Phi_a \models \tau \sqsubseteq \tau'}{\Delta; \Phi_a \models C \,\&\, \tau \sqsubseteq C' \,\&\, \tau'}\ \textbf{r-c-and}$$

$$\frac{}{\Delta; \Phi \models C \,\&\, \Box\tau \sqsubseteq \Box(C \,\&\, \tau)}\ \textbf{r-c-and-}\Box$$

$$\frac{\Delta; \Phi_a \wedge C' \models C \qquad \Delta; \Phi_a \models \tau \sqsubseteq \tau'}{\Delta; \Phi_a \models C \supset \tau \sqsubseteq C' \supset \tau'}\ \textbf{r-c-impl}$$

$$\frac{}{\Delta; \Phi \models \Box(C \supset \tau) \sqsubseteq C \supset \Box\tau}\ \textbf{r-c-impl-}\Box \qquad \frac{}{\Delta; \Phi \models \Box\tau \sqsubseteq \tau}\ \textbf{T}$$

$$\frac{}{\Delta; \Phi \models \Box\tau \sqsubseteq \Box\Box\tau}\ \textbf{D} \qquad \frac{\Delta; \Phi_a \models \tau_1 \sqsubseteq \tau_2}{\Delta; \Phi_a \models \Box\tau_1 \sqsubseteq \Box\tau_2}\ \textbf{B-}\Box$$

$$\frac{}{\Delta; \Phi \models \tau \sqsubseteq U(|\tau|_1, |\tau|_2)}\ \textbf{W}$$

$$\frac{\Delta; \Phi \models^A A_1 \sqsubseteq A_1' \qquad \Delta; \Phi \models^A A_2 \sqsubseteq A_2'}{\Delta; \Phi \models U(A_1, A_2) \sqsubseteq U(A_1', A_2')}\ \textbf{U} \qquad \frac{}{\Delta; \Phi \models \tau \sqsubseteq \tau}\ \textbf{r-refl}$$

$$\frac{\Delta; \Phi_a \models \tau_1 \sqsubseteq \tau_2 \qquad \Delta; \Phi_a \models \tau_2 \sqsubseteq \tau_3}{\Delta; \Phi_a \models \tau_1 \sqsubseteq \tau_3}\ \textbf{r-trans}$$

Figure 14: RelCost relational subtyping rules (Part 2)

equal arguments, produce equal results and have relative cost 0. Similar additional rules exist for universally quantified relational types.

The rule **r-l1** allows the number of elements that differ in a list to be weakened covariantly. The rule **r-l2** allows two related lists with zero differences to be retyped as two related lists whose elements are in the diagonal relation. The rule **r-l□** allows two related lists whose elements are equal to be retyped as two equal lists, represented by the outer □.

The rule **B-□** allows stripping the box annotations if the inner types are subtypes of one another. The rule **W** allows weakening the type τ to its weakest form $\mathsf{U}\,(|\tau|_1, |\tau|_2)$ where $|\tau|_j$ is the j-th unary projection for $j \in \{1, 2\}$ described earlier. The rule **U** allows lifting subtyping from unary types to relational types at the weakest relation $\mathsf{U}\ \cdot$. As usual, subtyping is reflexive (rules **u-refl** and **r-refl**) and transitive (rules **u-trans** and **r-trans**).

We note that the type $\Box\tau$ follows the standard co-monadic rules: $\Box\tau \sqsubseteq \tau$, $\Box(\tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2) \sqsubseteq \Box\tau_1 \xrightarrow{\mathrm{diff}(0)} \Box\tau_2$ and $\Box(\tau_1 \times \tau_2) \equiv \Box\tau_1 \times \Box\tau_2$.

5

METATHEORY AND SOUNDNESS OF RELCOST

▶ SYNOPSIS In this chapter, we first discuss *logical relations*, a proof technique that is particularly well-suited for proving RelCost's type system sound. Based on this technique, we then present a logical relations model for RelCost and use it to prove RelCost sound relative to the abstract cost semantics presented in Section 4.1.

LOGICAL RELATIONS AS A PROOF TECHNIQUE Logical relations [92] are a powerful proof technique for proving many important program properties such as strong normalization, program equivalence, parametricity, and type-safety. The technique has wide applicability not only to unary properties such as strong normalization but also to relational properties such as program equivalence. Moreover, as we demonstrate shortly, logical relations can be naturally extended with unary and relational effects. This makes logical relations an attractive tool for proving the soundness of RelCost's type and effect system.

Logical relations are defined by induction on types and the relations are crafted so that they capture the property of interest. However, in the presence of recursion (or recursive types in general), types themselves as induction measure do not suffice. To deal with this, Ahmed *et al.* have developed *step-indexed logical relations* where the relation is indexed with a step, capturing the number of future evaluation steps [7, 11]. With the help of step-indices, one can show the well-foundedness of recursive definitions.

In the rest of this chapter, we build two cost-annotated models of RelCost's types: a *non-relational* (unary) one for unary types and unary execution and a *relational* (binary) one for relational types and relational execution. Both models are step-indexed to handle recursive functions [7, 11]. The binary model depends on the unary one and a key novelty is how unary and relational step indices interact. Below, we discuss the models in detail.

5.1 UNARY INTERPRETATION OF RELCOST TYPES

For each unary type A, the value interpretation $[\![A]\!]_v$ is a set, containing pairs (m, v) of step indices and values. Intuitively, the pair (m, v)

is in the interpretation of $[\![A]\!]_v$, if the value v behaves like a value of type A in a larger term for at least m steps. Hence, $[\![A]\!]_v$ can be considered as an approximation of the set of values in A. Below, we briefly comment on the value interpretation $[\![A]\!]_v$, shown in Figure 15.

The base types int and unit are completely characterized by the set of their values for any step index m. Function types $A_1 \xrightarrow{\text{exec}(k,t)} A_2$ are characterized by the set of functions that, given an argument of type A_1, produce a computation of type A_2 with k and t minimum and maximum execution costs, respectively (in the expression relation $[\![A_2]\!]_\varepsilon^{k,t}$ discussed below). Note that the resulting computation is interpreted at *strictly smaller step-indices* than the function's step-index, essentially counting an additional cost for function application.

Universally quantified types $\forall i \overset{\text{exec}(k,t)}{::} S.\, A$ are interpreted so that *for any* well-sorted index term I, the resulting expression is in the interpretation of $A\{I/i\}$ with $k[I/i]$ and $t[I/i]$ minimum and maximum execution costs, respectively. Note that since index term applications are not counted as computation steps, the step index does not decrease. Dually, existentially quantified types $\exists i::S.\, A$ are interpreted so that *there exists* a well-sorted witness I so that the packed value v is in the interpretation of type $A\{I/i\}$.

Based on the interpretation of values, we can also define the interpretation of closed expressions $[\![A]\!]_\varepsilon^{k,t}$. Intuitively, the pair (m, e) is in the expression interpretation $[\![A]\!]_\varepsilon^{k,t}$, if the expression e behaves like an expression of type A with k and t minimum and maximum execution costs, respectively for at least m steps. Its definition is shown below.

$$[\![A]\!]_\varepsilon^{k,t} = \left\{ (m, e) \;\middle|\; (e \Downarrow^{c,r} v \wedge\ c < m) \implies \begin{array}{l} 1.\ k \leqslant r \leqslant t \\ 2.\ (m - c, v) \in [\![A]\!]_v \end{array} \right\}$$

The interpretation of $[\![A]\!]_\varepsilon^{k,t}$ states that if e evaluates to a value with $c < m$ steps, then k and t are lower and upper bounds on the execution cost r, respectively, and the resulting value is in the value interpretation with step-index $m - c$. Note that the step indices only interact with the reduction steps c, but not with the actual execution costs r.

As usual, we interpret open expressions under some semantic environment interpretation δ. We write $(m, \delta) \in \mathcal{G}[\![\Omega]\!]$ to mean that δ maps

$$\llbracket A \rrbracket_v \subseteq \text{Step index} \times \text{Value}$$
$$\llbracket A \rrbracket_\varepsilon^{k,t} \subseteq \text{Step index} \times \text{Expression}$$

$$\begin{aligned}
\llbracket \text{int} \rrbracket_v &= \{(m, n)\} \\
\llbracket \text{unit} \rrbracket_v &= \{(m, ())\} \\
\llbracket A_1 \times A_2 \rrbracket_v &= \{(m, \langle v_1, v_2 \rangle) \mid (m, v_1) \in \llbracket A_1 \rrbracket_v \wedge (m, v_2) \in \llbracket A_2 \rrbracket_v\} \\
\llbracket A_1 + A_2 \rrbracket_v &= \{(m, \text{inl } v) \mid (m, v) \in \llbracket A_1 \rrbracket_v\} \cup \\
&\quad \{(m, \text{inr } v) \mid (m, v) \in \llbracket A_2 \rrbracket_v\} \\
\llbracket A_1 \wedge A_2 \rrbracket_v &= \{(m, v) \mid (m, v) \in \llbracket A_1 \rrbracket_v \wedge (m, v) \in \llbracket A_2 \rrbracket_v\} \\
\llbracket \text{list}[0]\ A \rrbracket_v &= \{(m, \text{nil}\ )\} \\
\llbracket \text{list}[n{+}1]\ A \rrbracket_v &= \{(m, \text{cons}(e_1, e_2)) \mid (m, e_1) \in \llbracket A \rrbracket_v \wedge (m, e_2) \in \llbracket \text{list}[n]\ A \rrbracket_v\} \\
\llbracket A_1 \xrightarrow{\text{exec}(k,t)} A_2 \rrbracket_v &= \{(m, \text{fix } f(x).e) \mid \forall j < m.\ \forall v.\ (j, v) \in \llbracket A_1 \rrbracket_v \\
&\qquad \implies \quad (j, e[v/x, \text{fix } f(x).e/f]) \in \llbracket A_2 \rrbracket_\varepsilon^{k,t}\} \\
\llbracket \forall i \overset{\text{exec}(k,t)}{::} S.\ A \rrbracket_v &= \{(m, \Lambda.e) \mid \forall I.\ \vdash I :: S.\ (m, e) \in \llbracket A\{I/i\} \rrbracket_\varepsilon^{k[I/i], t[I/i]}\} \\
\llbracket \exists i{::}S.\ A \rrbracket_v &= \{(m, \text{pack } v) \mid \exists I.\ \vdash I :: S \ \wedge (m, v) \in \llbracket A\{I/i\} \rrbracket_v\} \\
\llbracket C \supset A \rrbracket_v &= \{(m, v) \mid \not\models C \ \vee \ (m, v) \in \llbracket A \rrbracket_v\} \\
\llbracket C \ \& \ A \rrbracket_v &= \{(m, v) \mid \models C \ \wedge \ (m, v) \in \llbracket A \rrbracket_v\}
\end{aligned}$$

$$\llbracket A \rrbracket_\varepsilon^{k,t} = \{(m, e) \mid (e \Downarrow^{c,r} v \ \wedge \ c < m) \implies \begin{array}{l} 1.\ k \leqslant r \leqslant t \\ 2.\ (m - c, v) \in \llbracket A \rrbracket_v \end{array}\}$$

Figure 15: Non-relational interpretation of types

all variables in the domain of the environment Ω to appropriately-typed semantic values for m steps.

$$\begin{aligned}\mathcal{G}[\![\cdot]\!] &= \{(m, \emptyset)\}\\ \mathcal{G}[\![\Omega, x : A]\!] &= \{(m, \delta[x \mapsto v]) \mid (m, \delta) \in \mathcal{G}[\![\Omega]\!] \wedge (m, v) \in [\![A]\!]_v\}\end{aligned}$$

We write $\sigma \in \mathcal{D}[\![\Delta]\!]$ to mean that σ is a valid (well-sorted) substitution for the index environment Δ.

5.2 RELCOST'S SOUNDNESS (UNARY)

We prove the following fundamental theorem for unary typing. Roughly, the theorem says that the expression e, if typed in RelCost at unary type A with k and t minimum and maximum execution costs, respectively, lies in the unary expression interpretation of A (with the given costs k and t) for any step-index and value substitution that respects the environment's types.

Theorem 1 (Fundamental Theorem for Unary Typing)**.** *Assume that* $\Delta; \Phi_a; \Omega \vdash^{t}_{k} e : A$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and there exists* Ω' *s.t.* $FV(e) \subseteq dom(\Omega')$, $\Omega' \subseteq \Omega$ *and* $(m, \gamma) \in \mathcal{G}[\![\sigma\Omega']\!]$. *Then,* $(m, \gamma e) \in [\![\sigma A]\!]^{\sigma k, \sigma t}_{\varepsilon}$. [1]

Proof. By induction on the typing derivation. (shown in Appendix A.2) □

An immediate corollary of the theorem is that the minimum and maximum execution costs established in the type system are lower and upper bounds on the actual execution cost of the program, respectively. For readability, we only state the theorem with a single input x, but generalized versions with any number of inputs hold as well.

Corollary 2 (Soundness for unary costs)**.** *Suppose that*

- $x : A \vdash^{t}_{k} e : A'$
- $\vdash_{-} v : A$
- $e[v/x] \Downarrow^{c,r} v'$

Then $k \leqslant r \leqslant t$.

1 The existence of Ω' that contains all the free variables of e is needed for proving asynchronous typing rules sound.

5.3 RELATIONAL INTERPRETATION OF RELCOST TYPES

The value interpretation $(\!|\tau|\!)_v$ is a set, containing triples (m, v_1, v_2) consisting of a step index m and two related values v_1 and v_2. Intuitively, the triple (m, v_1, v_2) is in the interpretation of $(\!|\tau|\!)_v$, if the values v_1 and v_2 behave like related values of type τ in a larger term for at least m steps. Hence, $(\!|\tau|\!)_v$ can be considered as an approximation of the set of related values in τ. We briefly comment on some salient points about $(\!|\tau|\!)_v$, shown in Figure 16.

The base relational types int_r and unit_r are completely characterized by the set of pairs of identical values for any step index m. The interpretation of $\tau_1 \xrightarrow{\text{diff}(t)} \tau_2$ relates a pair of functions that, given related arguments at $j < m$ steps, return related computations (in the expression relation $(\!|\tau|\!)^t_\varepsilon$ discussed below) at step-index j. In addition, the two functions are in the unary interpretation of $|\tau_1|_1 \xrightarrow{\text{exec}(0,\infty)} |\tau_2|_1$ and $|\tau_1|_2 \xrightarrow{\text{exec}(0,\infty)} |\tau_2|_2$, respectively for any step-index j. The latter allows any pair of related functions to be used in a unary context with the weakest cost bounds, 0 and ∞. In essence, we can *semantically* show that the relational judgment $\Delta; \Phi; \Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau$ entails the unary judgment $\Delta; \Phi; |\Gamma|_i \vdash^{\infty}_{0} e_i : |\tau|_i$ for $i \in \{1, 2\}$.

The interpretation of $\Box\tau$ forces the two related values to be identical. Semantically, the type $\Box$ is used in the interpretation of non-empty lists to make sure that α changes are distributed appropriately: if the two heads are identical (of type $\Box\tau$), the tails have α changes, otherwise the tails have $\alpha - 1$ changes.

The interpretation of $U(A_1, A_2)$ contains unrelated pairs of values (v_1, v_2) in which the individual values v_1 and v_2 are in the unary interpretation $[\![A_1]\!]_v$ and $[\![A_2]\!]_v$, respectively *at any step index* j, i.e., $(j, v_1) \in [\![A_1]\!]_v$ and $(j, v_2) \in [\![A_1]\!]_v$ for any j. Essentially, this means that when we switch from relational to unary reasoning, we can call out to any unary step index j. This works because the unary relation does not refer back to the binary relation.

Based on the relational interpretation of pairs of values, the expression interpretation $(\!|\tau|\!)^t_\varepsilon$ defines when two expressions are logically related. Intuitively, $(m, e_1, e_2) \in (\!|\tau|\!)^t_\varepsilon$, if the expressions e_1 and e_2 behave like related expressions of type τ with t relative cost for at least m steps. Its definition is shown below.

$$(\!|\tau|\!)^t_\varepsilon = \{(m, e_1, e_2) \mid (e_1 \Downarrow^{c_1, r_1} v_1 \ \wedge\ e_2 \Downarrow^{c_2, r_2} v_2 \ \wedge\ c_1 < m) \implies \begin{array}{l} 1.\ r_1 - r_2 \leqslant t \\ 2.\ (m - c_1, v_1, v_2) \in (\!|\tau|\!)_v \end{array} \}$$

$(\!|\tau|\!)_\nu \subseteq \text{Step index} \times \text{Value} \times \text{Value}$
$(\!|\tau|\!)^t_\varepsilon \subseteq \text{Step index} \times \text{Expression} \times \text{Expression}$

$$
\begin{aligned}
(\!|\Box\,\tau|\!)_\nu &= \{(m,v,v) \mid (m,v,v) \in (\!|\tau|\!)_\nu\} \\
(\!|U\,(A_1,A_2)|\!)_\nu &= \{(m,v_1,v_2) \mid \forall j.\ (j,v_1) \in [\![A_1]\!]_\nu \wedge (j,v_2) \in [\![A_2]\!]_\nu\} \\
(\!|\text{int}_r|\!)_\nu &= \{(m,n,n)\} \\
(\!|\text{unit}_r|\!)_\nu &= \{(m,(),())\} \\
(\!|\tau_1 \times \tau_2|\!)_\nu &= \{(m,\langle v_1,v_2\rangle,\langle v_1',v_2'\rangle) \mid (m,v_1,v_1') \in (\!|\tau_1|\!)_\nu \wedge (m,v_2,v_2') \in (\!|\tau_2|\!)_\nu\} \\
(\!|\tau_1 + \tau_2|\!)_\nu &= \{(m,\text{inl } v,\text{inl } v') \mid (m,v,v') \in (\!|\tau_1|\!)_\nu\} \cup \\
&\quad \{(m,\text{inr } v,\text{inr } v') \mid (m,v,v') \in (\!|\tau_2|\!)_\nu\} \\
(\!|\tau_1 \wedge \tau_2|\!)_\nu &= \{(m,v,v') \mid (m,v,v') \in (\!|\tau_1|\!)_\nu \ \wedge\ (m,v,v') \in (\!|\tau_2|\!)_\nu\} \\
(\!|\text{list}[0]^\alpha\,\tau|\!)_\nu &= \{(m,\text{nil},\text{nil})\} \\
(\!|\text{list}[n+1]^\alpha\,\tau|\!)_\nu &= \{(m,\text{cons}(v_1,v_2),\text{cons}(v_1',v_2')) \mid \\
&\quad ((m,v_1,v_1') \in (\!|\Box\,\tau|\!)_\nu \wedge (m,v_2,v_2') \in (\!|\text{list}[n]^\alpha\,\tau|\!)_\nu) \vee \\
&\quad ((m,v_1,v_1') \in (\!|\tau|\!)_\nu \wedge (m,v_2,v_2') \in (\!|\text{list}[n]^{\alpha-1}\,\tau|\!)_\nu \wedge \alpha > 0)\} \\
(\!|\tau_1 \xrightarrow{\text{diff}(t)} \tau_2|\!)_\nu &= \{(m,\text{fix } f(x).e_1,\text{fix } f(x).e_2) \mid (\forall j < m.\ \forall v_1,v_2.\ (j,v_1,v_2) \in (\!|\tau_1|\!)_\nu. \\
&\quad \implies (j,e_1[v_1/x,\text{fix } f(x).e_1/f],e_2[v_2/x,\text{fix } f(x).e_2/f]) \in (\!|\tau_2|\!)^t_\varepsilon) \wedge \\
&\quad (\forall j.(j,\text{fix } f(x).e_1) \in [\![|\tau_1|_1 \xrightarrow{\text{exec}(0,\infty)} |\tau_2|_1]\!]_\nu \wedge \\
&\quad\quad (j,\text{fix } f(x).e_2) \in [\![|\tau_1|_2 \xrightarrow{\text{exec}(0,\infty)} |\tau_2|_2]\!]_\nu)\} \\
(\!|\forall i \overset{\text{diff}(t)}{::} S.\,\tau|\!)_\nu &= \{(m,\Lambda.e,\Lambda.e') \mid \forall I.\ \vdash I :: S.\ ((m,e,e') \in (\!|\tau\{I/i\}|\!)^{t[I/i]}_\varepsilon) \wedge \\
&\quad\quad (\forall j.(j,e) \in [\![|\tau\{I/i\}|_1]\!]^{0,\infty}_\varepsilon \ \wedge\ (j,e') \in [\![|\tau\{I/i\}|_2]\!]^{0,\infty}_\varepsilon)\} \\
(\!|\exists i::S.\,\tau|\!)_\nu &= \{(m,\text{pack } v,\text{pack } v') \mid \exists I.\ \vdash I :: S \ \wedge (m,v,v') \in (\!|\tau\{I/i\}|\!)_\nu\} \\
(\!|C \supset \tau|\!)_\nu &= \{(m,v_1,v_2) \mid \not\models C \ \vee\ (m,v_1,v_2) \in (\!|\tau|\!)_\nu\} \\
(\!|C \,\&\, \tau|\!)_\nu &= \{(m,v_1,v_2) \mid \models C \ \wedge\ (m,v_1,v_2) \in (\!|\tau|\!)_\nu\}
\end{aligned}
$$

$$
\begin{aligned}
(\!|\tau|\!)^t_\varepsilon &= \{(m,e_1,e_2) \mid (e_1 \Downarrow^{c_1,r_1} v_1 \ \wedge\ e_2 \Downarrow^{c_2,r_2} v_2 \ \wedge\ c_1 < m) \implies \\
&\qquad\qquad 1.\ r_1 - r_2 \leqslant t \\
&\qquad\qquad 2.\ (m - c_1, v_1, v_2) \in (\!|\tau|\!)_\nu \\
&\quad\}
\end{aligned}
$$

Figure 16: Relational interpretation of types

The definition states that if expressions e_1 and e_2 evaluate to values in c_1 and c_2 steps, respectively, and $c_1 < m$, then t is an upper bound on the relative cost of e_1 with respect to e_2, i.e., $r_1 - r_2 \leqslant t$ and the resulting values are related at step-index $m - c_1$. The relational step-index m counts steps of the left expression but it could be set up to count steps of the right expression or both. Moreover, like in the unary expression relation $[\![A]\!]^{\cdot}_{\varepsilon}$, step-indices only interact with the reduction steps, but not with the actual execution costs r_1 and r_2.

We interpret pairs of open expressions under a related pair of substitutions, (δ_1, δ_2). We write $(m, \delta_1, \delta_2) \in \mathcal{G}(\!|\Gamma|\!)$ to mean that δ_1 and δ_2 map all the variables in the domain of the environment Γ to appropriately-typed semantic relational values for m steps.

$$\begin{aligned}\mathcal{G}(\!|\cdot|\!) &= \{(m, \emptyset, \emptyset)\}\\ \mathcal{G}(\!|\Gamma, x:\tau|\!) &= \{(m, \delta_1[x \mapsto v_1], \delta_2[x \mapsto v_2]) \mid (m, \delta_1, \delta_2) \in \mathcal{G}(\!|\Gamma|\!) \wedge \\ &\qquad (m, v_1, v_2) \in (\!|\tau|\!)_v\}\end{aligned}$$

5.4 RELCOST'S SOUNDNESS (RELATIONAL)

We prove the following fundamental theorem for our relational typing judgment. Roughly, the theorem says that the expressions e_1 and e_2, if typed in RelCost at relational type τ with relative cost t, lie in the relational expression interpretation of τ (with the given relative cost t) for any step-index and relational value substitution that respects the environment's types.

Theorem 3 (Fundamental Theorem for Relational Typing)**.** *Assume that* $\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, \delta_1, \delta_2) \in \mathcal{G}(\!|\sigma\Gamma|\!)$*. Then,* $(m, \delta_1 e_1, \delta_2 e_2) \in (\!|\sigma\tau|\!)^{\sigma t}_{\varepsilon}$.

Proof. Proof is by induction on the typing derivation. (shown in Appendix A.2) □

An immediate corollary of the theorem is that relative costs established in the type system are upper bounds on the actual execution cost differences. For readability, we only state the theorem with a single input x, but generalized versions with any number of inputs hold as well.

Corollary 4 (Soundness for relational costs)**.** *Suppose that*

- $x : \tau \vdash e_1 \ominus e_2 \lesssim t : \tau'$
- $\vdash v_1 \ominus v_2 \lesssim \_ : \tau$

- $e_1[v_1/x] \Downarrow^{c_1,r_1} v_1'$
- $e_2[v_2/x] \Downarrow^{c_2,r_2} v_2'$

Then $r_1 - r_2 \leqslant t$.

Finally, we prove that, semantically, relational typing is a refinement of unary typing with the weakest bounds—0 and ∞—on minimum and maximum costs, respectively.

Theorem 5 (Fundamental Theorem for Weak Relational Typing)**.** *Assume that* $\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$. *Then for* $i \in \{1, 2\}$, *if there exists* Γ_i' *s.t.* $FV(e_i) \subseteq dom(\Gamma_i')$, $\Gamma_i' \subseteq \Gamma$ *and* $(m, \gamma_i) \in \mathcal{G}[\![|\sigma\Gamma_i'|_i]\!]$, *then* $(m, \gamma_i e_i) \in [\![|\sigma\tau|_i]\!]_\varepsilon^{0,\infty}$.

Proof. Proof is by induction on the typing derivation. (shown in Appendix A.2) □

6

RELATED WORK : RELATIONAL COST ANALYSIS

> I know nothing except the fact of my own ignorance.
>
> Socrates

The work presented in Part I represents a convergence of two main bodies of research: static execution cost analysis and relational analysis. We discuss related work in each of these areas in order.

6.1 STATIC EXECUTION COST ANALYSIS

There are many static techniques for analyzing the execution cost/-complexity of programs ranging from semantic interpretation [21, 54] to type-based techniques such as linear dependent types [39, 41], amortized resource analysis [60, 61, 63], type and effect systems [81, 96], and type-based embedding via cost-counting monads [42]. However, all these techniques differ from our work in a fundamental way: They all reason about a single execution of a program, whereas relational cost analysis requires reasoning about a pair of programs.

In theory, one can simply combine best- and worst-case execution cost analysis computed using one of these techniques to reason about the relative costs of two programs. However, as we have demonstrated in Section 1.1, such combinations ignore the relations between programs and inputs, leading to imprecision. RelCost (as well as DuCostIt presented in Chapter 8) distinguishes itself from prior work in its *relational reasoning principles*, which provide the ability to establish precise bounds on relative cost by making use of similarities between inputs and programs in a much more *local* way. To achieve this, we build on type and effect systems [81, 96] and extend them to a relational setting.

Below, we survey some unary type-theoretic approaches to verifying and inferring resource usage bounds in functional programs.

TYPE AND EFFECT SYSTEMS FOR UNARY COST ANALYSIS Type and effect systems are a static program analysis technique that refines a usual type system with additional annotations, called *effects*, i.e. abstract descriptions of side-effects occurring during the program execu-

tion such as I/O events and exceptions. The execution cost of a program can be also considered as an effect. In fact, several type and effect systems have been designed for execution cost analysis. One such system is designed by Reistad and Gifford [96] for estimating the cost analysis of Lisp expressions and is partially based on the "time system" by Dornic *et al.* [45]. A second such system is designed by Crary and Weirich that extend type and effect systems with dependent types [38] to certify resource usage of programs.

SIZED TYPES AND LINEAR DEPENDENT TYPES Sized types were introduced by Hughes *et al.* for proving some functional properties of stream-manipulating reactive programs such as termination and productivity. The technique has been adapted to reason about resource usage of functional programs by combining it with various other techniques such as abstract interpretation [103], linear dependent types [41, 67] and type-and-effect systems [38].

Dal Lago and Petit present a complete time complexity analysis for PCF [41]. They use linear types to statically limit the number of times a function may be applied by the context in a call-by-name setting. This allows reasoning about the time complexity of recursive functions precisely. Recently, they carried out a similar development for a call-by-value language [40]. Extending their approach with relational reasoning would be an interesting direction.

AUTOMATIC AMORTIZED COST ANALYSIS The *potential method* derives amortized bounds on the worst-case execution cost of a program by assigning non-negative *potentials* to the input data of a program. The amortized cost of an operation is the sum of the actual execution cost of the operation plus the change in the potential between before and after the operation. Then, if the initial potential is non-zero and the potential is always non-negative, one can show that the accumulated amortized costs are an upper bound on the accumulated actual costs [83].

Based on such amortized cost analysis, Hoffmann *et al.* [60, 61] infer polynomial-shaped worst-case bounds on resource usage of RAML (Resource Aware ML) programs. Recently, the technique has been extended with support for lower bounds on the resource usage [80], making it possible to obtain naive relative cost bounds by simply establishing the difference on the upper and lower bounds. A significant advantage of their technique is automation. A similar analysis for *relational cost*—with support for tracking similarities—may be possible although the compatibility of logarithmic functions (which are necessary to state

the relational costs of interesting programs) with Hoffmann *et al.*'s approach remains an open problem.

AUTOMATIC RECURRENCE EXTRACTION AND SOLUTION A classic approach to reason about resource usage of programs is to automatically extract (and then solve) recurrence relations from programs using a variety of techniques such as program transformations [71], abstract interpretation [98], refinement types [53] and cost transformations [43].

For example, [96] develop a type and effect system for functional programs without general recursion but with a set of restricted combinators like `map` and `fold`. [53] has proposed a technique based on DML [105] that uses size information contained in dependent types to automatically extract recurrence relations from first-order DML programs. Danner *et al.* instrument programs with a clock and extract recurrence relations [43].

A common denominator of these type-theoretic techniques is that they are unary. Some of the techniques operate on a restricted set of constructs or cost terms. Applicability of these techniques to the relational setting is unclear: In a relational setting, recurrence relations (and their solutions) might get much more complicated: e. g. for recurrence relation of mergesort's relative cost (shown in Appendix D.2) is parametrized by not only the input size but also the number of changes between the two lists. Moreover, the resulting closed-form expression for the recurrence involves iterated sums over exponentials and logarithms which is difficult to automate.

6.2 RELATIONAL ANALYSIS AND VERIFICATION

There is a large body of work on verifying relational properties of programs. Many of the techniques for relational reasoning have been semantic, but recently, there is an increasing focus on developing practical approaches based on assertion checking [70], symbolic execution [87], static analysis [68], model checking [107], program logics [18, 109], and refinement types [14, 16]. In the past, researchers have developed specialized approaches for many relational properties, e.g., information flow [10, 78, 93], continuity [30], determinism [25], differential privacy [51, 95], or quantitative reliability [27, 28]. Other important applications of relational verification include regression verification [49, 52], semantical differences [69, 86] and cross or relative verification [57, 76, 88].

In the rest of this section, we focus on the more closely related approaches based on type systems and program logics. The former supports automatic typechecking and inference while the latter often is more expressive and contains a wider variety of connectives at the price of requiring the programmer to complete the proofs. One advantage of our work over many of these approaches is that we can freely switch from the relational world to the non-relational world when program executions diverge.

INFORMATION FLOW ANALYSIS Information flow analysis is a prime example of relational analysis where the aim is to determine whether secret inputs of a program influence its non-secret outputs. Several type systems for information flow analysis has been proposed: SLAM [58], FlowCaml [93], DCC [2]. These type systems use security-annotated types to ensure the *noninterference property*, i. e. a property that compares two executions of a single program differing only in its secret inputs and requires the non-secret outputs to be equal. RelCost differs from these works in two ways. First, unlike these systems which use security-annotated types, in RelCost we have a two-layered type grammar which makes the design of the type system cleaner. Second, these systems are designed to reason about non-interference, which is a relational functional property whereas RelCost can also handle cost, which is a relational quantitative property.

PARAMETRICITY One of the most fundamental relational properties of functional programs is parametricity: an abstract uniformity property which states that all instances of a polymorphic function *act* the same way [97]. Ramifications of relational parametricity are quite powerful: Researchers have developed deep connections of parametricity with representation independence and program transformations. In particular, parametricity demonstrates that one cannot distinguish between a pair of polymorphic programs that differ in their underlying data representation. This notion of representation independence yields "free" theorems about programs based on their types. Relational models like logical relations have been extensively used in the proofs of parametricity. Our work on relational cost analysis builds on the foundations of parametricity and extends it to the quantitative setting.

RELATIONAL FUNCTIONAL VERIFICATION Relational Hoare Logic [17] and Relational Separation Logic [108] are two program logics that extend their corresponding unary counterparts—Hoare and Separation Logic, respectively—to the relational setting to reason about relational

properties of imperative programs. These logics and their successors have been used to reason about not only program equivalence but also other relational properties such as probabilistic differential privacy [13], access control [79], and information flow [79]. Recently, Relational Hoare Logic has been extended to higher-order logic [8] for a pure fragment only. In general, these logics are quite powerful (powerful enough to embed RelCost into RHOL [8]) and they lie on the end of the spectrum of relational verification where the programmers must provide detailed proofs using proof assistants. Our work lies on the opposite end of the spectrum since we are interested in more lightweight methods with minimal burden on the programmers.

In [99], Sands introduces *improvements*, a semantic notion which naturally embeds relational cost reasoning, and uses them as artefacts for proving equivalence between functional programs. However, improvements only offer a qualitative guarantee that one program is faster than another (in all contexts). In contrast, RelCost can establish quantitative bounds.

Part II

DUCOSTIT

7

DUCOSTIT BY EXAMPLES

▶ SYNOPSIS This chapter demonstrates how the relational cost analysis technique presented in Part I can be used to reason about the update costs of incremental programs. We first recap dynamic stability and then explain how a type and effect system similar to RelCost, which we call DuCostIt, can be used to establish dynamic stability. We first give a mini overview of DuCostIt's type system and then present some of DuCostIt's features through examples.

7.1 DYNAMIC STABILITY AS AN INSTANCE OF RELATIONAL COST ANALYSIS

INCREMENTAL COMPUTATIONS Programs are often optimized under the implicit assumption that they will execute only once. However, in practice, many programs are executed again and again on slightly different inputs: spreadsheets compute the same formulas with modifications to some of their cells, search engines periodically crawl the web, and software build processes respond to small source code changes. In such settings, it is not enough to design a program that is efficient for the first (from-scratch) execution; the program must also be efficient for the subsequent incremental executions (ideally much more efficient than the from-scratch execution).

Incremental computation is a promising approach to this problem that aims to design software that can automatically and efficiently respond to changing inputs. The potential for efficient incremental updates comes from the fact that, in practice, large parts of the computation repeat between the first and the incremental run. As shown by prior work on self-adjusting computation [5, 6], by storing intermediate results in a trace in the first run, it is possible to re-execute only those parts that depend on the input changes during the incremental run, and to reuse the parts that didn't change (free of cost).

Existing work on incremental computation has been applied to different settings such as imperative [4, 55], demand-driven [56] and fully functional [6, 29]. In all of these settings, the approach has been very successful at improving the efficiency of incremental runs of a program. In addition, there has been also language based techniques that can au-

tomatically convert conventional programs to their incremental counterparts [32], helping ease the burden on the programmer. However, previous work does not consider the equally important question of how programmers can reason about and establish the computational complexity of incremental executions—a property which we call *dynamic stability*.

DYNAMIC STABILITY Assume that a program e is initially executed with input v and then the program is re-run with a slightly different input v'. Dynamic stability is the amount of time it takes to re-run the program with the modified input v' using *incremental computation*. However, unlike relational cost analysis where the two programs are executed using the same strategy, dynamic stability analysis requires a more complex evaluation semantics. Instead of reasoning about two programs, dynamic stability analysis requires reasoning about two executions of a program: a) The initial run in which all the intermediate results and input-output dependencies of the program are stored in a dynamic dependence graph, which is often called a *trace* and b) The incremental run in which the input changes are automatically propagated through the trace of the computation.

The former phase is called *from-scratch* execution and memoizes all the intermediate results. The latter phase is called *change propagation* and is implemented by storing all values in reference cells, representing the trace as a dynamic dependence graph over those references, and updating the references by traversing the graph starting from changed leaves (inputs) and re-computing all references that depend on the changed references. This is a bottom-up procedure, which incurs cost only for the parts of the trace that have changed. The graph can be traversed using many different strategies [3].[16]

[16] *A previous version of DuCostIt was shown sound with respect to one such strategy as explained in [35].*

In this thesis, we argue that even though the underlying evaluation technique of incremental computations is complex, dynamic stability analysis is a special instance of relational cost analysis. In the rest of this chapter, we demonstrate that relational cost analysis can be adjusted to track dynamic stability by using an enhanced operational and semantic model that is capable of modeling traces and incremental evaluation. In particular, we retrofit RelCost's design to a refinement type and effect system called DuCostIt that can establish dynamic stability of incremental programs.[17] Doing so allows us to not only statically verify when incremental computation is worthwhile—which was not possible using previous techniques—but also demonstrate that relational cost analysis is a powerful method that has applications in seemingly unrelated domains.

[17] *As in RelCost, DuCostIt operates over the higher-order functional programming language Cost^{ML}.*

Remark. *DuCostIt's type and effect system presented here differs substantially from the prior homonymous version of the author's work in [35]. Chapter 10 makes a detailed comparison to this prior version, which we call DuCostIt^0, to distinguish it from DuCostIt.*

7.2 RELATIONAL COST ANALYSIS FOR DYNAMIC STABILITY

Before we explain the details of DuCostIt, we highlight how dynamic stability analysis in DuCostIt can be considered a specific instance of relational cost analysis in RelCost. In doing so, we also describe how dynamic stability analysis differs from the relational cost analysis technique introduced in Chapter 4.

- *Relational reasoning:* Like relational cost analysis, dynamic stability is also inherently a *relational property* of two runs of a program: the initial run and the subsequent, incremental run.
- *Different cost model:* Since DuCostIt aims at establishing dynamic stability of programs, its cost model is geared towards incremental evaluation. Given an initial execution that stores intermediate results in a trace, a change propagation mechanism accounts for the cost of incremental update when the input changes. For cases where an input change requires executing a part of the program that has not been executed before (i. e. has no trace), the cost model must also account for from-scratch execution costs of programs. In contrast, RelCost only works with from-scratch executions of two programs.
- *Only single programs:* DuCostIt's type system and semantic model are inherently limited to two runs of the *same program* with possibly different inputs. In particular, asynchronous typing rules of RelCost, which are often necessary to obtain precision when programs differ structurally, are not present in DuCostIt.
- *Only upper bounds:* In general, change propagation may have to recompute an intermediate value if either (a) that value was obtained as the result of a primitive function whose inputs have changed, or (b) that value was obtained from a closure, but the closure has now changed, either due to a change in control flow or due to a non-trivial change to an input function.[18] In both of these cases, like in RelCost, we switch to the naive non-relational analysis. However, as opposed to RelCost's **switch** rule that requires obtaining upper and lower bounds on the two related programs,

[18] *[6, 33] has shown that by storing values in modifiable reference cells and updating them in-place during change propagation, the cost for structural operations like pairing, projection and list consing can be avoided during change propagation.*

DuCostIt's unary analysis only requires obtaining upper bounds on the execution costs of programs. This is justified since we are interested in the upper bound on the amount of work that must be done to execute parts of the program from scratch.

- *Relational refinement types:* Dynamic stability is a function of changes to a program's inputs and, hence, a precise analysis of dynamic stability requires knowing which of its free variables and, more generally, which of its subexpressions' result values may change after an update. To differentiate changeable and unchangeable values statically, like in RelCost, we rely on relational refinement types and also adopt a two-layered type grammar. $\square\,\tau$ ascribes values of type τ which cannot change whereas $\mathsf{U}\,A$ ascribes pairs of values of type A that may change arbitrarily.[19].
- *Biexpressions:* Unlike RelCost, DuCostIt's semantic model operates over "biexpressions"—pairs of expressions that are structurally identical (but may have different, related substitutions). Biexpressions capture how parts of the program change from initial to incremental run and they are instrumental for directing our abstract change-propagation semantics.[20].

[19] *Values of type τ may admit indirect changes in nested sub-values. This is explained in Chapter 8.*

[20] *Details are explained in Section 8.1*

EXAMPLE 1 (WARM-UP) Consider the boolean expression "$x \leqslant 5$" with one input x of type int. Assuming that computing $\leqslant$ from-scratch costs 1 unit of time, what is the the dynamic stability of this expression? Like in RelCost, the precise answer depends on whether x may change in the incremental run or not: If x may change, i.e. $x : \mathsf{U}\,\mathsf{int}$, then change propagation may recompute $\leqslant$, so the dynamic stability would be 1. If x cannot change, i.e. $x : \mathsf{int}_r$, then change propagation will simply bypass this expression, and the cost will be 0. Hence, the program can be typed in two different ways: $x : \mathsf{U}\,\mathsf{int} \vdash x \leqslant 5 : \mathsf{U}\,\mathsf{bool} \mid 1$ and $x : \mathsf{int}_r \vdash x \leqslant 5 : \mathsf{bool}_r \mid 0$, where the incremental run's cost is written on the right-hand side.

Note that this cost is relational: It is relative to the first run of the program, but the relation between the costs is not just a difference; it is determined by the change propagation semantics.

DUAL-MODE TYPING The typing judgment described above suffices for typing programs where only primitive functions are re-executed during change propagation. However, in general, change propagation may execute fresh closures from-scratch. To count the costs of these closures, we need a second "mode" of typing, that upper-bounds the *from-scratch* execution cost of an expression. Accordingly, we use two

typing judgments: a unary judgment $\vdash_{\mathbb{FS}} e : A \mid t$, which means that the cost of evaluating e *from-scratch* is at most t and a relational judgment $\vdash_{\mathbb{CP}} e : \tau \mid t$, which means that the cost of *change propagating* through e is at most t.[21]. The latter is relational in the sense that we consider a relational substitution for its free variables in the semantics. Just like in RelCost, the types are also two-layered: unary types A represent sets of values, whereas relational types τ represent pairs of initial and modified values. As a rule, the from-scratch cost always dominates the change propagation cost.

[21] *For now this intuition suffices. $\vdash_{\mathbb{FS}}$ has a double meaning, which is explained later.*

Going back to the program $x \leqslant 5$ from Example 1, it can be given a from-scratch execution cost 1 using the $\mathbb{FS}$-mode typing judgment: $x : \text{int} \vdash_{\mathbb{FS}} x \leqslant 5 : \text{bool} \mid 1$, where 1 accounts for the cost of the comparison function. As shown earlier, the same program can be given two different update costs using the $\mathbb{CP}$-mode typing judgments: $x : \mathrm{U}\,\text{int} \vdash_{\mathbb{CP}} x \leqslant 5 : \mathrm{U}\,\text{bool} \mid 1$ and $x : \text{int}_r \vdash_{\mathbb{CP}} x \leqslant 5 : \text{bool}_r \mid 0$.

EXAMPLE 2 (MODE-SWITCHING) Like in RelCost, the two modes of typing interact with each other at elimination points. Consider the change propagation cost of the following program "`if` x `then` e_1 `else` e_2". If $x : \text{bool}_r$, we know that x will not change. So, the incremental run will execute the same branch (e_1 or e_2) as the initial run. This means that change propagation can be continued in the branch. Consequently, in this case, like in RelCost, we only need to establish change propagation costs of the two branches e_i, not their from-scratch evaluation costs. In the type system, this means that the branches can be typed in $\mathbb{CP}$ mode, as in the following derivation.

$$\frac{x : \text{bool}_r \vdash_{\mathbb{CP}} x : \text{bool}_r \mid 0 \qquad x : \text{bool}_r \vdash_{\mathbb{CP}} e_1 : \tau \mid t \qquad x : \text{bool}_r \vdash_{\mathbb{CP}} e_2 : \tau \mid t}{x : \text{bool}_r \vdash_{\mathbb{CP}} \texttt{if}\ x\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : \tau \mid t}\ \textbf{if}$$

If $x : \mathrm{U}\,\text{bool}$, then x may change. Consequently, the initial and incremental runs may execute different branches. If the branches end up being different, change propagation must execute the new branch from-scratch. Hence, to deal with this, like in RelCost, DuCostIt uses the following switch rule.[22]:

$$\frac{|\Gamma| \vdash_{\mathbb{FS}} e : A \mid t}{\Gamma \vdash_{\mathbb{CP}} e : \mathrm{U}\,A \mid t}\ \textbf{switch}$$

[22] *Notice that **switch** rule in RelCost is slightly different, since it has to account for the difference in the worst- and best-case executions of the two related programs.*

where e is typed independently in the $\mathbb{FS}$-mode with maximum execution cost t. Note that like in RelCost, the premise is unary, while the

conclusion is relational. Then, the update time of e is upper bounded by its worst case execution cost t.[23]. Since the from-scratch execution cost of e is independent of changeability of its inputs, we can type it with a non-relational environment $|\Gamma|$ obtained from Γ (as in RelCost).

[23] *If the branch doesn't actually change, change propagation will not evaluate from-scratch, but in that case the cost will only be lower, so our established cost would be conservative.*

Using this **switch** rule and assuming that conditionals incur 1 cost and the executions costs of the branches are at most t', we can type "`if x then` e_1 `else` e_2" independently with maximum execution cost $t' + 1$ and obtain the below typing:

$$\frac{x : \mathsf{bool} \vdash_{\mathbb{FS}} \texttt{if } x \texttt{ then } e_1 \texttt{ else } e_2 : A \mid t' + 1}{x : \mathsf{U}\,\mathsf{bool} \vdash_{\mathbb{CP}} \texttt{if } x \texttt{ then } e_1 \texttt{ else } e_2 : \mathsf{U}\,A \mid t' + 1}\ \textbf{switch}$$

Note that because x might change in the incremental run, any computation that depends on it may change as well. Hence, the result type is also unrelated, i.e., $\mathsf{U}\,A$.

The attentive reader might wonder why we have switched to the unary typing for the whole statement, rather than establishing only the from-scratch costs of the two branches as follows:

$$\frac{\begin{array}{c} x : \mathsf{U}\,\mathsf{bool} \vdash_{\mathbb{CP}} x : \mathsf{U}\,\mathsf{bool} \mid 0 \\ x : \mathsf{U}\,\mathsf{bool} \vdash_{\mathbb{FS}} e_1 : \tau \mid t' \qquad x : \mathsf{U}\,\mathsf{bool} \vdash_{\mathbb{FS}} e_2 : \tau \mid t' \end{array}}{x : \mathsf{U}\,\mathsf{bool} \vdash_{\mathbb{CP}} \texttt{if } x \texttt{ then } e_1 \texttt{ else } e_2 : \tau \mid t' + 1}\ \textbf{if-2}$$

In this formulation, the branches are typed in the $\mathbb{FS}$ mode (not $\mathbb{CP}$ mode as in **if** rule above), hence t' is not the cost for change-propagation, but from-scratch execution.

Although this would work, we chose to have one generic rule like in RelCost that switches to the unary typing. The motivation is twofold. First, rather than duplicating all elimination rules for cases where the eliminated expression is of type $\mathsf{U}\,A$, we only have a single relational typing rule for all elimination forms, hence a simpler type system.[24]. Second, this formulation corresponds closely to RelCost's type system and hence highlights our point that relational cost analysis can be used for dynamic stability.

[24] *See [35] for a version with duplicated rules in $\mathbb{CP}$ mode.*

EXAMPLE 3 (MAP) Branch points are not the only reason why change propagation may end up executing a completely fresh expression. A second reason is that a function provided as input to another function may change non-trivially. To illustrate this, we type the standard list function `map`, introduced in Chapter 1.

Before we explain how `map` can be typed in DuCostIt, we briefly discuss the types necessary to express its dynamic stability. Like in Rel-

Cost, list types in DuCostIt are also refined to the form $\mathsf{list}[n]\,A$ and $\mathsf{list}[n]^{\alpha}\,\tau$, respectively for unary and relational lists. Relational function type $\tau_1 \to \tau_2$ is refined to $\tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2$ which says that the cost of change propagating through the body of the function is at most t whereas the unary function type $A_1 \to A_2$ is refined to $A_1 \xrightarrow{\mathbb{FS}(t)} A_2$ which says that the cost of from-scratch execution of the function body is at most t. For instance, based on Example 1, the function $\lambda x.\,(x \leqslant 5)$ can be given the relational types $\mathsf{int}_r \xrightarrow{\mathbb{CP}(0)} \mathsf{bool}_r$, $U\,\mathsf{int} \xrightarrow{\mathbb{CP}(1)} U\,\mathsf{bool}$, and $U\,(\mathsf{int} \xrightarrow{\mathbb{FS}(1)} \mathsf{bool})$.

Next, let us consider the standard `map` function that applies an input function `f` to every element of an input list `l`.

$$\begin{aligned}&\texttt{fix map(f).}\,\lambda \texttt{l.}\,\texttt{case l of}\;\; \texttt{nil} \;\to\; \texttt{nil}\\ &\qquad\qquad\qquad\qquad \mid \texttt{h :: tl} \;\to\; \texttt{cons(f h, map f tl)}\end{aligned}$$

Assume that `f` has type $\Box\,(\tau \xrightarrow{\mathbb{CP}(t)} \tau')$, i.e., `f` does not depend on anything that may change and its body change-propagates with cost at most t. In this case, to change propagate `map`'s body, we must only change propagate through `f` on changed elements of `l`, of which there are at most α. Hence, the cost is $O(\alpha \cdot t)$ and, indeed, `map` can be given the following type in DuCostIt for a suitable linear function P. We explain how `map`'s type is derived as it highlights our co-monadic reasoning principle.

$$\begin{aligned}\vdash_{\mathbb{CP}} \texttt{map} :\; \Box\,(\tau \xrightarrow{\mathbb{CP}(t)} \tau') \xrightarrow{\mathbb{CP}(0)} \forall n, \alpha{::}\mathbb{N}.\\ \mathsf{list}[n]^{\alpha}\,\tau \xrightarrow{\mathbb{CP}(P(\alpha\cdot t))} \mathsf{list}[n]^{\alpha}\,\tau' \mid 0.\end{aligned}$$

The interesting part of the typing is establishing the change propagation cost of the `h :: tl` branch in the definition of `map`. We are trying to bound this cost by $P(\alpha \cdot t)$. We know from `l`'s type that at most α elements in `h :: tl` will change in the second run. However, we do not know whether `h` is one of those elements. So, like in RelCost, our case analysis rule (Section 8.3.2, Figure 31) has *two premises for the* `cons` *branch* (a total of three premises, including the premise for `nil`). In the first of these two premises, we assume that `h` may change, so $\texttt{h} : \tau$ and $\texttt{tl} : \mathsf{list}[n-1]^{\alpha-1}\,\tau$. In the second premise, we assume that `h` cannot change, so $\texttt{h} : \Box\,\tau$ and $\texttt{tl} : \mathsf{list}[n-1]^{\alpha}\,\tau$.

Analysis of the first premise is straightforward: `(f h)` incurs cost t (from `f`'s type $\Box\,(\tau \xrightarrow{\mathbb{CP}(t)} \tau')$) and, inductively, `(map f tl)` incurs cost

$P((\alpha-1)\cdot t)$, for a total cost $t + P((\alpha-1)\cdot t) = P(\alpha\cdot t)$. Analysis of the second premise requires nonstandard reasoning. Here, $tl : \mathsf{list}[n-1]^{\alpha}\,\tau$, so the inductive cost of $(\mathtt{map}\ f\ tl)$ is already $P(\alpha\cdot t)$. Hence, we must show that $(f\ h)$ has 0 change propagation cost. For this, we rely on our co-monadic reasoning principle: If all of an expression's free variables have types of the form $\Box\cdot$ (i.e., their substitutions will not change), then the expression's change propagation cost is 0 (using the rule **cp-nochange** in Figure 31). Since we know that $f : \Box(\tau \xrightarrow{\mathbb{CP}(t)} \tau')$ and $h : \Box\tau$, we can immediately conclude that $(f\ h)$ has 0 change propagation cost. [25]

[25] *An alternative way of typing is to subtype* $\Box(\tau \xrightarrow{\mathbb{CP}(t)} \tau')$ *to* $\Box\tau \xrightarrow{\mathbb{CP}(0)} \Box\tau'$ *using the comonadic subtyping rule* $\to\Box_{cp}$ *in Figure 34.*

The more interesting question is what happens if we allow f to change, i.e., f has type $U(A \xrightarrow{\mathbb{FS}(t)} A')$. In this case, change propagation may have to re-execute the function on all list elements from scratch, so the cost of `map` is $O(n\cdot t)$. This yields the following second type for `map` for a suitable linear function Q.

$$\vdash_{\mathbb{CP}} \mathtt{map} : (U(A \xrightarrow{\mathbb{FS}(t)} A')) \xrightarrow{\mathbb{CP}(0)} \forall n, \alpha{::}\mathbb{N}.\ \mathsf{list}[n]^{\alpha}\,U\,A \xrightarrow{\mathbb{CP}(Q(n\cdot t))} \mathsf{list}[n]^{n}\,U\,A' \mid 0.$$

EXAMPLE 4 (BALANCED LIST FOLD) Standard list fold operations (`foldl` and `foldr`) can be typed easily in DuCostIt but are uninteresting for incremental computation because they have linear traces (linear dependency chains) and, hence, have $O(n)$ dynamic stability even for single element changes to an input list of length n. A more interesting operation is what we call the balanced fold. Given an *associative and commutative* binary function f of simple type $(\tau \times \tau) \to \tau$, a list of simple type $(\mathsf{list}\,\tau)$ can be folded by splitting it into two nearly equal sized lists, folding the sublists recursively and then applying f to the two results. This results in a balanced tree-like trace, whose depth is $\lceil \mathtt{log}_2(n) \rceil$. A single change to the list causes $\lceil \mathtt{log}_2(n) \rceil$ recomputations of f. So, if f has dynamic stability t, the dynamic stability with one change to the list is $O(t\cdot \mathtt{log}_2(n))$. More generally, it can be shown that if α changes are allowed to the list, then the dynamic stability is $O(t\cdot(\alpha + \alpha\cdot \mathtt{log}_2(n/\alpha)))$. This simplifies to $O(t\cdot n)$ when $\alpha = n$ (entire list may change) and $O(t\cdot \mathtt{log}_2(n))$ when $\alpha = 1$. In the following we implement such a balanced fold operation, `bfold`, and derive its dynamic stability in DuCostIt.

Our first ingredient is the function `bsplit`, which splits a list of length n into two lists of lengths $\lceil \frac{n}{2} \rceil$ and $\lfloor \frac{n}{2} \rfloor$.

$$
\begin{array}{l}
\texttt{fix bsplit}(\_).\Lambda.\Lambda.\lambda l.\texttt{case } l \texttt{ of} \\
\quad \text{nil} \ \to \langle \text{nil} \, , \text{nil} \, \rangle \\
\mid h_1 :: tl_1 \ \to \texttt{case } tl_1 \texttt{ of} \\
\qquad\qquad \text{nil} \ \to \langle \text{cons}(h_1, \text{nil} \,), \text{nil} \, \rangle \\
\qquad\quad \mid h_2 :: tl_2 \ \to \text{let } r = \texttt{bsplit} \ ()[\,][\,] \ tl_2 \text{ in} \\
\qquad\qquad\qquad\qquad \text{unpack } r \text{ as } r' \text{ in} \\
\qquad\qquad\qquad\qquad \text{clet } r' \text{ as } x \text{ in} \\
\qquad\qquad\qquad\qquad \text{pack } \langle \text{cons}(h_1, \pi_1 x), \text{cons}(h_2, \pi_2 x) \rangle
\end{array}
$$

This function is completely standard. Its DuCostIt type, although easily established, is somewhat interesting because it uses an existential quantifier to split the allowed number of changes α into the two split lists. The dynamic stability of `bsplit` is 0 because `bsplit` uses no primitive functions (*cf.* discussion earlier in this section).

$$
\begin{array}{l}
\texttt{bsplit} : \Box\,(\text{unit}_r \xrightarrow{\mathbb{CP}(0)} \forall n, \alpha{::}\mathbb{N}.\, \text{list}[n]^{\alpha}\, \tau \xrightarrow{\mathbb{CP}(0)} \\
\qquad \exists \beta{::}\mathbb{N}.\, \beta \leqslant \alpha \ \&\ (\text{list}[\left\lceil \frac{n}{2} \right\rceil]^{\beta}\, \tau \times \text{list}[\left\lfloor \frac{n}{2} \right\rfloor]^{\alpha-\beta}\, \tau))
\end{array}
$$

Using `bsplit` we define the balanced fold function, `bfold`. The function applies only to non-empty lists (reflected in its type later), so the nil case is omitted.

$$
\begin{array}{l}
\texttt{fix bfold}(\_).\Lambda.\Lambda.\lambda l.\texttt{case } l \texttt{ of} \\
\quad \text{nil} \ \to \cdots \\
\mid h_1 :: tl_1 \ \to \texttt{case } tl_1 \texttt{ of} \\
\qquad\qquad \text{nil} \ \to \text{cons}(h_1, \text{nil} \,) \\
\qquad\quad \mid \_ :: \_ \ \to \text{let } r = \texttt{bsplit} \ ()[\,][\,] \ l \text{ in} \\
\qquad\qquad\qquad\qquad \text{unpack } r \text{ as } r' \text{ in} \\
\qquad\qquad\qquad\qquad \text{clet } r' \text{ as } x \text{ in} \\
\qquad\qquad\qquad\qquad f \ (\texttt{bfold} \ ()[\,][\,] \ \pi_1 x, \texttt{bfold} \ ()[\,][\,] \ \pi_2 x)
\end{array}
$$

We first derive a type for `bfold` informally, and then show how the type is established in DuCostIt. Assume that the argument `l` has type $\text{list}[n]^{\alpha}\, \tau$. We count how many times change propagation may have to reapply `f` in updating `bfold`'s trace, which is a nearly balanced tree of height $H = \lceil \texttt{log}_2(n) \rceil$. Counting levels from the deepest leaves upward (leaves have level 0), the number of applications of `f` at level `i` in the trace is at most 2^{H-i}. If α leaves change, at most α of these applications must be recomputed. Consequently, the maximum number of recomputations of `f` at level `i` is $\texttt{min}(\alpha, 2^{H-i})$. If the dynamic stability of `f` is `t`, the

dynamic stability of `bfold` is $P(n,\alpha,t) = \sum_{k=0}^{\lceil \log_2(n)\rceil} t \cdot \min(\alpha, 2^{\lceil \log_2(n)\rceil - k})$. So, in principle, we should be able to give `bfold` the following type.

$$\texttt{bfold} : \text{unit}_r \xrightarrow{\mathbb{CP}(0)} \forall n, \alpha{::}\mathbb{N}.\, \text{list}[n]^{\alpha}\,(U\,\text{int}) \xrightarrow{\mathbb{CP}(P(n,\alpha,t))} U\,\text{int}$$

The expression $P(n,\alpha,t)$ may look complex, but it is in $O(t\cdot(\alpha+\alpha\cdot \log_2(n/\alpha)))$.[26] Although the type above is correct, we will see soon that in typing the recursive calls in `bfold`, we need to know that `bfold`'s type is annotated with $\Box$. Hence, the actual type we assign to `bfold` is stronger.

$$\texttt{bfold} : \Box\,(\text{unit}_r \xrightarrow{\mathbb{CP}(0)} \forall n, \alpha{::}\mathbb{N}.\, \text{list}[n]^{\alpha}\,(U\,\text{int}) \xrightarrow{\mathbb{CP}(P(n,\alpha,t))} U\,\text{int}) \quad (3)$$

[26] *To prove this, split the summation in $P(n,\alpha,t)$ into two: one for $k \leqslant \lceil \log_2(n)\rceil - \lceil \log_2(\alpha)\rceil$ and the other for $k > \lceil \log_2(n)\rceil - \lceil \log_2(\alpha)\rceil$. The appendix has the details.*

We explain how `bfold`'s type is established in DuCostIt. The interesting case starts where `bsplit` is invoked. From the type of `bsplit`, we know that $\pi_1 x$ and $\pi_2 x$ in the body of `bfold` have types $\text{list}[\lceil \frac{n}{2}\rceil]^{\beta}\,\tau$ and $\text{list}[\lfloor \frac{n}{2}\rfloor]^{\alpha-\beta}\,\tau$, respectively for some β. Inductively, the change propagation costs of (`bfold f` $\pi_1 x$) and (`bfold f` $\pi_2 x$) are $P(\lceil \frac{n}{2}\rceil,\beta,t)$ and $P(\lfloor \frac{n}{2}\rfloor,\alpha-\beta,t)$, respectively. Hence, the change propagation cost of the whole body of `bfold` is $t + P(\lceil \frac{n}{2}\rceil,\beta,t) + P(\lfloor \frac{n}{2}\rfloor,\alpha-\beta,t)$. The additional t accounts for the only application of `f` in the body of `bfold` (non-primitive operations have zero cost and `bsplit` also has zero cost). Hence, to complete the typing, we must establish the following inequality.

$$t + P(\left\lceil \frac{n}{2}\right\rceil,\beta,t) + P(\left\lfloor \frac{n}{2}\right\rfloor,\alpha-\beta,t) \leqslant P(n,\alpha,t) \quad (4)$$

This is an easily established arithmetic tautology (the proof is shown in Appendix D.1), *except* when $\alpha \doteq 0$. When $\alpha \doteq 0$, the right side of the inequality is 0 but we don't necessarily have $t \leqslant 0$. So, in order to proceed, we consider the cases $\alpha \doteq 0$ and $\alpha > 0$ separately. This requires a typing rule for case analysis on the index domain, which poses no theoretical difficult.[27]. The $\alpha > 0$ case succeeds as described above. For $\alpha \doteq 0$, we use our co-monadic reasoning principle. With $\alpha \doteq 0$, the types of $\pi_1 x$ and $\pi_2 x$ are equivalent (formally, via subtyping) to $\text{list}[\lceil \frac{n}{2}\rceil]^{0}\,\tau$ and $\text{list}[\lfloor \frac{n}{2}\rfloor]^{0}\,\tau$, respectively. Since, no elements in these lists can change, we use **1-□** subtyping rule (in Figure 34) to promote the types to $\Box\,\text{list}[\lceil \frac{n}{2}\rceil]^{0}\,\tau$ and $\Box\,\text{list}[\lfloor \frac{n}{2}\rfloor]^{0}\,\tau$, respectively. At this point, the type of every variable occurring in the expression `f` $(\langle$`bfold f` $\pi_1 x$, `bfold f` $\pi_2 x\rangle)$, including the variable `bfold`, has annotation $\Box\cdot$. By our co-monadic reasoning principle, the change propaga-

[27] *A similar rule, called **r-split**, exists in RelCost as well.*

tion cost of this expression and, hence, the body of `bfold`, must be $\Box$, which is trivially no more than $P(n, \alpha, t)$. This completes our argument.

Observe that the inference of the annotation $\Box \cdot$ on the types of $\pi_1 x$ and $\pi_2 x$ is conditional on the constraint $\alpha \doteq 0$. Subtyping, which is aware of constraints, plays an essential role in determining these annotations and in making our co-monadic reasoning principle useful. Also, the fact that we have to consider the cases $\alpha \doteq 0$ and $\alpha > 0$ separately is not as surprising as it may seem. The case $\alpha \doteq 0$ corresponds to a sub-trace whose leaves have not changed. Since change propagation is a bottom-up procedure, it will bypass this sub-trace completely, incurring no cost. This is exactly what our analysis for $\alpha \doteq 0$ establishes.

Using the type (3) of `bfold`, we can show that for $f : \Box((\tau \times \tau) \xrightarrow{\mathbb{CP}(t)} \tau)$ and $l : \text{list}[n]^{\alpha}\,\tau$, the dynamic stability of (`bfold f l`) is in $O(\texttt{log}_2(n))$ when $\alpha \in O(1)$ and in $O(n)$ when $\alpha \in O(n)$, assuming that t is constant. This dynamic stability is asymptotically tight.

8

DUCOSTIT'S TYPE SYSTEM

▶ SYNOPSIS In this chapter, we present the technical ideas behind DuCostIt, making comparisons to RelCost's type system as we go along. The underlying programming language for DuCostIt is the language Cost^{ML}—same as RelCost's, introduced in Chapter 4. The design of DuCostIt's type system reflects the underlying semantic model, presented in Chapter 9, which differs from RelCost's.

TYPES We briefly describe how DuCostIt's type syntax (shown in Figure 17) differs from RelCost's (shown in Figure 1). The only difference is in types that capture closures. For unary types, unlike RelCost's unary function type $A_1 \xrightarrow{\text{exec}(k,t)} A_2$, that tracks both upper and lower bounds on the execution cost of the function body, DuCostIt's unary function type $A_1 \xrightarrow{\mathbb{FS}(t)} A_2$ only tracks upper bounds t on the from-scratch execution cost of the function body (hence the $\mathbb{FS}$ annotation on the arrow). For relational types, unlike RelCost's relational type $\tau_1 \xrightarrow{\text{diff}(t)} \tau_2$, that tracks upper bounds t on the relative costs of the two function bodies, DuCostIt's relational type $\tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2$ tracks upper bounds on the change propagation cost of the function body (hence the $\mathbb{CP}$ annotation on the arrow). Similar annotations appear on universally quantified types as well as typing judgments.

The unrelated type $U\,A$ specifies values of type A that may change arbitrarily between the initial and incremental run. [28] Types other than

[28] $U\,A$ *can be generalized to* $U\,(A_1, A_2)$ *as in RelCost, but simplified versions suffice for all the examples we considered.*

$$
\begin{array}{lrcl}
\text{Unary types} & A & ::= & \text{int} \mid A_1 \times A_2 \mid A_1 + A_2 \mid \text{list}[n]\,A \mid \\
& & & A_1 \xrightarrow{\mathbb{FS}(t)} A_2 \mid \forall i \overset{\mathbb{FS}(t)}{::} S.\,A \mid \\
& & & \exists i{::}S.\,A \mid C \;\&\; A \mid C \supset A \mid \text{unit} \\
\text{Relational types} & \tau & ::= & \text{int}_r \mid \tau_1 \times \tau_2 \mid \tau_1 + \tau_2 \mid \text{list}[n]^{\alpha}\,\tau \mid \\
& & & \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2 \mid \forall i \overset{\mathbb{CP}(t)}{::} S.\,\tau \mid \exists i{::}S.\,\tau \mid \\
& & & C \;\&\; \tau \mid C \supset \tau \mid \text{unit}_r \mid U\,A \mid \Box\,\tau
\end{array}
$$

Figure 17: Syntax of DuCostIt's types

$\mathsf{U}\,A$ specify values that cannot change structurally. The stronger type $\Box\,\tau$ represents values of τ that cannot even *depend* on changeable variables from outer contexts and, hence, cannot change at all. Thus, if $y : \mathsf{U}\,\mathsf{int}$, then $\lambda x.y + x$ does not have type $\Box\,(\mathsf{U}\,\mathsf{int} \xrightarrow{\mathbb{CP}(t)} \mathsf{U}\,\mathsf{int})$, but $\lambda x.x$ does. As in RelCost, $\Box\,\tau$ is a co-monadic type.

DuCostIt's underlying programming language, index term and constraint grammar are identical to RelCost's; hence we don't relist them here.

Before we explain DuCostIt's typing rules, we describe DuCostIt's abstract evaluation and change propagation semantics.

8.1 ABSTRACT EVALUATION AND CHANGE PROPAGATION SEMANTICS

In this section, we define two abstract cost-counting semantics: one for from-scratch execution and one for change propagation.

EVALUATION SEMANTICS AND TRACES DuCostIt's big-step call-by-value evaluation judgment $e \Downarrow^{f} T$ states that expression e evaluates to a trace T with evaluation cost f. In DuCostIt's semantic model, the cost f also interacts with the step-index. [29] The trace T is a representation of the entire big-step derivation and explicitly includes the final and all intermediate values. It is a pair $\langle v, D\rangle$, where v is the result of the evaluation and D is a derivation, which recursively contains subtraces. Its syntax is shown in Figure 18. For every big-step evaluation rule, there is a corresponding derivation constructor. Evaluation rules are shown in Figures 19 and 20.

[29] *Like in RelCost, we could have tracked two costs: one to interact with the actual execution cost bounds and one to interact with the step-index. However, unlike RelCost, we are interested in accounting for the domain-specific cost, i. e. the cost of change propagation, so we see no harm in letting the execution cost bound interact with the step-index for simplicity of the proofs.*

Traces	T	$::=$	$\langle v, D\rangle$
Derivations	D	$::=$	$\mathtt{n} \mid \langle T_1, T_2\rangle \mid \pi_1\ T \mid \pi_2\ T \mid \mathtt{inl}\ T \mid \mathtt{inr}\ T \mid$
			$\mathtt{case}_{\mathtt{inl}}(T, T_r) \mid \mathtt{case}_{\mathtt{inr}}(T, T_r) \mid \mathtt{nil} \mid$
			$\mathtt{cons}(T_1, T_2) \mid \mathtt{case}_{\mathtt{nil}}(T, T_r) \mid \mathtt{case}_{\mathtt{cons}}(T, T_r) \mid$
			$\mathtt{fix} f(x).e \mid \mathtt{app}(T_1, T_2, T_r) \mid \mathtt{prim}_{\mathtt{app}}(T, \zeta) \mid$
			$\Lambda.e \mid \mathtt{iApp}(T, T_r) \mid \mathtt{pack}\ T \mid \mathtt{unpack}(T, x, T_r) \mid$
			$\mathtt{let}(x, T_1, T_2) \mid \mathtt{clet}_{\mathtt{as}}(x, T, T_r) \mid ()$

Figure 18: Traces

Derivation constructors for case analysis record which branch was taken using subscripts like `inl` or `inr`. The derivation construct for function applications records the final value in the trace, along with all the intermediate traces of the function, argument, and body expressions. Similar to RelCost's evaluation semantics, the cost model is parametric over construct-dependent meta-symbols like c_{app} that the type system also uses. The helper meta function $V(\cdot)$ returns the final value contained in a trace: $V(\langle v, D\rangle) = v$.

$\boxed{e \Downarrow^{f} T}$ Expression e evaluates with cost f to trace $T = \langle v, D\rangle$, containing the final value v and the derivation D.

$$\frac{}{n \Downarrow^{c_n} \langle n, n\rangle}\ \textbf{e-const} \qquad \frac{e_1 \Downarrow^{f_1} T_1 \qquad e_1 \Downarrow^{f_2} T_2 \qquad v_i = V(T_i)}{\langle e_1, e_2\rangle \Downarrow^{f_1+f_2} \langle\langle v_1, v_2\rangle, \langle T_1, T_2\rangle\rangle}\ \textbf{e-pair}$$

$$\frac{e \Downarrow^{f} T \qquad \langle v_1, v_2\rangle = V(T)}{\pi_1 e \Downarrow^{f+c_{proj}} \langle v_1, \pi_1\ T\rangle}\ \textbf{e-proj}_1 \qquad \frac{e \Downarrow^{f} T \qquad \langle v_1, v_2\rangle = V(T)}{\pi_2 e \Downarrow^{f+c_{proj}} \langle v_2, \pi_2\ T\rangle}\ \textbf{e-proj}_2$$

$$\frac{e \Downarrow^{f} T \qquad v = V(T)}{\mathrm{inl}\ e \Downarrow^{f} \langle \mathrm{inl}\ v, \mathtt{inl}\ T\rangle}\ \textbf{e-inl} \qquad \frac{e \Downarrow^{f} T \qquad v = V(T)}{\mathrm{inr}\ e \Downarrow^{f} \langle \mathrm{inr}\ v, \mathtt{inr}\ T\rangle}\ \textbf{r-inr}$$

$$\frac{e \Downarrow^{f} T \qquad \mathrm{inl}\ v = V(T) \qquad e_1[v/x] \Downarrow^{f_r} T_r \qquad v_r = V(T_r)}{\mathrm{case}\ (e, x.e_1, y.e_2) \Downarrow^{f+f_r+c_{case}} \langle v_r, \mathtt{case}_{\mathtt{inl}}(T, T_r)\rangle}\ \textbf{ev-case-l}$$

$$\frac{e \Downarrow^{f} T \qquad \mathrm{inr}\ v = V(T) \qquad e_2[v/y] \Downarrow^{f_r} T_r \qquad v_r = V(T_r)}{\mathrm{case}\ (e, x.e_1, y.e_2) \Downarrow^{f+f_r+c_{case}} \langle v_r, \mathtt{case}_{\mathtt{inr}}(T, T_r)\rangle}\ \textbf{ev-case-r}$$

$$\frac{}{\mathrm{nil}\ \Downarrow^{0} \langle \mathrm{nil}\ , \mathtt{nil}\rangle}\ \textbf{ev-nil}$$

$$\frac{e_1 \Downarrow^{f_1} T_1 \qquad e_2 \Downarrow^{f_2} T_2 \qquad v_i = V(T_i)}{\mathrm{cons}(e_1, e_2) \Downarrow^{f_1+f_2} \langle \mathrm{cons}(v_1, v_2), \mathtt{cons}(T_1, T_2)\rangle}\ \textbf{ev-cons}$$

$$\frac{e \Downarrow^{f} T \qquad e_1 \Downarrow^{f_r} T_1 \qquad \mathrm{nil}\ = V(T) \qquad v_r = V(T_r)}{\mathrm{case}\ e\ \mathrm{of\ nil}\ \to e_1 \mid h :: tl \to e_2 \Downarrow^{f+f_r+c_{caseL}} \langle v_r, \mathtt{case}_{\mathtt{nil}}(T, T_r)\rangle}\ \textbf{ev-case-nil}$$

$$\frac{\begin{array}{c} e \Downarrow^{f} T \\ \mathrm{cons}(v_h, v_{tl}) = V(T) \qquad e_2[v_h/h, v_{tl}/tl] \Downarrow^{f_r} T_r \qquad v_r = V(T_r) \end{array}}{\mathrm{case}\ e\ \mathrm{of\ nil}\ \to e_1 \mid h :: tl \to e_2 \Downarrow^{f+f_r+c_{caseL}} \langle v_r, \mathtt{case}_{\mathtt{cons}}(T, T_r)\rangle}\ \textbf{ev-case-cons}$$

$$\frac{}{\mathrm{fix}\ f(x).e \Downarrow^{0} \langle \mathrm{fix}\ f(x).e, \mathtt{fix} f(x).e\rangle}\ \textbf{ev-fix}$$

Figure 19: From-scratch evaluation semantics (Part 1)

$\boxed{e \Downarrow^{f} \langle v, D\rangle}$ Expression e evaluates with cost f to trace $T = \langle v, D\rangle$, containing the final value v and the derivation D.

$$\frac{e_1 \Downarrow^{f_1} T_1 \qquad e_2 \Downarrow^{f_2} T_2 \qquad \text{fix } f(x).e = V(T_1) \qquad v_2 = V(T_2) \qquad e[v_2/x, (\text{fix } f(x).e)/f] \Downarrow^{f_r} T_r \qquad v_r = V(T_r)}{e_1\ e_2 \Downarrow^{f_1+f_2+f_r+c_{app}} \langle v_r, \mathtt{app}(T_1, T_2, T_r)\rangle}\ \textbf{ev-app}$$

$$\frac{e \Downarrow^{f} T \qquad v = V(T) \qquad \zeta(v) = (f_r, v_r)}{\zeta\ e \Downarrow^{f+f_r+c_{primapp}} \langle v_r, \mathtt{prim}_{\mathtt{app}}(T, \zeta)\rangle}\ \textbf{ev-primapp}$$

$$\frac{}{\Lambda.e \Downarrow^{0} \langle \Lambda.e, \Lambda.e\rangle}\ \textbf{ev-Lam}$$

$$\frac{e \Downarrow^{f} T \qquad \Lambda.e' = V(T) \qquad e' \Downarrow^{f_r} T_r \qquad v_r = V(T_r)}{e[\,] \Downarrow^{f+f_r} \langle v_r, \mathtt{iApp}(T, T_r)\rangle}\ \textbf{ev-iApp}$$

$$\frac{e \Downarrow^{f} T \qquad v = V(T)}{\text{pack } e \Downarrow^{f} \langle \text{pack } v, \mathtt{pack}\ T\rangle}\ \textbf{ev-pack}$$

$$\frac{e_1 \Downarrow^{f_1} T_1 \qquad \text{pack } v = V(T_1) \qquad e_2[v/x] \Downarrow^{f_r} T_r \qquad v_r = V(T_r)}{\text{unpack } e_1 \text{ as } x \text{ in } e_2 \Downarrow^{f_1+f_r} \langle v_r, \mathtt{unpack}(T_1, x, T_r)\rangle}\ \textbf{ev-unpack}$$

$$\frac{e_1 \Downarrow^{f_1} T_1 \qquad v_1 = V(T_1) \qquad e_2[v_1/x] \Downarrow^{f_r} T_r \qquad v_r = V(T_r)}{\text{let } x = e_1 \text{ in } e_2 \Downarrow^{f_1+f_r+c_{let}} \langle v_r, \mathtt{let}(x, T_1, T_r)\rangle}\ \textbf{ev-let}$$

$$\frac{e_1 \Downarrow^{f_1} T_1 \qquad v_1 = V(T_1) \qquad e_2[v_1/x] \Downarrow^{f_r} T_r \qquad v_r = V(T_r)}{\text{clet } e_1 \text{ as } x \text{ in } e_2 \Downarrow^{f_1+f_r} \langle v_r, \mathtt{clet}_{\mathtt{as}}(x, T_1, T_r)\rangle}\ \textbf{ev-clet}$$

$$\frac{e \Downarrow^{f} T}{\text{celim}_{\supset}\ e \Downarrow^{f} T}\ \textbf{ev-celim} \qquad \frac{}{() \Downarrow^{0} \langle (), ()\rangle}\ \textbf{ev-unit}$$

Figure 20: From-scratch evaluation semantics (Part 2)

CHANGES AND BIEXPRESSIONS In order to formalize change propagation, we first need notation to specify where an expression has changed in the actual incremental run.[30] For this we define *bivalues* and *biexpressions*. A biexpression (bivalue), denoted $\mathfrak{ee}$ ($\mathfrak{w}$), represents in a single syntax two expressions (values)—the original one and the updated one—that share most structure, but may differ at some leaves. To represent differing leaves, we use the bivalue constructor $\mathtt{new}(v_1, v_2)$, which represents the initial value v_1 in the first run and the updated value v_2 in the second run. v_1 and v_2 do not have to be related to each other. For integer leaves that stay the same, we use the biexpression

[30] *RelCost does not need special constructs for specifying where the values for the two programs differ since the two related programs have the same semantics, whereas in DuCostIt, biexpressions are needed to trigger change propagation semantics to switch from propagating updates to re-execution (explained in Section 8.2).*

$$
\begin{array}{lll}
\text{Bi-values} & \mathtt{w} ::= & \mathtt{keep}(\mathtt{n}) \mid \mathtt{new}(\mathtt{v},\mathtt{v}') \mid \langle \mathtt{w}_1, \mathtt{w}_2 \rangle \mid \mathtt{inl}\ \mathtt{w} \mid \mathtt{inr}\ \mathtt{w} \mid \\
& & \mathtt{nil} \mid \mathtt{cons}(\mathtt{w}_1, \mathtt{w}_2) \mid \mathtt{fix}\ \mathtt{f}(\mathtt{x}).\mathtt{ee} \mid \Lambda.\mathtt{ee} \mid \mathtt{pack}\ \mathtt{w} \mid () \\
\text{Bi-expressions} & \mathtt{ee} ::= & \mathtt{x} \mid \mathtt{keep}(\mathtt{n}) \mid \mathtt{new}(\mathtt{v},\mathtt{v}') \mid \langle \mathtt{ee}_1, \mathtt{ee}_2 \rangle \mid \pi_1\ \mathtt{ee} \mid \pi_2\ \mathtt{ee} \mid \\
& & \mathtt{inl}\ \mathtt{ee} \mid \mathtt{inr}\ \mathtt{ee} \mid (\mathtt{case}(\mathtt{ee}, \mathtt{x}.\mathtt{ee}_1, \mathtt{y}, \mathtt{ee}_2)) \mid \\
& & \mathtt{nil} \mid \mathtt{cons}(\mathtt{ee}_1, \mathtt{ee}_2) \mid \mathtt{fix}\ \mathtt{f}(\mathtt{x}).\mathtt{ee} \mid \mathtt{ee}_1\ \mathtt{ee}_2 \mid \\
& & (\mathtt{case}_L\ \mathtt{ee}\ \mathtt{of}\ \mathtt{nil} \rightarrow \mathtt{ee}_1 \mid \mathtt{h} :: \mathtt{tl} \rightarrow \mathtt{ee}_2) \mid \\
& & \zeta\ \mathtt{ee} \mid \Lambda.\mathtt{ee} \mid \mathtt{ee}[] \mid \mathtt{pack}\ \mathtt{ee} \mid \mathtt{unpack}\ \mathtt{ee}\ \mathtt{as}\ \mathtt{x}\ \mathtt{in}\ \mathtt{ee}' \mid \\
& & \mathtt{let}\ \mathtt{x} = \mathtt{ee}_1\ \mathtt{in}\ \mathtt{ee}_2 \mid \mathtt{clet}\ \mathtt{ee}_1\ \mathtt{as}\ \mathtt{x}\ \mathtt{in}\ \mathtt{ee}_2 \mid ()
\end{array}
$$

$$\boxed{\mathtt{stable}(\mathtt{w}) \triangleq \mathtt{new}(v,v') \notin \mathtt{w} \text{ and } \mathtt{stable}(\mathtt{ee}) \triangleq \mathtt{new}(v,v') \notin \mathtt{ee}}$$

Figure 21: Syntax of bi-values and bi-expression

construct $\mathtt{keep}(\mathtt{n})$, all other constructs of Cost^{ML} can be lifted to the corresponding biexpression syntax as shown in Figure 21.

For instance, $\mathtt{fix}\ \mathtt{f}(\mathtt{x}).(\mathtt{x} + \mathtt{new}(1,2))$ represents $\mathtt{fix}\ \mathtt{f}(\mathtt{x}).\,\mathtt{x} + 1$ in the first run and $\mathtt{fix}\ \mathtt{f}(\mathtt{x}).\,\mathtt{x} + 2$ in the second run. More generally, we define the functions $L(\mathtt{ee})$ and $R(\mathtt{ee})$ that project the first-run ("left") and second-run ("right") expressions from $\mathtt{ee}$ as the homomorphic liftings of the following rules: $L(\mathtt{keep}(\mathtt{r})) = R(\mathtt{keep}(\mathtt{r})) = \mathtt{r}$, $L(\mathtt{new}(v_1, v_2)) = v_1$ and $R(\mathtt{new}(v_1, v_2)) = v_2$.

Both bivalues and biexpressions are typed to prevent modifying a stable input (of type $\Box\tau$) or to prevent ill-typed changes such as modifying 1 to true. For instance, changing two elements of a list that only allows a single change would not be permitted since the biexpression $\mathtt{cons}(\mathtt{new}(1,2), \mathtt{cons}(\mathtt{new}(0,5), \mathtt{w}))$ cannot be given a type $\text{list}[\mathtt{n}]^1\ U\ \text{int}$.

The typing rules for bivalues and biexpressions are shown in Figure 23. The bivalue typing judgment

$$\Delta; \Phi; \Gamma \vdash \mathtt{w} \gg \tau$$

states that the bivalue $\mathtt{w}$ represents a valid change from an initial value $L(\mathtt{w})$ of type τ to the modified value $R(\mathtt{w})$ of type τ. The typing rules for bivalues mirror those for values. The construct $\mathtt{keep}(\mathtt{n})$ is typed at int_r since it represents an integer that did not change. The construct $\mathtt{new}(v_1, v_2)$ can be typed at $U\,A$ if the values v_1 and v_2 can be typed in unary mode at type A.

There is only one rule, **bi-expr**, for typing biexpressions. This rule uses explicit substitutions for technical convenience. We could also have written equivalent syntax-directed rules for typing biexpressions.

$\boxed{\Delta; \Phi; \Gamma \vdash w \gg \tau}$ Bi-value typing

$\boxed{\Delta; \Phi; \Gamma \vdash ee \gg \tau \mid t}$ Bi-expression typing

$$\frac{}{\Delta; \Phi; \Gamma \vdash \mathsf{keep}(n) \gg \mathrm{int}_r}\ \textbf{bi-keep}$$

$$\frac{\Delta; \Phi; \cdot \vdash_{\mathbb{FS}} v : A \mid t \qquad \Delta; \Phi; \cdot \vdash_{\mathbb{FS}} v' : A \mid t'}{\Delta; \Phi; \Gamma \vdash \mathsf{new}(v, v') \gg \mathsf{U}\, A}\ \textbf{bi-new}$$

$$\frac{}{\Delta; \Phi; \Gamma \vdash () \gg \mathrm{unit}_r}\ \textbf{bi-unit} \qquad \frac{\Delta; \Phi; \Gamma \vdash w \gg \tau_1}{\Delta; \Phi; \Gamma \vdash \mathtt{inl}\ w \gg \tau_1 + \tau_2}\ \textbf{bi-inl}$$

$$\frac{\Delta; \Phi; \Gamma \vdash w \gg \tau_2}{\Delta; \Phi; \Gamma \vdash \mathtt{inr}\ w \gg \tau_1 + \tau_2}\ \textbf{bi-inr}$$

$$\frac{\Delta; \Phi; x : \tau_1, f : \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2, \Gamma \vdash ee \gg \tau_2 \mid t}{\Delta; \Phi; \Gamma \vdash \mathtt{fix}\ f(x).\, ee \gg \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2}\ \textbf{bi-fix}$$

$$\frac{\begin{array}{l}\Delta; \Phi; x : \tau_1, f : \square(\tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2), \Gamma \vdash ee \gg \tau_2 \mid t \\ \forall x \in \Gamma.\ \ \Delta; \Phi \models \Gamma(x) \sqsubseteq \square\Gamma(x) \qquad \mathtt{stable}(ee)\end{array}}{\Delta; \Phi; \Gamma, \Gamma' \vdash \mathtt{fix}\ f(x).ee \gg \square(\tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2)}\ \textbf{bi-fix-NC}$$

$$\frac{\Delta; \Phi; \Gamma \vdash w_1 \gg \tau_1 \qquad \Delta; \Phi; \Gamma \vdash w_2 \gg \tau_2}{\Delta; \Phi; \Gamma \vdash \langle w_1, w_2 \rangle \gg \tau_1 \times \tau_2}\ \textbf{bi-prod}$$

$$\frac{}{\Delta; \Phi; \Gamma \vdash \mathtt{nil} \gg \mathrm{list}[0]^{\alpha}\, \tau}\ \textbf{bi-nil}$$

$$\frac{\Delta; \Phi; \Gamma \vdash w_1 \gg \tau \qquad \Delta; \Phi; \Gamma \vdash w_2 \gg \mathrm{list}[n]^{\alpha}\, \tau}{\Delta; \Phi; \Gamma \vdash \mathsf{cons}(w_1, w_2) \gg \mathrm{list}[n+1]^{\alpha+1}\, \tau}\ \textbf{bi-cons}$$

$$\frac{\Delta; \Phi; \Gamma \vdash w_1 \gg \square\tau \qquad \Delta; \Phi; \Gamma \vdash w_2 \gg \mathrm{list}[n]^{\alpha}\, \tau}{\Delta; \Phi; \Gamma \vdash \mathsf{cons}(w_1, w_2) \gg \mathrm{list}[n+1]^{\alpha}\, \tau}\ \textbf{bi-cons-}\square$$

$$\frac{\Psi; i :: S, \Delta; \Phi; \Gamma \vdash ee \gg \tau \mid t \qquad i \notin \mathrm{FV}(\Phi; \Gamma)}{\Delta; \Phi; \Gamma \vdash \Lambda.ee \gg \forall i \overset{\mathbb{CP}(t)}{::} S.\, \tau}\ \textbf{bi-Lam}$$

$$\frac{\Delta; \Phi; \Gamma \vdash w \gg \tau\{I/t\} \qquad \Delta I :: S}{\Delta; \Phi; \Gamma \vdash \mathtt{pack}\ w \gg \exists t{::}S.\, \tau}\ \textbf{bi-pack}$$

Figure 22: DuCostIt bi-expression typing rules (Part 1)

$\boxed{\Delta;\Phi;\Gamma \vdash w \gg \tau \text{ and } \Delta;\Phi;\Gamma \vdash ee \gg \tau \mid t}$ Bi-value and bi-expression typing

$$\frac{\Delta;\Phi \wedge C;\Gamma \vdash w \gg \tau}{\Delta;\Phi;\Gamma \vdash w \gg C \supset \tau}\ \textbf{bi-c-imp}$$

$$\frac{\Delta;\Phi \models C \qquad \Delta;\Phi \wedge C;\Gamma \vdash w \gg \tau}{\Delta;\Phi;\Gamma \vdash w \gg C \,\&\, \tau}\ \textbf{bi-c-prod}$$

$$\frac{\Delta;\Phi;\Gamma \vdash w \gg \tau \qquad \forall x \in \Gamma.\ \Delta;\Phi \models \Gamma(x) \sqsubseteq \Box\Gamma(x) \qquad \texttt{stable}(w)}{\Delta;\Phi;\Gamma,\Gamma' \vdash w \gg \Box\tau}\ \textbf{bi-nochange}$$

$$\frac{\Delta;\Phi;\Gamma \vdash w \gg \tau \qquad \Delta;\Phi \models \tau \sqsubseteq \tau'}{\Delta;\Phi;\Gamma \vdash w \gg \tau'}\ \textbf{bi-}\sqsubseteq$$

$$\frac{\overline{\Delta;\Phi;\Gamma \vdash w_i \gg \tau_i} \qquad \Delta;\Phi;\overline{x_i : \tau_i},\Gamma \vdash_{\mathbb{CP}} e : \tau \mid t}{\Delta;\Phi;\Gamma \vdash \ulcorner e \urcorner[\overline{w_i/x_i}] \gg \tau \mid t}\ \textbf{bi-expr}$$

Figure 23: DuCostIt bivalue and biexpression typing rules (Part 2)

$$\begin{aligned}
\ulcorner x \urcorner &= x \\
\ulcorner n \urcorner &= \texttt{keep}(n) \\
\ulcorner () \urcorner &= () \\
\ulcorner \langle e_1, e_2 \rangle \urcorner &= \langle \ulcorner e_1 \urcorner, \ulcorner e_2 \urcorner \rangle \\
\ulcorner \texttt{nil} \urcorner &= \texttt{nil} \\
\ulcorner \texttt{cons}(e_1, e_2) \urcorner &= \texttt{cons}(\ulcorner e_1 \urcorner, \ulcorner e_2 \urcorner) \\
\ulcorner \texttt{pack}\ e \urcorner &= \texttt{pack}\ \ulcorner e \urcorner \\
\ulcorner \text{fix } f(x).e \urcorner &= \text{fix } f(x) \ulcorner e \urcorner \\
&\vdots
\end{aligned}$$

Figure 24: Lift a value (expression) into a bivalue (biexpression)

The notation $\ulcorner e \urcorner$ denotes the biexpression that represents e in both the first and second runs. It is obtained by replacing every primitive constant like $\mathtt{n}$ in e with $\mathtt{keep}(\mathtt{n})$ (definition shown in Figure 24).

8.2 CHANGE PROPAGATION

Change propagation is formalized abstractly by the judgment

$$\langle \mathsf{T}, \mathfrak{ee} \rangle \curvearrowright \mathfrak{w}', \mathsf{T}', c'.$$

It takes as inputs the trace T and the biexpression $\mathfrak{ee}$ and it returns $\mathfrak{w}'$, T' and c'. The input T must be the trace that is obtained from executing the original expression $\mathrm{L}(\mathfrak{ee})$. The bivalue $\mathfrak{w}'$ resulting from change propagation represents two values, $\mathrm{L}(\mathfrak{w}')$ and $\mathrm{R}(\mathfrak{w}')$, which are the results of evaluating the original and modified expressions, respectively. $\mathfrak{w}'$ is crucial for directing change-propagation on when to switch to from-scratch execution.[31] The output T' is the trace of the modified expression $\mathrm{R}(\mathfrak{ee})$. The non-negative number c' represents the total cost incurred in change propagation.

[31] *Actual implementations of change propagation never would construct bivalues (biexpressions) and, hence, we do not count any cost for constructing or analyzing it during change propagation.*

Before we explain individual change propagation rules, we review key ideas behind our abstract change propagation semantics.

- The total change propagation cost of a biexpression is obtained by summing the costs of its sub-biexpressions.
- During change propagation, whenever the resulting bi-value of an eliminated biexpression has changed, i. e., it is $\mathtt{new}(v, v')$, since there is no corresponding computation recorded in the trace, the continuation switches from change propagation to from-scratch execution (e. g. rules **cp-app-new, cp-case-inl$_\iota$, cp-unpack-new**).
- The change propagation rules case analyze the syntax of $\mathfrak{ee}$ and they are deterministic, i. e., for a given biexpression and a trace, there is only one way to propagate the changes.
- In all the rules except **cp-nochange**, we assume that the input $\mathfrak{ee}$ satisfies $\neg\mathtt{stable}(\mathfrak{ee})$, i. e. it has a change in its subparts.

The change propagation rules are shown in Figures 25 to 27. Below, we explain selected rules.

The most important rule is **cp-nochange** that captures re-use of non-changing subcomputations for free. Its premise, $\mathtt{stable}(\mathfrak{ee})$ holds when $\mathfrak{ee}$ does not contain $\mathtt{new}(\cdot,\cdot)$ anywhere, i.e., when $\mathfrak{ee}$ represents an expression that has not changed. In this case, the value v stored in the

$\boxed{\langle T, ee\rangle \curvearrowright w', T', c'}$ Change propagation with cost-counting

In all the remaining rules except **cp-nochange**, we assume that the input ee satisfies $\neg\mathtt{stable}(ee)$.

$$\frac{\mathtt{stable}(ee)}{\langle\langle v, D\rangle, ee\rangle \curvearrowright \ulcorner v \urcorner, \langle v, D\rangle, 0}\ \textbf{cp-nochange}$$

$$\frac{}{\langle\langle v, D\rangle, \mathtt{new}(\_, v')\rangle \curvearrowright \mathtt{new}(v, v'), \langle v', v'\rangle, 0}\ \textbf{cp-new}$$

$$\frac{\langle T_1, ee_1\rangle \curvearrowright w_1', T_1', c_1' \qquad \langle T_2, ee_2\rangle \curvearrowright w_2', T_2', c_2' \qquad v_i' = \mathtt{V}(T_i')}{\langle\langle \_, \langle T_1, T_2\rangle\rangle, (ee_1, ee_2)\rangle \curvearrowright (w_1', w_2'), \langle\langle v_1', v_2'\rangle, \langle T_1', T_2'\rangle\rangle, c_1' + c_2'}\ \textbf{cp-pair}$$

$$\frac{\langle T, ee\rangle \curvearrowright (w_1, w_2), T', c' \qquad \langle v_1', v_2'\rangle = \mathtt{V}(T')}{\langle \pi_1\ T, \pi_1 ee\rangle \curvearrowright w_1, \langle v_1', \pi_1\ T'\rangle, c'}\ \textbf{cp-proj}_1$$

$$\frac{}{\begin{array}{c}\langle\langle \mathrm{fix}\ f(x).e', T\rangle, \mathtt{fix}\ f(x).ee\rangle \curvearrowright \\ \mathtt{fix}\ f(x).ee, \langle R(\mathtt{fix}\ f(x).ee), R(\mathtt{fix}\ f(x).ee)\rangle, 0\end{array}}\ \textbf{cp-fix}$$

$$\frac{\begin{array}{c}\langle T_1, ee_1\rangle \curvearrowright \mathtt{fix}\ f(x).ee, T_1', c_1' \qquad \langle T_2, ee_2\rangle \curvearrowright w_2', T_2', c_2' \\ \langle T_r, ee[w_2'/x, (\mathtt{fix}\ f(x).ee)/f]\rangle \curvearrowright w_r', T_r', c_r' \qquad v_r' = \mathtt{V}(T_r')\end{array}}{\begin{array}{c}\langle\langle \_, \mathtt{app}(T_1, T_2, T_r)\rangle, ee_1\ ee_2\rangle \curvearrowright \\ w_r', \langle v_r', \mathtt{app}(T_1', T_2', T_r')\rangle, c_1' + c_2' + c_r'\end{array}}\ \textbf{cp-app}$$

$$\frac{\begin{array}{c}\langle T_1, ee_1\rangle \curvearrowright \mathtt{new}(\mathrm{fix}\ f(x).e, \mathrm{fix}\ f(x).e'), T_1', c_1' \qquad \langle T_2, ee_2\rangle \curvearrowright w_2', T_2', c_2' \\ e'[R(w_2')/x, (\mathrm{fix}\ f(x).e')/f] \Downarrow^{f_r'} T_r' \qquad v_r' = \mathtt{V}(T_r')\end{array}}{\begin{array}{c}\langle\langle v_r, \mathtt{app}(T_1, T_2, T_r)\rangle, ee_1\ ee_2\rangle \curvearrowright \\ \mathtt{new}(v_r, v_r'), \langle v_r', \mathtt{app}(T_1', T_2', T_r')\rangle, c_1' + c_2' + f_r' + c_{app}\end{array}}\ \textbf{cp-app-new}$$

$$\frac{\langle T, ee\rangle \curvearrowright w', T', c' \qquad v' = \mathtt{V}(T')}{\langle\langle \_, \mathtt{inl}\ T\rangle, \mathtt{inl}\ ee\rangle \curvearrowright \mathtt{inl}\ w', \langle \mathrm{inl}\ v', \mathtt{inl}\ T'\rangle, c'}\ \textbf{cp-inl}$$

$$\frac{\langle T, ee\rangle \curvearrowright w, T', c' \qquad v' = \mathtt{V}(T')}{\langle\langle \_, \mathtt{inr}\ T\rangle, \mathtt{inr}\ ee\rangle \curvearrowright \mathtt{inr}\ w, \langle \mathrm{inr}\ v', \mathtt{inr}\ T'\rangle, c'}\ \textbf{cp-inr}$$

$$\frac{\langle T, ee\rangle \curvearrowright \mathtt{inl}\ w, T', c' \qquad \langle T_r, ee_1[w/x]\rangle \curvearrowright w_r', T_r', c_r' \qquad v_r' = \mathtt{V}(T_r')}{\begin{array}{c}\langle\langle \_, \mathtt{case}_{\mathtt{inl}}(T, T_r)\rangle, \mathtt{case}(ee, x.ee_1, y, ee_2)\rangle \curvearrowright \\ w_r', \langle v_r', \mathtt{case}_{\mathtt{inl}}(T', T_r')\rangle, c' + c_r'\end{array}}\ \textbf{cp-case-inl}$$

$$\frac{\langle T, ee\rangle \curvearrowright \mathtt{inr}\ w, T', c' \qquad \langle T_r, ee_2[w/y]\rangle \curvearrowright w_r', T_r', c_r' \qquad v_r' = \mathtt{V}(T_r')}{\begin{array}{c}\langle\langle \_, \mathtt{case}_{\mathtt{inr}}(T, T_r)\rangle, \mathtt{case}(ee, x.ee_1, y, ee_2)\rangle \\ \curvearrowright w_r', \langle v_r', \mathtt{case}_{\mathtt{inr}}(T', T_r')\rangle, c' + c_r'\end{array}}\ \textbf{cp-case-inr}$$

Figure 25: Change propagation rules, part 1

$\boxed{\langle T, ee\rangle \curvearrowright w', T', c'}$ Change propagation with cost-counting

In all the remaining rules except **cp-nochange**, we assume that the input ee satisfies $\neg\mathtt{stable}(ee)$.

$$\frac{\langle T, ee\rangle \curvearrowright \mathtt{new}(\_, \mathtt{inl}\ v'), T', c' \qquad R(ee_1)[v'/x] \Downarrow^{f'_r} T'_r \qquad v'_r = V(T'_r)}{\begin{array}{c}\langle\langle v_r, \mathtt{case}_{\mathtt{inl}}(T, T_r)\rangle, \mathtt{case}(ee, x.ee_1, y, ee_2)\rangle \curvearrowright \\ \mathtt{new}(v_r, v'_r), \langle v'_r, \mathtt{case}_{\mathtt{inl}}(T', T'_r)\rangle, c' + f'_r + c_{case}\end{array}}\ \textbf{cp-case-inl}_l$$

$$\frac{\langle T, ee\rangle \curvearrowright \mathtt{new}(\_, \mathtt{inr}\ v'), T', c' \qquad R(ee_2)[v'/x] \Downarrow^{f'_r} T'_r \qquad v'_r = V(T'_r)}{\begin{array}{c}\langle\langle v_r, \mathtt{case}_{\mathtt{inl}}(T, T_r)\rangle, \mathtt{case}(ee, x.ee_1, y, ee_2)\rangle \curvearrowright \\ \mathtt{new}(v_r, v'_r), \langle v'_r, \mathtt{case}_{\mathtt{inr}}(T', T'_r)\rangle, c' + f'_r + c_{case}\end{array}}\ \textbf{cp-case-inl}_r$$

$$\frac{\langle T, ee\rangle \curvearrowright \mathtt{new}(\_, \mathtt{inl}\ v'), T', c' \qquad R(ee_1)[v'/x] \Downarrow^{f'_r} T'_r \qquad v'_r = V(T'_r)}{\begin{array}{c}\langle\langle v_r, \mathtt{case}_{\mathtt{inr}}(T, T_r)\rangle, \mathtt{case}(ee, x.ee_1, y.ee_2)\rangle \curvearrowright \\ \mathtt{new}(v_r, v'_r), \langle v'_r, \mathtt{case}_{\mathtt{inl}}(T', T'_r)\rangle, c' + f'_r + c_{case}\end{array}}\ \textbf{cp-case-inr}_l$$

$$\frac{\langle T, ee\rangle \curvearrowright \mathtt{new}(\_, \mathtt{inr}\ v'), T', c' \qquad R(ee_2)[v'/y] \Downarrow^{f'_r} T'_r \qquad v'_r = V(T'_r)}{\begin{array}{c}\langle\langle v_r, \mathtt{case}_{\mathtt{inr}}(T, T_r)\rangle, \mathtt{case}(ee, x.ee_1, y.ee_2)\rangle \curvearrowright \\ \mathtt{new}(v_r, v'_r), \langle v'_r, \mathtt{case}_{\mathtt{inr}}(T', T'_r)\rangle, c' + f'_r + c_{case}\end{array}}\ \textbf{cp-case-inr}_r$$

$$\frac{\langle T_1, ee_1\rangle \curvearrowright w'_1, T'_1, c'_1 \qquad \langle T_2, ee_2\rangle \curvearrowright w'_2, T'_2, c'_2 \qquad v'_i = V(T'_i)}{\begin{array}{c}\langle\langle\_, \mathtt{cons}(T_1, T_2)\rangle, \mathtt{cons}(ee_1, ee_2)\rangle \curvearrowright \\ \mathtt{cons}(w'_1, w'_2), \langle \mathtt{cons}(v'_1, v'_2), \mathtt{cons}(T'_1, T'_2)\rangle, c'_1 + c'_2\end{array}}\ \textbf{cp-cons}$$

$$\frac{\langle T, ee\rangle \curvearrowright \mathtt{nil}, T', c' \qquad \langle T_r, ee_1\rangle \curvearrowright w'_r, T'_r, c'_r \qquad v'_r = V(T'_r)}{\langle\langle\_, \mathtt{case}_{\mathtt{nil}}(T, T_r)\rangle, \begin{array}{l}\mathtt{case}_L\, ee\ \mathtt{of}\ \mathtt{nil} \to ee_1 \\ \mid\ h :: tl \to ee_2\end{array}\rangle \curvearrowright w'_r, \langle v'_r, \mathtt{case}_{\mathtt{nil}}(T', T'_r)\rangle, c' + c'_r}\ \textbf{cp-caseL-nil}$$

$$\frac{\begin{array}{c}\langle T, ee\rangle \curvearrowright \mathtt{cons}(w_h, w_{tl}), T', c' \qquad \langle T_r, ee_2[w_h/h, w_{tl}/tl]\rangle \curvearrowright \\ w'_r, T'_r, c'_r \qquad v'_r = V(T'_r)\end{array}}{\langle\langle\_, \mathtt{case}_{\mathtt{cons}}(T, T_r)\rangle, \begin{array}{l}\mathtt{case}_L\, ee\ \mathtt{of}\ \mathtt{nil} \to ee_1 \\ \mid\ h :: tl \to ee_2\end{array}\rangle \curvearrowright w'_r, \langle v'_r, \mathtt{case}_{\mathtt{cons}}(T', T'_r)\rangle, c' + c'_r}\ \textbf{cp-caseL-cons}$$

$$\frac{\langle T, ee\rangle \curvearrowright \mathtt{new}(\_, \mathtt{nil}\ ), T', c' \qquad R(ee_1) \Downarrow^{f'_r} T'_r \qquad v'_r = V(T'_r)}{\langle\langle v_r, \mathtt{case}_{\mathtt{nil}}(T, \_)\rangle, \begin{array}{l}\mathtt{case}_L\, ee\ \mathtt{of}\ \mathtt{nil} \to ee_1 \\ \mid\ h :: tl \to ee_2\end{array}\rangle \curvearrowright \mathtt{new}(v_r, v'_r), \langle v'_r, \mathtt{case}_{\mathtt{nil}}(T', T'_r)\rangle, c' + f'_r}\ \textbf{cp-caseL-nil}_{nil}$$

$$\frac{\begin{array}{c}\langle T, ee\rangle \curvearrowright \mathtt{new}(\_, \mathtt{cons}(v'_h, v'_{tl})), T', c' \\ R(ee_2)[v'_h/h, v'_{tl}/tl] \Downarrow^{f'_r} T'_r \qquad v'_r = V(T'_r)\end{array}}{\langle\langle v_r, \mathtt{case}_{\mathtt{nil}}(T, \_)\rangle, \begin{array}{l}\mathtt{case}_L\, ee\ \mathtt{of}\ \mathtt{nil} \to ee_1 \\ \mid\ h :: tl \to ee_2\end{array}\rangle \curvearrowright \mathtt{new}(v_r, v'_r), \langle v'_r, \mathtt{case}_{\mathtt{cons}}(T', T'_r)\rangle, c' + f'_r}\ \textbf{cp-caseL-nil}_{cons}$$

Figure 26: Change propagation rules, part 2

$\boxed{\langle T, ee\rangle \curvearrowright w', T', c'}$ Change propagation with cost-counting

In all the remaining rules except **cp-nochange**, we assume that the input ee satisfies $\neg\mathtt{stable}(ee)$.

$$\frac{\langle T, ee\rangle \curvearrowright \mathtt{new}(\_, \mathtt{nil}\ ), T', c' \qquad R(ee_1) \Downarrow^{f'_r} T'_r \qquad v'_r = V(T'_r)}{\langle\langle v_r, \mathtt{case}_{\mathtt{cons}}(T, \_)\rangle, \begin{array}{l}\mathtt{case}_L\, ee\ \mathtt{of}\ \mathtt{nil} \to ee_1 \\ \mid\ h :: tl \to ee_2\end{array}\rangle \curvearrowright \mathtt{new}(v_r, v'_r), \langle v'_r, \mathtt{case}_{\mathtt{nil}}(T', T'_r)\rangle, c' + f'_r}\ \textbf{cp-caseL-cons}_{nil}$$

$$\frac{\begin{array}{c}\langle T, ee\rangle \curvearrowright \mathtt{new}(\_, \mathtt{cons}(v'_h, v'_{tl})), T', c' \\ R(ee_2)[v'_h/h, v'_{tl}/tl] \Downarrow^{f'_r} T'_r \qquad v'_r = V(T'_r)\end{array}}{\langle\langle v_r, \mathtt{case}_{\mathtt{cons}}(T, \_)\rangle, \begin{array}{l}\mathtt{case}_L\, ee\ \mathtt{of}\ \mathtt{nil} \to ee_1 \\ \mid\ h :: tl \to ee_2\end{array}\rangle \curvearrowright \mathtt{new}(v_r, v'_r), \langle v'_r, \mathtt{case}_{\mathtt{cons}}(T', T'_r)\rangle, c' + f'_r}\ \textbf{cp-caseL-cons}_{cons}$$

$$\frac{}{\langle\langle \_, \Lambda.e'\rangle, \Lambda.ee\rangle \curvearrowright \Lambda.ee, \langle \Lambda.\, R(ee), \Lambda.R(ee)\rangle, 0}\ \textbf{cp-Lam}$$

$$\frac{\langle T, ee\rangle \curvearrowright \Lambda.ee', T', c' \qquad \langle T_r, ee'\rangle \curvearrowright w'_r, T'_r, c'_r \qquad v'_r = V(T'_r)}{\langle\langle \_, \mathtt{iApp}(T, T_r)\rangle, ee[]\rangle \curvearrowright w'_r, \langle v'_r, \mathtt{iApp}(T', T'_r)\rangle, c' + c'_r}\ \textbf{cp-iApp}$$

$$\frac{\langle T, ee\rangle \curvearrowright \mathtt{new}(\_, \Lambda.e'), T', c' \qquad e' \Downarrow^{f'_r} T'_r \qquad v'_r = V(T'_r)}{\langle\langle v_r, \mathtt{iApp}(T, T_r)\rangle, ee[\,]\ \rangle \curvearrowright \mathtt{new}(v_r, v'_r), \langle v'_r, \mathtt{iApp}(T', T'_r)\rangle, c' + f'_r}\ \textbf{cp-iApp-new}$$

$$\frac{\langle T, ee\rangle \curvearrowright w', T', c' \qquad v'_r = V(T')}{\langle\langle \_, \mathtt{pack}\ T\rangle, \mathtt{pack}\ ee\rangle \curvearrowright \mathtt{pack}\ w', \langle \mathtt{pack}\ v'_r, \mathtt{pack}\ T'\rangle, c'}\ \textbf{cp-pack}$$

$$\frac{\langle T, ee_1\rangle \curvearrowright \mathtt{pack}\ w', T'_1, c'_1 \qquad \langle T_r, ee_2[w'/x]\rangle \curvearrowright w'_r, T'_r, c'_r \qquad v'_r = V(T'_r)}{\langle\langle \_, \mathtt{unpack}(T, x, T_r)\rangle, \mathtt{unpack}\ ee_1\ \mathtt{as}\ x\ \mathtt{in}\ ee_2\rangle \curvearrowright w'_r, \langle v'_r, \mathtt{unpack}(T', x, T'_r)\rangle, c'_1 + c'_r}\ \textbf{cp-unpack}$$

$$\frac{\begin{array}{c}\langle T, ee_1\rangle \curvearrowright \mathtt{new}(\_, \mathtt{pack}\ v), T'_1, c'_1 \\ R(ee_2)[v/x] \Downarrow^{f'_r} T'_r \qquad v'_r = V(T'_r)\end{array}}{\langle\langle v_r, \mathtt{unpack}(T, x, T_r)\rangle, \mathtt{unpack}\ ee_1\ \mathtt{as}\ x\ \mathtt{in}\ ee_2\rangle \curvearrowright \mathtt{new}(v_r, v'_r), \langle v'_r, \mathtt{unpack}(T', x, T'_r)\rangle, c'_1 + f'_r}\ \textbf{cp-unpack-new}$$

$$\frac{\langle T, ee\rangle \curvearrowright w', T', c' \qquad v' = V(T') \qquad (f'_r, v'_r) = \zeta(v')}{\langle\langle v_r, \mathtt{prim}_{\mathtt{app}}(T, \zeta)\rangle, \zeta\ ee\rangle \curvearrowright \mathtt{merge}(v_r, v'_r), \langle v'_r, \mathtt{prim}_{\mathtt{app}}(T', \zeta)\rangle, c' + f'_r}\ \textbf{cp-prim}$$

$$\frac{\langle T_1, ee_1\rangle \curvearrowright w'_1, T'_1, c'_1 \qquad \langle T_r, ee_2[w'_1/x]\rangle \curvearrowright w'_r, T'_r, c'_r}{\langle \mathtt{let}(x, T_1, T_r), \mathtt{let}\ x\ =\ ee_1\ \mathtt{in}\ ee_2\rangle \curvearrowright w'_r, \mathtt{let}(x, T'_1, T'_r), c'_1 + c'_r}\ \textbf{cp-let}$$

$$\frac{\langle T, ee\rangle \curvearrowright w', T', c'}{\langle \mathtt{celim}\ T, \mathrm{celim}_{\supset}\ ee\rangle \curvearrowright w', \mathtt{celim}\ T', c'}\ \textbf{cp-celim}$$

$$\frac{\langle T_1, ee_1\rangle \curvearrowright w'_1, T'_1, c'_1 \qquad \langle T_r, ee_2[w'_1/x]\rangle \curvearrowright w'_r, T'_r, c'_r}{\langle \mathtt{clet}_{\mathtt{as}}(x, T_1, T_r), \mathtt{clet}\ ee_1\ \mathtt{as}\ x\ \mathtt{in}\ ee_2\rangle \curvearrowright w'_r, \mathtt{clet}_{\mathtt{as}}(x, T'_1, T'_r), c'_1 + c'_r}\ \textbf{cp-clet}$$

Figure 27: Change propagation rules, part 3

original trace is output immediately (technically, it must be cast into the bivalue $\ulcorner v \urcorner$) and the cost of change propagation is 0.

Functions are change propagated trivially with zero cost by just returning the same function bivalue and updating the trace with the modified function (rule **cp-fix**). To change propagate a function application $ee_1\ ee_2$, we first change propagate through the function ee_1. If the resulting function does not differ from the original one structurally, i.e., the resulting bivalue has the form $\texttt{fix}\ f(x).ee$, then we keep change propagating through the body (rule **cp-app**). However, if the resulting function is structurally different from the original one (bivalue $\texttt{new}(\_, \texttt{fix}\ f(x).\, e')$), then we switch to from-scratch execution for the body of the modified function (rule **cp-app-new**). This pattern of switching to from-scratch evaluation repeats in all rules that apply closures.

To change propagate $\texttt{case}(ee, x.ee_1, y, ee_2)$, we first change propagate through the scrutinee ee. If the initial and incremental runs both took the same branch, i. e., the bivalue resulting from ee is either $\texttt{inl}\ w$ or $\texttt{inr}\ w$, we keep change propagating through that branch (rules **cp-case-inl** and **cp-case-inr**). Otherwise, e. g. ee's result has changed from inl _ to inr _ (detected by a bivalue of the form $\texttt{new}(\_, \texttt{inr}\ v')$), then we execute the right branch from-scratch, as in rule **cp-case-inl**$_r$. In addition, we incur an extra cost, c_{case}, for switching to from-scratch mode.

IMPLEMENTATION The relation $\curvearrowright$ formalizes change propagation and its cost *abstractly*. An obvious question is whether change propagation can be *implemented* with the asymptotic costs stipulated by the $\curvearrowright$ relation. The answer is affirmative. Prior work on libraries and compilers for self-adjusting computation already shows how to implement change propagation with these costs using imperative traces, leaf-to-root traversals and in-place update of values [5, 32]. Since values are updated in-place, no cost is incurred for structural operations like pairing, projection, consing, etc; cost is incurred only for re-evaluating primitive functions on paths starting in updated leaves, exactly as in the judgment $\curvearrowright$. To double-check, we implemented most of our examples on an existing library, AFL [5], and observed (empirically) exactly the asymptotic costs stipulated by $\curvearrowright$. However, these observations are experimental. A more thorough study is conducted by Zoe Paraskevopoulou where as part of her masters thesis, she showed how the abstract change propagation semantics can be realized by translation to an ML-like language with runtime support for incremental evaluation with an actual low-level cost semantics [35, 85].

8.3 DUCOSTIT'S TYPING JUDGMENTS

Like RelCost, DuCostIt relies on two typing judgments: a unary and a relational one. The relational judgment

$$\Delta; \Phi; \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t$$

states that that t is an upper bound on the cost of change propagating through e. The unary judgment

$$\Delta; \Phi; \Omega \vdash_{\mathbb{FS}} e : A \mid t$$

states that t is an upper bound on the cost of evaluating e from-scratch. As in RelCost, these typing judgments use two kinds of type environments: Ω for unary typing and Γ relational typing. Beside these, both typing judgments have two other environments: Δ for index variables and Φ for assumed constraints. The judgments also include a fourth context that specifies the types of primitive functions ζ, but this context does not change in the rules, so we exclude it from the presentation.

Note that, the cost of change propagation is no more than the cost of evaluating from-scratch, so the second judgment $\Delta; \Phi; \Omega \vdash_{\mathbb{FS}} e : A \mid t$ implies the relational judgment $\Delta; \Phi; \mathrm{U}\,\Omega \vdash_{\mathbb{CP}} e : \mathrm{U}\,A \mid t$ *semantically* at the weakest environment and type, hence, it is perfectly sound to change propagate through expressions typed with either judgment. We rely on this property heavily in our semantics of types.

LOWER BOUNDS ON THE DYNAMIC STABILITY In Chapter 4, we discussed why tracking lower bounds on RelCost's relational judgment was redundant. In essence, since the two programs in RelCost have the same evaluation semantics, relative costs are symmetric (hence the inequality $k \leqslant \mathrm{cost}(e_1) - \mathrm{cost}(e_2) \leqslant t$ can be flipped). However, we cannot make the same claim in DuCostIt because unlike relative costs, dynamic stability is not symmetric: the two executions of the program do not have the same semantics. Hence, if one is interested in lower bounds on the dynamic stability, then DuCostIt's relational judgment as well as its unary judgment must be extended to track lower bounds. We do not proceed with this path since in the case of incremental computations, programmers are often interested in worst-case bounds on the dynamic stability.

DUCOSTIT'S TYPING PRINCIPLES AND DESIGN CHOICES Before we explain the details of DuCostIt's type system, we review the general design principles behind the unary and relational typing rules.

- As in RelCost, the total cost of an expression is obtained by summing the costs of its subexpressions. Moreover, for the unary typing, elimination constructs described in Section 8.1 incur additional costs.
- As in RelCost, DuCostIt only allows eliminating truly related expressions that are not of type $\mathsf{U}\,A$. For instance, case-elimination on $\mathsf{U}\,(A_1 + A_2)$ cannot be typed *relationally* in $\mathbb{CP}$-mode. All such cases are handled *uniformly*: If the eliminated expressions are unrelated, i.e., of type $\mathsf{U}\,A$, the verification can be done only by switching from $\mathbb{CP}$-mode to non-relational $\mathbb{FS}$-mode for the whole expression.
- RelCost has several *asynchronous* typing rules that combine relational and unary typing rules since the two programs may structurally differ. In contrast, DuCostIt only has a single typing rule that allows switching from relational $\mathbb{CP}$-mode typing to unary $\mathbb{FS}$-mode typing since we have the same program in the initial and the incremental run.

The typing rules for the unary and relational typing judgments are shown in Figures 28 and 29 and Figures 30 and 31, respectively. Below, we explain selected rules for the two judgments separately. Since most of DuCostIt's unary and relational typing mimics RelCost's, we only explain rules that differ from RelCost's.

8.3.1 *Unary Typing*

As mentioned before, we only track upper bounds on DuCostIt's unary typing, so its typing can be thought of as a simplification of RelCost's unary typing. We briefly discuss a few typing rules.

Like in RelCost, values have no effect—they are assumed to evaluate with zero cost. So, variables (rule **fs-var**), as well as all introduction forms including functions and index abstractions incur zero cost (rules **fs-fix** and **fs-iLam**). For functions, the from-scratch execution cost of the body, denoted t, is internalized into the type $A_1 \xrightarrow{\mathbb{FS}(t)} A_2$ (rule **fs-fix**). In the rule **fs-app**, this internalized latent cost t is added to the total cost of the application along with an additional symbolic cost c_{app} for the function application.

Since DuCostIt is geared towards estimating change propagation costs, the symbolic costs for elimination forms are assumed to be non-zero. As in RelCost, these costs can be adjusted if necessary.

8.3.2 *Relational Typing*

Relational typing establishes the dynamic stability of an expression and assigns the expression a relational type under the given relational environment that captures how the inputs of the program may change.

In a call-by-value language like Cost^{ML}, variables are substituted by values and the cost of updating (change propagating) the substitution for a variable is paid by the context that provides the substitution. So a variable incurs zero cost during change propagation (rule **cp-var**).

Rules **cp-fix** and **cp-app** type recursive functions and function applications, respectively. In rule **cp-fix**, the body of the function is typed in the same mode as the function itself. The annotation on the function's type is $\mathbb{CP}$ (and the latent cost t bounds the change-propagation cost) because the function is constructed within the program, so it will not change syntactically across runs.[32] In rule **cp-app**, the latent cost of the function is added to the total change-propagation cost of the application.

[32] *This principle also applies to all value introduction forms (for instance, the rules **cp-inl** and **cp-iLam**).*

Like RelCost, DuCostIt has similar rules for introduction and elimination forms for lists: The $\square\cdot$ type interacts in the same way with the number of changes α depending on whether the head might change or not (rules **cp-cons1**, **cp-cons2** and **cp-caseL**).

The rule **cp-nochange** captures the intuition that if no dependencies (substitutions for free variables) of an expression can change, then the expression's result cannot change and there is no need to change propagate through its trace (i.e., its change propagation cost is zero). The second premise of **cp-nochange** checks that the types of all variables can be subtyped to the form $\square\cdot$, which ensures that the dependencies of the expression cannot change. The rule's conclusion allows the type to be annotated $\square\cdot$ and, additionally, the cost to be 0. Notice that this rule only makes sense in relational $\mathbb{CP}$-mode typing, hence there is no counterpart in unary $\mathbb{FS}$-mode typing.

The rule **cp-switch** allows an expression of type A to be related at the weakest relation with type $\mathsf{U}\,A$. When read from bottom-to-top, it switches from *relational* reasoning to *unary* reasoning that types the expression independently in an erased environment $|\Gamma|$. Then, the change propagation cost is upper bounded by the expression's from-scratch execution cost. Like in RelCost, the type erasure operation $|.|$ is a function

$\boxed{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A \mid t}$ Execution cost of e is upper bounded by t, and e has the unary type A.

$$\frac{}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} n : \mathsf{int} \mid 0}\ \textbf{fs-const} \qquad \frac{\Omega(x) = A}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} x : A \mid 0}\ \textbf{fs-var}$$

$$\frac{}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} () : \mathsf{unit} \mid 0}\ \textbf{fs-unit}$$

$$\frac{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A_1 \mid t \qquad \Delta; \Phi \vdash^{A} A_2\ \mathtt{wf}}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} \mathsf{inl}\ e : A_1 + A_2 \mid t}\ \textbf{fs-inl}$$

$$\frac{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A_2 \mid t \qquad \Delta; \Phi \vdash^{A} A_1\ \mathtt{wf}}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} \mathsf{inr}\ e : A_1 + A_2 \mid t}\ \textbf{fs-inr}$$

$$\frac{\begin{array}{c}\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A_1 + A_2 \mid t \\ \Delta; \Phi; x : A_1, \Omega \vdash_{\mathbb{FS}} e_1 : A \mid t' \qquad \Delta; \Phi; y : A_2, \Omega \vdash_{\mathbb{FS}} e_2 : A \mid t'\end{array}}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}}\ \mathsf{case}\ (e, x.e_1, y.e_2) : A \mid t + t' + c_{case}}\ \textbf{fs-case}$$

$$\frac{\begin{array}{c}\Delta; \Phi \vdash^{A} A_1 \xrightarrow{\mathbb{FS}(t)} A_2\ \mathtt{wf} \\ \Delta; \Phi; x : A_1, f : A_1 \xrightarrow{\mathbb{FS}(t)} A_2, \Omega \vdash_{\mathbb{FS}} e : A_2 \mid t\end{array}}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} \mathsf{fix}\ f(x).e : A_1 \xrightarrow{\mathbb{FS}(t)} A_2 \mid 0}\ \textbf{fs-fix}$$

$$\frac{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e_1 : A_1 \xrightarrow{\mathbb{FS}(t)} A_2 \mid t_1 \qquad \Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e_2 : A_1 \mid t_2}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e_1\ e_2 : A_2 \mid t_1 + t_2 + t + c_{app}}\ \textbf{fs-app}$$

$$\frac{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e_1 : A_1 \mid t_1 \qquad \Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e_2 : A_2 \mid t_2}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} \langle e_1, e_2 \rangle : A_1 \times A_2 \mid t_1 + t_2}\ \textbf{fs-prod}$$

$$\frac{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A_1 \times A_2 \mid t}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} \pi_1(e) : A_1 \mid t + c_{proj}}\ \textbf{fs-proj1}$$

$$\frac{\Delta; \Phi \vdash^{A} A\ \mathtt{wf}}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} \mathsf{nil}\ : \mathsf{list}[0]\ A \mid 0}\ \textbf{fs-nil}$$

$$\frac{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e_1 : A \mid t_1 \qquad \Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e_2 : \mathsf{list}[n]\ A \mid t_2}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} \mathsf{cons}(e_1, e_2) : \mathsf{list}[n+1]\ A \mid t_1 + t_2}\ \textbf{fs-cons}$$

$$\frac{\begin{array}{c}\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : \mathsf{list}[n]\ A \mid t \qquad \Delta; \Phi \wedge n = 0; \Omega \vdash_{\mathbb{FS}} e_1 : A' \mid t' \\ i, \Delta; \Phi \wedge n = i + 1; h : A, tl : \mathsf{list}[i]\ A, \Omega \vdash_{\mathbb{FS}} e_2 : A' \mid t'\end{array}}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}}\ \mathsf{case}\ e\ \mathsf{of}\ \mathsf{nil}\ \to e_1 \mid h :: tl \to e_2 : A' \mid t + t' + c_{caseL}}\ \textbf{fs-caseL}$$

$$\frac{i :: S, \Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A \mid t \qquad i \notin \mathrm{FIV}(\Phi; \Omega)}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} \Lambda.e : \forall i \overset{\mathbb{FS}(t)}{::} S.\, A \mid 0}\ \textbf{fs-iLam}$$

$$\frac{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : \forall i \overset{\mathbb{FS}(t')}{::} S.\, A \mid t \qquad \Delta \vdash I : S}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e[\,] : A\{I/i\} \mid t + t'[I/i]}\ \textbf{fs-iApp}$$

Figure 28: DuCostIt unary typing rules (Part 1)

$\boxed{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A \mid t}$ Execution cost of e is upper bounded by t, and e has the unary type A.

$$\frac{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A\{I/i\} \mid t \qquad \Delta \vdash I :: S}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} \text{pack } e : \exists i{::}S.\, A \mid t}\ \textbf{fs-pack}$$

$$\frac{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e_1 : \exists i{::}S.\, A_1 \mid t_1 \qquad i :: S, \Delta; \Phi; x : A_1, \Omega \vdash_{\mathbb{FS}} e_2 : A_2 \mid t_2 \qquad i \notin FV(\Phi; \Gamma, A_2, t_2)}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} \text{unpack } e_1 \text{ as } x \text{ in } e_2 : A_2 \mid t_1 + t_2}\ \textbf{fs-unpack}$$

$$\frac{\Upsilon(\zeta) = A_1 \xrightarrow{\mathbb{FS}(t)} A_2 \qquad \Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A_1 \mid t'}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} \zeta\ e : A_2 \mid t + t' + c_{primapp}}\ \textbf{fs-primapp}$$

$$\frac{\Delta; \Phi \models C \qquad \Delta; \Phi \wedge C; \Omega \vdash_{\mathbb{FS}} e : A \mid t}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : C \,\&\, A \mid t}\ \textbf{fs-c-andI}$$

$$\frac{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e_1 : C \,\&\, A_1 \mid t_1 \qquad \Delta; \Phi \wedge C; x : A_1, \Omega \vdash_{\mathbb{FS}} e_2 : A_2 \mid t_2}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} \text{clet } e_1 \text{ as } x \text{ in } e_2 : A_2 \mid t_1 + t_2}\ \textbf{fs-c-andE}$$

$$\frac{\Delta; \Phi \wedge C; \Omega \vdash_{\mathbb{FS}} e : A \mid t}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : C \supset A \mid t}\ \textbf{fs-c-impI}$$

$$\frac{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : C \supset A \mid t \qquad \Delta; \Phi \models C}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} \text{celim}_{\supset}\ e : A \mid t}\ \textbf{fs-c-implE}$$

$$\frac{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A \mid t \qquad \Delta; \Phi \models A \sqsubseteq A' \qquad \Delta; \Phi \models t \leqslant t'}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A' \mid t'}\ \sqsubseteq_{\text{exec}}$$

Figure 29: DuCostIt unary typing rules (Part 2)

$\boxed{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t}$ Dynamic stability of e is upper bounded by t and e has relational type τ.

$$\frac{}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} n : \mathrm{int}_r \mid 0}\ \textbf{cp-const} \qquad \frac{\Gamma(x) = \tau}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} x : \tau \mid 0}\ \textbf{cp-var}$$

$$\frac{}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} () : \mathrm{unit}_r \mid 0}\ \textbf{cp-unit}$$

$$\frac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau_1 \mid t \qquad \Delta; \Phi \vdash \tau_2\ \mathtt{wf}}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \mathrm{inl}\ e : \tau_1 + \tau_2 \mid t}\ \textbf{cp-inl}$$

$$\frac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau_2 \mid t \qquad \Delta; \Phi \vdash \tau_1\ \mathtt{wf}}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \mathrm{inr}\ e : \tau_1 + \tau_2 \mid t}\ \textbf{cp-inr}$$

$$\frac{\begin{array}{c}\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau_1 + \tau_2 \mid t \\ \Delta; \Phi; x : \tau_1, \Gamma \vdash_{\mathbb{CP}} e_1 : \tau \mid t' \qquad \Delta; \Phi; y : \tau_2, \Gamma \vdash_{\mathbb{CP}} e_2 : \tau \mid t'\end{array}}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}}\ \mathrm{case}\ (e, x.e_1, y.e_2) : \tau \mid t + t'}\ \textbf{cp-case}$$

$$\frac{\Delta; \Phi \vdash \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2\ \mathtt{wf} \qquad \Delta; \Phi; x : \tau_1, f : \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2, \Gamma \vdash_{\mathbb{CP}} e : \tau_2 \mid t}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \mathrm{fix}\ f(x).e : \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2 \mid 0}\ \textbf{cp-fix}$$

$$\frac{\begin{array}{c}\Delta; \Phi \vdash \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2\ \mathtt{wf} \\ \Delta; \Phi; x : \tau_1, f : \Box(\tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2), \Gamma \vdash_{\mathbb{CP}} e : \tau_2 \mid t \\ \forall x \in \mathrm{dom}(\Gamma).\ \ \Delta; \Phi \models \Gamma(x) \sqsubseteq \Box\Gamma(x)\end{array}}{\Delta; \Phi; \Gamma, \Gamma' \vdash_{\mathbb{CP}} \mathrm{fix}\ f(x).e : \Box(\tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2) \mid 0}\ \textbf{cp-fixNC}$$

$$\frac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_1 : \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2 \mid t_1 \qquad \Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_2 : \tau_1 \mid t_2}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_1\ e_2 : \tau_2 \mid t_1 + t_2 + t}\ \textbf{cp-app}$$

$$\frac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_1 : \tau_1 \mid t_1 \qquad \Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_2 : \tau_2 \mid t_2}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \langle e_1, e_2 \rangle : \tau_1 \times \tau_2 \mid t_1 + t_2}\ \textbf{cp-prod}$$

$$\frac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau_1 \times \tau_2 \mid t}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \pi_1(e) : \tau_1 \mid t}\ \textbf{cp-proj1}$$

$$\frac{\Delta; \Phi \vdash \tau\ \mathtt{wf}}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \mathrm{nil}\ : \mathrm{list}[0]^{\alpha}\,\tau \mid 0}\ \textbf{cp-nil}$$

$$\frac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_1 : \tau \mid t_1 \qquad \Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_2 : \mathrm{list}[n]^{\alpha}\,\tau \mid t_2}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \mathrm{cons}(e_1, e_2) : \mathrm{list}[n+1]^{\alpha+1}\,\tau \mid t_1 + t_2}\ \textbf{cp-cons1}$$

$$\frac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_1 : \Box\tau \mid t_1 \qquad \Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_2 : \mathrm{list}[n]^{\alpha}\,\tau \mid t_2}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \mathrm{cons}(e_1, e_2) : \mathrm{list}[n+1]^{\alpha}\,\tau \mid t_1 + t_2}\ \textbf{cp-cons2}$$

Figure 30: DuCostIt relational typing rules (Part 1)

$\boxed{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t}$ Dynamic stability of e is upper bounded by t and e has relational type τ.

$$\frac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \mathrm{list}[n]^{\alpha}\,\tau \mid t \qquad \Delta; \Phi \wedge n = 0; \Gamma \vdash_{\mathbb{CP}} e_1 : \tau' \mid t' \qquad i, \Delta; \Phi \wedge n = i+1; h : \square\tau, tl : \mathrm{list}[i]^{\alpha}\,\tau, \Gamma \vdash_{\mathbb{CP}} e_2 : \tau' \mid t' \qquad i, \beta, \Delta; \Phi \wedge n = i+1 \wedge \alpha = \beta + 1; h : \tau, tl : \mathrm{list}[i]^{\beta}\,\tau, \Gamma \vdash_{\mathbb{CP}} e_2 : \tau' \mid t'}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \mathrm{case}\ e\ \mathrm{of\ nil}\ \to e_1 \mid h :: tl \to e_2 : \tau' \mid t + t'}\ \textbf{cp-caseL}$$

$$\frac{i :: S, \Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t \qquad i \notin \mathrm{FIV}(\Phi; \Gamma)}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \Lambda.e : \forall i \overset{\mathbb{CP}(t)}{::} S.\,\tau \mid 0}\ \textbf{cp-iLam}$$

$$\frac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \forall i \overset{\mathbb{CP}(t')}{::} S.\,\tau \mid t \qquad \Delta \vdash I : S}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e[\,] : \tau\{I/i\} \mid t + t'[I/i]}\ \textbf{cp-iApp}$$

$$\frac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau\{I/i\} \mid t \qquad \Delta \vdash I :: S}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \mathrm{pack}\ e : \exists i{::}S.\,\tau \mid t}\ \textbf{cp-pack}$$

$$\frac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_1 : \exists i{::}S.\,\tau_1 \mid t_1 \qquad i :: S, \Delta; \Phi; x : \tau_1, \Gamma \vdash_{\mathbb{CP}} e_2 : \tau_2 \mid t_2 \qquad i \notin \mathrm{FV}(\Phi; \Gamma, \tau_2, t_2)}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \mathrm{unpack}\ e_1\ \mathrm{as}\ x\ \mathrm{in}\ e_2 : \tau_2 \mid t_1 + t_2}\ \textbf{cp-unpack}$$

$$\frac{\Upsilon(\zeta) = \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2 \qquad \Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau_1 \mid t'}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \zeta\ e : \tau_2 \mid t + t'}\ \textbf{cp-primapp}$$

$$\frac{\Delta; \Phi \models C \qquad \Delta; \Phi \wedge C; \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : C\ \&\ \tau \mid t}\ \textbf{cp-c-andI}$$

$$\frac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_1 : e_1' \mid t_1\ C\ \&\ \tau_1 \qquad \Delta; \Phi \wedge C; x : \tau_1, \Gamma \vdash_{\mathbb{CP}} e_2 : e_2' \mid t_2\ \tau_2}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \mathrm{clet}\ e_1\ \mathrm{as}\ x\ \mathrm{in}\ e_2 : \tau_2 \mid t_1 + t_2}\ \textbf{cp-c-andE}$$

$$\frac{\Delta; \Phi \wedge C; \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : C \supset \tau \mid t}\ \textbf{cp-c-impI}$$

$$\frac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : C \supset \tau \mid t \qquad \Delta; \Phi \models C}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \mathrm{celim}_{\supset}\ e : \tau \mid t}\ \textbf{cp-c-implE}$$

$$\frac{\Delta; \Phi \wedge C; \Gamma \vdash_{\mathbb{CP}} e_1 : \tau \mid t \qquad \Delta; \Phi \wedge \neg C; \Gamma \vdash_{\mathbb{CP}} e_1 : \tau \mid t \qquad \Delta \vdash C\ \mathtt{wf}}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_1 : \tau \mid t}\ \textbf{cp-split}$$

$$\frac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t \qquad \Delta; \Phi \models \tau \sqsubseteq \tau' \qquad \Delta; \Phi \models t \leqslant t'}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau' \mid t'}\ \textbf{cp-}\sqsubseteq$$

$$\frac{\Delta; \Phi; \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t \qquad \forall x \in \mathrm{dom}(\Gamma).\ \Delta; \Phi \models \Gamma(x) \sqsubseteq \square\Gamma(x)}{\Delta; \Phi; \Gamma, \Gamma' \vdash_{\mathbb{CP}} e : \square\tau \mid 0}\ \textbf{cp-nochange}$$

$$\frac{\Delta; \Phi; |\Gamma| \vdash_{\mathbb{FS}} e : A \mid t}{\Delta; \Phi; \Gamma \vdash_{\mathbb{CP}} e : U\,A \mid t}\ \textbf{cp-switch}$$

Figure 31: DuCostIt relational typing rules (Part 2)

$$
\begin{array}{lcl}
|\cdot| & : & \text{Relational type} \to \text{Unary type} \\
\\
|\text{int}_r| & = & \text{int} \\
|\text{unit}_r| & = & \text{unit} \\
|\tau_1 \times \tau_2| & = & |\tau_1| \times |\tau_2| \\
|\tau_1 + \tau_2| & = & |\tau_1| + |\tau_2| \\
|\text{list}[n]^{\alpha}\,\tau| & = & \text{list}[n]\,|\tau| \\
|\tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2| & = & |\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\tau_2| \\
|\forall i \overset{\mathbb{CP}(t)}{::} S.\,\tau| & = & \forall i \overset{\mathbb{FS}(\infty)}{::} S.\,|\tau| \\
|\exists i{::}S.\,\tau| & = & \exists i{::}S.\,|\tau| \\
|C \,\&\, \tau| & = & C \,\&\, |\tau| \\
|C \supset \tau| & = & C \supset |\tau| \\
|U\,A| & = & A \\
|\Box\,\tau| & = & |\tau|
\end{array}
$$

Figure 32: DuCostIt refinement removal operation

from relational types to unary types that forgets the relational refinements. Its definition is shown in Figure 32.

Additionally, similar to RelCost, DuCostIt also has generic rules like **cp-split**.

8.4 SUBTYPING

Just like in RelCost, subtyping plays a crucial role in DuCostIt. Due to the use of bivalues in DuCostIt's semantic and operational model, DuCostIt's subtyping is slightly different from RelCost's. Still, subtyping follows the same core ideas: e.g. list types interact similarly with the $\Box$ annotation and subtyping is constraint dependent. We show DuCostIt's unary and relational subtyping rules in Figure 33 and Figures 34 and 35, respectively.

The main difference in the subtyping rules of DuCostIt and RelCost is the way unary types are lifted to relational types at the weakest relation $U\,\cdot$. This difference stems from the fact that DuCostIt cares about where and how a value is changed whereas RelCost doesn't. For instance, in RelCost, since relational types are interpreted as pairs of values, the values $\langle 2, 3\rangle$ and $\langle 20, 30\rangle$ can be given two different types: $U\,(\text{int} \times \text{int})$ and $U\,\text{int} \times U\,\text{int}$. Hence RelCost allows subtyping $U\,(A_1 \times A_2)$ to $U\,A_1 \times$

$\mathsf{U}\,A_2$. However, this would be unsound in DuCostIt since DuCostIt cares about how a value is changed, and according to which change propagation might either switch to from-scratch execution or keep change propagating. Therefore, relational types are interpreted as bivalues which pose some restrictions on truly relational types, i.e. types that are not of form $\mathsf{U}\,\cdot$. In particular, in DuCostIt, the type $\mathsf{U}\,A_1 \times \mathsf{U}\,A_2$ can admit no direct changes to itself but only to its sub-parts whereas the type $\mathsf{U}\,(A_1 \times A_2)$ may admit changes to itself. In other words, $\mathsf{new}(\langle 2,3\rangle, \langle 20,30\rangle)$ cannot be given a type $\mathsf{U}\,\mathsf{int} \times \mathsf{U}\,\mathsf{int}$. The only way a type of the form $\mathsf{U}\,(A_1 \times A_2)$ can be lifted to $\mathsf{U}\,A_1 \times \mathsf{U}\,A_2$ is if it is not a $\mathsf{new}(v, v')$ itself. Therefore, in DuCostIt, we can only subtype $\Box\,\mathsf{U}\,(A_1 \times A_2)$ to $\Box\,\mathsf{U}\,A_1 \times \Box\,\mathsf{U}\,A_2$ (rule $\times\Box\mathsf{U}$ in Figure 34). Similar subtyping rules apply for types $A_1 \xrightarrow{\mathbb{FS}(t)} A_2$ and $\forall i \overset{\mathbb{FS}(t)}{::} S.\,A$.

Like in RelCost, the type $\Box\,\tau$ follows the standard co-monadic rules: $\Box\,\tau \sqsubseteq \tau$, $\Box\,(\tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2) \sqsubseteq \Box\,\tau_1 \xrightarrow{\mathbb{CP}(0)} \Box\,\tau_2$ and $\Box\,(\tau_1 \times \tau_2) \equiv \Box\,\tau_1 \times \Box\,\tau_2$.

$\boxed{\Delta; \Phi \models^{A} A_1 \sqsubseteq A_2}$ Type A_1 is a subtype of type A_2

$$\frac{\Delta; \Phi \models^{A} A_1' \sqsubseteq A_1 \qquad \Delta; \Phi \models^{A} A_2 \sqsubseteq A_2' \qquad \Delta; \Phi \models t \leqslant t'}{\Delta; \Phi \models^{A} A_1 \xrightarrow{\mathbb{FS}(t)} A_2 \sqsubseteq A_1' \xrightarrow{\mathbb{FS}(t')} A_2'} \to \textbf{exec}$$

$$\frac{i :: S, \Delta; \Phi \models^{A} A \sqsubseteq A' \qquad i :: S, \Delta; \Phi \models t \leqslant t' \qquad i \notin FV(\Phi)}{\Delta; \Phi \models^{A} \forall i \overset{\mathbb{FS}(t)}{::} S. A \sqsubseteq \forall i \overset{\mathbb{FS}(t')}{::} S. A} \textbf{u-}\forall_{\textbf{exec}}$$

$$\frac{\Delta; \Phi \models^{A} A_1 \sqsubseteq A_1' \qquad \Delta; \Phi \models^{A} A_2 \sqsubseteq A_2'}{\Delta; \Phi \models^{A} A_1 \times A_2 \sqsubseteq A_1' \times A_2'} \textbf{u-}\times$$

$$\frac{\Delta; \Phi \models^{A} A_1 \sqsubseteq A_1' \qquad \Delta; \Phi \models^{A} A_2 \sqsubseteq A_2'}{\Delta; \Phi \models^{A} A_1 + A_2 \sqsubseteq A_1' + A_2'} \textbf{u-}+$$

$$\frac{\Delta; \Phi \models n \doteq n' \qquad \Delta; \Phi \models^{A} A \sqsubseteq A'}{\Delta; \Phi \models^{A} \mathrm{list}[n]\, A \sqsubseteq \mathrm{list}[n']\, A'} \textbf{u-l}$$

$$\frac{i :: S, \Delta; \Phi \models^{A} A \sqsubseteq A' \qquad i \notin FV(\Phi)}{\Delta; \Phi \models^{A} \exists i{::}S. A \sqsubseteq \exists i{::}S. A'} \textbf{u-}\exists$$

$$\frac{\Delta; \Phi \wedge C \models C' \qquad \Delta; \Phi \models^{A} A \sqsubseteq A'}{\Delta; \Phi \models^{A} C \,\&\, A \sqsubseteq C' \,\&\, A'} \textbf{u-c-and}$$

$$\frac{\Delta; \Phi \wedge C' \models C \qquad \Delta; \Phi \models^{A} A \sqsubseteq A'}{\Delta; \Phi \models^{A} C \supset A \sqsubseteq C' \supset A'} \textbf{u-c-impl} \qquad \frac{}{\Delta; \Phi \models^{A} A \sqsubseteq A} \textbf{u-refl}$$

$$\frac{\Delta; \Phi \models^{A} A_1 \sqsubseteq A_2 \qquad \Delta; \Phi \models^{A} A_2 \sqsubseteq A_3}{\Delta; \Phi \models^{A} A_1 \sqsubseteq A_3} \textbf{u-tran}$$

Figure 33: DuCostIt's unary subtyping rules

$\boxed{\Delta; \Phi \models \tau_1 \sqsubseteq \tau_2}$ Relational type τ_1 is a subtype of relational type τ_2

$\boxed{\Delta; \Phi \models^{A} A_1 \sqsubseteq A_2}$ Type A_1 is a subtype of type A_2

$$\frac{}{\Delta; \Phi \models \mathrm{int}_r \sqsubseteq \Box \mathrm{int}_r}\ \textbf{int-}\Box \qquad \frac{}{\Delta; \Phi \models \Box \mathsf{U}\, \mathrm{int} \sqsubseteq \mathrm{int}_r}\ \Box\textbf{U-int}$$

$$\frac{}{\Delta; \Phi \models \mathrm{unit}_r \sqsubseteq \Box \mathrm{unit}_r}\ \textbf{unit}$$

$$\frac{\Delta; \Phi \models \tau_1' \sqsubseteq \tau_1 \qquad \Delta; \Phi \models \tau_2 \sqsubseteq \tau_2' \qquad \Delta; \Phi \models t \leqslant t'}{\Delta; \Phi \models \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2 \sqsubseteq \tau_1' \xrightarrow{\mathbb{CP}(t')} \tau_2'}\ \to_{\mathbf{cp}}$$

$$\frac{}{\Delta; \Phi \models \Box (\tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2) \sqsubseteq \Box \tau_1 \xrightarrow{\mathbb{CP}(0)} \Box \tau_2}\ \to \Box_{\mathbf{cp}}$$

$$\frac{}{\Delta; \Phi \models \Box (\mathsf{U}\, (A_1 \xrightarrow{\mathbb{FS}(t)} A_2)) \sqsubseteq \Box \mathsf{U}\, A_1 \xrightarrow{\mathbb{CP}(0)} \Box \mathsf{U}\, A_2}\ \to \Box \mathsf{U}_{\mathbf{cp}}$$

$$\frac{i :: S, \Delta; \Phi \models \tau \sqsubseteq \tau' \qquad i :: S, \Delta; \Phi \models t \leqslant t' \qquad i \notin \mathrm{FV}(\Phi)}{\Delta; \Phi \models \forall i \overset{\mathbb{CP}(t)}{::} S.\, \tau \sqsubseteq \forall i \overset{\mathbb{CP}(t')}{::} S.\, \tau'}\ \forall_{\mathbf{cp}}$$

$$\frac{}{\Delta; \Phi \models \Box (\forall i \overset{\mathbb{CP}(t)}{::} S.\, \tau) \sqsubseteq \forall i \overset{\mathbb{CP}(0)}{::} S.\, \Box \tau}\ \forall\, \Box$$

$$\frac{}{\Delta; \Phi \models \Box (\mathsf{U}\, (\forall i \overset{\mathbb{FS}(t)}{::} S.\, A)) \sqsubseteq \forall i \overset{\mathbb{CP}(0)}{::} S.\, \Box \mathsf{U}\, A}\ \forall\, \Box\, \mathbf{U}$$

$$\frac{\Delta; \Phi \models \tau_1 \sqsubseteq \tau_1' \qquad \Delta; \Phi \models \tau_2 \sqsubseteq \tau_2'}{\Delta; \Phi \models \tau_1 \times \tau_2 \sqsubseteq \tau_1' \times \tau_2'}\ \times$$

$$\frac{}{\Delta; \Phi \models \Box \tau_1 \times \Box \tau_2 \equiv \Box (\tau_1 \times \tau_2)}\ \times\Box$$

$$\frac{}{\Delta; \Phi \models \Box (\mathsf{U}\, (A_1 \times A_2)) \sqsubseteq \Box \mathsf{U}\, A_1 \times \Box \mathsf{U}\, A_2}\ \times\Box\mathbf{U}$$

$$\frac{\Delta; \Phi \models \tau_1 \sqsubseteq \tau_1' \qquad \Delta; \Phi \models \tau_2 \sqsubseteq \tau_2'}{\Delta; \Phi \models \tau_1 + \tau_2 \sqsubseteq \tau_1' + \tau_2'}\ +$$

$$\frac{}{\Delta; \Phi \models \Box \tau_1 + \Box \tau_2 \sqsubseteq \Box (\tau_1 + \tau_2)}\ +\Box$$

$$\frac{\Delta; \Phi \models n \doteq n' \qquad \Delta; \Phi \models \alpha \leqslant \alpha' \qquad \Delta; \Phi \models \tau \sqsubseteq \tau'}{\Delta; \Phi \models \mathrm{list}[n]^{\alpha} \tau \sqsubseteq \mathrm{list}[n']^{\alpha'} \tau'}\ \mathbf{l_1}$$

$$\frac{\Delta; \Phi \models \alpha \doteq 0}{\Delta; \Phi \models \mathrm{list}[n]^{\alpha} \tau \sqsubseteq \mathrm{list}[n]^{\alpha} \Box \tau}\ \mathbf{l_2}$$

$$\frac{}{\Delta; \Phi \models \mathrm{list}[n]^{\alpha} \Box \tau \sqsubseteq \Box (\mathrm{list}[n]^{\alpha} \tau)}\ \mathbf{l}\Box$$

Figure 34: DuCostIt's relational subtyping rules (part 1)

$\boxed{\Delta; \Phi \models \tau_1 \sqsubseteq \tau_2}$ Binary type τ_1 is a subtype of type τ_2

$$\frac{i :: S, \Delta; \Phi \models \tau \sqsubseteq \tau' \qquad i \notin FV(\Phi)}{\Delta; \Phi \models \exists i{::}S.\, \tau \sqsubseteq \exists i{::}S.\, \tau'}\ \exists$$

$$\frac{}{\Delta; \Phi \models \exists i{::}S.\, \Box\tau \sqsubseteq \Box(\exists i{::}S.\, \tau)}\ \exists\Box$$

$$\frac{\Delta; \Phi \wedge C \models C' \qquad \Delta; \Phi \models \tau \sqsubseteq \tau'}{\Delta; \Phi \models C \,\&\, \tau \sqsubseteq C' \,\&\, \tau'}\textbf{c-and}$$

$$\frac{}{\Delta; \Phi \models C \,\&\, \Box\tau \sqsubseteq \Box(C \,\&\, \tau)}\ \textbf{c-and-}\Box$$

$$\frac{\Delta; \Phi \wedge C' \models C \qquad \Delta; \Phi \models \tau \sqsubseteq \tau'}{\Delta; \Phi \models C \supset \tau \sqsubseteq C' \supset \tau'}\textbf{c-impl}$$

$$\frac{}{\Delta; \Phi \models \Box(C \supset \tau) \sqsubseteq C \supset \Box\tau}\ \textbf{c-impl-}\Box \qquad \frac{}{\Delta; \Phi \models \Box\tau \sqsubseteq \tau}\ \textbf{T}$$

$$\frac{}{\Delta; \Phi \models \Box\tau \sqsubseteq \Box\Box\tau}\ \textbf{D} \qquad \frac{\Delta; \Phi \models \tau_1 \sqsubseteq \tau_2}{\Delta; \Phi \models \Box\tau_1 \sqsubseteq \Box\tau_2}\ \textbf{B-}\Box$$

$$\frac{}{\Delta; \Phi \models \tau \sqsubseteq \mathsf{U}\,|\tau|}\ \textbf{W} \qquad \frac{\Delta; \Phi \models^{A} A \sqsubseteq A'}{\Delta; \Phi \models \mathsf{U}\,A \sqsubseteq \mathsf{U}\,A'}\ \textbf{U} \qquad \frac{}{\Delta; \Phi \models \tau \sqsubseteq \tau}\textbf{refl}$$

$$\frac{\Delta; \Phi \models \tau_1 \sqsubseteq \tau_2 \qquad \Delta; \Phi \models \tau_2 \sqsubseteq \tau_3}{\Delta; \Phi \models \tau_1 \sqsubseteq \tau_3}\textbf{tran}$$

Figure 35: DuCostIt's relational subtyping rules (Part 2)

9

DUCOSTIT'S METATHEORY AND SOUNDNESS

▶ SYNOPSIS In this chapter, we present a logical relations model for DuCostIt and use it to prove DuCostIt sound relative to the from-scratch and abstract change propagation cost semantics presented in Section 8.1.

Like in RelCost, we build two cost-annotated models of types: a *non-relational* (unary) one for from-scratch execution and a *relational* (binary) one for change propagation . However, in addition to these two models, in DuCostIt, we need another relational model to handle bivalues of type $\mathcal{U}\ A$ that can still be change propagated.

9.1 UNARY INTERPRETATION OF DUCOSTIT TYPES

DuCostIt's unary model resembles RelCost's unary model (modulo the lower bounds in RelCost). For each unary type A, the value interpretation $[\![A]\!]_v$ is a set, containing pairs (m, v) of step indices and values (shown in Figure 36).

The expression interpretation $[\![A]\!]^t_\varepsilon$ is shown below and contains pairs (m, e) of step indices and expressions.

$$[\![A]\!]^t_\varepsilon = \{(m, e) \mid (e \Downarrow^f \langle v, D\rangle \wedge f < m) \Rightarrow \begin{array}{l} 1.\ f \leqslant t \\ 2.\ (m - f, v) \in [\![A]\!]_v \end{array}\}$$

The interpretation of $[\![A]\!]^t_\varepsilon$ states that if e evaluates to a value with cost $f < m$, then t is an upper bound on the from-scratch execution cost f, and the resulting value is in the value interpretation with step-index $m - f$.

As in RelCost's model, we interpret open expressions under some semantic environment interpretation γ. We write $(m, \gamma) \in \mathcal{G}[\![\Omega]\!]$ to mean that γ maps all variables in the domain of the environment Ω to appropriately-typed semantic values for m steps.

$$\begin{aligned} \mathcal{G}[\![\cdot]\!] &= \{(m, \emptyset)\} \\ \mathcal{G}[\![\Omega, x : A]\!] &= \{(m, \gamma[x \mapsto v]) \mid (m, \gamma) \in \mathcal{G}[\![\Omega]\!] \wedge (m, v) \in [\![A]\!]_v\} \end{aligned}$$

We write $\sigma \in \mathcal{D}[\![\Delta]\!]$ to mean that σ is a valid (well-sorted) substitution for the index environment Δ.

$[\![A]\!]_v \subseteq \text{Step index} \times \text{Value}$
$[\![A]\!]^k_\varepsilon \subseteq \text{Step index} \times \text{Expression}$

$$
\begin{aligned}
[\![\text{int}]\!]_v &= \{(m, n)\} \\
[\![\text{unit}]\!]_v &= \{(m, ())\} \\
[\![A_1 \times A_2]\!]_v &= \{(m, \langle v_1, v_2\rangle) \mid (m, v_1) \in [\![A_1]\!]_v \wedge \\
&\qquad\qquad (m, v_2) \in [\![A_2]\!]_v\} \\
[\![A_1 + A_2]\!]_v &= \{(m, \text{inl } v) \mid (m, v) \in [\![A_1]\!]_v\} \cup \\
&\quad \{(m, \text{inr } v) \mid (m, v) \in [\![A_2]\!]_v\} \\
[\![\text{list}[0]\ A]\!]_v &= \{(m, \text{nil }())\} \\
[\![\text{list}[n{+}1]\ A]\!]_v &= \{(m, \text{cons}(e_1, e_2)) \mid (m, e_1) \in [\![A]\!]_v \wedge \\
&\qquad\qquad (m, e_2) \in [\![\text{list}[n]\ A]\!]_v\} \\
[\![A_1 \xrightarrow{\mathbb{FS}(t)} A_2]\!]_v &= \{(m, \text{fix } f(x).e) \mid \forall j < m.\ \forall v.\ (j, v) \in [\![A_1]\!]_v \\
&\qquad\qquad \implies (j, e[v/x, \text{fix } f(x).e/f]) \in [\![A_2]\!]^t_\varepsilon\} \\
[\![\forall i \overset{\mathbb{FS}(t)}{::} S.\ A]\!]_v &= \{(m, \Lambda.e) \mid \forall I.\ \vdash I :: S.\ (m, e) \in [\![A\{I/i\}]\!]^{t[I/i]}_\varepsilon\} \\
[\![\exists i{::}S.\ A]\!]_v &= \{(m, \text{pack } v) \mid \exists I.\ \vdash I :: S \wedge (m, v) \in [\![A\{I/i\}]\!]_v\} \\
[\![C \supset A]\!]_v &= \{(m, v) \mid \not\models C \vee (m, v) \in [\![A]\!]_v\} \\
[\![C\ \&\ A]\!]_v &= \{(m, v) \mid \models C \wedge (m, v) \in [\![A]\!]_v\}
\end{aligned}
$$

Figure 36: Non-relational interpretation of DuCostIt's unary types

9.2 DUCOSTIT'S SOUNDNESS (UNARY)

We prove the following fundamental theorem for unary typing. Roughly, the theorem says that the expression e, if typed in DuCostIt at unary type A, lies in the unary expression interpretation of A for any step-index and value substitution that respects the environment's types.

Theorem 6 (Fundamental Theorem for Unary Typing)**.** *Assume that* $\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A \mid t$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, \gamma) \in \mathcal{G}[\![\sigma\Omega]\!]$*. Then,* $(m, \gamma e) \in [\![\sigma A]\!]^{\sigma t}_\varepsilon$*.*

Proof. Proof is by induction on the typing derivation (shown in Appendix B.2). □

An immediate corollary of the theorem is that execution costs established in the type system are upper bounds on the actual from-scratch execution costs of the program. For readability, we only state the theorem with a single input x, but it generalizes to any number of inputs.

Corollary 7 (Soundness for from-scratch execution)**.** *Suppose that*

- $x : A \vdash_{\mathbb{FS}} e : A' \mid t$
- $\vdash_{\mathbb{FS}} v : A \mid 0$
- $e[v/x] \Downarrow^{f} T$

Then $f \leqslant t$.

9.3 RELATIONAL INTERPRETATION OF DUCOSTIT TYPES

In contrast to RelCost, the relational model of DuCostIt is built based on bivalues and biexpressions that direct change propagation. This has significant ramifications on DuCostIt's relational model, making it more involved than RelCost's.

In particular, there are two relational interpretations in DuCostIt: one for unary types and one for relational types. The relational value interpretation of a relational type, written $(\!|\tau|\!)_v$, contains pairs $(m, \mathtt{w})$ of a step-index and a bivalue (shown in Figure 37). The relational value interpretation of unary types, written $\langle\!(\, A \,)\!\rangle_v$, contains also pairs $(m, \mathtt{w})$ of a step-index and a bivalue but requires that $\mathtt{w}$ is not equal to $\mathsf{new}(v, v')$. The relation $\langle\!(\, A \,)\!\rangle_v$ is needed to allow change propagating expressions that have not changed at the top level but have type $U\, A$.[33] Both $(\!|\tau|\!)_v$ and $\langle\!(\, A \,)\!\rangle_v$ relate the original value $L(\mathtt{w})$ to the updated value $R(\mathtt{w})$. We discuss salient points of $(\!|\tau|\!)_v$ and $\langle\!(\, A \,)\!\rangle_v$.

9.3.1 *Relational interpretation of relational types*

First, the interpretation of $\Box\tau$ contains only those bivalues $\mathtt{w}$ whose two projections are *identical* and do not contain any new's, i. e. $\mathtt{stable}(\mathtt{w})$, and are related at $(\!|\tau|\!)_v$. Hence, we have $(\!|\Box\tau|\!)_v \subseteq (\!|\tau|\!)_v$.

Second, the interpretation of $U\, A$ contains bivalues of the form $\mathsf{new}(v, v')$ only if (j, v) and (j, v') are in the unary relation $[\![A]\!]_v$ *for any step index* j [34]. In addition, it also contains other bivalues in $\langle\!(\, A \,)\!\rangle_v$ that can still be change-propagated even though they are typed at the weakest relation $U\, A$. Then, we can show that $(\!|\tau|\!)_v \subseteq (\!|U\, |\tau|\!)_v$, where $|\cdot|$ is the type erasure operation defined in Figure 32.

[33] *If they had changed, i. e. were equal to* $\mathsf{new}(v, v')$*, then the values would be in the unary relation at type* A*. But if they have not changed, we need a special relation that allows them to be change-propagated through.*

[34] *Like in RelCost, this means that when we switch from relational to unary reasoning, we can call out to any unary step index* j*. This works because the unary relation does not refer back to the binary relation.*

$(\!|\tau|\!)_\nu \subseteq \text{Step index} \times \text{Bi-value}$
$(\!|\tau|\!)^{\mathsf{t}}_\varepsilon \subseteq \text{Step index} \times \text{Bi-expression}$

$$
\begin{aligned}
(\!|\Box\,\tau|\!)_\nu &= \{(m, w) \mid \mathtt{stable}(w) \wedge (m, w) \in (\!|\tau|\!)_\nu\} \\
(\!|\mathsf{U}\,A|\!)_\nu &= \mathbb{C}\,A\,\mathbb{D}_\nu \cup \{(m, \mathtt{new}(\nu, \nu')) \mid \forall j.\ (j, \nu) \in [\![A]\!]_\nu \wedge \\
&\qquad\qquad (j, \nu') \in [\![A]\!]_\nu\} \\
(\!|\mathsf{int}_r|\!)_\nu &= \{(m, \mathtt{keep}(n))\} \\
(\!|\mathsf{unit}_r|\!)_\nu &= \{(m, ())\} \\
(\!|\tau_1 \times \tau_2|\!)_\nu &= \{(m, \langle w_1, w_2\rangle) \mid (m, w_1) \in (\!|\tau_1|\!)_\nu \wedge (m, w_2) \in (\!|\tau_2|\!)_\nu\} \\
(\!|\tau_1 + \tau_2|\!)_\nu &= \{(m, \mathtt{inl}\ w) \mid (m, w) \in (\!|\tau_1|\!)_\nu\} \cup \\
&\quad \{(m, \mathtt{inr}\ w) \mid (m, w) \in (\!|\tau_2|\!)_\nu\} \\
(\!|\mathsf{list}[0]^\alpha\,\tau|\!)_\nu &= \{(m, \mathtt{nil})\} \\
(\!|\mathsf{list}[n{+}1]^\alpha\,\tau|\!)_\nu &= \{(m, \mathtt{cons}(w_1, w_2)) \mid ((m, w_1) \in (\!|\Box\,\tau|\!)_\nu \wedge (m, w_2) \in (\!|\mathsf{list}[n]^\alpha\,\tau|\!)_\nu) \\
&\qquad \vee ((m, w_1) \in (\!|\tau|\!)_\nu \wedge (m, w_2) \in (\!|\mathsf{list}[n]^{\alpha-1}\,\tau|\!)_\nu \wedge \alpha > 0)\} \\
(\!|\tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2|\!)_\nu &= \{(m, \mathtt{fix}\ f(x).ee) \mid ((m, \mathtt{fix}\ f(x).ee) \in \mathbb{C}\,|\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\tau_2|\,\mathbb{D}_\nu) \wedge \\
&\qquad (\forall j < m. \forall w. (j, w) \in (\!|\tau_1|\!)_\nu \Longrightarrow (j, ee[w/x, \mathtt{fix}\ f(x).ee/f]) \in (\!|\tau_2|\!)^{\mathsf{t}}_\varepsilon)\} \\
(\!|\forall i \overset{\mathbb{CP}(t)}{::} S.\,\tau|\!)_\nu &= \{(m, \Lambda.ee) \mid (\forall I.\ \vdash I :: S.\ ((m, ee) \in (\!|\tau\{I/i\}|\!)^{\mathsf{t}[I/i]}_\varepsilon) \wedge \\
&\qquad (m, \Lambda.ee) \in \mathbb{C}\,\forall i \overset{\mathbb{FS}(\infty)}{::} S.\,|\tau|\,\mathbb{D}_\nu\} \\
(\!|\exists i{::}S.\,\tau|\!)_\nu &= \{(m, \mathtt{pack}\ w) \mid \exists I. \vdash I :: S \wedge (m, w) \in (\!|\tau\{I/i\}|\!)_\nu\} \\
(\!|C \supset \tau|\!)_\nu &= \{(m, w) \mid \not\models C \vee (m, w) \in (\!|\tau|\!)_\nu\} \\
(\!|C\ \&\ \tau|\!)_\nu &= \{(m, w) \mid \models C \wedge (m, w) \in (\!|\tau|\!)_\nu\}
\end{aligned}
$$

Figure 37: Relational interpretation of relational types

Third, the interpretation of $\tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2$ relates a function bivalue that, given related arguments at $j < m$ steps, return related computations (in the expression relation $(\!|\tau|\!)^t_\varepsilon$ discussed below) at step-index j. In addition, the function bivalue is in the relational unary interpretation of $|\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\tau_2|$. The latter allows any function bivalue to be used in a unary context with the weakest cost bound ∞. In essence, we can *semantically* show that the relational judgment $\Delta; \Phi; \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t$ entails the unary judgment $\Delta; \Phi; |\Gamma| \vdash_{\mathbb{FS}} e : |\tau| \mid \infty$.

We interpret open biexpressions under related substitutions, δ. We write $(m, \delta) \in \mathcal{G}(\!|\Gamma|\!)$ to mean that δ maps all the variables in the domain of the environment Γ to appropriately-typed semantic relational bivalues for m steps.

$$\begin{aligned}\mathcal{G}(\!|\cdot|\!) &= \{(m, \emptyset)\}\\ \mathcal{G}(\!|\Gamma, x : \tau|\!) &= \{(m, \delta[x \mapsto w]) \mid (m, \delta) \in \mathcal{G}(\!|\Gamma|\!) \wedge (m, w) \in (\!|\tau|\!)_v\}\end{aligned}$$

9.3.2 *Relational interpretation of unary types*

Next, we explain how $(\!| A |\!)_v$ is defined. Intuitively, $(\!| A |\!)_v$ contains bivalues that have not changed at the top-level, i. e. are not $\mathsf{new}(v, v')$. Hence, once eliminated, bivalues in $(\!| A |\!)_v$ don't trigger a switch to from-scratch execution, but may be change propagated further.

First, the interpretation of $(\!| A |\!)_v$ does not contain any $\mathsf{new}(v, v')$ bivalues at the top level. However, inner parts of the bivalue may contain $\mathsf{new}(v, v')$ bivalues. For instance, for pairs (w_1, w_2) that are in the relational interpretation of $(\!| A_1 \times A_2 |\!)_v$, left and right projections must be related at $(\!|U\, A_1|\!)_v$ and $(\!|U\, A_2|\!)_v$, respectively. We need to wrap the inner types A_1 and A_2 with unrelated types and call out to the relational interpretation since the projections w_i themselves might be $\mathsf{new}(v, v')$.

In other words, the relational interpretation of unary types $(\!| A |\!)_v$ may refer to the relational interpretation of relational types $(\!|U\, A'|\!)_v$ for some smaller A'. Still, by unrolling the definitions of $(\!|U\, A'|\!)_v$ in the definition of $(\!| A |\!)_v$, it can be shown that $(\!| A |\!)_v$ is well-founded. Second, the interpretation of $A_1 \xrightarrow{\mathbb{FS}(t)} A_2$ relates a function bivalue that, given possibly unrelated arguments related at $(\!|U\, A_1|\!)_v$ at $j < m$ steps, returns possibly unrelated computations (in the expression relation $(\!|U\, A_2|\!)^t_\varepsilon$ discussed below) at step-index j.[35] This is needed because we may change propagate through the body of a function even if that body was typed in $\mathbb{FS}$-mode. It also allows us to show that

[35] *Notice that there is no expression relation corresponding to the value relation $(\!| A |\!)_v$. Instead, it refers to the expression relation $(\!|U\, A|\!)^{\cdot}_\varepsilon$.*

$$\text{(} A \text{)}_\nu \subseteq \text{Step index} \times \text{Bi-value}$$

$$\begin{aligned}
\text{(} \text{int} \text{)}_\nu &= \{(m, \mathtt{keep}(n))\} \\
\text{(} \text{unit} \text{)}_\nu &= \{(m, ())\} \\
\text{(} A_1 \times A_2 \text{)}_\nu &= \{(m, \langle w_1, w_2\rangle) \mid (m, w_1) \in (\!|\mathsf{U}\, A_1|\!)_\nu \wedge \\
&\qquad\qquad (m, w_2) \in (\!|\mathsf{U}\, A_2|\!)_\nu\} \\
\text{(} A_1 + A_2 \text{)}_\nu &= \{(m, \mathtt{inl}\ w) \mid (m, w) \in (\!|\mathsf{U}\, A_1|\!)_\nu\} \cup \\
&\quad \{(m, \mathtt{inr}\ w) \mid (m, w) \in (\!|\mathsf{U}\, A_2|\!)_\nu\} \\
\text{(} \text{list}[0]\ A \text{)}_\nu &= \{(m, \mathtt{nil})\} \\
\text{(} \text{list}[n{+}1]\ A \text{)}_\nu &= \{(m, \mathtt{cons}(w_1, w_2)) \mid ((m, w_1) \in (\!|\mathsf{U}\, A|\!)_\nu \wedge \\
&\qquad\qquad (m, w_2) \in (\!|\mathsf{U}\, (\text{list}[n]\ A)|\!)_\nu)\} \\
\text{(} A_1 \xrightarrow{\mathbb{FS}(t)} A_2 \text{)}_\nu &= \{(m, \mathtt{fix}\ f(x).ee) \mid (\forall j < m.\ \forall w.\ (j, w) \in (\!|\mathsf{U}\, A_1|\!)_\nu \\
&\qquad\qquad \implies (j, ee[w/x, \mathtt{fix}\ f(x).ee/f]) \in (\!|\mathsf{U}\, A_2|\!)^t_\varepsilon) \wedge \\
&\qquad (\forall j.\ (j, \mathrm{L}(\mathtt{fix}\ f(x).ee)) \in [\![A_1 \xrightarrow{\mathbb{FS}(t)} A_2]\!]_\nu \wedge \\
&\qquad\quad (j, \mathrm{R}(\mathtt{fix}\ f(x).ee)) \in [\![A_1 \xrightarrow{\mathbb{FS}(t)} A_2]\!]_\nu)\} \\
\text{(} \forall i \overset{\mathbb{FS}(t)}{::} S.\, A \text{)}_\nu &= \{(m, \Lambda.ee) \mid \forall I.\ \vdash I :: S.\ ((m, ee) \in (\!|\mathsf{U}\, (A\{I/i\})|\!)^{t[I/i]}_\varepsilon) \wedge \\
&\qquad (\forall j.(j, \mathrm{L}(ee)) \in [\![A\{I/i\}]\!]^t_\varepsilon \wedge (j, \mathrm{R}(ee)) \in [\![A\{I/i\}]\!]^t_\varepsilon)\} \\
\text{(} \exists i{::}S.\, A \text{)}_\nu &= \{(m, \mathtt{pack}\ w) \mid \exists I.\ \vdash I :: S \wedge (m, w) \in (\!|\mathsf{U}\, (A\{I/t\})|\!)_\nu\} \\
\text{(} C \supset A \text{)}_\nu &= \{(m, w) \mid \not\models C \vee (m, w) \in (\!|\mathsf{U}\, A|\!)_\nu\} \\
\text{(} C\ \&\ A \text{)}_\nu &= \{(m, w) \mid \models C \wedge (m, w) \in (\!|\mathsf{U}\, A|\!)_\nu\}
\end{aligned}$$

Figure 38: Relational interpretation of DuCostIt's unary types

the unary judgment $\Delta; \Phi; \Omega \vdash_{\mathbb{FS}} e : A \mid t$ entails the relational judgment $\Delta; \Phi; \mathsf{U}\,\Omega \vdash_{\mathbb{CP}} e : \mathsf{U}\,A \mid t$ *semantically*.

In addition, the left and right projections of the function bivalue are in the unary interpretation of $A_1 \xrightarrow{\mathbb{FS}(t)} A_2$ for any step-index j.

Notice that in $\langle\!(\, A \,)\!\rangle_v$, we never refer to any expression relation at type A', but only at type $\mathsf{U}\,A'$ (where A' is smaller than A). Hence, there is no need to define a corresponding relational *expression* relation. Instead, based on the relational interpretation of relational types, we have the expression interpretation $(\!|\tau|\!)^t_\varepsilon$, defining when a biexpression (the initial expression and the updated one) is logically related at type τ with change-propagation cost t. It consists of a set of pairs of the form $(m, \mathfrak{ee})$ and ensures that change propagating $\mathfrak{ee}$ (using the rules of $\curvearrowright$) cost no more than t.

$$
\begin{aligned}
(\!|\tau|\!)^t_\varepsilon \quad = \{(m, \mathfrak{ee}) \mid \forall v, v', D, D', f, f'.& \\
L(\mathfrak{ee}) \Downarrow^f \langle v, D\rangle \;\wedge\; R(\mathfrak{ee}) \Downarrow^{f'} \langle v', D'\rangle \;\wedge\; f < m& \\
\Rightarrow \quad \exists\, \mathfrak{w}', c' \;\text{ such that}& \\
\quad 1.\ \langle\langle v, D\rangle, \mathfrak{ee}\rangle \curvearrowright \mathfrak{w}', \langle v', D'\rangle, c'& \\
\quad 2.\ v' = R(\mathfrak{w}') \;\wedge\; v = L(\mathfrak{w}')& \\
\quad 3.\ c' \leqslant t& \\
\quad 4.\ (m - f, \mathfrak{w}') \in (\!|\tau|\!)_v\}&
\end{aligned}
$$

The definition states that if the left and right projections of the biexpression $\mathfrak{ee}$ evaluate to values v and v' with derivations D and D' and in f and f' steps, respectively, and $f < m$, then we can change propagate $\mathfrak{ee}$ with cost c' and obtain an updated bivalue $\mathfrak{w}'$ specifying how the initial output v is modified to v'. More importantly, we know that t is an upper bound on the change propagation cost of $\mathfrak{ee}$, i.e., $c' \leqslant t$ and the resulting bivalue $\mathfrak{w}'$ is related at step-index $m - f$.

9.4 DUCOSTIT'S SOUNDNESS (RELATIONAL)

We prove DuCostIt's type system sound with respect to the abstract evaluation and change propagation semantics. We show that the costs t estimated by expression typing for relational judgment are upper bounds on the costs of change propagation, respectively.

Theorem 8 (Fundamental Theorem for DuCostIt's Relational Typing)**.** *Assume that* $\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$*. Then,* $(m, \delta e) \in (\!|\sigma\tau|\!)^{\sigma t}_\varepsilon$.

Proof. Proof is by induction on the typing derivation (shown in Appendix B.2) □

An immediate corollary of the theorem is that update costs established in the type system are upper bounds on the cost of change propagation. For readability, we only state the theorem with a single input x, but generalized versions with any number of inputs hold as well.

Corollary 9 (Soundness for change propagation)**.** *Suppose that*

- $x : \tau \vdash_{\mathbb{CP}} e : \tau' \mid t$
- $\vdash w \gg \tau$
- $e[L(w)/x] \Downarrow^{f} T$
- $e[R(w)/x] \Downarrow^{f'} T'$

Then the following hold for some w' and c':

1. $\langle T, \ulcorner e \urcorner[w/x] \rangle \curvearrowright w', T', c'$
2. $c' \leqslant t$.

Finally, we prove that, semantically, a) relational typing is a refinement of unary typing with the weakest bound ∞ on the from-scratch execution cost and b) the unary judgment entails the relational judgment at the weakest relation.

Theorem 10 (Fundamental Theorem for DuCostIt's Weak Relational Typing)**.** *Assume that* $\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, \gamma) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$*, then* $(m, \gamma e) \in [\![|\sigma\tau|]\!]^{\infty}_{\varepsilon}$.

Proof. Proof is by induction on the typing derivation (shown in Appendix B.2). □

Theorem 11 (Fundamental Theorem for DuCostIt's Weak Entailment)**.** *Assume that* $\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A \mid t$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, \delta) \in \mathcal{G}[\![U\, \sigma\Omega]\!]$*, then* $(m, \delta e) \in (\!|U\, \sigma A|\!)^{\sigma t}_{\varepsilon}$.

Proof. Proof is by induction on the typing derivation (shown in Appendix B.2). □

10

RELATED WORK : DYNAMIC STABILITY

In this chapter, we review related work on incremental computation and provide a detailed comparison to our work on the first version of DuCostIt, a homonymous system [35] which we refer to as DuCostIt^0. Then, we discuss related work in the context of program (input-output) sensitivity.

10.1 INCREMENTAL COMPUTATION

There is a vast amount of literature on incremental computation, ranging from algorithmic techniques like memoization [59, 75], and language-based approaches using dynamic dependence graphs [5, 29, 32], to static techniques like finite differencing [26, 66, 84].

LANGUAGE-BASED TECHNIQUES To speed up incremental runs, approaches based on dynamic dependency graphs store intermediate results from the initial run. A prominent language-based technique that uses this approach is self-adjusting computations (AFL) [5], which has been subsequently expanded to Standard ML [6] and a dialect of C [55]. Our change propagation semantics is mainly inspired by AFL.

Previous work by Chen *et al.* [32, 33] automatically translates purely functional programs to their incremental counterparts. However, Chen *et al.* only show that the initial run of the translated program is no slower (asymptotically) than the source program. They do not analyze costs for incremental runs. In contrast, we show that both incremental and from-scratch costs of translated programs are bounded by those estimated by our type system. (Chen *et al.*'s type system does not provide cost bounds.)

In general, in all prior work on incremental computation the efficiency of incremental updates is established either by empirical analysis of benchmark programs, algorithmic analysis or direct analysis of cost semantics [73]. My prior work CostIt [34] was the first proposal for statically analyzing dynamic stability. DuCostIt directly builds on CostIt, as well as DuCostIt^0, but our type system is richer: CostIt cannot type programs in which fresh closures may execute in the incremental runs. DuCostIt does away with this restriction by introducing a second typ-

ing mode that analyzes from-scratch execution costs. This requires a redesign of the type system and substantially complicates the metatheory. (DuCostIt uses both a binary and a unary logical relation, while CostIt needs only the former, and it allows the analysis of many programs that CostIt cannot handle.)

FINITE DIFFERENCING Approaches based on static transformations extract program derivatives, which can be executed in place of the original programs with only the updated inputs to produce updated results [26, 84]. Such techniques make use of the algebraic properties of a set of primitives and restrict the programmer to only those primitives. In contrast to these approaches, our change propagation semantics is based on dynamic dependence graphs and our static analysis only establishes the cost of incremental runs.

10.2 COMPARISON TO DUCOSTIT^0

DuCostIt presented in this thesis is partly based on my homonymous system DuCostIt^0 [35]. However, both the design of DuCostIt's type system as well as its semantic model differ significantly from DuCostIt^0.

$$
\begin{array}{lrcl}
\text{Unary types} & A & ::= & \mathsf{int} \mid A_1 \times A_2 \mid A_1 + A_2 \\
& & & \mid \mathsf{list}[n]\, A \mid A_1 \xrightarrow{\mathbb{FS}(t)} A_2 \mid \cdots \\
\text{Relational types} & \tau & ::= & \mathsf{int}_r \mid \tau_1 \times \tau_2 \mid \tau_1 + \tau_2 \\
& & & \mid \mathsf{list}[n]^{\alpha}\, \tau \mid \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2 \mid \cdots \\
\hline
\text{Unannot. types} & A & ::= & \mathsf{int} \mid \tau_1 \times \tau_2 \mid \tau_1 + \tau_2 \mid \mathsf{list}[n]^{\alpha}\, \tau \mid \\
& & & \tau_1 \xrightarrow{S(t)} \tau_2 \mid \tau_1 \xrightarrow{C(t)} \tau_2 \mid \cdots \\
\text{Types} & \tau & ::= & (A)^{S} \mid (A)^{C} \mid \Box\, \tau
\end{array}
$$

Figure 39: Types of DuCostIt and DuCostIt^0

The main source of the discrepancy between DuCostIt and DuCostIt^0 is their type grammar. While DuCostIt's type grammar is designed to be two layered, where types are syntactically separated into unary and relational types based on the typing mode $\mathbb{CP}/\mathbb{FS}$, DuCostIt^0's type grammar is not. Instead, DuCostIt^0's type grammar is uniform and is based on annotated types. Annotated types are often used in type sys-

tems for dependency analysis. For instance, in information-flow control analysis, by annotating types with high (changeable) and low (stable) labels, information flow from high to low values can be tracked and forbidden. Motivated by such annotations, DuCostIt^0 has two kinds of annotated types: $(A)^{\mathbb{S}}$ specifies those values of type A that will not change in the second execution ($\mathbb{S}$ is read "stable") whereas $(A)^{\mathbb{C}}$ specifies all values of type A ($\mathbb{C}$ is read "potentially changeable"). Although the type grammars might not seem like a big change, its ramifications are significant in the design of DuCostIt's type system as well as the semantic model.

Compared to annotated types of DuCostIt^0, the two-layered type grammar of DuCostIt has two advantages. First, two-layered types simplify the rules of the type system considerably. In DuCostIt^0, every type constructor has three elimination rules, one for use in unary reasoning and two for use in relational reasoning. In contrast, our type system has only two elimination rules for every type constructor, one for unary reasoning and the other for relational reasoning. Second, the separation reflects the unary and relational semantic interpretations *syntactically*. In contrast, in DuCostIt^0, the separation exists only in the model, with the unpleasant consequence that relational refinements like α in $(\mathrm{list}[n]^{\alpha}\,\tau)$ as well as all changeability annotations that are meaningless in unary reasoning must nonetheless be carried through unary typing derivations.

10.3 CONTINUITY AND PROGRAM SENSITIVITY

Program sensitivity/continuity analysis aims to establish how changes to inputs of a program affect the changes to its outputs. Such an analysis can be used to verify robustness properties in the context of embedded systems or continuity properties in the context of differential privacy. Although closely related to our work in concept, the end-goal in such analysis is more restricted compared to dynamic stability analysis. (Program continuity does not account for dynamic stability) DuCostIt also proves a limited form of program continuity, as an intermediate step in establishing dynamic stability. We discuss two lines of work that are related to our work.

Reed and Pierce present a linear type system called Fuzz for proving continuity [94], as an intermediate step in verifying differential privacy properties. Gaboardi *et al.* extend Fuzz with lightweight dependent types in a type system called DFuzz [51]. DFuzz's syntax and use of lightweight dependent types influenced our work significantly. A tech-

nical difference from DFuzz (and Fuzz) is that our types capture where two values differ whereas in DFuzz, the "distance" between related values is not explicit in the type, but only in the relational model. As a result, DuCostIt's type system does not need linearity, which DFuzz does. Unlike DuCostIt and DFuzz, Chaudhuri *et al.*'s static analysis can prove program continuity even with control flow changes as long as perturbations to the input result in branches that are close to each other [31].

Part III

BIRELCOST

11

BIDIRECTIONAL RELATIONAL COST ANALYSIS

▶ SYNOPSIS The goal of this chapter is to investigate issues in *implementing* a type and effect system for relational cost analysis. Specifically, we present the theory and implementation of a *bidirectional* type system for RelCost. At the end of Chapter 13, we explain how a similar development can be carried out for DuCostIt as well.

The key insight behind RelCost is that while the relative cost of two programs can be established naively by establishing the worst-case cost of one program, the best-case cost of the other program and taking their difference, this kind of a *unary* analysis is often imprecise and unnecessarily difficult, since it exploits neither the similarities between the two programs, nor the relation between their inputs. Accordingly, RelCost is a *relational* type system wherein programs are analyzed in synchrony and one falls back to the unary worst-case, best-case analysis only when the programs differ substantially in structure. Importantly, during the relational phase of the analysis, the cost established is also relative, which simplifies recurrence relations for cost in many cases and improves precision. Since costs usually depend on sizes of inputs, RelCost supports type refinements to track the sizes of data structures such as lists. To improve precision further, RelCost allows exploiting relations between corresponding values in the two programs. To this end, RelCost includes two modal refinement types ($\Box\,\tau$ and $U\,A$) that represent different relations (the diagonal relation and the trivial relation) on values as well as a relational list refinement that describes the number of places at which two related lists may differ. The subtyping rules corresponding to these relational refinements are nontrivial; in particular, subtyping is dependent on refinement constraints.

At first glance, it is unclear how such an expressive combination of relational effects, refinement types (including relational refinements), constraint-dependent subtyping, and the simultaneous combination of unary and relational typing judgments can be implemented. In this chapter, we show that, despite its rich set of features, RelCost can be implemented algorithmically and, more specifically, it can be implemented in a *bidirectional* style.

First, we introduce bidirectional typechecking and then discuss the challenges in designing a bidirectional type system for RelCost.

11.1 BIDIRECTIONAL TYPECHECKING

Bidirectional type checking is a well-established method for implementing type systems, wherein for every sub-expression, either a type is synthesized (inferred) or a given type is checked [91]. The advantage of bidirectional type checking is that it minimizes typing annotations; in most cases, type annotations are needed only on recursive functions and at explicit β-redexes. Our motivation for choosing this style is three-fold. First, bidirectional type systems can be formally described using rules that resemble standard typing rules; this simplifies proofs of soundness and completeness of the algorithmic implementation relative to the declarative type system. Second, it is known from prior work that bidirectional type checking is compatible with refinement types [44, 47, 106] and with subtyping [91], which are central to RelCost. Third, bidirectional typechecking has never been applied to relational typing and only rarely to type and effect systems [101], so a key motivation behind our development of the bidirectional type system for RelCost was to understand the interaction between bidirectionality and relational type effects.

BIDIRECTIONAL TYPECHECKING FOR RELCOST In designing and implementing the bidirectional type system, which we call BiRelCost, we discovered and addressed three main challenges.

1. **Non-syntax-directed typing rules:** As mentioned above, RelCost, like some other relational type systems, allows the relational reasoning to fall back to unary reasoning when the two programs being analyzed are dissimilar. However, the rule for switching to unary reasoning (**switch** rule in Figure 10) is not syntax-directed, since the optimal place to switch to unary reasoning depends on the specific programs being analyzed. Another example of a non-syntax-directed rule is one that allows splitting cases on the constraints (**r-split** rule in Figure 8). An implementation must manage this nondeterminism in the rules.

2. **Relational subtyping rules:** Central to the relational analysis are the relational refinements mentioned above. The subtyping rules for these refinements are not directed by the syntax of types and, in particular, transitivity of subtyping in inadmissible. Again, an implementation must somehow handle the subtyping rules.

3. **Polarity of effects:** As mentioned above, the type of every subexpression is either synthesized or checked in a bidirectional type

system. It is not clear upfront how this would generalize to the synthesis and checking of effects. As it turns out, the polarity of the type and the effect align with each other: we synthesize (check) the effect when we synthesize (check) the type. However, it is also possible to infer the effect in the checking mode with some additional constraints. We comment on this alternative design in Section 13.3.

To overcome these challenges, we follow a two-pronged approach.

On the *implementation*-side, we use several example-guided heuristics to resolve the nondeterminism in applying the typing and subtyping rules. We explain these heuristics and their effectiveness on examples. As expected, our heuristics are sound but incomplete. Nonetheless, they are sufficient for checking a variety of examples we considered, and we believe that the heuristics are quite effective. We support this claim by a case study and experimental evaluation in Section 14.4.

On the *theory*-side, our approach is more nuanced. We show that, modulo the nondeterminism, the bidirectional type checking loses no expressiveness, meaning that every program that can be typed in RelCost could also have been sufficiently annotated to resolve only the nondeterminism and then checked bidirectionally at the same type. To establish this, we follow an unusual approach. We first show that every well-typed RelCost program can be translated to a well-typed program in a core language, RelCost Core. This translation is type derivation-directed; it introduces annotations to resolve the nondeterminism in applying the typing rules and does away with relational subtyping, by replacing all instances of relational subtyping with explicit coercions (specifically, we prove that if τ is a subtype of τ' in RelCost, then there is a function of type $\tau \to \tau'$ in RelCost Core). Next, we develop the bidirectional type system, BiRelCost, for RelCost Core and prove it sound and complete w.r.t. RelCost Core's type system. It follows that every typeable RelCost program can be annotated to remove nondeterminism, and then bidirectionally type-checked.

Our implementation handles the two steps from RelCost to RelCost Core and from RelCost Core to BiRelCost simultaneously. It uses the aforementioned heuristics to implement the first step, and the bidirectional rules for the second. During both type checking and type synthesis, constraints are generated. As in prior work on DML [105, 106], these constraints capture arithmetic relationships between refinements (e.g., list sizes) of various subterms but, additionally, they also capture relational refinements and relationships between their unary and relative costs. We use SMT solvers to discharge these constraints.

However, this cannot be done immediately since the constraints contain existentially quantified variables over integers and reals, which cannot be eliminated by existing SMT solvers. Therefore, we design our own algorithm to eliminate existential variables by finding substitutions for them.

12

EMBEDDING RELCOST INTO RELCOST CORE

12.1 THE NEED FOR AN EMBEDDING

Before we delve into details of the embedding of RelCost into RelCost Core, we revisit which aspects of RelCost's type system make it hard to algorithmize. We highlight some of these aspects below.

Non-syntax-directed rules: The typing rules **switch**, **r-split**, **r-contra** and **nochange** in RelCost are not syntax-directed. Hence, these rules introduce nondeterminism in an implementation.

Two ways to type "cons" construct: There are two typing rules for constructing non-empty lists in RelCost: the rule **r-cons1** applies when the heads of the two related lists may differ and the rule **r-cons2** applies when the heads may not differ. Hence, the typing of the cons construct is context-specific.

List-case branch is typed twice: In the list case analysis rule, **r-caseL**, the cons case branch is typed twice in the premises, to account for the aforementioned two introduction rules. Hence, the branch must be typed in a non-syntax-directed manner. A consequence of this double-typing of the branch is that, in RelCost, index terms cannot appear in expressions (since the two typings of a cons-branch may instantiate universally quantified index variables differently). This makes typechecking harder.

Subtyping with $\Box\tau$ *and* $(\mathsf{U}\,A)$: It is unclear how to implement the **trans** rule of the relational subtyping rules of RelCost (Figures 13 and 14). The usual solution would be to prove that transitivity is admissible among the remaining rules. However, it is unclear how to prove the admissibility of transitivity in the presence of the comonadic type $\Box\tau$ and the modality $\mathsf{U}\,A$, whose subtyping rules interact with the other connectives in nontrivial ways.

All these difficulties, with the exception of those due to the rules **r-split** and **r-contra**, are a consequence of the presence of either relational refinements or relational effects. Consequently, they do not arise in other unary type and effect systems. In particular, the difficulty with

transitivity is specific to the relational subtyping; in unary subtyping, transitivity is admissible and poses no difficulty.

To address these difficulties, we give an *embedding* of RelCost into an intermediate language we call RelCost Core that has only type-directed rules and no relational subtyping. The goal of this embedding is to show that there is a language (RelCost Core) that is as expressive as RelCost and that is also amenable to a complete bidirectional typing procedure. RelCost Core has explicit syntactic markers to indicate which typing rules to apply where, thus resolving the nondeterminism in the first two points above. Additionally, RelCost Core's list case construct has two separate branches for the two typings of the cons case, thus allowing RelCost Core to include instantiations of universally quantified parameters explicitly in expressions. This resolves the difficulty mentioned in the third point above. Finally, we address the fourth point by replacing all occurrences of relational subtyping with explicit coercion functions (that we show to exist) in the embedding. Elimination of subtyping is a common technique for simplifying typechecking [24, 37]. However, as explained below, RelCost's subtyping is constraint-dependent, so extra care is needed when dealing with index terms appearing in the types.

We prove that our embedding is complete, in the sense that every well-typed RelCost program can be embedded into a well-typed RelCost Core program (Theorem 57).

12.2 RELCOST CORE TYPE SYSTEM

SYNTAX OF RELCOST CORE The expression syntax of RelCost Core is an extension of RelCost's syntax with several additional syntactic constructs. Each syntactic construct is a specific marker specifying how the nondeterminism in RelCost's typing rules is resolved.

The construct "`switch` e" marks the use of **switch** rule that switches to the unary reasoning whereas the construct "`NC` e" marks the use of the **nochange** rule. The construct "`split` (e_1, e_2) with C" records the constraint C that is used to case analyze the index domain (rule **r-split**). The construct "`contra` e" is used to make contradiction in the constraint domain explicit (rule **r-contra**). In addition, we add index terms to expressions. For instance, the elimination form for universally quantified types in RelCost Core is "e[I]" as opposed to RelCost's "e[]". The list constructor "cons" in RelCost is duplicated in RelCost Core as "cons_C" and "cons_NC" corresponding to the two rules **r-cons1** and **r-**

cons2. The list case construct has two separate branches for these two cons cases.

$$\begin{aligned}\text{Expression } e ::= \; & \cdots \mid \texttt{switch}\ e \mid \texttt{NC}\ e \mid \texttt{split}\ (e_1, e_2)\ \text{with C} \mid \\ & \texttt{contra}\ e \mid \texttt{der}\ e \mid \Lambda.\mathrm{i}.e \mid e[\mathrm{I}] \mid \text{pack } e \text{ with I} \mid \\ & \text{unpack } e_1 \text{ as } (x, \mathrm{i}) \text{ in } e_2 \mid \text{cons}_{\text{NC}}(e_1, e_2) \mid \\ & \text{cons}_{\text{C}}(e_1, e_2) \mid \begin{pmatrix} \text{case } e \text{ of nil} & \to e_1 \\ \mid \mathrm{h} ::_{\text{NC}} \mathrm{tl} & \to e_2 \mid \mathrm{h} ::_{\text{C}} \mathrm{tl} \to e_3 \end{pmatrix}\end{aligned}$$

RELCOST CORE TYPING RULES Like RelCost, there are two typing judgments in RelCost Core: $\Delta; \Phi_a; \Omega \vdash^{t}_{k} e :^{c} A$ for unary typing and $\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t :^{c} \tau$ for relational typing. These two judgments are distinguished from their RelCost counterparts by the superscript c in $:^{c}$. RelCost Core's typing rules are shown in Figures 40 to 42. Most of RelCost Core's unary and relational typing rules mimic RelCost's. However, there are two key differences.

First, the RelCost rules **nochange**, **switch**, **r-contra**, **r-split**, **r-cons1** and **r-cons2** are not syntax-directed. The corresponding RelCost Core rules **c-nochange**, **c-switch**, **c-r-contra**, **c-r-split**, **c-r-cons1** and **c-r-cons2** are distinguished from the their respective counterparts in RelCost by the syntactic markers $\texttt{NC}\ \cdot$, $\texttt{switch}\ \cdot$, $\texttt{contra}\ $, $\texttt{split}\ \cdot$ with $\text{C}, \text{cons}_{\text{C}}$ and cons_{NC}.

Second, there is no relational subtyping in RelCost Core (RelCost's unary subtyping is retained in RelCost Core, since it poses no difficulty for algorithmization). Instead, there is equivalence checking for relational types, written $\equiv$ (rule **c-r-≡**). Equivalence is a very simple relation. It only lifts equality modulo constraints to types (e.g., $\text{list}[1+2]^{\alpha}\tau \equiv \text{list}[3]^{\alpha}\tau$) and it can be easily implemented algorithmically (modulo constraint solving). Type equivalence rules are shown in Figure 43. In particular, equivalence at $\Box$- and U-annotated types is very straightforward (rules **eq-B-$\Box$** and **eq-U** for $\Box$- and U-annotated types, respectively in Figure 43).

SIMULATING RELATIONAL SUBTYPING WITH COERCIONS In the embedding of RelCost into RelCost Core, we replace all occurrences of relational subtyping with explicit coercion functions in RelCost Core. To do this, we have to show that if $\tau \sqsubseteq \tau'$ in RelCost, then there is a coercion function of type $\tau \to \tau'$ in RelCost Core. To handle cases with $\Box$, we add one additional syntactic construct, $\texttt{der}\ e$, and a corresponding typing rule, **c-der**, to RelCost Core (shown in Figure 40). This typing rule corresponds to the coercion $\Box\tau \to \tau$. Using this rule and the simple type equivalence $\equiv$, we show that all subtyping rules of RelCost can be

$\boxed{\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t :^{c} \tau}$ Relative cost of e_1 with respect to e_2 is upper bounded by t and the two RelCost Core expressions have relational type τ.

$$\frac{\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t :^{c} \Box\tau}{\Delta; \Phi_a; \Gamma \vdash \mathtt{der}\ e_1 \ominus \mathtt{der}\ e_2 \lesssim t :^{c} \tau}\ \textbf{c-der}$$

$$\frac{\Delta; \Phi_a; \Box\Gamma \vdash e \ominus e \lesssim t :^{c} \tau}{\Delta; \Phi_a; \Box\Gamma, \Gamma' \vdash \mathtt{NC}\ e \ominus \mathtt{NC}\ e \lesssim 0 :^{c} \Box\tau}\ \textbf{c-nochange}$$

$$\frac{\Delta; \Phi_a; |\Gamma|_1 \vdash^{t_1}_{k_1} e_1 :^{c} A_1 \qquad \Delta; \Phi_a; |\Gamma|_2 \vdash^{t_2}_{k_2} e_2 :^{c} A_2}{\Delta; \Phi_a; \Gamma \vdash \mathtt{switch}\ e_1 \ominus \mathtt{switch}\ e_2 \lesssim t_1 - k_2 :^{c} U(A_1, A_2)}\ \textbf{c-switch}$$

$$\frac{\Delta; \Phi_a \wedge C; \Gamma \vdash e_1 \ominus e_2 \lesssim t :^{c} \tau \qquad \Delta; \Phi_a \wedge \neg C; \Gamma \vdash e_1' \ominus e_2' \lesssim t :^{c} \tau}{\Delta; \Phi_a; \Gamma \vdash \mathtt{split}\ (e_1, e_1')\ \mathtt{with}\ C \ominus \mathtt{split}\ (e_2, e_2')\ \mathtt{with}\ C \lesssim t :^{c} \tau}\ \textbf{c-r-split}$$

$$\frac{\Delta; \Phi_a \models \bot \qquad \Delta; \Phi_a \vdash \Gamma\ \mathtt{wf}}{\Delta; \Phi_a; \Gamma \vdash \mathtt{contra}\ e_1 \ominus \mathtt{contra}\ e_2 \lesssim t :^{c} \tau}\ \textbf{c-r-contra}$$

$$\frac{\Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t :^{c} \tau \qquad \Delta; \Phi_a \models \tau \equiv \tau' \qquad \Delta; \Phi_a \models t \leqslant t'}{\Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t' :^{c} \tau'}\ \textbf{c-r-}\equiv$$

$$\frac{}{\Delta; \Phi_a; \Gamma \vdash n \ominus n \lesssim 0 :^{c} \mathrm{int}_r}\ \textbf{c-r-const} \qquad \frac{\Gamma(x) = \tau}{\Delta; \Phi_a; \Gamma \vdash x \ominus x \lesssim 0 :^{c} \tau}\ \textbf{c-r-var}$$

$$\frac{}{\Delta; \Phi_a; \Gamma \vdash () \ominus () \lesssim 0 :^{c} \mathrm{unit}_r}\ \textbf{c-r-unit}$$

$$\frac{\Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t :^{c} \tau_1 \qquad \Delta; \Phi_a \vdash \tau_2\ \mathtt{wf}}{\Delta; \Phi_a; \Gamma \vdash \mathrm{inl}\ e \ominus \mathrm{inl}\ e' \lesssim t :^{c} \tau_1 + \tau_2}\ \textbf{c-r-inl}$$

$$\frac{\Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t :^{c} \tau_2 \qquad \Delta; \Phi_a \vdash \tau_1\ \mathtt{wf}}{\Delta; \Phi_a; \Gamma \vdash \mathrm{inr}\ e \ominus \mathrm{inr}\ e' \lesssim t :^{c} \tau_1 + \tau_2}\ \textbf{c-r-inr}$$

$$\frac{\Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t :^{c} \tau_1 + \tau_2 \qquad \Delta; \Phi_a; x : \tau_1, \Gamma \vdash e_1 \ominus e_1' \lesssim t' :^{c} \tau \qquad \Delta; \Phi_a; y : \tau_2, \Gamma \vdash e_2 \ominus e_2' \lesssim t' :^{c} \tau}{\Delta; \Phi_a; \Gamma \vdash\ \mathrm{case}\ (e, x.e_1, y.e_2) \ominus\ \mathrm{case}\ (e', x.e_1', y.e_2') \lesssim t + t' :^{c} \tau}\ \textbf{c-r-case}$$

$$\frac{\Delta; \Phi_a \vdash \tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2\ \mathtt{wf} \qquad \Delta; \Phi_a; x : \tau_1, f : \tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2, \Gamma \vdash e_1 \ominus e_2 \lesssim t :^{c} \tau_2}{\Delta; \Phi_a; \Gamma \vdash \mathrm{fix}\ f(x).e_1 \ominus \mathrm{fix}\ f(x).e_2 \lesssim 0 :^{c} \tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2}\ \textbf{c-r-fix}$$

Figure 40: RelCost Core relational typing rules (Part 1)

$\boxed{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau}$ Relative cost of e_1 with respect to e_2 is upper bounded by t and the two expressions have relational type τ.

$$\frac{\Delta;\Phi_a \vdash \tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2 \ \mathtt{wf} \qquad \Delta;\Phi_a;x:\tau_1, f:\Box(\tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2), \Box\Gamma \vdash e \ominus e \lesssim t :^{\mathbf{c}} \tau_2}{\Delta;\Phi_a;\Box\Gamma \vdash \mathrm{fix}_{\mathrm{NC}}\ f(x).e \ominus \mathrm{fix}_{\mathrm{NC}}\ f(x).e \lesssim 0 :^{\mathbf{c}} \Box(\tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2)}\ \textbf{c-r-fixNC}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_1' \lesssim t_1 :^{\mathbf{c}} \tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2 \qquad \Delta;\Phi_a;\Gamma \vdash e_2 \ominus e_2' \lesssim t_2 :^{\mathbf{c}} \tau_1}{\Delta;\Phi_a;\Gamma \vdash e_1\ e_2 \ominus e_1'\ e_2' \lesssim t_1 + t_2 + t :^{\mathbf{c}} \tau_2}\ \textbf{c-r-app}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_1' \lesssim t_1 :^{\mathbf{c}} \tau_1 \qquad \Delta;\Phi_a;\Gamma \vdash e_2 \ominus e_2' \lesssim t_2 :^{\mathbf{c}} \tau_2}{\Delta;\Phi_a;\Gamma \vdash \langle e_1, e_2\rangle \ominus \langle e_1', e_2'\rangle \lesssim t_1 + t_2 :^{\mathbf{c}} \tau_1 \times \tau_2}\ \textbf{c-r-prod}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash e \ominus e' \lesssim t :^{\mathbf{c}} \tau_1 \times \tau_2 \qquad i \in \{1,2\}}{\Delta;\Phi_a;\Gamma \vdash \pi_i(e) \ominus \pi_i(e') \lesssim t :^{\mathbf{c}} \tau_i}\ \textbf{c-r-proj}_i$$

$$\frac{\Delta;\Phi_a \vdash \tau\ \mathtt{wf}}{\Delta;\Phi_a;\Gamma \vdash \mathrm{nil} \ominus \mathrm{nil} \lesssim 0 :^{\mathbf{c}} \mathrm{list}[0]^{\alpha}\tau}\ \textbf{c-r-nil}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_1' \lesssim t_1 :^{\mathbf{c}} \tau \qquad \Delta;\Phi_a;\Gamma \vdash e_2 \ominus e_2' \lesssim t_2 :^{\mathbf{c}} \mathrm{list}[n]^{\alpha}\tau}{\Delta;\Phi_a;\Gamma \vdash \mathrm{cons}_{\mathrm{C}}(e_1,e_2) \ominus \mathrm{cons}_{\mathrm{C}}(e_1',e_2') \lesssim t_1 + t_2 :^{\mathbf{c}} \mathrm{list}[n+1]^{\alpha+1}\tau}\ \textbf{c-r-cons1}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_1' \lesssim t_1 :^{\mathbf{c}} \Box\tau \qquad \Delta;\Phi_a;\Gamma \vdash e_2 \ominus e_2' \lesssim t_2 :^{\mathbf{c}} \mathrm{list}[n]^{\alpha}\tau}{\Delta;\Phi_a;\Gamma \vdash \mathrm{cons}_{\mathrm{NC}}(e_1,e_2) \ominus \mathrm{cons}_{\mathrm{NC}}(e_1',e_2') \lesssim t_1 + t_2 :^{\mathbf{c}} \mathrm{list}[n+1]^{\alpha}\tau}\ \textbf{c-r-cons2}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash e \ominus e' \lesssim t :^{\mathbf{c}} \mathrm{list}[n]^{\alpha}\tau \\ \Delta;\Phi_a \wedge n = 0;\Gamma \vdash e_1 \ominus e_1' \lesssim t' :^{\mathbf{c}} \tau' \\ i,\Delta;\Phi_a \wedge n = i+1;h:\Box\tau, tl:\mathrm{list}[i]^{\alpha}\tau,\Gamma \vdash e_2 \ominus e_2' \lesssim t' :^{\mathbf{c}} \tau' \\ i,\beta,\Delta;\Phi_a \wedge n = i+1 \wedge \alpha = \beta+1;h':\tau, tl:\mathrm{list}[i]^{\beta}\tau,\Gamma \vdash e_3 \ominus e_3' \lesssim t' :^{\mathbf{c}} \tau'\end{array}}{\Delta;\Phi_a;\Gamma \vdash \begin{array}{lll}\mathrm{case}\ e\ \mathrm{of}\ \mathrm{nil} \to e_1 & \ominus & \mathrm{case}\ e\ \mathrm{of}\ \mathrm{nil} \to e_1' \\ \mid h ::_{\mathrm{N}} tl \to e_2 & & \mid h ::_{\mathrm{N}} tl \to e_2' \\ \mid h' ::_{\mathrm{C}} tl' \to e_3 & & \mid h' ::_{\mathrm{C}} tl' \to e_3'\end{array} \lesssim t + t' :^{\mathbf{c}} \tau'}\ \textbf{c-r-caseL}$$

$$\frac{i :: S,\Delta;\Phi_a;\Gamma \vdash e \ominus e' \lesssim t :^{\mathbf{c}} \tau \qquad i \notin \mathrm{FIV}(\Phi_a;\Gamma)}{\Delta;\Phi_a;\Gamma \vdash \Lambda i.e \ominus \Lambda i.e' \lesssim 0 :^{\mathbf{c}} \forall i \overset{\mathrm{diff}(t)}{::} S.\tau}\ \textbf{c-r-iLam}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash e \ominus e' \lesssim t :^{\mathbf{c}} \forall i \overset{\mathrm{diff}(t')}{::} S.\tau \qquad \Delta \vdash I : S}{\Delta;\Phi_a;\Gamma \vdash e[I] \ominus e'[I] \lesssim t + t'[I/i] :^{\mathbf{c}} \tau\{I/i\}}\ \textbf{c-r-iApp}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash e \ominus e' \lesssim t :^{\mathbf{c}} \tau\{I/i\} \qquad \Delta \vdash I :: S}{\Delta;\Phi_a;\Gamma \vdash \mathrm{pack}\ e\ \mathrm{with}\ I \ominus \mathrm{pack}\ e'\ \mathrm{with}\ I \lesssim t :^{\mathbf{c}} \exists i{::}S.\tau}\ \textbf{c-r-pack}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_1' \lesssim t_1 :^{\mathbf{c}} \exists i{::}S.\tau_1 \\ i :: S,\Delta;\Phi_a;x:\tau_1,\Gamma \vdash e_2 \ominus e_2' \lesssim t_2 :^{\mathbf{c}} \tau_2 \qquad i \notin \mathrm{FV}(\Phi_a;\Gamma,\tau_2,t_2)\end{array}}{\Delta;\Phi_a;\Gamma \vdash \mathrm{unpack}\ e_1\ \mathrm{as}\ (x,i)\ \mathrm{in}\ e_2 \ominus \mathrm{unpack}\ e_1'\ \mathrm{as}\ (x,i)\ \mathrm{in}\ e_2' \lesssim t_1 + t_2 :^{\mathbf{c}} \tau_2}\ \textbf{c-r-unpack}$$

$$\frac{\Upsilon(\zeta) = \tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2 \qquad \Delta;\Phi_a;\Gamma \vdash e \ominus e' \lesssim t' :^{\mathbf{c}} \tau_1}{\Delta;\Phi_a;\Gamma \vdash \zeta\ e \ominus \zeta\ e' \lesssim t + t' :^{\mathbf{c}} \tau_2}\ \textbf{c-r-primapp}$$

Figure 41: RelCost Core relational typing rules (Part 2)

$\boxed{\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t :^{c} \tau}$ Relative cost of e_1 with respect to e_2 is upper bounded by t and the two expressions have relational type τ.

$$\frac{\Delta; \Phi_a \models C \qquad \Delta; \Phi_a \wedge C; \Gamma \vdash e \ominus e' \lesssim t :^{c} \tau}{\Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t :^{c} C \,\&\, \tau}\ \textbf{c-r-candI}$$

$$\frac{\begin{array}{c}\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_1' \lesssim t_1 :^{c} C \,\&\, \tau_1 \\ \Delta; \Phi_a \wedge C; x : \tau_1, \Gamma \vdash e_2 \ominus e_2' \lesssim t_2 :^{c} \tau_2\end{array}}{\Delta; \Phi_a; \Gamma \vdash \text{clet } e_1 \text{ as } x \text{ in } e_2 \ominus \text{clet } e_1' \text{ as } x \text{ in } e_2' \lesssim t_1 + t_2 :^{c} \tau_2}\ \textbf{c-r-candE}$$

$$\frac{\Delta; \Phi_a \wedge C; \Gamma \vdash e \ominus e' \lesssim t :^{c} \tau}{\Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t :^{c} C \supset \tau}\ \textbf{c-r-cimpI}$$

$$\frac{\Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t :^{c} C \supset \tau \qquad \Delta; \Phi_a \models C}{\Delta; \Phi_a; \Gamma \vdash \text{celim}_{\supset}\ e \ominus \text{celim}_{\supset}\ e' \lesssim t :^{c} \tau}\ \textbf{c-r-cimplE}$$

$$\frac{\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_1' \lesssim t_1 :^{c} \tau_1 \qquad \Delta; \Phi_a; x : \tau_1, \Gamma \vdash e_2 \ominus e_2' \lesssim t_2 :^{c} \tau_2}{\Delta; \Phi_a; \Gamma \vdash \text{let } x = e_1 \text{ in } e_2 \ominus \text{let } x = e_1' \text{ in } e_2' \lesssim t_1 + t_2 :^{c} \tau_2}\ \textbf{c-r-let}$$

$$\frac{\begin{array}{c}\Delta; \Phi_a; |\Gamma|_1 \vdash^{t_1}_{k_1} e_1 :^{c} A_1 \\ \Delta; \Phi_a; x : U\, A_1, \Gamma \vdash e_2 \ominus e \lesssim t_2 :^{c} \tau_2\end{array}}{\Delta; \Phi_a; \Gamma \vdash \text{let } x = e_1 \text{ in } e_2 \ominus e \lesssim t_1 + t_2 + c_{let} :^{c} \tau_2}\ \textbf{c-r-let-e}$$

$$\frac{\begin{array}{c}\Delta; \Phi_a; |\Gamma|_2 \vdash^{t_1}_{k_1} e_1 :^{c} A_1 \\ \Delta; \Phi_a; x : U\, A_1, \Gamma \vdash e \ominus e_2 \lesssim t_2 :^{c} \tau_2\end{array}}{\Delta; \Phi_a; \Gamma \vdash e \ominus \text{let } x = e_1 \text{ in } e_2 \lesssim t_2 - k_1 - c_{let} :^{c} \tau_2}\ \textbf{c-r-e-let}$$

$$\frac{\begin{array}{c}\Delta; \Phi_a; |\Gamma|_1 \vdash^{t} e :^{c} A_1 + A_2 \\ \Delta; \Phi_a; x : U\, A_1, \Gamma \vdash e_1 \ominus e' \lesssim t' :^{c} \tau \\ \Delta; \Phi_a; y : U\, A_2, \Gamma \vdash e_2 \ominus e' \lesssim t' :^{c} \tau\end{array}}{\Delta; \Phi_a; \Gamma \vdash \text{ case } (e, x.e_1, y.e_2) \ominus e' \lesssim t' + t + c_{case} :^{c} \tau}\ \textbf{c-r-case-e}$$

$$\frac{\begin{array}{c}\Delta; \Phi_a; |\Gamma|_2 \vdash_{k'} e' :^{c} A_1 + A_2 \\ \Delta; \Phi_a; x : U\, A_1, \Gamma \vdash e \ominus e_1' \lesssim t :^{c} \tau \\ \Delta; \Phi_a; y : U\, A_2, \Gamma \vdash e \ominus e_2' \lesssim t :^{c} \tau\end{array}}{\Delta; \Phi_a; \Gamma \vdash e \ominus \text{ case } (e', x.e_1', y.e_2') \lesssim t - k' - c_{case} :^{c} \tau}\ \textbf{c-r-e-case}$$

Figure 42: RelCost Core relational typing rules (Part 3)

realized as coercion functions in RelCost Core. Importantly, the coercion functions have zero relative cost, so applying the coercions (in place of the subtyping) does not change the relative costs of the expressions.

Lemma 12 (Existence of coercions for relational subtyping). *If* $\Delta; \Phi_a \models \tau \sqsubseteq \tau'$ *then there exists* $\mathtt{coerce}_{\tau,\tau'} \in$ *RelCost Core such that* $\Delta; \Phi; \cdot \vdash \mathtt{coerce}_{\tau,\tau'} \ominus \mathtt{coerce}_{\tau,\tau'} \lesssim 0 :^{\mathbf{c}} \tau \xrightarrow{diff(0)} \tau'$.

We show the coercions for some of the subtyping rules of Figures 13 and 14 below.[36]

[36] *Lemma 48 in Appendix C presents coercions for all of the subtyping rules.*

- (Rule $\to \Box$**diff**) For $\tau = \Box(\tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2)$ and $\tau' = \Box\tau_1 \xrightarrow{\mathrm{diff}(0)} \Box\tau_2$, $\mathtt{coerce}_{\tau,\tau'} = \lambda x.\lambda y.\mathtt{NC}\ ((\mathtt{der}\ x)\ (\mathtt{der}\ y))$.
- (Rule **T**) For $\tau = \Box\tau'$, $\mathtt{coerce}_{\tau,\tau'} = \lambda x.\mathtt{der}\ x$.
- (Rule **D**) For $\tau = \Box\tau_1$ and $\tau' = \Box\Box\tau_1$, $\mathtt{coerce}_{\tau,\tau'} = \lambda x.\mathtt{NC}\ x$.
- (Rule **r-l**$\Box$) For $\tau = \mathrm{list}[n]^{\alpha}\,\Box\tau$ and $\tau' = \Box(\mathrm{list}[n]^{\alpha}\,\tau)$, $\mathtt{coerce}_{\tau,\tau'} = \lambda x.\mathrm{fList}\ ()\ [n][\alpha]\ x$ where fList is

  $\mathrm{fix}\ \mathrm{fList}(\_).\Lambda n.\Lambda\alpha.\lambda x.$
  $\mathrm{case}\ e\ \mathrm{of}\ \mathrm{nil} \to \mathtt{NC}\ (\mathrm{nil})$
  $\quad \mid h ::_N tl \to \mathrm{let}\ r = \mathrm{fList}\ ()\ [n-1]\ [\alpha]\ tl\ \mathrm{in}\ \mathtt{NC}\ (\mathrm{cons}_{NC}(\mathtt{der}\ h, \mathtt{der}\ r))$
  $\quad \mid h ::_C tl \to \mathrm{let}\ r = \mathrm{fList}\ ()\ [n-1]\ [\alpha-1]\ tl\ \mathrm{in}\ \mathtt{NC}\ (\mathrm{cons}_{C}(\mathtt{der}\ h, \mathtt{der}\ r))$.
- (Rule **r-**$\to$ **execdiff**) For $\tau = U(A_1 \xrightarrow{\mathrm{exec}(k,t)} A_2, A_1' \xrightarrow{\mathrm{exec}(k',t')} A_2')$ and $\tau' = U(A_1, A_1') \xrightarrow{\mathrm{diff}(t-k')} U(A_2, A_2')$, $\mathtt{coerce}_{\tau,\tau'} = \lambda x.\lambda y.\mathtt{switch}\ (x\ y)$.
- (Rule **trans**) For $\tau = \tau_1$ and $\tau' = \tau_3$, Transitivity of subtyping corresponds to the composition of coercion functions. $\mathtt{coerce}_{\tau_1,\tau_3} = \mathtt{coerce}_{\tau_2,\tau_3} \circ \mathtt{coerce}_{\tau_1,\tau_2}$

EMBEDDING Our embedding transforms a well-typed RelCost program to a well-typed RelCost Core program and it is defined by induction on RelCost's typing derivations.

The unary embedding judgment

$$\Delta; \Phi_a; \Omega \vdash^{t}_{k} e \rightsquigarrow e^* : A$$

$\boxed{\Delta; \Phi_a \models \tau_1 \equiv \tau_2}$ checks whether τ_1 is equivalent to τ_2

$$\frac{}{\Delta; \Phi_a \models \text{int}_r \equiv \text{int}_r}\ \textbf{eq-int} \qquad \frac{}{\Delta; \Phi_a \models \text{unit}_r \equiv \text{unit}_r}\ \textbf{eq-unit}$$

$$\frac{\Delta; \Phi_a \models \tau_1 \equiv \tau_1' \qquad \Delta; \Phi_a \models \tau_2 \equiv \tau_2' \qquad \Delta; \Phi_a \models t \doteq t'}{\Delta; \Phi_a \models \tau_1 \xrightarrow{\text{diff}(t)} \tau_2 \equiv \tau_1' \xrightarrow{\text{diff}(t')} \tau_2'}\ \textbf{eq-fun}$$

$$\frac{\Delta; \Phi_a \models \tau_1 \equiv \tau_1' \qquad \Delta; \Phi_a \models \tau_2 \equiv \tau_2'}{\Delta; \Phi_a \models \tau_1 \times \tau_2 \equiv \tau_1' \times \tau_2'}\ \textbf{eq-prod}$$

$$\frac{\Delta; \Phi_a \models \tau_1 \equiv \tau_1' \qquad \Delta; \Phi_a \models \tau_2 \equiv \tau_2'}{\Delta; \Phi_a \models \tau_1 + \tau_2 \equiv \tau_1' + \tau_2'}\ \textbf{eq-sum}$$

$$\frac{\Delta; \Phi_a \models n \doteq n' \qquad \Delta; \Phi_a \models \alpha \doteq \alpha' \qquad \Delta; \Phi_a \models \tau \equiv \tau'}{\Delta; \Phi_a \models \text{list}[n]^{\alpha}\,\tau \equiv \text{list}[n']^{\alpha'}\,\tau'}\ \textbf{eq-list}$$

$$\frac{i, \Delta; \Phi_a \models \tau \equiv \tau' \qquad i, \Delta; \Phi_a \models t \doteq t'}{\Delta; \Phi_a \models \forall i \overset{\text{diff}(t)}{::} S.\,\tau \equiv \forall i \overset{\text{diff}(t')}{::} S.\,\tau'}\ \textbf{eq-}\forall$$

$$\frac{i, \Delta; \Phi_a \models \tau \equiv \tau' \qquad i \notin \text{FV}(\Phi_a)}{\Delta; \Phi_a \models \exists i{::}S.\,\tau \equiv \exists i{::}S.\,\tau'}\ \textbf{eq-}\exists \qquad \frac{\Delta; \Phi_a \models \tau \equiv \tau'}{\Delta; \Phi_a \models \Box\tau \equiv \Box\tau'}\ \textbf{eq-B-}\Box$$

$$\frac{\Delta; \Phi_a \models^A A_1' \sqsubseteq A_1 \qquad \Delta; \Phi_a \models^A A_1 \sqsubseteq A_1' \qquad \Delta; \Phi_a \models^A A_2 \sqsubseteq A_2' \qquad \Delta; \Phi_a \models^A A_2' \sqsubseteq A_2}{\Delta; \Phi_a \models U\,(A_1, A_2) \equiv U\,(A_1', A_2')}\ \textbf{eq-U}$$

$$\frac{\Delta; C \wedge \Phi_a \models C' \qquad \Delta; C' \wedge \Phi_a \models C \qquad \Delta; \Phi_a \models \tau \equiv \tau'}{\Delta; \Phi_a \models C \supset \tau \equiv C' \supset \tau'}\ \textbf{eq-c-impl}$$

$$\frac{\Delta; C' \wedge \Phi_a \models C \qquad \Delta; C \wedge \Phi_a \models C' \qquad \Delta; \Phi_a \models \tau \equiv \tau'}{\Delta; \Phi_a \models C\ \&\ \tau \equiv C'\ \&\ \tau'}\ \textbf{eq-c-prod}$$

Figure 43: RelCost Core binary type equivalence rules

means that the RelCost expression e of type A with minimum and maximum costs k and t, respectively, translates to the RelCost Core expression e^* of the same type and with the same minimum and maximum costs. In essence, the unary embedding is trivial since most of the nondeterminism lies in the relational typing. Nonetheless, to have a uniform syntax for RelCost Core, a unary embedding is necessary to deal with the syntactic markers introduced by the surrounding relational expressions (e.g. the two cons branches in list case analysis). The rules of the unary embedding are shown in Figures 44 and 45. [37]

[37] *In **e-u-cons** rule, there is a choice in picking either the $cons_C$ or $cons_{NC}$ construct.*

The relational embedding judgment

$$\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \rightsquigarrow e_1^* \ominus e_2^* \lesssim t : \tau$$

transforms a pair of (related) programs and means that the RelCost expressions e_1 and e_2 of relational type τ with relative costs t respectively translate to the RelCost Core expressions e_1^* and e_2^* of the same relational type and with the same relative cost. The relational embedding resolves the nondeterminism inherent in RelCost's non-syntax directed rules. The relational embedding rules are shown in Figures 46 to 49.

The rule **e-switch** adds the RelCost Core expression construct `switch` to the transformed left and right expressions. The rule **e-nochange** coerces all the free variables in the context Γ to their $\Box$-ed forms and then adds the RelCost Core construct `NC`. The rule **e-r-split** adds the RelCost Core construct `split` to the transformed left and right expressions with the case-analyzed constraint C. The rule **e-r-contra** adds the RelCost Core construct `contra` to the left and right expressions whenever there is a contradiction in the constraint domain.

The rule **e-r-caseL** duplicates the cons case branch in the list-case construct (with possibly different instantiations of universally quantified variables). The rules **e-r-iLam** and **e-r-iApp** add the index variable and the index term, respectively to the corresponding introduction and elimination forms of the universally quantified types. Conversely, the rules **e-r-pack** and **e-r-unpack** add the index term and the index variable, respectively to the corresponding introduction and elimination forms of the existentially quantified types. The rule **e-r-$\sqsubseteq$** transforms uses of relational subtyping to the application of coercion functions. The rest of the embedding rules is straightforward.

Our embedding preserves well-typedness and is complete, as formalized in the following theorems.

Theorem 13 (Types are preserved by embedding)**.**

1. *If* $\Delta; \Phi_a; \Omega \vdash^{t}_{k} e \rightsquigarrow e^* : A$, *then* $\Delta; \Phi_a; \Omega \vdash^{t}_{k} e^* :^{c} A$ *and* $\Delta; \Phi_a; \Omega \vdash^{t}_{k} e : A$.

2. *If* $\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \rightsquigarrow e_1^* \ominus e_2^* \lesssim t : \tau$, *then* $\Delta; \Phi_a; \Gamma \vdash e_1^* \ominus e_2^* \lesssim t :^{c} \tau$ *and* $\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau$.

Proof. By simultaneous induction on the given derivations (shown in Appendix C.1). □

Theorem 14 (Completeness of embedding)**.**

1. *If* $\Delta; \Phi_a; \Omega \vdash^{t}_{k} e : A$, *then there exists an* e^* *such that* $\Delta; \Phi_a; \Omega \vdash^{t}_{k} e \rightsquigarrow e^* : A$.

2. *If* $\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau$, *then there exist* e_1^*, e_2^* *such that* $\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \rightsquigarrow e_1^* \ominus e_2^* \lesssim t : \tau$.

Proof. By simultaneous induction on the given RelCost derivations (shown in Appendix C.1). □

$\boxed{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e \rightsquigarrow e^* : A}$ Expression e is embedded into $e*$ with the unary type A and the minimum and maximum execution costs k and t, respectively.

$$\frac{}{\Delta;\Phi_a;\Omega \vdash^{0}_{0} n \rightsquigarrow n : \mathsf{int}}\ \textbf{e-u-const} \qquad \frac{\Omega(x) = A}{\Delta;\Phi_a;\Omega \vdash^{0}_{0} x \rightsquigarrow x : A}\ \textbf{e-u-var}$$

$$\frac{}{\Delta;\Phi_a;\Omega \vdash^{0}_{0} () \rightsquigarrow () : \mathsf{unit}}\ \textbf{e-u-unit}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e \rightsquigarrow e^* : A_1 \qquad \Delta;\Phi_a \vdash A_2\ \mathtt{wf}}{\Delta;\Phi_a;\Omega \vdash^{t}_{k} \mathsf{inl}\ e \rightsquigarrow \mathsf{inl}\ e^* : A_1 + A_2}\ \textbf{e-u-inl}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e \rightsquigarrow e^* : A_2 \qquad \Delta;\Phi_a \vdash A_1\ \mathtt{wf}}{\Delta;\Phi_a;\Omega \vdash^{t}_{k} \mathsf{inr}\ e \rightsquigarrow \mathsf{inr}\ e^* : A_1 + A_2}\ \textbf{e-u-inr}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Omega \vdash^{t}_{k} e \rightsquigarrow e^* : A_1 + A_2 \\ \Delta;\Phi_a;x : A_1,\Omega \vdash^{t'}_{k'} e_1 \rightsquigarrow {e_1}^* : A \\ \Delta;\Phi_a;y : A_2,\Omega \vdash^{t'}_{k'} e_2 \rightsquigarrow {e_2}^* : A\end{array}}{\Delta;\Phi_a;\Omega \vdash^{t+t'+c_{case}}_{k+k'+c_{case}} \mathsf{case}\ (e, x.e_1, y.e_2) \rightsquigarrow \mathsf{case}\ (e^*, x.e_1^*, y.e_2^*) : A}\ \textbf{e-u-case}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a \vdash^{A} A_1 \xrightarrow{\mathsf{exec}(k,t)} A_2\ \mathtt{wf} \\ \Delta;\Phi_a;x : A_1, f : A_1 \xrightarrow{\mathsf{exec}(k,t)} A_2,\Omega \vdash^{t}_{k} e \rightsquigarrow e^* : A_2\end{array}}{\Delta;\Phi_a;\Omega \vdash^{0}_{0} \mathsf{fix}\ f(x).e \rightsquigarrow \mathsf{fix}\ f(x).e^* : A_1 \xrightarrow{\mathsf{exec}(k,t)} A_2}\ \textbf{e-u-fix}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Omega \vdash^{t_1}_{k_1} e_1 \rightsquigarrow {e_1}^* : A_1 \xrightarrow{\mathsf{exec}(k,t)} A_2 \\ \Delta;\Phi_a;\Omega \vdash^{t_2}_{k_2} e_2 \rightsquigarrow {e_2}^* : A_1\end{array}}{\Delta;\Phi_a;\Omega \vdash^{t_1+t_2+t+c_{app}}_{k_1+k_2+k+c_{app}} e_1\ e_2 \rightsquigarrow e_1^*\ e_2^* : A_2}\ \textbf{e-u-app}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{t_1}_{k_1} e_1 \rightsquigarrow {e_1}^* : A_1 \qquad \Delta;\Phi_a;\Omega \vdash^{t_2}_{k_2} e_2 \rightsquigarrow {e_2}^* : A_2}{\Delta;\Phi_a;\Omega \vdash^{t_1+t_2}_{k_1+k_2} \langle e_1, e_2\rangle \rightsquigarrow \langle e_1^*, e_2^*\rangle : A_1 \times A_2}\ \textbf{e-u-prod}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash t \ominus e \rightsquigarrow t^* \ominus e^* \lesssim k : A_1 \times A_2 \qquad i \in \{1,2\}}{\Delta;\Phi_a;\Omega \vdash^{t}_{k} \pi_i(e) \rightsquigarrow \pi_i(e^*) : A_i}\ \textbf{e-u-proj}_i$$

$$\frac{\Delta;\Phi_a \vdash A\ \mathtt{wf}}{\Delta;\Phi_a;\Omega \vdash^{0}_{0} \mathsf{nil} \rightsquigarrow \mathsf{nil} : \mathsf{list}[0]\ A}\ \textbf{e-u-nil}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{k_1}_{t_1} e_1 \rightsquigarrow {e_1}^* : A \qquad \Delta;\Phi_a;\Omega \vdash^{k_2}_{t_2} e_2 \rightsquigarrow {e_2}^* : \mathsf{list}[n]\ A}{\Delta;\Phi_a;\Omega \vdash^{t_1+t_2}_{k_1+k_2} \mathsf{cons}(e_1, e_2) \rightsquigarrow \mathsf{cons}_C(e_1^*, e_2^*) : \mathsf{list}[n+1]\ A}\ \textbf{e-u-cons}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Omega \vdash^{t}_{k} e \rightsquigarrow e^* : \mathsf{list}[n]\ A \qquad \Delta;\Phi_a \wedge n = 0;\Omega \vdash^{t'}_{k'} e_1 \rightsquigarrow {e_1}^* : A' \\ i,\Delta;\Phi_a \wedge n = i+1;h : A, tl : \mathsf{list}[i]\ A,\Omega \vdash^{t'}_{k'} e_2 \rightsquigarrow {e_2}^* : A'\end{array}}{\Delta;\Phi_a;\Omega \vdash^{t+t'+c_{caseL}}_{k+k'+c_{caseL}} \begin{array}{l}\mathsf{case}\ e\ \mathsf{of}\ \mathsf{nil} \to e_1 \\ \mid h :: tl \to e_2\end{array} \rightsquigarrow \begin{array}{l}\mathsf{case}\ e^*\ \mathsf{of}\ \mathsf{nil} \to e_1^* \\ \mid h ::_{NC} tl \to e_2^* \\ \mid h ::_C tl \to e_2^*\end{array} : A'}\ \textbf{e-u-caseL}$$

Figure 44: RelCost Core unary embedding rules (Part 1)

$\boxed{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e \rightsquigarrow e^* : A}$ Expression e is embedded into $e*$ with the unary type A and the minimum and maximum execution costs k and t, respectively.

$$\frac{i :: S, \Delta;\Phi_a;\Omega \vdash^{t}_{k} e \rightsquigarrow e^* : A \qquad i \notin FIV(\Phi_a;\Omega)}{\Delta;\Phi_a;\Omega \vdash^{0}_{0} \Lambda.e \rightsquigarrow \Lambda i.e^* : \forall i \overset{exec(k,t)}{::} S.\, A}\ \textbf{e-u-iLam}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e \rightsquigarrow e^* : \forall i \overset{exec(k',t')}{::} S.\, A \qquad \Delta \vdash I : S}{\Delta;\Phi_a;\Omega \vdash^{t+t'[I/i]+c_{iApp}}_{k+k'[I/i]+c_{iApp}} e[] \rightsquigarrow e^*[I] : A\{I/i\}}\ \textbf{e-u-iApp}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e \rightsquigarrow e^* : A\{I/i\} \qquad \Delta \vdash I :: S}{\Delta;\Phi_a;\Omega \vdash^{t}_{k} \text{pack } e \rightsquigarrow \text{pack } e^* \text{ with } I : \exists i{::}S.\, A}\ \textbf{e-u-pack}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Omega \vdash^{t_1}_{k_1} e_1 \rightsquigarrow {e_1}^* : \exists i{::}S.\, A_1 \\ i :: S, \Delta;\Phi_a;x : A_1, \Omega \vdash^{t_2}_{k_2} e_2 \rightsquigarrow {e_2}^* : A_2 \\ i \notin FV(\Phi_a;\Omega, A_2, k_2, t_2)\end{array}}{\Delta;\Phi_a;\Omega \vdash^{t_1+t_2}_{k_1+k_2+c_{unp}} \text{unpack } e_1 \text{ as } x \text{ in } e_2 \rightsquigarrow \text{unpack } e_1^* \text{ as } (x,i) \text{ in } e_2^* : A_2}\ \textbf{e-u-unpack}$$

$$\frac{\Upsilon(\zeta) = A_1 \xrightarrow{exec(k,t)} A_2 \qquad \Delta;\Phi_a;\Omega \vdash^{t'}_{k'} e \rightsquigarrow e^* : A_1}{\Delta;\Phi_a;\Omega \vdash^{t+t'+c_{primapp}}_{k+k'+c_{primapp}} \zeta\ e \rightsquigarrow \zeta\ e^* : A_2}\ \textbf{e-u-primapp}$$

$$\frac{\Delta;\Phi_a \models C \qquad \Delta;\Phi_a \wedge C;\Omega \vdash^{t}_{k} e \rightsquigarrow e^* : A}{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e \rightsquigarrow e^* : C \,\&\, A}\ \textbf{e-u-andI}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Omega \vdash^{t_1}_{k_1} e_1 \rightsquigarrow {e_1}^* : C \,\&\, A_1 \\ \Delta;\Phi_a \wedge C;x : A_1, \Omega \vdash^{t_2}_{k_u} e_2 \rightsquigarrow {e_2}^* : A_2\end{array}}{\Delta;\Phi_a;\Gamma \vdash^{t_1+t_2}_{k_1+k_2} \text{clet } e_1 \text{ as } x \text{ in } e_2 \rightsquigarrow \text{clet } e_1^* \text{ as } x \text{ in } e_2^* : A_2}\ \textbf{e-u-c-andE}$$

$$\frac{\Delta;\Phi_a \wedge C;\Omega \vdash^{t}_{k} e \rightsquigarrow e^* : A}{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e \rightsquigarrow e^* : C \supset A}\ \textbf{e-u-c-impI}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e \rightsquigarrow e^* : C \supset A \qquad \Delta;\Phi_a \models C}{\Delta;\Phi_a;\Omega \vdash^{t}_{k} \text{celim}_{\supset}\ e \rightsquigarrow \text{celim}_{\supset}\ e^* : A}\ \textbf{e-u-c-implE}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{t_1}_{k_1} e_1 \rightsquigarrow {e_1}^* : A_1 \qquad \Delta;\Phi_a;x : A_1, \Omega \vdash^{t_2}_{k_2} e_2 \rightsquigarrow {e_2}^* : A_2}{\Delta;\Phi_a;\Omega \vdash^{t_1+t_2+c_{let}}_{k_1+k_2+c_{let}} \text{let } x = e_1 \text{ in } e_2 \rightsquigarrow \text{let } x = e_1^* \text{ in } e_2^* : A_2}\ \textbf{e-u-let}$$

$$\frac{\begin{array}{c}\Delta;C \wedge \Phi_a;\Omega \vdash^{t}_{k} e \rightsquigarrow e^* : A \\ \Delta;\neg C \wedge \Phi_a;\Omega \vdash^{t}_{k} e \rightsquigarrow e^{**} : A \qquad \Delta \vdash C\ \texttt{wf}\end{array}}{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e \rightsquigarrow \texttt{split}\ (e^*, e^{**}) \text{ with } C : A}\ \textbf{e-u-split}$$

$$\frac{\Delta;\Phi_a \models \bot \qquad \Delta;\Phi_a \vdash \Omega\ \texttt{wf}}{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e \rightsquigarrow \texttt{contra}\ e : \tau}\ \textbf{e-u-contra}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Omega \vdash^{t}_{k} e \rightsquigarrow e^* : A \qquad \Delta;\Phi_a \models^{A} A \sqsubseteq A' \\ \Delta;\Phi_a \models k' \leqslant k \qquad \Delta;\Phi_a \models t \leqslant t'\end{array}}{\Delta;\Phi_a;\Omega \vdash^{t'}_{k'} e \rightsquigarrow e^* : A'}\ \textbf{e-u-}\sqsubseteq$$

Figure 45: RelCost Core unary embedding rules (Part 2)

$\boxed{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \rightsquigarrow e_1^* \ominus e_2^* \lesssim t : \tau}$ Expressions $e_1 \ominus e_2$ are embedded into $e_1^* \ominus e_1^*$ with the relational type τ and the relational cost t.

$$\frac{\Delta;\Phi_a;|\Gamma|_1 \vdash_{k_1}^{t_1} e_1 \rightsquigarrow e_1^* : A_1 \qquad \Delta;\Phi_a;|\Gamma|_2 \vdash_{k_2}^{t_2} e_2 \rightsquigarrow e_2^* : A_2 \qquad E = \texttt{switch}\ e_1^* \qquad E' = \texttt{switch}\ e_2^*}{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \rightsquigarrow E \ominus E' \lesssim t_1 - k_2 : U(A_1, A_2)}\ \textbf{e-switch}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash e \ominus e \rightsquigarrow e^* \ominus e^* \lesssim t : \tau \qquad \forall x_i \in \mathrm{dom}(\Gamma),\ \ e_i = \texttt{coerce}_{\Gamma(x_i),\Box\Gamma(x_i)} \qquad e' = \texttt{let}\ \overline{y_i = e_i\ x_i}\ \texttt{in}\ \texttt{NC}\ e^*[y_i/x_i]}{\Delta;\Phi_a;\Gamma,\Gamma' \vdash e \ominus e \rightsquigarrow e' \ominus e' \lesssim 0 : \Box\tau}\ \textbf{e-nochange}$$

$$\frac{\Delta;C \wedge \Phi_a;\Gamma \vdash e_1 \ominus e_2 \rightsquigarrow {e_1}^* \ominus {e_2}^* \lesssim t : \tau \qquad \Delta;\neg C \wedge \Phi_a;\Gamma \vdash e_1 \ominus e_2 \rightsquigarrow e_1^{**} \ominus e_2^{**} \lesssim t : \tau \qquad \Delta \vdash C\ \texttt{wf} \qquad E = \texttt{split}\ (e_1^*, e_1^{**})\ \text{with}\ C \qquad E' = \texttt{split}\ (e_2^*, e_2^{**})\ \text{with}\ C}{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \rightsquigarrow E \ominus E' \lesssim t : \tau}\ \textbf{e-r-split}$$

$$\frac{\Delta;\Phi_a \models \bot \qquad \Delta;\Phi_a \vdash \Gamma\ \texttt{wf}}{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \rightsquigarrow \texttt{contra}\ e_1 \ominus \texttt{contra}\ e_2 \lesssim t : \tau}\ \textbf{e-r-contra}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \rightsquigarrow {e_1}^* \ominus {e_2}^* \lesssim t : \tau \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad e' = \texttt{coerce}_{\tau,\tau'} \qquad \Delta;\Phi_a \models t \leqslant t'}{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \rightsquigarrow e'\ e_1^* \ominus e'\ e_2^* \lesssim t' : \tau'}\ \textbf{e-r-}\sqsubseteq$$

$$\frac{}{\Delta;\Phi_a;\Gamma \vdash n \ominus n \rightsquigarrow n \ominus n \lesssim 0 : \mathrm{int}_r}\ \textbf{e-r-const}$$

$$\frac{\Gamma(x) = \tau}{\Delta;\Phi_a;\Gamma \vdash x \ominus x \rightsquigarrow x \ominus x \lesssim 0 : \tau}\ \textbf{e-r-var}$$

$$\frac{}{\Delta;\Phi_a;\Gamma \vdash () \ominus () \rightsquigarrow () \ominus () \lesssim 0 : \mathrm{unit}_r}\ \textbf{e-r-unit}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \rightsquigarrow {e_1}^* \ominus {e_2}^* \lesssim t : \tau_1 \qquad \Delta;\Phi_a \vdash \tau_2\ \texttt{wf}}{\Delta;\Phi_a;\Gamma \vdash \mathrm{inl}\ e_1 \ominus \mathrm{inl}\ e_2 \rightsquigarrow \mathrm{inl}\ {e_1}^* \ominus \mathrm{inl}\ {e_2}^* \lesssim t : \tau_1 + \tau_2}\ \textbf{e-r-inl}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \rightsquigarrow {e_1}^* \ominus {e_2}^* \lesssim t : \tau_2 \qquad \Delta;\Phi_a \vdash \tau_1\ \texttt{wf}}{\Delta;\Phi_a;\Gamma \vdash \mathrm{inr}\ e_1 \ominus \mathrm{inl}\ e_2 \rightsquigarrow \mathrm{inr}\ {e_1}^* \ominus \mathrm{inl}\ {e_2}^* \lesssim t : \tau_1 + \tau_2}\ \textbf{e-r-inr}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash e \ominus e' \rightsquigarrow e^* \ominus e'^* \lesssim t : \tau_1 + \tau_2 \qquad \Delta;\Phi_a;x : \tau_1,\Gamma \vdash e_1 \ominus e_1' \rightsquigarrow {e_1}^* \ominus {e_1'}^* \lesssim t' : \tau \qquad \Delta;\Phi_a;y : \tau_2,\Gamma \vdash e_2 \ominus e_2' \rightsquigarrow {e_2}^* \ominus {e_2'}^* \lesssim t' : \tau \qquad E = \ \mathrm{case}\ (e^*, x.e_1^*, y.e_2^*) \qquad E' = \ \mathrm{case}\ (e', x.e_1'^*, y.e_2'^*)}{\Delta;\Phi_a;\Gamma \vdash \ \mathrm{case}\ (e, x.e_1, y.e_2) \ominus \ \mathrm{case}\ (e', x.e_1', y.e_2') \rightsquigarrow E \ominus E' \lesssim t + t' : \tau}\ \textbf{e-r-case}$$

$$\frac{\Delta;\Phi_a \vdash \tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2\ \texttt{wf} \qquad \Delta;\Phi_a;x : \tau_1, f : \tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2,\Gamma \vdash e_1 \ominus e_2 \rightsquigarrow {e_1}^* \ominus {e_2}^* \lesssim t : \tau_2}{\Delta;\Phi_a;\Gamma \vdash \mathrm{fix}\ f(x).e_1 \ominus \mathrm{fix}\ f(x).e_2 \rightsquigarrow \mathrm{fix}\ f(x).{e_1}^* \ominus \mathrm{fix}\ f(x).{e_2}^* \lesssim 0 : \tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2}\ \textbf{e-r-fix}$$

Figure 46: RelCost Core relational embedding rules (Part 1)

$\boxed{\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \rightsquigarrow e_1^* \ominus e_2^* \lesssim t : \tau}$ Expressions $e_1 \ominus e_2$ are embedded into $e_1^* \ominus e_1^*$ with the relational type τ and the relational cost t.

$$\frac{\begin{array}{c}\Delta; \Phi_a \vdash \tau_1 \xrightarrow{\text{diff}(t)} \tau_2 \ \texttt{wf} \\ \Delta; \Phi_a; x : \tau_1, f : \square(\tau_1 \xrightarrow{\text{diff}(t)} \tau_2), \Gamma \vdash e \ominus e \rightsquigarrow e^* \ominus e^* \lesssim t : \tau_2 \\ \forall x_i \in \text{dom}(\Gamma),\ \ e_i = \texttt{coerce}_{\Gamma(x_i), \square\Gamma(x_i)} \\ e^{**} = \text{let } \overline{y_i = e_i\ x_i} \text{ in } \text{fix}_{NC}\ f(x).e^*[\overline{y_i/x_i}]\end{array}}{\Delta; \Phi_a; \Gamma \vdash \text{fix } f(x).e \ominus \text{fix } f(x).e \rightsquigarrow e^{**} \ominus e^{**} \lesssim 0 : \square(\tau_1 \xrightarrow{\text{diff}(t)} \tau_2)}\ \textbf{e-r-fixNC}$$

$$\frac{\begin{array}{c}\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_1' \rightsquigarrow {e_1}^* \ominus {e_1'}^* \lesssim t_1 : \tau_1 \xrightarrow{\text{diff}(t)} \tau_2 \\ \Delta; \Phi_a; \Gamma \vdash e_2 \ominus e_2' \rightsquigarrow {e_2}^* \ominus {e_2'}^* \lesssim t_2 : \tau_1\end{array}}{\Delta; \Phi_a; \Gamma \vdash e_1\ e_2 \ominus e_1'\ e_2' \rightsquigarrow e_1^*\ e_2^* \ominus {e_1'}^*\ {e_2'}^* \lesssim t_1 + t_2 + t : \tau_2}\ \textbf{e-r-app}$$

$$\frac{\begin{array}{c}\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_1' \rightsquigarrow {e_1}^* \ominus {e_1'}^* \lesssim t_1 : \tau_1 \\ \Delta; \Phi_a; \Gamma \vdash e_2 \ominus e_2' \rightsquigarrow {e_2}^* \ominus {e_2'}^* \lesssim t_2 : \tau_2\end{array}}{\Delta; \Phi_a; \Gamma \vdash \langle e_1, e_2\rangle \ominus \langle e_1', e_2'\rangle \rightsquigarrow \langle e_1^*, e_2^*\rangle \ominus \langle {e_1'}^*, {e_2'}^*\rangle \lesssim t_1 + t_2 : \tau_1 \times \tau_2}\ \textbf{e-r-prod}$$

$$\frac{\Delta; \Phi_a; \Gamma \vdash e \ominus e' \rightsquigarrow e^* \ominus {e'}^* \lesssim t : \tau_1 \times \tau_2}{\Delta; \Phi_a; \Gamma \vdash \pi_i(e) \ominus \pi_i(e') \rightsquigarrow \pi_i(e^*) \ominus \pi_i({e'}^*) \lesssim t : \tau_i}\ \textbf{e-r-proj}_i$$

$$\frac{\Delta; \Phi_a \vdash \tau\ \texttt{wf}}{\Delta; \Phi_a; \Gamma \vdash \text{nil} \ominus \text{nil} \rightsquigarrow \text{nil} \ominus \text{nil} \lesssim 0 : \text{list}[0]^\alpha\ \tau}\ \textbf{e-r-nil}$$

$$\frac{\begin{array}{c}\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_1' \rightsquigarrow {e_1}^* \ominus {e_1'}^* \lesssim t_1 : \tau \\ \Delta; \Phi_a; \Gamma \vdash e_2 \ominus e_2' \rightsquigarrow {e_2}^* \ominus {e_2'}^* \lesssim t_2 : \text{list}[n]^\alpha\ \tau \\ E = \text{cons}_C(e_1^*, e_2^*) \qquad E' = \text{cons}_C({e_1'}^*, {e_2'}^*)\end{array}}{\Delta; \Phi_a; \Gamma \vdash \text{cons}(e_1, e_2) \ominus \text{cons}(e_1', e_2') \rightsquigarrow E \ominus E' \lesssim t_1 + t_2 : \text{list}[n+1]^{\alpha+1}\ \tau}\ \textbf{e-r-cons1}$$

$$\frac{\begin{array}{c}\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_1' \rightsquigarrow {e_1}^* \ominus {e_1'}^* \lesssim t_1 : \square\tau \\ \Delta; \Phi_a; \Gamma \vdash e_2 \ominus e_2' \rightsquigarrow {e_2}^* \ominus {e_2'}^* \lesssim t_2 : \text{list}[n]^\alpha\ \tau \\ E = \text{cons}_{NC}(e_1^*, e_2^*) \qquad E' = \text{cons}_{NC}({e_1'}^*, {e_2'}^*)\end{array}}{\Delta; \Phi_a; \Gamma \vdash \text{cons}(e_1, e_2) \ominus \text{cons}(e_1', e_2') \rightsquigarrow E \ominus E' \lesssim t_1 + t_2 : \text{list}[n+1]^\alpha\ \tau}\ \textbf{e-r-cons2}$$

$$\frac{\begin{array}{c}\Delta; \Phi_a; \Gamma \vdash e \ominus e' \rightsquigarrow e^* \ominus {e'}^* \lesssim t : \text{list}[n]^\alpha\ \tau \\ \Delta; \Phi_a \wedge n = 0; \Gamma \vdash e_1 \ominus e_1' \rightsquigarrow {e_1}^* \ominus {e_1'}^* \lesssim t' : \tau' \\ \Phi_a' = \Phi_a \wedge n = i + 1 \\ i, \Delta; \Phi_a'; h : \square\tau, tl : \text{list}[i]^\alpha\ \tau, \Gamma \vdash e_2 \ominus e_2' \rightsquigarrow {e_2}^* \ominus {e_2'}^* \lesssim t' : \tau' \\ \Phi_a'' = \Phi_a \wedge n = i + 1 \wedge \alpha = \beta + 1 \\ i, \beta, \Delta; \Phi_a''; h : \tau, tl : \text{list}[i]^\beta\ \tau, \Gamma \vdash e_2 \ominus e_2' \rightsquigarrow e_3^* \ominus {e_3'}^* \lesssim t' : \tau' \\ \begin{array}{cc}\text{case } e^* \text{ of nil } \to e_1^* & \text{case } {e'}^* \text{ of nil } \to {e_1'}^* \\ E = \mid h ::_{NC} tl \to e_2^* & E' = \mid h ::_{NC} tl \to {e_2'}^* \\ \quad \mid h ::_C tl \to e_3^* & \quad \mid h ::_C tl \to {e_3'}^*\end{array}\end{array}}{\Delta; \Phi_a; \Gamma \vdash \begin{array}{l}\text{case } e \text{ of nil } \to e_1 \\ \mid h :: tl \to e_2\end{array} \ominus \begin{array}{l}\text{case } e \text{ of nil } \to e_1' \\ \mid h :: tl \to e_2'\end{array} \rightsquigarrow E \ominus E' \lesssim t + t' : \tau'}\ \textbf{e-r-caseL}$$

Figure 47: RelCost Core relational embedding rules (Part 2)

$\boxed{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \rightsquigarrow e_1^* \ominus e_2^* \lesssim t : \tau}$ Expressions $e_1 \ominus e_2$ are embedded into $e_1^* \ominus e_1^*$ with the relational type τ and the relational cost t.

$$\frac{i :: S, \Delta;\Phi_a;\Gamma \vdash e \ominus e' \rightsquigarrow e^* \ominus e'^* \lesssim t : \tau \qquad i \notin FIV(\Phi_a;\Gamma)}{\Delta;\Phi_a;\Gamma \vdash \Lambda.e \ominus \Lambda.e' \rightsquigarrow \Lambda i.e^* \ominus \Lambda i.e'^* \lesssim 0 : \forall i \overset{diff(t)}{::} S.\tau}\ \textbf{e-r-iLam}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash e \ominus e' \rightsquigarrow e^* \ominus e'^* \lesssim t : \forall i \overset{diff(t')}{::} S.\tau \qquad \Delta \vdash I : S}{\Delta;\Phi_a;\Gamma \vdash e[] \ominus e'[] \rightsquigarrow e^*[I] \ominus e'^*[I] \lesssim t + t'[I/i] : \tau\{I/i\}}\ \textbf{e-r-iApp}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash e \ominus e' \rightsquigarrow e^* \ominus e'^* \lesssim t : \tau\{I/i\} \\ \Delta \vdash I :: S \qquad E = pack\ e^*\ with\ I \qquad E' = pack\ e'^*\ with\ I\end{array}}{\Delta;\Phi_a;\Gamma \vdash pack\ e \ominus pack\ e' \rightsquigarrow E \ominus E' \lesssim t : \exists i::S.\tau}\ \textbf{e-r-pack}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_1' \rightsquigarrow {e_1}^* \ominus e_1'^* \lesssim t_1 : \exists i::S.\tau_1 \\ i :: S, \Delta;\Phi_a;x : \tau_1,\Gamma \vdash e_2 \ominus e_2' \rightsquigarrow {e_2}^* \ominus e_2'^* \lesssim t_2 : \tau_2 \\ i \notin FV(\Phi_a;\Gamma,\tau_2,t_2) \\ E = unpack\ e_1^*\ as\ (x,i)\ in\ e_2^* \qquad E' = unpack\ e_1'^*\ as\ (x,i)\ in\ e_2'^*\end{array}}{\Delta;\Phi_a;\Gamma \vdash unpack\ e_1\ as\ x\ in\ e_2 \ominus unpack\ e_1'\ as\ x\ in\ e_2' \rightsquigarrow E \ominus E' \lesssim t_1 + t_2 : \tau_2}\ \textbf{e-r-unpack}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_1' \rightsquigarrow {e_1}^* \ominus e_1'^* \lesssim t_1 : \tau_1 \\ \Delta;\Phi_a;x : \tau_1,\Gamma \vdash e_2 \ominus e_2' \rightsquigarrow {e_2}^* \ominus e_2'^* \lesssim t_2 : \tau_2 \\ E = let\ x = e_1^*\ in\ e_2^* \qquad E' = let\ x = e_1'^*\ in\ e_2'^*\end{array}}{\Delta;\Phi_a;\Gamma \vdash let\ x = e_1\ in\ e_2 \ominus let\ x = e_1'\ in\ e_2' \rightsquigarrow E \ominus E' \lesssim t_1 + t_2 : \tau_2}\ \textbf{e-r-let}$$

$$\frac{\Upsilon(\zeta) = \tau_1 \xrightarrow{diff(t)} \tau_2 \qquad \Delta;\Phi_a;\Gamma \vdash e \ominus e' \rightsquigarrow e^* \ominus e'^* \lesssim t' : \tau_1}{\Delta;\Phi_a;\Gamma \vdash \zeta\ e \ominus \zeta\ e' \rightsquigarrow \zeta\ e^* \ominus \zeta\ e'^* \lesssim t + t' : \tau_2}\ \textbf{e-r-primapp}$$

$$\frac{\Delta;\Phi_a \models C \qquad \Delta;\Phi_a \wedge C;\Gamma \vdash e \ominus e' \rightsquigarrow e^* \ominus e'^* \lesssim t : \tau}{\Delta;\Phi_a;\Gamma \vdash e \ominus e' \rightsquigarrow e^* \ominus e'^* \lesssim t : C\ \&\ \tau}\ \textbf{e-r-andI}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_1' \rightsquigarrow {e_1}^* \ominus e_1'^* \lesssim t_1 : C\ \&\ \tau_1 \\ \Delta;\Phi_a \wedge C;x : \tau_1,\Gamma \vdash e_2 \ominus e_2' \rightsquigarrow {e_2}^* \ominus e_2'^* \lesssim t_2 : \tau_2 \\ E = clet\ e_1^*\ as\ x\ in\ e_2^* \qquad E' = clet\ e_1'^*\ as\ x\ in\ e_2'^*\end{array}}{\Delta;\Phi_a;\Gamma \vdash clet\ e_1\ as\ x\ in\ e_2 \ominus clet\ e_1'\ as\ x\ in\ e_2' \rightsquigarrow E \ominus E' \lesssim t_1 + t_2 : \tau_2}\ \textbf{e-r-c-andE}$$

$$\frac{\Delta;\Phi_a \wedge C;\Gamma \vdash e \ominus e' \rightsquigarrow e^* \ominus e'^* \lesssim t : \tau}{\Delta;\Phi_a;\Gamma \vdash e \ominus e' \rightsquigarrow e^* \ominus e'^* \lesssim t : C \supset \tau}\ \textbf{e-r-c-impI}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash e \ominus e' \rightsquigarrow e^* \ominus e'^* \lesssim t : C \supset \tau \qquad \Delta;\Phi_a \models C}{\Delta;\Phi_a;\Gamma \vdash celim_\supset\ e \ominus celim_\supset\ e' \rightsquigarrow celim_\supset\ e^* \ominus celim_\supset\ e'^* \lesssim t : \tau}\ \textbf{e-r-c-implE}$$

Figure 48: RelCost Core relational embedding rules (Part 3)

$\boxed{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \rightsquigarrow e_1^* \ominus e_2^* \lesssim t : \tau}$ Expressions $e_1 \ominus e_2$ are embedded into $e_1^* \ominus e_1^*$ with the relational type τ and the relational cost t.

$$\frac{\Delta;\Phi_a;|\Gamma|_1 \vdash^{t_1}_{k_1} e_1 \rightsquigarrow {e_1}^* : A_1 \qquad \Delta;\Phi_a;x : U\,A_1,\Gamma \vdash e_2 \ominus e \rightsquigarrow {e_2}^* \ominus e^* \lesssim t_2 : \tau_2 \qquad E = \text{let } x = e_1^* \text{ in } e_2^*}{\Delta;\Phi_a;\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 \ominus e \rightsquigarrow E \ominus e^* \lesssim t_1 + t_2 + c_{let} : \tau_2}\ \textbf{e-r-let-e}$$

$$\frac{\Delta;\Phi_a;|\Gamma|_2 \vdash^{t_1}_{k_1} e_1 \rightsquigarrow {e_1}^* : A_1 \qquad \Delta;\Phi_a;x : U\,A_1,\Gamma \vdash e \ominus e_2 \rightsquigarrow e^* \ominus {e_2}^* \lesssim t_2 : \tau_2 \qquad E' = \text{let } x = e_1^* \text{ in } e_2^*}{\Delta;\Phi_a;\Gamma \vdash e \ominus \text{let } x = e_1 \text{ in } e_2 \rightsquigarrow e^* \ominus E' \lesssim t_2 - k_1 - c_{let} : \tau_2}\ \textbf{e-r-e-r-let}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;|\Gamma|_1 \vdash^{t}_{\_} e \rightsquigarrow e^* : A_1 + A_2 \\ \Delta;\Phi_a;x : U\,A_1,\Gamma \vdash e_1 \ominus e' \rightsquigarrow {e_1}^* \ominus {e'}^* \lesssim t' : \tau \\ \Delta;\Phi_a;y : U\,A_2,\Gamma \vdash e_2 \ominus e' \rightsquigarrow {e_2}^* \ominus {e'}^* \lesssim t' : \tau \\ E = \ \text{case}\ (e^*, x.e_1^*, y.e_2^*)\end{array}}{\Delta;\Phi_a;\Gamma \vdash \ \text{case}\ (e, x.e_1, y.e_2) \ominus e' \rightsquigarrow E \ominus {e'}^* \lesssim t' + t + c_{case} : \tau}\ \textbf{e-r-case-r-e}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;|\Gamma|_2 \vdash_{\overline{k'}} e' \rightsquigarrow {e'}^* : A_1 + A_2 \\ \Delta;\Phi_a;x : U\,A_1,\Gamma \vdash e \ominus e_1' \rightsquigarrow e^* \ominus {e_1'}^* \lesssim t : \tau \\ \Delta;\Phi_a;y : U\,A_2,\Gamma \vdash e \ominus e_2' \rightsquigarrow e^* \ominus {e_2'}^* \lesssim t : \tau \\ E' = \ \text{case}\ (e'^*, x.e_1'^*, y.e_2'^*)\end{array}}{\Delta;\Phi_a;\Gamma \vdash \ \text{case}\ (e', x.e_1', y.e_2') \ominus e \rightsquigarrow e^* \ominus E' \lesssim t - k' - c_{case} : \tau}\ \textbf{e-r-e-r-case}$$

Figure 49: RelCost Core relational embedding rules (Part 4)

13

ALGORITHMIC (BIDIRECTIONAL) TYPE SYSTEM

▶ SYNOPSIS This chapter presents an algorithmic type system for RelCost Core.In Section 13.4, we also discuss how a similar system can be designed for DuCostIt.

Our algorithmic type system relies on bidirectionality, which allows us to type check with very few type annotations. The bidirectional system is called BiRelCost. Like all other bidirectional systems [91, 105, 106], the goal of BiRelCost is to eliminate the nondeterminism inherent in typing Curry-style unannotated terms (e.g., the term $\lambda x.x$ can be given the type $\tau \xrightarrow{\mathrm{diff}(k)} \tau$ for any τ and k) using minimal type annotations. This is done by typing every expression construct in one of the two modes: *synthesis/inference* mode, where the type is inferred or *checking* mode, where a provided type is checked.

The following three aspects of BiRelCost differ from existing bidirectional typecheckers:

- Since BiRelCost is a type and effect system, it must infer (check) not only a type, but also a cost (effect). In general, the cost follows the same mode as the type: If for an expression, the type is inferred (checked), then so is the cost. Still, as an alternative design, we sketch a formalization in which the costs are inferred in the checking mode with several additional constraints. We comment on this in Section 13.3.
- Since BiRelCost is relational, it must check (infer) not only a single expression, but also a pair of expressions in the relational typing.
- BiRelCost heavily relies on unary and relational type refinements. To handle these, our type system generates arithmetic constraints that stipulate relations between, e.g., sizes of various lists and the costs of various subexpressions. This is similar to the treatment of refinements in DML [105, 106], but we additionally handle relational refinements and both unary and relational costs.

13.1 ALGORITHMIC TYPECHECKING AS CONSTRAINT SATISFACTION

To develop the bidirectional type system, we add some syntactic classes and extend the existing ones.

meta variables	M	$::=$	$i, n, m, \cdots$
index terms	I, k, t	$::=$	$\cdots \mid M$
meta contexts	ψ	$::=$	$\emptyset \mid \psi, M : S$
meta substitutions	θ	$::=$	$[] \mid \theta[M \mapsto I]$
constraints	C	$::=$	$\cdots \mid \forall i : S.C \mid \exists i : S.C$
expressions	e	$::=$	$\cdots \mid (e : A, k, t) \mid (e : \tau, t)$

One class of note is the meta variables i, n, m, etc. Also known as existential variables, meta variables represent unknown index terms that appear in types and costs. These meta variables are resolved by constraint solving.

Since RelCost Core has two typing judgments—a unary judgment and a relational judgment—BiRelCost has four mutually recursive algorithmic typing judgments: one each to synthesize and check a unary type, and one each to synthesize and check a relational type. The relational *checking* judgment

$$\Delta; \psi_a; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \downarrow \tau, t \Rightarrow \Phi$$

means that, *assuming that the constraint Φ holds*, e_1 and e_2 check against the relational input type τ and the relative cost t under the assumption Φ_a. All index and meta variables must occur in Δ and ψ_a. When the judgment is read operationally or is implemented, the constraint Φ is an output; all other components are inputs. As a convention, we write all outputs in red and all inputs in black. The relational *inference* judgment

$$\Delta; \psi_a; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \uparrow \tau \Rightarrow [\psi], t, \Phi$$

means that, if Φ holds, then it can be inferred that the relational type of e_1 and e_2 is τ and their relative cost is t under the assumption $\exists \psi.\Phi_a$. Here, Φ, τ, ψ and t are all outputs. The meta variable context ψ tracks newly generated cost variables that may appear in the inferred type or cost. [38] Its significance will be explained below.

[38] τ, t, Φ *can contain variables from* ψ *as well as* Δ.

Intuitively, the checking mode is used when the surrounding outer context determines the type of the expression. This happens at all introduction forms and in case-like elimination forms. The inference mode is used when there is no outer restriction on the type of an expression. This happens for the principal term in all elimination forms.

The unary checking and inference judgments, $\Delta; \psi_a; \Phi_a; \Omega \vdash e \downarrow A, k, t \Rightarrow \Phi$ and $\Delta; \psi_a; \Phi_a; \Omega \vdash e \uparrow A \Rightarrow [\psi], k, t, \Phi$, respectively, are to be understood similarly.

13.1.1 *Algorithmic typing rules*

We first explain the main principles behind the bidirectional typing rules and then discuss selected relational rules. We focus our attention on relational rules that are shown in Figures 50 to 52. Algorithmic rules for unary typing are shown at the end of this chapter in Figures 56 to 58. While some of these principles are standard, those related to cost are new. We also highlight other BiRelCost-specific principles.

- Types and costs of variables and elimination forms are inferred (e.g., rules **alg-r-var-↑**, **alg-r-app-↑**) whereas types and costs of introduction forms are checked (e.g., rules **alg-r-fix-↓**, **alg-r-consC-↓**). There are a few exceptions to this principle: the type of the branches in case analyses or the continuation in let-bindings is checked.

- (Specific to BiRelCost) If an expression consists of a subexpression that must be checked against a type, we generate a fresh meta variable for the subexpression's cost. This is necessary since at the point we reach the subexpression, we don't know what cost to check against. When we check the subexpression, constraints that determine this meta variable are generated. If the whole expression is in checking mode, we existentially quantify over the freshly generated variable in the constraint (e.g., rule **alg-r-consC-↓**). If the whole expression is in inference mode, we simply add the meta variable to the inferred cost and append it to ψ (e.g. rule **alg-r-app-↑**).

- In checking mode, if no other checking rule matches the given expression, then the rule **alg-r-↑↓** allows switching to inference mode in the premise. The requirement is that the inferred type must be equivalent to the checked type and the inferred cost must be no greater than the checked cost. In the algorithmic system, we have an algorithmic type equivalence relation $\models \tau \equiv \tau' \Rightarrow \Phi$ whose rules are shown in Figure 54. The meaning of the judgment is that if Φ holds, then $\tau \equiv \tau'$. Like other bidirectional systems, this is the only place where type equivalence (RelCost Core's reduct of subtyping) is used.

- In inference mode, it is permissible to switch to checking mode when an expression's type and cost have been explicitly annotated by the programmer (rule **alg-r-anno-↑**). It can be shown that it suffices to annotate only at explicit β-redexes (although there is no prohibition on annotating at other places).

- (Specific to BiRelCost) The algorithmic version of the rule **c-nochange**, called **alg-r-nochange-↓**, applies in checking mode since it introduces the type $\Box\,\tau$. The algorithmic version of the rule **c-der**, called **alg-r-der-↑**, applies in inference mode since it eliminates the type $\Box\,\tau$.

Switching to unary mode in BiRelCost is possible both in checking (rule **alg-r-switch-↓**) and inference (rule **alg-r-switch-↑**) modes. This is because often we switch to the unary mode for elimination forms where the eliminated type is $U\,A$ for some A. Hence, since elimination forms can be typed both in inference mode (e.g. rule **alg-r-app-↑**) as well as checking mode (e.g. rule **alg-r-caseL-↓**), to eliminate unnecessary annotations, BiRelCost supports two modes for typing `switch` expressions.

Variables are typed in inference mode (rule **alg-r-var-↑**) where the output type is synthesized from the type environment Γ, which is given. Functions are typed in checking mode (rule **alg-r-fix-↓**) by checking the related function bodies in the checking mode against the relative cost of the function bodies. Since functions are values, we add the additional constraint that the total cost of the functions, t, is equal to zero.

Dually, function applications are typed in inference mode (rule **alg-r-app-↑**). We first infer the type and the cost of the related functions and then use the (inferred) argument type to switch to the checking mode for the related argument expressions. Since we do not know the cost with which to check the arguments, we also generate a fresh cost variable t_2 to check the arguments. Finally, the result type is the return type of the inferred function type, and the total cost $t_1 + t_2 + t_e$, i. e., the sum of the inferred cost for the functions, the yet-unknown cost of the arguments and the relative cost of the function bodies. Note that since all the newly generated variables, i.e., ψ and t_2, may occur in the resulting type and the cost, inference mode passes them to the surrounding context. In the **alg-r-app-↑** rule, if e_1 is a fixpoint, its type (and cost) must be given by explicitly annotating it with the construct $(e : \tau, k)$. The rule **alg-r-anno-↑** allows us to take advantage of such annotations by switching from inference to checking mode.

The list constructors are typed in checking mode. We explain only the rule **alg-r-consC-↓** for typing two non-empty lists with type $\mathrm{list}[n]^{\alpha}\,\tau$

$\boxed{\Delta;\psi_a;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \downarrow \tau, t \Rightarrow \Phi}$ $e_1 \ominus e_2$ checks against the input type τ and the difference cost t. Finally, it generates the constraint Φ.

$\boxed{\Delta;\psi_a;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \uparrow \tau \Rightarrow [\psi], t, \Phi}$ $e_1 \ominus e_2$ synthesizes the output type τ and the relative cost t where all the newly generated existential variables are defined in ψ. Finally, it generates the constraint Φ.

$$\frac{}{\Delta;\psi_a;\Phi_a;\Gamma \vdash n \ominus n \uparrow \mathrm{int}_r \Rightarrow [.], 0, \top}\ \textbf{alg-r-n-}\uparrow$$

$$\frac{\Gamma(x) = \tau}{\Delta;\psi_a;\Phi_a;\Gamma \vdash x \ominus x \uparrow \tau \Rightarrow [.], 0, \top}\ \textbf{alg-r-var-}\uparrow$$

$$\frac{}{\Delta;\psi_a;\Phi_a;\Gamma \vdash () \ominus () \uparrow \mathrm{unit}_r \Rightarrow [.], 0, \top}\ \textbf{alg-r-unit-}\uparrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \downarrow \tau_1, t \Rightarrow \Phi \qquad \Delta,\psi_a;\Phi_a \vdash \tau_2\ \mathtt{wf}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathrm{inl}\ e_1 \ominus \mathrm{inl}\ e_2 \downarrow \tau_1 + \tau_2, t \Rightarrow \Phi}\ \textbf{alg-r-inl-}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \downarrow \tau_2, t \Rightarrow \Phi \qquad \Delta,\psi_a;\Phi_a \vdash \tau_1\ \mathtt{wf}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathrm{inr}\ e_1 \ominus \mathrm{inr}\ e_2 \downarrow \tau_1 + \tau_2, t \Rightarrow \Phi}\ \textbf{alg-r-inr-}\downarrow$$

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Gamma \vdash e \ominus e' \uparrow \tau_1 + \tau_2 \Rightarrow [\psi], t_e, \Phi_1 \\ t' \in \mathrm{fresh}(\mathbb{R}) \qquad \Delta; t', \psi, \psi_a;\Phi_a;\Gamma, x:\tau_1 \vdash e_1 \ominus e_1' \downarrow \tau, t' \Rightarrow \Phi_2 \\ \Delta; t', \psi, \psi_a;\Phi_a;\Gamma, y:\tau_2 \vdash e_2 \ominus e_2' \downarrow \tau, t' \Rightarrow \Phi_3 \\ \Phi = \exists(\psi).\Phi_1 \wedge (\exists t' :: \mathbb{R}.\Phi_2 \wedge \Phi_3 \wedge (t' + t_e \doteq t))\end{array}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathrm{case}\ (e, x.e_1, y.e_2) \ominus \mathrm{case}\ (e', x.e_1', y.e_2') \downarrow \tau, t \Rightarrow \Phi}\ \textbf{alg-r-case-}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;f:\tau_1 \xrightarrow{\mathrm{diff}(t')} \tau_2, x:\tau_1, \Gamma \vdash e \ominus e' \downarrow \tau_2, t' \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathrm{fix}\ f(x).e \ominus \mathrm{fix}\ f(x).e' \downarrow \tau_1 \xrightarrow{\mathrm{diff}(t')} \tau_2, t \Rightarrow \Phi \wedge 0 \doteq t}\ \textbf{alg-r-fix-}\downarrow$$

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;f:\Box(\tau_1 \xrightarrow{\mathrm{diff}(t')} \tau_2), x:\tau_1, \Box\Gamma \vdash e \ominus e' \downarrow \tau_2, t' \Rightarrow \Phi \\ \Phi' = \Phi \wedge 0 \doteq t\end{array}}{\Delta;\psi_a;\Phi_a;\Gamma', \Box\Gamma \vdash \mathrm{fix}_{NC}\ f(x).e \ominus \mathrm{fix}_{NC}\ f(x).e \downarrow \Box(\tau_1 \xrightarrow{\mathrm{diff}(t')} \tau_2), t \Rightarrow \Phi'}\ \textbf{alg-r-fix-}\downarrow\Box$$

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Gamma \vdash e_1 \ominus e_1' \uparrow \tau_1 \xrightarrow{\mathrm{diff}(t_e)} \tau_2 \Rightarrow [\psi], t_1, \Phi_1 \\ t_2 \in \mathrm{fresh}(\mathbb{R}) \qquad \Delta; t_2, \psi, \psi_a;\Phi_a;\Gamma \vdash e_2 \ominus e_2' \downarrow \tau_1, t_2 \Rightarrow \Phi_2\end{array}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash e_1\ e_2 \ominus e_1'\ e_2' \uparrow \tau_2 \Rightarrow [t_2, \psi], t_1 + t_2 + t_e, \Phi_1 \wedge \Phi_2}\ \textbf{alg-r-app-}\uparrow$$

$$\frac{\begin{array}{c}t_1, t_2 \in \mathrm{fresh}(\mathbb{R}) \qquad \Delta; t_1, \psi_a;\Phi_a;\Gamma \vdash e_1 \ominus e_1' \downarrow \tau_1, t_1 \Rightarrow \Phi_1 \\ \Delta; t_1, \psi_a;\Phi_a;\Gamma \vdash e_2 \ominus e_2' \downarrow \tau_1, t_2 \Rightarrow \Phi_2 \\ \Phi = \exists t_1 :: \mathbb{R}.\Phi_1 \wedge \exists t_1 :: \mathbb{R}.\Phi_2 \wedge t_1 + t_2 \doteq t\end{array}}{\Delta;\psi_a;\Phi_a;\Omega \vdash \langle e_1, e_2\rangle \ominus \langle e_1', e_2'\rangle \downarrow \tau_1 \times \tau_2, t \Rightarrow \Phi}\ \textbf{alg-r-prod-}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma \vdash e \ominus e' \uparrow \tau_1 \times \tau_2 \Rightarrow [\psi], t, \Phi \qquad i \in \{1,2\}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \pi_i(e) \ominus \pi_i(e') \uparrow \tau_i \Rightarrow [\psi], t, \Phi}\ \textbf{alg-r-proj}_i\textbf{-}\uparrow$$

Figure 50: BiRelCost binary algorithmic typing rules (Part 1)

$$\frac{\Delta, \psi_a; \Phi \vdash \tau \; \mathtt{wf}}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \mathrm{nil} \ominus \mathrm{nil} \downarrow \mathrm{list}[n]^{\alpha}\,\tau, t \Rightarrow n \doteq 0 \wedge 0 \doteq t} \textbf{ alg-r-nil-}\downarrow$$

$$\frac{\begin{array}{c} t_1, t_2 \in \mathrm{fresh}(\mathbb{R}) \qquad i, \beta \in \mathrm{fresh}(\mathbb{N}) \\ \Delta; t_1, \psi_a; \Phi_a; \Gamma \vdash e_1 \ominus e_1' \downarrow \tau, t_1 \Rightarrow \Phi_1 \\ \Delta; i, \beta, t_2, \psi_a; \Phi_a; \Gamma \vdash e_2 \ominus e_2' \downarrow \mathrm{list}[i]^{\beta}\,\tau, t_2 \Rightarrow \Phi_2 \\ \Phi_2' = n \doteq (i+1) \wedge \exists \beta :: \mathbb{N}.\Phi_2 \wedge \alpha \doteq \beta + 1 \wedge t_1 + t_2 \doteq t \\ \Phi = \exists t_1 :: \mathbb{R}.(\Phi_1 \wedge \exists t_2 :: \mathbb{R}.\exists i :: \mathbb{N}.\Phi_2') \end{array}}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \mathrm{cons}_C(e_1, e_2) \ominus \mathrm{cons}_C(e_1', e_2') \downarrow \mathrm{list}[n]^{\alpha}\,\tau, t \Rightarrow \Phi} \textbf{ alg-r-consC-}\downarrow$$

$$\frac{\begin{array}{c} t_1, t_2 \in \mathrm{fresh}(\mathbb{R}) \qquad i \in \mathrm{fresh}(\mathbb{N}) \\ \Delta; t_1, \psi_a; \Phi_a; \Gamma \vdash e_1 \ominus e_1' \downarrow \Box\tau, t_1 \Rightarrow \Phi_1 \\ \Delta; i, t_2, \psi_a; \Phi_a; \Gamma \vdash e_2 \ominus e_2' \downarrow \mathrm{list}[i]^{\alpha}\,\tau, t_2 \Rightarrow \Phi_2 \\ \Phi_2' = \Phi_2 \wedge n \doteq (i+1) \wedge t_1 + t_2 \doteq t \\ \Phi = \exists t_1 :: \mathbb{R}.(\Phi_1 \wedge \exists t_2 :: \mathbb{R}.\exists i :: \mathbb{N}.\Phi_2') \end{array}}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \mathrm{cons}_{NC}(e_1, e_2) \ominus \mathrm{cons}_{NC}(e_1', e_2') \downarrow \mathrm{list}[n]^{\alpha}\,\tau, t \Rightarrow \Phi} \textbf{ alg-r-consNC-}\downarrow$$

$$\frac{\begin{array}{c} \Delta; \psi_a; \Phi_a; \Gamma \vdash e \ominus e' \uparrow \mathrm{list}[n]^{\alpha}\,\tau \Rightarrow [\psi], t_1, \Phi_e \\ t_2 \in \mathrm{fresh}(\mathbb{R}) \qquad \Delta; t_2, \psi, \psi_a; n \doteq 0 \wedge \Phi_a; \Gamma \vdash e_1 \ominus e_1' \downarrow \tau', t_2 \Rightarrow \Phi_1 \\ \Phi_a' = n \doteq i + 1 \wedge \Phi_a \\ i, \Delta; t_2, \psi, \psi_a; \Phi_a'; h : \Box\tau, tl : \mathrm{list}[i]^{\alpha}\,\tau, \Gamma \vdash e_2 \ominus e_2' \downarrow \tau', t_2 \Rightarrow \Phi_2 \\ \Phi_a'' = n \doteq i + 1 \wedge \alpha \doteq \beta + 1 \wedge \Phi_a \\ i, \beta, \Delta; t_2, \psi, \psi_a; \Phi_a''; h : \tau, tl : \mathrm{list}[i]^{\beta}\,\tau, \Gamma \vdash e_3 \ominus e_3' \downarrow \tau', t_2 \Rightarrow \Phi_3 \\ \Phi_{cons} = \forall i :: \mathbb{N}.(n \doteq i + 1) \to (\Phi_2 \wedge \forall \beta :: \mathbb{N}.(\alpha \doteq \beta + 1) \to \Phi_3) \\ \Phi = \exists(\psi).(\Phi_e \wedge \exists t_2 :: \mathbb{R}.((n \doteq 0 \to \Phi_1) \wedge \Phi_{cons} \wedge t_1 + t_2 \doteq t)) \end{array}}{\begin{array}{c} \mathrm{case}\ e\ \mathrm{of\ nil} \ \to e_1 \quad \mathrm{case}\ e'\ \mathrm{of\ nil} \ \to e_1' \\ \Delta; \psi_a; \Phi_a; \Gamma \vdash \mid h ::_{NC} tl \to e_2 \quad \ominus \mid h ::_{NC} tl \to e_2' \quad \downarrow \tau', t \Rightarrow \Phi \\ \mid h ::_{C} tl \to e_3 \quad \mid h ::_{C} tl \to e_3' \end{array}} \textbf{ alg-r-caseL-}\downarrow$$

$$\frac{i :: S, \Delta; \psi_a; \Phi_a; \Gamma \vdash e \ominus e' \downarrow \tau, t_e \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \Lambda i.e \ominus \Lambda i.e' \downarrow \forall i \overset{\mathrm{diff}(t_e)}{::} S.\,\tau, t \Rightarrow (\forall i :: S.\Phi) \wedge 0 \doteq t} \textbf{ alg-r-iLam-}\downarrow$$

$$\frac{\Delta; \psi_a; \Phi_a; \Gamma \vdash e \ominus e' \uparrow \forall i \overset{\mathrm{diff}(t_e)}{::} S.\,\tau' \Rightarrow [\psi], t, \Phi \qquad \Delta \vdash I :: S}{\Delta; \psi_a; \Phi_a; \Gamma \vdash e\,[I] \ominus e'\,[I] \uparrow \tau'\{I/i\} \Rightarrow [\psi], t + t_e[I/i], \Phi} \textbf{ alg-r-iApp-}\uparrow$$

$$\frac{\Delta; \psi_a; \Phi_a; \Gamma \vdash e \ominus e' \downarrow \tau\{I/i\}, t \Rightarrow \Phi \qquad \Delta \vdash I :: S}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \mathrm{pack}\ e\ \mathrm{with}\ I \ominus \mathrm{pack}\ e'\ \mathrm{with}\ I \downarrow \exists i :: S.\,\tau, t \Rightarrow \Phi} \textbf{ alg-r-pack-}\downarrow$$

$$\frac{\begin{array}{c} \Delta; \psi_a; \Phi_a; \Gamma \vdash e_1 \ominus e_1' \uparrow \exists i :: S.\,\tau_1 \Rightarrow [\psi], t_1, \Phi_1 \\ t_2 \in \mathrm{fresh}(\mathbb{R}) \\ i :: S, \Delta; t_2, \psi, \psi_a; \Phi_a; x : \tau_1, \Gamma \vdash e_2 \ominus e_2' \downarrow \tau_2, t_2 \Rightarrow \Phi_2 \\ i \notin \mathrm{FV}(\Phi_a; \Gamma, \tau_2, t_2) \\ \Phi = \exists(\psi).(\Phi_1 \wedge \exists t_2 :: \mathbb{R}.\forall i :: S.\Phi_2 \wedge t_1 + t_2 \doteq t) \end{array}}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \mathrm{unpack}\ e_1\ \mathrm{as}\ (x, i)\ \mathrm{in}\ e_2 \ominus \mathrm{unpack}\ e_1'\ \mathrm{as}\ (x, i)\ \mathrm{in}\ e_2' \downarrow \tau_2, t \Rightarrow \Phi} \textbf{ alg-r-unpac}$$

$$\frac{\begin{array}{c} \Upsilon(\zeta) : \tau_1 \xrightarrow{\mathrm{diff}(t_e)} \tau_2 \qquad t \in \mathrm{fresh}(\mathbb{R}) \\ \Delta; t, \psi_a; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \downarrow \tau_1, t \Rightarrow \Phi \end{array}}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \zeta e_1 \ominus \zeta\, e_2 \uparrow \tau_2 \Rightarrow [t, \psi], t + t_e, \Phi} \textbf{ alg-r-primapp-}\uparrow$$

Figure 51: BiRelCost binary algorithmic typing rules (Part 2)

$$\frac{\Delta; \psi_a; \Phi \wedge C; \Gamma \vdash e_1 \ominus e_1 \downarrow \tau, t \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a; \Omega \vdash e_1 \ominus e_2 \downarrow C \,\&\, \tau, t \Rightarrow C \wedge (C \to \Phi)} \textbf{ alg-r-c-andI-}\downarrow$$

$$\frac{\begin{array}{c}\Delta; \psi_a; \Phi_a; \Omega \vdash e_1 \ominus e_1' \uparrow C \,\&\, \tau_1 \Rightarrow [\psi], t_1, \Phi_1 \\ t_2 \in \text{fresh}(\mathbb{R}) \qquad \Delta; t_2, \psi, \psi_a; \Phi \wedge C; x : \tau_1, \Omega \vdash e_2 \ominus e_2' \downarrow \tau_2, t_2 \Rightarrow \Phi_2 \\ \Phi_2' = C \to \Phi_2 \wedge (t_1 + t_2) \doteq t \qquad \Phi = \exists(\psi).(\Phi_1 \wedge \exists t_2 :: \mathbb{R}.\Phi_2')\end{array}}{\Delta; \psi_a; \Phi_a; \Omega \vdash \text{clet } e_1 \text{ as } x \text{ in } e_2 \ominus \text{clet } e_1' \text{ as } x \text{ in } e_2' \downarrow \tau_2, t \Rightarrow \Phi} \textbf{ alg-r-c-andE-}\downarrow$$

$$\frac{\Delta; \Phi \wedge C; \Gamma \vdash e \ominus e' \downarrow \tau, t \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a; \Gamma \vdash e \ominus e' \downarrow C \supset \tau, t \Rightarrow C \to \Phi} \textbf{ alg-r-c-impI-}\downarrow$$

$$\frac{\Delta; \psi_a; \Phi_a; \Gamma \vdash e \ominus e' \uparrow C \supset \tau \Rightarrow [\psi], t, \Phi}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \text{celim}_{\supset}\ e \ominus \text{celim}_{\supset}\ e' \uparrow \tau \Rightarrow [\psi], t, C \wedge \Phi} \textbf{ alg-r-c-implE-}\uparrow$$

$$\frac{\begin{array}{c}\Delta; \psi_a; \Phi_a; \Gamma \vdash e_1 \ominus e_1' \uparrow \tau_1 \Rightarrow [\psi], t_1, \Phi_1 \\ t_2 \in \text{fresh}(\mathbb{R}) \qquad \Delta; t_2, \psi, \psi_a; x : \tau_1, \Gamma \vdash e_2 \ominus e_2' \downarrow \tau_2, t_2 \Rightarrow \Phi_2 \\ \Phi = \exists(\psi).\Phi_1 \wedge \exists t_2 :: \mathbb{R}.\Phi_2 \wedge t_1 + t_2 \doteq t\end{array}}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \text{let } x = e_1 \text{ in } e_2 \ominus \text{let } x = e_1' \text{ in } e_2' \downarrow \tau_2, t \Rightarrow \Phi} \textbf{ alg-r-let-}\downarrow$$

$$\frac{\begin{array}{c}\Delta; \psi_a; C \wedge \Phi_a; \Gamma \vdash e_1 \ominus e_1' \downarrow \tau, t \Rightarrow \Phi_1 \\ \Delta; \psi_a; \neg C \wedge \Phi_a; \Gamma \vdash e_2 \ominus e_2' \downarrow \tau, t \Rightarrow \Phi_2 \\ \Delta \vdash C \text{ wf} \qquad \Phi = (C \to \Phi_1) \wedge (\neg C \to \Phi_2)\end{array}}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \texttt{split } (e_1, e_2) \text{ with } C \ominus \texttt{split } (e_1', e_2') \text{ with } C \downarrow \tau, t \Rightarrow \Phi} \textbf{ alg-r-split-}\downarrow$$

$$\frac{\Delta; \psi_a; \Phi_a \models \bot}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \texttt{contra } e \ominus \texttt{contra } e' \downarrow \tau, t \Rightarrow \top} \textbf{ alg-r-contra-}\downarrow$$

$$\frac{\begin{array}{c}\Delta; \psi_a; \Phi_a; \Gamma \vdash e \ominus e' \uparrow \tau' \Rightarrow [\psi], t', \Phi_1 \\ \Delta; \psi, \psi_a; \Phi_a \models \tau' \equiv \tau \Rightarrow \Phi_2\end{array}}{\Delta; \psi_a; \Phi_a; \Gamma \vdash e \ominus e' \downarrow \tau, t \Rightarrow \exists(\psi).\Phi_1 \wedge \Phi_2 \wedge t' \leqslant t} \textbf{ alg-r-}\uparrow\downarrow$$

$$\frac{\begin{array}{c}\Delta; \psi_a; \Phi_a; \Gamma \vdash e \ominus e' \downarrow \tau, t \Rightarrow \Phi \\ \Delta; \Phi_a \vdash \tau \text{ wf} \qquad \Delta \vdash t :: \mathbb{R}\end{array}}{\Delta; \psi_a; \Phi_a; \Gamma \vdash (e : \tau, t) \ominus (e' : \tau, t) \uparrow \tau \Rightarrow [\cdot], t, \Phi} \textbf{ alg-r-anno-}\uparrow$$

$$\frac{t' \in \text{fresh}(\mathbb{R}) \qquad \Delta; t', \psi_a; \Phi_a; \Box\Gamma \vdash e \ominus e \downarrow \tau, t' \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a; \Gamma', \Box\Gamma \vdash \texttt{NC } e \ominus \texttt{NC } e \downarrow \Box\tau, t \Rightarrow 0 \doteq t \wedge (\exists t' :: \mathbb{R}.\Phi)} \textbf{ alg-r-nochange-}\downarrow$$

$$\frac{\Delta; \psi_a; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \uparrow \Box\tau \Rightarrow [\psi], t, \Phi}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \texttt{der } e_1 \ominus \texttt{der } e_2 \uparrow \tau \Rightarrow [\psi], t, \Phi} \textbf{ alg-r-der-}\uparrow$$

Figure 52: BiRelCost binary algorithmic typing rules (Part 3)

$\boxed{\Delta;\psi_a;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \downarrow \tau, t \Rightarrow \Phi}$ $e_1 \ominus e_2$ checks against the input type τ and the difference cost t. Finally, it generates the constraint Φ.

$\boxed{\Delta;\psi_a;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \uparrow \tau \Rightarrow [\psi], t, \Phi}$ $e_1 \ominus e_2$ synthesizes the output type τ and the relative cost t where all the newly generated existential variables are defined in ψ. Finally, it generates the constraint Φ.

$$\frac{k_1, t_1, k_2, t_2 \in \text{fresh}(\mathbb{R}) \qquad \Delta; k_1, t_1, \psi_a; \Phi; |\Gamma|_1 \vdash e_1 \downarrow A_1, k_1, t_1 \Rightarrow \Phi_1 \qquad \Delta; k_2, t_2, \psi_a; \Phi; |\Gamma|_2 \vdash e_2 \downarrow A_2, k_2, t_2 \Rightarrow \Phi_2 \qquad \exists k_1, t_1 :: \mathbb{R}.(\Phi_1 \wedge \exists k_2, t_2 :: \mathbb{R}.\Phi_2 \wedge t_1 - k_2 \doteq t)}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \texttt{switch}\ e_1 \ominus \texttt{switch}\ e_2 \downarrow t, U(A_1, A_2) \Rightarrow \Phi}\ \textbf{alg-r-switch-}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi;|\Gamma|_1 \vdash e_1 \uparrow A_1 \Rightarrow [\psi_1], \_, t_1, \Phi_1 \qquad \Delta;\psi_a;\Phi;|\Gamma|_2 \vdash e_2 \uparrow A_2 \Rightarrow [\psi_2], k_2, \_, \Phi_2 \qquad \Phi = \Phi_1 \wedge \Phi_2}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \texttt{switch}\ e_1 \ominus \texttt{switch}\ e_2 \uparrow U(A_1, A_2) \Rightarrow [\psi_1, \psi_2], t_1 - k_2, \Phi}\ \textbf{alg-r-switch-}\uparrow$$

$$\frac{\Delta;\psi_a;\Phi_a;|\Gamma|_1 \vdash e_1 \uparrow A_1 \Rightarrow [\psi], k_1, t_1, \Phi_1 \qquad t_2 \in \text{fresh}(\mathbb{R}) \qquad \Delta; t_2, \psi, \psi_a; \Phi_a; x : U A_1, \Gamma \vdash e_2 \ominus e \downarrow \tau_2, t_2 \Rightarrow \Phi_2 \qquad \Phi = \exists(\psi).(\Phi_1 \wedge \exists t_2 :: \mathbb{R}.\Phi_2 \wedge t_1 + t_2 + c_{let} \doteq t)}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 \ominus e \downarrow \tau_2, t \Rightarrow \Phi}\ \textbf{alg-r-let-e-}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;|\Gamma|_2 \vdash e_1 \uparrow A_1 \Rightarrow [\psi], k_1, t_1, \Phi_1 \qquad t_2 \in \text{fresh}(\mathbb{R}) \qquad \Delta; t_2, \psi, \psi_a; \Phi_a; x : U A_1, \Gamma \vdash e \ominus e_2 \downarrow \tau_2, t_2 \Rightarrow \Phi_2 \qquad \Phi = \exists(\psi).(\Phi_1 \wedge \exists t_2 :: \mathbb{R}.\Phi_2 \wedge t_2 - k_1 - c_{let} \doteq t)}{\Delta;\psi_a;\Phi_a;\Gamma \vdash e \ominus \text{let } x = e_1 \text{ in } e_2 \downarrow \tau_2, t \Rightarrow \Phi}\ \textbf{alg-r-e-let-}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;|\Gamma|_2 \vdash e \uparrow A_1 + A_2 \Rightarrow [\psi], k_1, t_1, \Phi_1 \qquad t_2 \in \text{fresh}(\mathbb{R}) \qquad \Delta;\psi_a;\Phi_a;x : U A_1, \Gamma \vdash e_1 \ominus e' \downarrow \tau, t_2 \Rightarrow \Phi_2 \qquad \Delta;\psi_a;\Phi_a;y : U A_2, \Gamma \vdash e_2 \ominus e' \downarrow \tau, t_2 \Rightarrow \Phi_3 \qquad \Phi = \exists(\phi).\Phi_1 \wedge (\exists t_2 :: \mathbb{R}.\Phi_2 \wedge t_1 + t_2 + c_{case} \doteq t)}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \text{ case } (e, x.e_1, y.e_2) \ominus e' \downarrow \tau, t \Rightarrow \Phi}\ \textbf{alg-r-case-e-}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;|\Gamma|_2 \vdash e' \uparrow A_1 + A_2 \Rightarrow [\psi], k_1, t_1, \Phi_1 \qquad t_2 \in \text{fresh}(\mathbb{R}) \qquad \Delta;\psi_a;\Phi_a;x : U A_1, \Gamma \vdash e \ominus e_1' \downarrow \tau, t_2 \Rightarrow \Phi_2 \qquad \Delta;\psi_a;\Phi_a;y : U A_2, \Gamma \vdash e \ominus e_2' \downarrow \tau, t_2 \Rightarrow \Phi_3 \qquad \Phi = \exists(\phi).\Phi_1 \wedge (\exists t_2 :: \mathbb{R}.\Phi_2 \wedge t_2 - k_1 - c_{case} \doteq t)}{\Delta;\psi_a;\Phi_a;\Gamma \vdash e \ominus \text{ case } (e', x.e_1', y.e_2') \downarrow \tau, t \Rightarrow \Phi}\ \textbf{alg-r-e-case-}\downarrow$$

Figure 53: BiRelCost binary asynchronous algorithmic typing rules (Part 4)

where the heads of the two lists may differ. The two related tails are checked with type $\mathsf{list}[i]^{\beta}\,\tau$, where i and β are newly generated meta variables with the constraint that $\alpha = \beta + 1$ and $n = i + 1$. Since we do not know how the total cost t is distributed between the heads and the tails, we generate two new cost variables t_1 and t_2 to check the heads and the tails, respectively, and generate the constraint $t = t_1 + t_2$.

The list destructors (and also sum type destructors) are typed in checking mode against some result type, since we cannot know the result type by just examining the conditionals. Hence, all branches must be checked against the given result type τ (rule **alg-r-caseL-↓**). [39] To be able to type the branches, we infer that the type of the pattern matched expression e is of form $\mathsf{list}[n]^{\alpha}\,\tau$. Then, we can type each branch in an appropriate environment, following the RelCost Core **c-r-caseL** rule.

[39] *Inferring the types of the branches as well as the whole* `caseL` *expression would require us the compute the least upper bounds of the inferred types of the branches. Instead, when checking, we require that the branches must be checked separately against the given type* τ.

Similar to functions, pairs of universally quantified expressions $\Lambda i.e$ and $\Lambda i.e'$ are typed in checking mode by checking the related closures e and e' in checking mode against the the latent relative cost of the closures (rule **alg-r-iLam-↓**). The resulting constraint contains $t \doteq 0$ on the total cost and also universally quantifies over the constraint of the closure. Dually, in the rule **alg-r-iApp-↑**, the latent cost of the closure is first substituted with the given index term I and then added to the total relative cost of the index term application.

Constructors for existentially quantified types are typed in checking mode by checking the enclosed expressions e and e' in checking mode (rule **alg-r-pack-↓**). Similar to list or sum types, the destructors for existentially quantified types are also typed in checking mode by first inferring the type of the unpacked expressions (rule **alg-r-unpack-↓**). Then, we check the continuations in an extended sort environment that includes i and also in an extended meta variable environment that includes all the meta variables ψ inferred in unpacking the expressions. Due to the former, the final constraint universally quantifies over the constraint of the continuation with i and, due to the latter, the whole final constraint is existentially quantified with ψ.

Asynchronous typing rules of RelCost Core are typed in checking mode with the same principle as the typing of let bindings.

Remark. *A particularly interesting aspect of bidirectional typing in BiRelCost is that the effects (costs), both unary and relational, are constraint-dependent. For example, the relative cost of a function's body might depend on the sizes of its inputs. In this case, a constraint will relate the cost and the inputs' sizes. In existing bidirectional systems with refinement types, similar interactions show up among the sizes of data structures like lists [105, 106].*

$\boxed{\Delta; \psi_a; \Phi_a \models^{A} A_1 \sqsubseteq A_2 \Rightarrow \Phi}$ checks whether A_1 is subtype of A_2 and generates constraints Φ

$\boxed{\Delta; \psi_a; \Phi_a \models \tau_1 \equiv \tau_2 \Rightarrow \Phi}$ checks whether τ_1 is equivalent to τ_2 and generates constraints Φ

$$\frac{}{\Delta; \psi_a; \Phi_a \models \mathrm{int}_r \equiv \mathrm{int}_r \Rightarrow \top}\ \textbf{alg-r-int}$$

$$\frac{}{\Delta; \psi_a; \Phi_a \models \mathrm{unit}_r \equiv \mathrm{unit}_r \Rightarrow \top}\ \textbf{alg-r-unit}$$

$$\frac{\Delta; \psi_a; \Phi_a \models \tau_1 \equiv \tau_1' \Rightarrow \Phi_1 \qquad \Delta; \psi_a; \Phi_a \models \tau_2 \equiv \tau_2' \Rightarrow \Phi_2}{\Delta; \psi_a; \Phi_a \models \tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2 \equiv \tau_1' \xrightarrow{\mathrm{diff}(t')} \tau_2' \Rightarrow \Phi_1 \wedge \Phi_2 \wedge t \doteq t'}\ \textbf{alg-r-fun}$$

$$\frac{\Delta; \psi_a; \Phi_a \models \tau_1 \equiv \tau_1' \Rightarrow \Phi_1 \qquad \Delta; \psi_a; \Phi_a \models \tau_2 \equiv \tau_2' \Rightarrow \Phi_2}{\Delta; \psi_a; \Phi_a \models \tau_1 \times \tau_2 \equiv \tau_1' \times \tau_2' \Rightarrow \Phi_1 \wedge \Phi_2}\ \textbf{alg-r-prod}$$

$$\frac{\Delta; \psi_a; \Phi_a \models \tau_1 \equiv \tau_1' \Rightarrow \Phi_1 \qquad \Delta; \psi_a; \Phi_a \models \tau_2 \equiv \tau_2' \Rightarrow \Phi_2}{\Delta; \psi_a; \Phi_a \models \tau_1 + \tau_2 \equiv \tau_1' + \tau_2' \Rightarrow \Phi_1 \wedge \Phi_2}\ \textbf{alg-r-sum}$$

$$\frac{\Delta; \psi_a; \Phi_a \models \tau \equiv \tau' \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a \models \mathrm{list}[n]^{\alpha}\,\tau \equiv \mathrm{list}[n']^{\alpha'}\,\tau' \Rightarrow \Phi \wedge n \doteq n' \wedge \alpha \doteq \alpha'}\ \textbf{alg-r-list}$$

$$\frac{i, \Delta; \psi_a; \Phi_a \models \tau \equiv \tau' \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a \models \forall i \overset{\mathrm{diff}(t)}{::} S.\,\tau \equiv \forall i \overset{\mathrm{diff}(t')}{::} S.\,\tau' \Rightarrow \forall i :: S.\Phi \wedge t \doteq t'}\ \forall$$

$$\frac{i, \Delta; \psi_a; \Phi_a \models \tau \equiv \tau' \Rightarrow \Phi \qquad i \notin \mathrm{FV}(\Phi_a)}{\Delta; \psi_a; \Phi_a \models \exists i::S.\,\tau \equiv \exists i::S.\,\tau' \Rightarrow \forall i :: S.\Phi}\ \textbf{alg-r-}\exists$$

$$\frac{\Delta; \psi_a; \Phi_a \models \tau_1 \equiv \tau_2 \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a \models \Box\,\tau_1 \equiv \Box\,\tau_2 \Rightarrow \Phi}\ \textbf{B-}\Box$$

$$\frac{\begin{array}{ll}\Delta; \psi_a; \Phi_a \models^{A} A_1 \sqsubseteq A_1' \Rightarrow \Phi_1 & \Delta; \psi_a; \Phi_a \models^{A} A_1' \sqsubseteq A_1 \Rightarrow \Phi_1' \\ \Delta; \psi_a; \Phi_a \models^{A} A_2 \sqsubseteq A_2' \Rightarrow \Phi_2 & \Delta; \psi_a; \Phi_a \models^{A} A_2' \sqsubseteq A_2 \Rightarrow \Phi_2'\end{array}}{\Delta; \psi_a; \Phi_a \models \mathrm{U}\,(A_1, A_2) \equiv \mathrm{U}\,(A_1', A_2') \Rightarrow \Phi_1 \wedge \Phi_1' \wedge \Phi_2 \wedge \Phi_2'}\ \textbf{U}$$

$$\frac{\Delta; \psi_a; \Phi_a \models \tau \equiv \tau' \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a \models C \supset \tau \equiv C' \supset \tau' \Rightarrow C \leftrightarrow C' \wedge \Phi}\ \textbf{c-impl}$$

$$\frac{\Delta; \psi_a; \Phi_a \models \tau \equiv \tau' \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a \models C \,\&\, \tau \equiv C' \,\&\, \tau' \Rightarrow C \leftrightarrow C' \wedge \Phi}\ \textbf{c-prod}$$

Figure 54: Algortihmic type equivalence rules

13.2 SOUNDNESS AND COMPLETENESS OF BIRELCOST

We prove that the algorithmic type system of BiRelCost is sound and complete w.r.t. the type system of RelCost Core. Specifically, soundness says that any inference or checking judgment provable in the algorithmic type system can be simulated in RelCost Core if the output constraints Φ are satisfiable. Dually, completeness says that any typeable RelCost Core program can be sufficiently annotated (with types) to make its type checkable in BiRelCost, with satisfiable output constraints. We define $|e|$ to be the function that erases typing annotations from a BiRelCost expression e to yield a RelCost Core expression. Several cases of its definition is shown in Figure 55; it is a homomorphic function.

$$
\begin{array}{lcl}
|.| & : & \text{Expression} \to \text{Expression} \\
 & & \\
|n| & = & n \\
|()| & = & () \\
|x| & = & x \\
|\text{fix } f(x).e| & = & \text{fix } f(x).|e| \\
|\text{fix}_{NC}\ f(x).e| & = & \text{fix}_{NC}\ f(x).|e| \\
|e_1\ e_2| & = & |e_1|\ |e_2| \\
\vdots & & \\
|(e : A, k, t)| & = & |e| \\
|(e : \tau, t)| & = & |e|
\end{array}
$$

Figure 55: Annotation erasure

Theorem 15 (Soundness of algorithmic typechecking)**.**

1. *Assume that* $\Delta; \psi_a; \Phi_a; \Omega \vdash e \downarrow A, k, t \Rightarrow \Phi$ *and* $FIV(\Phi_a, \Omega, A, k, t) \subseteq dom(\Delta, \psi_a)$ *and* θ_a *is a valid substitution for* ψ_a *such that* $\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$ *holds. Then,* $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t[\theta_a]}_{k[\theta_a]} |e| :^{c} A[\theta_a]$.

2. *Assume that* $\Delta; \psi_a; \Phi_a; \Omega \vdash e \uparrow A \Rightarrow [\psi], k, t, \Phi$ *and* $FIV(\Phi_a, \Omega) \subseteq dom(\Delta, \psi_a)$ *and* θ *and* θ_a *are valid substitutions for* ψ *and* ψ_a *such that* $\Delta; \Phi_a[\theta_a] \models \Phi[\theta\,\theta_a]$ *holds. Then,* $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t[\theta\,\theta_a]}_{k[\theta\,\theta_a]} |e| :^{c} A[\theta\,\theta_a]$.

3. *Assume that* $\Delta; \psi_a; \Phi_a; \Gamma \vdash e \ominus e' \downarrow \tau, t \Rightarrow \Phi$ *and* $FIV(\Phi_a, \Gamma, \tau, t) \subseteq dom(\Delta, \psi_a)$ *and* θ_a *is a valid substitution for* ψ_a *such that* $\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$ *holds. Then,* $\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |e| \ominus |e'| \lesssim t[\theta_a] :^{c} \tau[\theta_a]$.

4. *Assume that* $\Delta; \psi_a; \Phi_a; \Gamma \vdash e \ominus e' \uparrow \tau \Rightarrow [\psi], t, \Phi$ *and* $FIV(\Phi_a, \Gamma) \subseteq dom(\Delta, \psi_a)$ *and* θ *and* θ_a *are valid substitutions for* ψ *and* ψ_a *such that* $\Delta; \Phi_a[\theta_a] \models \Phi[\theta\,\theta_a]$ *holds.*
Then, $\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |e| \ominus |e'| \lesssim t[\theta\,\theta_a] :^{c} \tau[\theta\,\theta_a]$.

Proof. By simultaneous induction on the given algorithmic typing derivations (shown in Appendix C.2).[40] □

[40] FIV *stands for free index variables.*

Theorem 16 (Completeness of algorithmic typechecking)**.**

1. *Assume that* $\Delta; \Phi_a; \Omega \vdash^{t}_{k} e :^{c} A$. *Then, there exists an annotated term* e' *such that* $\Delta; \cdot; \Phi_a; \Omega \vdash e' \downarrow A, k, t \Rightarrow \Phi$ *and* $\Delta; \Phi_a \models \Phi$ *and* $|e'| = e$.

2. *Assume that* $\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t :^{c} \tau$. *Then, there exist two annotated terms* e_1', e_2' *such that* $\Delta; \cdot; \Phi_a; \Gamma \vdash e_1' \ominus e_2' \downarrow \tau, t \Rightarrow \Phi$ *and* $\Delta; \Phi_a \models \Phi$ *and* $|e_1'| = e_1$ *and* $|e_2'| = e_2$.

Proof. By simultaneous induction on the given RelCost Core typing derivations (shown in Appendix C.2). □

13.3 INFERENCE OF COSTS IN CHECKING MODE

In BiRelCost's checking and inference judgments, the cost follows the same polarity as the type: whenever we can check (infer) the type, we can also check (infer) the cost. This design choice is mainly motivated by the fact that the type provided in the checking mode imposes some restrictions on the cost of the program, making it natural to check the cost along with the type. For instance, in **alg-r-fix-↓** rule, the relative cost of the function bodies, t', is already given in the type $\tau_1 \xrightarrow{diff(t')} \tau_2$, which is provided by the surrounding context in the checking mode. Hence, when checking the bodies with the return type τ_2, we can also check that their relative cost is the given cost t'. Another way to interpret this would be to consider the translation of the effects to the monadic setting. For instance, in such a translation, the effect annotated relational type $\tau_1 \xrightarrow{diff(t')} \tau_2$ is translated to the monadic type $\tau_1 \to M_{t'}\,\tau_2$, where $M_{t'}$ is a cost-indexed monad. Then, the function bodies would be checked with the given monadic type $M_{t'}\,\tau_2$, justifying the alignment of the polarities.

We believe this is a principled way of aligning the polarity of costs with types. Hence, in the preceding chapters, the theoretical development assumes that the cost follows the same polarity as the type.

Still, we find it instructive to sketch a formalization of the algorithmic typing rules where the cost could be *partially* inferred in the checking mode. The inference is not fully possible, since additional constraints are needed to deal with the restriction on the costs imposed by the given type. This alternative design aims to highlight the importance of constraint-dependency of costs and trade-offs in generating existential variables in the design of refinement type and effect systems.

As a starting point, we modify BiRelCost's *checking* judgment to the following form

$$\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \downarrow \tau \Rightarrow [\psi], t, \Phi$$

where unlike the type τ, which is given, the cost t is inferred. The additional output, ψ, contains freshly generated existential meta variables that occur in this inferred cost t (similarly, we infer such a context ψ in the inference judgment). Below, we discuss a few rules.

First, **alg-r-fix-$\downarrow$** rule of BiRelCost must be adjusted to the following version:

$$\frac{\begin{array}{c}\Delta; \phi_a; f : \tau_1 \xrightarrow{\mathrm{diff}(t')} \tau_2, x : \tau_1, \Gamma \vdash e \ominus e' \downarrow \tau_2 \Rightarrow [\psi], t'', \Phi' \\ \Phi = \exists(\psi).\Phi' \wedge t'' \leqslant t'\end{array}}{\Delta; \Phi_a; \Gamma \vdash \mathrm{fix}\ f(x).e \ominus \mathrm{fix}\ f(x).e' \downarrow \tau_1 \xrightarrow{\mathrm{diff}(t')} \tau_2 \Rightarrow [\cdot], 0, \Phi}\ \textbf{alg-r-fix-}\downarrow$$

where the inferred relative cost t'' for the function bodies must be no more than t', the relative latent cost provided on the arrow type. A similar additional constraint is needed for checking universally quantified types.

Second, rules that pattern match on list types must be adjusted. For instance, let us consider the skeleton of BiRelCost's **alg-u-caseL-$\downarrow$** rule: [1]

$$\frac{\begin{array}{c}\Delta; \Phi_a; \Gamma \vdash e \ominus e' \uparrow \mathrm{list}[n]^{\alpha}\,\tau \Rightarrow [\cdots], t, \Phi_e \\ \Delta; n \doteq 0 \wedge \Phi_a; \Gamma \vdash e_1 \ominus e_1' \downarrow \tau' \Rightarrow [\cdots], t_1, \Phi_1 \\ i, \Delta; \Phi_a'; h : \square\,\tau, tl : \mathrm{list}[i]^{\alpha}\,\tau, \Gamma \vdash e_2 \ominus e_2' \downarrow \tau' \Rightarrow [\cdots], t_2, \Phi_2 \\ i, \beta, \Delta; \Phi_a''; h : \tau, tl : \mathrm{list}[i]^{\beta}\,\tau, \Gamma \vdash e_3 \ominus e_3' \downarrow \tau' \Rightarrow [\cdots], t_3, \Phi_3 \\ t_f = t + max(t_1, t_2, t_3)\end{array}}{\begin{array}{rl}\Delta; \Phi_a; \Gamma \vdash & \mathrm{case}\ e\ \mathrm{of}\ \mathrm{nil} \to e_1 \quad \mathrm{case}\ e'\ \mathrm{of}\ \mathrm{nil} \to e_1' \\ & \mid h ::_{NC} tl \to e_2 \quad \ominus \mid h ::_{NC} tl \to e_2' \quad \downarrow \tau' \Rightarrow [\cdots], t_f, \cdots \\ & \mid h ::_{C} tl \to e_3 \quad\ \ \mid h ::_{C} tl \to e_3'\end{array}}\ \textbf{alg-r-caseL-}\downarrow$$

where we infer a cost t_i for each branch of the case construct ($i \in \{1, 2, 3\}$). Can we claim that the total cost is $t_f = t + max(t_1, t_2, t_3)$?

1 Here, we have $\Phi_a' = n \doteq i + 1 \wedge \Phi_a$ and $\Phi_a'' = n \doteq i + 1 \wedge \alpha \doteq \beta + 1 \wedge \Phi_a$.

If yes, we would have an advantage over the original rule where we have to generate a fresh variable for the costs of the branches. Unfortunately, the answer is no. Doing so would be unsound because we have a constraint-dependent system. That means, the cost of each branch is only valid under the respective assumption: the cost t_1 is only valid if $\Phi_a \wedge n \doteq 0$ holds. By taking the maximum of the costs of the branches, we would be ignoring the conditions in which the bounds are valid, hence obtaining an unsound bound.

Instead, we have no choice but to generate a fresh cost variable t' for the relative costs of the branches *and* make sure that for each branch, the inferred cost is no more than t'. Then, the total cost is $t + t'$, i. e., the sum of the inferred cost of the scrutinee and the yet-unknown cost of the branches:

$$\frac{\begin{array}{c}\Delta; \Phi_a; \Gamma \vdash e \ominus e' \uparrow \mathrm{list}[n]^{\alpha}\,\tau \Rightarrow [\psi], t, \Phi_e \qquad t' \in \mathrm{fresh}(\mathbb{R}) \\ \Delta; n \doteq 0 \wedge \Phi_a; \Gamma \vdash e_1 \ominus e_1' \downarrow \tau' \Rightarrow [\psi_1], t_1, \Phi_1 \\ i, \Delta; \Phi_a'; h : \square\,\tau, tl : \mathrm{list}[i]^{\alpha}\,\tau, \Gamma \vdash e_2 \ominus e_2' \downarrow \tau' \Rightarrow [\psi_2], t_2, \Phi_2 \\ i, \beta, \Delta; \Phi_a''; h : \tau, tl : \mathrm{list}[i]^{\beta}\,\tau, \Gamma \vdash e_3 \ominus e_3' \downarrow \tau' \Rightarrow [\psi_3], t_3, \Phi_3 \\ \psi' = t', \psi_1, \psi_2, \psi_3 \qquad t_1 \leqslant t' \qquad t_2 \leqslant t' \qquad t_3 \leqslant t'\end{array}}{\begin{array}{l}\Delta; \Phi_a; \Gamma \vdash \begin{array}{l}\text{case } e \text{ of nil } \to e_1 \\ \quad \mid h ::_{NC} tl \to e_2\end{array} \ominus \begin{array}{l}\text{case } e' \text{ of nil } \to e_1' \\ \quad \mid h ::_{C} tl \to e_3'\end{array} \downarrow \tau' \Rightarrow [\psi'], t + t', \cdots \\ \quad \mid h ::_{C} tl \to e_3\end{array}}\ \textbf{alg-r-caseL-}\downarrow$$

Moreover, since the total cost $t + t'$ now contains a freshly generated meta-variable t', which must be passed to the surrounding context by tracking an environment ψ in the checking mode (we would have to also pass ψ_1, ψ_2, ψ_3, i. e., all the freshly generated variables of the branches). Contrast this to the original **alg-r-caseL-**$\downarrow$ rule, where we immediately quantify over the cost meta-variable t'. The advantage of local quantification is twofold: a) constraints can be potentially solved locally when possible and b) scope of the eliminated existential variable is also localized. Instead, in this revised case, by passing t' to the surrounding context, we end up pushing all the existential variables upwards (outer), consequently making the existential elimination much harder and constraint-solving less local. This suggests that aligning the polarity of types and costs is better.

We discuss one more observation: the above rule also embeds subeffecting which was present in BiRelCost's **alg-r-↑↓** rule. In the revised version of **alg-r-↑↓**, there is no need for subeffecting:

$$\frac{\Delta;\Phi_a;\Gamma \vdash e \ominus e' \uparrow \tau' \Rightarrow [\psi], t, \Phi_1 \qquad \Delta;\psi,\psi_a;\Phi_a \models \tau' \equiv \tau \Rightarrow \Phi_2}{\Delta;\Phi_a;\Gamma \vdash e \ominus e' \downarrow \tau \Rightarrow [\psi], t, \Phi_1 \wedge \Phi_2}\ \textbf{alg-r-}\uparrow\downarrow$$

since the inferred cost in the premise can be directly passed to the checking mode. Instead, subeffecting switches polarities, and can be embedded in the **alg-r-anno-↑** rule:

$$\frac{\Delta;\Phi_a;\Gamma \vdash e \ominus e' \downarrow \tau \Rightarrow [\psi], t', \Phi \qquad \mathrm{FIV}(\tau, t) \in \Delta}{\Delta;\Phi_a;\Gamma \vdash (e:\tau, t) \ominus (e':\tau, t) \uparrow \tau \Rightarrow [\psi], t, \Phi \wedge t' \leqslant t}\ \textbf{alg-r-anno-}\uparrow$$

where the inferred cost t' in the premise must be smaller than the cost t given in the annotation.

13.4 BIDIRECTIONAL TYPE SYSTEM FOR DUCOSTIT

Since type and effect systems of DuCostIt and RelCost are very similar, the key ideas behind our algorithmic technique can be directly applied to DuCostIt. Hence, we will not reiterate a similar algorithmic system for DuCostIt. Instead, we briefly describe what changes are needed for transforming RelCost's algorithmic system and implementation to DuCostIt's.

In essence, many of DuCostIt's typing and subtyping rules are simpler than RelCost's:

- In DuCostIt, we do not need asynchronous rules that relate two different expressions. Hence, the corresponding asynchronous algorithmic rules in BiRelCost can be omitted for DuCostIt, hereby simplifying the typechecker and the implementation.
- In DuCostIt's unary typing, we do not track lower bounds. Hence, in the corresponding algorithmic version, we can omit lower bounds in BiRelCost's unary typing, hence simplifying the system further.
- Type grammars of DuCostIt and RelCost are almost identical except the types of unary and binary closures. DuCostIt's function (and universally quantified) types do not include a lower bound on the execution cost.
- As mentioned in Section 8.4, DuCostIt's subtyping rules are almost identical except that the rules $\to \Box\mathbf{U}_{\mathrm{cp}}$ and $\forall\Box\mathbf{U}_{\mathrm{cp}}$ in DuCostIt are

stronger. Correspondingly, type equality and heuristic algorithmic subtyping for these rules need to reflect this change.

$\boxed{\Delta; \psi_a; \Phi_a; \Omega \vdash e \downarrow A, k, t \Rightarrow \Phi}$ e checks against the unary input type A and the minimum execution cost k and maximum execution cost t. Finally, it generates the constraint Φ.

$\boxed{\Delta; \psi_a; \Phi_a; \Omega \vdash e \uparrow A \Rightarrow [\psi], k, t, \Phi}$ e synthesizes the unary output type A, the minimum cost k and maximum cost t, where all the newly generated existential variables are defined in ψ. Finally, it generates the constraint Φ.

$$\frac{}{\Delta; \psi_a; \Phi_a; \Omega \vdash n \uparrow \mathsf{int} \Rightarrow [.], 0, 0, \top}\ \textbf{alg-u-n-}\uparrow$$

$$\frac{\Omega(x) = A}{\Delta; \psi_a; \Phi_a; \Omega \vdash x \uparrow A \Rightarrow [.], 0, 0, \top}\ \textbf{alg-u-var-}\uparrow$$

$$\frac{}{\Delta; \psi_a; \Phi_a; \Omega \vdash () \uparrow \mathsf{unit} \Rightarrow [.], 0, 0, \top}\ \textbf{alg-u-unit-}\uparrow$$

$$\frac{\Delta; \psi_a; \Phi_a; \Omega \vdash e \downarrow A_1, k, t \Rightarrow \Phi \qquad \Delta, \psi_a; \Phi_a \vdash^A A_2\ \mathtt{wf}}{\Delta; \psi_a; \Phi_a; \Omega \vdash \mathsf{inl}\ e \downarrow A_1 + A_2, k, t \Rightarrow \Phi}\ \textbf{alg-u-inl-}\downarrow$$

$$\frac{\Delta; \psi_a; \Phi_a; \Omega \vdash e \downarrow A_2, k, t \Rightarrow \Phi \qquad \Delta, \psi_a; \Phi_a \vdash^A A_1\ \mathtt{wf}}{\Delta; \psi_a; \Phi_a; \Omega \vdash \mathsf{inr}\ e \downarrow A_1 + A_2, k, t \Rightarrow \Phi}\ \textbf{alg-u-inr-}\downarrow$$

$$\frac{\begin{array}{c}\Delta; \psi_a; \Phi_a; \Omega \vdash e \uparrow A_1 + A_2 \Rightarrow [\psi], k_e, t_e, \Phi_1 \qquad k', t' \in \mathrm{fresh}(\mathbb{R}) \\ \Delta; k', t', \psi, \psi_a; \Phi_a; \Omega, x : A_1 \vdash e_1 \downarrow A, k', t' \Rightarrow \Phi_2 \\ \Delta; k', t', \psi, \psi_a; \Phi_a; \Omega, y : A_2 \vdash e_2 \downarrow A, k', t' \Rightarrow \Phi_3 \\ \Phi' = \exists k', t' :: \mathbb{R}.\Phi_2 \wedge \Phi_3 \wedge k \doteq k' + k_e + c_{case} \wedge t' + t_e + c_{case} \doteq t\end{array}}{\Delta; \psi_a; \Phi_a; \Omega \vdash\ \mathsf{case}\ (e, x.e_1, y.e_2) \downarrow A, k, t \Rightarrow \exists(\psi).\Phi_1 \wedge \Phi'}\ \textbf{alg-u-case-}\downarrow$$

$$\frac{\Delta; \psi_a; \Phi_a; f : A_1 \xrightarrow{\mathrm{exec}(k', t')} A_2, x : A_1, \Omega \vdash e \downarrow A_2, k', t' \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a; \Omega \vdash \mathsf{fix}\ f(x).e \downarrow A_1 \xrightarrow{\mathrm{exec}(k', t')} A_2, k, t \Rightarrow \Phi \wedge k \doteq 0 \wedge 0 \doteq t}\ \textbf{alg-u-fix-}\downarrow$$

$$\frac{\begin{array}{c}\Delta; \psi_a; \Phi_a; \Omega \vdash e_1 \uparrow A_1 \xrightarrow{\mathrm{exec}(k_e, t_e)} A_2 \Rightarrow [\psi], k_1, t_1, \Phi_1 \\ k_2, t_2 \in \mathrm{fresh}(\mathbb{R}) \qquad \Delta; k_2, t_2, \psi, \psi_a; \Phi_a; \Omega \vdash e_2 \downarrow A_1, k_2, t_2 \Rightarrow \Phi_2 \\ k = k_1 + k_2 + k_e + c_{app} \qquad t = t_1 + t_2 + t_e + c_{app}\end{array}}{\Delta; \psi_a; \Phi_a; \Omega \vdash e_1\ e_2 \uparrow A_2 \Rightarrow [k_2, t_2, \psi], k, t, \Phi_1 \wedge \Phi_2}\ \textbf{alg-u-app-}\uparrow$$

$$\frac{\begin{array}{c}k_1, t_1, k_2, t_2 \in \mathrm{fresh}(\mathbb{R}) \qquad \Delta; k_1, t_1, \psi_a; \Phi_a; \Omega \vdash e_1 \downarrow A_1, k_1, t_1 \Rightarrow \Phi_1 \\ \Delta; k_2, t_2, \psi_a; \Phi_a; \Omega \vdash e_2 \downarrow A_1, k_2, t_2 \Rightarrow \Phi_2 \\ \Phi = \exists k_1, t_1 :: \mathbb{R}.\Phi_1 \wedge \exists k_2, t_2 :: \mathbb{R}.\Phi_2 \wedge t_1 + t_2 \doteq t \wedge k \doteq k_1 + k_2\end{array}}{\Delta; \psi_a; \Phi_a; \Omega \vdash \langle e_1, e_2 \rangle \downarrow A_1 \times A_2, k, t \Rightarrow \Phi}\ \textbf{alg-u-prod-}\downarrow$$

$$\frac{\Delta; \psi_a; \Phi_a; \Omega \vdash e \uparrow A_1 \times A_2 \Rightarrow [\psi], k, t, \Phi \qquad i \in \{1, 2\}}{\Delta; \psi_a; \Phi_a; \Omega \vdash \pi_i(e) \uparrow A_i \Rightarrow [\psi], k, t, \Phi}\ \textbf{alg-u-proj}_i\textbf{-}\uparrow$$

Figure 56: BiRelCost unary algorithmic typing rules (Part 1)

$$\frac{\Delta, \psi_a; \Phi \vdash^{A} A \ \mathtt{wf}}{\Delta; \psi_a; \Phi_a; \Omega \vdash \mathrm{nil} \downarrow \mathrm{list}[n]\, A, k, t \Rightarrow n \doteq 0 \wedge k \doteq 0 \wedge 0 \doteq t} \textbf{alg-u-nil-}\downarrow$$

$$\frac{\begin{array}{c} k_1, t_1, k_2, t_2 \in \mathrm{fresh}(\mathbb{R}) \\ i \in \mathrm{fresh}(\mathbb{N}) \qquad \Delta; k_1, t_1, \psi_a; \Phi_a; \Omega \vdash e_1 \downarrow A, k_1, t_1 \Rightarrow \Phi_1 \\ \Delta; i, k_2, t_2, \psi_a; \Phi_a; \Omega \vdash e_2 \downarrow \mathrm{list}[i]\, A, k_2, t_2 \Rightarrow \Phi_2 \\ \Phi_2' = (\Phi_2 \wedge n \doteq (i+1) \wedge k \doteq k_1 + k_2 \wedge t_1 + t_2 \doteq t) \\ \Phi = \exists k_1, t_1 :: \mathbb{R}.(\Phi_1 \wedge \exists k_2, t_2 :: \mathbb{R}. \exists i :: \mathbb{N}. \Phi_2') \end{array}}{\Delta; \psi_a; \Phi_a; \Omega \vdash \mathrm{cons}_C(e_1, e_2) \downarrow \mathrm{list}[n]\, A, k, t \Rightarrow \Phi} \textbf{alg-u-cons-}\downarrow$$

$$\frac{\begin{array}{c} \Delta; \psi_a; \Phi_a; \Omega \vdash e \uparrow \mathrm{list}[n]\, A \Rightarrow [\psi], k_1, t_1, \Phi_1 \qquad k_2, t_2 \in \mathrm{fresh}(\mathbb{R}) \\ \Delta; k_2, t_2, \psi, \psi_a; n \doteq 0 \wedge \Phi_a; \Omega \vdash e_1 \downarrow A', k_2, t_2 \Rightarrow \Phi_2 \\ i \in \mathrm{fresh}(\mathbb{N}) \qquad \Phi_a' = n \doteq i + 1 \wedge \Phi_a \\ i :: \mathbb{N}, \Delta; k_2, t_2, \psi, \psi_a; \Phi_a'; h : A, tl : \mathrm{list}[i]\, A, \Omega \vdash e_2 \downarrow A', k_2, t_2 \Rightarrow \Phi_3 \\ \Phi_c = k \doteq (k_1 + k_2 + c_{caseL}) \wedge t \doteq (t_1 + t_2 + c_{caseL}) \\ \Phi' = \Phi_1 \wedge \exists k_2, t_2 :: \mathbb{R}.((n \doteq 0 \to \Phi_2) \wedge (\forall i :: \mathbb{N}.(n \doteq i+1) \to \Phi_3) \wedge \Phi_c) \end{array}}{\begin{array}{c} \mathrm{case}\ e\ \mathrm{of\ nil} \ \to e_1 \\ \Delta; \psi_a; \Phi_a; \Omega \vdash \mid h ::_{NC} tl \to e_2 \qquad \downarrow A', k, t \Rightarrow \exists(\psi).\Phi' \\ \mid h ::_C tl \ \to e_3 \end{array}} \textbf{alg-u-caseL-}\downarrow$$

$$\frac{\begin{array}{c} i :: S, \Delta; \psi_a; \Phi_a; \Omega \vdash e \downarrow A, k_e, t_e \Rightarrow \Phi \\ \Phi' = (\forall i :: S.\Phi) \wedge k \doteq 0 \wedge 0 \doteq t \end{array}}{\Delta; \psi_a; \Phi_a; \Omega \vdash \Lambda i.e \downarrow \forall i \overset{\mathrm{exec}(k_e, t_e)}{::} S.\, A, k, t \Rightarrow \Phi'} \textbf{alg-u-iLam-}\downarrow$$

$$\frac{\Delta; \psi_a; \Phi_a; \Omega \vdash e \uparrow \forall i \overset{\mathrm{exec}(k_e, t_e)}{::} S.\, A' \Rightarrow [\psi], k, t, \Phi \qquad \Delta \vdash I :: S}{\Delta; \psi_a; \Phi_a; \Omega \vdash e\,[I] \uparrow A'\{I/i\} \Rightarrow [\psi], k + k_e[I/i], t + t_e[I/i], \Phi} \textbf{alg-u-iApp-}\uparrow$$

$$\frac{\Delta; \psi_a; \Phi_a; \Omega \vdash e \downarrow A\{I/i\}, k, t \Rightarrow \Phi \qquad \Delta \vdash I :: S}{\Delta; \psi_a; \Phi_a; \Omega \vdash \mathrm{pack}\ e\ \mathrm{with}\ I \downarrow \exists i :: S.\, A, k, t \Rightarrow \Phi} \textbf{alg-u-pack-}\downarrow$$

$$\frac{\begin{array}{c} \Delta; \psi_a; \Phi_a; \Omega \vdash e_1 \uparrow \exists i :: S.\, A_1 \Rightarrow [\psi], k_1, t_1, \Phi_1 \\ k_2, t_2 \in \mathrm{fresh}(\mathbb{R}) \\ i :: S, \Delta; k_2, t_2, \psi, \psi_a; \Phi_a; x : A_1, \Omega \vdash e_2 \downarrow A_2, k_2, t_2 \Rightarrow \Phi_2 \\ i \notin FV(\Phi_a; \Omega, A_2, k_2, t_2) \\ \Phi_c = k \doteq k_1 + k_2 + c_{unp} \wedge t_1 + t_2 + c_{unp} \doteq t \\ \Phi = \Phi_1 \wedge \exists k_2, t_2 :: \mathbb{R}. \forall i :: S. \Phi_2 \wedge \Phi_c \end{array}}{\Delta; \psi_a; \Phi_a; \Omega \vdash \mathrm{unpack}\ e_1\ \mathrm{as}\ (x, i)\ \mathrm{in}\ e_2 \downarrow A_2, k, t \Rightarrow \exists(\psi).\Phi} \textbf{alg-u-unpack-}\downarrow$$

$$\frac{\begin{array}{c} \Upsilon(\zeta) : A_1 \xrightarrow{\mathrm{exec}(k_e, t_e)} A_2 \qquad k, t \in \mathrm{fresh}(\mathbb{R}) \\ \Delta; k, t, \psi_a; \Phi_a; \Omega \vdash e \downarrow A_1, k, t \Rightarrow \Phi \end{array}}{\Delta; \psi_a; \Phi_a; \Omega \vdash \zeta\ e \uparrow A_2 \Rightarrow [k, t, \psi], k + k_e + c_{primapp}, t + t_e + c_{primapp}, \Phi} \textbf{alg-u-primapp-}\uparrow$$

$$\frac{\Delta; \psi_a; \Phi \wedge C; \Omega \vdash e \downarrow A, k, t \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a; \Omega \vdash e \downarrow k, t, C \,\&\, A \Rightarrow C \wedge (C \to \Phi)} \textbf{alg-u-c-andI-}\downarrow$$

Figure 57: BiRelCost unary algorithmic typing rules (Part 2)

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash e_1 \uparrow C \,\&\, A_1 \Rightarrow [\psi], k_1, t_1, \Phi_1 \qquad k_2, t_2 \in \text{fresh}(\mathbb{R}) \quad \Delta; k_2, t_2, \psi, \psi_a; \Phi \wedge C; x : A_1, \Omega \vdash e_2 \downarrow A_2, k_2, t_2 \Rightarrow \Phi_2 \quad \Phi_c = k \doteq (k_1 + k_2) \wedge (t_1 + t_2) \doteq t \quad \Phi' = \exists k_2, t_2 :: \mathbb{R}.C \to \Phi_2 \wedge \Phi_c}{\Delta;\psi_a;\Phi_a;\Omega \vdash \text{clet } e_1 \text{ as } x \text{ in } e_2 \downarrow A_2, k, t \Rightarrow \exists(\psi).(\Phi_1 \wedge \Phi')} \textbf{ alg-u-c-andE-}\downarrow$$

$$\frac{\Delta;\Phi \wedge C;\Omega \vdash e \downarrow A, k, t \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Omega \vdash e \downarrow C \supset A, k, t \Rightarrow C \to \Phi} \textbf{ alg-u-c-implI-}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash e \uparrow C \supset A \Rightarrow [\psi], k, t, \Phi}{\Delta;\psi_a;\Phi_a;\Omega \vdash \text{celim}_{\supset}\ e \uparrow A \Rightarrow [\psi], k, t, C \wedge \Phi} \textbf{ alg-u-c-implE-}\uparrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash e_1 \uparrow A_1 \Rightarrow [\psi], k_1, t_1, \Phi_1 \quad k_2, t_2 \in \text{fresh}(\mathbb{R}) \quad \Delta; k_2, t_2, \psi, \psi_a; x : A_1, \Omega \vdash e_2 \downarrow A_2, k_2, t_2 \Rightarrow \Phi_2 \quad \Phi_2' = \Phi_2 \wedge k \doteq (k_1 + k_2 + c_{let}) \wedge (t_1 + t_2 + c_{let}) \doteq t}{\Delta;\psi_a;\Phi_a;\Omega \vdash \text{let } x = e_1 \text{ in } e_2 \downarrow A_2, k, t \Rightarrow \exists(\psi).\Phi_1 \wedge \exists k_2, t_2 :: \mathbb{R}.\Phi_2'} \textbf{ alg-u-let-}\downarrow$$

$$\frac{\Delta;\psi_a;C \wedge \Phi_a;\Omega \vdash e_1 \downarrow A, k, t \Rightarrow \Phi_1 \quad \Delta;\psi_a;\neg C \wedge \Phi_a;\Omega \vdash e_2 \downarrow A, k, t \Rightarrow \Phi_2 \quad \Delta \vdash C\ \texttt{wf} \quad \Phi = C \to \Phi_1 \wedge \neg C \to \Phi_2}{\Delta;\psi_a;\Phi_a;\Omega \vdash \texttt{split}\ (e_1, e_2) \text{ with } C \downarrow A, k, t \Rightarrow \Phi} \textbf{ alg-u-split-}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a \models \bot}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \texttt{contra}\ e \downarrow A, k, t \Rightarrow \top} \textbf{ alg-u-contra-}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash e \uparrow A' \Rightarrow [\psi], k', t', \Phi_1 \quad \Delta;\psi,\psi_a;\Phi_a \models^{A} A' \sqsubseteq A \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a;\Omega \vdash e \downarrow A, k, t \Rightarrow \exists(\psi).\Phi_1 \wedge \Phi_2 \wedge t' \leqslant t \wedge k \leqslant k'} \textbf{ alg-}\uparrow\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash e \downarrow A, k, t \Rightarrow \Phi \qquad \Delta;\Phi_a \vdash^{A} A\ \texttt{wf} \qquad \text{FIV}(A, k, t) \in \Delta}{\Delta;\psi_a;\Phi_a;\Omega \vdash (e : A, k, t) \uparrow A \Rightarrow [\cdot], k, t, \Phi} \textbf{ alg-u-anno-}\uparrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash e \uparrow A' \Rightarrow [\psi], k', t', \Phi_1 \quad \Delta;\psi,\psi_a;\Phi_a \models^{A} A' \sqsubseteq A \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a;\Omega \vdash e \downarrow A, k, t \Rightarrow \exists(\psi).\Phi_1 \wedge \Phi_2 \wedge t' \leqslant t \wedge k \leqslant k'} \textbf{ alg-}\uparrow\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash e \downarrow A, k, t \Rightarrow \Phi \qquad \Delta;\Phi_a \vdash^{A} A\ \texttt{wf} \qquad \text{FIV}(A, k, t) \in \Delta}{\Delta;\psi_a;\Phi_a;\Omega \vdash (e : A, k, t) \uparrow A \Rightarrow [\cdot], k, t, \Phi} \textbf{ alg-u-anno-}\uparrow$$

Figure 58: BiRelCost unary algorithmic typing rules (Part 3)

14

IMPLEMENTATION AND CASE STUDIES

> You think you know when you learn, are more sure when you can write, even more when you can teach, but certain when you can program.
>
> –Alan J. Perlis, "Epigrams on programming" #116

▶ SYNOPSIS This chapter presents an implementation of BiRelCost's typechecker and then presents a case study in which we use our prototype implementation to typecheck a wide variety of example programs. Finally, an experimental evaluation of the typechecker is presented.

We have implemented the bidirectional type checker described in Chapter 13 as a standalone tool in OCaml[1]. Our type checker accepts programs not in the elaborate, annotated syntax of RelCost Core, which is rather inconvenient to use, but instead in the syntax of RelCost. Internally, it performs the RelCost to RelCost Core embedding of Chapter 12 using sound but incomplete heuristics to decide when to apply the non-syntax-directed typing rules of RelCost, and a sound but incomplete procedure for RelCost's relational subtyping. The bidirectional implementation is sound in the sense of Theorem 59, and also complete in the sense of Theorem 60, modulo the incompleteness of our heuristics.

In the following, we describe our heuristics, give an overview of our bidirectional implementation, and present the results of an experimental evaluation along with a detailed description of two examples demonstrating how some of the heuristics are useful for type checking. All the prior examples presented in Chapter 4 along with many additional examples are typechecked in this implementation. Throughout this chapter, we give explanations for some decisions we made in the implementation of BiRelCost.

1 The implementation and the examples are online at: https://github.com/ezgicicek/bi_relcost

14.1 HEURISTICS

The heuristics we use to treat nondeterminism in RelCost's typing and relational subtyping rules are sound but incomplete. We describe some of the current heuristics here.

1. When typing a function that takes an argument of type $\mathsf{list}[n]^{\alpha}\,\tau$, we immediately apply the rule **alg-r-split-↓** from Figure 52 with $C = (\alpha = 0)$ to split cases on whether $\alpha = 0$ or not. For the case $\alpha = 0$, we first try to complete the typing by invoking the **alg-r-nochange-↓** rule (if the two functions being typed are identical), then we try invoking the **alg-r-fix-↓** rule. This is because many recursive list programs require this analysis (e. g.. mergesort). Moreover, the rule **alg-r-split** is invertible: Applying the rule does not reduce the possibility of finding the proof.

2. When typing a pair of cons-ed lists, we try the algorithmic analogues of both the rules **r-cons1** and **r-cons2** (rules **alg-r-consC-↓** and **alg-r-consNC-↓** in Figure 51). If neither of the rules fails, i.e. we are able to generate constraints under which they might typecheck, we proceed by combining the resulting constraints via disjunction. Otherwise, there are two cases: a) either both fail, whence we output a type error without needing to solve any constraints, or b) one of them fails, whence we return the constraints of the successful rule.

3. Specific RelCost relational subtyping rules that mention $\Box$ are applied lazily at specific elimination points. For instance, in typing a function application, if the applied expression's inferred type is $\Box\,(\tau_1 \xrightarrow{\mathsf{diff}(k)} \tau_2)$, we try to complete the typing by subtyping to $\Box\,\tau_1 \xrightarrow{\mathsf{diff}(0)} \Box\,\tau_2$ and $\tau_1 \xrightarrow{\mathsf{diff}(k)} \tau_2$, in that order. A similar heuristic is applied for type $\Box\,(U\,A_1 \xrightarrow{\mathsf{exec}(k,t)} A_2)$ so that the typing is completed by subtyping to $\Box\,U\,A_1 \xrightarrow{\mathsf{diff}(0)} \Box\,U\,A_2$ and $U\,A_1 \xrightarrow{\mathsf{exec}(k,t)} A_2$, in that order.

4. We implement a sound but incomplete algorithmic procedure for relational subtyping, which generates necessary constraints. Algorithmic subtyping is only invoked in two places: a) for switching from checking to inference mode (rule **alg-r-↑↓**) and b) for the algorithmic counterpart of the **nochange** rule (rule **alg-r-nochange-↓** in Figure 52) which needs to check that all the free variables in the context can be supertyped to their $\Box$-ed counterparts. Below,

we list some of these subtyping rules (it is fairly easy to check that these rules are sound):

$$\frac{\Delta; \Phi_a \models \Box\tau_1 \sqsubseteq \Box\tau_2 \Rightarrow \Phi}{\Delta; \Phi_a \models \Box\tau_1 \sqsubseteq \Box\Box\tau_2 \Rightarrow \Phi}\textbf{alg-D-}\Box$$

$$\frac{\Delta; \Phi_a \models \tau_1 \sqsubseteq \Box\tau_2 \Rightarrow \Phi_1 \qquad \Delta; \Phi_a \models \tau_1 \sqsubseteq \tau_2 \Rightarrow \Phi_2}{\Delta; \Phi_a \models \Box\tau_1 \sqsubseteq \Box\tau_2 \Rightarrow \Phi_1 \vee \Phi_2}\textbf{alg-B-}\Box$$

$$\frac{\Delta; \Phi_a \models (\tau_1)^{\downarrow\Box} \sqsubseteq \tau_2 \Rightarrow \Phi}{\Delta; \Phi_a \models \Box\tau_1 \sqsubseteq \tau_2 \Rightarrow \Phi}\textbf{alg-}\Box$$

$$\frac{\Delta; \Phi_a \models \tau \sqsubseteq \tau' \Rightarrow \Phi}{\Delta; \Phi_a \models \mathrm{list}[n]^{\alpha}\tau \sqsubseteq \mathrm{list}[n']^{\alpha'}\tau' \Rightarrow n \doteq n' \wedge \alpha \leqslant \alpha' \wedge \Phi}\textbf{alg-list}$$

$$\frac{\Delta; \Phi_a \models \Box\tau \sqsubseteq \tau' \Rightarrow \Phi}{\Delta; \Phi_a \models \mathrm{list}[n]^{\alpha}\tau \sqsubseteq \Box(\mathrm{list}[n']^{\alpha'}\tau') \Rightarrow n \doteq n' \wedge \alpha \doteq 0 \wedge \Phi}\textbf{alg-list-}\Box$$

$(\tau)^{\downarrow\Box}$ in the rule **alg-**$\Box$ pushes the $\Box$ constructor one-level down into τ, in accordance with RelCost's subtyping rules. For instance, $(\tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2)^{\downarrow\Box} = \Box\tau_1 \xrightarrow{\mathrm{diff}(0)} \Box\tau_2$, $(\tau_1 \times \tau_2)^{\downarrow\Box} = \Box\tau_1 \times \Box\tau_2$ and $(U(A_1 \times A_2))^{\downarrow\Box} = \Box(UA_1) \times \Box(UA_2)$. The rule **alg-**$\Box$ corresponds to the following sound, admissible subtyping: $\Box\tau \sqsubseteq (\tau)^{\downarrow\Box}$.

5. We switch to the unary reasoning (rules **alg-r-switch-**$\downarrow$ and **alg-r-switch-**$\uparrow$) only when it is absolutely necessary: when (a) eliminating expressions of the type UA, (b) checking a pair of expressions at type UA, and (c) inferring a type for two structurally dissimilar expressions.

Since there is no standard library of examples for relational cost analysis so far, we designed our heuristics based on examples that cover a breadth of applications. Appendix D.2 lists all our examples, and our heuristics suffice for all of them. More heuristics can be added if necessary.

14.2 IMPLEMENTATION OF BIDIRECTIONAL RULES

In the implementation, corresponding to BiRelCost's relational type equality judgment, we have a subtyping judgment $\models \tau_1 \sqsubseteq \tau_2 \Rightarrow \Phi$ that makes use of the subtyping heuristics listed in Section 14.1. This judgment and its unary counterpart $\models^{A} A_1 \sqsubseteq A_2 \Rightarrow \Phi$ can be imple-

mented as recursive functions, with the following specifications. Here, $\texttt{ctx} = \Delta; \Phi_a$.

$$\texttt{subtype}_r(\texttt{ctx}, \tau_1, \tau_2) = \begin{cases} \Phi & \text{if } \texttt{ctx} \models \tau_1 \sqsubseteq \tau_2 \Rightarrow \Phi \\ \text{error} & \text{otherwise} \end{cases}$$

$$\texttt{subtype}(\texttt{ctx}, A_1, A_2) = \begin{cases} \Phi & \text{if } \texttt{ctx} \models^A A_1 \sqsubseteq A_2 \Rightarrow \Phi \\ \text{error} & \text{otherwise} \end{cases}$$

Function $\texttt{subtype}_r$ ($\texttt{subtype}$) is defined by case analysis on the types τ_1 and τ_2 (A_1 and A_2). Both functions output a constraint Φ, which, if satisfied, implies that a subtyping derivation exists in RelCost.

The four BiRelCost judgments can be implemented as mutually recursive functions. For instance, the two relational inference and checking judgments are implemented as two functions, $\texttt{infer}_r$ and $\texttt{check}_r$, with the following specifications. Here, $\texttt{ctx} = \Delta; \psi_a; \Phi_a; \Gamma$.

$$\texttt{infer}_r(\texttt{ctx}, e_1, e_2) = \begin{cases} \tau, t, \Phi, \psi & \text{if } \texttt{ctx} \vdash e_1 \ominus e_2 \uparrow \tau \Rightarrow [\psi], t, \Phi \\ \text{error} & \text{otherwise} \end{cases}$$

$$\texttt{check}_r(\texttt{ctx}, e_1, e_2, \tau, t) = \begin{cases} \Phi & \text{if } \texttt{ctx} \vdash e_1 \ominus e_2 \downarrow \tau, t \Rightarrow \Phi \\ \text{error} & \text{otherwise} \end{cases}$$

Function $\texttt{infer}_r$ is defined by case analysis on the expressions e_1 and e_2 whereas $\texttt{check}_r$ is defined by case analysis on both the expressions e_1, e_2 and the type τ. Both functions output a constraint Φ, which, if satisfied, implies that a typing derivation exists in RelCost. In the case of the inference judgment, the constraint Φ is existentially quantified by all the variables in ψ.

Next, we explain how we solve the output constraint Φ.

14.3 CONSTRAINT SOLVING

In principle, we could directly pass the output constraint Φ to an SMT solver that understands the domain of integers (for sizes) and real numbers (for costs). However, the constraint typically contains many existentially quantified variables which cannot be fully eliminated by current SMT solvers.

14.3.1 *Existential elimination*

To solve this problem, we wrote our own pre-processing pass that tries to find substitutions for existentially quantified variables. For a constraint of the form $\exists n.\Phi$, we look inside Φ to find sub-constraints of the form $n = I$ or $n \leqslant I$. In either case, I is candidate substitution for n. We have to be careful that I does not contain variables that are quantified within Φ (else the scope of those variables is not respected). The rules for existential variable elimination are shown in Figure 59.

We use the judgment $\texttt{find}(i, \Phi) \downarrow I, \Phi'$ to mean that finding a substitution for the index variable i in Φ results in an index term I and a constraint Φ'. Then, we can lift this to constraints with arbitrarily nested existential variables using the judgment $\texttt{elim}_\exists(\Phi) \downarrow \Phi'$ which means that eliminating all the existential variables in Φ results in a constraint Φ'.

The rules for existential elimination are not deterministic. Hence, our implementation uses several heuristics in combination with a lazy search mechanism with backtracking to try all candidate substitutions: The implementation traverses the constraint top-down from the root towards the leaves, finding candidate substitutions for existential quantifiers it encounters. The priority is given to the rightmost constraint c_2 in constraints of the form $c_1 \wedge c_2$ and $c_1 \vee c_2$ (since we always append arithmetic cost and type constraints to the end). For constraints of the form $i \leqslant I$ and $i \doteq I$, the priority is given to equality. As soon as a substitution for existential variables is found, it is applied and the resulting existential-free formula is sent to an SMT solver (described next). If the SMT solver proves the formula, we are done. If it fails or times out, our search backtracks, looking for the next candidate substitution. This process is potentially expensive, but it terminates very quickly (in less than 1s) on all examples we have tried.

14.3.2 *Solving the constraints*

To prove individual existential-free constraints, we invoke an SMT solver. Specifically, we use Why3 [50], a common front-end for many SMT solvers. Empirically, we have observed that only one SMT solver, Alt-Ergo [20], can handle our constraints and, so our implementation uses this solver behind Why3. Why3 provides libraries of lemmas for exponentiation, logarithms and iterated sums, which we use in some of the examples. For typing programs that use divide-and-conquer over lists (e.g., merge sort), we have to provide as an axiom one additional

$\boxed{\Delta \vdash \mathtt{find}(i;\Phi) \downarrow (I;\Phi')}$ solves Φ for the index variable i and returns in an index term I and a constraint Φ'.
$\boxed{\Delta \vdash \mathtt{elim}_\exists(\Phi) \downarrow \Phi'}$ eliminates all the existential variables in Φ and returns the resulting constraint Φ'.

$$\frac{i', \Delta \vdash \mathtt{find}(i;\Phi) \downarrow (I;\Phi')}{\Delta \vdash \mathtt{find}(i;\exists i' :: S.\Phi) \downarrow (I;\exists i' :: S.\Phi')}\ \textbf{find-}\exists$$

$$\frac{\Delta \vdash \mathtt{find}(i;\Phi_2) \downarrow (I;\Phi_2')}{\Delta \vdash \mathtt{find}(i;\Phi_1 \to \Phi_2) \downarrow (I;\Phi_1 \to \Phi_2')}\ \textbf{find-}\to$$

$$\frac{i \notin \Delta}{\Delta \vdash \mathtt{find}(i;I \leqslant i) \downarrow (I;\top)}\ \textbf{find-}\leqslant\textbf{1} \qquad \frac{i \notin \Delta}{\Delta \vdash \mathtt{find}(i;i \leqslant I) \downarrow (I;\top)}\ \textbf{find-}\leqslant\textbf{2}$$

$$\frac{i \notin \Delta}{\Delta \vdash \mathtt{find}(i;I \doteq i) \downarrow (I;\top)}\ \textbf{find-}=\textbf{1} \qquad \frac{i \notin \Delta}{\Delta \vdash \mathtt{find}(i;i \doteq I) \downarrow (I;\top)}\ \textbf{find-}=\textbf{2}$$

$$\frac{\Delta \vdash \mathtt{find}(i;\Phi_2) \downarrow (I;\Phi_2') \qquad \star \in \{\wedge, \vee\}}{\Delta \vdash \mathtt{find}(i;\Phi_1 \star \Phi_2) \downarrow (I;\Phi_1 \star \Phi_2')}\ \textbf{find-}\star\textbf{1}$$

$$\frac{\Delta \vdash \mathtt{find}(i;\Phi_1) \downarrow (I;\Phi_1') \qquad \star \in \{\wedge, \vee\}}{\Delta \vdash \mathtt{find}(i;\Phi_1 \star \Phi_2) \downarrow (I;\Phi_1' \star \Phi_2)}\ \textbf{find-}\star\textbf{2}$$

$$\frac{i', \Delta \vdash \mathtt{find}(i;\Phi) \downarrow (I;\Phi')}{\Delta \vdash \mathtt{find}(i;\forall i' :: S.\Phi) \downarrow (I;\forall i' :: S.\Phi')}\ \textbf{find-}\forall$$

$$\frac{i, \Delta \vdash \mathtt{elim}_\exists(\Phi) \downarrow \Phi' \qquad \Delta \vdash \mathtt{find}(i;\Phi') \downarrow (I;\Phi'') \qquad \mathrm{FIV}(I) \subseteq \Delta}{\Delta \vdash \mathtt{elim}_\exists(\exists i :: S.\Phi) \downarrow \Phi''[I/i]}\ \textbf{elim-}\exists$$

$$\frac{i, \Delta \vdash \mathtt{elim}_\exists(\Phi) \downarrow \Phi'}{\Delta \vdash \mathtt{elim}_\exists(\forall i :: S.\Phi) \downarrow \forall i :: S.\Phi'}\ \textbf{elim-}\forall$$

$$\frac{\Delta \vdash \mathtt{elim}_\exists(\Phi_1) \downarrow \Phi_1' \qquad \Delta \vdash \mathtt{elim}_\exists(\Phi_2) \downarrow \Phi_2' \qquad \star \in \{\wedge, \vee\}}{\Delta \vdash \mathtt{elim}_\exists(\Phi_1 \star \Phi_2) \downarrow \Phi_1' \star \Phi_2'}\ \textbf{elim-}\star$$

$$\frac{\Delta \vdash \mathtt{elim}_\exists(\Phi_2) \downarrow \Phi_2'}{\Delta \vdash \mathtt{elim}_\exists(\Phi_1 \to \Phi_2) \downarrow \Phi_1 \to \Phi_2'}\ \textbf{elim-}\to$$

Figure 59: The rules for eliminating existential variables

lemma that solves a general, relevant recurrence related to costs. This lemma is proven in the appendix (Lemma 62 in Appendix D.1).

14.4 CASE STUDIES

In this section, we evaluate applicability of our bidirectional typechecking technique on several example programs. Since relational cost analysis is a new verification problem, there are yet no real-world applications or set of benchmarks we can use. Furthermore, Cost^{ML} is a research language that lacks many useful features for realistic applications (e. g. input-output, exceptions, state, etc.). Therefore, we have chosen to evaluate the analysis and the typechecker using a small but representative set of benchmarks that we have developed ourselves.

In the rest of this section, we first describe how our bidirectional typechecker can typecheck two example programs using the heuristics described in Section 14.1. Then, we present a list of benchmark programs along with an experimental evaluation.

We first list some conventions.

- Because our rules for typing fixpoints (e. g. **r-fix**) apply at types such as $\tau_1 \xrightarrow{\text{diff}(t)} \tau_2$, but not at more general types $\forall i \overset{\text{diff}(t')}{::} S.\, \tau_1 \xrightarrow{\text{diff}(t)} \tau_2$, a recursive function whose type should have been $\forall i \overset{\text{diff}(t')}{::} S.\, \tau_1 \xrightarrow{\text{diff}(t)} \tau_2$ may have to be given the type $\text{unit}_r \xrightarrow{\text{diff}(0)} \forall i \overset{\text{diff}(t')}{::} S.\, \tau_1 \xrightarrow{\text{diff}(t)} \tau_2$. Its first argument is a dummy. When this happens, we explicitly write the unit_r type. A similar adjustment is necessary for unary fixpoints as well.
- We write $\lambda x.e$ for $\text{fix}\, f(x).e$ when f does not appear in e.
- We use pattern matching syntax for pairs and let bindings, which is easily encoded:
  e.g., $\lambda(x, y).\, e \triangleq (\lambda z.\, \text{let}\, x = \pi_1 z\, \text{in let}\, y = \pi_2 z\ \text{in}\ e)$.
- When the annotation t is omitted from types $\tau_1 \xrightarrow{\text{diff}(t)} \tau_2$ and $\forall i \overset{\text{diff}(t)}{::} S.\, \tau$, it defaults to 0 (similarly for unary costs).

14.4.1 *Heuristics illustrated*

We explain how we type two examples—the standard list map function, and the merge sort function—using our implementation. The goal of this exercise is primarily to illustrate some of our heuristics.

EXAMPLE (MAP) We describe how the map example from Chapter 3 type checks in BiRelCost.

$$\Lambda.\mathtt{fix}\ \mathtt{map}(\mathtt{f}).\Lambda.\Lambda.\lambda\mathtt{l}.\ \mathtt{case}\ \mathtt{l}\ \mathtt{of}\ \mathtt{nil}\ \to \mathtt{nil} \mid \mathtt{h} :: \mathtt{tl} \to \mathtt{cons}(\mathtt{f}\ \mathtt{h},\ \mathtt{map}\ \mathtt{f}[]\,[]\ \mathtt{tl})$$

Our aim is to type this function relative to itself at the following type:

$$\forall \mathrm{t}.(\Box\,(\tau_1 \xrightarrow{\mathrm{diff}(\mathrm{t})} \tau_2)) \to \forall n, \alpha.\mathrm{list}[n]^{\alpha}\,\tau_1 \xrightarrow{\mathrm{diff}(\mathrm{t}\cdot\alpha)} \mathrm{list}[n]^{\alpha}\,\tau_2 \tag{5}$$

We start in the checking mode with the above type. Using rule **alg-r-iLam-↓**, we introduce the index variable t into the context. We continue in the checking mode using rule **alg-r-fix-↓**. Next, we add the function $\mathtt{f}$ and the input list $\mathtt{l}$ to the typing context and the index variables n and α to the sort context. For typechecking the pattern match on the list $\mathtt{l}$, we use the rule **alg-r-caseL-↓**. BiRelCost first infers that the type of $\mathtt{l}$ is $\mathrm{list}[n]^{\alpha}\,\tau_1$ using the rule **r-var-↑**. The nil-branch is straightforwardly typechecked with 0 cost. We focus here on cons-branch, considering the two cases where the heads of the two lists may differ or may not differ (third and fourth promises in rule **alg-r-caseL-↓**). In both cases, we aim to check that $\mathtt{cons}(\mathtt{f}\ \mathtt{h},\ \mathtt{map}\ \mathtt{f}[]\,[]\ \mathtt{tl})$ can be given type $\mathrm{list}[n]^{\alpha}\,\tau_2$ using the heuristic (2) for cons-ed lists.

In the first case where the heads of the two lists may not differ, we have $\mathtt{h} : \Box\tau_1$ and $\mathtt{tl} : \mathrm{list}[i]^{\alpha}\,\tau_1$, where $n = i + 1$ for some freshly-generated meta variable i. Following heuristic (2), we try to type "$\mathtt{f}\ \mathtt{h}$" in checking mode first with type $\Box\tau_2$ and then with type τ_2 (corresponding to the rules **alg-r-consNC-↓** and **alg-r-consC-↓**). For the sake of brevity, we focus on the former, which succeeds and generates constraints that are satisfiable.[41] Since function applications (like "$\mathtt{f}\ \mathtt{h}$") are typed in inference mode, we switch from checking to inference mode using the rule **alg-r-↑↓**. Then, we proceed to type the function application "$\mathtt{f}\ \mathtt{h}$" in inference mode where we can infer that $\mathtt{f} : \Box\,(\tau_1 \xrightarrow{\mathrm{diff}(\mathrm{t})} \tau_2)$. However, the function $\mathtt{f}$ cannot be directly applied since it has a $\Box$-ed type. At this point, using the heuristic (3), we subtype $\Box\,(\tau_1 \xrightarrow{\mathrm{diff}(\mathrm{t})} \tau_2)$ to $\Box\tau_1 \xrightarrow{\mathrm{diff}(0)} \Box\tau_2$. Then, we can type the argument $\mathtt{h}$ in the checking mode with type $\Box\tau_1$ successfully and conclude that "$\mathtt{f}\ \mathtt{h}$" can be given the type $\Box\tau_2$ with relative cost 0. Next, we aim to show that the tail of the cons, i.e. "$\mathtt{map}\ \mathtt{f}[]\,[]\ \mathtt{tl}$", can be typed in checking mode with type $\mathrm{list}[i']^{\alpha}\,\tau_2$ and cost $\alpha \cdot \mathrm{t}$ for some i' such that $n = i' + 1$. As in the head case, we use the rule **alg-r-↑↓** to switch from checking to inference mode and generate the constraint $i' = i$ (which can be proved easily

[41] *The latter case also succeeds here, but it generates constraints that cannot be satisfied.*

since $n = i' + 1 = i + 1$). The rest of the typing can be concluded by invoking the function and index term application rules multiple times.

In the second case where the heads of the two lists may differ, we have $h : \tau_1$ and $tl : \text{list}[i]^{\beta}\,\tau_1$, where $n = i + 1$ and $\alpha = \beta + 1$ for some i and β. Similar to above, we try to typecheck "f h" first at type $\square\tau_2$ and then at type τ_2. This time, only the latter case succeeds and generates satisfiable constraints. In this case, like above, the function application "f h" is typed in inference mode, where we can infer that $f : \square(\tau_1 \xrightarrow{\text{diff}(t)} \tau_2)$. However, this time, we subtype $\square(\tau_1 \xrightarrow{\text{diff}(t)} \tau_2)$ to $\tau_1 \xrightarrow{\text{diff}(t)} \tau_2$ (again using heuristic (3)) and type the argument h in the checking mode at type τ_1. The total cost of the application is t, not 0. The recursive call to `map` follows a similar reasoning as in the previous case, but with cost $\beta \cdot t$. Hence, the total cost is $t + \beta \cdot t = (\beta + 1) \cdot t = \alpha \cdot t$, as required.

EXAMPLE (MERGE SORT) Next, to illustrate how we use heuristics (1) and (4), we consider merge sort, a divide and conquer algorithm which has a somewhat nontrivial relational cost. The merge sort function, `msort`, splits a list into two nearly equal-sized sublists using the function `bsplit`, recursively sorts each sublist and then merges the two sorted sublists using the function `merge`. The relative cost of two runs of `msort` with two input lists of length n that differ in at most α positions is $Q(n, \alpha) = \sum_{i=0}^{H} h(\lceil \frac{2^i}{2} \rceil) \cdot \texttt{min}(\alpha, 2^{H-i})$, where $H = \lceil \texttt{log}_2(n) \rceil$. This open-form expression lies in $O(n \cdot (1 + \texttt{log}_2(\alpha)))$.[42] Next, we explain at a high-level how this relative cost bound is typechecked bidirectionally.

[42] *The analysis of* `msort` *is similar to* `bfold`*'s analysis in Chapter 7. It is shown in Lemma 64.*

```
fix msort(_).Λ.Λ.λl.case l of
  nil  → nil
| h1 :: tl1  → case tl1 of
                  nil  → cons(h1, nil )
                | _ :: _  → let r = bsplit ()[][] l in
                            unpack r as r′ in
                            clet r′ as (z1, z2) in
                            merge ()[][] (msort ()[][] z1, msort ()[][] z2)
```

First, let us assume that we can typecheck the helper functions `bsplit` and `merge` at the following types. This typechecking has been done with our implementation, but we do not explain its details here.

$$\texttt{bsplit} : \square(\text{unit}_r \to \forall n, \alpha{::}\mathbb{N}.\, \text{list}[n]^{\alpha}\,\tau \xrightarrow{\text{diff}(0)} \exists \beta{::}\mathbb{N}.\, \beta \leqslant \alpha \;\&\; (\text{list}[\lceil \tfrac{n}{2} \rceil]^{\beta}\,\tau \times \text{list}[\lfloor \tfrac{n}{2} \rfloor]^{\alpha-\beta}\,\tau))$$

$$\texttt{merge} : \Box\,(\mathrm{U}\,(\mathrm{unit} \to \forall n, m{::}\mathbb{N}.\,(\mathrm{list}[n]\,\mathrm{int} \times \mathrm{list}[m]\,\mathrm{int}) \xrightarrow{\mathrm{exec}(h(\texttt{min}(n,m)),h(n+m))} \mathrm{list}[n+m]\,\mathrm{int}))$$

(Note the $\Box$ outside the types; their significance will be clear soon). Our aim is to typecheck `msort` relative to itself at the following type:

$$\Box\,(\mathrm{unit}_r \to \forall n, \alpha{::}\mathbb{N}.\,\mathrm{list}[n]^{\alpha}\,(\mathrm{U}\,\mathrm{int}) \xrightarrow{\mathrm{diff}(Q(n,\alpha))} \mathrm{U}\,(\mathrm{list}[n]\,\mathrm{int}))$$

We focus on the most interesting part where we call `merge` on the results of the two recursive calls to `msort`. At this point, we have $z_1 : \mathrm{list}[\lceil \frac{n}{2} \rceil]^{\beta}\,(\mathrm{U}\,\mathrm{int})$ and $z_2 : \mathrm{list}[\lfloor \frac{n}{2} \rfloor]^{\alpha-\beta}\,(\mathrm{U}\,\mathrm{int})$ (from the type of `bsplit`). Considering that only the calls to `merge` and `msort` incur additional costs (all the remaining operations occur synchronously on both sides and the relative cost of `bsplit` is 0 from its type), if we were to naively establish the bound $Q(n, \alpha)$, we would have to show the following inequality:

$$h(\left\lceil \frac{n}{2} \right\rceil) + Q(\left\lceil \frac{n}{2} \right\rceil, \beta) + Q(\left\lfloor \frac{n}{2} \right\rfloor, \alpha - \beta) \leqslant Q(n, \alpha) \tag{6}$$

where the cost $h(\lceil \frac{n}{2} \rceil) = h(n) - h(\mathrm{min}(\lceil \frac{n}{2} \rceil, \lfloor \frac{n}{2} \rfloor))$ comes from the relative cost of `merge`. However, this inequality holds only when $\alpha > 0$. When $\alpha = 0$, the right hand side is 0 whereas the left hand side is $h(\lceil \frac{n}{2} \rceil)$. Nevertheless, when $\alpha = 0$, the two input lists do not differ at all, so the relative cost of `merge` can be trivially established as 0 using the **nochange** rule.

Consequently, the verification of `merge`'s body differs based on whether $\alpha = 0$ or not. In our implementation, this case analysis is provided for by heuristic (1). As soon as the list `l` is introduced into the context, we apply the algorithmic rule **alg-r-split**$\downarrow$, introducing the two cases $\alpha = 0$ and $\alpha > 0$. For the case $\alpha = 0$, we immediately invoke the rule **alg-r-nochange**$\downarrow$, which requires us to show that all the free variables in the context have $\Box$-ed types. Since we know that the functions `merge`, `bsplit` and `msort` are all $\Box$-ed, what remains to be shown is that $\mathrm{list}[n]^{\alpha}\,\tau_1 \sqsubseteq \Box\,(\mathrm{list}[n]^{\alpha}\,\tau_1)$. Using the algorithmic subtyping rule **alg-list-$\Box$** in heuristic (4), this can be shown when $\alpha = 0$. For the case $\alpha > 0$, we proceed with the usual typing of the function body, which eventually generates the satisfiable constraint (6).

14.5 EXPERIMENTAL EVALUATION

We have used our implementation to typecheck a set of programs, including all the examples shown so far and additional ones in Appendix D.2. Some of the examples, such as the relational analysis of merge sort (`msort`), have rather complex paper proofs. However, in all cases, the total typechecking time (including existential elimination and SMT solving) is less than 1s, suggesting that the approach is practical. Table 1 shows the experimental results over a subset of our example programs (our appendix lists all our example programs, including their code and experimental results). A "-" indicates a negligible value. Our experiments were performed on a 3.20GHz 4-core Intel Core i5-6500 processor with 16 GB of RAM.

We briefly describe the example programs in Table 1. Appendix D.2 contains the code and the types for all of the example programs.

list operations The programs `map` and `filter` are the usual list map and filter functions. The program `append` takes two lists and returns the first list appended to the second list. The program `rev` reverses a list using an accumulating parameter. The program `flatten` takes a list of lists and flattens them. The program `zip` takes two lists of the same length and returns a list of pairs where the projections are taken from the two lists. The program `shuffle` takes a list and shuffles its elements deterministically by reversing its tail at each recursive call. The benchmark `foldCmp` compares the relative costs of standard fold functions `foldr` and `foldl`.

examples from Chapter 4 The program `comp` is a constant-time (0 relative cost) comparison function that checks the equality of two passwords, represented as lists of bits. The program `sam` (square-and-multiply) computes the positive powers of a number, represented as a list of bits. The program `find` compares two functions that find a given element by scanning a list from head to tail and tail to head, respectively. The program `2Dcount` counts the number of rows of a matrix, represented as a list of lists in row-major form, that satisfy a predicate `p` and contain a key `x`. The program `bsplit` splits a list into two nearly equal length lists. The program `merge` merges two sorted lists and the program `msort` is the standard merge sort function.

additional examples The program `ssort` is the standard selection sort function. The program `bfold` is the balanced fold function explained in Chapter 7. The program `appSum` is a program that

compares two implementations of summing over a list: one exact and the other approximate.

In all cases except `find`, `foldCmp` and `appSum`, the goal of the analysis is to find an upper bound on the relative cost of the same function as its input changes. In `find`, `foldCmp` and `appSum`, we compare slightly similar programs using also the asynchronous rules. In all example programs, the bounds we obtain via type-checking are asymptotically tight.

Benchmark	Total time	Type-checking	Existential elimination	Constraint solving
`map`	0.11	-	-	0.11
`filter`	0.13	-	-	0.13
`append`	0.14	-	-	0.13
`rev`	0.15	-	-	0.15
`flatten`	0.06	-	-	0.05
`zip`	0.13	-	-	0.13
`shuffle`	0.14	-	-	0.13
`foldCmp`	0.13	-	-	0.12
`comp`	0.08	-	-	0.07
`sam`	0.09	0.01	-	0.08
`find`	0.05	-	-	0.04
`2Dcount`	0.06	-	-	0.06
`bsplit`	0.17	-	-	0.17
`merge`	0.12	-	-	0.12
`msort`	0.40	0.01	0.02	0.36
`bfold`	0.77	-	0.01	0.77
`appSum`	0.10	-	-	0.09
`ssort`	0.07	-	-	0.07

Table 1: BiRelCost runtime on benchmarks. All times are in seconds.

ANNOTATION EFFORT AND USABILITY In a traditional bidirectional type system, the programmer's annotation effort is limited to providing the eliminated type at every explicit β-redex and the type of every top-level function definition in the program. In our setting, the burden is similar, except that type annotations on functions also include a cost

(on the arrow). In all but one of the examples we have tried, annotations are only necessary at each top-level function. One example has an explicit beta-redex (in the form of a let-binding) and needs an additional annotation.

To give an idea of the annotation effort needed in our benchmarks, Table 2 presents total number of lines of code in each benchmark program along with the number of lines of type and cost annotations.[43]

[43] *For all benchmark programs, if the two related programs are identical, we only count the line numbers of one.*

Benchmark	Total # of lines	# of lines of annotations
`map`	5	2
`filter`	7	2
`append`	6	2
`rev`	6	2
`flatten`	13	4
`zip`	8	2
`shuffle`	10	3
`foldCmp`	8	2
`comp`	8	2
`sam`	10	2
`find`	8	2
`2Dcount`	14	3
`bsplit`	10	2
`merge`	10	2
`msort`	33	6
`bfold`	32	6
`appSum`	14	1
`ssort`	18	4

Table 2: BiRelCost number of lines of benchmarks.

In our experience, the typechecker is quite usable and error reporting is generally useful. The prototype tool points out the location of parsing and typechecking errors for majority of the cases quite accurately except when there is an error in the constraint solving. We leave improving this aspect to the future work.

15

RELATED WORK : BIDIRECTIONAL RELATIONAL COST ANALYSIS

There is a lot of literature on type checking various combinations of lightweight dependent types, effect systems, comonadic types, and subtyping. However, a distinctive feature of BiRelCost is that it combines all these aspects in a relational setting with support for real numbers. This chapter briefly surveys the closely related work in the following two areas: refinement types and bidirectional typechecking.

15.1 DEPENDENT/REFINEMENT TYPES

There is enormous literature on dependent types, which is witnessed by the ever increasing list of dependently typed languages such as Cayenne [12], Epigram [77], Omega [100], DML [105], Coq [19], Agda [82] and Idris [23]. We do not attempt to survey this vast field but instead we focus on the most relevant precursors to our work, namely refinement types.

Like DML, we use a bidirectional typechecking algorithm and generate arithmetic constraints that must be satisfied for program to be typed [105, 106]. However, there are several differences that we would like to note. First, the index domain considered by DML is integers with linear inequalities, whereas in BiRelCost, due to the costs, we additionally consider reals with non-linear inequalities. Second, DML lacks comonadic types, costs, and relational types. The challenges we face in BiRelCost come mostly from these components.

TYPECHECKING LINEAR DEPENDENT TYPES The DML approach has also been used by [39] in combination with linear types for asymptotic complexity analysis. A type checker for this approach is presented by [41]. Similarly, [51]'s DFuzz use a combination of linear types and lightweight dependent types for reasoning about differential privacy. A type checker for DFuzz is presented by [9] in which a program is typechecked in two steps: first by inferring a type (along with the sensitivities) and then checking whether the resulting type is a subtype of the desired type. Besides lightweight dependent types, these papers also consider the comonadic modality of linear logic. This modality's struc-

tural properties are quite different from those of RelCost's □. Moreover, none of these papers consider relational types.

TYPECHECKERS WITH SUPPORT FOR RELATIONAL REASONING Some other type systems establish relational properties of programs. For instance, [14] consider a relational variant of a fragment of F* for the verification of cryptographic implementations, while [15] consider a relational refinement type system for differential privacy. However, some of the key technical challenges of BiRelCost, including those that arise from the interaction between unary and relational typing, as well as costs, do not show up in these settings. Moreover, these systems use verification condition generation, not bidirectionality.

15.2 BIDIRECTIONAL TYPECHECKING

The idea of bidirectional type systems appeared in literature early on. The idea was popularized only more recently by Pierce and Turner [91]. The technique has shown great applicability—it has been used for dependent types [36], indexed and refinement types [105, 106], intersection and union types [44, 48], higher-rank polymorphism [46, 47, 89], contextual modal types [90] and most recently for effect handlers [74]. The design of BiRelCost is inspired by many of these papers but departs in the technical design of the algorithmic type system due to new challenges offered by relational and modal types, and unary and relational costs. In particular, in all the previous work on bidirectional typechecking, the reasoning principle is *unary*, i.e. a single program is checked (inferred) in isolation. Moreover, almost all the previous work on bidirectional typechecking does not track effects explicitly: e. g. DML supports effects like exceptions, but there is no corresponding effect system for statically tracking the effects. One exception is the bidirectional effect system of [101], which uses bidirectional typechecking for gradual unary effects. However, their end goal is different since they infer minimal effects at compile time and then check dynamic effects at runtime.

15.2.1 *Elimination of subtyping*

Prior work has also studied methods of eliminating subtyping as a way of simplifying type checking, e.g. [24, 37]. While the approach this thesis takes is similar in motivation, the technical challenges are quite different. The main difficulties in simplifying subtyping in our

work arise from the interaction of the modalities $\Box$ and $\mathcal{U}$ with other connectives.

Part IV

EPILOGUE

16

CONCLUSION

▶ SYNOPSIS In this chapter, we conclude the thesis by reviewing our contributions and pointing out several directions for further research.

This thesis introduces the problem of relational cost analysis—establishing relative costs of programs, relationally. In particular, we demonstrate four claims:

- Relational cost analysis can be *understood and enhanced* through reasoning about relational properties of programs, which enables verification that is more precise and local compared to a naive unary cost analysis.
- Relational costs can be *formalized* syntactically through a combination of *unary* and *relational* refinement type and effect systems, and semantically through a combination of *unary* and *relational* logical relations.
- Relational cost analysis can be *applied* not only to compare program costs but also in the setting of incremental computations in which the underlying evaluation semantics is much more complex.
- Relational costs can be *verified* through bidirectional typechecking, which can be implemented.

The thesis supports these claims with the following three distinct research artifacts:

- RelCost: a type theory for reasoning about relational costs. The approach enables proofs of relative cost bounds via relational refinement type and effect systems, and scales to high-level languages (with higher-order functions and recursion).
- DuCostIt: a type theory for reasoning about update times of incremental programs. The approach enables proofs of dynamic stability via an abstract change propagation semantics, relational refinement types and effect systems.

- BiRelCost: an algorithmic typechecking mechanism and a prototype implementation for verifying upper bounds on relative costs (as well as incremental update times). The approach demonstrates that bidirectional typechecking scales to relational reasoning in the existence of unary and relational effects. Using the bidirectional typechecker, programmers can verify relative cost bounds with minimal annotations.

Combined together, these contributions make a significant step forward in our understanding of verification of quantitative execution cost bounds *not only for one program but also for a pair of related programs*. Still, as with any static analysis technique, our relational cost analysis has many limitations. Some of the limitations result from the trade-offs we made initially to ensure the practicality of typechecking, and some can be eliminated through further research and development.

16.1 FUTURE WORK

In this section, we point out several possible research directions for improving relational cost analysis.

16.1.1 *Embedding functional equivalences*

We designed RelCost to reason about the execution cost differences of programs relationally. Although our relational analysis is powerful enough to analyze a wide variety of examples, there are programs whose analysis requires more involved reasoning such as the ability to benefit from functional equivalences or the ability to relationally reason about index terms. As an example, consider the sieve of Eratosthenes, a standard algorithm for finding all prime numbers up to a given integer n. There are several variations of this algorithm, but the main idea is to start with a list of all natural numbers that are less than or equal to n and then repeatedly drop all the composites until all the remaining numbers are the primes. Below, we only show the top level function `erat` that takes as input the list `l` containing the values $[2, 3, \ldots, n]$. The function `drop` drops all multiples of its numerical first argument from its second argument, which is a list of natural numbers.

```
fix erat(drop).λl.case l of
  nil  → nil
| h :: tl  →  cons(h, erat drop (drop h tl))
```

Suppose that `drop` is implemented in two functionally equivalent ways but the execution costs of these two versions are different. To establish a precise bound on the relative cost of `erat` with respect to these two implementations of `drop`, we would need to show that two versions of `drop` are functionally equivalent. Such reasoning is not possible in RelCost. This is not an inherent limitation, but a design choice we made to simplify the type system. We believe our analysis can be extended to more expressive relations, by building on previous work on relational refinement types [14, 16], which can be used for capturing the necessary invariants. However, the more involved the relational invariants, the more difficult it is for non-experts to use our analysis and perhaps to automate the type-checking. In this thesis, we have chosen a lightweight form of relational reasoning that still yields a powerful analysis.

16.1.2 *Allowing relational reasoning on index terms*

In RelCost's and DuCostIt's relational typing, size and cost refinements are assumed to be identical for the two related programs. For the examples we have considered such an assumption is sufficient, but allowing relations on index terms could enable more fine-grained analysis and increase the set of examples we can analyze.

For instance, in the `map` example in Chapter 7, we assume that the two lists have the same length, whereas an analysis that allows the two lists to have different lengths should be possible without disrupting the relational reasoning. Indeed, in the context of incremental computation, such an extended analysis could be used to show that the dynamic stability of `map` with insertions and deletions is still linear in the number of changes.

16.1.3 *Support for algebraic datatypes*

A natural extension to both RelCost and DuCostIt is adding support for user-defined algebraic datatypes. The main theoretical challenge is in describing general size functions and expressing costs with them. Currently, size and changeability refinements in DuCostIt's and RelCost's types are data-structure specific. Ideally, there should be a more generic framework in which the programmer can specify a particular size metric along with each algebraic data type. For instance, our types can be easily extended with trees that are refined with the number of nodes. Depending on the application, there could be cases where the depth of

a tree is a useful size metric. In the non-relational setting, existing work by Danner et al. considers such generalizations [43].

In RelCost's semantic model, we have anticipated generalization to recursive types—our step-indexed logical relations are capable of modeling recursive types. However, we have not yet worked out the generalization to recursive types with user-defined size or difference metrics.

16.1.4 *Reasoning about non-termination and co-inductive types*

In our relational cost framework, the bounds on the relative costs (as well as incremental costs) are valid for terminating programs—our semantic model is set up as such. In addition, the data structures we consider are finite. However, reasoning about the relative cost of two non-terminating programs can also be useful for interactive or reactive programs like web servers. Extending our analysis to reason about non-terminating programs (e. g. that operate on streams) is non-trivial, but it is an interesting future direction. For such an extension, we anticipate the use of bisimulation-based proof techniques that can reason about co-inductive data structures.

16.1.5 *Support for effectful programs*

Another future direction is to extend our language and type theory to effectful programs. Possible kinds of effects include state, probability and exceptions. One of the challenges in supporting effectful programs is tracking multiple effects at the same time. Recent research on extensible algebraic effects and their lifting to type and effect systems are promising directions [65].

16.1.6 *Support for polymorphism*

Polymorphism allows abstracting expressions over types and is a useful feature for increasing code reuse. A prior version of DuCostIt, CostIt, had support for polymorphism but we have not specifically addressed polymorphism in RelCost and DuCostIt. Although the development of relational cost analysis is orthogonal to polymorphism and technically straightforward, it would be nonetheless useful to add support for polymorphism for pragmatic reasons.

16.1.7 *Different kinds of resources and support for state*

Our relational cost model can also be adapted to track different kinds of resources. For instance, our model can be modified to track the span or work of parallel programs [62, 102]. Alternatively, the resource model can be adapted to track resources other than execution time such as space, energy usage or trace size of incremental programs. We believe that adding support for modifiable state would also be useful in this regard.

16.1.8 *Different reduction strategies*

Execution cost of a program, or resource usage, depends on the underlying reduction strategy. In our setting, the underlying language $\mathsf{Cost}^{\mathsf{ML}}$ has a call-by-value evaluation semantics. Consequently, the effects in our type and effect systems, DuCostIt and RelCost, can be interpreted in the monadic setting where the monad of the computational lambda calculus is graded with the cost (effect) as in [64]. On the other hand, in call-by-name languages, costs are often represented as *coeffects* which can be interpreted as the logical dual of a monad—-the comonad—in conjunction with a linear type system [67]. One possible direction is to investigate relational cost analysis in the context of a call-by-name language. Another potential direction in investigation of reduction strategies is to consider call-by-push value which subsumes both call-by-value and call-by-name [72].

16.2 FUTURE IMPLEMENTATIONS

The current prototype implementation of BiRelCost could be improved in many ways. For instance, currently we collect all the constraints generated during typechecking before we pass them to a constraint solver. However, it might be possible to intertwine constraint generation with constraint solving. The advantages of this alternative design are twofold. First, failure in solving the constraints can be detected earlier. Second, error reporting would improve since the provenance can be tracked to the exact location of failing constraints, which is not possible at the moment.

Part V

APPENDIX

A

APPENDIX FOR RELCOST

In this chapter, we first describe the necessary definitions, lemmas and theorems for proving the soundness of the RelCost's unary and binary (relational) typing with respect to the abstract cost semantics.

We use some abbreviations throughout. STS stands for "suffices to show", TS stands for "to show", and RTS stands for "remains to show".

$\boxed{\Delta \vdash \tau\ \mathtt{wf}}$ Relational type τ is well-formed.
$\boxed{\Delta \vdash^{A} A\ \mathtt{wf}}$ Type A is well-formed.

$$\frac{}{\Delta \vdash \mathrm{unit}_r\ \mathtt{wf}}\ \textbf{wf-unit} \qquad \frac{}{\Delta \vdash \mathrm{int}_r\ \mathtt{wf}}\ \textbf{wf-int}$$

$$\frac{\Delta \vdash \tau_1\ \mathtt{wf} \qquad \Delta \vdash \tau_2\ \mathtt{wf}}{\Delta \vdash \tau_1 \times \tau_2\ \mathtt{wf}}\ \textbf{wf-prod} \qquad \frac{\Delta \vdash \tau_1\ \mathtt{wf} \qquad \Delta \vdash \tau_2\ \mathtt{wf}}{\Delta \vdash \tau_1 + \tau_2\ \mathtt{wf}}\ \textbf{wf-sum}$$

$$\frac{\Delta \vdash \tau_1\ \mathtt{wf} \qquad \Delta \vdash \tau_2\ \mathtt{wf} \qquad \Delta \vdash t :: \mathbb{R}}{\Delta \vdash \tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2\ \mathtt{wf}}\ \textbf{wf-fun}$$

$$\frac{\Delta \vdash n :: \mathbb{N} \qquad \Delta \vdash \alpha :: \mathbb{N} \qquad \Delta \vdash \tau\ \mathtt{wf}}{\Delta \vdash \mathrm{list}[n]^{\alpha}\,\tau\ \mathtt{wf}}\ \textbf{wf-list}$$

$$\frac{i :: S, \Delta \vdash \tau\ \mathtt{wf} \qquad i :: S, \Delta \vdash t :: \mathbb{R}}{\Delta \vdash \forall i \overset{\mathrm{diff}(t)}{::} S.\,\tau\ \mathtt{wf}}\ \textbf{wf-}\forall \qquad \frac{i :: S, \Delta \vdash \tau\ \mathtt{wf}}{\Delta \vdash \exists i{::}S.\,\tau\ \mathtt{wf}}\ \textbf{wf-}\exists$$

$$\frac{\Delta \vdash^{A} A_1\ \mathtt{wf} \qquad \Delta \vdash^{A} A_2\ \mathtt{wf}}{\Delta \vdash U\,(A_1, A_2)\ \mathtt{wf}}\ \textbf{wf-U} \qquad \frac{\Delta; \Phi \vdash \tau\ \mathtt{wf}}{\Delta \vdash \Box\,\tau\ \mathtt{wf}}\ \textbf{wf-box}$$

$$\frac{\Delta \vdash C\ \mathtt{wf} \qquad \Delta; C \wedge \Phi \vdash \tau\ \mathtt{wf}}{\Delta \vdash C \supset \tau\ \mathtt{wf}}\ \textbf{wf-C}\supset$$

$$\frac{\Delta \vdash C\ \mathtt{wf} \qquad \Delta; C \wedge \Phi \vdash \tau\ \mathtt{wf}}{\Delta \vdash C\ \&\ \tau\ \mathtt{wf}}\ \textbf{wf-C\&}$$

Figure 60: Well-formedness of relational types

$\boxed{\Delta \vdash^{A} A\ \mathtt{wf}}$ Type A is well-formed.

$$\frac{}{\Delta \vdash^{A} \mathrm{unit}\ \mathtt{wf}}\ \textbf{wf-u-unit} \qquad \frac{}{\Delta \vdash^{A} \mathrm{int}\ \mathtt{wf}}\ \textbf{wf-u-int}$$

$$\frac{\Delta \vdash^{A} A_1\ \mathtt{wf} \qquad \Delta \vdash^{A} A_2\ \mathtt{wf}}{\Delta \vdash^{A} A_1 \times A_2\ \mathtt{wf}}\ \textbf{wf-u-prod}$$

$$\frac{\Delta \vdash^{A} A_1\ \mathtt{wf} \qquad \Delta \vdash^{A} A_2\ \mathtt{wf}}{\Delta \vdash^{A} A_1 + A_2\ \mathtt{wf}}\ \textbf{wf-u-sum}$$

$$\frac{\Delta \vdash^{A} A_1\ \mathtt{wf} \qquad \Delta \vdash^{A} A_2\ \mathtt{wf} \qquad \Delta \vdash k :: \mathbb{R} \qquad \Delta \vdash t :: \mathbb{R}}{\Delta \vdash^{A} A_1 \xrightarrow{\mathrm{exec}(k,t)} A_2\ \mathtt{wf}}\ \textbf{wf-u-fun}$$

$$\frac{\Delta \vdash n :: \mathbb{N} \qquad \Delta \vdash^{A} A\ \mathtt{wf}}{\Delta \vdash^{A} \mathrm{list}[n]\ A\ \mathtt{wf}}\ \textbf{wf-u-list}$$

$$\frac{i :: S, \Delta \vdash^{A} A\ \mathtt{wf} \qquad i :: S, \Delta \vdash k :: \mathbb{R} \qquad i :: S, \Delta \vdash t :: \mathbb{R}}{\Delta \vdash^{A} \forall i \overset{\mathrm{exec}(k,t)}{::} S.\, A\ \mathtt{wf}}\ \textbf{wf-u-}\forall$$

$$\frac{i :: S, \Delta \vdash^{A} A\ \mathtt{wf}}{\Delta \vdash^{A} \exists i{::}S.\, A\ \mathtt{wf}}\ \textbf{wf-u-}\exists \qquad \frac{\Delta \vdash C\ \mathtt{wf} \qquad \Delta;\vdash^{A} A\ \mathtt{wf}}{\Delta \vdash^{A} C \supset A\ \mathtt{wf}}\ \textbf{wf-u-C}\supset$$

$$\frac{\Delta \vdash C\ \mathtt{wf} \qquad \Delta;\vdash^{A} A\ \mathtt{wf}}{\Delta \vdash^{A} C\ \&\ A\ \mathtt{wf}}\ \textbf{wf-u-C\&}$$

Figure 61: Well-formedness of types

$\boxed{\Delta \vdash C\ \mathtt{wf}}$

$$\frac{\Delta \vdash I_1 :: S \qquad \Delta \vdash I_2 :: S \qquad S \in \{\mathbb{N}, \mathbb{R}\}}{\Delta \vdash I_1 < I_2\ \mathtt{wf}}\ \textbf{wf-cs}\ < \qquad \frac{\Delta \vdash I_1 :: S \qquad \Delta \vdash I_2 :: S \qquad S \in \{\mathbb{N}, \mathbb{R}\}}{\Delta \vdash I_1 \doteq I_2\ \mathtt{wf}}\ \textbf{wf-cs}\ \doteq$$

$$\frac{\Delta \vdash C\ \mathtt{wf}}{\Delta \vdash \neg C\ \mathtt{wf}}\ \textbf{wf-cs}\ \neg$$

Figure 62: Constraint well-formedness

A.1 RELCOST LEMMAS

Lemma 17 (Value evaluation). $v \Downarrow^{0,0} v$

Proof. Proof is by induction on the value term v. □

Lemma 18 (Value interpretation containment). *The following hold.*

1. $(m, v_1, v_2) \in (\!|\tau|\!)_v$ *then* $(m, v_1, v_2) \in (\!|\tau|\!)^0_\varepsilon$.
2. $(m, v) \in [\![A]\!]_v$ *then* $(m, v) \in [\![A]\!]^{0,0}_\varepsilon$.

Proof of (1). Assume that $(m, v_1, v_2) \in (\!|\tau|\!)_v$ $(\star)$.
TS: $(m, v_1, v_2) \in (\!|\tau|\!)^0_\varepsilon$.
Following the definition of $(\!|\tau|\!)^0_\varepsilon$, and assume that $(v_1 \Downarrow^{0,0} v_1 \wedge 0 < m)$ (cost and resulting value obtained by Lemma 17).
Then, we can immediately show

1. $v_2 \Downarrow^{0,0} v_2$ by Lemma 17
2. $0 - 0 \leqslant 0$ is trivially true.
3. $(m - 0, v_1, v_2) \in (\!|\tau|\!)_v$ follows from the main assumption $(\star)$.

□

Proof of (2). Assume that $(m, v) \in [\![A]\!]_v$ $(\star)$.
TS: $(m, v) \in [\![A]\!]^{0,0}_\varepsilon$.
Following the definition of $[\![A]\!]^{0,0}_\varepsilon$,assume that $v \Downarrow^{0,0} v$ (cost and the resulting value obtained by Lemma 17) and $0 < m$.
Then, we can immediately show

1. $0 \leqslant 0 \leqslant 0$
2. $(m - 0, v) \in [\![A]\!]_v$ which follows from the assumption $(\star)$.

□

Lemma 19 (Value Projection). *The following holds.*

1. *If* $(m, v_1, v_2) \in (\!|\tau|\!)_v$ *then* $\forall j.(j, v_1) \in [\![|\tau|_1]\!]_v$ *and* $(j, v_2) \in [\![|\tau|_2]\!]_v$.

2. *If* $(m, \delta_1, \delta_2) \in \mathcal{G}(\!|\Gamma|\!)$ *then* $\forall j.(j, \delta_1) \in \mathcal{G}[\![|\Gamma|_1]\!]$ *and* $(j, \delta_2) \in \mathcal{G}[\![|\Gamma|_2]\!]$.

Proof. Proof of statement (1) is by induction on $(\!|\tau|\!)_v$. Proof of statement (2) follows by proof of (1). □

Lemma 20 (Downward Closure)**.** *The following hold.*

1. *If* $(m, v_1, v_2) \in (\!|\tau|\!)_v$ *and* $m' \leqslant m$, *then* $(m', v_1, v_2) \in (\!|\tau|\!)_v$
2. *If* $(m, v) \in [\![A]\!]_v$ *and* $m' \leqslant m$, *then* $(m', v) \in [\![A]\!]_v$
3. *If* $(m, e_1, e_2) \in (\!|\tau|\!)^t_\varepsilon$ *and* $m' \leqslant m$, *then* $(m', e_1, e_2) \in (\!|\tau|\!)^t_\varepsilon$
4. *If* $(m, e) \in [\![A]\!]^{k,t}_\varepsilon$ *and* $m' \leqslant m$, *then* $(m', e) \in [\![A]\!]^{k,t}_\varepsilon$
5. *If* $(m, \delta_1, \delta_2) \in \mathcal{G}(\!|\Gamma|\!)$ *and* $m' \leqslant m$, *then* $(m', \delta_1, \delta_2) \in \mathcal{G}(\!|\Gamma|\!)$
6. *If* $(m, \gamma) \in \mathcal{G}[\![\Omega]\!]$ *and* $m' \leqslant m$, *then* $(m', \gamma) \in \mathcal{G}[\![\Omega]\!]$

Proof. (1,3) and (2,4) are proved simultaneously by induction on τ. (5,6) follows from (1,2).
We just show the proofs of statement (3) and (4) below.

Proof of statement (3). Assume that $(m, e_1, e_2) \in (\!|\tau|\!)^t_\varepsilon$ and $m' \leqslant m$.
TS: $(m', e_1, e_2) \in (\!|\tau|\!)^t_\varepsilon$.
By unrolling its definition, assume that $e_1 \Downarrow^{c_1,r_1} v_1$ $(\star)$ and $e_2 \Downarrow^{c_2,r_2} v_2$ $(\star\star)$ and $c < m'$ $(\diamond)$.
By $(\diamond)$ and $m' \leqslant m$, we can show that $c < m$ $(\diamond\diamond)$.
Then, we can unroll the main assumption with $(\star)$, $(\star\star)$ and $(\diamond\diamond)$ to get

a) $r_1 - r_2 \leqslant t$
b) $(m - c, v_1, v_2) \in (\!|\tau|\!)_v$

Then, we can conclude as follows

1. By a)
2. By IH 2 on b) using $m' \leqslant m$, we get $(m' - c, v_1, v_2) \in (\!|\tau|\!)_v$.

□

Proof of statement (4). Assume that $(m, e) \in [\![A]\!]^{k,t}_{\varepsilon}$ and $m' \leqslant m$.
TS: $(m', e) \in [\![A]\!]^{k,t}_{\varepsilon}$.
By unrolling its definition, assume that $e \Downarrow^{c,r} v$ $(\star)$ and $c < m'$ $(\diamond)$.
By $(\diamond)$ and $m' \leqslant m$, we can show that $c < m$ $(\diamond\diamond)$.
Then, we can unroll the main assumption with $(\star)$ and $(\diamond\diamond)$ to get

a) $k \leqslant r \leqslant t$
b) $(m - c, v) \in [\![A]\!]_v$

Then, we can conclude as follows

1. By a)
2. By IH 2 on b) using $m' \leqslant m$, we get $(m' - c, v) \in [\![A]\!]_v$.

□

□

Lemma 21 (Subtyping Soundness)**.** *The following hold.*

1. *If* $\Delta; \Phi \models \tau \sqsubseteq \tau'$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $(m, v, v') \in (\!|\sigma\tau|\!)_v$*, then* $(m, v, v') \in (\!|\sigma\tau'|\!)_v$.

2. *If* $\Delta; \Phi \models^A A \sqsubseteq A'$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $(m, v) \in [\![\sigma A]\!]_v$*, then* $(m, v) \in [\![\sigma A']\!]_v$.

3. *If* $\Delta; \Phi \models \tau \sqsubseteq \tau'$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $(m, e, e') \in (\!|\sigma\tau|\!)^t_{\varepsilon}$ *and* $t \leqslant t'$*, then* $(m, e, e') \in (\!|\sigma\tau'|\!)^{t'}_{\varepsilon}$.

4. *If* $\Delta; \Phi \models^A A \sqsubseteq A'$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $(m, e) \in [\![\sigma A]\!]^{k,t}_{\varepsilon}$ *and* $k' \leqslant k$ *and* $t \leqslant t'$*, then* $(m, e) \in [\![\sigma A']\!]^{k',t'}_{\varepsilon}$.

5. *If* $\Delta; \Phi \models \tau \sqsubseteq \tau'$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\forall i \in \{1, 2\}$. $(m, v) \in [\![|\sigma\tau|_i]\!]_v$*, then* $(m, v) \in [\![|\sigma\tau'|_i]\!]_v$.

6. *If* $\Delta; \Phi \models \tau \sqsubseteq \tau'$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\forall i \in \{1, 2\}$. $(m, e) \in [\![|\sigma\tau|_i]\!]^{k,t}_{\varepsilon}$ *and* $k' \leqslant k$ *and* $t \leqslant t'$*, then* $(m, e) \in [\![|\sigma\tau'|_i]\!]^{k',t'}_{\varepsilon}$.

Proof. Statements (1),(2) and (5) are proven simultaneously by induction on the subtyping derivation. We first show the proof of statements (3), (4) and (6) that pertain to expression relations. □

Proof of Item 3. Assume that $\Delta; \Phi \models \tau \sqsubseteq \tau'$ and $\sigma \in \mathcal{D}[\![\Delta]\!]$ and $(m, e, e') \in (\!|\sigma\tau|\!)^{t}_{\varepsilon}$ and $t \leqslant t'$.
TS: $(m, e, e') \in (\!|\sigma\tau'|\!)^{t'}_{\varepsilon}$
Assume that

a) $e \Downarrow^{c,r} v$
b) $e' \Downarrow^{c',r'} v'$
c) $c < m$

By unfolding the assumption $(m, e, e') \in (\!|\sigma\tau|\!)^{t}_{\varepsilon}$ using (a-c), we obtain

d) $r - r' \leqslant t$
e) $(m - c, v, v') \in (\!|\sigma\tau|\!)_{v}$

We can conclude as follows:

1. Since $r - r' \leqslant t$ from d) and $t \leqslant t'$ from the assumption, we get $r - r' \leqslant t'$.
2. By IH 1 on the main assumption using e), we get $(m - c, v, v') \in (\!|\sigma\tau'|\!)_{v}$.

□

Proof of Item 4. Assume that $\Delta; \Phi \models A \sqsubseteq A'$ and $\sigma \in \mathcal{D}[\![\Delta]\!]$ and $(m, e) \in [\![\sigma A]\!]^{k,t}_{\varepsilon}$ and $k' \leqslant k$ and $t \leqslant t'$.
TS: $(m, e) \in [\![\sigma A']\!]^{k',t'}_{\varepsilon}$.
Assume that

a) $e \Downarrow^{c,r} v$
b) $c < m$

By unfolding the main assumption $(m, e) \in [\![\sigma A]\!]^{k,t}_{\varepsilon}$ with (a-b), we get

c) $k \leqslant r \leqslant t$
d) $(m - c, v) \in [\![\sigma A]\!]_{v}$

We can conclude as follows:

1. Since $k' \leqslant k$ and $t \leqslant t'$ (from the assumption) and $k \leqslant r \leqslant t$ (from (c)), we get $k' \leqslant r \leqslant t'$.

2. By IH 2 on the main assumption using d).

□

Proof of Item 6. Assume that $\Delta; \Phi \models \tau \sqsubseteq \tau'$ and $\sigma \in \mathcal{D}[\![\Delta]\!]$ and $(m, e) \in [\![|\sigma\tau|_i]\!]_\varepsilon^{k,t}$ and $k' \leqslant k$ and $t \leqslant t'$.
TS: $(m, e) \in [\![|\sigma\tau'|_i]\!]_\varepsilon^{k',t'}$
Assume that

a) $e \Downarrow^{c,r} v$

b) $c < m$

By unfolding the main assumption $(m, e) \in [\![|\sigma\tau|_i]\!]_\varepsilon^{k,t}$ with (a-b), we get

c) $k \leqslant r \leqslant t$

d) $(m - c, v) \in [\![|\sigma\tau|_i]\!]_v$

We can conclude as follows:

1. Since $k' \leqslant k$ and $t \leqslant t'$ (from the assumption) and $k \leqslant r \leqslant t$ (from (a)), we get $k' \leqslant r \leqslant t'$.

2. By IH 5 on the main assumption using d).

□

Proof of Item 1. Proof is by induction on the subtyping derivation.

Case: $\dfrac{\Delta; \Phi_a \models \tau_1' \sqsubseteq \tau_1 \qquad \Delta; \Phi_a \models \tau_2 \sqsubseteq \tau_2' \qquad \Delta; \Phi_a \models t \leqslant t'}{\Delta; \Phi_a \models \tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2 \sqsubseteq \tau_1' \xrightarrow{\mathbf{diff}(t')} \tau_2'}\ \mathbf{r\text{-}\rightarrow\ diff}$

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$.

We have

$$(m, \mathrm{fix}\ f(x).e, \mathrm{fix}\ f(x).e') \in (\!|\sigma\tau_1 \xrightarrow{\mathrm{diff}(\sigma t)} \sigma\tau_2|\!)_v \tag{1}$$

TS: $(m, \mathrm{fix}\ f(x).e, \mathrm{fix}\ f(x).e') \in (\!|\sigma\tau_1' \xrightarrow{\mathrm{diff}(\sigma t')} \sigma\tau_2'|\!)_v$.

There are two cases to show.

subcase 1: Assume that $j < m$ and $(j, v, v') \in (\!|\sigma\tau_1'|\!)_v$.
STS: $(j, e[v/x, (\text{fix } f(x).e)/f], e'[v'/x, (\text{fix } f(x).e')/f]) \in (\!|\sigma\tau_2'|\!)_\varepsilon^{\sigma t'}$.
By IH 1 on $(j, v, v') \in (\!|\sigma\tau_1'|\!)_v$ using the first premise, we get

$$(j, v, v') \in (\!|\sigma\tau_1|\!)_v \tag{2}$$

By unrolling (eq. (1)) with (eq. (2)) using $j < m$, we get

$$(j, e[v/x, (\text{fix } f(x).e)/f], e'[v'/x, (\text{fix } f(x).e')/f]) \in (\!|\sigma\tau_2|\!)_\varepsilon^{\sigma t} \tag{3}$$

By Assumption 25 on the third premise, we get $\sigma t \leqslant \sigma t'$.
We conclude by applying IH 3 to (eq. (3)) using the second premise and $\sigma t \leqslant \sigma t'$.

subcase 2: STS: $\forall j.(j, \text{fix } f(x).e) \in [\![|\sigma\tau_1'|_1 \xrightarrow{\text{exec}(0,\infty)} |\sigma\tau_2'|_1]\!]_v \wedge (j, \text{fix } f(x).e') \in [\![|\sigma\tau_1'|_2 \xrightarrow{\text{exec}(0,\infty)} |\sigma\tau_2'|_2]\!]_v$.
Pick j.
We just show the first part, the second one is similar.
Pick j' and assume that

$$j' < j \tag{4}$$

$$(j', v) \in [\![|\sigma\tau_1'|_1]\!]_v \tag{5}$$

STS: $(j', e[v/x, (\text{fix } f(x).e)/f]) \in [\![|\sigma\tau_2'|_1]\!]_\varepsilon^{0,\infty}$.
By IH 5 on (eq. (5)) using the first premise, we get

$$(j', v) \in [\![|\sigma\tau_1|_1]\!]_v \tag{6}$$

By unrolling the second part of the definition of (eq. (1)), we get

$$\forall j.(j, \text{fix } f(x).e) \in [\![|\sigma\tau_1|_1 \xrightarrow{\text{exec}(0,\infty)} |\sigma\tau_2|_1]\!]_v \tag{7}$$

Instantiating eq. (7) with $j' + 1$, we get

$$(j' + 1, \text{fix } f(x).e) \in [\![|\sigma\tau_1|_1 \xrightarrow{\text{exec}(0,\infty)} |\sigma\tau_2|_1]\!]_\nu \tag{8}$$

Then, by unrolling the definition of (eq. (8)) with (eq. (6)) and (eq. (4)) using $j' < j' + 1$, we get

$$(j', e[\nu/x, (\text{fix } f(x).e)/f]) \in [\![|\sigma\tau_2|_1]\!]^{0,\infty}_\varepsilon \tag{9}$$

We can conclude by IH 6 on the second premise using (eq. (9)).

Case: $\dfrac{}{\Delta; \Phi \models U\,(A_1 \xrightarrow{\text{exec}(k,t)} A_2, A_1' \xrightarrow{\text{exec}(k',t')} A_2') \sqsubseteq U\,(A_1, A_1') \xrightarrow{\text{diff}(t-k')} U\,(A_2, A_2')}$ **r-→ execdiff**

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$.

We have

$$(m, \text{fix } f(x).e, \text{fix } f(x).e') \in (\!|U\,(\sigma A_1 \xrightarrow{\text{exec}(\sigma k,\sigma t)} \sigma A_2, \sigma A_1' \xrightarrow{\text{exec}(\sigma k',\sigma t')} \sigma A_2')|\!)_\nu \tag{1}$$

TS: $(m, \text{fix } f(x).e, \text{fix } f(x).e') \in (\!|U\,(\sigma A_1, \sigma A_1') \xrightarrow{\text{diff}(\sigma t - \sigma k')} U\,(\sigma A_2, \sigma A_2')|\!)_\nu$.

There are two cases to show.

subcase 1: Assume that

a) $j < m$

b) $(j, \nu, \nu') \in (\!|U\,(\sigma A_1, \sigma A_1')|\!)_\nu$

STS: $(j, e[\nu/x, (\text{fix } f(x).e)/f], e'[\nu'/x, (\text{fix } f(x).e')/f]) \in (\!|U\,(\sigma A_2, \sigma A_2')|\!)^{\sigma t - \sigma k'}_\varepsilon$.

Assume that

c) $e[\nu/x, (\text{fix } f(x).e)/f] \Downarrow^{c_r, r_r} \nu_r$

d) $e'[\nu'/x, (\text{fix } f(x).e')/f] \Downarrow^{c_r', r_r'} \nu_r'$

e) $c_r < j$

STS 1: $r_r - r'_r \leqslant \sigma t - \sigma k'$
STS 2: $(m - c_r, v_r, v'_r) \in (\!|U\,(\sigma A_2, \sigma A'_2)|\!)_v$.
We first show the second statement, the first one is shown later.
Then, it suffices to show $\forall j.(j, v_r) \in [\![\sigma A_2]\!]_v \ \wedge\ (j, v'_r)[\![\sigma A'_2]\!]_v$.
Pick j.
RTS1 : $(j, v_r) \in [\![\sigma A_2]\!]_v$
RTS2 : $(j, v'_r)[\![\sigma A'_2]\!]_v$
By (eq. (1)), we know that

$$\forall j'.(j', \text{fix } f(x).e) \in (\!|A_1 \xrightarrow{\text{exec}(k,t)} A_2|\!)_v \wedge (j', \text{fix } f(x).e') \in [\![A'_1 \xrightarrow{\text{exec}(k',t')} A'_2]\!]_v \tag{2}$$

By instantiating j' in (eq. (2)) with $j + c_r + 2$, we get

$$(j + c_r + 2, \text{fix } f(x).e) \in (\!|\sigma A_1 \xrightarrow{\text{exec}(\sigma k,\sigma t)} \sigma A_2|\!)_v \tag{3}$$

By unrolling the definition of b) and instantiating the universal quantifier with $j + c_r + 1$, we get

$$(j + c_r + 1, v) \in [\![\sigma A_1]\!]_v \tag{4}$$

Then, unrolling the definition of (eq. (3)) with (eq. (4)) using $j + c_r + 1 < j + c_r + 2$, we get

$$(j + c_r + 1, e[v/x, \text{fix } f(x).e/f]) \in [\![\sigma A_2]\!]_\varepsilon^{\sigma k,\sigma t} \tag{5}$$

By unrolling the definition of (eq. (5)) using (c) and $c_r < j + c_r + 1$ (obtained by (e)), we get

f) $\sigma k \leqslant r_r \leqslant st$
g) $(j + 1, v_r) \in [\![\sigma A_2]\!]_v$

Next, we instantiate j' in the second part of (eq. (2)) with $j + c'_r + 2$, we get

$$(j + c'_r + 2, \text{fix } f(x).e) \in (\!|\sigma A'_1 \xrightarrow{\text{exec}(\sigma k', \sigma t')} \sigma A'_2|\!)_\nu \tag{6}$$

By unrolling the definition of b) and instantiating the universal quantifier with $j + c'_r + 1$, we get

$$(j + c'_r + 1, \nu') \in [\![\sigma A'_1]\!]_\nu \tag{7}$$

Then, unrolling the definition of (eq. (6)) with (eq. (7)) using $j + c'_r + 1 < j + c'_r + 2$, we get

$$(j + c'_r + 1, e'[\nu'/x, \text{fix } f(x).e'/f]) \in [\![\sigma A'_2]\!]_\varepsilon^{\sigma k', \sigma t'} \tag{8}$$

By unrolling the definition of (eq. (8)) using (d), $c'_r < j + c'_r + 1$, we get

h) $\sigma k' \leqslant r'_r \leqslant \sigma t'$
i) $(j + 1, \nu'_r) \in [\![\sigma A'_2]\!]_\nu$

Now, we can conclude as follows

1. By f) and h), we get $r_r - r'_r \leqslant \sigma t - \sigma k'$
2. By downward closure (Lemma 20) on g) using

   $$j \leqslant j + 1$$

   We get $(j, \nu'_r) \in [\![\sigma A_2]\!]_\nu$
   By downward closure (Lemma 20) on i) using

   $$j \leqslant j + 1$$

   We get $(j, \nu'_r) \in [\![\sigma A'_2]\!]_\nu$ These conclude this subcase.

subcase 2: STS: $\forall j.(j, \text{fix } f(x).e) \in [\![\sigma A_1 \xrightarrow{\text{exec}(0,\infty)} \sigma A_2]\!]_\nu \wedge (j, \text{fix } f(x).e') \in [\![\sigma A_1' \xrightarrow{\text{exec}(0,\infty)} \sigma A_2']\!]_\nu$.
Pick j and assume that for some j'

$$j' < j \tag{9}$$

$$(j', \nu) \in [\![\sigma A_1]\!]_\nu \tag{10}$$

STS: $(j', e[\nu/x, (\text{fix } f(x).e)/f]) \in [\![\sigma A_2]\!]_\varepsilon^{0,\infty}$.
By unrolling second part of (eq. (1))'s definition and instantiating it with j, we have

$$(j, \text{fix } f(x).e) \in (\!|\sigma A_1 \xrightarrow{\text{exec}(\sigma k,\sigma t)} \sigma A_2|\!)_\nu \tag{11}$$

Unrolling this with (eq. (9)) and (eq. (10)), we get

$$(j', e[\nu/x, (\text{fix } f(x).e)/f]) \in [\![\sigma A_2]\!]_\varepsilon^{\sigma k,\sigma t} \tag{12}$$

We can conclude by applying IH 4 to (eq. (12)) using $0 \leqslant \sigma k$ and $\sigma t \leqslant \infty$.

Case: $\dfrac{}{\Delta; \Phi \models \Box(\tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2) \sqsubseteq \Box\tau_1 \xrightarrow{\mathbf{diff}(0)} \Box\tau_2}\ \mathbf{r\text{-}\to\Box_{diff}}$
Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$.
We have

$$(m, \text{fix } f(x).e, \text{fix } f(x).e) \in (\!|\Box(\sigma\tau_1 \xrightarrow{\text{diff}(\sigma t)} \sigma\tau_2)|\!)_\nu \tag{1}$$

TS: $(m, \text{fix } f(x).e, \text{fix } f(x).e) \in (\!|\Box\sigma\tau_1 \xrightarrow{\text{diff}(0)} \Box\sigma\tau_2|\!)_\nu$.
There are two cases:

subcase 1: Assume that $j < m$ and $(j, \nu, \nu) \in (\!|\Box\sigma\tau_1|\!)_\nu$ (we have the same values due to box).

STS: $(j, e[v/x, (\text{fix } f(x).e)/f], e[v/x, (\text{fix } f(x).e)/f]) \in (\!|\Box\, \sigma\tau_2|\!)^{0}_{\varepsilon}$.
Assume that

a) $e[v/x, (\text{fix } f(x).e)/f] \Downarrow^{c_r, r_r} v_r$
b) $e[v/x, (\text{fix } f(x).e)/f] \Downarrow^{c_r, r_r} v_r$
c) $c_r < m$

By unrolling first part of the definition of (eq. (1)) with $j < m$ and $(j, v, v) \in (\!|\sigma\tau_1|\!)_v$, we get

$$(j, e[v/x, (\text{fix } f(x).e)/f], e[v/x, (\text{fix } f(x).e)/f]) \in (\!|\sigma\tau_2|\!)^{\sigma t}_{\varepsilon} \qquad (2)$$

Unrolling the definition of (eq. (2)) with (a-c), we get

d) $r_r - r_r \leqslant \sigma t$
e) $(m - c_r, v_r, v_r) \in (\!|\sigma\tau_2|\!)_v$

We can conclude as follows

1. Trivially $r_r - r_r \leqslant 0$
2. By e), we get $(m - c_r, v_r, v_r) \in (\!|\Box\, \sigma\tau_2|\!)_v$

subcase 2: STS: $\forall j.(j, \text{fix } f(x).e) \in [\![|\Box\, \sigma\tau_1 \xrightarrow{\text{diff}(0)} \Box\, \sigma\tau_2|_1]\!]_v \equiv [\![|\sigma\tau_1|_1 \xrightarrow{\text{exec}(0,\infty)} |\sigma\tau_2|_1]\!]_v$.
Immediately follows by unrolling the second part of the definition of (eq. (1)) since we have $|\Box\, (\sigma\tau_1 \xrightarrow{\text{diff}(0)} \sigma\tau_2)|_1 = |\Box\, \sigma\tau_1 \xrightarrow{\text{diff}(0)} \Box\, \sigma\tau_2|_1$.

Case: $\dfrac{}{\Delta; \Phi \models \Box\tau \sqsubseteq \tau}$ **T**
Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$.
We have

$$(m, v_1, v_2) \in (\!|\Box\, \sigma\tau|\!)_v \qquad (1)$$

TS: $(m, v_1, v_2) \in (\!|\sigma\tau|\!)_v$.
From eq. (1), we know that

a) $(m, v_1, v_2) \in (\!|\sigma\tau|\!)_v$

b) $v_1 = v_2$

We can immediately conclude by a).

Case: $\dfrac{\Delta; \Phi_a \models \tau_1 \sqsubseteq \tau_2}{\Delta; \Phi_a \models \Box\tau_1 \sqsubseteq \Box\tau_2}$ **B-$\Box$**

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$.

We have

$$(m, v_1, v_2) \in (\!|\Box\, \sigma\tau_1|\!)_v \tag{1}$$

TS: $(m, v_1, v_2) \in (\!|\Box\, \sigma\tau_2|\!)_v$.

From eq. (1), we know that

a) $(m, v_1, v_2) \in (\!|\sigma\tau_1|\!)_v$

b) $v_1 = v_2$

By IH 1 on a), we get $(m, v_1, v_2) \in (\!|\sigma\tau_2|\!)_v$ $(\star)$.

Then, we can conclude by $(\star)$ and b).

Case: $\dfrac{}{\Delta; \Phi \models \tau \sqsubseteq U\,(|\tau|_1, |\tau|_2)}$ **W**

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$.

We have

$$(m, v_1, v_2) \in (\!|\sigma\tau|\!)_v \tag{1}$$

TS: $(m, v_1, v_2) \in (\!|U\,(|\sigma\tau|_1, |\sigma\tau|_2)|\!)_v$.

Proof is by induction on τ.

We show a few representative cases below.

subcase 1: $(m, v_1, v_2) \in (\!|U\,(A_1, A_2)|\!)_v$ $(\star)$

Since $\sigma\tau = U\,(A_1, A_2) = U\,(|\sigma\tau|_1, |\sigma\tau|_2)$, we immediately conclude by $(\star)$.

subcase 2: $(m, \text{inl}\, v_1, \text{inl}\, v_2) \in (\!|\sigma\tau_1 + \sigma\tau_2|\!)_v$ $(\star)$

TS: $(m, \text{inl}\, v_1, \text{inl}\, v_2) \in (\!|U\,(|\sigma\tau_1 + \sigma\tau_2|_1, |\sigma\tau_1 + \sigma\tau_2|_2)|\!)_v$.

STS: $\forall j.(j, \text{inl } v_1) \in [\![|\sigma\tau_1 + \sigma\tau_2|_1]\!]_v \wedge (j, \text{inl } v_2) \in [\![|\sigma\tau_1 + \sigma\tau_2|_2]\!]_v$.
By unrolling their definition and noting that $|\sigma\tau_1 + \sigma\tau_2|_i = |\sigma\tau_1|_i + |\sigma\tau_2|_i \; \forall i \in \{1,2\}$,
RTS:

$$\forall j.(j, v_1) \in [\![|\sigma\tau_1|_1]\!]_v \wedge (j, v_2) \in [\![|\sigma\tau_1|_2]\!]_v \tag{2}$$

By unrolling the definition of $(\star)$, we have $(m, v_1, v_2) \in (\!|\sigma\tau_1|\!)_v$.
By IH 1, we get $(m, v_1, v_2) \in (\!|U\,(|\sigma\tau_1|_1, |\sigma\tau_1|_2)|\!)_v$ which is equivalent to (eq. (2)).

subcase 3: $(m, \text{fix } f(x).e_1, \text{fix } f(x).e_2) \in (\!|\sigma\tau_1 \xrightarrow{\text{diff}(k)} \sigma\tau_2|\!)_v \; (\star)$
TS: $(m, \text{fix } f(x).e_1, \text{fix } f(x).e_2) \in (\!|U\,(|\sigma\tau_1 \xrightarrow{\text{diff}(k)} \sigma\tau_2|_1, |\sigma\tau_1 \xrightarrow{\text{diff}(k)} \sigma\tau_2|_2)|\!)_v$
STS: $\forall j.(j, \text{fix } f(x).e_1) \in [\![|\sigma\tau_1|_1 \xrightarrow{\text{exec}(0,\infty)} |\sigma\tau_2|_1]\!]_v \wedge (j, \text{fix } f(x).e_2) \in [\![|\sigma\tau_1|_2 \xrightarrow{\text{exec}(0,\infty)} |\sigma\tau_2|_2]\!]_v$.
Follows by unrolling the second part of the definition of $(\star)$.

Case:
$$\frac{\Delta; \Phi_a \models n \doteq n' \qquad \Delta; \Phi_a \models \alpha \leqslant \alpha' \qquad \Delta; \Phi_a \models \tau \sqsubseteq \tau'}{\Delta; \Phi_a \models \textbf{list}[n]^{\alpha}\,\tau \sqsubseteq \textbf{list}[n']^{\alpha'}\,\tau'} \textbf{ r-l1}$$

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$ and $\models \sigma\Phi$ and $(m, v, v') \in (\!|\text{list}[n]^{\alpha}\,\tau|\!)_v$.
TS: $(m, v, v') \in (\!|\text{list}[\sigma n']^{\sigma\alpha'}\,\sigma\tau'|\!)_v$
From Assumption 25 applied to the first premise, $\sigma n = \sigma n'$. Therefore,
STS: $(m, v, v') \in (\!|\text{list}[\sigma n]^{\sigma\alpha'}\,\sigma\tau'|\!)_v$
From Assumption 25 applied to the second premise, $\sigma\alpha \leqslant \sigma\alpha'$. Therefore,
We prove the following more general statement (the proof follows by instantiating this statement):
$\forall\, m, v, v', n, \alpha, \alpha'$. if $\alpha \leqslant \alpha'$ and $(m, v, v') \in (\!|\text{list}[n]^{\alpha}\,\tau|\!)_v$, then $(m, v, v') \in (\!|\text{list}[n]^{\alpha'}\,\tau'|\!)_v$.
We establish this statement by subinduction on v and v'.

subcase 1: $v = v' = \text{nil}$

We can immediately conclude that $(m, \text{nil}, \text{nil}) \in (\!|\text{list}[0]^{\alpha'}\ \tau'|\!)_v$ by definition.

subcase 2: $v = \text{cons}(v_1, v_2)$ and $v' = \text{cons}(v_1', v_2')$

TS: $(m, \text{cons}(v_1, v_2), \text{cons}(v_1', v_2')) \in (\!|\text{list}[I+1]^{\alpha'}\ \tau'|\!)_v$ for some $I + 1 = n$.

We have two possible cases:

- $(m, v_1, v_1') \in (\!|\Box\,\tau|\!)_v$ $(\dagger)$ and $(m, v_2, v_2') \in (\!|\text{list}[I]^{\alpha}\ \tau|\!)_v$ $(\dagger\dagger)$.
  By subIH on $(\dagger\dagger)$, we get

  $$(m, v_2, v_2') \in (\!|\text{list}[I]^{\alpha'}\ \tau'|\!)_v \tag{1}$$

  By IH on $(\dagger)$, we get

  $$(m, v_1, v_1') \in (\!|\Box\,\tau'|\!)_v \tag{2}$$

  Combining (eq. (2)) with (eq. (1)), we get
  $(m, \text{cons}(v_1, v_2), \text{cons}(v_1', v_2')) \in (\!|\text{list}[I+1]^{\alpha'}\ \tau'|\!)_v$.
- $(m, v_1, v_1') \in (\!|\tau|\!)_v$ $(\diamond)$ and $(m, v_2, v_2') \in (\!|\text{list}[I]^{\alpha-1}\ \tau|\!)_v$ $(\diamond\diamond)$.
  By subIH on $(\diamond\diamond)$, we get

  $$(m, v_2, v_2') \in (\!|\text{list}[I]^{\alpha'-1}\ \sigma\tau'|\!)_v \tag{3}$$

  By IH on $(\diamond)$, we get

  $$(m, v_1, v_1') \in (\!|\tau'|\!)_v \tag{4}$$

  Combining (eq. (4)) with (eq. (3)), we get
  $(m, \text{cons}(v_1, v_2), \text{cons}(v_1', v_2')) \in (\!|\text{list}[I+1]^{\alpha'}\ \sigma\tau'|\!)_v$.

subcase 3: $v = \text{nil}$ and $v' = \text{cons}(v_1', v_2')$

This case is impossible since v and v' can't be related at the given type.

subcase 4: $v = \mathsf{cons}(v_1, v_2)$ and $v' = \mathsf{nil}$

This case is impossible since v and v' can't be related at the given type.

Case: $\dfrac{\Delta; \Phi \models \alpha \doteq 0}{\Delta; \Phi \models \mathbf{list}[n]^{\alpha}\,\tau \sqsubseteq \mathbf{list}[n]^{\alpha}\,\Box\,\tau}$ **r-l2**

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$ and $\models \sigma\Phi$ and $(m, v, v') \in (\!|\mathsf{list}[n]^{\alpha}\,\tau|\!)_v$.

TS: $(m, v, v') \in (\!|\mathsf{list}[\sigma n]^{\sigma\alpha}\,\Box\,\sigma\tau|\!)_v$

We prove the following more general statement by subinduction on n.

subcase 1: $n = 0$

Then, we know that $v = v' = \mathsf{nil}$

We can immediately conclude that $(m, \mathsf{nil}\,, \mathsf{nil}\,) \in (\!|\mathsf{list}[0]^{0}\,\Box\,\sigma\tau|\!)_v$ by definition.

subcase 2: $n = I + 1$

Then, we know that $v = \mathsf{cons}(v_1, v_2)$ and $v' = \mathsf{cons}(v_1', v_2')$

TS: $(m, \mathsf{cons}(v_1, v_2), \mathsf{cons}(v_1', v_2')) \in (\!|\mathsf{list}[I+1]^{0}\,\Box\,\sigma\tau|\!)_v$.

We have two possible cases:

- $(m, v_1, v_1') \in (\!|\Box\,\sigma\tau|\!)_v$ (†) and $(m, v_2, v_2') \in (\!|\mathsf{list}[I]^{0}\,\sigma\tau|\!)_v$ (††).
  By subIH on (††), we get $(m, v_2, v_2') \in (\!|\mathsf{list}[I]^{0}\,\Box\,\sigma\tau|\!)_v$.
  Combining the (†) with the previous statement, we get $(m, \mathsf{cons}(v_1, v_2), \mathsf{cons}(v_1', v_2')) \in (\!|\mathsf{list}[I+1]^{0}\,\Box\,\sigma\tau|\!)_v$.
- $(m, v_1, v_1') \in (\!|\sigma\tau|\!)_v$ and $(m, v_2, v_2') \in (\!|\mathsf{list}[I]^{0-1}\,\sigma\tau|\!)_v$.
  This case is impossible since $0 - 1 \not\geqslant 0$.

Case: $\dfrac{}{\Delta; \Phi \models \mathbf{list}[n]^{\alpha}\,\Box\,\tau \sqsubseteq \Box\,(\mathbf{list}[n]^{\alpha}\,\tau)}$ **r-l$\Box$**

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$ and $\models \sigma\Phi$ and $(m, v, v') \in (\!|\mathsf{list}[\sigma n]^{\sigma\alpha}\,\Box\,\sigma\tau|\!)_v$.

TS: $(m, v, v') \in (\!|\Box\,(\mathsf{list}[\sigma n]^{\sigma\alpha}\,\sigma\tau)|\!)_v$

We prove the following more general statement

$\forall i, \beta, \tau$. if $(m, v, v) \in (\!|\mathsf{list}[i]^{\beta}\,\Box\,\sigma\tau|\!)_v$, then $(m, v, v) \in (\!|\Box\,(\mathsf{list}[i]^{\beta}\,\sigma\tau)|\!)_v$

by subinduction on i.

subcase 1: $i = 0$

Then, we know that $v = v' = \text{nil}$

We can immediately conclude that $(m, \text{nil}, \text{nil}) \in (\!|\Box\, \text{list}[0]^{\beta}\ \sigma\tau|\!)_v$ by definition.

subcase 2: $i = I + 1$

TS: $(m, \text{cons}(v_1, v_2), \text{cons}(v_1', v_2')) \in (\!|\Box\, \text{list}[I+1]^{\beta}\ \sigma\tau|\!)_v$.

We have two possible cases:

- $(m, v_1, v_1') \in (\!|\Box\, \Box\, \sigma\tau|\!)_v$ (†) and $(m, v_2, v_2) \in (\!|\text{list}[I]^{\beta}\, \Box\, \sigma\tau|\!)_v$ (††).
  Instantiating subIH on (††), we get

  $$(m, v_2, v_2') \in (\!|\Box\, \text{list}[I]^{\beta}\ \sigma\tau|\!)_v \text{ and } v_2 = v_2' \tag{1}$$

  By (†), we also know that

  $$(m, v_1, v_1) \in (\!|\Box\, \sigma\tau|\!)_v \tag{2}$$

  Combining (eq. (2)) with (eq. (1)), we get $(m, \text{cons}(v_1, v_2), \text{cons}(v_1, v_2)) \in (\!|\Box\, \text{list}[I+1]^{\beta}\ \sigma\tau|\!)_v$.
- $(m, v_1, v_1) \in (\!|\Box\, \sigma\tau|\!)_v$ (◇) and $(m, v_2, v_2) \in (\!|\text{list}[I]^{\beta-1}\, \Box\, \sigma\tau|\!)_v$ (◇◇).
  Instantiating subIH on (◇◇), we get

  $$(m, v_2, v_2') \in (\!|\Box\, \text{list}[I]^{\beta-1}\ \sigma\tau|\!)_v \text{ and } v_2 = v_2' \tag{3}$$

  Combining (◇) with (eq. (3)), we get $(m, \text{cons}(v_1, v_2), \text{cons}(v_1, v_2)) \in (\!|\Box\, \text{list}[I+1]^{\beta}\ \sigma\tau|\!)_v$.

Then the proof follows by instantiating the generalized statement with $\beta = \sigma\alpha$ and $i = \sigma n$.

□

Proof of Item 2. Proof is by induction on the subtyping derivation.

Case:
$$\frac{\Delta;\Phi \models^{A} A_1' \sqsubseteq A_1 \qquad \Delta;\Phi \models^{A} A_2 \sqsubseteq A_2' \qquad \Delta;\Phi \models k' \leqslant k \qquad \Delta;\Phi \models t \leqslant t'}{\Delta;\Phi \models^{A} A_1 \xrightarrow{\mathbf{exec}(k,t)} A_2 \sqsubseteq A_1' \xrightarrow{\mathbf{exec}(k',t')} A_2'}\ \mathbf{u\text{-}\longrightarrow\ exec}$$

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$.

We have

$$(m, \text{fix } f(x).e) \in [\![\sigma A_1 \xrightarrow{\text{exec}(\sigma k,\sigma t)} \sigma A_2]\!]_v \tag{1}$$

TS: $(m, \text{fix } f(x).e) \in [\![\sigma A_1' \xrightarrow{\text{exec}(\sigma k',\sigma t')} \sigma A_2']\!]_v$.

Pick j and assume that

$$j < m \tag{2}$$

$$(j, v) \in [\![\sigma A_1']\!]_v \tag{3}$$

STS: $(j, e[v/x, (\text{fix } f(x).e)/f]) \in [\![\sigma A_2']\!]_\varepsilon^{\sigma k',\sigma t'}$.

By IH 2 on (eq. (3)) using the first premise, we get

$$(j, v) \in [\![\sigma A_1]\!]_v \tag{4}$$

By unrolling the definition of (eq. (1)) with (eq. (4)) and $j < m$, we get

$$(j, e[v/x, (\text{fix } f(x).e)/f]) \in [\![\sigma A_2]\!]_\varepsilon^{\sigma k,\sigma t} \tag{5}$$

By Assumption 25 on the third and fourth premises, we get $\sigma k' \leqslant \sigma k$ and $\sigma t \leqslant \sigma t'$.

We conclude by applying IH 4 to (eq. (5)) using σ, i.e $\sigma t \leqslant \sigma t'$ and $\sigma k' \leqslant \sigma k$.

Case: $\dfrac{i :: S, \Delta; \Phi \models^{A} A \sqsubseteq A' \qquad i \notin FV(\Phi)}{\Delta; \Phi \models^{A} \exists i{::}S.\, A \sqsubseteq \exists i{::}S.\, A'}$ **u-∃**

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$.

We have

$$(m, \text{pack } v) \in [\![\exists i{::}S.\, \sigma A]\!]_v \tag{1}$$

TS: $(m, \text{pack } v) \in [\![\exists i{::}S.\, \sigma A']\!]_v$.

By unrolling its definition, assume that $\vdash I :: S$ $(\star)$.

STS: $(m, v) \in [\![\sigma A'\{I/i\}]\!]_v$.

By unrolling (eq. (1)) with $\star$, we get

$$(m, v) \in [\![\sigma A\{I/i\}]\!]_v \tag{2}$$

Then, we can conclude by IH 2 on (eq. (2)).

□

Proof of Item 5. Proof is by induction on the subtyping derivation. We focus on the left projection where $i = 1$. The case where $i = 2$ is similar.

Case: $\dfrac{\Delta; \Phi_a \models \tau_1' \sqsubseteq \tau_1 \qquad \Delta; \Phi_a \models \tau_2 \sqsubseteq \tau_2' \qquad \Delta; \Phi_a \models t \leqslant t'}{\Delta; \Phi_a \models \tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2 \sqsubseteq \tau_1' \xrightarrow{\mathbf{diff}(t')} \tau_2'}$ **r-→ diff**

Assume that $(m, \text{fix } f(x).e) \in [\![|\sigma\tau_1|_1 \xrightarrow{\text{exec}(0,\infty)} |\sigma\tau_2|_1]\!]_v$ $(\star)$.

TS: $(m, \text{fix } f(x).e) \in [\![|\sigma\tau_1'|_1 \xrightarrow{\text{exec}(0,\infty)} |\sigma\tau_2'|_1]\!]_v$.

Pick j and assume that

$$j < m \tag{1}$$

$$(j, v) \in [\![|\sigma\tau_1'|_1]\!]_v \tag{2}$$

STS: $(j, e[v/x, (\text{fix } f(x).e)/f]) \in [\![|\sigma\tau_2'|_1]\!]_\varepsilon^{0,\infty}$.
By IH 5 on the first premise using (eq. (2)), we get

$$(j, v) \in [\![|\sigma\tau_1|_1]\!]_v \tag{3}$$

By unrolling the definition of $(\star)$ with (eq. (3)) and (eq. (1)), we get

$$(j, e[v/x, (\text{fix } f(x).e)/f]) \in [\![|\sigma\tau_2|_1]\!]_\varepsilon^{0,\infty} \tag{4}$$

We can conclude by IH 6 on the second premise using (eq. (4)).

Case: $\dfrac{}{\Delta; \Phi \models U\,(A_1 \xrightarrow{\mathbf{exec}(k,t)} A_2, A_1' \xrightarrow{\mathbf{exec}(k',t')} A_2') \sqsubseteq U\,(A_1, A_1') \xrightarrow{\mathbf{diff}(t-k')} U\,(A_2, A_2')}\ \mathbf{r\text{-}\to\ execdiff}$

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$.
We immediately focus our attention to the left projections.
We have $(j, \text{fix } f(x).e) \in [\![\sigma A_1 \xrightarrow{\text{exec}(\sigma k,\sigma t)} \sigma A_2]\!]_v \quad (\star)$.
STS: $(j, \text{fix } f(x).e) \in [\![\sigma A_1 \xrightarrow{\text{exec}(0,\infty)} \sigma A_2]\!]_v$.
Assume that for some j'

$$j' < j \tag{1}$$

$$(j', v) \in [\![\sigma A_1]\!]_v \tag{2}$$

STS: $(j', e[v/x, (\text{fix } f(x).e)/f]) \in [\![\sigma A_2]\!]_\varepsilon^{0,\infty}$.
By unrolling $(\star)$'s definition with (eq. (1)) and (eq. (2)), we get

$$(j', e[v/x, (\text{fix } f(x).e)/f]) \in [\![\sigma A_2]\!]_\varepsilon^{\sigma k,\sigma t} \tag{3}$$

We can conclude by applying IH 4 to (eq. (3)) using $0 \leqslant \sigma k$ and $\sigma t \leqslant \infty$.

Case: $\dfrac{\Delta; \Phi_a \models n \doteq n' \qquad \Delta; \Phi_a \models \alpha \leqslant \alpha' \qquad \Delta; \Phi_a \models \tau \sqsubseteq \tau'}{\Delta; \Phi_a \models \mathbf{list}[n]^{\alpha}\,\tau \sqsubseteq \mathbf{list}[n']^{\alpha'}\,\tau'}$ **r-l1**

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$ and $\models \sigma\Phi$ and $(m, v) \in [\![\mathrm{list}[\sigma n]\,|\sigma\tau|_1]\!]_v$.

TS: $(m, v) \in [\![\mathrm{list}[\sigma n']\,|\sigma\tau'|_1]\!]_v$

From Assumption 25 applied to the first premise, $\sigma n = \sigma n'$. Therefore,

STS: $(m, v) \in [\![\mathrm{list}[\sigma n]\,|\sigma\tau'|_1]\!]_v$

We prove the following more general statement

$\forall\, m, v, n$. if $(m, v) \in [\![\mathrm{list}[n]\,|\sigma\tau|_1]\!]_v$, then $(m, v) \in [\![\mathrm{list}[|\sigma\tau'|_1]\,]\!]_v$.

We establish this statement by subinduction on v.

subcase 1: $v = \mathrm{nil}$

We can immediately conclude that $(m, \mathrm{nil}\,) \in [\![\mathrm{list}[0]\,|\sigma\tau'|_1]\!]_v$ by definition.

subcase 2: $v = \mathrm{cons}(v_1, v_2)$

TS: $(m, \mathrm{cons}(v_1, v_2)) \in [\![\mathrm{list}[I+1]\,|\sigma\tau'|_1]\!]_v$ for some $I + 1 = n$.

By the main assumption, we have $(m, v_1) \in [\![|\sigma\tau|_1]\!]_v$ ($\diamond$) and $(m, v_2) \in [\![\mathrm{list}[I]\,|\sigma\tau|_1]\!]_v$ ($\diamond\diamond$).

By subIH on ($\diamond\diamond$), we get

$$(m, v_2) \in [\![\mathrm{list}[I]\,|\sigma\tau'|_1]\!]_v \tag{1}$$

By IH 5 on ($\diamond$), we get

$$(m, v_1) \in [\![|\sigma\tau'|_1]\!]_v \tag{2}$$

Combining (eq. (2)) with (eq. (1)), we get $(m, \mathrm{cons}(v_1, v_2)) \in [\![\mathrm{list}[I+1]\,|\sigma\tau'|_1]\!]_v$.

Case: $\dfrac{i :: S, \Delta; \Phi_a \models \tau \sqsubseteq \tau' \qquad i \notin FV(\Phi_a)}{\Delta; \Phi_a \models \exists i{::}S.\, \tau \sqsubseteq \exists i{::}S.\, \tau'}$ **r-$\exists$**

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$.

We have

$$(m, \text{pack } v) \in [\![\exists i{::}S.\, |\sigma\tau|_1]\!]_v \tag{1}$$

TS: $(m, \text{pack } v) \in [\![\exists i{::}S.\, |\sigma\tau'|_1]\!]_v$.

By unrolling its definition, assume that $\vdash I :: S$ $(\star)$.

STS: $(m, v) \in [\![|\sigma\tau'|_1\{I/i\}]\!]_v$.

By unrolling (eq. (1)) with $(\star)$, we get

$$(m, v) \in [\![|\sigma\tau|_1\{I/i\}]\!]_v \tag{2}$$

Then, we can conclude by IH 5 on (eq. (2)).

Case: $\dfrac{}{\Delta; \Phi \models \Box\, \tau \sqsubseteq \tau}$ **T**

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$.

We have $(m, v) \in [\![|\Box\, \sigma\tau|_i]\!]_v$.

TS: $(m, v) \in [\![|\sigma\tau|_1]\!]_v$.

Immediately follows since by definition of $|\cdot|_1$, we know that $|\Box\, \sigma\tau|_1 = |\sigma\tau|_1$.

Case: $\dfrac{}{\Delta; \Phi \models \tau \sqsubseteq U\, (|\tau|_1, |\tau|_2)}$ **W**

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$.

We have $(m, v) \in [\![|\sigma\tau|_1]\!]_v$.

TS: $(m, v) \in [\![|U\, (|\sigma\tau|_1, |\sigma\tau|_2)|_1]\!]_v$.

Immediately follows by the main assumption since by definition of $|\cdot|_i$, we know that $|\sigma\tau|_1 = |U\, (|\sigma\tau|_1, |\sigma\tau|_2)|_1$.

□

Lemma 22 (Sort Substitution)**.** *The following hold.*

1. *If* $\Delta \vdash I :: S$ *and* $\Delta, i :: S \vdash I' :: S'$*, then* $\Delta \vdash I'[I/i] :: S'$.

2. *If* $\Delta \vdash I :: S$ *and* $\Delta, \vdash i :: S \vdash C$ `wf`, *then* $\Delta \vdash C[I/i]$ `wf`.

3. *If* $\Delta \vdash I :: S$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$, *then* $\vdash \sigma I :: S$.

Proof. (1) and (2) are established by simultaneous induction on the second given derivations. (3) follows from (1). □

Both of our fundamental theorems rely on the assumption that the semantic interpretation of every primitive function lies in the interpretation of the function's type. This is explained below.

Assumption 23 (Soundness of primitive functions (relational))**.** *Suppose that* $\zeta : \tau_1 \xrightarrow{diff(t)} \tau_2$ *and* $(m, v, v') \in (\!|\tau_1|\!)_v$ *and* $\hat{\zeta}\, v = (c_r, r_r, v_r)$ *and* $\hat{\zeta}\, v' = (c_r', r_r', v_r')$, *then*

- $(m - c_r, v_r, v_r') \in (\!|\tau_2|\!)_v$
- $r_r - r_r' \leqslant t$.

Assumption 24 (Soundness of primitive functions (non-relational))**.** *Suppose that* $\zeta : A_1 \xrightarrow{exec(k,t)} A_2$ *and* $(m, v) \in [\![A_1]\!]_v$ *and* $\hat{\zeta}\, v = (c_r, r_r, v_r)$, *then*

- $(m - c_r, v_r) \in [\![A_2]\!]_v$
- $k \leqslant r_r \leqslant t$.

We assume that the constraint judgment $\Delta; \Phi \models C$ satisfies some standard properties.

Assumption 25 (Constraint conditions)**.** *The following hold.*

1. *[Subst1] If* $\Delta, i :: S; \Phi \models C$ *and* $\Delta \vdash I :: S$, *then* $\Delta; \Phi[I/i] \models C[I/i]$.

2. *[Subst2] If* $\Delta; \Phi \models C$ *and* $\Delta; \Phi \wedge C \models C'$, *then* $\Delta; \Phi \models C'$.

3. *[Neg]* $\Delta; \Phi \models \neg C$ *iff* $\Delta; \Phi \not\models C$.

4. *[Corr1] If* $\models n_1 \leqslant n_2$, *then* $n_1 \leqslant n_2$.

5. *[Corr2] If* $\models I \doteq I'$, *then* $I = I'$.

Assumption 26 (Constraint Well-formedness). *If* $\Delta; \Phi \models C$ *then* $\Delta \vdash C\ \mathtt{wf}$

Lemma 27 (Well-formedness). *The following hold.*

1. *If* $\Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t : \tau$ *and* $\Delta \vdash \Gamma\ \mathtt{wf}$ *and* $FIV(\Gamma) \subseteq dom(\Delta)$, *then* $\Delta \vdash \tau\ \mathtt{wf}$ *and* $FIV(t, \tau) \subseteq dom(\Delta)$.
2. *If* $\Delta; \Phi_a; \Omega \vdash^{t}_{k} e : A$ *and* $\Delta \vdash^{A} \Omega\ \mathtt{wf}$ *and* $FIV(\Omega) \subseteq dom(\Delta)$, *then* $\Delta \vdash^{A} A\ \mathtt{wf}$ *and* $FIV(k, t, A) \subseteq dom(\Delta)$.
3. *If* $\Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t : \tau$, *then* $FV(e) \subseteq dom(\Gamma)$ *and* $FV(e') \subseteq dom(\Gamma)$.
4. *If* $\Delta; \Phi_a; \Omega \vdash^{t}_{k} e : A$, *then* $FV(e) \subseteq dom(\Omega)$.

Proof. The proof is by induction on the typing derivations. □

Lemma 28 (Refinement Removal Well-formedness). *The following hold.*

- *If* $\Delta \vdash \tau\ \mathtt{wf}$, *then* $\Delta \vdash^{A} |\tau|_i\ \mathtt{wf}$ *for* $i \in \{1, 2\}$.
- *If* $\Delta \vdash \Gamma\ \mathtt{wf}$, *then* $\Delta \vdash^{A} |\Gamma|_i\ \mathtt{wf}$ *for* $i \in \{1, 2\}$.

Lemma 29 (Subtyping well-formedness). *The following hold.*

- *If* $\Delta; \Phi \models \tau \sqsubseteq \tau'$ *and* $\Delta \vdash \tau\ \mathtt{wf}$, *then* $\Phi \vdash \tau'\ \mathtt{wf}$.
- *If* $\Delta; \Phi \models^{A} A \sqsubseteq A'$ *and* $\Delta \vdash^{A} A\ \mathtt{wf}$, *then* $\Delta \vdash A'\ \mathtt{wf}$.

Proof. The proof is by induction on the subtyping derivations. □

A.2 RELCOST THEOREMS

Theorem 30 (Fundamental theorem)**.** *The following holds.*

1. *Assume that* $\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$*. Then,* $(m, \delta e_1, \delta' e_2) \in (\!|\sigma\tau|\!)^{\sigma t}_{\varepsilon}$.

2. *Assume that* $\Delta; \Phi_a; \Omega \vdash^{t}_{k} e : A$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and there exists* Ω' *s.t.* $FV(e) \subseteq dom(\Omega')$ *and* $\Omega' \subseteq \Omega$ *and* $(m, \gamma) \in \mathcal{G}[\![\sigma\Omega']\!]$*. Then,* $(m, \gamma e) \in [\![\sigma A]\!]^{\sigma k, \sigma t}_{\varepsilon}$.

3. *Assume that* $\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$*. Then for* $i \in \{1, 2\}$*, if there exists* Γ_i' *s.t.* $FV(e_i) \subseteq dom(\Gamma_i')$ *and* $\Gamma_i' \subseteq \Gamma$ *and* $(m, \delta) \in \mathcal{G}[\![|\sigma\Gamma_i'|_i]\!]$*, then* $(m, \delta e_i) \in [\![|\sigma\tau|_i]\!]^{0, \infty}_{\varepsilon}$.

Proof. Proofs are by induction on typing derivations. We show each statement separately.

Proof of Statement (1). We proceed by induction on the typing derivation. We show the most important cases below.

Case: $\dfrac{\Gamma(x) = \tau}{\Delta; \Phi_a; \Gamma \vdash x \ominus x \lesssim 0 : \tau}$ **r-var**

Assume that $\models \sigma\Phi$ and $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$.

TS: $(m, \delta(x), \delta'(x)) \in (\!|\sigma\tau|\!)^{0}_{\varepsilon}$.

By Value Lemma (Lemma 18), STS: $(m, \delta(x), \delta'(x)) \in (\!|\sigma\tau|\!)_{v}$.

This follows by the premise $\Gamma(x) = \tau$ and the assumption $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$.

Case:

$$\frac{\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_1' \lesssim t_1 : \tau \qquad \Delta; \Phi_a; \Gamma \vdash e_2 \ominus e_2' \lesssim t_2 : \mathbf{list}[n]^{\alpha}\,\tau}{\Delta; \Phi_a; \Gamma \vdash \mathbf{cons}(e_1, e_2) \ominus \mathbf{cons}(e_1', e_2') \lesssim t_1 + t_2 : \mathbf{list}[n+1]^{\alpha+1}\,\tau}\ \textbf{r-cons1}$$

Assume that $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m, \mathrm{cons}(\delta e_1, \delta e_2), \mathrm{cons}(\delta' e_1', \delta' e_2')) \in (\!|\mathrm{list}[\sigma n + 1]^{\sigma\alpha+1}\,\sigma\tau|\!)^{\sigma t_1 + \sigma t_2}_{\varepsilon}$.

Following the definition of $(\!|\cdot|\!)^{\cdot}_{\varepsilon}$, assume that

$$\frac{\delta e_1 \Downarrow^{c_1, r_1} v_1 \quad (\star) \qquad \delta e_2 \Downarrow^{c_2, r_2} v_2 \quad (\diamond)}{\mathrm{cons}(\delta e_1, \delta e_2) \Downarrow^{c_1 + c_2, r_1 + r_2} \mathrm{cons}(v_1, v_2)}\ \textbf{cons}$$ and

$$\frac{\delta' e_1' \Downarrow^{c_1',r_1'} v_1' \quad (\star\star) \qquad \delta' e_2' \Downarrow^{c_2',r_2'} v_2' \quad (\diamond\diamond)}{\mathrm{cons}(\delta' e_1', \delta' e_2') \Downarrow^{c_1'+c_2',r_1'+r_2'} \mathrm{cons}(v_1', v_2')}\ \textbf{cons} \quad \text{and}$$

$c_1 + c_2 < m$.

By IH 1 on the first premise, we get $(m, \delta e_1, \delta' e_1') \in (\!|\sigma\tau|\!)_\varepsilon^{\sigma t_1}$.

Unrolling its definition with $(\star)$ and $(\star\star)$ and $c_1 < m$, we get

a) $r_1 - r_1' \leqslant \sigma t_1$

b) $(m - c_1, v_1, v_1') \in (\!|\sigma\tau|\!)_v$

By IH 1 on the second premise, we get $(m, \delta e_2, \delta' e_2') \in (\!|\mathrm{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|\!)_\varepsilon^{\sigma t_2}$. Unrolling its definition with $(\diamond)$ and $(\diamond\diamond)$, and $c_2 < m$, we get

c) $r_2 - r_2' \leqslant \sigma t_2$

d) $(m - c_2, v_2, v_2') \in (\!|\mathrm{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|\!)_v$

Now, we can conclude as follows:

1. Using a) and c), we get $(r_1 + r_2) - (r_1' + r_2') \leqslant \sigma t_1 + \sigma t_2$
2. By downward closure (Lemma 20) on b) and d) using

$$m - (c_1 + c_2) \leqslant m - c_1$$

$$m - (c_1 + c_2) \leqslant m - c_2$$

we get $(m - (c_1 + c_2), v_1, v_1') \in (\!|\sigma\tau|\!)_v$ and $(m - (c_1 + c_2), v_2, v_2') \in (\!|\mathrm{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|\!)_v$, when combined, gives us $(m - (c_1 + c_2), \mathrm{cons}(v_1, v_2), \mathrm{cons}(v_1', v_2')) \in (\!|\mathrm{list}[\sigma n + 1]^{\sigma\alpha+1}\ \sigma\tau|\!)_v$

Case:

$$\frac{\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_1' \lesssim t_1 : \Box\tau \qquad \Delta; \Phi_a; \Gamma \vdash e_2 \ominus e_2' \lesssim t_2 : \mathbf{list}[n]^\alpha\ \tau}{\Delta; \Phi_a; \Gamma \vdash \mathbf{cons}(e_1, e_2) \ominus \mathbf{cons}(e_1', e_2') \lesssim t_1 + t_2 : \mathbf{list}[n+1]^\alpha\ \tau}\ \textbf{r-cons2}$$

Assume that $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m, \mathrm{cons}(\delta e_1, \delta e_2), \mathrm{cons}(\delta' e_1', \delta' e_2')) \in (\!|\mathrm{list}[\sigma n + 1]^{\sigma\alpha}\ \sigma\tau|\!)_\varepsilon^{\sigma t_1 + \sigma t_2}$.

Following the definition of $(\!|\cdot|\!)_\varepsilon^{\cdot}\cdot$, assume that

$$\frac{\delta e_1 \Downarrow^{c_1,r_1} v_1 \quad (\star) \qquad \delta e_2 \Downarrow^{c_2,r_2} v_2 \quad (\diamond)}{\mathrm{cons}(\delta e_1, \delta e_2) \Downarrow^{c_1+c_2,r_1+r_2} \mathrm{cons}(v_1, v_2)}\ \textbf{cons} \text{ and}$$

$$\frac{\delta' e_1' \Downarrow^{c_1',r_1'} v_1' \quad (\star\star) \qquad \delta' e_2' \Downarrow^{c_2',r_2'} v_2' \quad (\diamond\diamond)}{\mathrm{cons}(\delta' e_1', \delta' e_2') \Downarrow^{c_1'+c_2',r_1'+r_2'} \mathrm{cons}(v_1', v_2')}\ \textbf{cons} \quad \text{and}$$

$c_1 + c_2 < m$.

By IH 1 on the first premise, we get $(m, \delta e_1, \delta' e_1') \in (\!|\Box\, \sigma\tau|\!)_\varepsilon^{\sigma t_1}$.
Unrolling its definition with $(\star)$ and $(\star\star)$, and $c_1 < m$, we get

a) $r_1 - r_1' \leqslant \sigma t_1$
b) $(m - c_1, v_1, v_1') \in (\!|\Box\, \sigma\tau|\!)_v$

By IH 1 on the second premise, we get
$(m, \delta e_2, \delta' e_2') \in (\!|\mathrm{list}[\sigma n]^{\sigma\alpha}\, \sigma\tau|\!)_\varepsilon^{\sigma t_2}$. Unrolling its definition with $(\diamond)$ and $(\diamond\diamond)$, and $c_2 < m$, we get

c) $r_2 - r_2' \leqslant \sigma t_2$
d) $(m - c_2, v_2, v_2') \in (\!|\mathrm{list}[\sigma n]^{\sigma\alpha}\, \sigma\tau|\!)_v$

Now, we can conclude as follows:

1. Using a) and c), we get $(r_1 + r_2) - (r_1' + r_2') \leqslant \sigma t_1 + \sigma t_2$
2. By downward-closure (Lemma 20) on b) and d) using

$$m - (c_1 + c_2) \leqslant m - c_1$$

$$m - (c_1 + c_2) \leqslant m - c_2$$

we get $(m - (c_1 + c_2), v_1, v_1') \in (\!|\Box\, \sigma\tau|\!)_v$ and $(m - (c_1 + c_2), v_2, v_2') \in (\!|\mathrm{list}[\sigma n]^{\sigma\alpha}\, \sigma\tau|\!)_v$, when combined, gives us
$(m - (c_1 + c_2), \mathrm{cons}(v_1, v_2), \mathrm{cons}(v_1', v_2')) \in (\!|\mathrm{list}[\sigma n + 1]^{\sigma\alpha}\, \sigma\tau|\!)_v$

Case:
$$\frac{\begin{array}{c}\Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t : \mathbf{list}[n]^\alpha\, \tau \\ \Delta; \Phi_a \wedge n = 0; \Gamma \vdash e_1 \ominus e_1' \lesssim t' : \tau' \\ i, \Delta; \Phi_a \wedge n = i + 1; h : \Box\tau, tl : \mathbf{list}[i]^\alpha\, \tau, \Gamma \vdash e_2 \ominus e_2' \lesssim t' : \tau' \\ i, \beta, \Delta; \Phi_a \wedge n = i + 1 \wedge \alpha = \beta + 1; h : \tau, tl : \mathbf{list}[i]^\beta\, \tau, \Gamma \vdash e_2 \ominus e_2' \lesssim t' : \tau'\end{array}}{\Delta; \Phi_a; \Gamma \vdash \begin{array}{l}\mathbf{case}\ e\ \mathbf{of\ nil} \to e_1 \\ \mid h :: tl \to e_2\end{array} \ominus \begin{array}{l}\mathbf{case}\ e'\ \mathbf{of\ nil} \to e_1' \\ \mid h :: tl \to e_2'\end{array} \lesssim t + t' : \tau'}\ \textbf{r-caseL}$$

Assume that $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m,\ \text{case } \delta e \text{ of nil } \to \delta e_1 \mid h :: tl \to \delta e_2,\ \text{case } \delta' e' \text{ of nil } \to \delta' e'_1 \mid h :: tl \to \delta' e'_2) \in (\!|\sigma\tau'|\!)^{\sigma t+\sigma t'}_{\varepsilon}$.

Following the definition of $(\!|\cdot|\!)^{:}_{\varepsilon}$, assume that

$$\text{case } \delta e \text{ of nil } \to \delta e_1 \mid h :: tl \to \delta e_2 \Downarrow^{C,R} v_r \tag{1}$$

and

$$\text{case } \delta' e' \text{ of nil } \to \delta' e'_1 \mid h :: tl \to \delta' e'_2 \Downarrow^{C',R'} v'_r \tag{2}$$

and $C < m$.

Depending on what δe and $\delta' e'$ evaluate to, there are four cases.

subcase 1:

$$\frac{\delta e \Downarrow^{c,r} \text{nil} \quad (\star) \qquad \delta e_1 \Downarrow^{c_r,r_r} v_r \quad (\diamond)}{\text{case } \delta e \text{ of nil } \to \delta e_1 \mid h :: tl \to \delta e_2 \Downarrow^{c+c_r+1,r+r_r+c_{caseL}} v_r}\ \textbf{caseL-nil}$$

and

$$\frac{\delta' e' \Downarrow^{c',r'} \text{nil} \quad (\star\star) \qquad \delta' e'_1 \Downarrow^{c'_r,r'_r} v'_r \quad (\diamond\diamond)}{\text{case } \delta' e' \text{ of nil } \to \delta' e'_1 \mid h :: tl \to \delta' e'_2 \Downarrow^{c'+c'_r+1,r'+r'_r+c_{caseL}} v'_r}\ \textbf{caseL-nil}$$

and $C = c + c_r + 1 < m$ and $R = r + r_r + c_{caseL}$ and $R' = r' + r'_r + c_{caseL}$.

By IH 1 on the first premise, we get $(m, \delta e, \delta' e') \in (\!|\text{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|\!)^{\sigma t}_{\varepsilon}$. Unrolling its definition with $(\star)$, $(\star\star)$ and $c < m$, we get

a) $r - r' \leqslant \sigma t$

b) $(m - c, \text{nil}, \text{nil}) \in (\!|\text{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|\!)_{v}$

By b), $\sigma n = 0$.

Then, we can instantiate IH 1 on the second premise using $\models \sigma\Phi \wedge \sigma n \doteq 0$, to obtain $(m, \delta e_1, \delta' e'_1) \in (\!|\sigma\tau'|\!)^{\sigma t'}_{\varepsilon}$.

Unrolling its definition using $(\diamond)$ and $(\diamond\diamond)$ and $c_r < m$, we get

c) $r_r - r'_r \leqslant \sigma t'$

d) $(m - c_r, v_r, v'_r) \in (\!|\sigma\tau'|\!)_{v}$

We conclude with

1. By a) and c), we get $(r + r_r + c_{caseL}) - (r' + r'_r + c_{caseL}) \leqslant \sigma t + \sigma t'$
2. By downward closure (Lemma 20) on d) using

$$m - (c + c_r + 1) \leqslant m - c_r$$

we get $(m - (c + c_r + 1), v_r, v'_r) \in (\!|\sigma\tau'|\!)_v$.

subcase 2:

$$\frac{\delta e \Downarrow^{c,r} \text{nil} \quad (\star) \qquad \delta e_1 \Downarrow^{c_r,r_r} v_r \quad (\diamond)}{\text{case } \delta e \text{ of nil } \to \delta e_1 \mid h :: tl \to \delta e_2 \Downarrow^{c+c_r+1,r+r_r+c_{caseL}} v_r}\ \textbf{caseL-nil}$$

and

$$\frac{\delta' e' \Downarrow^{c',r'} \text{cons}(v'_1, v'_2) \quad (\star\star) \qquad \delta' e'_2[v'_1/h, v'_2/tl] \Downarrow^{c'_r,r'_r} v'_r \quad (\diamond\diamond)}{\text{case } \delta' e' \text{ of nil } \to \delta' e'_1 \mid h :: tl \to \delta' e'_2 \Downarrow^{c'+c'_r+1,r'+r'_r+c_{caseL}} v'_r}\ \textbf{caseL-cons}$$

and $C = c + c_r + 1 < m$, $R = r + r_r + c_{caseL}$ and $R' = r' + r'_r + c_{caseL}$.

By IH 1 on the first premise, we get $(m, \delta e, \delta' e') \in (\!|\text{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|\!)^{\sigma t}_{\varepsilon}$. Unrolling its definition with $(\star)$, $(\star\star)$ and $c < m$, we get

a) $r - r' \leqslant \sigma t$

b) $(m - c, \text{nil}, \text{cons}(v'_1, v'_2)) \in (\!|\text{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|\!)_v$

However, b) is false since two lists of different length are not related at the given type, therefore this case is vacuously true.

subcase 3:

$$\frac{\delta e \Downarrow^{c,r} \text{cons}(v_1, v_2) \quad (\star) \qquad \delta e_2[v_1/h, v_2/tl] \Downarrow^{c_r,r_r} v_r \quad (\diamond)}{\text{case } \delta e \text{ of nil } \to \delta e_1 \mid h :: tl \to \delta e_2 \Downarrow^{c+c_r+1,r+r_r+c_{caseL}} v_r}\ \textbf{caseL-cons}$$

and

$$\frac{\delta' e' \Downarrow^{c',r'} \text{cons}(v'_1, v'_2) \quad (\star\star) \qquad \delta' e'_2[v'_1/h, v'_2/tl] \Downarrow^{c'_r,r'_r} v'_r \quad (\diamond\diamond)}{\text{case } \delta' e' \text{ of nil } \to \delta' e'_1 \mid h :: tl \to \delta' e'_2 \Downarrow^{c'+c'_r+1,r'+r'_r+c_{caseL}} v'_r}\ \textbf{caseL-cons}$$

and $C = c + c_r + 1$, $R = r + r_r + c_{caseL}$ and $C' = c' + c'_r + 1$, $R' = r' + r'_r + c_{caseL}$.

By IH 1 on the first premise, we get
$(m, \delta e, \delta' e') \in (\!|\text{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|\!)^{\sigma t}_{\varepsilon}$. Unrolling its definition with $(\star)$ and $(\star\star)$ and $c < m$, we get

a) $r - r' \leqslant \sigma t$
b) $(m - c, \text{cons}(v_1, v_2), \text{cons}(v_1', v_2')) \in (\!|\text{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|\!)_v$

For b), there are two cases:

subsubcase 1: $\sigma n = I + 1$ such that we have

$$(m - c, v_1, v_1') \in (\!|\Box\, \sigma\tau|\!)_v \tag{3}$$

$$(m - c, v_2, v_2') \in (\!|\text{list}[I]^{\sigma\alpha}\ \sigma\tau|\!)_v \tag{4}$$

In addition, by downward closure (Lemma 20) on $(m, \delta, \delta') \in \mathcal{G}(\!|\Gamma|\!)$, we have

$$(m - c, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!) \tag{5}$$

Then, we can instantiate IH 1 on the third premise using

- $\sigma[i \mapsto I] \in \mathcal{D}[\![i :: \mathbb{N}, \Delta]\!]$
- $\models \sigma[i \mapsto I](\Phi \wedge n \doteq i + 1)$ obtained by
  - $\models \sigma\Phi$ by main assumption
  - $\models \sigma n \doteq I + 1$ by sub-assumption
- $(m - c, \delta[h \mapsto v_1, tl \mapsto v_2], \delta'[h \mapsto v_1', tl \mapsto v_2']) \in \mathcal{G}(\!|\sigma[i \mapsto I](\Gamma, x : \Box\tau, tl : \text{list}[i]^{\alpha}\ \tau)|\!)$ using (3) and (4) and (3).

we get $(m - c, \delta e_2[v_1/h, v_2/tl], \delta' e_2'[v_1'/h, v_2'/tl]) \in (\!|\sigma[i \mapsto I]\tau'|\!)^{\sigma[i\mapsto I]t'}_{\varepsilon}$.
Since, $i \notin \text{FV}(t', \tau, \tau')$, we have
$(m - c, \delta e_2[v_1/h, v_2/tl], \delta' e_2'[v_1'/h, v_2'/tl]) \in (\!|\sigma\tau'|\!)^{\sigma t'}_{\varepsilon}$.

Unrolling its definition using ($\diamond$), ($\diamond\diamond$) and $c_r < m - c$, we get

c) $r_r - r'_r \leqslant \sigma t'$
d) $(m - (c + c_r), v_r, v'_r) \in (\!|\sigma\tau'|\!)_v$

We conclude with

1. By a) and c), we get $(r + r_r + c_{caseL}) - (r' + r'_r + c_{caseL}) \leqslant \sigma t + \sigma t' + c_{caseL}$
2. By downward closure (Lemma 20) on d) using

$$m - (c + c_r + 1) \leqslant m - (c + c_r)$$

we get $(m - (c + c_r + 1), v_r, v'_r) \in (\!|\sigma\tau'|\!)_v$.

subsubcase 2: $\sigma n = I + 1$ and $\sigma\alpha = J + 1$ such that we have

$$(m - c, v_1, v'_1) \in (\!|\sigma\tau|\!)_v \tag{6}$$

$$(m - c, v_2, v'_2) \in (\!|\text{list}[I]^J\ \sigma\tau|\!)_v \tag{7}$$

In addition, by downward closure (Lemma 20) on $(m, \delta, \delta') \in \mathcal{G}(\!|\Gamma|\!)$, we have

$$(m - c, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!) \tag{8}$$

Then, we can instantiate IH 1 on the fourth premise using

- $\sigma[i \mapsto I, \beta \mapsto J] \in \mathcal{D}[\![i :: \mathbb{N}, \beta :: \mathbb{N}, \Delta]\!]$
- $\models \sigma[i \mapsto I, \beta \mapsto J](\Phi \wedge n \doteq i + 1 \wedge \alpha \doteq \beta + 1)$ obtained
  - $\models \sigma\Phi$ by main assumption
  - $\models \sigma n \doteq I + 1$ by sub-assumption
  - $\models \sigma\alpha \doteq J + 1$ by sub-assumption

- $(m - c, \delta[h \mapsto v_1, tl \mapsto v_2], \delta'[h \mapsto v_1', tl \mapsto v_2']) \in \mathcal{G}(\!|\sigma[i \mapsto I, \beta \mapsto J](\Gamma, x : \tau, tl : \mathrm{list}[i]^{\beta}\ \tau)|\!)$ using (4) and (5) and (6)

we get $(m - c, \delta e_2[v_1/h, v_2/tl], \delta' e_2'[v_1'/h, v_2'/tl]) \in (\!|\sigma[i \mapsto I, \beta \mapsto J]\tau'|\!)_{\varepsilon}^{\sigma[i\mapsto I,\beta\mapsto J]t'}$.
Since, $i, \beta \notin FV(t', \tau, \tau')$, we have
$(m - c, \delta e_2[v_1/h, v_2/tl], \delta' e_2'[v_1'/h, v_2'/tl]) \in (\!|\sigma\tau'|\!)_{\varepsilon}^{\sigma t'}$.
Unrolling its definition using ($\diamond$), ($\diamond\diamond$) and $c_r < m - c$, we get

e) $r_r - r_r' \leqslant \sigma t'$
f) $(m - (c + c_r), v_r, v_r') \in (\!|\sigma\tau'|\!)_{\nu}$

We conclude with

1. By a) and e), we get $(r + r_r + c_{caseL}) - (r' + r_r' + c_{caseL}) \leqslant \sigma t + \sigma t' + c_{caseL}$
2. By downward closure (Lemma 20) on f) using

$$m - (c + c_r + 1) \leqslant m - (c + c_r)$$

we get $(m - (c + c_r + 1), v_r, v_r') \in (\!|\sigma\tau'|\!)_{\nu}$.

subcase 4:

$$\frac{\delta e \Downarrow^{c,r} \mathrm{cons}(v_1, v_2) \quad (\star) \qquad \delta e_2[v_1/h, v_2/tl] \Downarrow^{c_r, r_r} v_r \quad (\diamond)}{\mathrm{case}\ \delta e\ \mathrm{of\ nil}\ \to \delta e_1 \mid h :: tl \to \delta e_2 \Downarrow^{c + c_r + 1, r + r_r + c_{caseL}} v_r}\ \textbf{caseL-cons}$$

and

$$\frac{\delta' e' \Downarrow^{c', r'} \mathrm{nil} \quad (\star\star) \qquad \delta' e_1' \Downarrow^{c_r', r_r'} v_r' \quad (\diamond\diamond)}{\mathrm{case}\ \delta' e'\ \mathrm{of\ nil}\ \to \delta' e_1' \mid h :: tl \to \delta' e_2' \Downarrow^{c' + c_r' + 1, r' + r_r' + c_{caseL}} v_r'}\ \textbf{caseL-nil}.$$

By IH 1 on the first premise, we get
$(m, \delta e, \delta' e') \in (\!|\mathrm{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|\!)_{\varepsilon}^{\sigma t}$. Unrolling its definition with ($\star$), ($\star\star$) and $c < m$, we get

a) $r - r' \leqslant \sigma t$
b) $(m - c, \mathrm{cons}(v_1, v_2), \mathrm{nil}\ ) \in (\!|\mathrm{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|\!)_{\nu}$

However, b) is false since two lists of different length are not related at the given type, therefore this case is vacuously true.

Case: $$\frac{\Delta \vdash \tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2 \ \mathtt{wf} \qquad \Delta; \Phi_a; x : \tau_1, f : \tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2, \Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau_2}{\Delta; \Phi_a; \Gamma \vdash \mathbf{fix}\ f(x).e_1 \ominus \mathbf{fix}\ f(x).e_2 \lesssim 0 : \tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2}\ \textbf{r-fix}$$

Assume that $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.
TS: $(m, \mathrm{fix}\ f(x).\delta e_1, \mathrm{fix}\ f(x).\delta' e_2) \in (\!|\sigma\tau_1 \xrightarrow{\mathrm{diff}(\sigma t)} \sigma\tau_2|\!)^0_\varepsilon$.
By Lemma 18, STS: $(m, \mathrm{fix}\ f(x).\delta e_1, \mathrm{fix}\ f(x).\delta' e_2) \in (\!|\sigma\tau_1 \xrightarrow{\mathrm{diff}(\sigma t)} \sigma\tau_2|\!)_v$.
Let $F = \mathrm{fix}\ f(x).\delta e_1$ and $F' = \mathrm{fix}\ f(x).\delta' e_2$.
We prove the more general statement

$$\forall\, m' \leqslant m.\ (m', F, F') \in (\!|\sigma\tau_1 \xrightarrow{\mathrm{diff}(\sigma t)} \sigma\tau_2|\!)_v$$

by subinduction on m'.

There are two parts to show:

subcase 1: $m' = 0$

By the definition of function types, there are two parts to show:

subsubcase 1: $\forall j < m' = 0 \cdots$

Since there is no non-negative j such that $j < 0$, the goal is vacuously true.

subsubcase 2: STS: $\forall j.(j, F) \in [\![|\sigma\tau_1|_1 \xrightarrow{\mathrm{exec}(0,\infty)} |\sigma\tau_2|_1]\!]_v \wedge (j, F')[\![|\sigma\tau_1|_2 \xrightarrow{\mathrm{exec}(0,\infty)} |\sigma\tau_2|_2]\!]_v$.
Pick j.

- STS 1: $(j, F) \in [\![|\sigma\tau_1|_1 \xrightarrow{\mathrm{exec}(0,\infty)} |\sigma\tau_2|_1]\!]_v$
  We prove the more general statement

  $$\forall m' \leqslant j.\ (m', F) \in [\![|\sigma\tau_1|_1 \xrightarrow{\mathrm{exec}(0,\infty)} |\sigma\tau_2|_1]\!]_v$$

  by subinduction on m'.
  There are two cases:

– $m' = 0$
Since there is no non-negative j such that $j < 0$, the goal is vacuously true.

– $m' = m'' + 1 \leqslant m$
By sub-IH

$$(m'', \text{fix } f(x).\delta e_1) \in [\![|\sigma\tau_1|_1 \xrightarrow{\text{exec}(0,\infty)} |\sigma\tau_2|_1]\!]_\nu \qquad (1)$$

STS: $(m''+1, \text{fix } f(x).\delta e_1) \in [\![|\sigma\tau_1|_1 \xrightarrow{\text{exec}(0,\infty)} |\sigma\tau_2|_1]\!]_\nu$.
Pick $j'' < m''+1$ and assume that $(j'', \nu) \in [\![|\sigma\tau_1|_1]\!]_\nu$.
STS: $(j'', \delta e_1[\nu/x, F/f]) \in [\![|\sigma\tau_2|_1]\!]_\varepsilon^{0,\infty}$.
This follows by IH 3 on the premise instantiated with $(j'', \delta[x \mapsto \nu, f \mapsto F]) \in \mathcal{G}[\![x : |\sigma\tau_1|_1, f : |\sigma\tau_1|_1 \xrightarrow{\text{exec}(0,\infty)} |\sigma\tau_2|_1, |\sigma\Gamma|_1]\!]$ which holds because

* $FV(e_1) \subseteq \text{dom}(x : \tau_1, f : \tau_1 \xrightarrow{\text{diff}(t)} \tau_2, \Gamma)$ using Lemma 43 on the second premise
* $(j'', \delta) \in \mathcal{G}[\![|\sigma\Gamma|_1]\!]$ using Lemma 19 on $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$
* $(j'', \nu) \in [\![|\sigma\tau_1|_1]\!]_\nu$, from the assumption above
* $(j'', \text{fix } f(x).\delta e_1) \in [\![|\sigma\tau_1|_1 \xrightarrow{\text{exec}(0,\infty)} |\sigma\tau_2|_1]\!]_\nu$, obtained by downward closure (Lemma 20) on (1) using $j'' \leqslant m''$

- STS 2: $(j, F') \in [\![|\sigma\tau_1|_2 \xrightarrow{\text{exec}(0,\infty)} |\sigma\tau_2|_2]\!]_\nu$
We prove the more general statement

$$\forall m' \leqslant j.\ (m', F') \in [\![|\sigma\tau_1|_2 \xrightarrow{\text{exec}(0,\infty)} |\sigma\tau_2|_2]\!]_\nu$$

by subinduction on m'.
There are two cases:

– $m' = 0$
Since there is no non-negative j such that $j < 0$, the goal is vacuously true.

– $m' = m'' + 1 \leqslant j$
By sub-IH

$$(m'', F') \in [\![|\sigma\tau_1|_2 \xrightarrow{\text{exec}(0,\infty)} |\sigma\tau_2|_2]\!]_v \tag{2}$$

STS: $(m'' + 1, \text{fix } f(x).\delta' e_2) \in [\![|\sigma\tau_1|_2 \xrightarrow{\text{exec}(0,\infty)} |\sigma\tau_2|_2]\!]_v$.
Pick $j'' < m'' + 1$ and assume that $(j'', v) \in [\![|\sigma\tau_1|_2]\!]_v$.
STS: $(j'', \delta' e_2[v/x, F'/f]) \in [\![|\sigma\tau_2|_2]\!]_\varepsilon^{0,\infty}$.
This follows by IH 3 on the premise instantiated with $(j'', \delta[x \mapsto v, f \mapsto (\text{fix } f(x).\delta' e_2)]) \in \mathcal{G}[\![x : |\sigma\tau_1|_2, f : |\sigma\tau_1|_2 \xrightarrow{\text{exec}(0,\infty)} |\sigma\tau_2|_2, |\sigma\Gamma|_2]\!]$ which holds because

* $FV(e_2) \subseteq \text{dom}(x : \tau_1, f : \tau_1 \xrightarrow{\text{diff}(t)} \tau_2, \Gamma)$ using Lemma 43 on the second premise
* $(j'', v) \in [\![|\sigma\tau_1|_2]\!]_v$, from the assumption above
* $(j'', \delta) \in \mathcal{G}[\![|\sigma\Gamma|_2]\!]$ using Lemma 19 on $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$
* $(j'', F') \in [\![|\sigma\tau_1|_2 \xrightarrow{\text{exec}(0,\infty)} |\sigma\tau_2|_2]\!]_v$, obtained by downward closure (Lemma 20) on (2) using $j'' \leqslant m''$

subcase 2: $m' = m'' + 1 \leqslant m$
By sub-IH

$$(m'', F, F') \in (\!|\sigma\tau_1 \xrightarrow{\text{diff}(\sigma t)} \sigma\tau_2|\!)_v \tag{3}$$

TS: $(m'' + 1, \text{fix } f(x).\delta e_1, \text{fix } f(x).\delta' e_2) \in (\!|\sigma\tau_1 \xrightarrow{\text{diff}(\sigma t)} \sigma\tau_2|\!)_v$
Pick $j < m'' + 1$ and assume that $(j, v_1, v_2) \in (\!|\sigma\tau_1|\!)_v$.
STS: $(j, \delta e_1[v_1/x, F/f], \delta' e_2[v_2/x, F'/f]) \in (\!|\sigma\tau_2|\!)_\varepsilon^{\sigma t}$.
This follows by IH on the premise instantiated with $(j, \delta[x \mapsto v_1, f \mapsto F], \delta'[x \mapsto v_2, f \mapsto F']) \in \mathcal{G}(\!|\sigma\Gamma, x : \sigma\tau_1, f : \sigma\tau_1 \xrightarrow{\text{diff}(\sigma t)} \sigma\tau_2|\!)$ which holds because

- $(j, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ obtained by downward closure (Lemma 20) using $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $j < m' \leqslant m$.
- $(j, v_1, v_2) \in (\!|\sigma\tau_1|\!)_v$, from the assumption above

- $(j, F, F') \in (\!|\sigma\tau_1 \xrightarrow{\mathbf{diff}(\sigma t)} \sigma\tau_2|\!)_v$, obtained by downward closure (Lemma 20) on (2) using $j \leqslant m''$

This completes the proof of this case.

Case:
$$\frac{\begin{array}{c}\Delta; \Phi_a \vdash \tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2 \ \mathtt{wf} \\ \Delta; \Phi_a; x : \tau_1, f : \Box(\tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2), \Gamma \vdash e \ominus e \lesssim t : \tau_2 \\ \forall x \in \mathrm{dom}(\Gamma).\ \Delta; \Phi_a \models \Gamma(x) \sqsubseteq \Box\Gamma(x)\end{array}}{\Delta; \Phi_a; \Gamma \vdash \mathbf{fix}\ f(x).e \ominus \mathbf{fix}\ f(x).e \lesssim 0 : \Box(\tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2)}\ \textbf{r-fixNC}$$

Assume that $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.
TS: $(m, \mathrm{fix}\ f(x).\delta e, \mathrm{fix}\ f(x).\delta' e) \in (\!|\Box(\sigma\tau_1 \xrightarrow{\mathrm{diff}(\sigma t)} \sigma\tau_2)|\!)^0_\varepsilon$.
By Lemma 18, STS: $(m, \mathrm{fix}\ f(x).\delta e, \mathrm{fix}\ f(x).\delta' e) \in (\!|\Box(\sigma\tau_1 \xrightarrow{\mathrm{diff}(\sigma t)} \sigma\tau_2)|\!)_v$.
By Lemma 21 using $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and the third premise, we get $(m, \delta, \delta') \in \mathcal{G}(\!|\Box\sigma\Gamma|\!)$, i.e. $\delta = \delta'$.
Therefore, STS: $(m, \mathrm{fix}\ f(x).\delta e, \mathrm{fix}\ f(x).\delta e) \in (\!|\sigma\tau_1 \xrightarrow{\mathrm{diff}(\sigma t)} \sigma\tau_2|\!)_v$.
Let $F = \mathrm{fix}\ f(x).\delta e$.
We prove the more general statement

$$\forall\, m' \leqslant m.\ (m', F, F) \in (\!|\sigma\tau_1 \xrightarrow{\mathrm{diff}(\sigma t)} \sigma\tau_2|\!)_v$$

by subinduction on m'.
There are two parts to show:

subcase 1: $m' = 0$

By the definition of function types, there are two parts to show:

subsubcase 1: $\forall j < m' = 0 \cdots$

Since there is no non-negative j such that $j < 0$, the goal is vacuously true.

subsubcase 2: STS: $\forall j.(j, F) \in [\![|\sigma\tau_1|_1 \xrightarrow{\mathrm{exec}(0,\infty)} |\sigma\tau_2|_1]\!]_v$.
Pick j. STS: $(j, F) \in [\![|\sigma\tau_1|_1 \xrightarrow{\mathrm{exec}(0,\infty)} |\sigma\tau_2|_1]\!]_v$
We prove the more general statement

$$\forall m' \leqslant j.\ (m', F) \in [\![|\sigma\tau_1|_1 \xrightarrow{\mathrm{exec}(0,\infty)} |\sigma\tau_2|_1]\!]_v$$

by subinduction on m'.
There are two cases:

- $m' = 0$
  Since there is no non-negative j such that $j < 0$, the goal is vacuously true.
- $m' = m'' + 1 \leqslant j$
  By sub-IH

$$(m'', \text{fix } f(x).\delta e_1) \in [\![|\sigma\tau_1|_1 \xrightarrow{\text{exec}(0,\infty)} |\sigma\tau_2|_1]\!]_\nu \tag{1}$$

  STS: $(m'' + 1, \text{fix } f(x).\delta e_1) \in [\![|\sigma\tau_1|_1 \xrightarrow{\text{exec}(0,t)} |\sigma\tau_2|_1]\!]_\nu$.
  Pick $j'' < m'' + 1$ and assume that $(j'', \nu) \in [\![|\sigma\tau_1|_1]\!]_\nu$.
  STS: $(j'', \delta e_1[\nu/x, F/f]) \in [\![|\sigma\tau_2|_1]\!]_\varepsilon^{0,\infty}$.
  This follows by IH 3 on the premise instantiated with
  - $FV(e) \subseteq \text{dom}(x : \tau_1, f : \tau_1 \xrightarrow{\text{diff}(t)} \tau_2, \Gamma)$ using Lemma 43 on the second premise
  - $(j'', \delta[x \mapsto \nu, f \mapsto F]) \in \mathcal{G}[\![x : |\sigma\tau_1|_1, f : |\sigma\tau_1|_1 \xrightarrow{\text{exec}(0,\infty)} |\sigma\tau_2|_1, |\sigma\Gamma|_1]\!]$ which holds because
    * $(j'', \delta) \in \mathcal{G}[\![|\sigma\Gamma|_1]\!]$ using Lemma 19 on $(m, \delta, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$
    * $(j'', \nu) \in [\![|\sigma\tau_1|_1]\!]_\nu$, from the assumption above
    * $(j'', \text{fix } f(x).\delta e_1) \in [\![|\sigma\tau_1|_1 \xrightarrow{\text{exec}(0,\infty)} |\sigma\tau_2|_1]\!]_\nu$, obtained by downward closure (Lemma 20) on (1) using $j'' \leqslant m''$

subcase 2: $m' = m'' + 1 \leqslant m$
By sub-IH

$$(m'', F, F) \in (\!|\sigma\tau_1 \xrightarrow{\text{diff}(\sigma t)} \sigma\tau_2|\!)_\nu \tag{2}$$

TS: $(m'' + 1, \text{fix } f(x).\delta e_1, \text{fix } f(x).\delta e_2) \in (\!|\sigma\tau_1 \xrightarrow{\text{diff}(\sigma t)} \sigma\tau_2|\!)_\nu$
Pick $j < m'' + 1$ and assume that $(j, \nu_1, \nu_2) \in (\!|\sigma\tau_1|\!)_\nu$.
STS: $(j, \delta e_1[\nu_1/x, F/f], \delta e_2[\nu_2/x, F/f]) \in (\!|\sigma\tau_2|\!)_\varepsilon^{\sigma t}$.
This follows by IH on the premise instantiated with

$(j, \delta[x \mapsto v_1, f \mapsto F], \delta[x \mapsto v_2, f \mapsto F]) \in \mathcal{G}(\!|\sigma\Gamma, x : \sigma\tau_1, f : \Box(\sigma\tau_1 \xrightarrow{\mathrm{diff}(\sigma t)} \sigma\tau_2)|\!)$ which holds because

- $(j, \delta, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ obtained by downward closure (Lemma 20) using $(m, \delta, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $j < m' \leqslant m$.
- $(j, v_1, v_2) \in (\!|\sigma\tau_1|\!)_v$, from the assumption above
- $(j, F, F) \in (\!|\Box(\sigma\tau_1 \xrightarrow{\mathrm{diff}(\sigma t)} \sigma\tau_2)|\!)_v$, obtained by downward closure (Lemma 20) on (2) using $j \leqslant m''$

This completes the proof of this case.

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_1' \lesssim t_1 : \tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2 \quad \Delta; \Phi_a; \Gamma \vdash e_2 \ominus e_2' \lesssim t_2 : \tau_1}{\Delta; \Phi_a; \Gamma \vdash e_1\ e_2 \ominus e_1'\ e_2' \lesssim t_1 + t_2 + t : \tau_2}$ **r-app**

Assume that $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m, \delta e_1\ \delta e_2, \delta' e_1'\ \delta' e_2') \in (\!|\sigma\tau_2|\!)_\varepsilon^{\sigma t_1 + \sigma t_2 + \sigma t}$.

Following the definition of $(\!|\cdot|\!)_\varepsilon^{\cdot}$, assume that

$$\frac{\delta e_1 \Downarrow^{c_1, r_1} \mathrm{fix}\ f(x).e \quad (\star) \qquad \delta e_2 \Downarrow^{c_2, r_2} v_2 \quad (\diamond) \qquad e[v_2/x, (\mathrm{fix}\ f(x).e)/f] \Downarrow^{c_r, r_r} v_r \quad (\dagger)}{\delta e_1\ \delta e_2 \Downarrow^{c_1 + c_2 + c_r + 1, r_1 + r_2 + r_r + c_{app}} v_r}\ \textbf{app}$$ and

$$\frac{\delta' e_1' \Downarrow^{c_1', r_1'} \mathrm{fix}\ f(x).e' \quad (\star\star) \qquad \delta' e_2' \Downarrow^{c_2', r_2'} v_2' \quad (\diamond\diamond) \qquad e'[v_2'/x, (\mathrm{fix}\ f(x).e')/f] \Downarrow^{c_r', r_r'} v_r' \quad (\dagger\dagger)}{\delta' e_1'\ \delta' e_2' \Downarrow^{c_1' + c_2' + c_r' + 1, r_1' + r_2' + r_r' + c_{app}} v_r'}\ \textbf{app}$$ and

$(c_1 + c_2 + c_r + 1) < m$.

By IH 1 on the first premise, we get $(m, \delta e_1, \delta' e_1') \in (\!|\sigma\tau_1 \xrightarrow{\mathrm{diff}(\sigma t)} \sigma\tau_2|\!)_\varepsilon^{\sigma t_1}$. Unrolling its definition with $(\star)$ and $(\star\star)$, and $c_1 < m$, we get

a) $r_1 - r_1' \leqslant \sigma t_1$

b) $(m - c_1, \mathrm{fix}\ f(x).e, \mathrm{fix}\ f(x).e') \in (\!|\sigma\tau_1 \xrightarrow{\mathrm{diff}(\sigma t)} \sigma\tau_2|\!)_v$

By IH 1 on the second premise, we get $(m, \delta e_2, \delta' e_2') \in (\!|\sigma\tau_1|\!)_\varepsilon^{\sigma t_2}$.
Unrolling its definition with $(\diamond)$ and $(\diamond\diamond)$, and $c_2 < m$, we get

c) $r_2 - r_2' \leqslant \sigma t_2$

d) $(m - c_2, v_2, v_2') \in (\!|\sigma\tau_1|\!)_v$

Next, we apply downward-closure (Lemma 20) to d) using

$$m - (c_1 + c_2 + 1) \leqslant m - c_2$$

and we get

$$(m - (c_1 + c_2 + 1), v_2, v_2') \in (\!|\sigma\tau_1|\!)_v \tag{1}$$

We unroll b) using (1) since

$$m - (c_1 + c_2 + 1) < m - c_1$$

and get

$$(m - (c_1 + c_2 + 1), e[v_2/x, \text{fix } f(x).e/f], e'[v_2'/x, \text{fix } f(x).e'/f]) \in (\!|\sigma\tau_2|\!)_\varepsilon^{\sigma t} \tag{2}$$

Next, we unroll (2) using (†) and (††) and $c_r < m - (c_1 + c_2 + 1)$ to obtain

e) $r_r - r_r' \leqslant \sigma t$
f) $(m - (c_1 + c_2 + c_r + 1), v_r, v_r') \in (\!|\sigma\tau_2|\!)_v$

Now, we can conclude as follows:

1. Using a), c) and e), we get $(r_1 + r_2 + r_r + c_{app}) - (r_1' + r_2' + r_r' + c_{app}) \leqslant \sigma t_1 + \sigma t_2 + \sigma t$
2. By f)

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_1' \lesssim t_1 : \tau_1 \qquad \Delta; \Phi_a; \Gamma \vdash e_2 \ominus e_2' \lesssim t_2 : \tau_2}{\Delta; \Phi_a; \Gamma \vdash \langle e_1, e_2\rangle \ominus \langle e_1', e_2'\rangle \lesssim t_1 + t_2 : \tau_1 \times \tau_2}$ **r-prod**

Assume that $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.
TS: $(m, \langle \delta e_1, \delta e_2\rangle, \langle \delta e_1', \delta' e_2'\rangle) \in (\!|\sigma\tau_1 \times \sigma\tau_2|\!)_\varepsilon^{\sigma t_1 + \sigma t_2}$.
Following the definition of $(\!|\cdot|\!)_\varepsilon^{\cdot} \cdot$, assume that

$$\frac{\delta e_1 \Downarrow^{c_1,r_1} v_1 \quad (\star) \qquad \delta e_2 \Downarrow^{c_2,r_2} v_2 \quad (\diamond)}{\langle \delta e_1, \delta e_2 \rangle \Downarrow^{c_1+c_2,r_1+r_2} \langle v_1, v_2 \rangle}\ \textbf{prod} \text{ and}$$

$$\frac{\delta' e_1 \Downarrow^{c_1',r_1'} v_1' \quad (\star\star) \qquad \delta' e_2 \Downarrow^{c_2',r_2'} v_2' \quad (\diamond\diamond)}{\langle \delta' e_1, \delta' e_2 \rangle \Downarrow^{c_1'+c_2',r_1'+r_2'} \langle v_1', v_2' \rangle}\ \textbf{prod} \text{ and } c_1 + c_2 < m.$$

By IH 1 on the first premise, we get $(m, \delta e_1, \delta' e_1') \in (\!|\sigma\tau_1|\!)_\varepsilon^{\sigma t_1}$.
Unrolling its definition with $(\star)$ and $(\star\star)$ and $c_1 < m$, we get

a) $r_1 - r_1' \leqslant \sigma t_1$
b) $(m - c_1, v_1, v_1') \in (\!|\sigma\tau_1|\!)_v$

By IH 1 on the second premise, we get $(m, \delta e_2, \delta' e_2') \in (\!|\sigma\tau_2|\!)_\varepsilon^{\sigma t_2}$.
Unrolling its definition with $(\diamond)$ and $(\diamond\diamond)$ and $c_2 < m$, we get

c) $r_2 - r_2' \leqslant \sigma t_2$
d) $(m - c_2, v_2, v_2') \in (\!|\sigma\tau_2|\!)_v$

We can conclude as follows:

1. By a) and c), we get $(r_1 + r_2) - (r_1' + r_2') \leqslant \sigma t_1 + \sigma t_2$
2. By downward closure (Lemma 20) on b) using

   $$m - (c_1 + c_2) \leqslant m - c_1$$

   we get

   $$(m - (c_1 + c_2), v_1, v_2) \in (\!|\sigma\tau_1|\!)_v \tag{1}$$

   By downward closure (Lemma 20) on d) using

   $$m - (c_1 + c_2) \leqslant m - c_2$$

   we get

   $$(m - (c_1 + c_2), v_1', v_2') \in (\!|\sigma\tau_2|\!)_v \tag{2}$$

   By combining (1) and (2), we can show that $(m - (c_1 + c_2), \langle v_1, v_2 \rangle, \langle v_1', v_2' \rangle) \in (\!|\sigma\tau_1 \times \sigma\tau_2|\!)_v$

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t : \tau_1 \times \tau_2 \qquad i \in \{1,2\}}{\Delta; \Phi_a; \Gamma \vdash \pi_i(e) \ominus \pi_i(e') \lesssim t : \tau_i}$ **r-proj**$_i$

Assume that $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m, \pi_i(\delta e), \pi_i(\delta' e')) \in (\!|\sigma\tau_i|\!)^{\sigma t}_{\varepsilon}$.

Following the definition of $(\!|\cdot|\!)^{\cdot}_{\varepsilon}\cdot$, assume that

$\dfrac{\delta e \Downarrow^{c,r} \langle v_1, v_2\rangle \quad (\star)}{\pi_1(\delta e) \Downarrow^{c+1, r+c_{proj}} v_1}$ **proj1** and $\dfrac{\delta' e \Downarrow^{c,r} \langle v_1', v_2'\rangle \quad (\star\star)}{\pi_1(\delta' e) \Downarrow^{c'+1, r'+c_{proj}} v_1'}$ **proj1** and $c + 1 < m$.

By IH 1 on the first premise, we get $(m, \delta e, \delta' e') \in (\!|\sigma\tau_1 \times \sigma\tau_2|\!)^{\sigma t}_{\varepsilon}$.

Unrolling its definition with $(\star)$ and $(\star\star)$ and $c < m$, we get

a) $r - r' \leqslant \sigma t$

b) $(m - c, \langle v_1, v_2\rangle, \langle v_1', v_2'\rangle) \in (\!|\sigma\tau_1 \times \sigma\tau_2|\!)_v$

We can conclude as follows:

1. By a), $(r + c_{proj}) - (r' + c_{proj}) \leqslant \sigma t$
2. By unrolling the definition of b), we get $(m - c, v_i, v_i') \in (\!|\sigma\tau_i|\!)_v$. Then, by invoking downward closure (Lemma 20) on this using

$$m - (c+1) \leqslant m - c$$

we get $(m - (c+1), v_i, v_i') \in (\!|\sigma\tau_i|\!)_v$.

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t : \tau_1 \qquad \Delta \vdash \tau_2 \ \mathtt{wf}}{\Delta; \Phi_a; \Gamma \vdash \mathbf{inl}\ e \ominus \mathbf{inl}\ e' \lesssim t : \tau_1 + \tau_2}$ **r-inl**

Assume that $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m, \mathrm{inl}\ (\delta e), \mathrm{inl}\ (\delta' e')) \in (\!|\sigma\tau_1 + \sigma\tau_2|\!)^{\sigma t}_{\varepsilon}$.

Following the definition of $(\!|\cdot|\!)^{\cdot}_{\varepsilon}\cdot$, assume that

$\dfrac{\delta e \Downarrow^{c,r} v \quad (\star)}{\mathrm{inl}\ \delta e \Downarrow^{c,r} \mathrm{inl}\ v}$ **inl** and $\dfrac{\delta' e' \Downarrow^{c',r'} v' \quad (\star\star)}{\mathrm{inl}\ \delta' e' \Downarrow^{c',r'} \mathrm{inl}\ v'}$ **inl** and $c < m$.

By IH 1 on the first premise, we get $(m, \delta e, \delta' e') \in (\!|\sigma\tau_1|\!)^{\sigma t}_{\varepsilon}$. Unrolling its definition with $(\star)$ and $(\star\star)$ and $c < m$, we get

a) $r - r' \leqslant \sigma t$

b) $(m - c, v, v') \in (\!|\sigma\tau_1|\!)_v$

We can conclude as follows:

1. By a), $r - r' \leqslant \sigma t$
2. Using b), we can show that $(m - c, \text{inl}\, v, \text{inl}\, v') \in (\!|\sigma\tau_1 + \sigma\tau_2|\!)_v$

Case: $$\frac{\Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t : \tau_1 + \tau_2 \qquad \Delta; \Phi_a; x : \tau_1, \Gamma \vdash e_1 \ominus e_1' \lesssim t' : \tau \qquad \Delta; \Phi_a; y : \tau_2, \Gamma \vdash e_2 \ominus e_2' \lesssim t' : \tau}{\Delta; \Phi_a; \Gamma \vdash \textbf{case}\ (e, x.e_1, y.e_2) \ominus \textbf{case}\ (e', x.e_1', y.e_2') \lesssim t + t' : \tau}\ \textbf{r-case}$$

Assume that $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m,\ \text{case}\ (\delta e, \delta e_1, \delta e_2),\ \text{case}\ (\delta' e', \delta' e_1', \delta' e_2')) \in (\!|\sigma\tau|\!)_\varepsilon^{\sigma t + \sigma t'}$.

Following the definition of $(\!|\cdot|\!)_\varepsilon^{\cdot}$, assume that

$\text{case}\ (\delta e, \delta e_1, \delta e_2) \Downarrow^{C,R} v_r$ and $\text{case}\ (\delta' e', \delta' e_1', \delta' e_2') \Downarrow^{C',R'} v_r'$ and $C < m$.

Depending on what δe and $\delta' e'$ evaluate to, there are two cases:

subcase 1: $$\frac{\delta e \Downarrow^{c,r} \text{inl}\, v \quad (\star) \qquad \delta e_1[v/x] \Downarrow^{c_r, r_r} v_r \quad (\diamond)}{\text{case}\ (\delta e, x.\delta e_1, y.\delta e_2) \Downarrow^{c + c_r + 1, r + r_r + c_{case}} v_r}\ \textbf{case-inl}$$ and $$\frac{\delta' e' \Downarrow^{c',r'} \text{inl}\, v' \quad (\star\star) \qquad \delta' e_1'[v'/x] \Downarrow^{c_r', r_r'} v_r' \quad (\diamond\diamond)}{\text{case}\ (\delta' e, x.\delta' e_1, y.\delta' e_2) \Downarrow^{c' + c_r' + 1, r' + r_r' + c_{case}} v_r'}\ \textbf{case-inl}$$

Note that $C = c + c_r + 1 < m$.

By IH 1 on the first premise, we get $(m, \delta e, \delta' e') \in (\!|\sigma\tau_1 + \sigma\tau_2|\!)_\varepsilon^{\sigma t}$.

Unrolling its definition with $(\star)$ and $(\star\star)$ and $c < m$, we get

a) $r - r' \leqslant \sigma t$

b) $(m - c, \text{inl}\, v, \text{inl}\, v') \in (\!|\sigma\tau_1 + \sigma\tau_2|\!)_v$

By IH 1 on the second premise using $(m - c, \delta[x \mapsto v], \delta'[x \mapsto v']) \in \mathcal{G}(\!|\sigma\Gamma, x : \sigma\tau_1|\!)$ obtained by

- $(m - c, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ by downward-closure (Lemma 20) on $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ using $m - c \leqslant m$
- $(m - c, v, v') \in (\!|\sigma\tau_1|\!)_v$ by unfolding b)

we get $(m - c, \delta e_1[v/x], \delta' e_1'[v'/x]) \in (\!|\sigma\tau|\!)_\varepsilon^{\sigma t'}$. Unrolling its definition with $(\diamond)$ and $(\diamond\diamond)$, and $c_r < m - c$, we get

c) $r_r - r_r' \leqslant \sigma t'$

d) $(m-(c+c_r), v_r, v_r') \in (\!|\sigma\tau|\!)_v$

Now, we can conclude this subcase by

1. By a) and c) $(r + r_r + c_{case}) - (r' + r_r' + c_{case}) \leqslant \sigma t + \sigma t'$
2. By downward closure (Lemma 20) on d) using

$$m-(c+c_r+1) \leqslant m-(c+c_r)$$

we get $(m-(c+c_r+1), v_r, v_r') \in (\!|\sigma\tau|\!)_v$.

subcase 2: $\dfrac{\delta e \Downarrow^{c,r} \text{inr } v \quad (\star) \qquad \delta e_2[v/y] \Downarrow^{c_r,r_r} v_r \quad (\diamond)}{\text{case } (\delta e, x.\delta e_1, y.\delta e_2) \Downarrow^{c+c_r+1, r+r_r+c_{case}} v_r}$ **case-inr** and

$\dfrac{\delta' e' \Downarrow^{c',r'} \text{inr } v' \quad (\star\star) \qquad \delta' e_2'[v'/y] \Downarrow^{c_r',r_r'} v_r' \quad (\diamond\diamond)}{\text{case } (\delta' e, x.\delta' e_1, y.\delta' e_2) \Downarrow^{c'+c_r'+1, r'+r_r'+c_{case}} v_r'}$ **case-inr**

This case is symmetric, hence we skip its proof.

Case: $\dfrac{i :: S, \Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t : \tau \qquad i \notin \textbf{FIV}(\Phi_a; \Gamma)}{\Delta; \Phi_a; \Gamma \vdash \Lambda.e \ominus \Lambda.e' \lesssim 0 : \forall i \overset{\text{diff}(t)}{::} S.\, \tau}$ **r-iLam**

Assume that $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m, \Lambda.\delta e, \Lambda.\delta' e') \in (\!|\forall i \overset{\text{diff}(\sigma t)}{::} S.\, \sigma\tau|\!)_\varepsilon^0$.

By Lemma 18, STS: $(m, \Lambda.\delta e, \Lambda.\delta' e') \in (\!|\forall i \overset{\text{diff}(\sigma t)}{::} S.\, \sigma\tau|\!)_v$.

By unrolling its definition, assume that $\vdash I :: S$.

There are two cases to show:

subcase 1: STS: $(m, \delta e, \delta' e') \in (\!|\sigma\tau\{I/i\}|\!)_\varepsilon^{\sigma t[I/i]}$.

This follows by IH 1 on the premise instantiated with the substitution $\sigma[i \mapsto I] \in \mathcal{D}[\![i :: S, \Delta]\!]$.

subcase 2: STS: $\forall j.(j, \delta e) \in [\![|\sigma\tau\{I/i\}|_1]\!]_\varepsilon^{0,\infty} \ \wedge \ (j, \delta' e') \in [\![|\sigma\tau\{I/i\}|_2]\!]_\varepsilon^{0,\infty}$.

Pick j.

subsubcase 1: STS1: $(j, \delta e) \in [\![|\sigma\tau\{I/i\}|_1]\!]_\varepsilon^{0,\infty}$

Follows by IH 3 on the premise using

- $FV(e) \subseteq \text{dom}(\Gamma)$ using Lemma 43 on the first premise

- $\sigma[i \mapsto I] \in \mathcal{D}[\![i :: S, \Delta]\!]$
- $(j, \delta) \in \mathcal{G}[\![|\sigma[i \mapsto I]\Gamma|_1]\!] \equiv \mathcal{G}[\![|\sigma\Gamma|_1]\!]$ by Lemma 19 on the main assumption (note that $i \notin FV(\Gamma; \Phi)$)

subsubcase 2: STS2: $(j, \delta' e') \in [\![|\sigma\tau\{I/i\}|_2]\!]^{0,\infty}_{\varepsilon}$

Follows by IH 3 on the premise using

- $FV(e') \subseteq dom(\Gamma)$ using Lemma 43 on the first premise
- $\sigma[i \mapsto I] \in \mathcal{D}[\![i :: S, \Delta]\!]$
- $(j, \delta') \in \mathcal{G}[\![|\sigma[i \mapsto I]\Gamma|_2]\!] \equiv \mathcal{G}[\![|\sigma\Gamma|_2]\!]$ by Lemma 19 on the main assumption (note that $i \notin FV(\Gamma; \Phi)$)

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t : \forall i \overset{\mathbf{diff}(t')}{::} S. \tau \qquad \Delta \vdash I : S}{\Delta; \Phi_a; \Gamma \vdash e[] \ominus e'[] \lesssim t + t'[I/i] : \tau\{I/i\}}$ **r-iApp**

Assume that $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m, \delta e[], \delta' e'[]) \in (\!|\sigma\tau\{\sigma I/i\}|\!)^{\sigma t + \sigma t'[\sigma I/i]}_{\varepsilon}$.

Following the definition of $(\!|\cdot|\!)^{:}_{\varepsilon}$, assume that

$\dfrac{\delta e \Downarrow^{c,r} \Lambda.e_b \quad (\star) \qquad e_b \Downarrow^{c_r, r_r} v_r \quad (\diamond)}{\delta e[] \Downarrow^{c + c_r, r + r_r} v_r}$ **iApp** and

$\dfrac{\delta' e' \Downarrow^{c',r'} \Lambda.e'_b \quad (\star\star) \qquad e'_b \Downarrow^{c'_r, r'_r} v'_r \quad (\diamond\diamond)}{\delta' e'[] \Downarrow^{c' + c'_r, r' + r'_r} v'_r}$ **iApp** and

$(c + c_r) < m$.

By IH on the first premise, we get $(m, \delta e, \delta' e') \in (\!|\forall i \overset{\mathrm{diff}(\sigma t')}{::} S. \sigma\tau|\!)^{\sigma t}_{\varepsilon}$.

By unrolling its definition with $(\star)$, $(\star\star)$ and $c < m$, we get

a) $r - r' \leqslant \sigma t$

b) $(m - c, \Lambda.e_b, \Lambda.e'_b) \in (\!|\forall i \overset{\mathrm{diff}(\sigma t')}{::} S. \sigma\tau|\!)_{v}$

By Lemma 22 on the second premise using $\sigma \in \mathcal{D}[\![\Delta]\!]$, we get

$$\vdash \sigma I :: S \tag{1}$$

By unrolling the definition of b) with (1), we get

$$(m - c, e_b, e'_b) \in (\!|\sigma\tau\{\sigma I/i\}|\!)^{\sigma t'[\sigma I/i]}_{\varepsilon} \tag{2}$$

By unrolling the definition of (2) with ($\diamond$) and ($\diamond\diamond$) and $c_r < m - c$, we get

c) $r_r - r'_r \leqslant \sigma t'[\sigma I/i]$
d) $(m - (c + c_r), v_r, v'_r) \in (\!|\sigma\tau\{\sigma I/i\}|\!)_v$

We conclude as follows

1. By a) and c), we get $(r + r_r) - (r' + r'_r) \leqslant \sigma t + \sigma t'[\sigma I/i]$
2. By d)

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t : \tau\{I/i\} \qquad \Delta \vdash I :: S}{\Delta; \Phi_a; \Gamma \vdash \textbf{pack}\ e \ominus \textbf{pack}\ e' \lesssim t : \exists i{::}S.\, \tau}$ **r-pack**

Assume that $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.
TS: $(m, \text{pack}\ \delta e, \text{pack}\ \delta' e') \in (\!|\exists i{::}S.\, \sigma\tau|\!)^{\sigma t}_{\varepsilon}$.
Following the definition of $(\!|\cdot|\!)^{\cdot}_{\varepsilon}$, assume that
$\dfrac{\delta e \Downarrow^{c,r} v \quad (\star)}{\text{pack}\ \delta e \Downarrow^{c,r} \text{pack}\ v}$ **pack** and $\dfrac{\delta' e' \Downarrow^{c',r'} v' \quad (\star\star)}{\text{pack}\ \delta' e' \Downarrow^{c',r'} \text{pack}\ v'}$ **pack** and $c < m$.
By IH on the first premise, we get $(m, \delta e, \delta' e') \in (\!|\sigma\tau\{\sigma I/i\}|\!)^{\sigma t}_{\varepsilon}$.
By unrolling its definition with $(\star)$, $(\star\star)$ and $c < m$, we get

a) $r - r' \leqslant \sigma t$
b) $(m - c, v, v') \in (\!|\sigma\tau\{\sigma I/i\}|\!)_v$

By Lemma 22 on the second premise, we get

$$\vdash \sigma I :: S \tag{1}$$

We can conclude as follows

1. By a)
2. TS: $(m - c, \text{pack}\ e, \text{pack}\ v') \in (\!|\exists i{::}S.\, \sigma\tau|\!)_v$
   STS1: $\vdash \sigma I :: S$ follows directly by (1).
   STS2: $(m - c, v, v') \in (\!|\sigma\tau\{\sigma I/i\}|\!)_v$ follows by b)

Case:

$$\frac{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_1' \lesssim t_1 : \exists i{::}S.\,\tau_1 \qquad i :: S, \Delta;\Phi_a;x:\tau_1,\Gamma \vdash e_2 \ominus e_2' \lesssim t_2 : \tau_2 \qquad i \notin FV(\Phi_a;\Gamma,\tau_2,t_2)}{\Delta;\Phi_a;\Gamma \vdash \textbf{unpack}\ e_1\ \textbf{as}\ x\ \textbf{in}\ e_2 \ominus \textbf{unpack}\ e_1'\ \textbf{as}\ x\ \textbf{in}\ e_2' \lesssim t_1 + t_2 : \tau_2}\ \textbf{r-unpack}$$

Assume that $(m,\delta,\delta') \in \mathcal{G}(\lfloor\sigma\Gamma\rfloor)$ and $\models \sigma\Phi$.

TS:

$(m, \text{unpack } \delta e_1 \text{ as } x \text{ in } \delta e_2, \text{unpack } \delta' e_1' \text{ as } x \text{ in } \delta' e_2') \in \lfloor\sigma\tau_2\rfloor_\varepsilon^{\sigma t_1 + \sigma t_2}$.

Following the definition of $\lfloor\cdot\rfloor_\varepsilon^{\cdot}$, assume that

$$\frac{\delta e_1 \Downarrow^{c_1,r_1} \text{pack } v \quad (\star) \qquad \delta e_2[v/x] \Downarrow^{c_2,r_2} v_r \quad (\diamond)}{\text{unpack } \delta e_1 \text{ as } x \text{ in } \delta e_2 \Downarrow^{c_1+c_2,r_1+r_2} v_r}\ \textbf{unpack}$$ and

$$\frac{\delta' e_1' \Downarrow^{c_1',r_1'} \text{pack } v' \quad (\star\star) \qquad \delta' e_2'[v'/x] \Downarrow^{c_2',r_2'} v_r' \quad (\diamond\diamond)}{\text{unpack } \delta' e_1' \text{ as } x \text{ in } \delta' e_2' \Downarrow^{c_1'+c_2',r_1'+r_2'} v_r'}\ \textbf{unpack}$$ and

$(c_1 + c_2) < m$.

By IH 1 on the first premise, we get $(m, \delta e_1, \delta' e_1') \in \lfloor\exists i{::}S.\,\sigma\tau_1\rfloor_\varepsilon^{\sigma t_1}$.

By unrolling its definition with $(\star)$, $(\star\star)$ and $c_1 < m$, we get

a) $r_1 - r_1' \leqslant \sigma t_1$

b) $(m - c_1, \text{pack } v, \text{pack } v') \in \lfloor\exists i{::}S.\,\sigma\tau_1\rfloor_v$

By unrolling the definition of b), we get

$$\vdash I :: S \tag{1}$$

$$(m - c_1, v, v') \in \lfloor\sigma\tau_1\{I/i\}\rfloor_v \tag{2}$$

By downward closure (Lemma 20) on $(m,\delta,\delta') \in \mathcal{G}(\lfloor\Gamma\rfloor)$, we have

$$(m - c_1, \delta, \delta') \in \mathcal{G}(\lfloor\sigma\Gamma\rfloor) \tag{3}$$

By IH 1 on the second premise using

- $\sigma[i \mapsto I] \in \mathcal{D}[\![i :: S, \Delta]\!]$ using (1)

- $(m - c_1, \delta[x \mapsto v], \delta'[x \mapsto v']) \in \mathcal{G}(\!|\sigma[i \mapsto I](\Gamma, x : \tau_1)|\!)$ using (2) and (3)

we get

$$(m - c_1, \delta e_2[v/x], \delta' e_2'[v'/x]) \in (\!|\sigma\tau_2|\!)_\varepsilon^{\sigma t_2} \tag{4}$$

By unrolling (4)'s definition using ($\diamond$), ($\diamond\diamond$) and $c_2 < m - c_1$, we get

c) $r_2 - r_2' \leqslant \sigma t_2$
d) $(m - (c_1 + c_2), v_r, v_r') \in (\!|\sigma\tau_2|\!)_v$

We can conclude as follows

1. By a) and c), we get $(r_1 + r_2) - (r_1' + r_2') \leqslant \sigma t_1 + \sigma t_2$
2. Follows by d)

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_1' \lesssim t_1 : \tau_1 \qquad \Delta; \Phi_a; x : \tau_1, \Gamma \vdash e_2 \ominus e_2' \lesssim t_2 : \tau_2}{\Delta; \Phi_a; \Gamma \vdash \textbf{let } x = e_1 \textbf{ in } e_2 \ominus \textbf{let } x = e_1' \textbf{ in } e_2' \lesssim t_1 + t_2 : \tau_2}$ **r-let**

Assume that $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.
TS: $(m, \text{let } x = \delta e_1 \text{ in } \delta e_2, \text{let } x = \delta' e_1' \text{ in } \delta' e_2') \in (\!|\sigma\tau_2|\!)_\varepsilon^{\sigma t_1 + \sigma t_2}$.
Following the definition of $(\!|\cdot|\!)_\varepsilon^{\cdot}$, assume that

$\dfrac{\delta e_1 \Downarrow^{c_1, r_1} v_1 \quad (\diamond) \qquad \delta e_2[v_1/x] \Downarrow^{c_r, r_r} v_r \quad (\dagger)}{\text{let } x = \delta e_1 \text{ in } \delta e_2 \Downarrow^{c_1 + c_r + 1, r_1 + r_r + c_{let}} v_r}$ **let** and

$\dfrac{\delta' e_1' \Downarrow^{c_1', r_1'} v_1' \quad (\diamond\diamond) \qquad \delta' e_2'[v_1'/x] \Downarrow^{c_r', r_r'} v_r' \quad (\dagger\dagger)}{\text{let } x = \delta' e_1 \text{ in } \delta' e_2 \Downarrow^{c_1' + c_r' + 1, r_1' + r_r' + c_{let}} v_r'}$ **let** and

$(c_1 + c_r + 1) < m$.
By IH 1 on the first premise, we get $(m, \delta e_1, \delta' e_1') \in (\!|\sigma\tau_1|\!)_\varepsilon^{\sigma t_1}$.
Unrolling its definition with ($\diamond$) and ($\diamond\diamond$) and $c_1 < m$, we get

a) $r_1 - r_1' \leqslant \sigma t_1$
b) $(m - c_1, v_1, v_1') \in (\!|\sigma\tau_1|\!)_v$

By IH 1 on the second premise using $(m - c_1, \delta[x \mapsto v_1], \delta'[x \mapsto v_1']) \in \mathcal{G}(\!|\sigma\Gamma, x : \sigma\tau_1|\!)$ obtained by

- $(m - c_1, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ by downward closure (Lemma 20) on $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ using $m - c_1 \leqslant m$

- $(m - c_1, \nu, \nu') \in (\!|\sigma\tau_1|\!)_\nu$ by b)

we get $(m - c_1, \delta e_2[\nu_1/x], \delta' e_2'[\nu_1'/x]) \in (\!|\sigma\tau_2|\!)_\varepsilon^{\sigma t_2}$. Unrolling its definition with (†) and (††), and $c_r < m - c_1$, we get

c) $r_r - r_r' \leqslant \sigma t_2$
d) $(m - (c_1 + c_r), \nu_r, \nu_r') \in (\!|\sigma\tau_2|\!)_\nu$

Now, we can conclude with

1. By a) and c) $(r_1 + r_r + c_{let}) - (r_1' + r_r' + c_{let}) \leqslant \sigma t_1 + \sigma t_2$
2. By downward closure (Lemma 20) on d) using

$$m - (c + c_r + 1) \leqslant m - (c + c_r)$$

we get $(m - (c + c_r + 1), \nu_r, \nu_r') \in (\!|\sigma\tau_2|\!)_\nu$.

Case: $\dfrac{\Delta; \Phi_a; |\Gamma|_1 \vdash_{k_1}^{t_1} e_1 : A_1 \qquad \Delta; \Phi_a; |\Gamma|_2 \vdash_{k_2}^{t_2} e_2 : A_2}{\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t_1 - k_2 : U(A_1, A_2)}$ **switch**

Assume that $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.
TS: $(m, \delta e_1, \delta' e_2) \in (\!|U(\sigma A_1, \sigma A_2)|\!)_\varepsilon^{\sigma t_1 - \sigma k_2}$.
Assume that

a) $\delta e_1 \Downarrow^{c_1, r_1} \nu_1$
b) $\delta' e_2 \Downarrow^{c_2, r_2} \nu_2$
c) $c_1 < m$

TS 1: $r_1 - r_2 \leqslant \sigma t_1 - \sigma k_2$
TS 2: $(m - c_1, \nu_1, \nu_2) \in (\!|U(\sigma A_1, \sigma A_2)|\!)_\nu$
We first show the second statement, the first one will be shown later.
By unrolling $(\!|U(\sigma A_1, \sigma A_2)|\!)_\nu$'s definition,
STS: $\forall m.(m, \nu_1) \in [\![\sigma A_1]\!]_\nu \wedge (m, \nu_2) \in [\![\sigma A_2]\!]_\nu$.
Pick m.

By IH 2 on the first premise using

- $FV(e_1) \subseteq \mathrm{dom}(|\sigma\Gamma|_1)$ using Lemma 43 on the first premise

- $\models \sigma\Phi$
- $\forall j.(j,\delta) \in \mathcal{G}[\![|\sigma\Gamma|_1]\!]$ obtained by Lemma 19 on $(m,\delta,\delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$

we get

$$\forall j.(j,\delta e_1) \in [\![\sigma A_1]\!]_\varepsilon^{\sigma k_1,\sigma t_1} \tag{1}$$

By IH 2 on the second premise using

- $FV(e_2) \subseteq dom(|\sigma\Gamma|_2)$ using Lemma 43 on the second premise
- $\models \sigma\Phi$
- $\forall j.(j,\delta') \in \mathcal{G}[\![|\sigma\Gamma|_2]\!]$ obtained by Lemma 19 on $(m,\delta,\delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$

we get

$$\forall j.(j,\delta' e_2) \in [\![\sigma A_2]\!]_\varepsilon^{\sigma k_2,\sigma t_2} \tag{2}$$

We instantiate j with $m + c_1 + 1$ in (1) and we get

$$(m + c_1 + 1, \delta e_1) \in [\![\sigma A_1]\!]_\varepsilon^{\sigma k_1,\sigma t_1} \tag{3}$$

We instantiate j with $m + c_2 + 1$ in (2) and we get

$$(m + c_2 + 1, \delta' e_2) \in [\![\sigma A_2]\!]_\varepsilon^{\sigma k_2,\sigma t_2} \tag{4}$$

Next, unrolling (3) using (a) and $c_1 < m + c_1 + 1$, we get

d) $\sigma k_1 \leqslant r_1 \leqslant \sigma t_1$
e) $(m+1, v_1) \in [\![\sigma A_1]\!]_v$

Next, unrolling second part of (4) using (b) and $c_2 < m + c_2 + 1$, we get

f) $\sigma k_2 \leqslant c_2 \leqslant \sigma t_2$
g) $(m+1, v_2) \in [\![\sigma A_2]\!]_v$

Now, we can conclude as follows:

1. By e) and g), we get $r_1 - r_2 \leqslant \sigma t_1 - \sigma k_2$

2. By downward closure (Lemma 20) on f) using

$$m \leqslant m + 1$$

we get $(m, v_1) \in [\![\sigma A_1]\!]_v$.
By downward closure (Lemma 20) on h) using

$$m \leqslant m + 1$$

we get $(m, v_2) \in [\![\sigma A_2]\!]_v$.

Case: $\dfrac{\Delta; \Phi_a \models C \qquad \Delta; \Phi_a \wedge C; \Gamma \vdash e \ominus e' \lesssim t : \tau}{\Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t : C \,\&\, \tau}$ **r-c-andI**

Assume that $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.
TS: $(m, \delta e, \delta' e') \in (\!|\sigma C \,\&\, \sigma\tau|\!)^{\sigma t}_{\varepsilon}$.
Following the definition of $(\!|\cdot|\!)^{\cdot}_{\varepsilon}\cdot$, assume that

a) $\delta e \Downarrow^{c,r} v$
b) $\delta' e' \Downarrow^{c',r'} v'$
c) $c < m$.

By IH 1 on the first premise using

- $\models \sigma(C \wedge \Phi)$ hold by the main assumption $\models \sigma\Phi$ and $\models \sigma C$ $(\star)$ obtained by Lemma 22 using the premise $\Delta; \Phi \models C$

we get $(m, \delta e, \delta' e') \in (\!|\sigma\tau|\!)^{\sigma t}_{\varepsilon}$. Unrolling its definition with (a-c), we get

d) $r - r' \leqslant \sigma t$
e) $(m - c, v, v') \in (\!|\sigma\tau|\!)_v$

We can conclude as follows:

1. By d), $r - r' \leqslant \sigma t$
2. Using e) and $(\star)$, we can show that $(m - c, v, v') \in (\!|\sigma C \,\&\, \sigma\tau|\!)_v$

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_1' \lesssim t_1 : C \,\&\, \tau_1 \quad \Delta; \Phi_a \wedge C; x : \tau_1, \Gamma \vdash e_2 \ominus e_2' \lesssim t_2 : \tau_2}{\Delta; \Phi_a; \Gamma \vdash \textbf{clet}\ e_1\ \textbf{as}\ x\ \textbf{in}\ e_2 \ominus \textbf{clet}\ e_1'\ \textbf{as}\ x\ \textbf{in}\ e_2' \lesssim t_1 + t_2 : \tau_2}$ **r-c-andE**

Assume that $(m, \delta, \delta') \in \mathcal{G}(\lfloor \sigma\Gamma \rfloor)$ and $\models \sigma\Phi$.

TS: $(m, \textbf{clet}\ \delta e_1\ \text{as}\ x\ \text{in}\ \delta e_2, \textbf{clet}\ \delta' e_1'\ \text{as}\ x\ \text{in}\ \delta' e_2') \in \lfloor \sigma\tau_2 \rfloor_\varepsilon^{\sigma t_1 + \sigma t_2}$.

Following the definition of $\lfloor \cdot \rfloor_\varepsilon^{\cdot}$, assume that

$\dfrac{\delta e_1 \Downarrow^{c_1, r_1} v_1 \quad (\diamond) \qquad \delta e_2[v_1/x] \Downarrow^{c_r, r_r} v_r \quad (\dagger)}{\textbf{clet}\ \delta e_1\ \text{as}\ x\ \text{in}\ \delta e_2 \Downarrow^{c_1 + c_r, r_1 + r_r} v_r}$ **clet** and

$\dfrac{\delta' e_1' \Downarrow^{c_1', r_1'} v_1' \quad (\diamond\diamond) \qquad \delta' e_2'[v_1'/x] \Downarrow^{c_r', r_r'} v_r' \quad (\dagger\dagger)}{\textbf{clet}\ \delta' e_1\ \text{as}\ x\ \text{in}\ \delta' e_2 \Downarrow^{c_1' + c_r', r_1' + r_r'} v_r'}$ **clet** and $(c_1 + c_r) < m$.

By IH 1 on the first premise, we get $(m, \delta e_1, \delta' e_1') \in \lfloor \sigma C \,\&\, \sigma\tau_1 \rfloor_\varepsilon^{\sigma t_1}$.

Unrolling its definition with $(\diamond)$ and $(\diamond\diamond)$ and $c_1 < m$, we get

a) $r_1 - r_1' \leqslant \sigma t_1$

b) $(m - c_1, v_1, v_1') \in \lfloor \sigma C \,\&\, \sigma\tau_1 \rfloor_v$

By IH 1 on the second premise using $(m - c_1, \delta[x \mapsto v_1], \delta'[x \mapsto v_1']) \in \mathcal{G}(\lfloor \sigma\Gamma, x : \sigma\tau_1 \rfloor)$ obtained by

- $\models \sigma(C \wedge \Phi)$ hold by the main assumption $\models \sigma\Phi$ and $\models \sigma C$ obtained by unrolling the definition of b)
- $(m - c_1, \delta, \delta') \in \mathcal{G}(\lfloor \sigma\Gamma \rfloor)$ by downward closure (Lemma 20) on $(m, \delta, \delta') \in \mathcal{G}(\lfloor \sigma\Gamma \rfloor)$ using $m - c_1 \leqslant m$
- $(m - c_1, v_1, v_1') \in \lfloor \sigma\tau_1 \rfloor_v$ by unrolling the definition of b)

we get $(m - c_1, \delta e_2[v_1/x], \delta' e_2'[v_1'/x]) \in \lfloor \sigma\tau_2 \rfloor_\varepsilon^{\sigma t_2}$. Unrolling its definition with $(\dagger)$ and $(\dagger\dagger)$, and $c_r < m - c_1$, we get

c) $r_r - r_r' \leqslant \sigma t_2$

d) $(m - (c_1 + c_r), v_r, v_r') \in \lfloor \sigma\tau_2 \rfloor_v$

Now, we can conclude with

1. By a) and c) $(r_1 + r_r) - (r_1' + r_r') \leqslant \sigma t_1 + \sigma t_2$

2. By downward closure (Lemma 20) on d) using

$$m - (c_1 + c_r) \leqslant m - (c_1 + c_r)$$

we obtain $(m - (c_1 + c_r), v_r, v_r') \in (\!|\sigma\tau_2|\!)_v$.

Case: $\dfrac{\Delta; \Phi_a \wedge C; \Gamma \vdash e \ominus e' \lesssim t : \tau \qquad \Delta \vdash C \ \mathtt{wf}}{\Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t : C \supset \tau}$ **r-c-impI**

Assume that $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.
TS: $(m, \delta e, \delta' e') \in (\!|\sigma C \,\&\, \sigma\tau|\!)^{\sigma t}_{\varepsilon}$.
Following the definition of $(\!|\cdot|\!)^{\cdot}_{\varepsilon}\cdot$, assume that

a) $\delta e \Downarrow^{c,r} v$
b) $\delta' e' \Downarrow^{c',r'} v'$
c) $c < m$.

We first show the second statement.
TS2: $(m - c, v, v') \in (\!|\sigma C \supset \sigma\tau|\!)_v$
Assume that $\models \sigma C$ $(\star)$.
STS: $(m - c, v, v') \in (\!|\sigma\ \tau|\!)_v$
By IH 1 on the first premise using

- $\models \sigma(C \wedge \Phi)$ hold by the main assumption $\models \sigma\Phi$ and $\models \sigma C$ (by $\star$)

we get $(m, \delta e, \delta' e') \in (\!|\sigma\tau|\!)^{\sigma t}_{\varepsilon}$. Unrolling its definition with (a-c), we get

d) $r - r' \leqslant \sigma t$
e) $(m - c, v, v') \in (\!|\sigma\tau|\!)_v$

We can conclude as follows:

1. By d), $r - r' \leqslant \sigma t$
2. Using e) , we can show that $(m - c, v, v') \in (\!|\sigma C \supset \sigma\tau|\!)_v$

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t : C \supset \tau \qquad \Delta; \Phi_a \models C}{\Delta; \Phi_a; \Gamma \vdash \mathbf{celim}_{\supset}\ e \ominus \mathbf{celim}_{\supset}\ e' \lesssim t : \tau}$ **r-c-implE**

Assume that $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.
TS: $(m, \mathrm{celim}_{\supset}\ \delta e, \mathrm{celim}_{\supset}\ \delta' e') \in (\!|\sigma\tau|\!)^{\sigma t}_{\varepsilon}$.

Following the definition of $(\!|\cdot|\!)^{\cdot}_{\varepsilon}\cdot$, assume that $\dfrac{\delta e \Downarrow^{c,r} v \quad (\diamond)}{\mathrm{celim}_{\supset}\ \delta e \Downarrow^{c,r} v}$ **celim** and $\dfrac{\delta' e \Downarrow^{c,r} v \quad (\diamond\diamond)}{\mathrm{celim}_{\supset}\ \delta' e \Downarrow^{c,r} v}$ **celim** and $c < m$ $(\star)$.
By IH 1 on the first premise, we get $(m, \delta e, \delta' e') \in (\!|\sigma C \supset \sigma\tau|\!)^{\sigma t}_{\varepsilon}$.
Unrolling its definition using $(\diamond)$, $(\diamond\diamond)$ and $(\star)$, we get

a) $r - r' \leqslant \sigma t$
b) $(m - c, v, v') \in (\!|\sigma C \supset \sigma\tau|\!)_{v}$

We can conclude as follows:

1. By a), $r - r' \leqslant \sigma t$
2. Using b) and $\models \sigma C$ (obtained by Lemma 22 on the second premise) , we can show that $(m - c, v, v') \in (\!|\sigma\tau|\!)_{v}$

Case: $\dfrac{\Upsilon(\zeta) = \tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2 \qquad \Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t' : \tau_1}{\Delta; \Phi_a; \Gamma \vdash \zeta\ e \ominus \zeta\ e' \lesssim t + t' : \tau_2}$ **r-primapp**
Assume that $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.
TS: $(m, \zeta\ \delta e, \zeta\ \delta' e') \in (\!|\sigma\tau_2|\!)^{\sigma t + \sigma t'}_{\varepsilon}$.
Following the definition of $(\!|\cdot|\!)^{\cdot}_{\varepsilon}$, assume that
$\dfrac{\delta e \Downarrow^{c,r} v \quad (\star) \qquad \hat{\zeta}(v) = (c_r, r_r, v_r) \quad (\diamond)}{\zeta\ \delta e \Downarrow^{c + c_r + 1, r + r_r + c_{primapp}} v_r}$ **primapp** and
$\dfrac{\delta' e' \Downarrow^{c', r'} v' \quad (\star\star) \qquad \hat{\zeta}(v)' = (c'_r, r'_r, v'_r) \quad (\diamond\diamond)}{\zeta\ \delta' e' \Downarrow^{c' + c'_r + 1, r' + r'_r + c_{primapp}} v'_r}$ **primapp** and
$(c + c_r + 1) < m$.
By IH 1 on the second premise, we get $(m, \delta e, \delta' e') \in (\!|\sigma\tau_1|\!)^{\sigma t'}_{\varepsilon}$.
Unrolling its definition with $(\star)$ and $(\star\star)$, and $c < m$, we get

a) $r - r' \leqslant \sigma t'$
b) $(m - c, v, v') \in (\!|\sigma\tau_1|\!)_{v}$

Next, by Assumption (44) using $\zeta : \sigma\tau_1 \xrightarrow{\mathrm{diff}(\sigma t)} \sigma\tau_2$ (obtained by substitution on the first premise), b), $(\star)$ and $(\star\star)$, we get

c) $r_r - r'_r \leqslant \sigma t$

d) $(m - c - c_r, v_r, v_r') \in (\!|\sigma\tau_2|\!)_v$

Now, we can conclude as follows:

1. Using a) and c), we get $(r + r_r + c_{primapp}) - (r' + r_r' + c_{primapp}) \leqslant \sigma t + \sigma t'$
2. By downward closure (Lemma 20) on d) using

$$m - (c + c_r + 1) \leqslant m - (c + c_r)$$

we get $(m - (c + c_r + 1), v_r, v_r') \in (\!|\sigma\tau_2|\!)_v$.

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash e \ominus e \lesssim t : \tau \quad \forall x \in dom(\Gamma).\ \Delta; \Phi_a \models \Gamma(x) \sqsubseteq \Box\Gamma(x)}{\Delta; \Phi_a; \Gamma, \Gamma'; \Omega \vdash e \ominus e \lesssim 0 : \Box\tau}$ **nochange**

Assume that $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma, \sigma\Gamma'|\!)$ and $\models \sigma\Phi$.
Then, $\delta = \delta_1 \cup \delta_2$ and $\delta' = \delta_1' \cup \delta_2'$ such that $(m, \delta_1, \delta_1') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $(m, \delta_2, \delta_2') \in \mathcal{G}(\!|\sigma\Gamma'|\!)$.
TS: $(m, \delta e, \delta' e) \in (\!|\Box\,\sigma\tau|\!)_\varepsilon^0$.
Since e doesn't have any free variables from Γ' by the first premise,
STS: $(m, \delta_1 e, \delta_1' e) \in (\!|\Box\,\sigma\tau|\!)_\varepsilon^0$.
Assume that

a) $\delta_1 e \Downarrow^{c,r} v$
b) $\delta_1' e \Downarrow^{c',r'} v'$
c) $c < m$

TS 1: $r - r' \leqslant 0$
TS 2: $(m - c, v, v') \in (\!|\Box\,\sigma\tau|\!)_v$
By IH 1 on the first premise using

- $(m, \delta_1, \delta_1') \in \mathcal{G}(\!|\sigma\Gamma|\!)$
- $\models \sigma\Phi$

we get $(m, \delta_1 e, \delta_1' e) \in (\!|\sigma\tau|\!)_\varepsilon^{\sigma t}$.
Unfolding its definition with a), b) and c), we get

d) $r - r' \leqslant \sigma t$

e) $(m - c, v, v') \in (\!|\sigma\tau|\!)_v$

We can conclude as follows

1. By Lemma 21 using $(m, \delta_1, \delta_1') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and the second premise, we get $(m, \delta_1, \delta_1) \in \mathcal{G}(\!|\Box\,\sigma\Gamma|\!)$. This means that $\delta_1 = \delta_1'$.
   Therefore, a) and b) are equal, that is $c = c'$, $r = r'$ and $v = v'$. Hence, trivially we get $r - r' \leqslant 0$.
2. Since $v = v'$ and $c = c'$, by using e), we get$(m - c, v, v) \in (\!|\Box\,\sigma\tau|\!)_v$.

Case: $\dfrac{\Delta; \Phi_a \wedge C; \Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau \qquad \Delta; \Phi_a \wedge \neg C; \Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau \qquad \Delta \vdash C\ \mathtt{wf}}{\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau}$ **r-split**

Assume that $\models \sigma\Phi$ and $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$.

TS: $(m, \delta e_1, \delta' e_2) \in (\!|\sigma\tau|\!)_\varepsilon^{\sigma k}$.

There are two cases:

subcase 1: $\models \sigma\Phi \wedge \sigma C$

Follows immediately by IH on the first premise using the assumption σC.

subcase 2: $\models \sigma\Phi \wedge \neg\sigma C$

Follows immediately by IH on the second premise using the assumption $\neg\sigma C$.

Case:

$$\frac{\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau \qquad \Delta; \Phi_a \models \tau \sqsubseteq \tau' \qquad \Delta; \Phi_a \models t \leqslant t'}{\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t' : \tau'}\ \textbf{r-}\sqsubseteq$$

Assume that $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m, \delta e, \delta' e') \in (\!|\sigma\tau'|\!)_\varepsilon^{\sigma t'}$.

Following the definition of $(\!|\cdot|\!)_\varepsilon^{\cdot}\cdot$, assume that

a) $\delta e \Downarrow^{c,r} v$
b) $\delta' e' \Downarrow^{c',r'} v'$
c) $c < m$.

By IH 1 on the first premise using (a-c), we get

d) $r - r' \leqslant \sigma t$

e) $(m - c, v, v') \in (\!|\sigma\tau|\!)_v$

We can conclude as

1. By Assumption (25) on the third premise, we get $\sigma t \leqslant \sigma t'$. Combining this with d), we get $r - r' \leqslant \sigma t'$.
2. By Lemma 21 on the second premise using e), we get $(m - c, v, v') \in (\!|\sigma\tau'|\!)_v$

Case: $\dfrac{\Delta; \Phi_a; |\Gamma|_1 \vdash^{t_1}_{k_1} e_1 : A_1 \qquad \Delta; \Phi_a; x : \mathrm{U}\, A_1, \Gamma \vdash e_2 \ominus e \lesssim t_2 : \tau_2}{\Delta; \Phi_a; \Gamma \vdash \textbf{let}\ x = e_1\ \textbf{in}\ e_2 \ominus e \lesssim t_1 + t_2 + c_{let} : \tau_2}$ **r-let-e**

Assume that $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m, \text{let } x = \delta e_1 \text{ in } \delta e_2, \delta' e) \in (\!|\sigma\tau_2|\!)^{\sigma t_1 + \sigma t_2 + c_{let}}_{\varepsilon}$.

Following the definition of $(\!|\cdot|\!)^{\cdot}_{\varepsilon}$, assume that

$\dfrac{\delta e_1 \Downarrow^{c_1, r_1} v_1 \quad (\diamond) \qquad \delta e_2[v_1/x] \Downarrow^{c_r, r_r} v_r \quad (\dagger)}{\text{let } x = \delta e_1 \text{ in } \delta e_2 \Downarrow^{c_1 + c_r + 1, r_1 + r_r + c_{let}} v_r}$ **let** and $\delta' e \Downarrow^{c', r'} v'$ $(\star)$ and $(c_1 + c_r + 1) < m$.

To be able to instantiate the IH 1 on the second premise, we first show

$$\forall m.(m, v_1) \in [\![\sigma A_1]\!]_v \tag{1}$$

Subproof. Pick m.

By IH 2 on the first premise using

- $FV(e_1) \subseteq \text{dom}(|\sigma\Gamma|_1)$ using Lemma 43 on the first premise
- $(m + c_1 + 1, \delta) \in \mathcal{G}[\![|\sigma\Gamma|_1]\!]$ obtained by Lemma 19 using $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$

we get

$$(m + c_1 + 1, \delta e_1) \in [\![\sigma A_1]\!]^{\sigma k_1, \sigma t_1}_{\varepsilon} \tag{2}$$

Unfolding the definition of (2) with ($\diamond$) and $c_1 < m + c_1 + 1$, we get

a) $\sigma k_1 \leqslant r_1 \leqslant \sigma t_1$
b) $(m+1, v_1) \in [\![\sigma A_1]\!]_v$

RTS: $(m, v_1) \in [\![\sigma A_1]\!]_v$.
This follows by downward closure (Lemma 20) on (b) using $m \leqslant m+1$. ■

Next, we instantiate IH 1 on the second premise using

- $(m, \delta[x \mapsto v_1], \delta'[x \mapsto v_1]) \in \mathcal{G}(\!|\sigma\Gamma, x : U\,\sigma A_1|\!)$ using
  - $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$
  - $(m, v_1, v_1) \in (\!|U\,\sigma A_1|\!)_v$ using (1)

and we get $(m, \delta e_2[v_1/x], \delta' e[v_1/x]) \in (\!|\sigma\tau_2|\!)_\varepsilon^{\sigma t_2}$.
Since x doesn't occur free in e, we have
$(m, \delta e_2[v_1/x], \delta' e) \in (\!|\sigma\tau_2|\!)_\varepsilon^{\sigma t_2}$.
Unrolling its definition with ($\dagger$) and ($\star$), and $c_r < m$, we get

f) $r_r - r' \leqslant \sigma t_2$
g) $(m - c_r, v_r, v') \in (\!|\sigma\tau_2|\!)_v$

Now, we can conclude by

1. By a) and g) $(r_1 + r_r + c_{let}) - r' \leqslant \sigma t_1 + \sigma t_2 + c_{let}$
2. By downward closure (Lemma 20) on h) using

$$m - (c + c_r + 1) \leqslant m - (c + c_r)$$

we get $(m - (c + c_r + 1), v_r, v') \in (\!|\sigma\tau_2|\!)_v$.

Case: $\dfrac{\Delta; \Phi_a; |\Gamma|_2 \vdash^{t_1}_{k_1} e_1 : A_1 \qquad \Delta; \Phi_a; x : U\,A_1, \Gamma \vdash e \ominus e_2 \lesssim t_2 : \tau_2}{\Delta; \Phi_a; \Gamma \vdash e \ominus \textbf{let}\ x = e_1\ \textbf{in}\ e_2 \lesssim t_2 - k_1 - c_{let} : \tau_2}$ **r-e-let**

Assume that $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.
TS: $(m, \delta e, \text{let } x = \delta' e_1 \text{ in } \delta' e_2) \in (\!|\sigma\tau_2|\!)_\varepsilon^{\sigma t_2 - \sigma k_1 - c_{let}}$.
Following the definition of $(\!|\cdot|\!)_\varepsilon$, assume that

$\delta e \Downarrow^{c,r} v$ $(\star)$ and $\dfrac{\delta' e_1 \Downarrow^{c_1,r_1} v_1 \quad (\diamond) \qquad \delta' e_2[v_1/x] \Downarrow^{c_r,r_r} v_r \quad (\dagger)}{\text{let } x = \delta' e_1 \text{ in } \delta' e_2 \Downarrow^{c_1+c_r+1,r_1+r_r+c_{let}} v_r}$ **let** and $c < m$.

To be able to instantiate the IH 1 on the second premise, we first show

$$\forall m.(m, v_1) \in [\![\sigma A_1]\!]_v \tag{1}$$

Subproof. Pick m.
By IH 2 on the first premise using

- $FV(e_1) \subseteq dom(|\sigma\Gamma|_2)$ using Lemma 43 on the first premise
- $(m + c_1 + 1, \delta') \in \mathcal{G}[\![|\sigma\Gamma|_2]\!]$ obtained by Lemma 19 using $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$

we get

$$(m + c_1 + 1, \delta' e_1) \in [\![\sigma A_1]\!]_\varepsilon^{\sigma k_1, \sigma t_1} \tag{2}$$

Unfolding the definition of (2) with $(\diamond)$ and $c_1 < m + c_1 + 1$, we get

a) $\sigma k_1 \leqslant r_1 \leqslant \sigma t_1$
b) $(m + 1, v_1) \in [\![\sigma A_1]\!]_v$

RTS: $(m, v_1) \in [\![\sigma A_1]\!]_v$.
This follows by downward closure (Lemma 20) on (b) using $m \leqslant m + 1$. ■

Next, we instantiate IH 1 on the second premise using

- $(m, \delta[x \mapsto v_1], \delta'[x \mapsto v_1]) \in \mathcal{G}(\!|\sigma\Gamma, x : U\ \sigma A_1|\!)$ using
  - $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$
  - $(m, v_1, v_1) \in (\!|U\ \sigma A_1|\!)_v$ using (1)

and we get $(m, \delta e[v_1/x], \delta' e_2[v_1/x]) \in (\!|\sigma\tau_2|\!)_\varepsilon^{\sigma t_2}$.
Since x doesn't occur free in e, we have

$(m, \delta e, \delta' e_2[v_1/x]) \in (\!|\sigma\tau_2|\!)_\varepsilon^{\sigma t_2}$.
Unrolling its definition with $(\dagger)$ and $(\star)$, and $c < m$, we get

h) $r - r_r \leqslant \sigma t_2$
i) $(m - c, v, v_r) \in (\!|\sigma\tau_2|\!)_v$

Now, we can conclude by

1. By a) and h) $r - (r_1 + r_r + c_{let}) - r \leqslant \sigma t_2 - \sigma k_1 - c_{let}$
2. By i), we have $(m - c, v, v_r) \in (\!|\sigma\tau_2|\!)_v$.

Case: $\dfrac{\Delta; \Phi_a; |\Gamma|_1 \vdash^{t} e : A_1 + A_2 \qquad \Delta; \Phi_a; x : U\,A_1, \Gamma \vdash e_1 \ominus e' \lesssim t' : \tau \qquad \Delta; \Phi_a; y : U\,A_2, \Gamma \vdash e_2 \ominus e' \lesssim t' : \tau}{\Delta; \Phi_a; \Gamma \vdash \textbf{case}\ (e, x.e_1, y.e_2) \ominus e' \lesssim t' + t + c_{case} : \tau}$ **r-case-e**

Assume that $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.
TS: $(m,\ \text{case}\ (\delta e, \delta e_1, \delta e_2), \delta' e') \in (\!|\sigma\tau|\!)_\varepsilon^{\sigma t + \sigma t'}$.
Following the definition of $(\!|\cdot|\!)_\varepsilon^{\cdot}$, assume that
$\text{case}\ (\delta e, \delta e_1, \delta e_2) \Downarrow^{C,R} v_r$ and $\delta' e' \Downarrow^{c',r'} v'$ $(\dagger)$ and $C < m$.
Depending on what δe evaluates to, there are two cases:

subcase 1: $\dfrac{\delta e \Downarrow^{c,r} \text{inl}\ v \quad (\star) \qquad \delta e_1[v/x] \Downarrow^{c_r, r_r} v_r \quad (\diamond)}{\text{case}\ (\delta e, x.\delta e_1, y.\delta e_2) \Downarrow^{c + c_r + 1, r + r_r + c_{case}} v_r}$ **case-inl**

Note that $C = c + c_r + 1 < m$.
To be able to instantiate the IH 1 on the second premise, we first show

$$\forall m.(m, \text{inl}\ v) \in [\![\sigma A_1 + \sigma A_2]\!]_v \tag{1}$$

Subproof. Pick m.
By IH 2 on the first premise using

- $FV(e) \subseteq \text{dom}(|\sigma\Gamma|_1)$ using Lemma 43 on the first premise
- $(m + c + 1, \delta) \in \mathcal{G}[\![|\sigma\Gamma|_1]\!]$ obtained by Lemma 19 using $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$

we get

$$(m + c + 1, \delta e_1) \in [\![\sigma A_1 + \sigma A_2]\!]_\varepsilon^{\sigma k, \sigma t} \tag{2}$$

Unfolding the definition of (2) with $(\star)$ and $c < m + c + 1$, we get

a) $c \leqslant \sigma t$
b) $(m + c + 1 - c, \text{inl } v) \in [\![\sigma A_1 + \sigma A_2]\!]_v$

RTS: $(m, \text{inl } v) \in [\![\sigma A_1 + \sigma A_2]\!]_v$.
This follows by downward closure (Lemma 20) on b) using $m \leqslant m + 1$. ■

Next, we instantiate IH 1 on the second premise using

- $(m, \delta[x \mapsto v], \delta'[x \mapsto v]) \in \mathcal{G}(\!|\sigma\Gamma, x : U\ \sigma A_1|\!)$ using
  - $(m, \delta, \delta') \in \mathcal{G}(\!|\sigma\Gamma|\!)$
  - $(m, v, v) \in (\!|U\ \sigma A_1|\!)_v$ by unrolling the definition of (1)

and we get $(m, \delta e_1[v/x], \delta' e'[v/x]) \in (\!|\sigma\tau|\!)_\varepsilon^{\sigma t'}$.
Since x doesn't occur free in e', we have
$(m, \delta e_1[v/x], \delta' e') \in (\!|\sigma\tau|\!)_\varepsilon^{\sigma t'}$.
Unrolling its definition with $(\diamond)$ and $(\dagger)$, and $c_r < m$, we get

j) $r_r - r' \leqslant \sigma t'$
k) $(m - c_r, v_r, v') \in (\!|\sigma\tau|\!)_v$

Now, we can conclude by

1. By b) and i) $(r + r_r + c_{case}) - r' \leqslant \sigma t + \sigma t' + c_{case}$
2. By downward closure (Lemma 20) on j), using

$$m - (c + c_r + 1) \leqslant m - c_r$$

we obtain $(m - (c + c_r + 1), v_r, v') \in (\!|\sigma\tau|\!)_v$.

subcase 2: $\dfrac{\delta e \Downarrow^{c,r} \text{inr } v \quad (\star) \qquad \delta e_2[v/y] \Downarrow^{c_r,r_r} v_r \quad (\diamond)}{\text{case } (\delta e, x.\delta e_1, y.\delta e_2) \Downarrow^{c+c_r+1,r+r_r+c_{case}} v_r}$ **case-inr**

Note that $C = c + c_r < m$.

Like in the previous case, we have

$$\forall m.(m, \text{inr } v) \in [\![\sigma A_1 + \sigma A_2]\!]_v \tag{3}$$

Next, we instantiate IH 1 on the third premise using

- $(m, \delta[y \mapsto v], \delta'[y \mapsto v]) \in \mathcal{G}(|\sigma\Gamma, y : U\ \sigma A_2|)$ using
  - $(m, \delta, \delta') \in \mathcal{G}(|\sigma\Gamma|)$
  - $(m, v, v) \in (|U\ \sigma A_2|)_v$ by unrolling the definition of (3)

and we get $(m, \delta e_2[v/y], \delta' e'[v/y]) \in (|\sigma\tau|)_\varepsilon^{\sigma t'}$.

Since y doesn't occur free in e', we have

$(m, \delta e_2[v/y], \delta' e') \in (|\sigma\tau|)_\varepsilon^{\sigma t'}$.

Unrolling its definition with $(\diamond)$ and $(\dagger)$, and $c_r < m$, we get

l) $r_r - r' \leqslant \sigma t'$

m) $(m - c_r, v_r, v') \in (|\sigma\tau|)_v$

Now, we can conclude by

1. By b) and k) $(r + r_r + c_{case}) - r' \leqslant \sigma t + \sigma t' + c_{case}$
2. By downward closure (Lemma 20) on l), using

$$m - (c + c_r + 1) \leqslant m - c_r$$

we obtain $(m - (c + c_r + 1), v_r, v') \in (|\sigma\tau|)_v$.

□

Proof of Statement (2). Remember the statement (2) of Theorem 30:
Assume that $\Delta; \Phi_a; \Omega \vdash^t_k e : A$ and $\sigma \in \mathcal{D}[\![\Delta]\!]$ and $\models \sigma\Phi$ and there exists Ω' s.t. $FV(e) \subseteq \text{dom}(\Omega')$ and $\Omega' \subseteq \Omega$ and $(m, \gamma) \in \mathcal{G}[\![\sigma\Omega']\!]$. Then, $(m, \gamma e) \in [\![\sigma A]\!]_\varepsilon^{\sigma k, \sigma t}$.

Proof is by induction on the typing of e. We show a few selected cases.

Case: $\dfrac{\Omega(x) = A}{\Delta; \Phi_a; \Omega \vdash^{0}_{0} x : A}$ **var**

Assume that $\models \sigma\Phi$ and there exists Ω' s.t. $FV(x) \subseteq dom(\Omega')$ and $\Omega' \subseteq \Omega$ and $(m, \delta) \in \mathcal{G}[\![\sigma\Omega']\!]$

TS: $(m, \gamma(x)) \in [\![\sigma A]\!]^{0,0}_{\varepsilon}$.

By Value Lemma (Lemma 18),

STS: $(m, \gamma(x)) \in [\![\sigma A]\!]_{v}$.

Note that $x \in dom(\Omega')$ and $\Omega' \subseteq \Omega$, therefore $\Omega'(x) = A$,

RTS: $(m, \gamma(x)) \in [\![\sigma A]\!]_{v}$.

This follows by the premise $\Omega(x) = A$ and the main assumption $(m, \gamma) \in \mathcal{G}[\![\sigma\Omega]\!]$

Case: $\dfrac{\Delta; \Phi_a; \Omega \vdash^{t_1}_{k_1} e_1 : A \qquad \Delta; \Phi_a; \Omega \vdash^{t_2}_{k_2} e_2 : \mathbf{list}[n]\, A}{\Delta; \Phi_a; \Omega \vdash^{t_1+t_2}_{k_1+k_2} \mathbf{cons}(e_1, e_2) : \mathbf{list}[n+1]\, A}$ **cons**

Assume that $\models \sigma\Phi$ and there exists Ω' s.t. $FV(\mathrm{cons}(e_1, e_2)) \subseteq dom(\Omega')$ and $\Omega' \subseteq \Omega$ and $(m, \delta) \in \mathcal{G}[\![\sigma\Omega']\!]$

TS: $(m, \mathrm{cons}(\gamma e_1, \gamma e_2)) \in [\![\mathrm{list}[\sigma n + 1]\, \sigma A]\!]^{\sigma k_1 + \sigma k_2, \sigma t_1 + \sigma t_2}_{\varepsilon}$.

Following the definition of $[\![\cdot]\!]^{\cdot,\cdot}_{\varepsilon}$,

Assume that $\dfrac{\gamma e_1 \Downarrow^{c_1, r_1} v_1 \quad (\star) \qquad \gamma e_2 \Downarrow^{c_2, r_2} v_2 \quad (\diamond)}{\mathrm{cons}(\gamma e_1, \gamma e_2) \Downarrow^{c_1+c_2, r_1+r_2} \mathrm{cons}(v_1, v_2)}$ **cons** and $c_1 + c_2 < m$.

By IH 2 on the first premise using

$$FV(e_1) \subseteq dom(\Omega') \text{ and } \Omega' \subseteq \Omega \text{ and } (m, \delta) \in \mathcal{G}[\![\sigma\Omega']\!]$$

we get $(m, \gamma e_1) \in [\![\sigma A]\!]^{\sigma k_1, \sigma t_1}_{\varepsilon}$. Unrolling its definition with $(\star)$ and $c_1 < m$, we get

a) $\sigma k_1 \leqslant r_1 \leqslant \sigma t_1$

b) $(m - c_1, v_1) \in [\![\sigma A]\!]_{v}$

By IH 2 on the second premise using

$$\mathrm{FV}(e_2) \subseteq \mathrm{dom}(\Omega') \text{ and } \Omega' \subseteq \Omega \text{ and } (m, \delta) \in \mathcal{G}[\![\sigma\Omega']\!]$$

we get $(m, \gamma e_2) \in [\![\mathrm{list}[\sigma n]\ \sigma A]\!]_\varepsilon^{\sigma k_2, \sigma t_2}$.
Unrolling its definition with $(\diamond)$ and $c_2 < m$, we get

c) $\sigma k_2 \leqslant r_2 \leqslant \sigma t_2$
d) $(m - c_2, v_2) \in [\![\mathrm{list}[\sigma n]\ \sigma A]\!]_v$

Now, we can conclude as follows:

1. Using a) and c), we get $\sigma k_1 + \sigma k_2 \leqslant (r_1 + r_2) \leqslant \sigma t_1 + \sigma t_2$
2. By downward closure (Lemma 20) on b) using

$$m - (c_1 + c_2) \leqslant m - c_1$$

we get $(m - (c_1 + c_2), v_1) \in [\![\sigma A]\!]_v$.
By downward closure (Lemma 20) on d) using

$$m - (c_1 + c_2) \leqslant m - c_2$$

we get $(m - (c_1 + c_2), v_2) \in [\![\mathrm{list}[\sigma n]\ \sigma A]\!]_v$.
By combining these two statements, we can conclude as $(m - (c_1 + c_2), \mathrm{cons}(v_1, v_2)) \in [\![\mathrm{list}[\sigma n + 1]\ \sigma A]\!]_v$

Case: $\dfrac{\Delta \vdash^A A_1 \xrightarrow{\mathbf{exec}(k,t)} A_2\ \mathtt{wf} \qquad \Delta; \Phi_a; x : A_1, f : A_1 \xrightarrow{\mathbf{exec}(k,t)} A_2, \Omega \vdash_k^t e : A_2}{\Delta; \Phi_a; \Omega \vdash_0^0 \mathbf{fix}\ f(x).e : A_1 \xrightarrow{\mathbf{exec}(k,t)} A_2}\ \mathbf{fix}$

Assume that $\models \sigma\Phi$ and there exists Ω' s.t. $\mathrm{FV}(\mathrm{fix}\ f(x)) \subseteq \mathrm{dom}(\Omega')$ and $\Omega' \subseteq \Omega$ and $(m, \delta) \in \mathcal{G}[\![\sigma\Omega']\!]$
TS: $(m, \mathrm{fix}\ f(x).\gamma e) \in [\![\sigma A_1 \xrightarrow{\mathrm{exec}(\sigma k, \sigma t)} \sigma A_2]\!]_\varepsilon^{0,0}$.

By Lemma 18, STS: $(m, \text{fix } f(x).\gamma e) \in [\![\sigma A_1 \xrightarrow{\text{exec}(\sigma k,\sigma t)} \sigma A_2]\!]_v$.
We prove the more general statement

$$\forall m' \leqslant m.\ (m', \text{fix } f(x).\gamma e) \in [\![\sigma A_1 \xrightarrow{\text{exec}(\sigma k,\sigma t)} \sigma A_2]\!]_v$$

by subinduction on m'.
There are two cases:

subcase 1: $m' = 0$

Since there is no non-negative j such that $j < 0$, the goal is vacuously true.

subcase 2: $m' = m'' + 1 \leqslant m$

By sub-IH

$$(m'', \text{fix } f(x).\gamma e) \in [\![\sigma A_1 \xrightarrow{\text{exec}(\sigma k,\sigma t)} \sigma A_2]\!]_v \tag{1}$$

STS: $(m'' + 1, \text{fix } f(x).\gamma e) \in [\![\sigma A_1 \xrightarrow{\text{exec}(\sigma k,t)} \sigma A_2]\!]_v$.
Pick $j < m'' + 1$ and assume that $(j, v) \in [\![\sigma A_1]\!]_v$.
STS: $(j, \gamma e[v/x, (\text{fix } f(x).\gamma e)/f]) \in [\![\sigma A_2]\!]_\varepsilon^{\sigma k,\sigma t}$.
This follows by IH on the premise instantiated with

- $FV(e) \subseteq \text{dom}(x : A_1, f : A_1 \xrightarrow{\text{exec}(k,t)} A_2, \Omega')$ and $x : A_1, f : A_1 \xrightarrow{\text{exec}(k,t)} A_2, \Omega' \subseteq x : A_1, f : A_1 \xrightarrow{\text{exec}(k,t)} A_2, \Omega$
- $(j, \gamma[x \mapsto v, f \mapsto (\text{fix } f(x).\gamma e)]) \in \mathcal{G}[\![\sigma\Omega', x : \sigma A_1, f : \sigma A_1 \xrightarrow{\text{exec}(\sigma k,\sigma t)} \sigma A_2]\!]$ which holds because
  - $(j, \gamma) \in \mathcal{G}[\![\sigma\Omega']\!]$ obtained by downward closure (Lemma 20) on $(m, \gamma) \in \mathcal{G}[\![\sigma\Omega']\!]$ using $j < m'' + 1 \leqslant m$.
  - $(j, v) \in [\![\sigma A_1]\!]_v$, from the assumption above
  - $(j, \text{fix } f(x).\gamma e) \in [\![\sigma A_1 \xrightarrow{\text{exec}(\sigma k,\sigma t)} \sigma A_2]\!]_v$, obtained by downward closure (Lemma 20) on (1) using $j \leqslant m''$

Case: $\dfrac{\Delta; \Phi_a; \Omega \vdash^{t_1}_{k_1} e_1 : A_1 \xrightarrow{\text{exec}(k,t)} A_2 \qquad \Delta; \Phi_a; \Omega \vdash^{t_2}_{k_2} e_2 : A_1}{\Delta; \Phi_a; \Omega \vdash^{t_1+t_2+t+c_{app}}_{k_1+k_2+k+c_{app}} e_1\ e_2 : A_2}$ **app**

Assume that $\models \sigma\Phi$ and there exists Ω' s.t. $FV(e_1\ e_2) \subseteq \text{dom}(\Omega')$

and $\Omega' \subseteq \Omega$ and $(m, \delta) \in \mathcal{G}[\![\sigma\Omega']\!]$
TS: $(m, \gamma e_1\ \gamma e_2) \in [\![\sigma A_2]\!]_\varepsilon^{\sigma k_1+\sigma k_2+\sigma k+c_{app},\sigma t_1+\sigma t_2+\sigma t+c_{app}}$.
Following the definition of $[\![\cdot]\!]_\varepsilon^{\cdot,\cdot}$, Assume that

$$\frac{\gamma e_1 \Downarrow^{c_1,r_1} \text{fix } f(x).e \quad (\star) \qquad \gamma e_2 \Downarrow^{c_2,r_2} v_2 \quad (\diamond) \qquad e[v_2/x, (\text{fix } f(x).e)/f] \Downarrow^{c_r,r_r} v_r \quad (\dagger)}{\gamma e_1\ \gamma e_2 \Downarrow^{c_1+c_2+c_r+1,r_1+r_2+r_r+c_{app}} v_r}\ \textbf{app}$$

and $c_1 + c_2 + c_r + 1 < m$.
By IH 2 on the first premise using

$$FV(e_1) \subseteq \text{dom}(\Omega') \text{ and } \Omega' \subseteq \Omega \text{ and } (m, \delta) \in \mathcal{G}[\![\sigma\Omega']\!]$$

we get $(m, \gamma e_1) \in [\![\sigma A_1 \xrightarrow{\text{exec}(\sigma k,\sigma t)} \sigma A_2]\!]_\varepsilon^{\sigma k_1,\sigma t_1}$.
Unrolling its definition with $(\star)$ and $c_1 < m$, we get

a) $\sigma k_1 \leqslant r_1 \leqslant \sigma t_1$
b) $(m - c_1, \text{fix } f(x).e) \in [\![\sigma A_1 \xrightarrow{\text{exec}(\sigma k,\sigma t)} \sigma A_2]\!]_v$

By IH 2 on the second premise using

$$FV(e_2) \subseteq \text{dom}(\Omega') \text{ and } \Omega' \subseteq \Omega \text{ and } (m, \delta) \in \mathcal{G}[\![\sigma\Omega']\!]$$

we get $(m, \gamma e_2) \in [\![\sigma A_1]\!]_\varepsilon^{\sigma k_2,\sigma t_2}$. Unrolling its definition with $(\diamond)$ and $c_2 < m$, we get

c) $\sigma k_2 \leqslant r_2 \leqslant \sigma t_2$
d) $(m - c_2, v_2) \in [\![\sigma A_1]\!]_v$

By downward closure (Lemma 20) on d) using $m - c_1 - c_2 - 1 \leqslant m - c_2$, we get

$$(m - (c_1 + c_2 + 1), v_2) \in [\![\sigma A_1]\!]_v \tag{1}$$

Next, we unroll b) with (1) and $m - (c_1 + c_2 + 1) < m - c_1$ to obtain

$$(m - (c_1 + c_2 + 1), e[v_2/x, (\text{fix } f(x).e)]) \in [\![\sigma A_2]\!]_\varepsilon^{\sigma k,\sigma t} \tag{2}$$

By unrolling (2)'s definition using (†) and $c_r < m - (c_1 + c_2 + 1)$, we get

e) $\sigma k \leqslant r_r \leqslant \sigma t$

f) $(m - (c_1 + c_2 + c_r + 1), v_r) \in [\![\sigma A_2]\!]_v$

Now, we can conclude as follows:

1. Using a), c) and e), we get $\sigma k_1 + \sigma k_2 + \sigma k + c_{app} \leqslant (r_1 + r_2 + r_r + c_{app}) \leqslant \sigma t_1 + \sigma t_2 + \sigma t + c_{app}$
2. By f)

Case: $\dfrac{\Delta; \Phi_a; \Omega \vdash^t_k e : A_1 \qquad \Delta \vdash^A A_2 \ \mathtt{wf}}{\Delta; \Phi_a; \Omega \vdash^t_k \mathbf{inl}\ e : A_1 + A_2}\ \mathbf{inl}$

Assume that $\models \sigma\Phi$ and there exists Ω' s.t. $FV(\text{inl } e) \subseteq dom(\Omega')$ and $\Omega' \subseteq \Omega$ and $(m, \delta) \in \mathcal{G}[\![\sigma\Omega']\!]$

TS: $(m, \text{inl } (\gamma e)) \in [\![\sigma A_1 + \sigma A_2]\!]^{\sigma k, \sigma t}_{\varepsilon}$.

Following the definition of $[\![\cdot]\!]^{\cdot,\cdot}_{\varepsilon}$, assume that

$\dfrac{\gamma e \Downarrow^{c,r} v \quad (\star)}{\text{inl } \gamma e \Downarrow^{c,r} \text{inl } v}\ \mathbf{inl}$ and $c < m$.

By IH 2 on the first premise using

$$FV(e) \subseteq dom(\Omega') \text{ and } \Omega' \subseteq \Omega \text{ and } (m, \delta) \in \mathcal{G}[\![\sigma\Omega']\!]$$

we get $(m, \gamma e) \in [\![\sigma A]\!]^{\sigma k, \sigma t}_{\varepsilon}$.

Unrolling its definition with $(\star)$ and $c < m$, we get

a) $\sigma k \leqslant r \leqslant \sigma t$

b) $(m - c, v) \in [\![\sigma A]\!]_v$

We can conclude as follows:

1. By a), $\sigma k \leqslant r \leqslant \sigma t$
2. By b), we can show that $(m - c, \text{inl } v) \in [\![\sigma A_1 + \sigma A_2]\!]_v$

Case: $\dfrac{\begin{array}{c}\Delta;\Phi_a;\Omega \vdash^{t}_{k} e : A_1 + A_2 \\ \Delta;\Phi_a;x:A_1,\Omega \vdash^{t'}_{k'} e_1 : A \\ \Delta;\Phi_a;y:A_2,\Omega \vdash^{t'}_{k'} e_2 : A\end{array}}{\Delta;\Phi_a;\Omega \vdash^{t+t'+c_{case}}_{k+k'+c_{case}} \textbf{case}\ (e, x.e_1, y.e_2) : A}\ \textbf{case}$

Assume that $\models \sigma\Phi$ and there exists Ω' s.t. $FV(\text{ case } (e, e_1, e_2)) \subseteq dom(\Omega')$ and $\Omega' \subseteq \Omega$ and $(m, \delta) \in \mathcal{G}[\![\sigma\Omega']\!]$

TS: $(m, \text{ case } (\gamma e, \gamma e_1, \gamma e_2)) \in [\![\sigma A]\!]_\varepsilon^{\sigma k, \sigma t + \sigma t' + c_{case}}$.

Following the definition of $[\![\cdot]\!]'_\varepsilon$, assume that

$\dfrac{\gamma e \Downarrow^{c,r} \text{inl } v \quad (\star) \qquad \gamma e_1[v/x] \Downarrow^{c_r,r_r} v_r \quad (\diamond)}{\text{case } (\gamma e, x.\gamma e_1, y.\gamma e_2) \Downarrow^{c+c_r+1, r+r_r+c_{case}} v_r}\ \textbf{case-inl}$ and $c + c_r + 1 < m$.

By IH 2 on the first premise using

$$FV(e) \subseteq dom(\Omega') \text{ and } \Omega' \subseteq \Omega \text{ and } (m, \delta) \in \mathcal{G}[\![\sigma\Omega']\!]$$

we get $(m, \gamma e) \in [\![\sigma A_1 + \sigma A_2]\!]_\varepsilon^{\sigma k, \sigma t}$.

Unrolling second part of its definition with $(\star)$ and $c < m$, we get

a) $\sigma k \leqslant r \leqslant \sigma t$

b) $(m - c, \text{inl } v) \in [\![\sigma A_1 + \sigma A_2]\!]_v$

By IH 2 on the second premise using $(m - c, \gamma[x \mapsto v]) \in \mathcal{G}[\![\sigma\Omega', x : \sigma A_1]\!]$ obtained by

- $FV(e_1) \subseteq dom(x : A_1, \Omega')$ and $x : A_1, \Omega' \subseteq x : A_1, \Omega$
- $(m - c), \gamma) \in \mathcal{G}[\![\sigma\Omega']\!]$ by downward-closure (Lemma 20) on $(m, \gamma) \in \mathcal{G}[\![\sigma\Omega']\!]$ using $m - c \leqslant m$
- $(m - c, v) \in [\![\sigma A_1]\!]_v$ by downward closure (Lemma 20) on c), and unfolding its definition

we get

$$(m - c, \gamma e_1[v/x]) \in [\![\sigma A]\!]_\varepsilon^{\sigma k', \sigma t'} \tag{1}$$

By unrolling second part of (1)'s definition using ($\diamond$) and $c_r < m - c$, we get

c) $\sigma k' \leqslant r_r \leqslant \sigma t'$

d) $(m - (c + c_r), v_r) \in [\![\sigma A]\!]_v$

Now, we can conclude as follows

1. By a) and c) $\sigma k + \sigma k' + c_{case} \leqslant (r + r_r + c_{case}) \leqslant \sigma t + \sigma t' + c_{case}$
2. By downward closure (Lemma 20) on d) using

$$m - (c + c_r + 1) \leqslant m - (c + c_r)$$

we get $(m - (c + c_r + 1), v_r) \in [\![\sigma A]\!]_v$.

Case:
$$\frac{\begin{array}{c}\Delta; \Phi_a; \Omega \vdash^{t}_{k} e : \textbf{list}[n]\ A \\ \Delta; \Phi_a \wedge n = 0; \Omega \vdash^{t'}_{k'} e_1 : A' \\ i, \Delta; \Phi_a \wedge n = i + 1; h : A, tl : \textbf{list}[i]\ A, \Omega \vdash^{t'}_{k'} e_2 : A'\end{array}}{\Delta; \Phi_a; \Omega \vdash^{t+t'+c_{caseL}}_{k+k'+c_{caseL}} \textbf{case}\ e\ \textbf{of nil}\ \to e_1 \mid h :: tl \to e_2 : A'}\ \textbf{caseL}$$

Assume that $\models \sigma\Phi$ and there exists Ω' s.t. $FV(E) \subseteq dom(\Omega')$ and $\Omega' \subseteq \Omega$ and $(m, \delta) \in \mathcal{G}[\![\sigma\Omega']\!]$ where

$E =$ case e of nil $\to e_1 \mid h :: tl \to e_2$

TS: $(m,$ case γe of nil $\to \gamma e_1 \mid h :: tl \to \gamma e_2) \in [\![\sigma A']\!]^{\sigma k + \sigma k' + c_{caseL}, \sigma t + \sigma t' + c_{caseL}}_{\varepsilon}$.

Following the definition of $[\![\cdot]\!]^{\cdot,\cdot}_{\varepsilon}$, assume that

case γe of nil $\to \gamma e_1 \mid h :: tl \to \gamma e_2 \Downarrow^{C,R} v_r$ and $C < m$.

Depending on what γe evaluates to, there are two cases.

subcase 1:
$$\frac{\gamma e \Downarrow^{c,r} \text{nil} \quad (\star) \qquad \gamma e_1 \Downarrow^{c_r, r_r} v_r \quad (\diamond)}{\text{case } \gamma e \text{ of nil } \to \gamma e_1 \mid h :: tl \to \gamma e_2 \Downarrow^{c + c_r + 1, r + r_r + c_{caseL}} v_r}\ \textbf{caseL-nil}$$

and $C = c + c_r + 1$

By IH 2 on the first premise using

$$FV(e) \subseteq dom(\Omega') \text{ and } \Omega' \subseteq \Omega \text{ and } (m, \delta) \in \mathcal{G}[\![\sigma\Omega']\!]$$

we get $(m, \gamma e) \in [\![\text{list}[\sigma n]\ \sigma A]\!]^{\sigma k, \sigma t}_{\varepsilon}$. Unrolling its definition with $(\star)$ and $c < m$, we get

a) $\sigma k \leqslant r \leqslant \sigma t$

b) $(m - c, \text{nil}\ ) \in [\![\text{list}[\sigma n]\ \sigma A]\!]_v$

By b), $\sigma n = 0$ since $v = \text{nil}$.
Then, we can instantiate IH 2 on the second premise using

- $FV(e_1) \subseteq \text{dom}(\Omega')$ and $\Omega' \subseteq \Omega$ and $(m, \delta) \in \mathcal{G}[\![\sigma\Omega']\!]$
- $\models \sigma\Phi \wedge \sigma n \doteq 0$ obtained by combining $\models \sigma\Phi$ with $\models \sigma n \doteq 0$

we get $(m, \gamma e_1) \in [\![\sigma A']\!]_\varepsilon^{\sigma k', \sigma t'}$.
Unrolling its definition using $(\diamond)$ and $c_r < m$, we get

c) $\sigma k' \leqslant r_r \leqslant \sigma t'$

d) $(m - c_r, v_r) \in [\![\sigma A']\!]_v$

We conclude with

1. By a) and c), we get $\sigma k + \sigma k' + c_{\text{caseL}} \leqslant r + r_r + c_{\text{caseL}} \leqslant \sigma t + \sigma t' + c_{\text{caseL}}$
2. By downward closure (Lemma 20) on d) using

$$m - (c + c_r + 1) \leqslant m - (c + c_r)$$

we get $(m - (c + c_r + 1), v_r) \in [\![\sigma A']\!]_v$.

subcase 2:

$$\frac{\gamma e \Downarrow^{c,r} \text{cons}(v_1, v_2)\ \ (\star) \qquad \gamma e_2[v_1/h, v_2/tl] \Downarrow^{c_r, r_r} v_r\ \ (\diamond\diamond)}{\text{case}\ \gamma e\ \text{of nil}\ \to \gamma e_1 \mid h :: tl \to \gamma e_2 \Downarrow^{c+c_r+1, r+r_r+c_{\text{caseL}}} v_r}\ \textbf{caseL-cons}$$

By IH 2 on the first premise, we get $(m, \gamma e) \in [\![\text{list}[\sigma n]\ \sigma A]\!]_\varepsilon^{\sigma k, \sigma t}$.
Unrolling its definition with $(\star)$ and $c < m$, we get

a) $\sigma k \leqslant r \leqslant \sigma t$

b) $(m - c, \text{cons}(v_1, v_2)) \in [\![\text{list}[\sigma n]\ \sigma A]\!]_v$

By b), $\sigma n = I + 1$ for some I and we have

$$(m - c, v_1) \in [\![\sigma A]\!]_v \tag{1}$$

$$(m - c, v_2) \in [\![\text{list}[I]\ \sigma A]\!]_v \tag{2}$$

Then, we can instantiate IH 2 on the third premise using

- $FV(e_2) \subseteq \text{dom}(h : A, tl : \text{list}[i]\ A, \Omega')$ and $h : A, tl : \text{list}[i]\ A, \Omega' \subseteq h : A, tl : \text{list}[i]\ A, \Omega$
- $\sigma[i \mapsto I] \in \mathcal{D}[\![i :: \mathbb{N}, \Delta]\!]$
- $\models \sigma[i \mapsto I](\Phi \wedge n \doteq i + 1)$ obtained by combining $\models \sigma\Phi$ with $\models \sigma n \doteq I + 1$,
- $(m - c, \gamma[h \mapsto v_1, tl \mapsto v_2]) \in \mathcal{G}[\![\sigma[i \mapsto I](\Omega', x : A, tl : \text{list}[i]\ A)]\!]$ using (1) and (2) and $(m - c, \gamma) \in \mathcal{G}[\![\sigma\Omega']\!]$ (obtained by downward closure (Lemma 20)).

we get $(m, \gamma e_2[v_1/h, v_2/tl]) \in [\![\sigma[i \mapsto I]A]\!]_\varepsilon^{\sigma[i\mapsto I]k', \sigma[i\mapsto I]t'}$.
Since, $i \notin FV(k', t', A, A')$, we have
$(m, \gamma e_2[v_1/h, v_2/tl]) \in [\![\sigma A']\!]_\varepsilon^{\sigma k', \sigma t'}$.
Unrolling its definition using ($\diamond\diamond$) and $c_r < m - c$, we get

c) $\sigma k' \leqslant r_r \leqslant \sigma t'$
d) $(m - c - c_r, v_r) \in [\![\sigma A']\!]_v$

We conclude with

1. By a) and c), we get $\sigma k + \sigma k' + c_{caseL} \leqslant r + r_r + c_{caseL} \leqslant \sigma t + \sigma t' + c_{caseL}$
2. By downward closure (Lemma 20) on d) using

$$m - (c + c_r + 1) \leqslant m - (c + c_r)$$

we get $(m - (c + c_r + 1), v_r) \in (\!|\sigma A'|\!)_v$.

Case: $\dfrac{\Delta; \Phi_a; \Omega \vdash^t_k e : A\{I/i\} \qquad \Delta \vdash I :: S}{\Delta; \Phi_a; \Omega \vdash^t_k \textbf{pack}\ e : \exists i{::}S.\ A}$ **pack**

Assume that $\models \sigma\Phi$ and there exists Ω' s.t. $FV(\text{pack}\ e) \subseteq \text{dom}(\Omega')$

and $\Omega' \subseteq \Omega$ and $(m, \delta) \in \mathcal{G}[\![\sigma\Omega']\!]$
TS: $(m, \text{pack } \gamma e) \in [\![\exists i::S.\, A]\!]_\varepsilon^{\sigma k, \sigma t}$.
Following the definition of $[\![\cdot]\!]_\varepsilon^{\cdot,\cdot}$, assume that

$$\frac{\gamma e \Downarrow^{c,r} v \quad (\star)}{\text{pack } \gamma e \Downarrow^{c,r} \text{pack } v}\ \textbf{pack}$$

and $c < m$.
By IH 2 on the first premise using

$$FV(e) \subseteq \text{dom}(\Omega') \text{ and } \Omega' \subseteq \Omega \text{ and } (m, \delta) \in \mathcal{G}[\![\sigma\Omega']\!]$$

we get $(m, \gamma e) \in [\![\sigma A\{\sigma I/i\}]\!]_\varepsilon^{\sigma k, \sigma t}$.
Unrolling its definition with $(\star)$ and $c < m$, we get

a) $\sigma k \leqslant r \leqslant \sigma t$
b) $(m - c, v) \in [\![\sigma A\{\sigma I/i\}]\!]_v$

Then we can conclude as follows:

1. By a), $\sigma k \leqslant r \leqslant \sigma t$
2. TS: $(m - c, \text{pack } v) \in [\![\exists i::S.\, A]\!]_v$.
   By Lemma 22 on the second premise we know that $\vdash \sigma I :: S$.
   STS: $(m - c, v) \in [\![\sigma A\{\sigma I/i\}]\!]_v$.
   This follows by b).

Case:
$$\frac{\Delta; \Phi_a; \Omega \vdash_{k_1}^{t_1} e_1 : A_1 \qquad \Delta; \Phi_a; x : A_1, \Omega \vdash_{k_2}^{t_2} e_2 : A_2}{\Delta; \Phi_a; \Omega \vdash_{k_1+k_2+c_{let}}^{t_1+t_2+c_{let}} \textbf{let } x = e_1 \textbf{ in } e_2 : A_2}\ \textbf{let}$$

Assume that $\models \sigma\Phi$ and there exists Ω' s.t. $FV(\text{let } x = e_1 \text{ in } e_2) \subseteq \text{dom}(\Omega')$ and $\Omega' \subseteq \Omega$ and $(m, \delta) \in \mathcal{G}[\![\sigma\Omega']\!]$
TS: $(m, \text{let } x = \gamma e_1 \text{ in } \gamma e_2) \in [\![\sigma A_2]\!]_\varepsilon^{\sigma k_1 + \sigma k_2 + c_{let}, \sigma t_1 + \sigma t_2 + c_{let}}$.
Following the definition of $[\![\cdot]\!]_\varepsilon^{\cdot,\cdot}$, assume that

$$\frac{\gamma e_1 \Downarrow^{c_1,r_1} v_1 \quad (\star) \qquad \gamma e_2[v_1/x] \Downarrow^{c_r,r_r} v_r \quad (\diamond)}{\text{let } x = \gamma e_1 \text{ in } \gamma e_2 \Downarrow^{c_1+c_r+1, r_1+r_r+c_{let}} v_r}\ \textbf{let}$$

and $c_1 + c_r + 1 < m$.
By IH 2 on the first premise using

$$FV(e_1) \subseteq \text{dom}(\Omega') \text{ and } \Omega' \subseteq \Omega \text{ and } (m, \delta) \in \mathcal{G}[\![\sigma\Omega']\!]$$

we get $(m, \gamma e_1) \in [\![\sigma A_1]\!]_\varepsilon^{\sigma k_1, \sigma t_1}$. Unrolling its definition with $(\star)$ and $c_1 < m$, we get

a) $\sigma k_1 \leqslant r_1 \leqslant \sigma t_1$
b) $(m - c_1, v_1) \in [\![\sigma A_1]\!]_v$

By IH 2 on the second premise using $(m - c_1, \gamma[x \mapsto v]) \in \mathcal{G}[\![\sigma\Omega', x : \sigma A_1]\!]$ obtained by

- $FV(e_2) \subseteq dom(x : A_1, \Omega')$ and $x : A_1, \Omega' \subseteq x : A_1, \Omega$
- $(m - c_1, \gamma) \in \mathcal{G}[\![\sigma\Omega']\!]$ by downward closure (Lemma 20) on $(m, \gamma) \in \mathcal{G}[\![\sigma\Omega']\!]$ using $m - c_1 \leqslant m$
- $(m - c_1, v) \in [\![\sigma A_1]\!]_v$ by downward closure (Lemma 20) on c)

we get

$$(m - c_1, \gamma e_1[v/x]) \in [\![\sigma A_2]\!]_\varepsilon^{\sigma k_2, \sigma t_2} \tag{1}$$

Unrolling (1)'s definition using $(\diamond)$ and $c_r < m - c_1$, we get

c) $\sigma k_2 \leqslant r_r \leqslant \sigma t_2$
d) $(m - (c_1 + c_r), v_r) \in [\![\sigma A]\!]_v$

Now, we can conclude as follows

1. By a) and c) $\sigma k_1 + \sigma k_2 + c_{let} \leqslant (r_1 + r_r + c_{let}) \leqslant \sigma t_1 + \sigma t_2 + c_{let}$
2. By downward closure (Lemma 20) on d) using

$$m - (c_1 + c_r + 1) \leqslant m - (c_1 + c_r)$$

we get $(m - (c_1 + c_r + 1), v_r) \in [\![\sigma A]\!]_v$.

Case: $\dfrac{\Upsilon(\zeta) = A_1 \xrightarrow{\mathbf{exec}(k,t)} A_2 \qquad \Delta; \Phi_a; \Omega \vdash_{k'}^{t'} e : A_1}{\Delta; \Phi_a; \Omega \vdash_{k+k'+c_{primapp}}^{t+t'+c_{primapp}} \zeta\ e : A_2}$ **primapp**

Assume that $\models \sigma\Phi$ and there exists Ω' s.t. $FV(\zeta\ e) \subseteq dom(\Omega')$ and $\Omega' \subseteq \Omega$ and $(m, \delta) \in \mathcal{G}[\![\sigma\Omega']\!]$
TS: $(m, \zeta\ \gamma e) \in [\![\sigma A_2]\!]_\varepsilon^{\sigma k + \sigma k' + c_{primapp}, \sigma t + \sigma t' + c_{primapp}}$.

Following the definition of $[\![\cdot]\!]^{\cdot,\cdot}_{\varepsilon}$, assume that

$$\frac{\gamma e \Downarrow^{c,r} v \quad (\star) \qquad \hat{\zeta}(v) = (c_r, r_r, v_r) \quad (\diamond)}{\zeta\, \gamma e \Downarrow^{c+c_r+1, r+r_r+c_{primapp}} v_r}\ \textbf{primapp}$$

and $c + c_r + 1 < m$.

By IH 2 on the second premise using

$$FV(e) \subseteq dom(\Omega') \text{ and } \Omega' \subseteq \Omega \text{ and } (m, \delta) \in \mathcal{G}[\![\sigma\Omega']\!]$$

we get $(m, \gamma e) \in [\![\sigma A_1]\!]^{\sigma k', \sigma t'}_{\varepsilon}$.

Unrolling its definition with $c < m$, we get

a) $\sigma k' \leqslant r \leqslant \sigma t'$

b) $(m - c, v) \in [\![\sigma A_1]\!]_v$

Next, by Assumption (45) using $\zeta : \sigma A_1 \xrightarrow{\text{exec}(\sigma k, \sigma t)} \sigma A_2$ (obtained by substitution on the first premise), $(\diamond)$ and (b), we get

c) $\sigma k \leqslant r_r \leqslant \sigma t$

d) $(m - c - c_r, v_r) \in [\![\sigma A_2]\!]_v$

Now, we can conclude as follows:

1. Using a) and d), we get $\sigma k + \sigma k' + c_{primapp} \leqslant (c + c_r + c_{primapp}) \leqslant \sigma t + \sigma t' + c_{primapp}$
2. By downward closure (Lemma 20) on d) using

$$m - (c + c_r + 1) \leqslant m - (c + c_r)$$

we get $(m - (c + c_r + 1), v_r) \in [\![\sigma A_2]\!]_v$.

Case:
$$\frac{\Delta; \Phi_a; \Omega \vdash^{t}_{k} e : A \qquad \Delta; \Phi_a \models A \sqsubseteq A' \qquad \Delta; \Phi_a \models k' \leqslant k \qquad \Delta; \Phi_a \models t \leqslant t'}{\Delta; \Phi_a; \Omega \vdash^{t'}_{k'} e : A'}\ \sqsubseteq\ \textbf{exec}$$

Assume that $\models \sigma\Phi$ and there exists Ω' s.t. $FV(e) \subseteq dom(\Omega')$ and $\Omega' \subseteq \Omega$ and $(m, \delta) \in \mathcal{G}[\![\sigma\Omega']\!]$

TS: $(m, \gamma e) \in [\![\sigma A']\!]^{\sigma k', \sigma t'}_{\varepsilon}$.

Following the definition of $[\![\cdot]\!]^{\cdot,\cdot}_{\varepsilon}$, assume that

a) $\gamma e \Downarrow^{c,r} v$

b) $c < m$.

By IH 2 on the first premise using

$$FV(e) \subseteq dom(\Omega') \text{ and } \Omega' \subseteq \Omega \text{ and } (m, \delta) \in \mathcal{G}[\![\sigma\Omega']\!]$$

we get $(m, \gamma e) \in [\![\sigma A]\!]_\varepsilon^{\sigma k', \sigma t'}$.
Unrolling its definition with a) and $c < m$, we get

c) $\sigma k \leqslant r \leqslant \sigma t$

d) $(m - c, v) \in [\![\sigma A]\!]_v$

We can conclude this subcase

1. By Assumption (25) on the third and forth premises, we get $\sigma k' \leqslant \sigma k$ and $\sigma t' \leqslant \sigma t'$. By c) we know $\sigma k \leqslant r \leqslant \sigma t$, therefore we get $\sigma k' \leqslant r \leqslant \sigma t'$
2. By Lemma 21 on the second premise using c), we get $(m - c, v) \in [\![\sigma A']\!]_v$

□

Proof of Statement (3). Remember the statement (3) of Theorem 30:
Assume that $\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau$ and $\sigma \in \mathcal{D}[\![\Delta]\!]$ and $\models \sigma\Phi$. Then for $i \in \{1, 2\}$, if there exists Γ_i' s.t. $FV(e_i) \subseteq dom(\Gamma_i')$ and $\Gamma_i' \subseteq \Gamma$ and $(m, \delta) \in \mathcal{G}[\![|\sigma\Gamma_i'|_i]\!]$, then $(m, \delta e_i) \in [\![|\sigma\tau|_i]\!]_\varepsilon^{0,\infty}$.

For the structural rules, we will only show the left side since the right side is similar. For asynchronous rules, we first show the left side and then the right side in the same case.

Case: $\dfrac{\Gamma(x) = \tau}{\Delta; \Phi_a; \Gamma \vdash x \ominus x \lesssim 0 : \tau}$ **r-var**

Assume that $\models \sigma\Phi$ and there exists Γ' s.t. $FV(x) \subseteq dom(\Gamma')$ and $\Gamma' \subseteq \Gamma$ and $(m, \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_1]\!]$.

TS: $(m, \delta(x)) \in [\![|\sigma\tau|_1]\!]^{0,\infty}_{\varepsilon}$.

By Lemma 18, STS: $(m, \delta(x)) \in [\![|\sigma\tau|_1]\!]_v$.

By $(m, \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_1]\!]$ and $x \in \mathrm{dom}(\Gamma')$, we can conclude that $(m, \delta(x)) \in [\![|\sigma\tau|_1]\!]_v$.

Case:
$$\frac{\Delta \vdash \tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2 \ \mathtt{wf} \qquad \Delta; \Phi_a; x : \tau_1, f : \tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2, \Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau_2}{\Delta; \Phi_a; \Gamma \vdash \mathbf{fix}\ f(x).e_1 \ominus \mathbf{fix}\ f(x).e_2 \lesssim 0 : \tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2}\ \textbf{r-fix}$$

Assume that $\models \sigma\Phi$ and there exists Γ' s.t. $\mathrm{FV}(\mathrm{fix}\ f(x).e) \subseteq \mathrm{dom}(\Gamma')$ and $\Gamma' \subseteq \Gamma$ and $(m, \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_1]\!]$.

TS: $(m, \mathrm{fix}\ f(x).\delta e_1) \in [\![|\sigma\tau_1|_1 \xrightarrow{\mathrm{exec}(0,\infty)} |\sigma\tau_2|_1]\!]^{0,\infty}_{\varepsilon}$.

By Lemma 18, STS: $(m, \mathrm{fix}\ f(x).\delta e_1) \in [\![|\sigma\tau_1|_1 \xrightarrow{\mathrm{exec}(0,\infty)} |\sigma\tau_2|_1]\!]_v$.

We prove the more general statement

$$\forall m' \leqslant m.\ (m', \mathrm{fix}\ f(x).\delta e_1) \in [\![|\sigma\tau_1|_1 \xrightarrow{\mathrm{exec}(0,\infty)} |\sigma\tau_2|_1]\!]_v$$

by subinduction on m'.

There are two cases:

subcase 1: $m' = 0$

Since there is no non-negative j such that $j < 0$, the goal is vacuously true.

subcase 2: $m' = m'' + 1 \leqslant m$

By sub-IH

$$(m'', \mathrm{fix}\ f(x).\delta e_1) \in [\![|\sigma\tau_1|_1 \xrightarrow{\mathrm{exec}(0,\infty)} |\sigma\tau_2|_1]\!]_v \tag{1}$$

STS: $(m'' + 1, \mathrm{fix}\ f(x).\delta e_1) \in [\![|\sigma\tau_1|_1 \xrightarrow{\mathrm{exec}(0,\infty)} |\sigma\tau_2|_1]\!]_v$.

Pick $j < m'' + 1$ and assume that $(j, v) \in [\![|\sigma\tau_1|_1]\!]_v$.

STS: $(j, \delta e_1[v/x, (\mathrm{fix}\ f(x).\delta e_1)/f]) \in [\![|\sigma\tau_2|_1]\!]^{0,\infty}_{\varepsilon}$.

This follows by IH 3 on the premise instantiated with

- $(j, \delta[x \mapsto v, f \mapsto (\mathrm{fix}\ f(x).\delta e_1)]) \in \mathcal{G}[\![x : |\sigma\tau_1|_1, f : |\sigma\tau_1|_1 \xrightarrow{\mathrm{exec}(0,\infty)} |\sigma\tau_2|_1, |\sigma\Gamma|_1]\!]$ which holds because

- $FV(e_1) \subseteq dom(x : \tau_1, f : \tau_1 \xrightarrow{diff(t)} \tau_2), \Gamma'$ and $x : \tau_1, f : \tau_1 \xrightarrow{diff(t)} \tau_2, \Gamma' \subseteq x : \tau_1, f : \tau_1 \xrightarrow{diff(t)} \tau_2, \Gamma$
- $(j, \delta) \in \mathcal{G}[\![|\sigma\Gamma|_1]\!]$ using downward closure (Lemma 20) on $(m, \delta) \in \mathcal{G}[\![|\sigma\Gamma|_1]\!]$ using $j < m'' + 1 \leqslant m$.
- $(j, \nu) \in [\![|\sigma\tau_1|_1]\!]_\nu$, from the assumption above
- $(j, \text{fix } f(x).\delta e_1) \in [\![|\sigma\tau_1|_1 \xrightarrow{exec(0,\infty)} |\sigma\tau_2|_1]\!]_\nu$, obtained by downward closure (Lemma 20) on (1) using $j \leqslant m''$

Case:
$$\frac{\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_1' \lesssim t_1 : \tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2 \qquad \Delta; \Phi_a; \Gamma \vdash e_2 \ominus e_2' \lesssim t_2 : \tau_1}{\Delta; \Phi_a; \Gamma \vdash e_1\ e_2 \ominus e_1'\ e_2' \lesssim t_1 + t_2 + t : \tau_2}\ \textbf{r-app}$$

Assume that $\models \sigma\Phi$ and there exists Γ' s.t. $FV(e_1\ e_2) \subseteq dom(\Gamma')$ and $\Gamma' \subseteq \Gamma$ and $(m, \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_1]\!]$.

TS: $(m, \delta e_1\ \delta e_2) \in [\![|\sigma\tau_2|_1]\!]^{0,\infty}_{\varepsilon}$.

Following the definition of $[\![\cdot]\!]^{\cdot,\cdot}_{\varepsilon}$, assume that

$$\frac{\delta e_1 \Downarrow^{c_1,r_1} \text{fix } f(x).e \quad (\star) \qquad \delta e_2 \Downarrow^{c_2,r_2} v_2 \quad (\diamond) \qquad e[v_2/x, (\text{fix } f(x).e)/f] \Downarrow^{c_r,r_r} v_r \quad (\dagger)}{\delta e_1\ \delta e_2 \Downarrow^{c_1+c_2+c_r+1, r_1+r_2+r_r+c_{app}} v_r}\ \textbf{app}$$

and $c_1 + c_2 + c_r + 1 < m$.

By IH 3 on the first premise using

$$FV(e_1) \subseteq dom(\Gamma') \text{ and } \Gamma' \subseteq \Gamma \text{ and } (m, \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_1]\!]$$

we get $(m, \delta e_1) \in [\![|\sigma\tau_1|_1 \xrightarrow{exec(0,\infty)} |\sigma\tau_2|_1]\!]^{0,\infty}_{\varepsilon}$.

Unrolling its definition with $(\star)$ and $c_1 < m$, we get

a) $0 \leqslant r_1 \leqslant \infty$

b) $(m - c_1, \text{fix } f(x).e) \in [\![|\sigma\tau_1|_1 \xrightarrow{exec(0,\infty)} |\sigma\tau_2|_1]\!]_\nu$

By IH 3 on the second premise using

$$FV(e_2) \subseteq dom(\Gamma') \text{ and } \Gamma' \subseteq \Gamma \text{ and } (m, \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_1]\!]$$

we get $(m, \delta e_2) \in [\![|\sigma\tau_1|_1]\!]_\varepsilon^{0,\infty}$. Unrolling its definition with ($\diamond$) and $c_2 < m$, we get

c) $0 \leqslant r_2 \leqslant \infty$
d) $(m - c_2, v_2) \in [\![|\sigma\tau_1|_1]\!]_v$

By downward closure (Lemma 20) on d) using $m - c_1 - c_2 - 1 \leqslant m - c_2$, we get

$$(m - (c_1 + c_2 + 1), v_2) \in [\![|\sigma\tau_1|_1]\!]_v \tag{1}$$

Next, we unroll b) with (1) and $m - (c_1 + c_2 + 1) < m - c_1$ to obtain

$$(m - (c_1 + c_2 + 1), e[v_2/x, (\text{fix } f(x).e)]) \in [\![|\sigma\tau_2|_1]\!]_\varepsilon^{0,\infty} \tag{2}$$

By unrolling (2)'s definition using ($\dagger$) and $c_r < m - (c_1 + c_2 + 1)$, we get

e) $0 \leqslant r_r \leqslant \infty$
f) $(m - (c_1 + c_2 + c_r + 1), v_r) \in [\![|\sigma\tau_2|_1]\!]_v$

Now, we can conclude as follows:

1. We can trivially show $0 \leqslant (r_1 + r_2 + r_r + c_{app}) \leqslant \infty$
2. By f)

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_1' \lesssim t_1 : \tau \qquad \Delta; \Phi_a; \Gamma \vdash e_2 \ominus e_2' \lesssim t_2 : \mathbf{list}[n]^\alpha \tau}{\Delta; \Phi_a; \Gamma \vdash \mathbf{cons}(e_1, e_2) \ominus \mathbf{cons}(e_1', e_2') \lesssim t_1 + t_2 : \mathbf{list}[n+1]^{\alpha+1} \tau}$ **r-cons1**

Assume that $\models \sigma\Phi$ and there exists Γ' s.t. $FV(\text{cons}(e_1, e_2)) \subseteq \text{dom}(\Gamma')$ and $\Gamma' \subseteq \Gamma$ and $(m, \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_1]\!]$.
TS: $(m, \text{cons}(\delta e_1, \delta e_2)) \in [\![|\text{list}[\sigma n + 1]^{\sigma\alpha+1}\ \sigma\tau|_1]\!]_\varepsilon^{0,\infty} \equiv [\![\text{list}[\sigma n + 1]\ |\sigma\tau|_1]\!]_\varepsilon^{0,\infty}$.
Following the definition of $[\![\cdot]\!]_\varepsilon^{\cdot,\cdot}$, assume that

$$\frac{\delta e_1 \Downarrow^{c_1,r_1} v_1 \quad (\star) \qquad \delta e_2 \Downarrow^{c_2,r_2} v_2 \quad (\diamond)}{\mathrm{cons}(\delta e_1, \delta e_2) \Downarrow^{c_1+c_2,r_1+r_2} \mathrm{cons}(v_1, v_2)}\ \textbf{cons} \text{ and } c_1 + c_2 < m.$$

By IH 3 on the first premise using

$$\mathrm{FV}(e_1) \subseteq \mathrm{dom}(\Gamma') \text{ and } \Gamma' \subseteq \Gamma \text{ and } (m, \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_1]\!]$$

we get $(m, \delta e_1) \in [\![|\sigma\tau|_1]\!]^{0,\infty}_{\varepsilon}$. Unrolling its definition with $(\star)$ and $c_1 < m$, we get

a) $0 \leqslant r_1 \leqslant \infty$
b) $(m - c_1, v_1) \in [\![|\sigma\tau|_1]\!]_v$

By IH 3 on the second premise using

$$\mathrm{FV}(e_2) \subseteq \mathrm{dom}(\Gamma') \text{ and } \Gamma' \subseteq \Gamma \text{ and } (m, \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_1]\!]$$

we get $(m, \delta e_2) \in [\![|\mathrm{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|_1]\!]^{0,\infty}_{\varepsilon}$.
Unrolling its definition with $(\diamond)$ and $c_2 < m$, we get

c) $0 \leqslant r_2 \leqslant \infty$
d) $(m - c_2, v_2) \in [\![\mathrm{list}[\sigma n]\ |\sigma\tau|_1]\!]_v$

Now, we can conclude as follows:

1. We can trivially show that $0 \leqslant (r_1 + r_2) \leqslant \infty$
2. By downward closure (Lemma 20) on b) and d), we get $(m - (c_1 + c_2), v_1) \in [\![|\sigma\tau|_1]\!]_v$ and $(m - (c_1 + c_2), v_2) \in [\![\mathrm{list}[\sigma n]\ |\sigma\tau|_1]\!]_v$, when combined, gives us $(m - (c_1 + c_2), \mathrm{cons}(v_1, v_2)) \in [\![\mathrm{list}[\sigma n + 1]\ |\sigma\tau|_1]\!]_v \equiv [\![|\mathrm{list}[\sigma n]^{\sigma\alpha+1}\ \sigma\tau|_1]\!]_v$

Case: $$\frac{\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_1' \lesssim t_1 : \Box\tau \qquad \Delta; \Phi_a; \Gamma \vdash e_2 \ominus e_2' \lesssim t_2 : \mathbf{list}[n]^{\alpha}\tau}{\Delta; \Phi_a; \Gamma \vdash \mathbf{cons}(e_1, e_2) \ominus \mathbf{cons}(e_1', e_2') \lesssim t_1 + t_2 : \mathbf{list}[n+1]^{\alpha}\tau}\ \textbf{r-cons2}$$

Assume that $\models \sigma\Phi$ and there exists Γ' s.t. $\mathrm{FV}(\mathrm{cons}(e_1, e_2)) \subseteq \mathrm{dom}(\Gamma')$ and $\Gamma' \subseteq \Gamma$ and $(m, \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_1]\!]$.
TS: $(m, \mathrm{cons}(\delta e_1, \delta e_2)) \in [\![|\mathrm{list}[\sigma n + 1]^{\sigma\alpha}\ \sigma\tau|_1]\!]^{0,\infty}_{\varepsilon} \equiv [\![\mathrm{list}[\sigma n + 1]\ |\sigma\tau|_1]\!]^{0,\infty}_{\varepsilon}$.
Following the definition of $[\![\cdot]\!]^{\cdot}_{\varepsilon}$, assume that

$$\frac{\delta e_1 \Downarrow^{c_1,r_1} v_1 \quad (\star) \qquad \delta e_2 \Downarrow^{c_2,r_2} v_2 \quad (\diamond)}{\mathrm{cons}(\delta e_1, \delta e_2) \Downarrow^{c_1+c_2,r_1+r_2} \mathrm{cons}(v_1, v_2)}\ \textbf{cons}$$ and $c_1 + c_2 < m$.
By IH 3 on the first premise using

$$\mathrm{FV}(e_1) \subseteq \mathrm{dom}(\Gamma') \text{ and } \Gamma' \subseteq \Gamma \text{ and } (m, \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_1]\!]$$

we get $(m, \delta e_1) \in [\![|\Box\, \sigma\tau|_1]\!]^{0,\infty}_{\varepsilon}$. Unrolling its definition with $(\star)$ and $c_1 < m$, we get

a) $0 \leqslant r_1 \leqslant \infty$
b) $(m - c_1, v_1) \in [\![|\Box\, \sigma\tau|_1]\!]_v \equiv [\![|\sigma\tau|_1]\!]_v$

By IH 3 on the second premise using

$$\mathrm{FV}(e_2) \subseteq \mathrm{dom}(\Gamma') \text{ and } \Gamma' \subseteq \Gamma \text{ and } (m, \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_1]\!]$$

we get $(m, \delta e_2) \in [\![|\mathrm{list}[\sigma n]^{\sigma\alpha}\, \sigma\tau|_1]\!]^{0,\infty}_{\varepsilon}$.
Unrolling its definition with $(\diamond)$ and $c_2 < m$, we get

c) $0 \leqslant r_2 \infty$
d) $(m - c_2, v_2) \in [\![\mathrm{list}[\sigma n]\, |\sigma\tau|_1]\!]_v$

Now, we can conclude as follows:

1. We can trivially show that $0 \leqslant (r_1 + r_2) \leqslant \infty$
2. By downward closure (Lemma 20) on b) and d), we get $(m - (c_1 + c_2), v_1) \in [\![|\sigma\tau|_1]\!]_v$ and $(m - (c_1 + c_2), v_2) \in [\![\mathrm{list}[\sigma n]\, |\sigma\tau|_1]\!]_v$, when combined, gives us $(m - (c_1 + c_2), \mathrm{cons}(v_1, v_2)) \in [\![\mathrm{list}[\sigma n + 1]\, |\sigma\tau|_1]\!]_v \equiv [\![|\mathrm{list}[\sigma n]^{\sigma\alpha}\, \sigma\tau|_1]\!]_v$

Case: $$\frac{\Delta; \Phi_a; |\Gamma|_1 \vdash^{t_1}_{k_1} e_1 : A_1 \qquad \Delta; \Phi_a; |\Gamma|_2 \vdash^{t_2}_{k_2} e_2 : A_2}{\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t_1 - k_2 : U\,(A_1, A_2)}\ \textbf{switch}$$
The are two parts to show.

subcase 1: Assume that $\models \sigma\Phi$ and there exists Γ' s.t. $\mathrm{FV}(e_1) \subseteq \mathrm{dom}(\Gamma')$ and $\Gamma' \subseteq \Gamma$ and $(m, \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_1]\!]$.

TS: $(m, \delta e_1) \in [\![|U\,(\sigma A_1, \sigma A_2)|_1]\!]^{0,\infty}_{\varepsilon} \equiv [\![\sigma A_1]\!]^{0,\infty}_{\varepsilon}$.
Assume that

a) $\delta e_1 \Downarrow^{c_r, r_r} v_r$
b) $c_r < m$.

By IH 2 on the first premise using

$$FV(e_1) \subseteq dom(|\Gamma'|_1) \text{ and } |\Gamma'|_1 \subseteq |\Gamma|_1 \text{ and } (m, \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_1]\!]$$

we get $(m, \delta e_1) \in [\![\sigma A_1]\!]^{\sigma k_1, \sigma t_1}_{\varepsilon}$
By unrolling its definition with a) and b), we get

c) $\sigma k_1 \leqslant r_r \leqslant \sigma t_1$
d) $(m - c_r, v_r) \in [\![\sigma A_1]\!]_v$

We can conclude as follows

1. Trivially, $0 \leqslant r_r \leqslant \infty$
2. By d)

subcase 2: Assume that $\models \sigma\Phi$ and there exists Γ' s.t. $FV(e_2) \subseteq dom(\Gamma')$ and $\Gamma' \subseteq \Gamma$ and $(m, \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_2]\!]$.
TS: $(m, \delta e_2) \in [\![|U\,(\sigma A_1, \sigma A_2)|_2]\!]^{0,\infty}_{\varepsilon} \equiv [\![\sigma A_2]\!]^{0,\infty}_{\varepsilon}$.
Assume that

a) $\delta e_2 \Downarrow^{c_r, r_r} v_r$
b) $c_r < m$.

By IH 2 on the second premise using

$$FV(e_2) \subseteq dom(|\Gamma'|_2) \text{ and } |\Gamma'|_2 \subseteq |\Gamma|_2 \text{ and } (m, \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_2]\!]$$

we get $(m, \delta e_2) \in [\![\sigma A_2]\!]^{\sigma k_2, \sigma t_2}_{\varepsilon}$
By unrolling its definition with a) and b), we get

c) $\sigma k_2 \leqslant r_r \leqslant \sigma t_2$

d) $(m - c_r, v_r) \in [\![\sigma A_2]\!]_v$

We can conclude as follows

1. Trivially, $0 \leqslant r_r \leqslant \infty$
2. By d)

Case: $\dfrac{\Delta; \Phi_a; |\Gamma|_2 \vdash^{t_1}_{k_1} e_1 : A_1 \qquad \Delta; \Phi_a; x : U\, A_1, \Gamma \vdash e \ominus e_2 \lesssim t_2 : \tau_2}{\Delta; \Phi_a; \Gamma \vdash e \ominus \textbf{let}\ x = e_1\ \textbf{in}\ e_2 \lesssim t_2 - k_1 - c_{let} : \tau_2}$ **r-e-let**

There are two parts to show.

subcase 1: Assume that $\models \sigma\Phi$ and there exists Γ' s.t. $FV(e) \subseteq dom(\Gamma')$ and $\Gamma' \subseteq \Gamma$ and $(m, \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_1]\!]$.
TS: $(m, \delta e) \in [\![|\sigma\tau_2|_1]\!]^{0,\infty}_{\varepsilon}$
Follows by IH 3 on the second premise using $(m, \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_1]\!]$ since we know that $FV(e) \subseteq dom(\Gamma')$ and $\Gamma' \subseteq \Gamma, x : A_1$ since x doesn't occur free in e, we get immediately $(m, \delta e) \in [\![|\sigma\tau_2|_1]\!]^{0,\infty}_{\varepsilon}$.

subcase 2:

Assume that $\models \sigma\Phi$ and there exists Γ' s.t. $FV(\text{let } x = e_1 \text{ in } e_2) \subseteq dom(\Gamma')$ and $\Gamma' \subseteq \Gamma$ and $(m, \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_2]\!]$.
TS: $(m, \text{let } x = \delta e_1 \text{ in } \delta e_2) \in [\![|\sigma\tau_2|_2]\!]^{0,\infty}_{\varepsilon}$
Assume that

a) $\dfrac{\delta e_1 \Downarrow^{c_1,r_1} v_1 \quad (\star) \qquad \delta e_2[v_1/x] \Downarrow^{c_r,r_r} v_r \quad (\diamond)}{\text{let } x = \delta e_1 \text{ in } \delta e_2 \Downarrow^{c_1+c_r+1,r_1+r_r+c_{let}} v_r}$ **let**

b) $c_1 + c_r + 1 < m$

By IH 2 on the first premise using

$$FV(e_1) \subseteq dom(|\Gamma'|_2) \text{ and } |\Gamma'|_2 \subseteq |\Gamma|_2 \text{ and } (m, \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_2]\!]$$

we get $(m, \delta e_1) \in [\![\sigma A_1]\!]^{\sigma k_1, \sigma t_1}_{\varepsilon}$.
Unrolling second part of its definition using $(\star)$ and $c_1 < m$, we get

c) $\sigma k_1 \leqslant r_1 \leqslant \sigma t_1$

d) $(m - c_1, v_1) \in [\![\sigma A_1]\!]_v$

By IH 3 on the second premise using $(m - c, \delta[x \mapsto v_1]) \in \mathcal{G}[\![x : \sigma A_1, |\sigma\Gamma'|_2]\!]$ which hold since

- $FV(e_2) \subseteq dom(x : U\, A_1, \Gamma')$ and $x : U\, A_1, \Gamma' \subseteq x : U\, A_1, \Gamma$
- $(m - c, \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_2]\!]$ by downward closure (Lemma 20) on $(m, \delta) \in \mathcal{G}[\![|\sigma\Gamma|_2]\!]$ using $m - c \leqslant m$
- $(m - c, v_1) \in [\![\sigma A_1]\!]_v$

we get $(m - c, \delta e_2[v_1/x]) \in [\![|\sigma\tau_2|_2]\!]^{0,\infty}_{\varepsilon}$.
Unfolding its definition using $(\diamond)$ and $c_r < m - c_1$, we get

e) $0 \leqslant r_r \leqslant \infty$
f) $(m - (c_1 + c_r), v_r) \in [\![|\sigma\tau_2|_2]\!]_v$

Then we can conclude as follows

1. Trivially, $0 \leqslant r_1 + r_r + c_{let} \leqslant \infty$
2. By downward closure (Lemma 20) on f) using

$$m - (c + c_r + 1) \leqslant m - (c + c_r)$$

we get $(m - (c + c_r + 1), v_r) \in (\!|\sigma\tau_2|_2|\!)_v$.

Case: $$\frac{\Delta; \Phi_a; |\Gamma|_2 \vdash_{\overline{k'}} e' : A_1 + A_2 \qquad \Delta; \Phi_a; x : U\, A_1, \Gamma \vdash e \ominus e_1' \lesssim t : \tau \qquad \Delta; \Phi_a; y : U\, A_2, \Gamma \vdash e \ominus e_2' \lesssim t : \tau}{\Delta; \Phi_a; \Gamma \vdash e \ominus \textbf{ case } (e', x.e_1', y.e_2') \lesssim t - k' - c_{case} : \tau}\ \textbf{r-e-case}$$

There are two parts to show.

subcase 1: Assume that $\models \sigma\Phi$ and there exists Γ' s.t. $FV(e) \subseteq dom(\Gamma')$ and $\Gamma' \subseteq \Gamma$ and $(m, \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_1]\!]$.

TS: $(m, \delta e) \in [\![|\sigma\tau|_1]\!]^{0,\infty}_{\varepsilon}$
We conclude by IH 3 on the second premise using $(m, \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_1]\!]$ since we know that $FV(e) \subseteq dom(\Gamma')$ and $\Gamma' \subseteq \Gamma, x : A_1$ since x

doesn't occur free in e, we get immediately $(m, \delta e) \in [\![|\sigma\tau|_1]\!]^{0,\infty}_{\varepsilon}$.

subcase 2:

Assume that $\models \sigma\Phi$ and there exists Γ' s.t. FV(case $(e', e'_1, e'_2)) \subseteq \mathrm{dom}(\Gamma')$ and $\Gamma' \subseteq \Gamma$ and $(m, \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_2]\!]$.
TS: $(m,$ case $(\delta e', \delta e'_1, \delta e'_2)) \in [\![|\sigma\tau|_2]\!]^{0,\infty}_{\varepsilon}$
There are also two parts to show here depending on what δe evaluates to. We only show one for brevity, the other one is similar.
Assume that

a) $\dfrac{\delta e' \Downarrow^{c',r'} \mathrm{inl}\, v' \quad (\star) \qquad \delta e'_1[v'/x] \Downarrow^{c'_r,r'_r} v'_r \quad (\diamond)}{\mathrm{case}\ (\delta e, x.\delta e_1, y.\delta e_2) \Downarrow^{c'+c'_r+1,r'+r'_r+c_{case}} v'_r}$ **case-inl**

b) $c' + c'_r + 1 < m$

By IH 2 on the first premise using
$\mathrm{FV}(e') \subseteq \mathrm{dom}(|\Gamma'|_2)$ and $|\Gamma'|_2 \subseteq |\Gamma|_2$ and $(m, \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_2]\!]$
we get $(m, \delta e') \in [\![\sigma A_1 + \sigma A_2]\!]^{\sigma k', -'}_{\varepsilon}$.
Unrolling second part of its definition using $(\star)$ and $c < m$, we get

c) $\sigma k' \leqslant r' \leqslant \sigma t'$

d) $(m - c', \mathrm{inl}\, v') \in [\![\sigma A_1 + \sigma A_2]\!]_v$

By IH 3 on the second premise using $(m - c', \delta[x \mapsto v']) \in \mathcal{G}[\![x : \sigma A_1, |\sigma\Gamma'|_2]\!]$ which hold since

- $\mathrm{FV}(e'_1) \subseteq \mathrm{dom}(x : \mathrm{U}\, A_1, \Gamma')$ and $x : \mathrm{U}\, A_1, \Gamma' \subseteq x : \mathrm{U}\, A_1, \Gamma$
- $(m - c', \delta) \in \mathcal{G}[\![|\sigma\Gamma'|_2]\!]$ by downward closure (Lemma 20) on $(m, \delta) \in \mathcal{G}[\![|\sigma\Gamma|_2]\!]$ using $m - c \leqslant m$
- $(m - c', v') \in [\![\sigma A_1]\!]_v$

we get $(m - c', \delta e'_2[v'/x]) \in [\![|\sigma\tau|_2]\!]^{0,\infty}_{\varepsilon}$.
Unfolding its definition using $(\diamond)$ and $c'_r < m - c'$, we get

e) $0 \leqslant r'_r \leqslant \infty$

f) $(m - (c' + c'_r), \nu'_r) \in [\![|\sigma\tau|_2]\!]_\nu$

Then we can conclude as follows

1. Trivially, $0 \leqslant r' + r'_r + c_{let} \leqslant \infty$
2. By downward closure (Lemma 20) on f) using

$$m - (c' + c'_r + 1) \leqslant m - (c' + c'_r)$$

we get $(m - (c' + c'_r + 1), \nu_r) \in (\!|\sigma\tau|_2|\!)_\nu$.

□

□

B

APPENDIX FOR DUCOSTIT

In this chapter, we first describe the necessary definitions, lemmas and theorems for proving the soundness of the DuCostIt's unary and binary (relational) typing with respect to the abstract change propagation and from-scratch cost semantics.

We use some abbreviations throughout. STS stands for "suffices to show", TS stands for "to show", and RTS stands for "remains to show".

B.1 DUCOSTIT LEMMAS

Lemma 31 (Value interpretation containment)**.** *The following hold.*

1. $(m, w) \in (\!|\tau|\!)_v$ *then* $(m, w) \in (\!|\tau|\!)^0_\varepsilon$.
2. $(m, v) \in [\![A]\!]_v$ *then* $(m, v) \in [\![A]\!]^0_\varepsilon$.

Proof of (1). Assume that $(m, w) \in (\!|\tau|\!)_v$ $(\star)$.
Following the definition of $(\!|\tau|\!)^0_\varepsilon$, assume that

a) $L(w) \Downarrow^f \langle v, D\rangle$
b) $R(w') \Downarrow^{f'} \langle v', D'\rangle$
c) $f < m$.

We have to show that there exist w' and c' such that:

1. $\langle\langle v, D\rangle, w\rangle \curvearrowright w', \langle v', D'\rangle, c'$
2. $v = L(w') \ \wedge \ v' = R(w')$
3. $c' = 0$
4. $(m - f, w') \in (\!|\tau|\!)_v$

Since w is a bi-value, $L(w)$ and $R(w)$ are values and, hence, by **value** evaluation rule combined with (a) & (b), we have

$\boxed{\Delta \vdash \tau\ \mathtt{wf}}$ Relational type τ is well-formed.
$\boxed{\Delta \vdash^{A} A\ \mathtt{wf}}$ Type A is well-formed.

$$\frac{}{\Delta \vdash \mathrm{unit}_r\ \mathtt{wf}}\ \textbf{wf-unit} \qquad \frac{}{\Delta \vdash \mathrm{int}_r\ \mathtt{wf}}\ \textbf{wf-int}$$

$$\frac{\Delta \vdash \tau_1\ \mathtt{wf} \qquad \Delta \vdash \tau_2\ \mathtt{wf}}{\Delta \vdash \tau_1 \times \tau_2\ \mathtt{wf}}\ \textbf{wf-prod} \qquad \frac{\Delta \vdash \tau_1\ \mathtt{wf} \qquad \Delta \vdash \tau_2\ \mathtt{wf}}{\Delta \vdash \tau_1 + \tau_2\ \mathtt{wf}}\ \textbf{wf-sum}$$

$$\frac{\Delta \vdash \tau_1\ \mathtt{wf} \qquad \Delta \vdash \tau_2\ \mathtt{wf} \qquad \Delta \vdash t :: \mathbb{R}}{\Delta \vdash \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2\ \mathtt{wf}}\ \textbf{wf-fun}$$

$$\frac{\Delta \vdash n :: \mathbb{N} \qquad \Delta \vdash \alpha :: \mathbb{N} \qquad \Delta \vdash \tau\ \mathtt{wf}}{\Delta \vdash \mathrm{list}[n]^{\alpha}\ \tau\ \mathtt{wf}}\ \textbf{wf-list}$$

$$\frac{i :: S, \Delta \vdash \tau\ \mathtt{wf} \qquad i :: S, \Delta \vdash t :: \mathbb{R}}{\Delta \vdash \forall i \overset{\mathbb{CP}(t)}{::} S.\, \tau\ \mathtt{wf}}\ \textbf{wf-}\forall \qquad \frac{i :: S, \Delta \vdash \tau\ \mathtt{wf}}{\Delta \vdash \exists i{::}S.\, \tau\ \mathtt{wf}}\ \textbf{wf-}\exists$$

$$\frac{\Delta \vdash^{A} A\ \mathtt{wf}}{\Delta \vdash \mathcal{U} A\ \mathtt{wf}}\ \textbf{wf-U} \qquad \frac{\Delta \vdash \tau\ \mathtt{wf}}{\Delta \vdash \Box \tau\ \mathtt{wf}}\ \textbf{wf-box} \qquad \frac{\Delta \vdash C\ \mathtt{wf} \qquad \Delta \vdash \tau\ \mathtt{wf}}{\Delta \vdash C \supset \tau\ \mathtt{wf}}\ \textbf{wf-C}\supset$$

$$\frac{\Delta \vdash C\ \mathtt{wf} \qquad \Delta \vdash \tau\ \mathtt{wf}}{\Delta \vdash C\ \&\ \tau\ \mathtt{wf}}\ \textbf{wf-C\&}$$

Figure 63: Well-formedness of relational types

$\boxed{\Delta \vdash^{A} A\ \mathtt{wf}}$ Type A is well-formed.

$$\frac{}{\Delta \vdash^{A} \mathrm{unit}\ \mathtt{wf}}\ \textbf{wf-u-unit} \qquad \frac{}{\Delta \vdash^{A} \mathrm{int}\ \mathtt{wf}}\ \textbf{wf-u-int}$$

$$\frac{\Delta \vdash^{A} A_1\ \mathtt{wf} \qquad \Delta \vdash^{A} A_2\ \mathtt{wf}}{\Delta \vdash^{A} A_1 \times A_2\ \mathtt{wf}}\ \textbf{wf-u-prod}$$

$$\frac{\Delta \vdash^{A} A_1\ \mathtt{wf} \qquad \Delta \vdash^{A} A_2\ \mathtt{wf}}{\Delta \vdash^{A} A_1 + A_2\ \mathtt{wf}}\ \textbf{wf-u-sum}$$

$$\frac{\Delta \vdash^{A} A_1\ \mathtt{wf} \qquad \Delta \vdash^{A} A_2\ \mathtt{wf} \qquad \Delta \vdash k :: \mathbb{R} \qquad \Delta \vdash t :: \mathbb{R}}{\Delta \vdash^{A} A_1 \xrightarrow{\mathbb{FS}(t)} A_2\ \mathtt{wf}}\ \textbf{wf-u-fun}$$

$$\frac{\Delta \vdash n :: \mathbb{N} \qquad \Delta \vdash^{A} A\ \mathtt{wf}}{\Delta \vdash^{A} \mathrm{list}[n]\ A\ \mathtt{wf}}\ \textbf{wf-u-list}$$

$$\frac{i :: S, \Delta \vdash^{A} A\ \mathtt{wf} \qquad i :: S, \Delta \vdash k :: \mathbb{R} \qquad i :: S, \Delta \vdash t :: \mathbb{R}}{\Delta \vdash^{A} \forall i \overset{\mathbb{FS}(t)}{::} S.\, A\ \mathtt{wf}}\ \textbf{wf-u-}\forall$$

$$\frac{i :: S, \Delta \vdash^{A} A\ \mathtt{wf}}{\Delta \vdash^{A} \exists i{::}S.\, A\ \mathtt{wf}}\ \textbf{wf-u-}\exists \qquad \frac{\Delta \vdash C\ \mathtt{wf} \qquad \Delta \vdash^{A} A\ \mathtt{wf}}{\Delta \vdash^{A} C \supset A\ \mathtt{wf}}\ \textbf{wf-u-C}\supset$$

$$\frac{\Delta \vdash C\ \mathtt{wf} \qquad \Delta \vdash^{A} A\ \mathtt{wf}}{\Delta \vdash^{A} C\ \&\ A\ \mathtt{wf}}\ \textbf{wf-u-C\&}$$

Figure 64: Well-formedness of types

d) $L(w) \Downarrow^0 \langle L(w), L(w)\rangle$

e) $R(w) \Downarrow^0 \langle R(w), R(w)\rangle$

Then, we can conclude as follows:

1. From lemma 34, $\langle\langle L(w), L(w)\rangle, w\rangle \curvearrowright w, \langle R(w), R(w)\rangle, 0$.
2. By (d) and (e), we have $v = L(w)$ and $v' = R(w)$
3. As obtained in the first statement, $c' = 0$
4. Since we have $w' = w$ as obtained in the first statement, we conclude by the main assumption $(\star)$.

□

Proof of (2). Assume that $(m, v) \in [\![A]\!]_v$ $(\star)$.
Following the definition of $[\![A]\!]^0_\varepsilon$, assume that $v \Downarrow^f \langle v, v\rangle$ and $f < m$. Then, we can immediately show

1. By **value** evaluation rule, f = o. Hence, $f = 0 \leqslant 0$
2. $(m - 0, v) \in [\![A]\!]_v$ which follows from the assumption $(\star)$.

□

Lemma 32 (Bi-value projection)**.** *The following hold.*

1. *If* $(m, w) \in (\!|\tau|\!)_v$ *then* $\forall j.\ (j, L(w)) \in [\![|\tau|]\!]_v$ *and* $(j, R(w)) \in [\![|\tau|]\!]_v$.
2. *If* $(m, w) \in \mathfrak{C}\ A\ \mathfrak{D}_v$ *then* $\forall j.\ (j, L(w)) \in [\![A]\!]_v$ *and* $(j, R(w)) \in [\![A]\!]_v$.

Proof. (1) and (2) are proven simultenously by induction on the type. □

Proof of statement (1). Proof is by induction on the type.

Case: $(m, \mathsf{keep}(n)) \in (\!|\mathrm{int}_r|\!)_A$
Since $L(\mathsf{keep}(n)) = R(\mathsf{keep}(n)) = n$, by definition we have $\forall j.(j, n) \in [\![\mathrm{int}]\!]_A$.

Case: $(m, (w_1, w_2)) \in (\!|\tau_1 \times \tau_2|\!)_v$ $(\star)$
TS: $\forall j.\ (j, L((w_1, w_2))) \in [\![|\tau_1| \times |\tau_2|]\!]_v \wedge (j, R((w_1, w_2))) \in [\![|\tau_1| \times |\tau_2|]\!]_v$.
Pick j.

STS 1: $(j, L(w_1)) \in [\![|\tau_1|]\!]_v \wedge (j, R(w_1)) \in [\![|\tau_1|]\!]_v$ ($\diamond$).

STS 2: $(j, L(w_2)) \in [\![|\tau_2|]\!]_v \wedge (j, R(w_2)) \in [\![|\tau_2|]\!]_v$ ($\diamond\diamond$).

By unrolling the ($\star$), we get $(m, w_1) \in (\!|\tau_1|\!)_v$ ($\dagger$) and $(m, w_2) \in (\!|\tau_2|\!)_v$ ($\dagger\dagger$).

By IH 1 on ($\dagger$), we get ($\diamond$), and by IH 1 on ($\dagger\dagger$) we get ($\diamond\diamond$).

Case: $(m, w) \in (\!|\tau_1 + \tau_2|\!)_A$

TS:$\forall j.\ (j, L(w)) \in [\![|\tau_1 + \tau_2|]\!]_A \wedge (j, R(w)) \in [\![|\tau_1 + \tau_2|]\!]_A$.

There are two cases. We only show the left projection:

We have $(m, \texttt{inl}\ w) \in (\!|\tau_1 + \tau_2|\!)_v$, that is $(m, w) \in (\!|\tau_1|\!)_v$ ($\dagger$).

TS: $\forall j.\ (j, L(\texttt{inl}\ w)) \in [\![|\tau_1 + \tau_2|]\!]_v \wedge (j, R(\texttt{inl}\ w)) \in [\![|\tau_1 + \tau_2|]\!]_v$ ($\star$).

Pick j.

STS: $(j, L(w)) \in [\![|\tau_1|]\!]_v \wedge (j, R(w)) \in [\![|\tau_1|]\!]_v$

By IH 1 on ($\dagger$), we get $\forall j.\ (j, L(w)) \in [\![|\tau_1|]\!]_v \wedge (j, R(w)) \in [\![|\tau_1|]\!]_v$.

By instantiating with j, we conclude.

Case: $(m, \texttt{nil}) \in (\!|\text{list}[0]^\alpha\, \tau|\!)_v$

TS: $\forall j.\ (j, R(\texttt{nil})) \in [\![|\text{list}[0]^\alpha\, \tau|]\!]_v = [\![\text{list}[0]\, |\tau|]\!]_v$

This follows immediately by definition.

Case: $(m, \texttt{cons}(w_1, w_2)) \in (\!|\text{list}[I+1]^\alpha\, \tau|\!)_v$ ($\star$)

TS: $\forall j.(j, L(\texttt{cons}(w_1, w_2))) \in [\![|\text{list}[I+1]^\alpha\, \tau|]\!]_v \wedge (j, R(\texttt{cons}(w_1, w_2))) \in [\![|\text{list}[I+1]^\alpha\, \tau|]\!]_v = [\![\text{list}[I+1]\, |\tau|]\!]_v$

Pick j.

There are two cases for unrolling the definition of ($\star$).

subcase 1: We have $(m, w_1) \in (\!|\tau|\!)_v$ ($\dagger$) and $(m, w_2) \in (\!|\text{list}[I]^{\alpha-1}\, \tau|\!)_v$ ($\dagger\dagger$)

By IH 1 on ($\dagger$), we get $\forall j.\ (j, L(w_1)) \in [\![|\tau|]\!]_v \wedge (j, R(w_1)) \in [\![|\tau|]\!]_v$ ($\diamond$)

By IH 1 on ($\dagger\dagger$), we get $\forall j.\ (j, L(w_2)) \in [\![|\text{list}[I]^{\alpha-1}\, \tau|]\!]_v \wedge (j, R(w_2)) \in [\![|\text{list}[I]^{\alpha-1}\, \tau|]\!]_v = [\![\text{list}[I]\, |\tau|]\!]_v$ ($\diamond\diamond$).

By instantiating ($\diamond$) and ($\diamond\diamond$) with the j we picked above, we get $(j, L(\texttt{cons}(w_1, w_2))) \in [\![\text{list}[I+1]\, |\tau|]\!]_v \wedge (j, R(\texttt{cons}(w_1, w_2))) \in [\![\text{list}[I+1]\, |\tau|]\!]_v$.

subcase 2: We have $(m, w_1) \in (\!|\Box\, \tau|\!)_v$ ($\dagger$) and $(m, w_2) \in (\!|\text{list}[I]^\alpha\, \tau|\!)_v$ ($\dagger\dagger$)

By IH 1 on ($\dagger$), we get $\forall j.\ (j, L(w_1)) \in [\![|\tau|]\!]_v \wedge (j, R(w_1)) \in$

$[\![|\tau|]\!]_\nu$ ($\diamond$)

By IH 1 on ($\dagger\dagger$), we get $\forall j.\ (j, L(w_2)) \in [\![|\text{list}[I]^\alpha\ \tau|]\!]_\nu \wedge (j, R(w_2)) \in [\![|\text{list}[I]^\alpha\ \tau|]\!]_\nu = [\![\text{list}[I]\ |\tau|]\!]_\nu$ ($\diamond\diamond$).

By instantiating ($\diamond$) and ($\diamond\diamond$) with the j we picked above, we get $(j, L(\texttt{cons}(w_1, w_2))) \in [\![\text{list}[I+1]\ |\tau|]\!]_\nu \wedge (j, R(\texttt{cons}(w_1, w_2))) \in [\![\text{list}[I+1]\ |\tau|]\!]_\nu$.

Case: $(m, \texttt{fix}\ f(x).e) \in (\!|\tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2|\!)_\nu$ ($\star$)

TS: $\forall j.\ (j, L(\texttt{fix}\ f(x).e)) \in [\![|\tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2|]\!]_\nu \wedge (j, R(\texttt{fix}\ f(x).e)) \in [\![|\tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2|]\!]_\nu = [\![|\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\tau_2|]\!]_\nu$.

By unrolling the definition of ($\star$), we have

$$(m, \texttt{fix}\ f(x).e) \in \mathcal{C}\ |\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\tau_2|\ \mathcal{D}_\nu \tag{1}$$

By IH 2 on eq. (1), we get $\forall j.\ (j, L(\texttt{fix}\ f(x).e)) \in [\![|\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\tau_2|]\!]_\nu \wedge (j, R(\texttt{fix}\ f(x).e)) \in [\![|\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\tau_2|]\!]_\nu$.

Case: $(m, w) \in (\!|U\ A|\!)_\nu(\star)$

TS: $\forall j.\ (j, L(w)) \in [\![|U\ A|]\!]_\nu \wedge$
$(j, R(w)) \in [\![|U\ A|]\!]_\nu = [\![A]\!]_\nu$.

There are two cases for ($\star$).

subcase 1: $w = \texttt{new}(\nu, \nu')$

We can conclude by definition of ($\star$).

subcase 2: $w \neq \texttt{new}(\nu, \nu')$, hence by ($\star$), we have $(m, w) \in \mathcal{C}\ A\ \mathcal{D}_\nu$.

We can conclude by IH 2.

Case: $(m, w) \in (\!|\Box\tau|\!)_\nu$

By unrolling the definition we know that $(m, w) \in (\!|\tau|\!)_\nu(\spadesuit)$.

TS: $\forall j.\ (j, L(w)) \in [\![|\Box\tau|]\!]_\nu \wedge (j, R(w)) \in [\![|\Box\tau|]\!]_\nu = [\![|\tau|]\!]_\nu$.

Follows immediately by IH 2 on ($\spadesuit$).

□

Proof of statement (2). Proof is by induction on the unary type.

Case: $(m, w) \in (\!|A_1 + A_2|\!)_A$

There are two cases. We only show the left projection:

We have $(m, \texttt{inl}\ w) \in (\!|A_1 + A_2|\!)_v$, that is $(m, w) \in (\!|\mathsf{U}\, A_1|\!)_v$ (†).

TS: $\forall j.\ (j, L(\texttt{inl}\ w)) \in [\![A_1 + A_2]\!]_v \wedge (j, R(\texttt{inl}\ w)) \in [\![A_1 + A_2]\!]_v$ ($\star$).

Pick j.

STS: $(j, R(w)) \in [\![A_1]\!]_v$

By IH 1 on (†), we get $\forall j.\ (j, L(w)) \in [\![A_1]\!]_v \wedge (j, R(w)) \in [\![A_1]\!]_v$.

By instantiating with j, we conclude.

Case: $(m, \texttt{fix}\ f(x).ee) \in (\![\, A_1 \xrightarrow{\mathbb{FS}(t)} A_2 \,]\!)_v$ ($\star$)

TS: $\forall j.\ (j, R(\texttt{fix}\ f(x).ee)) \in [\![A_1 \xrightarrow{\mathbb{FS}(t)} A_2]\!]_v$.

This immediately follows by the definition of ($\star$).

□

Lemma 33 (Downward closure)**.** *The following hold.*

1. *If* $(m, w) \in (\!|\tau|\!)_v$ *and* $m' \leqslant m$, *then* $(m', w) \in (\!|\tau|\!)_v$.
2. *If* $(m, v) \in [\![\tau]\!]_v$ *and* $m' \leqslant m$, *then* $(m', v) \in [\![\tau]\!]_v$.
3. *If* $(m, w) \in (\![\, A \,]\!)_v$ *and* $m' \leqslant m$, *then* $(m', w) \in (\![\, A \,]\!)_v$.
4. *If* $(m, ee) \in (\!|\tau|\!)^t_\varepsilon$ *and* $m' \leqslant m$, *then* $(m', ee) \in (\!|\tau|\!)^t_\varepsilon$.
5. *If* $(m, e) \in [\![\tau]\!]^t_\varepsilon$ *and* $m' \leqslant m$, *then* $(m', e) \in [\![\tau]\!]^t_\varepsilon$.
6. *If* $(m, \delta) \in \mathcal{G}(\!|\Gamma|\!)$ *and* $m' \leqslant m$, *then* $(m', \delta) \in \mathcal{G}(\!|\Gamma|\!)$.
7. *If* $(m, \gamma) \in \mathcal{G}[\![\Gamma]\!]$ *and* $m' \leqslant m$, *then* $(m', \gamma) \in \mathcal{G}[\![\Gamma]\!]$.

Proof. (1,3,4) and (2,5) are proved simultaneously by induction on τ. (6,7) follows from (1,2).

We just show the proofs of statement (4) and (5) below.

□

Proof of statement (4). Assume that $(m, ee) \in (\!|\sigma\tau|\!)^t_\varepsilon$ and $m' \leqslant m$.

TS: $(m', ee) \in (\!|\sigma\tau|\!)^t_\varepsilon$

Assume that

a) $L(\text{ee}) \Downarrow^{f} \langle v, D \rangle$

b) $R(\text{ee}) \Downarrow^{f'} \langle v', D' \rangle$

c) $f < m'$

By unfolding the assumption $(m, \text{ee}) \in (\!|\sigma\tau|\!)^{t}_{\varepsilon}$ using (a-b) and $f < m' \leqslant m$ (using (c)), we obtain

d) $\langle \langle v, D \rangle, \text{ee} \rangle \curvearrowright w', \langle v', D' \rangle, c'$

e) $v = L(w') \wedge v' = R(w')$

f) $c' \leqslant t$

g) $(m - f, w') \in (\!|\sigma\tau|\!)_{v}$

We can conclude as follows:

1. By (d)
2. By (e)
3. By (f)
4. By IH 1 on g) using $m' - f \leqslant m - f$, we get $(m' - f, w') \in (\!|\sigma\tau|\!)_{v}$.

□

Proof of statement (5). Assume that $(m, e) \in [\![\sigma A]\!]^{t}_{\varepsilon}$ and $m' \leqslant m$.
TS: $(m', e) \in [\![\sigma A]\!]^{t}_{\varepsilon}$
Assume that

a) $e \Downarrow^{f} \langle v, D \rangle$

b) $f < m'$.

By unfolding the main assumption $(m, e) \in [\![\sigma A]\!]^{t}_{\varepsilon}$ with a) and $f < m' \leqslant m$ (by (b)), we get

c) $f \leqslant t$

d) $(m - f, v) \in [\![\sigma A]\!]_{v}$

We can conclude as follows:

1. By d)
2. By IH 2 on d) using $m' - f \leqslant m - f$, we get $(m' - f, v) \in [\![\sigma A]\!]_{v}$.

□

Lemma 34 (Bi-value propagation). $\langle\langle L(\mathtt{w}), L(\mathtt{w})\rangle, \mathtt{w}\rangle \curvearrowright \mathtt{w}, \langle R(\mathtt{w}), R(\mathtt{w})\rangle, 0$.

Proof. By induction on $\mathtt{w}$. We show some representative cases.

Case: $\mathtt{w} = \mathtt{keep}(\mathtt{n})$
$\mathrm{L}(\mathtt{keep}(\mathtt{n})) = \mathtt{n}$. Immediate from rule **cp-nochange**.

Case: $\mathtt{w} = \mathtt{new}(v, v')$
$\mathrm{L}(\mathtt{new}(v, v')) = v$ and $\mathrm{R}(\mathtt{new}(v, v')) = v'$. Immediate from rule **cp-new**.

Case: $\mathtt{w} = (\mathtt{w}_1, \mathtt{w}_2)$
By IH on $\mathtt{w}_1$, we get $\langle\langle \mathrm{L}(\mathtt{w}_1), \mathrm{L}(\mathtt{w}_1)\rangle, \mathtt{w}_1\rangle \curvearrowright \mathtt{w}_1, \langle \mathrm{R}(\mathtt{w}_1), \mathrm{R}(\mathtt{w}_1)\rangle, 0$ $(\star)$
By IH on $\mathtt{w}_2$, we get $\langle\langle \mathrm{L}(\mathtt{w}_2), \mathrm{L}(\mathtt{w}_2)\rangle, \mathtt{w}_2\rangle \curvearrowright \mathtt{w}_2, \langle \mathrm{R}(\mathtt{w}_2), \mathrm{R}(\mathtt{w}_2)\rangle, 0$ $(\dagger)$
Therefore, by instantiating **cp-pair** rule using $(\star)$ and $(\dagger)$, we get
$\langle\langle \mathrm{L}((\mathtt{w}_1, \mathtt{w}_2)), \mathrm{L}((\mathtt{w}_1, \mathtt{w}_2))\rangle, (\mathtt{w}_1, \mathtt{w}_2)\rangle \curvearrowright (\mathtt{w}_1, \mathtt{w}_2), \langle \mathrm{R}((\mathtt{w}_1, \mathtt{w}_2)), \mathrm{R}((\mathtt{w}_1, \mathtt{w}_2))\rangle, 0$.

Case: $\mathtt{w} = \mathtt{nil}$
Follows immediately from the **cp-nochange** rule $\langle \mathtt{nil}, \mathtt{nil}\rangle \curvearrowright \mathtt{nil}, \mathtt{nil}, 0$.

Case: $\mathtt{w} = \mathtt{cons}(\mathtt{w}_1, \mathtt{w}_2)$
By IH on $\mathtt{w}_1$, we get $\langle\langle \mathrm{L}(\mathtt{w}_1), \mathrm{L}(\mathtt{w}_1)\rangle, \mathtt{w}_1\rangle \curvearrowright \mathtt{w}_1, \langle \mathrm{R}(\mathtt{w}_1), \mathrm{R}(\mathtt{w}_1)\rangle, 0$ $(\star)$
By IH on $\mathtt{w}_2$, we get $\langle\langle \mathrm{L}(\mathtt{w}_2), \mathrm{L}(\mathtt{w}_2)\rangle, \mathtt{w}_2\rangle \curvearrowright \mathtt{w}_2, \langle \mathrm{R}(\mathtt{w}_2), \mathrm{R}(\mathtt{w}_2)\rangle, 0$ $(\dagger)$
Therefore, by instantiating **cp-cons** rule using $(\star)$ and $(\dagger)$, we get
$\langle\langle \mathrm{L}(\mathtt{cons}(\mathtt{w}_1, \mathtt{w}_2)), \mathrm{L}(\mathtt{cons}(\mathtt{w}_1, \mathtt{w}_2))\rangle, \mathtt{cons}(\mathtt{w}_1, \mathtt{w}_2)\rangle \curvearrowright$
$\mathtt{cons}(\mathtt{w}_1, \mathtt{w}_2), \langle \mathrm{R}(\mathtt{cons}(\mathtt{w}_1, \mathtt{w}_2)), \mathrm{R}(\mathtt{cons}(\mathtt{w}_1, \mathtt{w}_2))\rangle, 0$.

Case: $\mathtt{w} = \mathtt{nil}$
Follows immediately from the **cp-nochange** rule $\langle \mathtt{nil}, \mathtt{nil}\rangle \curvearrowright \mathtt{nil}, \mathtt{nil}, 0$.

Case: $\mathtt{w} = \mathtt{fix}\ \mathrm{f}(\mathrm{x}).\text{æ}$
Immediate from rule **cp-fix**.

□

Lemma 35 (No input change). *If* $L(\text{æ}) \Downarrow^{\mathrm{f}} \mathsf{T}$ *and* $\langle \mathsf{T}, \text{æ}\rangle \curvearrowright \mathtt{w}', \mathsf{T}', \mathrm{c}'$ *and* $\mathtt{stable}(\text{æ})$ *then* $\mathtt{stable}(\mathtt{w}')$ *and* $\mathrm{c}' = 0$.

Proof. Immediate from **cp-nochange** change-propagation rule where $w' = \ulcorner v \urcorner$ (hence, $\texttt{stable}(\ulcorner v \urcorner)$)and $T' = \langle v, D \rangle$ and $c' = 0$. □

Lemma 36 (Stable context soundness)**.** *Suppose* $(\forall x \in \Gamma \quad \Delta; \Phi \models \Gamma(x) \sqsubseteq \Box\,\Gamma(x))$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, \theta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$*. Then, the following hold.*

1. *If* $\Delta; \Phi; \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t$ *, then* $\texttt{stable}(\theta\ulcorner e \urcorner)$.
2. *If* $\Delta; \Phi; \Gamma \vdash w \gg \tau$ *and* $\texttt{stable}(w)$*, then* $\texttt{stable}(\theta w)$.
3. *If* $\Delta; \Phi; \Gamma \vdash ee \gg \tau \mid t$ *and* $\texttt{stable}(ee)$*, then* $\texttt{stable}(\theta ee)$.

Proof. All three statements have similar proofs. We show the proof of (1).

Proof of Statement (1). By definition, $\ulcorner e \urcorner$ does not have any occurrence of `new`.
Therefore, it suffices to show that for any $x \in \Gamma$, $\texttt{stable}(\theta(x))$.
Pick any $x \in \Gamma$.
From the definition of $\mathcal{G}(\!|\sigma\Gamma|\!)$, $(m, \theta(x)) \in (\!|\sigma(\Gamma(x))|\!)_v$.
By lemma 39, $(m, \theta(x)) \in (\!|\Box\,\sigma(\Gamma(x))|\!)_v$.
From the definition of $(\!|\Box\,\sigma(\Gamma(x))|\!)_v$, we get $\texttt{stable}(\theta(x))$, as needed. □

□

Lemma 37 (Stable type)**.** *The following hold.*

1. *If and only if* $(m, w) \in (\!|\tau|\!)_v$ *and* $\texttt{stable}(w)$*, then* $(m, w) \in (\!|\Box\,\tau|\!)_v$.
2. *If* $(m, ee) \in (\!|\tau|\!)^t_\varepsilon$ *and* $0 \leqslant t$ *and* $\texttt{stable}(ee)$*, then* $(m, ee) \in (\!|\Box\,\tau|\!)^0_\varepsilon$.

Proof. (1) follows immediately by definition. (2) is proved using statement (1).

Proof of statement (2)

Assume that $(m, ee) \in (\!|\tau|\!)^t_\varepsilon$ $(\star)$ and $0 \leqslant t$ and $\texttt{stable}(ee)$ $(\star\star)$.
TS: $(m, ee) \in (\!|\Box\,\tau|\!)^0_\varepsilon$
Assume $f < m$ such that $L(ee) \Downarrow^f \langle v, D \rangle$ $(\dagger)$ and $R(ee) \Downarrow^{f'} \langle v', D' \rangle$ $(\spadesuit)$.
By unrolling the $(\star)$ with $(\dagger)$ and $(\spadesuit)$, we obtain

a) $\langle\langle v, D\rangle, \text{ææ}\rangle \curvearrowright w', \langle v', D'\rangle, c'$
b) $v' = R(w') \ \wedge \ v = L(w')$
c) $c' \leqslant t$
d) $(m - f, w') \in (\!|\tau|\!)_v$

Note that since $\texttt{stable}(\text{ææ})$, we know that

e) $L(\text{ææ}) = R(\text{ææ})$, therefore $v = v'$.
f) By lemma 35 using (†) and $\texttt{stable}(\text{ææ})$, we know that $c' = 0$ and $w' = \ulcorner v \urcorner$.

Hence, we can conclude as follows:

1. By (a)
2. By (b)
3. By (c), $c' = 0 \leqslant 0$
4. By (d) and (f), we know that $\texttt{stable}(w') = \texttt{stable}(\ulcorner v \urcorner)$, hence $(m - f, w') \in (\!|\Box\, \tau|\!)_v$

□

Lemma 38 (Bi-value inclusion)**.** *The following hold.*

1. *If* $(m, w) \in (\!|\tau|\!)_v$, *then* $(m, w) \in (\!|U\, |\tau|\!)_v$.
2. *If* $(m, \delta) \in \mathcal{G}(\!|\Gamma|\!)$, *then* $(m, \delta) \in \mathcal{G}(\!|U\, |\Gamma|\!)$.

Proof. (1) is proven by induction on τ and (2) follows from (1).
We show a few representative cases of (1) below.

Case: $(m, \texttt{keep}(n)) \in (\!|\text{int}_r|\!)_v$.
TS: $(m, \texttt{keep}(n)) \in (\!|U\, |\text{int}_r|\!)_v = (\!|U\, \text{int}|\!)_v$.
Follows by definition since $(m, \texttt{keep}(n)) \in \mathcal{C}\ \text{int}\ \mathcal{D}_v \subseteq (\!|U\, \text{int}|\!)_v$.

Case: $(m, w) \in (\!|U\, A|\!)_v$ $(\star)$
TS: $(m, w) \in (\!|U\, |U\, A|\!)_v = (\!|U\, A|\!)_v$.
We immediately conclude by $(\star)$.

Case: $(m, \text{inl}\ w) \in (\!|\tau_1 + \tau_2|\!)_v$ $(\star)$
TS: $(m, \text{inl}\ w) \in (\!|U\, |\tau_1 + \tau_2|\!)_v$.

STS:$(m, \text{inl } w) \in \mathbb{C}\, |\tau_1 + \tau_2|\, \mathbb{D}_v$.
By unrolling its definition and noting that $|\tau_1 + \tau_2| = |\tau_1| + |\tau_2|$,
RTS: $(m, w) \in (\!|\mathsf{U}\, |\tau_1|\!|)_v$.
By unrolling the definition of $(\star)$, we have

$$(m, w) \in (\!|\tau_1|\!|)_v \tag{2}$$

By IH 1 on eq. (2), we conclude.

Case: $(m, \text{fix } f(x).e) \in (\!|\tau_1 \xrightarrow{\mathbb{CP}(k)} \tau_2|\!|_v$ $(\star)$
TS: $(m, \text{fix } f(x).e) \in (\!|\mathsf{U}\, |\tau_1 \xrightarrow{\mathbb{CP}(k)} \tau_2|\!|_v = (\!|\mathsf{U}\, (|\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\tau_2|)|\!|_v$.
STS: $(m, \text{fix } f(x).e) \in \mathbb{C}\, |\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\tau_2|\, \mathbb{D}_v$.
Follows immediately by unrolling the second part of the definition of $(\star)$.

Case: $(m, \texttt{nil}) \in (\!|\text{list}[0]^{\alpha}\, \tau|\!|_v$
TS: $(m, \texttt{nil}) \in (\!|\mathsf{U}\, (\text{list}[0]\, |\tau|)|\!|_v$
This follows immediately by definition since $(m, \texttt{nil}) \in \mathbb{C}\, \text{list}[0]\, |\tau|\, \mathbb{D}_v \subseteq (\!|\mathsf{U}\, (\text{list}[0]\, |\tau|)|\!|_v$

Case: $(m, \texttt{cons}(w_1, w_2)) \in (\!|\text{list}[I+1]^{\alpha}\, \tau|\!|_v$ $(\star)$
TS:$(m, \texttt{cons}(w_1, w_2)) \in (\!|\mathsf{U}\, (|\text{list}[I+1]^{\alpha}\, \tau|)|\!|_v$.
By unrolling its definition, STS: $(m, \texttt{cons}(w_1, w_2)) \in \mathbb{C}\, \text{list}[I+1]\, |\tau|\, \mathbb{D}_v$.
STS 1: $(m, w_1) \in (\!|\mathsf{U}\, |\tau|\!|_v$ $(\diamond)$
STS 2: $(m, w_2) \in (\!|\mathsf{U}\, (\text{list}[I]\, |\tau|)|\!|_v$ $(\diamond\diamond)$.
There are two cases for unrolling the definition of $(\star)$.

subcase 1: We have $(m, w_1) \in (\!|\tau|\!|_v$ $(\dagger)$ and $(m, w_2) \in (\!|\text{list}[I]^{\alpha-1}\, \tau|\!|_v$ $(\dagger\dagger)$
By IH 1 on $(\dagger)$, we get $(\diamond)$
By IH 1 on $(\dagger\dagger)$, we get $(m, w_2) \in (\!|\mathsf{U}\, (|\text{list}[I]^{\alpha-1}\, \tau|)|\!|_v = (\!|\mathsf{U}\, (\text{list}[I]\, |\tau|)|\!|_v$.

subcase 2: We have $(m, w_1) \in (\!|\Box\, \tau|\!|_v$ $(\dagger)$ and $(m, w_2) \in (\!|\text{list}[I]^{\alpha}\, \tau|\!|_v$ $(\dagger\dagger)$
By IH 1 on $(\dagger)$, we get $(\diamond)$
By IH 1 on $(\dagger\dagger)$, we get $(m, w_2) \in (\!|\mathsf{U}\, (|\text{list}[I]^{\alpha}\, \tau|)|\!|_v = (\!|\mathsf{U}\, (\text{list}[I]\, |\tau|)|\!|_v$.

□

Lemma 39 (Bi-value subtyping soundness). *The following hold.*

1. *If* $\Delta; \Phi \models \tau \sqsubseteq \tau'$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, w) \in (\!|\sigma\tau|\!)_v$, *then* $(m, w) \in (\!|\sigma\tau'|\!)_v$.

2. *If* $\Delta; \Phi \models^A A \sqsubseteq A'$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, v) \in [\![\sigma A]\!]_v$, *then* $(m, v) \in [\![\sigma A']\!]_v$.

3. *If* $\Delta; \Phi \models \tau \sqsubseteq \tau'$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, ee) \in (\!|\sigma\tau|\!)^t_\varepsilon$ *and* $t \leqslant t'$, *then* $(m, ee) \in (\!|\sigma\tau'|\!)^{t'}_\varepsilon$.

4. *If* $\Delta; \Phi \models A \sqsubseteq A'$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, e) \in [\![\sigma A]\!]^t_\varepsilon$ *and* $t \leqslant t'$, *then* $(m, e) \in [\![\sigma A']\!]^{t'}_\varepsilon$.

5. *If* $\Delta; \Phi \models \tau \sqsubseteq \tau'$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $(m, v) \in [\![|\sigma\tau|]\!]_v$, *then* $(m, v) \in [\![|\sigma\tau'|]\!]_v$.

6. *If* $\Delta; \Phi \models \tau \sqsubseteq \tau'$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $(m, e) \in [\![|\sigma\tau|]\!]^t_\varepsilon$ *and* $t \leqslant t'$, *then* $(m, e) \in [\![|\sigma\tau'|]\!]^{t'}_\varepsilon$.

7. *If* $\Delta; \Phi \models^A A \sqsubseteq A'$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, w) \in (\!|\, \sigma A \,|\!)_v$, *then* $(m, w) \in (\!|\, \sigma A' \,|\!)_v$.

8. *If* $\Delta; \Phi \models^A A \sqsubseteq A'$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, w) \in (\!|U\, \sigma A|\!)_v$, *then* $(m, w) \in (\!|U\, \sigma A'|\!)_v$.

9. *If* $\Delta; \Phi \models^A A \sqsubseteq A'$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, ee) \in (\!|U\, \sigma A|\!)^t_\varepsilon$ *and* $t \leqslant t'$, *then* $(m, ee) \in (\!|U\, \sigma A'|\!)^{t'}_\varepsilon$.

10. *If* $\Delta; \Phi \models \tau \sqsubseteq \tau'$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, w) \in (\!|\, |\sigma\tau| \,|\!)_v$, *then* $(m, w) \in (\!|\, |\sigma\tau'| \,|\!)_v$.

11. *If* $\Delta; \Phi \models \tau \sqsubseteq \tau'$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, w) \in (\!|U\, |\sigma\tau||\!)_v$, *then* $(m, w) \in (\!|U\, |\sigma\tau'||\!)_v$.

12. *If* $\Delta; \Phi \models \tau \sqsubseteq \tau'$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, ee) \in (\!|U\, |\sigma\tau||\!)^{\sigma t}_\varepsilon$ *and* $t \leqslant t'$, *then* $(m, ee) \in (\!|U\, |\sigma\tau'||\!)^{\sigma t'}_\varepsilon$.

Proof. Statements (1) and (2) are by proven simultaneously by induction on the subtyping derivation. We first show the proof of statements (3), (4), (6), ((11) and (12). □

Proof of statement (3). Assume that $\Delta; \Phi \models \tau \sqsubseteq \tau'$ and $\sigma \in \mathcal{D}[\![\Delta]\!]$ and $(m, ee) \in (\!|\sigma\tau|\!)^t_\varepsilon$ and $t \leqslant t'$.
TS: $(m, ee) \in (\!|\sigma\tau'|\!)^{t'}_\varepsilon$
Assume that

a) $L(ee) \Downarrow^f \langle v, D\rangle$
b) $R(ee) \Downarrow^{f'} \langle v', D'\rangle$
c) $f < m$

By unfolding the assumption $(m, ee) \in (\!|\sigma\tau|\!)^t_\varepsilon$ using (a-c), we obtain

d) $\langle\langle v, D\rangle, ee\rangle \curvearrowright w', \langle v', D'\rangle, c'$
e) $v = L(w') \wedge v' = R(w')$
f) $c' \leqslant t$
g) $(m - f, w') \in (\!|\sigma\tau|\!)_v$

We can conclude as follows:

1. By (d)
2. By (e)
3. Since $c' \leqslant t$ from (f) and $t \leqslant t'$ from the assumption, we get $c' \leqslant t'$.
4. By IH 1 on g), we get $(m - f, w') \in (\!|\sigma\tau'|\!)_v$.

□

Proof of statement (4). Assume that $\Delta; \Phi \models A \sqsubseteq A'$ and $\sigma \in \mathcal{D}[\![\Delta]\!]$ and $(m, e) \in [\![\sigma A]\!]^t_\varepsilon$ and $t \leqslant t'$.
TS: $(m, e) \in [\![\sigma A']\!]^{t'}_\varepsilon$
Assume that $e \Downarrow^f \langle v, D\rangle$ and $f < m$.
By unfolding the main assumption $(m, e) \in [\![\sigma A]\!]^t_\varepsilon$ with $e \Downarrow^f \langle v, D\rangle$ and $f < m$, we get

a) $f \leqslant t$
b) $(m - f, v) \in [\![\sigma A]\!]_v$

We can conclude as follows:

1. Since $t \leqslant t'$ (from the assumption) and $f \leqslant t$ (from (a)), we get $f \leqslant t'$.

2. By IH 2 on the main assumption using b).

□

Proof of statement (6). Assume that $\Delta; \Phi \models \tau \sqsubseteq \tau'$ and $\sigma \in \mathcal{D}[\![\Delta]\!]$ and $(m, e) \in [\![|\sigma\tau|]\!]^t_\varepsilon$ and $t \leqslant t'$.
TS: $(m, e) \in [\![|\sigma\tau'|]\!]^{t'}_\varepsilon$
Assume that

a) $e \Downarrow^f v$

b) $f < m$

By unfolding the main assumption $(m, e) \in [\![|\sigma\tau|]\!]^t_\varepsilon$ with (a-b), we get

c) $f \leqslant t$

d) $(m - f, v) \in [\![|\sigma\tau|]\!]_v$

We can conclude as follows:

1. Since $t \leqslant t'$ (from the assumption) and $f \leqslant t$ (from (c)), we get $f \leqslant t'$.

2. By IH 5 on the main assumption using d).

□

Proof of statement (11). Assume that $\Delta; \Phi \models \tau \sqsubseteq \tau'$ and $\sigma \in \mathcal{D}[\![\Delta]\!]$ and $(m, w) \in (\!|U\,|\sigma\tau|)\!|_v$ $(\star)$.
TS: $(m, w) \in (\!|U\,|\sigma\tau'|)\!|_v$
There are two cases for the main assumption $(\star)$.

subcase 1: $w = \mathsf{new}(v, v')$
Then, we have $\forall j.(j, v) \in [\![|\sigma\tau|]\!]_v \wedge (j, v) \in [\![|\sigma\tau|]\!]_v$ $(\star\star)$.
TS: $\forall j.(j, v) \in [\![|\sigma\tau'|]\!]_v \wedge (j, v) \in [\![|\sigma\tau'|]\!]_v$.
We conclude by instantiating IH 5 on $\Delta; \Phi \models \tau \sqsubseteq \tau'$ using $(\star\star)$.

subcase 2: $w \neq \mathsf{new}(\nu, \nu')$

Then, we have $(m, w) \in \mathbb{C}\,|\sigma\tau|\,\mathbb{D}_\nu$ $(\star\star)$.

We conclude by instantiating IH 10 on $\Delta; \Phi \models \tau \sqsubseteq \tau'$ using $(\star\star)$.

□

Proof of statement (12). Assume that $\Delta; \Phi \models \tau \sqsubseteq \tau'$ and $\sigma \in \mathcal{D}[\![\Delta]\!]$ and $(m, ee) \in (\!|\mathsf{U}\,|\sigma\tau|\!|)^t_\varepsilon$ and $t \leqslant t'$.

TS: $(m, ee) \in (\!|\mathsf{U}\,|\sigma\tau'|\!|)^{t'}_\varepsilon$

Assume that

a) $L(ee) \Downarrow^f \langle \nu, D \rangle$

b) $R(ee) \Downarrow^{f'} \langle \nu', D' \rangle$

c) $f < m$

By unfolding the assumption $(m, ee) \in (\!|\mathsf{U}\,|\sigma\tau|\!|)^t_\varepsilon$ using (a-c), we obtain

d) $\langle \langle \nu, D \rangle, ee \rangle \curvearrowright w', \langle \nu', D' \rangle, c'$

e) $\nu = L(w') \;\wedge\; \nu' = R(w')$

f) $c' \leqslant t$

g) $(m - f, w') \in (\!|\mathsf{U}\,|\sigma\tau|\!|)_\nu$

We can conclude as follows:

1. By (d)
2. By (e)
3. Since $c' \leqslant t$ from (f) and $t \leqslant t'$ from the assumption, we get $c' \leqslant t'$.
4. By IH 11 on g), we get $(m - f, w') \in (\!|\mathsf{U}\,|\sigma\tau'|\!|)_\nu$.

□

Proof of statement (1). Proof is by induction on the subtyping derivation.

Case: $\dfrac{}{\Delta; \Phi \models \Box\,\mathsf{U}\,\mathbf{int} \sqsubseteq \mathbf{int}_\tau}\ \Box\mathbf{U\text{-}int}$

We have $(m, w) \in (\!|\Box\,\mathsf{U}\,\mathrm{int}|\!)_\nu$.

Unrolling its definition, we have $w = \mathsf{keep}(n)$.

TS: $(m, \mathsf{keep}(n)) \in (\!|\mathrm{int}_\tau|\!)_\nu$.

Follows directly by definition.

Case: $\dfrac{\Delta;\Phi \models \tau_1' \sqsubseteq \tau_1 \qquad \Delta;\Phi \models \tau_2 \sqsubseteq \tau_2' \qquad \Delta;\Phi \models t \leqslant t'}{\Delta;\Phi \models \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2 \sqsubseteq \tau_1' \xrightarrow{\mathbb{CP}(t')} \tau_2'} \to \text{cp}$

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$.

We have

$$(m, \text{fix } f(x).ee) \in (\!|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)_v \tag{1}$$

TS: $(m, \text{fix } f(x).ee) \in (\!|\sigma\tau_1' \xrightarrow{\mathbb{CP}(\sigma t')} \sigma\tau_2'|\!)_v$.

There are two cases to show.

subcase 1: Assume that $j < m$ and $(j, w) \in (\!|\sigma\tau_1'|\!)_v$.

STS: $(j, e[w/x, (\text{fix } f(x).ee)/f]) \in (\!|\sigma\tau_2'|\!)_\varepsilon^{\sigma t'}$.

By IH 1 on $(j, w) \in (\!|\sigma\tau_1'|\!)_v$ using the first premise, we get

$$(j, w) \in (\!|\sigma\tau_1|\!)_v \tag{2}$$

By unrolling eq. (1) with eq. (2) using $j < m$, we get

$$(j, e[w/x, (\text{fix } f(x).ee)/f]) \in (\!|\sigma\tau_2|\!)_\varepsilon^{\sigma t} \tag{3}$$

By Assumption assumption 25 on the third premise, we get $\sigma t \leqslant \sigma t'$.

We conclude by applying IH 3 to eq. (3) using the second premise and $\sigma t \leqslant \sigma t'$.

subcase 2: TS: $(m, \text{fix } f(x).ee) \in \mathbb{C}\,|\sigma\tau_1' \xrightarrow{\mathbb{CP}(\sigma t')} \sigma\tau_2'|\,\mathbb{D}_v = \mathbb{C}\,|\sigma\tau_1'| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2'|\,\mathbb{D}_v$.

Assume that $j < m$ and $(j, w) \in (\!|\mathbb{U}\,|\sigma\tau_1'|\!|)_v$ $(\star)$.

STS: $(j, e[w/x, (\text{fix } f(x).ee)/f]) \in (\!|\mathbb{U}\,|\sigma\tau_2'|\!|)_\varepsilon^{\infty}$.

By IH 11 on the first premise using $(\star)$, we get

$$(j, w) \in (\!|\mathbb{U}\,|\sigma\tau_1|\!|)_v \tag{4}$$

By unrolling the second part of eq. (1)'s definition, we get

$$(m, \text{fix } f(x).ee) \in \mathbb{C}\,|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|\,\mathbb{D}_v \tag{5}$$

Next, we unroll eq. (5) with eq. (4) using $j < m$, we get

$$(j, e[\mathtt{w}/x, (\text{fix } f(x).\mathtt{ee})/f]) \in (\!|\mathcal{U}\,|\sigma\tau_2|\!|)^{\infty}_{\varepsilon} \tag{6}$$

We conclude by applying IH 12 to eq. (6) using the second premise.

Case: $\dfrac{}{\Delta; \Phi \models \Box\,(\tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2) \sqsubseteq \Box\,\tau_1 \xrightarrow{\mathbb{CP}(0)} \Box\,\tau_2} \to \Box_{\mathbf{cp}}$

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$.

We have

$$(m, \text{fix } f(x).\mathtt{ee}) \in (\!|\Box\,(\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2)|\!|_v \tag{1}$$

and

$$\mathtt{stable}(\text{fix } f(x).\mathtt{ee}) \tag{2}$$

TS: $(m, \text{fix } f(x).\mathtt{ee}) \in (\!|\Box\,\sigma\tau_1 \xrightarrow{\mathbb{CP}(0)} \Box\,\sigma\tau_2|\!|_v$.

There are two cases:

subcase 1: Assume that

a) $j < m$

b) $(j, \mathtt{w}) \in (\!|\Box\,\sigma\tau_1|\!|_v$ (note that $\mathtt{stable}(\mathtt{w})$ $(\star)$).

STS: $(j, \mathtt{ee}[\mathtt{w}/x, (\text{fix } f(x).\mathtt{ee})/f]) \in (\!|\Box\,\sigma\tau_2|\!|^{0}_{\varepsilon}$.

Since we know by eq. (2) and $(\star)$ that $\mathtt{stable}(\mathtt{ee}[\mathtt{w}/x, (\text{fix } f(x).\mathtt{ee})/f])$, by lemma 37,

RTS: $(j, \mathtt{ee}[\mathtt{w}/x, (\text{fix } f(x).\mathtt{ee})/f]) \in (\!|\sigma\tau_2|\!|^{\sigma k}_{\varepsilon}$.

This can be shown by unrolling the definition of eq. (1) with (a) and (b).

subcase 2: STS: $(m, \text{fix } f(x).\mathtt{ee}) \in \mathbb{C}\,|\Box\,\sigma\tau_1 \xrightarrow{\mathbb{CP}(0)} \Box\,\sigma\tau_2|\,\mathbb{D}_v = \mathbb{C}\,|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|\,\mathbb{D}_v$.

Immediately follows by unrolling the second part of the defini-

tion of eq. (1) since $\mathbb{C}\,|\Box\,(\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} s\tau_2)|\,\mathbb{D}_v = \mathbb{C}\,|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|\,\mathbb{D}_v$

Case: $\dfrac{}{\Delta;\Phi \models \Box\,(U\,(A_1 \xrightarrow{\mathbb{FS}(t)} A_2)) \sqsubseteq \Box\,U\,A_1 \xrightarrow{\mathbb{CP}(0)} \Box\,U\,A_2} \to \Box U_{cp}$

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$.

We have

$$(m, \text{fix } f(x).ee) \in (\!|\Box\,U\,(\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2)|\!)_v \tag{1}$$

and

$$\texttt{stable}(\text{fix } f(x).ee) \tag{2}$$

TS: $(m, \text{fix } f(x).ee) \in (\!|\Box\,U\,\sigma A_1 \xrightarrow{\mathbb{CP}(0)} \Box\,U\,\sigma A_2|\!)_v$.

There are two cases:

subcase 1: Assume that

a) $j < m$

b) $(j, w) \in (\!|\Box\,U\,\sigma A_1|\!)_v$ (note that $\texttt{stable}(w)$ $(\star)$).

STS: $(j, ee[w/x, (\text{fix } f(x).ee)/f]) \in (\!|\Box\,U\,\sigma A_2|\!)^0_\varepsilon$.

Unrolling its definition, assume that

c) $L(ee[w/x, (\text{fix } f(x).ee)/f]) \Downarrow^{f_r} \langle v_r, D_r\rangle$

d) $R(ee[w/x, (\text{fix } f(x).ee)/f]) \Downarrow^{f'_r} \langle v'_r, D'_r\rangle$.

e) $f_r < j$

Now, we can conclude as follows:

1. By $(\star)$ and eq. (2), we have $\texttt{stable}(ee[w/x, (\text{fix } f(x).ee)/f])$ $(\diamond)$. Therefore, by **cp-nochange** rule, we get

$$\frac{\texttt{stable}(ee[w/x, (\text{fix } f(x).ee)/f])}{\langle\langle v_r, D_r\rangle, ee[w/x, (\text{fix } f(x).ee)/f]\rangle \curvearrowright \ulcorner v_r \urcorner, \langle v_r, D_r\rangle, 0}\ \textbf{cp-nochange}$$

2. Since $v_r = v'_r$ (by ($\diamond$)), trivially $v_r = L(\ulcorner v_r \urcorner) \wedge v_r = R(\ulcorner v_r \urcorner)$.
3. $c' = 0 \leqslant 0$
4. TS: $(j - f_r, \ulcorner v_r \urcorner) \in (\!|\Box \cup \sigma A_2|\!)_v$.
   Since $\texttt{stable}(\ulcorner v_r \urcorner)$,
   TS: $(m - f_r, \ulcorner v_r \urcorner) \in (\!|\cup \sigma A_2|\!)_v$.
   Next, by unrolling the definition of eq. (1), we get

$$(m, \text{fix } f(x).ee) \in \mathbb{C}\, |\Box \cup (\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2)|\, \mathbb{D}_v = \mathbb{C}\, \sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2\, \mathbb{D}_v \tag{3}$$

   Unrolling the definition of eq. (3) with (a-b) we get

$$(j, ee[w/x, (\text{fix } f(x).ee)/f]) \in (\!|\cup \sigma A_2|\!)^{\sigma t}_{\varepsilon} \tag{4}$$

   Next, by unrolling the definition of eq. (4) with (c-e),we get

   f) $\langle T_r, ee[w/x, (\text{fix } f(x).ee)/f]\rangle \curvearrowright w'_r, T'_r, c'_r$ where we know that $w'_r = \ulcorner v_r \urcorner$
   g) $(j - f_r, \ulcorner v_r \urcorner) \in (\!|\cup \sigma A_2|\!)_v$

   We conclude this subcase by g).

subcase 2: STS: $(m, \text{fix } f(x).ee) \in \mathbb{C}\, |\Box \cup \sigma A_1 \xrightarrow{\mathbb{CP}(0)} \Box \cup \sigma A_2|\, \mathbb{D}_v = \mathbb{C}\, \sigma A_1 \xrightarrow{\mathbb{FS}(\infty)} \sigma A_2\, \mathbb{D}_v$.
Pick j and assume that

a) $j < m$
b) $(j, w) \in (\!|\cup \sigma A_1|\!)_v$.

STS: $(j, ee[w/x, (\text{fix } f(x).ee)/f]) \in (\!|\cup \sigma A_2|\!)^{\infty}_{\varepsilon}$
Next, by unrolling the definition of eq. (1), we get

$$(m, \text{fix } f(x).ee) \in \mathbb{C}\, |\Box \cup (\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2)|\, \mathbb{D}_v = \mathbb{C}\, \sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2\, \mathbb{D}_v \tag{5}$$

By unrolling the definition of eq. (5) with (a-b), we get

$$(j, \text{ææ}[w/x, (\text{fix } f(x).\text{ææ})/f]) \in (\!|\mathcal{U}\ \sigma A_2|\!)_{\varepsilon}^{\sigma t} \tag{6}$$

Then, we can conclude by IH 3 on eq. (6) using $\sigma t \leqslant \infty$.

Case: $\dfrac{}{\Delta; \Phi \models \tau \sqsubseteq \mathcal{U}\,|\tau|}$ **W**

By lemma 38.

Case: $\dfrac{\Delta; \Phi \models n \doteq n' \qquad \Delta; \Phi \models \alpha \leqslant \alpha' \qquad \Delta; \Phi \models \tau \sqsubseteq \tau'}{\Delta; \Phi \models \mathbf{list}[n]^{\alpha}\,\tau \sqsubseteq \mathbf{list}[n']^{\alpha'}\,\tau'}$ **l1**

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$ and $\models \sigma\Phi$ and $(m, w) \in (\!|\text{list}[n]^{\alpha}\,\tau|\!)_v$.

TS: $(m, w) \in (\!|\text{list}[\sigma n']^{\sigma\alpha'}\,\sigma\tau'|\!)_v$

From assumption 25 applied to the first premise, $\sigma n = \sigma n'$. Therefore,

STS: $(m, w) \in (\!|\text{list}[\sigma n]^{\sigma\alpha'}\,\sigma\tau'|\!)_v$

From assumption 25 applied to the second premise, $\sigma\alpha \leqslant \sigma\alpha'$. Therefore,

We prove the following more general statement

$\forall\ m, w, n, \alpha, \alpha'$. if $\alpha \leqslant \alpha'$ and $(m, w) \in (\!|\text{list}[\sigma n]^{\sigma\alpha}\,\sigma\tau|\!)_v$, then $(m, w) \in (\!|\text{list}[\sigma n]^{\sigma\alpha'}\,\sigma\tau'|\!)_v$.

We establish this statement by subinduction on w.

subcase 1: $w = \texttt{nil}$

We can immediately conclude that $(m, \texttt{nil}) \in (\!|\text{list}[0]^{\sigma\alpha'}\,\sigma\tau'|\!)_v$ by definition.

subcase 2: $v = \texttt{cons}(w_1, w_2)$ and $v' = \text{cons}(v_1', v_2')$

TS: $(m, \texttt{cons}(w_1, w_2)) \in (\!|\text{list}[I + 1]^{\sigma\alpha'}\,\sigma\tau'|\!)_v$ for some $I + 1 = \sigma n$.

We have two possible cases:

- $(m, v_1, v_1') \in (\!|\square\,\sigma\tau|\!)_v$ (†) and $(m, v_2, v_2') \in (\!|\text{list}[I]^{\sigma\alpha}\,\sigma\tau|\!)_v$ (††).
  By subIH on (††), we get

$$(m, w_2) \in (\!|\text{list}[I]^{\sigma\alpha'}\,\sigma\tau'|\!)_v \tag{1}$$

By IH on (†), we get

$$(m, v_1, v_1') \in (\!|\Box\, \sigma\tau'|\!)_v \tag{2}$$

Combining eq. (2) with eq. (1), we get $(m, \mathtt{cons}(w_1, w_2)) \in (\!|\mathrm{list}[I+1]^{\sigma\alpha'}\, \sigma\tau'|\!)_v$.

- $(m, w_1) \in (\!|\sigma\tau|\!)_v$ ($\diamond$) and $(m, w_2) \in (\!|\mathrm{list}[I]^{\sigma\alpha-1}\, \sigma\tau|\!)_v$ ($\diamond\diamond$).
  By subIH on ($\diamond\diamond$), we get

$$(m, w_2) \in (\!|\mathrm{list}[I]^{\sigma\alpha'-1}\, \sigma\tau'|\!)_v \tag{3}$$

By IH on ($\diamond$), we get

$$(m, w_1) \in (\!|\sigma\tau'|\!)_v \tag{4}$$

Combining eq. (4) with eq. (3), we get $(m, \mathtt{cons}(w_1, w_2)) \in (\!|\mathrm{list}[I+1]^{\sigma\alpha'}\, \sigma\tau'|\!)_v$.

Case: $\dfrac{}{\Delta; \Phi \models \mathbf{list}[n]^{\alpha}\, \Box\, \tau \sqsubseteq \Box\, (\mathbf{list}[n]^{\alpha}\, \tau)}\ \mathbf{l}\Box$

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$ and $\models \sigma\Phi$ and $(m, w) \in (\!|\mathrm{list}[\sigma n]^{\sigma\alpha}\, \Box\, \sigma\tau|\!)_v$.

TS: $(m, w) \in (\!|\Box\, (\mathrm{list}[n]^{\alpha}\, \tau)|\!)_v$

We prove the following more general statement

$\forall i, \beta, \tau$. if $(m, w) \in (\!|\mathrm{list}[i]^{\beta}\, \Box\, \tau|\!)_v$, then $(m, w) \in (\!|\Box\, (\mathrm{list}[i]^{\beta}\, \tau)|\!)_v$ by subinduction on i.

subcase 1: $n = 0$

Then, we know that $w = \mathtt{nil}$

We can immediately conclude that $(m, \mathtt{nil}) \in (\!|\Box\, \mathrm{list}[0]^{\sigma\alpha}\, \sigma\tau|\!)_v$ by definition.

subcase 2: $n = I + 1$

TS: $(m, \mathtt{cons}(w_1, w_2)) \in (\!|\Box\, \mathrm{list}[I+1]^{\sigma\alpha}\, \sigma\tau|\!)_v$.

For the sub-assumption, we have two possible cases:

- $(m, w_1) \in (\!|\Box\,\Box\,\sigma\tau|\!)_\nu$ (†) and $(m, w_2) \in (\!|\mathsf{list}[I]^{\sigma\alpha}\,\Box\,\sigma\tau|\!)_\nu$ (††).
  Instantiating subIH on (††), we get

$$(m, w_2) \in (\!|\Box\,\mathsf{list}[I]^{\sigma\alpha}\,\sigma\tau|\!)_\nu \text{ i.e. } \mathtt{stable}(w_2) \tag{1}$$

  By (†), we also know that

$$(m, w_1) \in (\!|\Box\,\sigma\tau|\!)_\nu \text{ i.e. } \mathtt{stable}(w_1) \tag{2}$$

  Combining eq. (2) with eq. (1), we get $(m, \mathtt{cons}(w_1, w_2)) \in (\!|\Box\,\mathsf{list}[I+1]^{\sigma\alpha}\,\sigma\tau|\!)_\nu$.
- $(m, w_1) \in (\!|\Box\,\sigma\tau|\!)_\nu$ (◇) and $(m, w_2) \in (\!|\mathsf{list}[I]^{\sigma\alpha-1}\,\Box\,\sigma\tau|\!)_\nu$ (◇◇).
  Instantiating subIH on (◇◇), we get

$$(m, w_2) \in (\!|\Box\,\mathsf{list}[I]^{\sigma\alpha-1}\,\sigma\tau|\!)_\nu \text{ and } \mathtt{stable}(w_2) \tag{3}$$

  Combining (◇) with eq. (3), we get $(m, \mathtt{cons}(w_1, w_2)) \in (\!|\Box\,\mathsf{list}[I+1]^{\sigma\alpha}\,\sigma\tau|\!)_\nu$.

□

Proof of statement (2). Proof is by induction on the subtyping derivation.

Case: $\dfrac{\Delta;\Phi \models^A A_1' \sqsubseteq A_1 \qquad \Delta;\Phi \models^A A_2 \sqsubseteq A_2' \qquad \Delta;\Phi \models t \leqslant t'}{\Delta;\Phi \models^A A_1 \xrightarrow{\mathbb{FS}(t)} A_2 \sqsubseteq A_1' \xrightarrow{\mathbb{FS}(t')} A_2'} \to \textbf{exec}$

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$.

We have

$$(m, \mathsf{fix}\ f(x).e) \in [\![\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2]\!]_\nu \tag{1}$$

TS: $(m, \mathsf{fix}\ f(x).e) \in [\![\sigma A_1' \xrightarrow{\mathbb{FS}(\sigma t')} \sigma A_2']\!]_\nu$.

STS: $(m, \mathsf{fix}\ f(x).e) \in [\![\sigma A_1' \xrightarrow{\mathbb{FS}(\sigma t')} \sigma A_2']\!]_\nu$.

Pick j and assume that

$$j < m \tag{2}$$

$$(j, v) \in [\![\sigma A_1']\!]_v \tag{3}$$

STS: $(j, e[v/x, (\text{fix } f(x).e)/f]) \in [\![\sigma A_2']\!]_\varepsilon^{\sigma t'}$.
By IH 2 on eq. (3) using the first premise, we get

$$(j, v) \in [\![\sigma A_1]\!]_v \tag{4}$$

By unrolling the definition of eq. (1) with eq. (4) and $j < m$, we get

$$(j, e[v/x, (\text{fix } f(x).e)/f]) \in [\![\sigma A_2]\!]_\varepsilon^{\sigma t} \tag{5}$$

By Assumption 25 on the third and fourth premises, we get $\sigma t \leqslant \sigma t'$.
We conclude by applying IH 4 to eq. (5) using σ, i.e $\sigma t \leqslant \sigma t'$.

Case: $\dfrac{i :: S, \Delta; \Phi \models^A A \sqsubseteq A' \qquad i \notin FV(\Phi)}{\Delta; \Phi \models^A \exists i{::}S.\, A \sqsubseteq \exists i{::}S.\, A'}$ **u-∃**

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$.
We have

$$(m, \text{pack } v) \in [\![\exists i{::}S.\, \sigma A]\!]_v \tag{1}$$

TS: $(m, \text{pack } v) \in [\![\exists i{::}S.\, \sigma A']\!]_v$.
By unrolling its definition, assume that $\vdash I :: S$ $(\star)$.
STS: $(m, v) \in [\![\sigma A'\{I/i\}]\!]_v$.
By unrolling eq. (1) with $\star$, we get

$$(m, v) \in [\![\sigma A\{I/i\}]\!]_v \tag{2}$$

Then, we can conclude by IH 2 on eq. (2).

□

Proof of statement (5). Proof is by induction on the subtyping derivation.

Case: $\dfrac{\Delta;\Phi \models \tau_1' \sqsubseteq \tau_1 \qquad \Delta;\Phi \models \tau_2 \sqsubseteq \tau_2' \qquad \Delta;\Phi \models t \leqslant t'}{\Delta;\Phi \models \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2 \sqsubseteq \tau_1' \xrightarrow{\mathbb{CP}(t')} \tau_2'} \to \textbf{cp}$

Assume that $(m, \text{fix } f(x).e) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_v$ $(\star)$.
TS: $(m, \text{fix } f(x).e) \in [\![|\sigma\tau_1'| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2'|]\!]_v$.
Pick j and assume that

$$j < m \tag{1}$$

$$(j, v) \in [\![|\sigma\tau_1'|]\!]_v \tag{2}$$

STS: $(j, e[v/x, (\text{fix } f(x).e)/f]) \in [\![|\sigma\tau_2'|]\!]_\varepsilon^\infty$.
By IH 5 on eq. (2) using the first premise, we get

$$(j, v) \in [\![|\sigma\tau_1|]\!]_v \tag{3}$$

By unrolling the definition of $(\star)$ with eq. (3) and eq. (1), we get

$$(j, e[v/x, (\text{fix } f(x).e)/f]) \in [\![|\sigma\tau_2|]\!]_\varepsilon^\infty \tag{4}$$

We can conclude by IH 6 on the second premise using eq. (4).

Case: $\dfrac{}{\Delta;\Phi \models \Box\,(\mathsf{U}\,(A_1 \xrightarrow{\mathbb{FS}(t)} A_2)) \sqsubseteq \Box\,\mathsf{U}\,A_1 \xrightarrow{\mathbb{CP}(0)} \Box\,\mathsf{U}\,A_2} \to \Box\mathsf{U}_{\textbf{cp}}$

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$.
We have $(m, \text{fix } f(x).e) \in [\![\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2]\!]_v$ $(\star)$.
STS: $(m, \text{fix } f(x).e) \in [\![\sigma A_1 \xrightarrow{\mathbb{FS}(\infty)} \sigma A_2]\!]_v$.
Assume that for some j

$$j < m \tag{1}$$

$$(j, v) \in [\![\sigma A_1]\!]_v \tag{2}$$

STS: $(j, e[v/x, (\text{fix } f(x).e)/f]) \in [\![\sigma A_2]\!]^{\infty}_{\varepsilon}$.
By unrolling $(\star)$'s definition with eq. (1) and eq. (2), we get

$$(j, e[v/x, (\text{fix } f(x).e)/f]) \in [\![\sigma A_2]\!]^{\sigma t}_{\varepsilon} \tag{3}$$

We can conclude by applying IH 4 to eq. (3) using $\sigma t \leqslant \infty$.

Case: $\dfrac{\Delta; \Phi \models n \doteq n' \qquad \Delta; \Phi \models \alpha \leqslant \alpha' \qquad \Delta; \Phi \models \tau \sqsubseteq \tau'}{\Delta; \Phi \models \mathbf{list}[n]^{\alpha}\,\tau \sqsubseteq \mathbf{list}[n']^{\alpha'}\,\tau'}\ \mathbf{l1}$

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$ and $\models \sigma\Phi$ and $(m, v) \in [\![\text{list}[\sigma n]\,|\sigma\tau|]\!]_v$.
TS: $(m, v) \in [\![\text{list}[\sigma n']\,|\sigma\tau'|]\!]_v$
From Assumption 25 applied to the first premise, $\sigma n = \sigma n'$. Therefore,
STS: $(m, v) \in (\!|\text{list}[\sigma n]\,|\sigma\tau'|)\!|_v$
We prove the following more general statement
$\forall\, m, v, n$. if $(m, v) \in [\![\text{list}[\sigma n]\,|\sigma\tau|]\!]_v$, then $(m, v) \in (\!|\text{list}[\sigma n]\,|\sigma\tau'|)\!|_v$.
We establish this statement by subinduction on v.

subcase 1: $v = \text{nil}$

We can immediately conclude that $(m, \text{nil}\,) \in (\!|\text{list}[0]\,|\sigma\tau'|)\!|_v$ by definition.

subcase 2: $v = \text{cons}(v_1, v_2)$

TS: $(m, \text{cons}(v_1, v_2)) \in (\!|\text{list}[I + 1]\,|\sigma\tau'|)\!|_v$ for some $I + 1 = \sigma n$.
By the main assumption, we have $(m, v_1) \in [\![|\sigma\tau|]\!]_v$ $(\diamond)$ and $(m, v_2) \in [\![\text{list}[|\sigma\tau|]\,]\!]_v$ $(\diamond\diamond)$.
By subIH on $(\diamond\diamond)$, we get

$$(m, v_2) \in [\![\text{list}[|\sigma\tau'|]^{[\![}\,]\!]_v \tag{1}$$

By IH 5 on ($\diamond$), we get

$$(m, v_1) \in [\![|\sigma\tau'|]\!]_v \tag{2}$$

Combining eq. (2) with eq. (1), we get $(m, \text{cons}(v_1, v_2)) \in [\![\text{list}[I + 1]\,|\sigma\tau'|]\!]_v$.

Case: $\dfrac{i :: S, \Delta; \Phi \models \tau \sqsubseteq \tau' \qquad i \notin \text{FV}(\Phi)}{\Delta; \Phi \models \exists i{::}S.\,\tau \sqsubseteq \exists i{::}S.\,\tau'}\ \exists$

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$.

We have

$$(m, \text{pack } v) \in [\![\exists|\sigma\tau|{::}S.\,]\!]_v \tag{1}$$

TS: $(m, \text{pack } v) \in [\![\exists|\sigma\tau'|{::}S.\,]\!]_v$.

By unrolling its definition, assume that $\vdash S :: (\star)$.

STS: $(m, v) \in [\![|\sigma\tau'|\{I/i\}]\!]_v$.

By unrolling eq. (1) with ($\star$), we get

$$(m, v) \in [\![|\sigma\tau|\{I/i\}]\!]_v \tag{2}$$

Then, we can conclude by IH 5 on eq. (2).

Case: $\dfrac{}{\Delta; \Phi \models \Box\,\tau \sqsubseteq \tau}$ **T**

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$.

We have $(m, v) \in [\![|\Box\,\sigma\tau|]\!]_v$.

TS: $(m, v) \in [\![|\sigma\tau|]\!]_v$.

Immediately follows since by definition of $|\cdot|$, we know that $|\Box\,\sigma\tau| = |\sigma\tau|$.

Case: $\dfrac{}{\Delta; \Phi \models \tau \sqsubseteq \mathsf{U}\,|\tau|}$ **W**

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$.

We have $(m, v) \in [\![|\sigma\tau|]\!]_v$.

TS: $(m, v) \in [\![|\mathsf{U}\,|\sigma\tau||]\!]_v$.

Immediately follows since by definition of $|\cdot|$, we know that $|\sigma\tau| = |\mathsf{U}\,|\sigma\tau||$.

□

Proof of statement (7). Remember that we are trying to prove:
If $\Delta;\Phi \models^{A} A \sqsubseteq A'$ and $\sigma \in \mathcal{D}[\![\Delta]\!]$ and $\models \sigma\Phi$ and $(m, w) \in (\!|\, \sigma A \,|\!)_v$, then $(m, w) \in (\!|\, \sigma A' \,|\!)_v$.

Proof is by induction on the subtyping derivation.

Case: $\dfrac{\Delta;\Phi \models^{A} A_1' \sqsubseteq A_1 \qquad \Delta;\Phi \models^{A} A_2 \sqsubseteq A_2' \qquad \Delta;\Phi \models t \leqslant t'}{\Delta;\Phi \models^{A} A_1 \xrightarrow{\mathbb{FS}(t)} A_2 \sqsubseteq A_1' \xrightarrow{\mathbb{FS}(t')} A_2'} \to \textbf{exec}$

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$.

We have

$$(m, \mathrm{fix}\ f(x).ee) \in (\!|\, \sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2 \,|\!)_v \tag{1}$$

TS: $(m, \mathrm{fix}\ f(x).ee) \in (\!|\sigma A_1' \xrightarrow{\mathbb{FS}(\sigma t')} \sigma A_2'|\!)_v$.

There are two cases to show.

subcase 1: Assume that $j < m$ and $(j, w) \in (\!|\mathsf{U}\,\sigma A_1'|\!)_v$ $(\star)$.
STS: $(j, e[w/x, (\mathrm{fix}\ f(x).ee)/f]) \in (\!|\mathsf{U}\,\sigma A_2'|\!)_\varepsilon^{\sigma t'}$.
By IH 8 on $(\star)$ using the first premise, we get

$$(j, w) \in (\!|\mathsf{U}\,\sigma A_1|\!)_v \tag{2}$$

By unrolling eq. (1) with eq. (2) using $j < m$, we get

$$(j, e[w/x, (\mathrm{fix}\ f(x).ee)/f]) \in (\!|\mathsf{U}\,\sigma A_2|\!)_\varepsilon^{\sigma t} \tag{3}$$

We conclude by applying IH 12 to eq. (3) using the second premise.

subcase 2: STS: $\forall j.(j, \mathrm{fix}\ f(x).\mathsf{L}(ee)) \in [\![\sigma A_1' \xrightarrow{\mathbb{FS}(\sigma t')} \sigma A_2']\!]_v \wedge (j, \mathrm{fix}\ f(x).\mathsf{R}(ee)) \in [\![\sigma A_1' \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2']\!]_v$.
Pick j.

We just show the left projection part, the right one is similar.
Pick j' and assume that

$$j' < j \tag{4}$$

$$(j', v) \in [\![|\sigma A_1'|]\!]_v \tag{5}$$

STS: $(j', L(\text{æ})[v/x, (\text{fix } f(x).L(\text{æ}))/f]) \in [\![|\sigma A_2'|]\!]^{\infty}_{\varepsilon}$.
By IH 5 on eq. (5) using the first premise, we get

$$(j', v) \in [\![|\sigma A_1|]\!]_v \tag{6}$$

By unrolling the second part of the definition of eq. (1), we get

$$\forall j.(j, \text{fix } f(x).L(\text{æ})) \in [\![|\sigma A_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma A_2|]\!]_v \tag{7}$$

Instantiating eq. (7) with $j' + 1$, we get

$$(j' + 1, \text{fix } f(x).L(\text{æ})) \in [\![|\sigma A_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma A_2|]\!]_v \tag{8}$$

Then unrolling the definition of eq. (8) with eq. (6) using $j' < j' + 1$, we get

$$(j', L(\text{æ})[v/x, (\text{fix } f(x).L(\text{æ}))/f]) \in [\![|\sigma A_2|]\!]^{\infty}_{\varepsilon} \tag{9}$$

We can conclude by IH 9 on the second premise using eq. (9).

□

Proof of statement (10). Remember that we are trying to prove:
If $\Delta; \Phi \models \tau \sqsubseteq \tau'$ and $\sigma \in \mathcal{D}[\![\Delta]\!]$ and $\models \sigma\Phi$ and $(m, w) \in (\!|\sigma\tau|\!)_v$, then $(m, w) \in (\!|\sigma\tau'|\!)_v$.

Proof is by induction on the subtyping derivation.

Case: $\dfrac{}{\Delta; \Phi \models \Box\, \mathsf{U}\, \mathbf{int} \sqsubseteq \mathbf{int}_r}\ \Box\textbf{U-int}$

We have $(m, w) \in \mathcal{C}\,|\Box\, \mathsf{U}\, \mathrm{int}|\,\mathcal{D}_\nu = \mathcal{C}\,\mathrm{int}\,\mathcal{D}_\nu$.

Unrolling its definition, we have $w = \mathsf{keep}(n)$.

TS: $(m, \mathsf{keep}(n)) \in \mathcal{C}\,|\mathrm{int}_r|\,\mathcal{D}_\nu = \mathcal{C}\,\mathrm{int}\,\mathcal{D}_\nu$.

Follows directly by definition.

Case: $\dfrac{\Delta; \Phi \models \tau_1' \sqsubseteq \tau_1 \qquad \Delta; \Phi \models \tau_2 \sqsubseteq \tau_2' \qquad \Delta; \Phi \models t \leqslant t'}{\Delta; \Phi \models \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2 \sqsubseteq \tau_1' \xrightarrow{\mathbb{CP}(t')} \tau_2'} \to_{\mathbf{cp}}$

Assume that $\sigma \in \mathcal{D}[\![\Delta]\!]$.

We have

$$(m, \mathrm{fix}\ f(x).e\!e) \in \mathcal{C}\,|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\,\mathcal{D}_\nu = \mathcal{C}\,|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|\,\mathcal{D}_\nu \quad (1)$$

TS: $(m, \mathrm{fix}\ f(x).e\!e) \in (\!|\sigma\tau_1' \xrightarrow{\mathbb{CP}(\sigma t')} \sigma\tau_2'|\!)_\nu = \mathcal{C}\,|\sigma\tau_1'| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2'|\,\mathcal{D}_\nu$.

There are two cases to show.

subcase 1: Assume that $j < m$ and $(j, w) \in (\!|\mathsf{U}\,|\sigma\tau_1'|\!|)_\nu$ $(\star)$.

STS: $(j, e[w/x, (\mathrm{fix}\ f(x).e\!e)/f]) \in (\!|\mathsf{U}\,|\sigma\tau_2'|\!|)_\varepsilon^{\sigma t'}$.

By IH 11 on $(\star)$ using the first premise, we get

$$(j, w) \in (\!|\mathsf{U}\,|\sigma\tau_1|\!|)_\nu \quad (2)$$

By unrolling eq. (1) with eq. (2) using $j < m$, we get

$$(j, e[w/x, (\mathrm{fix}\ f(x).e\!e)/f]) \in (\!|\mathsf{U}\,|\sigma\tau_2|\!|)_\varepsilon^{\sigma t} \quad (3)$$

We conclude by applying IH 12 to eq. (3) using the second premise.

subcase 2: STS: $\forall j.(j, \mathrm{fix}\ f(x).\mathsf{L}(e\!e)) \in [\![|\sigma\tau_1'| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2'|]\!]_\nu \wedge (j, \mathrm{fix}\ f(x).\mathsf{R}(e\!e)) \in [\![|\sigma\tau_1'| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2'|]\!]_\nu$.

Pick j.

We just show the left projection part, the right one is similar.
Pick j' and assume that

$$j' < j \tag{4}$$

$$(j', \nu) \in [\![|\sigma\tau_1'|]\!]_\nu \tag{5}$$

STS: $(j', L(\text{æ})[\nu/x, (\text{fix } f(x).L(\text{æ}))/f]) \in [\![|\sigma\tau_2'|]\!]^\infty_\varepsilon$.
By IH 5 on eq. (5) using the first premise, we get

$$(j', \nu) \in [\![|\sigma\tau_1|]\!]_\nu \tag{6}$$

By unrolling the second part of the definition of eq. (1), we get

$$\forall j.(j, \text{fix } f(x).L(\text{æ})) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu \tag{7}$$

Instantiating eq. (7) with $j' + 1$, we get

$$(j' + 1, \text{fix } f(x).L(\text{æ})) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu \tag{8}$$

Then unrolling the definition of eq. (8) with eq. (6) using $j' < j' + 1$, we get

$$(j', L(\text{æ})[\nu/x, (\text{fix } f(x).L(\text{æ}))/f]) \in [\![|\sigma\tau_2|]\!]^\infty_\varepsilon \tag{9}$$

We can conclude by IH 6 on the second premise using eq. (9).

Case: $\dfrac{}{\Delta; \Phi \models \tau \sqsubseteq \mathcal{U}\,|\tau|}\ \mathbf{W}$
We have $(m, w) \in \mathcal{C}\,|\sigma\tau|\,\mathcal{D}_\nu$.
TS: $(m, w) \in \mathcal{C}\,|\mathcal{U}\,|\sigma\tau||\,\mathcal{D}_\nu$.

Immediately follows since by definition of $|\cdot|$, we know that $|\sigma\tau| = |\mathsf{U}\,|\sigma\tau||$.

Case: $\dfrac{}{\Delta;\Phi \models \Box\,(\mathsf{U}\,(A_1 \xrightarrow{\mathbb{FS}(t)} A_2)) \sqsubseteq \Box\,\mathsf{U}\,A_1 \xrightarrow{\mathbb{CP}(0)} \Box\,\mathsf{U}\,A_2} \to \Box\mathsf{U}_{\mathbf{cp}}$

We have $(m, \text{fix } f(x).ee) \in \mathbb{C}\,|\Box\,\mathsf{U}\,(\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2)|\,\mathbb{D}_v = \mathbb{C}\,\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2\,\mathbb{D}_v$ $(\star)$.

STS: $(m, \text{fix } f(x).ee) \in \mathbb{C}\,|\Box\,\mathsf{U}\,\sigma A_1 \xrightarrow{\mathbb{CP}(0)} \Box\,\mathsf{U}\,\sigma A_2|\,\mathbb{D}_v = \mathbb{C}\,\sigma A_1 \xrightarrow{\mathbb{FS}(\infty)} \sigma A_2\,\mathbb{D}_v$.

Assume that for some j

$$j < m \tag{1}$$

$$(j, w) \in (\!|\mathsf{U}\,\sigma A_1|\!)_v \tag{2}$$

STS: $(j, ee[w/x, (\text{fix } f(x).ee)/f]) \in (\!|\mathsf{U}\,\sigma A_2|\!)^{\infty}_{\varepsilon}$.

By unrolling $(\star)$'s definition with eq. (1) and eq. (2), we get

$$(j, e[v/x, (\text{fix } f(x).e)/f]) \in (\!|\mathsf{U}\,\sigma A_2|\!)^{\sigma t}_{\varepsilon} \tag{3}$$

We can conclude by applying IH 3 to eq. (3) using $\sigma t \leqslant \infty$.

□

Assumption 40 (Constraint Well-formedness). *If* $\Delta;\Phi \models C$ *then* $\Delta \vdash C$ `wf`

Lemma 41 (Refinement Removal Well-formedness). *If* $\Phi;\Delta \vdash \tau$ `wf`, *then* $\Phi;\Delta \vdash^{A} |\tau|$ `wf`.

Lemma 42 (Subtyping well-formedness). *The following hold.*

- *If* $\Delta;\Phi \models \tau \sqsubseteq \tau'$ *and* $\Delta;\Phi \vdash \tau$ `wf` *and* $\mathrm{FIV}(\tau) \subseteq \Delta$, *then* $\Phi;\Delta \vdash \tau'$ `wf` *and* $\mathrm{FIV}(\tau') \subseteq \Delta$.

- *If* $\Delta; \Phi \models^{A} A \sqsubseteq A'$ *and* $\Delta; \Phi \vdash^{A} A\ \mathtt{wf}$ *and* $\mathrm{FIV}(A) \subseteq \Delta$, *then* $\Phi; \Delta \vdash A'\ \mathtt{wf}$ *and* $\mathrm{FIV}(A') \subseteq \Delta$.

Proof. The proof is by induction on the subtyping derivations. □

Lemma 43 (Well-formedness)**.** *The following hold.*

1. *If* $\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t$ *and* $\Delta; \Phi \vdash \Gamma\ \mathtt{wf}$ *and* $FIV(\Gamma) \subseteq dom(\Delta)$, *then* $\Phi; \Delta \vdash \tau\ \mathtt{wf}$ *and* $FIV(t, \tau) \subseteq dom(\Delta)$.
2. *If* $\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A \mid t$ *and* $\Delta; \Phi \vdash^{A} \Omega\ \mathtt{wf}$ *and* $FIV(\Omega) \subseteq dom(\Delta)$, *then* $\Phi; \Delta \vdash^{A} A\ \mathtt{wf}$ *and* $FIV(t, A) \subseteq dom(\Delta)$.
3. *If* $\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t$, *then* $FV(e) \subseteq dom(\Gamma)$.
4. *If* $\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A \mid t$, *then* $FV(e) \subseteq dom(\Omega)$.

Proof. The proof is by induction on the typing derivations. □

Both of our fundamental theorems rely on the assumption that the semantic interpretation of every primitive function lies in the interpretation of the function's type. This is explained below.

Assumption 44 (Soundness of primitive functions (relational))**.** *Suppose that* $\zeta : \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2$ *and* $(m, w) \in (\!|\tau_1|\!)_\nu$ *and* $\hat{\zeta}\ L(w) = (\nu_r, f_r)$ *and* $\hat{\zeta}\ R(w) = (\nu'_r, f'_r)$, *then*

- $f'_r \leqslant t$
- $(m - f_r, \mathtt{merge}(\nu_r, \nu'_r)) \in (\!|\tau_2|\!)_\nu$

Assumption 45 (Soundness of primitive functions (non-relational))**.** *Suppose that* $\zeta : A_1 \xrightarrow{\mathbb{FS}(t)} A_2$ *and* $(m, \nu) \in [\![A_1]\!]_\nu$ *and* $\hat{\zeta}\ \nu = (\nu_r, f_r)$, *then*

- $f_r \leqslant t$
- $(m - f_r, \nu_r) \in [\![A_2]\!]_\nu$

B.2 DUCOSTIT THEOREMS

Theorem 46 (Fundamental theorem)**.** *The following holds.*

1. *Assume that* $\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$*. Then,* $(m, \delta^{\ulcorner} e^{\urcorner}) \in (\!|\sigma\tau|\!)^{\sigma t}_{\varepsilon}$.

2. *Assume that* $\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A \mid t$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, \gamma) \in \mathcal{G}[\![\sigma\Omega]\!]$*. Then,* $(m, \gamma e) \in [\![\sigma A]\!]^{\sigma t}_{\varepsilon}$.

3. *Assume that* $\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, \gamma) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$*, then* $(m, \gamma e) \in [\![|\sigma\tau|]\!]^{\infty}_{\varepsilon}$.

4. *Assume that* $\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A \mid t$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, \delta) \in \mathcal{G}(\!|\mathcal{U}\ \sigma\Omega|\!)$*, then* $(m, \delta^{\ulcorner} e^{\urcorner}) \in (\!|\mathcal{U}\ \sigma A|\!)^{\sigma t}_{\varepsilon}$.

5. *Assume that* $\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, \delta) \in \mathcal{G}(\!|\mathcal{U}\ |\sigma\Gamma|\!)$*, then* $(m, \delta^{\ulcorner} e^{\urcorner}) \in (\!|\mathcal{U}\ |\sigma\tau|\!)^{\infty}_{\varepsilon}$.

Proof. Proofs are by induction on typing derivations with a sub-induction on step-indices for recursive functions. We show select cases of each statement separately.

Proof of Statement (1). We proceed by induction on the typing derivation. We show the most important cases below.

Case: $\dfrac{\Gamma(x) = \tau}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} x : \tau \mid 0}$ **cp-var**

Assume that $\models \sigma\Phi$ and $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$.
TS: $(m, \delta(\ulcorner x \urcorner)) \in (\!|\sigma\tau|\!)^{0}_{\varepsilon}$.
By Value Lemma (lemma 31), STS: $(m, \delta(x)) \in (\!|\sigma\tau|\!)_{v}$.
This follows by $\Gamma(x) = \tau$ and $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$.

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_1 : \tau \mid t_1 \qquad \Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_2 : \mathbf{list}[n]^{\alpha}\,\tau \mid t_2}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \mathbf{cons}(e_1, e_2) : \mathbf{list}[n+1]^{\alpha+1}\,\tau \mid t_1 + t_2}$ **cp-cons1**

Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.
TS: $(m, \mathrm{cons}(\delta^{\ulcorner} e_1 {}^{\urcorner}, \delta^{\ulcorner} e_2 {}^{\urcorner})) \in (\!|\mathrm{list}[\sigma n + 1]^{\sigma\alpha+1}\ \sigma\tau|\!)^{\sigma t_1 + \sigma t_2}_{\varepsilon}$.

Following the definition of $(\!|\cdot|\!)^{\cdot}_{\varepsilon}$, assume that

$$\frac{L(\delta^{\ulcorner} e_1^{\urcorner}) \Downarrow^{f_1} T_1 \quad (\star) \qquad L(\delta^{\ulcorner} e_2^{\urcorner}) \Downarrow^{f_2} T_2 \quad (\diamond) \qquad v_i = V(T_i)}{\mathsf{cons}(L(\delta^{\ulcorner} e_1^{\urcorner}), L(\delta^{\ulcorner} e_2^{\urcorner})) \Downarrow^{f_1+f_2} \langle \mathsf{cons}(v_1, v_2), \mathsf{cons}(T_1, T_2)\rangle}\ \textbf{ev-cons}$$

$$\frac{R(\delta^{\ulcorner} e_1^{\urcorner}) \Downarrow^{f'_1} T'_1 \quad (\star\star) \qquad R(\delta^{\ulcorner} e_2^{\urcorner}) \Downarrow^{f'_2} T'_2 \quad (\diamond\diamond) \qquad v'_i = V(T'_i)}{\mathsf{cons}(R(\delta^{\ulcorner} e_1^{\urcorner}), R(\delta^{\ulcorner} e_2^{\urcorner})) \Downarrow^{f'_1+f'_2} \langle \mathsf{cons}(v'_1, v'_2), \mathsf{cons}(T'_1, T'_2)\rangle}\ \textbf{ev-cons}$$

and $f_1 + f_2 < m$.

By IH 1 on the first premise, we get $(m, \delta^{\ulcorner} e_1^{\urcorner}) \in (\!|\sigma\tau|\!)^{\sigma t_1}_{\varepsilon}$. Unrolling its definition with $(\star)$ and $f_1 < m$, we get

a) $\langle T_1, \delta^{\ulcorner} e_1^{\urcorner}\rangle \curvearrowright w'_1, T'_1, c'_1$
b) $v_1 = L(w'_1) \ \wedge\ v'_1 = R(w'_1)$
c) $c'_1 \leqslant \sigma t_1$
d) $(m - f_1, w'_1) \in (\!|\sigma\tau|\!)_v$

By IH 1 on the second premise, we get $(m, \delta^{\ulcorner} e_2^{\urcorner}) \in (\!|\mathrm{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|\!)^{\sigma t_2}_{\varepsilon}$. Unrolling its definition with $(\diamond)$ and $f_2 < m$, we get

e) $\langle T_2, \delta^{\ulcorner} e_2^{\urcorner}\rangle \curvearrowright w'_2, T'_2, c'_2$
f) $v_2 = L(w'_2) \ \wedge\ v'_2 = R(w'_2)$
g) $c'_2 \leqslant \sigma t_2$
h) $(m - f_2, w'_2) \in (\!|\mathrm{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|\!)_v$

Now, we can conclude as follows:

1. Using a) and e)
$$\frac{\langle T_1, R(\delta^{\ulcorner} e_1^{\urcorner})\rangle \curvearrowright w'_1, T'_1, c'_1 \qquad \langle T_2, e e_2\rangle \curvearrowright w'_2, T'_2, c'_2 \qquad v'_i = V(T'_i)}{\langle \langle \_, \mathsf{cons}(T_1, T_2)\rangle, \mathsf{cons}(R(\delta^{\ulcorner} e_1^{\urcorner}), R(\delta^{\ulcorner} e_2^{\urcorner}))\rangle \curvearrowright \mathsf{cons}(w'_1, w'_2), \langle \mathsf{cons}(v'_1, v'_2), \mathsf{cons}(T'_1, T'_2)\rangle, c'_1 + c'_2}\ \textbf{cp-cons}$$
2. Using b) and f),$\mathsf{cons}(v_1, v_2) = L(\mathsf{cons}(w'_1, w'_2)) \ \wedge\ \mathsf{cons}(v'_1, v'_2) = R(\mathsf{cons}(w'_1, w'_2))$
3. By using c) and g), we get $c'_1 + c'_2 \leqslant \sigma t_1 + \sigma t_2$

4. By downward closure (Lemma 33) on d) and h) using

$$m - (f_1 + f_2) \leqslant m - f_1$$

$$m - (f_1 + f_2) \leqslant m - f_2$$

we get $(m-(f_1+f_2), w_1') \in (\!|\sigma\tau|\!)_v$ and $(m-(f_1+f_2), w_2') \in (\!|\mathsf{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|\!)_v$, when combined, gives us $(m-(f_1+f_2), \mathsf{cons}(w_1', w_2')) \in (\!|\mathsf{list}[\sigma n + 1]^{\sigma\alpha+1}\ \sigma\tau|\!)_v$

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CIP}} e_1 : \Box\tau \mid t_1 \qquad \Delta; \Phi_a; \Gamma \vdash_{\mathbb{CIP}} e_2 : \mathbf{list}[n]^\alpha \tau \mid t_2}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CIP}} \mathbf{cons}(e_1, e_2) : \mathbf{list}[n+1]^\alpha \tau \mid t_1 + t_2}$ **cp-cons2**

Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m, \mathsf{cons}(\delta^\ulcorner e_1 \urcorner, \delta^\ulcorner e_2 \urcorner)) \in (\!|\mathsf{list}[\sigma n + 1]^{\sigma\alpha+1}\ \sigma\tau|\!)_\varepsilon^{\sigma t_1 + \sigma t_2}$.

Following the definition of $(\!|\cdot|\!)_\varepsilon^{\cdot}$, assume that

$$\frac{L(\delta^\ulcorner e_1 \urcorner) \Downarrow^{f_1} T_1 \ \ (\star) \qquad L(\delta^\ulcorner e_2 \urcorner) \Downarrow^{f_2} T_2 \ \ (\diamond) \qquad v_i = V(T_i)}{\mathsf{cons}(L(\delta^\ulcorner e_1 \urcorner), L(\delta^\ulcorner e_2 \urcorner)) \Downarrow^{f_1+f_2} \langle \mathsf{cons}(v_1, v_2), \mathsf{cons}(T_1, T_2)\rangle}\ \textbf{ev-cons}$$

$$\frac{R(\delta^\ulcorner e_1 \urcorner) \Downarrow^{f_1'} T_1' \ \ (\star) \qquad R(\delta^\ulcorner e_2 \urcorner) \Downarrow^{f_2'} T_2' \ \ (\diamond) \qquad v_i' = V(T_i')}{\mathsf{cons}(R(\delta^\ulcorner e_1 \urcorner), R(\delta^\ulcorner e_2 \urcorner)) \Downarrow^{f_1'+f_2'} \langle \mathsf{cons}(v_1', v_2'), \mathsf{cons}(T_1', T_2')\rangle}\ \textbf{ev-cons}$$

and $f_1 + f_2 < m$.

By IH 1 on the first premise, we get $(m, \delta^\ulcorner e_1 \urcorner) \in (\!|\sigma\tau|\!)_\varepsilon^{\sigma t_1}$. Unrolling its definition with $(\star)$ and $f_1 < m$, we get

a) $\langle T_1, \delta^\ulcorner e_1 \urcorner\rangle \curvearrowright w_1', T_1', c_1'$
b) $v_1 = L(w_1') \ \wedge\ v_1' = R(w_1')$
c) $c_1' \leqslant \sigma t_1$
d) $(m - f_1, w_1') \in (\!|\Box\,\sigma\tau|\!)_v$

By IH 1 on the second premise, we get $(m, \delta^\ulcorner e_2 \urcorner) \in (\!|\mathsf{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|\!)_\varepsilon^{\sigma t_2}$. Unrolling its definition with $(\diamond)$ and $f_2 < m$, we get

e) $\langle T_2, \delta^\ulcorner e_2 \urcorner\rangle \curvearrowright w_2', T_2', c_2'$
f) $v_2 = L(w_2') \ \wedge\ v_2' = R(w_2')$
g) $c_2' \leqslant \sigma t_2$

h) $(m - f_2, w_2') \in (\!|\text{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|\!)_v$

Now, we can conclude as follows:

1. Using a) and e)
$$\frac{\begin{array}{c}\langle T_1, R(\delta^\ulcorner e_1 {}^\urcorner)\rangle \curvearrowright w_1', T_1', c_1' \\ \langle T_2, ee_2\rangle \curvearrowright w_2', T_2', c_2' \qquad v_i' = V(T_i')\end{array}}{\begin{array}{l}\langle\langle \_, \text{cons}(T_1, T_2)\rangle, \text{cons}(R(\delta^\ulcorner e_1 {}^\urcorner), R(\delta^\ulcorner e_2 {}^\urcorner))\rangle \curvearrowright \\ \text{cons}(w_1', w_2'), \langle \text{cons}(v_1', v_2'), \text{cons}(T_1', T_2')\rangle, c_1' + c_2'\end{array}}\ \textbf{cp-cons}$$
2. Using b) and f),$\text{cons}(v_1, v_2) = L(\text{cons}(w_1', w_2')) \wedge \text{cons}(v_1', v_2') = R(\text{cons}(w_1', w_2'))$
3. By using c) and g), we get $c_1' + c_2' \leqslant \sigma t_1 + \sigma t_2$
4. By downward closure (Lemma 33) on d) and h) using

$$m - (f_1 + f_2) \leqslant m - f_1$$

$$m - (f_1 + f_2) \leqslant m - f_2$$

we get $(m - (f_1 + f_2), w_1') \in (\!|\Box\, \sigma\tau|\!)_v$ and $(m - (f_1 + f_2), w_2') \in (\!|\text{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|\!)_v$, when combined, gives us $(m - (f_1 + f_2), \text{cons}(w_1', w_2')) \in (\!|\text{list}[\sigma n + 1]^{\sigma\alpha}\ \sigma\tau|\!)_v$

Case:
$$\frac{\begin{array}{c}\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CIP}} e : \textbf{list}[n]^\alpha\, \tau \mid t \qquad \Delta; \Phi \wedge n = 0; \Gamma \vdash_{\mathbb{CIP}} e_1 : \tau' \mid t' \\ i, \Delta; \Phi \wedge n = i + 1; h : \Box\, \tau, tl : \textbf{list}[i]^\alpha\, \tau, \Gamma \vdash_{\mathbb{CIP}} e_2 : \tau' \mid t' \\ i, \beta, \Delta; \Phi \wedge n = i + 1 \wedge \alpha = \beta + 1; h : \tau, tl : \textbf{list}[i]^\beta\, \tau, \Gamma \vdash_{\mathbb{CIP}} e_2 : \tau' \mid t'\end{array}}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CIP}} \textbf{case}\ e\ \textbf{of nil}\ \to e_1 \mid h :: tl \to e_2 : \tau' \mid t + t'}\ \textbf{cp-caseL}$$

Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m,\ \text{case}\ \delta^\ulcorner e^\urcorner\ \text{of nil}\ \to \delta^\ulcorner e_1 {}^\urcorner \mid h :: tl \to \delta^\ulcorner e_2 {}^\urcorner) \in (\!|\sigma\tau'|\!)_\varepsilon^{\sigma t + \sigma t'}$.

Following the definition of $(\!|\cdot|\!)_\varepsilon^{\cdot}$, assume that

$L(\ \text{case}\ \delta^\ulcorner e^\urcorner\ \text{of nil}\ \to \delta^\ulcorner e_1 {}^\urcorner \mid h :: tl \to \delta^\ulcorner e_2 {}^\urcorner) \Downarrow^F T$

$R(\ \text{case}\ \delta^\ulcorner e^\urcorner\ \text{of nil}\ \to \delta^\ulcorner e_1 {}^\urcorner \mid h :: tl \to \delta^\ulcorner e_2 {}^\urcorner) \Downarrow^{F'} T'$ and $F < m$.

Depending on what $L(\delta^\ulcorner e^\urcorner)$ and $R(\delta^\ulcorner e^\urcorner)$ evaluate to, there are four cases.

subcase 1:

$$\frac{\mathrm{L}(\delta\ulcorner e\urcorner) \Downarrow^{f} T \quad (\star) \qquad \mathrm{L}(\delta\ulcorner e_1\urcorner) \Downarrow^{f_r} T_1 \quad (\diamond) \qquad \mathsf{nil} = \mathsf{V}(T) \qquad v_r = \mathsf{V}(T_r)}{\mathrm{case}\ \mathrm{L}(\delta\ulcorner e\urcorner)\ \mathrm{of}\ \mathsf{nil} \to \mathrm{L}(\delta\ulcorner e_1\urcorner) \mid h :: tl \to \mathrm{L}(\delta\ulcorner e_2\urcorner) \Downarrow^{f+f_r+c_{caseL}} \langle v_r, \mathsf{case}_{\mathsf{nil}}(T, T_r)}$$

and

$$\frac{\mathrm{R}(\delta\ulcorner e\urcorner) \Downarrow^{f'} T' \quad (\star\star) \qquad \mathrm{R}(\delta\ulcorner e_1\urcorner) \Downarrow^{f'_r} T'_1 \quad (\diamond\diamond) \qquad \mathsf{nil} = \mathsf{V}(T') \qquad v'_r = \mathsf{V}(T'_r)}{\mathrm{case}\ \mathrm{R}(\delta\ulcorner e\urcorner)\ \mathrm{of}\ \mathsf{nil} \to \mathrm{R}(\delta\ulcorner e_1\urcorner) \mid h :: tl \to \mathrm{R}(\delta\ulcorner e_2\urcorner) \Downarrow^{f'+f'_r+c_{caseL}} \langle v'_r, \mathsf{case}_{\mathsf{nil}}(T', T}$$

and $F = f + f_r + c_{caseL} < m$.

By IH 1 on the first premise, we get $(m, \delta\ulcorner e\urcorner) \in (\!|\mathrm{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|\!)^{\sigma t}_{\varepsilon}$.

Unrolling its definition with $(\star)$, $(\star\star)$ and $f < m$, we get

a) $\langle T, \delta\ulcorner e\urcorner\rangle \curvearrowright w', T', c'$

b) $\mathsf{nil} = \mathrm{L}(w') \ \wedge\ v' = \mathrm{R}(w')$

c) $c' \leqslant \sigma t$

d) $(m - f, w') \in (\!|\mathrm{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|\!)_{v}$

By (b) and (d), we know that $w' = \mathsf{nil}$ and $\sigma n = 0$.

Then, we can instantiate IH 1 on the second premise using $\models \sigma\Phi \wedge \sigma n \doteq 0$, to obtain $(m, \delta\ulcorner e_1\urcorner) \in (\!|\sigma\tau'|\!)^{\sigma t'}_{\varepsilon}$.

Unrolling its definition using $(\diamond)$ and $f_r < m$, we get

e) $\langle T_1, \delta\ulcorner e_1\urcorner\rangle \curvearrowright w'_r, T'_r, c'_r$

f) $v_r = \mathrm{L}(w'_r) \ \wedge\ v'_r = \mathrm{R}(w'_r)$

g) $c'_r \leqslant \sigma t'$

h) $(m - f_r, w'_r) \in (\!|\sigma\tau'|\!)_{v}$

We conclude with

1. Using a) and e)

$$\frac{\langle T, \delta\ulcorner e\urcorner\rangle \curvearrowright \mathsf{nil}, T', c' \qquad \langle T_r, \delta\ulcorner e_1\urcorner\rangle \curvearrowright w'_r, T'_r, c'_r \qquad v'_r = \mathsf{V}(T'_r)}{\mathsf{case}_{\mathrm{L}}\,\delta\ulcorner e\urcorner\ \mathrm{of}\ \mathsf{nil} \to \langle\langle \_, \mathsf{case}_{\mathsf{nil}}(T, T_r)\rangle, \delta\ulcorner e_1\urcorner \mid h :: tl \to \delta\ulcorner e_2\urcorner\rangle \curvearrowright w'_r, \langle v'_r, \mathsf{case}_{\mathsf{nil}}(T', T'_r)\rangle, c' + c'_r}\ \mathbf{c}$$

2. Using f)

3. By using c) and g), we get $c' + c'_r \leqslant \sigma t + \sigma t'$
4. By downward closure (Lemma 33) on h) using

$$m - (f + f_r + c_{caseL}) \leqslant m - f_r$$

we get $(m - (f + f_r + c_{caseL}), w'_r) \in (\!|\sigma\tau'|\!)_v$.

subcase 2:

$$\frac{\begin{array}{c} L(\delta^\ulcorner e^\urcorner) \Downarrow^f T \quad (\star) \qquad cons(v_h, v_{tl}) = V(T) \\ L(\delta^\ulcorner e_2^\urcorner)[v_h/h, v_{tl}/tl] \Downarrow^{f_r} T_r \quad (\diamond) \qquad v_r = V(T_r) \end{array}}{\text{case } L(\delta^\ulcorner e^\urcorner) \text{ of nil } \to L(\delta^\ulcorner e_1^\urcorner) \mid h :: tl \to L(\delta^\ulcorner e_2^\urcorner) \Downarrow^{f+f_r+c_{caseL}} \langle v_r, \mathsf{case}_{cons}(T, T_r)\rangle}\ \textbf{ev}$$

and

$$\frac{\begin{array}{c} R(\delta^\ulcorner e^\urcorner) \Downarrow^{f'} T' \quad (\star\star) \qquad cons(v'_h, v'_{tl}) = V(T') \\ R(\delta^\ulcorner e_2^\urcorner)[v'_h/h, v'_{tl}/tl] \Downarrow^{f'_r} T'_r \quad (\diamond\diamond) \qquad v'_r = V(T'_r) \end{array}}{\text{case } R(\delta^\ulcorner e^\urcorner) \text{ of nil } \to R(\delta^\ulcorner e_1^\urcorner) \mid h :: tl \to R(\delta^\ulcorner e_2^\urcorner) \Downarrow^{f'+f'_r+c_{caseL}} \langle v'_r, \mathsf{case}_{cons}(T', T'_r)\rangle}\ \textbf{e}$$

and $F = f + f_r + c_{caseL} < m$.

By IH 1 on the first premise, we get $(m, \delta^\ulcorner e^\urcorner) \in (\!|\text{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|\!)^{\sigma t}_{\varepsilon}$.
Unrolling its definition with $(\star)$, $(\star\star)$ and $f < m$, we get

a) $\langle T, \delta^\ulcorner e^\urcorner\rangle \curvearrowright w', T', c'$
b) $cons(v_h, v_{tl}) = L(w') \ \wedge\ cons(v'_h, v'_{tl}) = R(w')$
c) $c' \leqslant \sigma t$
d) $(m - f, w') \in (\!|\text{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|\!)_v$

By b) and d), we know that $w' = \mathsf{cons}(w_h, w_{tl})$
For d), there are two cases:

subsubcase 1: $\sigma n = I + 1$ such that we have

$$(m - f, w_h) \in (\!|\square\ \sigma\tau|\!)_v \tag{1}$$

$$(m - f, w_{tl}) \in (\!|\text{list}[I]^{\sigma\alpha}\ \sigma\tau|\!)_v \tag{2}$$

In addition, by downward closure (Lemma 33) on $(m, \delta) \in \mathcal{G}(\!|\Gamma|\!)$, we have

$$(m - f, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!) \tag{3}$$

Then, we can instantiate IH 1 on the third premise using

- $\sigma[i \mapsto I] \in \mathcal{D}[\![i :: \mathbb{N}, \Delta]\!]$
- $\models \sigma[i \mapsto I](\Phi \wedge n \doteq i + 1)$ obtained by
  - $\models \sigma\Phi$ by main assumption
  - $\models \sigma n \doteq I + 1$ by sub-assumption
- $(m - f, \delta[h \mapsto w_h, tl \mapsto w_{tl}]) \in \mathcal{G}(\!|\sigma[i \mapsto I](\Gamma, x : \square\tau, tl : \mathsf{list}[i]^{\alpha}\,\tau)|\!)$ using (1) and (2) and (3).

we get $(m - f, \delta^{\ulcorner} e_2 {}^{\urcorner}[w_h/h, w_{tl}/tl]) \in (\!|\sigma[i \mapsto I]\tau'|\!)_{\varepsilon}^{\sigma[i \mapsto I]t'}$.
Since, $i \notin FV(t', \tau, \tau')$, we have
$(m - f, \delta^{\ulcorner} e_2 {}^{\urcorner}[w_h/h, w_{tl}/tl]) \in (\!|\sigma\tau'|\!)_{\varepsilon}^{\sigma t'}$.
Unrolling its definition using ($\diamond$), ($\diamond\diamond$) and $f_r < m - f$, we get

e) $\langle T_r, \delta^{\ulcorner} e_2 {}^{\urcorner}[w_h/h, w_{tl}/tl]\rangle \curvearrowright w'_r, T'_r, c'_r$
f) $v_r = L(w'_r) \ \wedge \ v'_r = R(w'_r)$
g) $c'_r \leqslant \sigma t'$
h) $(m - f - f_r, w'_r) \in (\!|\sigma\tau'|\!)_v$

We conclude with

1. Using a) and e)

$$\frac{\begin{array}{c}\langle T, \delta^{\ulcorner} e^{\urcorner}\rangle \curvearrowright \mathsf{cons}(w_h, w_{tl}), T', c' \\ \langle T_r, \delta^{\ulcorner} e_2 {}^{\urcorner}[w_h/h, w_{tl}/tl]\rangle \curvearrowright \\ w'_r, T'_r, c'_r \qquad v'_r = V(T'_r)\end{array}}{\begin{array}{l}\mathsf{case}_L\, \delta^{\ulcorner} e^{\urcorner} \text{ of } \mathsf{nil} \ \to \\ \langle\langle \_, \mathsf{case}_{\mathsf{cons}}(T, T_r)\rangle, \delta^{\ulcorner} e_1 {}^{\urcorner} \qquad\qquad\qquad \rangle \curvearrowright \\ \qquad\qquad\qquad\qquad\qquad\qquad | \ h :: tl \to \delta^{\ulcorner} e_2 {}^{\urcorner} \\ w'_r, \langle v'_r, \mathsf{case}_{\mathsf{cons}}(T', T'_r)\rangle, c' + c'_r\end{array}}\ \textbf{cp-caseL-cons}$$

2. Using g)

3. By using c) and f), we get $c' + c'_r \leqslant \sigma t + \sigma t'$
4. By downward closure (Lemma 33) on h) using

$$m - (f + f_r + c_{caseL}) \leqslant m - f - f_r$$

we get $(m - (f + f_r + c_{caseL}), w'_r) \in (\!|\sigma\tau'|\!)_v$.

subsubcase 2: $\sigma n = I + 1$ and $\sigma\alpha = J + 1$ such that we have

$$(m - f, w_h) \in (\!|\sigma\tau|\!)_v \tag{4}$$

$$(m - f, w_{tl}) \in (\!|\mathrm{list}[I]^J\, \sigma\tau|\!)_v \tag{5}$$

In addition, by downward closure (Lemma 33) on $(m, \delta) \in \mathcal{G}(\!|\Gamma|\!)$, we have

$$(m - f, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!) \tag{6}$$

Then, we can instantiate IH 1 on the fourth premise using

- $\sigma[i \mapsto I, \beta \mapsto J] \in \mathcal{D}[\![i :: \mathbb{N}, \beta :: \mathbb{N}, \Delta]\!]$
- $\models \sigma[i \mapsto I, \beta \mapsto J](\Phi \wedge n \doteq i + 1 \wedge \alpha \doteq \beta + 1)$ obtained
  - $\models \sigma\Phi$ by main assumption
  - $\models \sigma n \doteq I + 1$ by sub-assumption
  - $\models \sigma\alpha \doteq J + 1$ by sub-assumption
- $(m - f, \delta[h \mapsto w_h, tl \mapsto w_{tl}]) \in \mathcal{G}(\!|\sigma[i \mapsto I, \beta \mapsto J](\Gamma, x : \tau, tl : \mathrm{list}[i]^\beta\, \tau)|\!)$ using (4) and (5) and (6)

we get
$(m - f, \delta^{\ulcorner} e_2 {}^{\urcorner}[w_h/h, w_{tl}/tl]) \in (\!|\sigma[i \mapsto I, \beta \mapsto J]\tau'|\!)_\varepsilon^{\sigma[i \mapsto I, \beta \mapsto J]t'}$.
Since, $i, \beta \notin \mathrm{FV}(t', \tau, \tau')$, we have
$(m - f, \delta^{\ulcorner} e_2 {}^{\urcorner}[w_h/h, w_{tl}/tl]) \in (\!|\sigma\tau'|\!)_\varepsilon^{\sigma t'}$.

Unrolling its definition using ($\diamond$), ($\diamond\diamond$) and $f_r < m - f$, we get

i) $\langle T_r, \delta^\ulcorner e_2 \urcorner[w_h/h, w_{tl}/tl]\rangle \curvearrowright w'_r, T'_r, c'_r$
j) $v_r = L(w'_r) \ \wedge \ v'_r = R(w'_r)$
k) $c'_r \leqslant \sigma t'$
l) $(m - f - f_r, w'_r) \in (\!|\sigma\tau'|\!)_v$

We conclude with

1. Using a) and e)

$$\frac{\begin{array}{c}\langle T, \delta^\ulcorner e^\urcorner\rangle \curvearrowright \mathtt{cons}(w_h, w_{tl}), T', c' \\ \langle T_r, \delta^\ulcorner e_2 \urcorner[w_h/h, w_{tl}/tl]\rangle \curvearrowright \\ w'_r, T'_r, c'_r \qquad v'_r = \mathtt{V}(T'_r)\end{array}}{\begin{array}{l}\mathtt{case_L}\,\delta^\ulcorner e^\urcorner \text{ of } \mathtt{nil} \ \to \\ \langle\langle \_, \mathtt{case_{cons}}(T, T_r)\rangle, \delta^\ulcorner e_1 \urcorner \qquad\qquad \rangle \curvearrowright \\ \qquad\qquad\qquad\qquad \mid h :: tl \to \delta^\ulcorner e_2 \urcorner \\ w'_r, \langle v'_r, \mathtt{case_{cons}}(T', T'_r)\rangle, c' + c'_r\end{array}}\ \textbf{cp-caseL-cons}$$

2. Using g)
3. By using d) and f), we get $c' + c'_r \leqslant \sigma t + \sigma t'$
4. By downward closure (Lemma 33) on h) using

$$m - (f + f_r + c_{caseL}) \leqslant m - f - f_r$$

we get $(m - (f + f_r + c_{caseL}), w'_r) \in (\!|\sigma\tau'|\!)_v$.

subcase 3:

$$\frac{\begin{array}{c}L(\delta^\ulcorner e^\urcorner) \Downarrow^f T \ \ (\star) \qquad \mathrm{cons}(v_h, v_{tl}) = \mathtt{V}(T) \\ L(\delta^\ulcorner e_2 \urcorner)[v_h/h, v_{tl}/tl] \Downarrow^{f_r} T_r \ \ (\diamond) \qquad v_r = \mathtt{V}(T_r)\end{array}}{\text{case } L(\delta^\ulcorner e^\urcorner) \text{ of nil } \to L(\delta^\ulcorner e_1 \urcorner) \mid h :: tl \to L(\delta^\ulcorner e_2 \urcorner) \Downarrow^{f+f_r+c_{caseL}} \langle v_r, \mathtt{case_{cons}}(T, T_r}$$

and

$$\frac{\begin{array}{c}R(\delta^\ulcorner e^\urcorner) \Downarrow^{f'} T' \ \ (\star\star) \\ R(\delta^\ulcorner e_1 \urcorner) \Downarrow^{f'_r} T'_1 \ \ (\diamond\diamond) \qquad \mathrm{nil} = \mathtt{V}(T') \qquad v'_r = \mathtt{V}(T'_r)\end{array}}{\text{case } R(\delta^\ulcorner e^\urcorner) \text{ of nil } \to R(\delta^\ulcorner e_1 \urcorner) \mid h :: tl \to R(\delta^\ulcorner e_2 \urcorner) \Downarrow^{f'+f'_r+c_{caseL}} \langle v'_r, \mathtt{case_{nil}}(T', T}$$

By IH 1 on the first premise, we get $(m, \delta^{\ulcorner} e^{\urcorner}) \in (\!|\text{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|\!)^{\sigma t}_{\varepsilon}$.
Unrolling its definition with $(\star)$, $(\star\star)$ and $f < m$, we get

a) $\langle T, \delta^{\ulcorner} e^{\urcorner}\rangle \curvearrowright w', T', c'$
b) $\text{cons}(v_h, v_{tl}) = L(w') \ \wedge\ \text{nil} = R(w')$
c) $c' \leqslant \sigma t$
d) $(m - f, w') \in (\!|\text{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|\!)_v$

However, d) is false since two lists of different length are not related, therefore this case is vacuously true.

subcase 4:

$$\frac{\begin{array}{c} L(\delta^{\ulcorner} e^{\urcorner}) \Downarrow^{f} T \quad (\star) \\ L(\delta^{\ulcorner} e_1{}^{\urcorner}) \Downarrow^{f_r} T_1 \quad (\diamond) \qquad \text{nil} = V(T) \qquad v_r = V(T_r) \end{array}}{\text{case } L(\delta^{\ulcorner} e^{\urcorner}) \text{ of nil } \to L(\delta^{\ulcorner} e_1{}^{\urcorner}) \mid h :: tl \to L(\delta^{\ulcorner} e_2{}^{\urcorner}) \Downarrow^{f+f_r+c_{caseL}} \langle v_r, \mathsf{case}_{\mathsf{nil}}(T, T_r)\rangle}\ \textbf{ev-c}$$

and

$$\frac{\begin{array}{c} R(\delta^{\ulcorner} e^{\urcorner}) \Downarrow^{f'} T' \quad (\star\star) \qquad \text{cons}(v'_h, v'_{tl}) = V(T') \\ R(\delta^{\ulcorner} e_2{}^{\urcorner})[v'_h/h, v'_{tl}/tl] \Downarrow^{f'_r} T'_r \quad (\diamond\diamond) \qquad v'_r = V(T'_r) \end{array}}{\text{case } R(\delta^{\ulcorner} e^{\urcorner}) \text{ of nil } \to R(\delta^{\ulcorner} e_1{}^{\urcorner}) \mid h :: tl \to R(\delta^{\ulcorner} e_2{}^{\urcorner}) \Downarrow^{f'+f'_r+c_{caseL}} \langle v'_r, \mathsf{case}_{\mathsf{cons}}(T', T'_r)\rangle}\ \textbf{e}$$

By IH 1 on the first premise, we get $(m, \delta^{\ulcorner} e^{\urcorner}) \in (\!|\text{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|\!)^{\sigma t}_{\varepsilon}$.
Unrolling its definition with $(\star)$, $(\star\star)$ and $f < m$, we get

a) $\langle T, \delta^{\ulcorner} e^{\urcorner}\rangle \curvearrowright w', T', c'$
b) $\text{nil} = L(w') \ \wedge\ \text{cons}(v'_h, v'_{tl}) = R(w')$
c) $c' \leqslant \sigma t$
d) $(m - f, w') \in (\!|\text{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|\!)_v$

However, d) is false since two lists of different length are not related, therefore this case is vacuously true.

Case:
$$\frac{\Delta; \Phi \vdash \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2\ \mathsf{wf} \qquad \Delta; \Phi; x : \tau_1, f : \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2, \Gamma \vdash_{\mathbb{CP}} e : \tau_2 \mid t}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \textbf{fix } f(x).e : \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2 \mid 0}\ \textbf{cp-fix}$$

Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.
TS: $(m, \text{fix } f(x).\delta^{\ulcorner} e^{\urcorner}) \in (\!|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)^{0}_{\varepsilon}$.
By lemma 31, STS: $(m, \text{fix } f(x).\delta^{\ulcorner} e^{\urcorner}) \in (\!|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)_v$.

Let $F = \text{fix } f(x).\delta\ulcorner e \urcorner$.
We prove the more general statement

$$\forall\, m' \leqslant m.\ (m', F) \in (\!|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)_\nu$$

by subinduction on m'.

There are two parts to show:

subcase 1: $m' = 0$
By the definition of the interpretation of function types, there are two parts to show:

subsubcase 1: $\forall j < m' = 0 \cdots$
Since there is no non-negative j such that $j < 0$, the goal is vacuously true.

subsubcase 2: TS: $(0, F) \in \mathbb{C}\, |\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|\, \mathbb{D}_\nu$
As above, since there is no $j < 0$,
RTS: $\forall j.(j, L(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu \wedge (j, R(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$.
Pick j.
We show the left projection only, the right one is similar.

- STS 1: $(j, L(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$
  We prove the more general statement

  $$\forall m' \leqslant j.\ (m', L(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$$

  by subinduction on m'.
  There are two cases:

  - $m' = 0$
    Since there is no non-negative j such that $j < 0$, the goal is vacuously true.

– $m' = m'' + 1 \leqslant m$
By sub-IH

$$(m'', \text{fix } f(x).L(\delta^{\ulcorner} e^{\urcorner})) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu \quad (1)$$

STS: $(m'' + 1, \text{fix } f(x).L(\delta^{\ulcorner} e^{\urcorner})) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$.
Pick $j'' < m'' + 1$ and assume that $(j'', \nu) \in [\![|\sigma\tau_1|]\!]_\nu$.
STS: $(j'', L(\delta^{\ulcorner} e^{\urcorner})[\nu/x, L(F)/f]) \in [\![|\sigma\tau_2|]\!]^\infty_\varepsilon$.
This follows by IH 3 on the premise instantiated with $(j'', \delta[x \mapsto \nu, f \mapsto L(F)]) \in \mathcal{G}[\![x : |\sigma\tau_1|, f : |\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|, |\sigma\Gamma|]\!]$ which holds because

* $(j'', \delta) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$ using lemma 32 on $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$
* $(j'', \nu) \in [\![|\sigma\tau_1|]\!]_\nu$, from the assumption above
* $(j'', \text{fix } f(x).L(\delta^{\ulcorner} e^{\urcorner})) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$, obtained by downward closure (Lemma 33) on (1) using $j'' \leqslant m''$

subcase 2: $m' = m'' + 1 \leqslant m$
By sub-IH

$$(m'', F) \in (\!|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)_\nu \quad (2)$$

TS: $(m'' + 1, \text{fix } f(x).\delta^{\ulcorner} e^{\urcorner}) \in (\!|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)_\nu$
There are two cases to show:

subsubcase 1: Pick $j < m'' + 1$ and assume that $(j, w) \in (\!|\sigma\tau_1|\!)_\nu$.
STS: $(j, \delta^{\ulcorner} e^{\urcorner}[w/x, F/f]) \in (\!|\sigma\tau_2|\!)^{\sigma t}_\varepsilon$.
This follows by IH 1 on the second premise instantiated with $(j, \delta[x \mapsto w, f \mapsto F]) \in \mathcal{G}(\!|\sigma\Gamma, x : \sigma\tau_1, f : \sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)$ which holds because

- $(j, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ obtained by downward closure (lemma 33) using $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $j < m' \leqslant m$.
- $(j, w) \in (\!|\sigma\tau_1|\!)_\nu$, from the assumption above

- $(j, F) \in (\!|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)_v$, obtained by downward closure (Lemma 33) on (2) using $j \leqslant m''$

subsubcase 2: STS: $(m''+1, \text{fix } f(x).\delta^{\ulcorner}e^{\urcorner}) \in \mathcal{C}\,|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|\,\mathcal{D}_v$

There are also two cases to show here.

- Pick $j < m''+1$ and assume that $(j, w) \in (\!|\mathcal{U}\,|\sigma\tau_1|\!|)_v$.
  STS: $(j, \delta^{\ulcorner}e^{\urcorner}[w/x, F/f]) \in (\!|\mathcal{U}\,\sigma\tau_2|\!|)^{\infty}_{\varepsilon}$.
  This follows by IH 5 on the second premise instantiated with
  $(j, \delta[x \mapsto w, f \mapsto F]) \in \mathcal{G}(\!|\mathcal{U}\,|\sigma\Gamma|, x : \mathcal{U}\,|\sigma\tau_1|, f : \mathcal{U}\,(|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|)|\!)$ which holds because
  - $(j, \delta) \in \mathcal{G}(\!|\mathcal{U}\,|\sigma\Gamma|\!|)$ obtained by downward closure (Lemma 33) on $(m, \delta) \in \mathcal{G}(\!|\mathcal{U}\,|\sigma\Gamma|\!|)$ (obtained by inclusion lemma on $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!|)$) and $j < m' \leqslant m$.
  - $(j, w) \in (\!|\mathcal{U}\,|\sigma\tau_1|\!|)_v$, from the assumption above
  - $(j, F) \in (\!|\mathcal{U}\,(|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|)|\!)_v$, obtained by downward closure (Lemma 33) on (2) using $j \leqslant m''$
- STS: $\forall j.(j, L(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_v \wedge (j, R(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_v$.
  The proof is same as above subcase where $m' = 0$.

This completes the proof of this case.

Case:
$$\frac{\begin{array}{c}\Delta; \Phi \vdash \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2\ \mathtt{wf} \\ \Delta; \Phi; x : \tau_1, f : \Box\,(\tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2), \Gamma \vdash_{\mathbb{CP}} e : \tau_2 \mid t \\ \forall x \in \mathrm{dom}(\Gamma).\ \ \Delta; \Phi \models \Gamma(x) \sqsubseteq \Box\,\Gamma(x)\end{array}}{\Delta; \Phi; \Gamma, \Gamma' \vdash_{\mathbb{CP}} \textbf{fix } f(x).e : \Box\,(\tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2) \mid 0}\ \textbf{cp-fixNC}$$

Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma, \sigma\Gamma'|\!)$ and $\models \sigma\Phi$.
Then, $\delta = \delta_1 \cup \delta_2$ such that $(m, \delta_1) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $(m, \delta_2) \in \mathcal{G}(\!|\sigma\Gamma'|\!)$.
TS: $(m, \text{fix } f(x).\delta^{\ulcorner}e^{\urcorner}) \in (\!|\Box\,(\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2)|\!)^{0}_{\varepsilon}$.
Since e doesn't have any free variables from Γ' by the second premise,
TS: $(m, \text{fix } f(x).\delta_1{}^{\ulcorner}e^{\urcorner}) \in (\!|\Box\,(\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2)|\!)^{0}_{\varepsilon}$.
By lemma 31, STS: $(m, \text{fix } f(x).\delta_1{}^{\ulcorner}e^{\urcorner}) \in (\!|\Box\,(\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2)|\!)_v$.

By lemma 39 using $(m, \delta_1) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and the third premise, we get $(m, \delta_1) \in \mathcal{G}(\!|\Box\,\sigma\Gamma|\!)$, i.e. $\forall x \in \mathtt{dom}(\Gamma).\mathtt{stable}(\delta_1(x))$.
We also know that by definition, $\mathtt{stable}(\ulcorner e \urcorner)$.
Hence, $\mathtt{stable}(\text{fix } f(x).\delta_1 \ulcorner e \urcorner)$.
Therefore, STS: $(m, \text{fix } f(x).\delta_1 \ulcorner e \urcorner) \in (\!|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)_v$.
Let $F = \text{fix } f(x).\delta_1 e$.
We prove the more general statement

$$\forall\, m' \leqslant m.\ (m', F) \in (\!|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)_v$$

by subinduction on m'.
There are two parts to show:

subcase 1: $m' = 0$
By the definition of the interpretation of function types, there are two parts to show:

subsubcase 1: $\forall j < m' = 0$
Since there is no non-negative j such that $j < 0$, the goal is vacuously true.

subsubcase 2: TS: $(0, F) \in (\!|\!(\, |\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2| \,)\!|\!)_v$
As above, since there is no $j < 0$,
RTS: $\forall j.(j, L(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_v \wedge (j, R(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_v$.
Pick j.
We show the left projection only, the right one is similar.

- STS 1: $(j, L(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_v$
  We prove the more general statement

  $$\forall m' \leqslant j.\ (m', L(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_v$$

  by subinduction on m'.
  There are two cases:

- $m' = 0$
  Since there is no non-negative j such that $j < 0$, the goal is vacuously true.
- $m' = m'' + 1 \leqslant m$
  By sub-IH

$$(m'', \text{fix } f(x).L(\delta_1 \ulcorner e \urcorner)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu \quad (1)$$

  STS: $(m'' + 1, \text{fix } f(x).L(\delta_1 \ulcorner e \urcorner)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$.
  Pick $j'' < m'' + 1$ and assume that $(j'', \nu) \in [\![|\sigma\tau_1|]\!]_\nu$.
  STS: $(j'', L(\delta_1 \ulcorner e \urcorner)[\nu/x, L(F)/f]) \in [\![|\sigma\tau_2|]\!]^\infty_\varepsilon$.
  This follows by IH 3 on the premise instantiated with $(j'', \delta_1[x \mapsto \nu, f \mapsto L(F)]) \in \mathcal{G}[\![x : |\sigma\tau_1|, f : |\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|, |\sigma\Gamma|]\!]$ which holds because
  - $(j'', \delta_1) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$ using lemma 32 on $(m, \delta_1) \in \mathcal{G}(\!|\sigma\Gamma|\!)$
  - $(j'', \nu) \in [\![|\sigma\tau_1|]\!]_\nu$, from the assumption above
  - $(j'', \text{fix } f(x).L(\delta_1 \ulcorner e \urcorner)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$, obtained by downward closure (Lemma 33) on (1) using $j'' \leqslant m''$

subcase 2: $m' = m'' + 1 \leqslant m$
By sub-IH

$$(m'', F) \in (\!|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)_\nu \quad (2)$$

TS: $(m'' + 1, \text{fix } f(x).\delta_1 \ulcorner e \urcorner) \in (\!|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)_\nu$
There are two cases to show:

subsubcase 1: Pick $j < m'' + 1$ and assume that $(j, w) \in (\!|\sigma\tau_1|\!)_\nu$.
STS: $(j, \delta_1 \ulcorner e \urcorner[w/x, F/f]) \in (\!|\sigma\tau_2|\!)^{\sigma t}_\varepsilon$.
This follows by IH 1 on the second premise instantiated with $(j, \delta_1[x \mapsto w, f \mapsto F]) \in \mathcal{G}(\!|\sigma\Gamma, x : \sigma\tau_1, f : \square(\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2)|\!)$ which holds because

- $(j, \delta_1) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ obtained by downward closure (lemma 33) using $(m, \delta_1) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $j < m' \leqslant m$.
- $(j, w) \in (\!|\sigma\tau_1|\!)_v$, from the assumption above
- $(j, F) \in (\!|\Box\,(\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2)|\!)_v$, obtained by downward closure (Lemma 33) on (2) using $j \leqslant m''$ and also by $\texttt{stable}(F)$

subsubcase 2: STS: $(m''+1, \text{fix } f(x).\delta_1{\ulcorner}e{\urcorner}) \in \mathcal{C}\,|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|\,\mathcal{D}_v$

There are also two cases to show here.

- Pick $j < m''+1$ and assume that $(j, w) \in (\!|\mathcal{U}\,|\sigma\tau_1|\!|)_v$.
  STS: $(j, \delta_1{\ulcorner}e{\urcorner}[w/x, F/f]) \in (\!|\mathcal{U}\,\sigma\tau_2|\!|)^{\infty}_{\varepsilon}$.
  This follows by IH 5 on the second premise instantiated with
  $(j, \delta_1[x \mapsto w, f \mapsto F]) \in \mathcal{G}(\!|\mathcal{U}\,|\sigma\Gamma|, x : \mathcal{U}\,|\sigma\tau_1|, f : \mathcal{U}\,(|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|)|\!)$ which holds because
  - $(j, \delta_1) \in \mathcal{G}(\!|\mathcal{U}\,|\sigma\Gamma|\!|)$ obtained by downward closure (Lemma 33) on $(m, \delta_1) \in \mathcal{G}(\!|\mathcal{U}\,|\sigma\Gamma|\!|)$ (obtained by inclusion lemma on $(m, \delta_1) \in \mathcal{G}(\!|\sigma\Gamma|\!)$) and $j < m' \leqslant m$.
  - $(j, w) \in (\!|\mathcal{U}\,|\sigma\tau_1|\!|)_v$, from the assumption above
  - $(j, F) \in (\!|\mathcal{U}\,(|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|)|\!)_v$, obtained by downward closure (Lemma 33) on (2) using $j \leqslant m''$
- STS: $\forall j.(j, L(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_v \wedge (j, R(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_v$.
  The proof is same as above subcase where $m' = 0$.

This completes the proof of this case.

Case:
$$\frac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_1 : \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2 \mid t_1 \qquad \Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_2 : \tau_1 \mid t_2}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_1\ e_2 : \tau_2 \mid t_1 + t_2 + t}\ \textbf{cp-app}$$

Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.
TS: $(m, \delta{\ulcorner}e_1\ e_2{\urcorner}) \in (\!|\sigma\tau_2|\!)^{\sigma t_1 + \sigma t_2 + \sigma t}_{\varepsilon}$.
Following the definition of $(\!|\cdot|\!)^{\cdot}_{\varepsilon}$, assume that

$$\frac{\begin{array}{c} L(\delta\ulcorner e_1\urcorner) \Downarrow^{f_1} T_1 \quad (\star) \\ L(\delta\ulcorner e_2\urcorner) \Downarrow^{f_2} T_2 \quad (\diamond) \qquad \text{fix } f(x).e = V(T_1) \qquad v_2 = V(T_2) \\ e[v_2/x, (\text{fix } f(x).e)/f] \Downarrow^{f_r} T_r \quad (\dagger) \qquad v_r = V(T_r) \end{array}}{L(\delta\ulcorner e_1\urcorner)\ L(\delta\ulcorner e_2\urcorner) \Downarrow^{f_1+f_2+f_r+c_{app}} \langle v_r, \mathsf{app}(T_1, T_2, T_r)\rangle}\ \textbf{ev-app} \text{ and}$$

$$\frac{\begin{array}{c} R(\delta\ulcorner e_1\urcorner) \Downarrow^{f'_1} T'_1 \quad (\star\star) \\ R(\delta\ulcorner e_2\urcorner) \Downarrow^{f'_2} T'_2 \quad (\diamond\diamond) \qquad \text{fix } f(x).e' = V(T'_1) \qquad v'_2 = V(T'_2) \\ e[v'_2/x, (\text{fix } f(x).e')/f] \Downarrow^{f'_r} T'_r \quad (\dagger\dagger) \qquad v'_r = V(T'_r) \end{array}}{R(\delta\ulcorner e_1\urcorner)\ R(\delta\ulcorner e_2\urcorner) \Downarrow^{f'_1+f'_2+f'_r+c_{app}} \langle v'_r, \mathsf{app}(T'_1, T'_2, T'_r)\rangle}\ \textbf{ev-app} \text{ and}$$

$(f_1 + f_2 + f_r + c_{app}) < m$.

By IH 1 on the first premise, we get $(m, \delta\ulcorner e_1\urcorner) \in (\!|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)_\varepsilon^{\sigma t_1}$. Unrolling its definition with $(\star)$, $(\star\star)$ and $f_1 < m$, we get

a) $\langle T_1, \delta\ulcorner e_1\urcorner\rangle \curvearrowright w'_1, T'_1, c'_1$ where $w'_1 = \text{fix } f(x).ee$
b) $\text{fix } f(x).e = L(\text{fix } f(x).ee) \ \wedge\ \text{fix } f(x).e' = R(\text{fix } f(x).ee)$
c) $c'_1 \leqslant \sigma t_1$
d) $(m - f_1, \text{fix } f(x).ee) \in (\!|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)_v$

By IH 1 on the second premise, we get $(m, \delta\ulcorner e_2\urcorner) \in (\!|\sigma\tau_1|\!)_\varepsilon^{\sigma t_2}$. Unrolling its definition with $(\diamond)$ and $(\diamond\diamond)$ and $f_2 < m$, we get

e) $\langle T_2, \delta\ulcorner e_2\urcorner\rangle \curvearrowright w'_2, T'_2, c'_2$
f) $v_2 = L(w'_2) \ \wedge\ v'_2 = R(w'_2)$
g) $c'_2 \leqslant \sigma t_2$
h) $(m - f_2, w'_2) \in (\!|\sigma\tau_1|\!)_v$

Next, we apply downward-closure (lemma 33) to h) using

$$m - (f_1 + f_2 + c_{app}) \leqslant m - f_2$$

and we get

$$(m - (f_1 + f_2 + c_{app}), w'_2) \in (\!|\sigma\tau_1|\!)_v \tag{1}$$

We unroll d) using (1) since

$$m - (f_1 + f_2 + c_{app}) < m - f_1 \qquad \text{Note that here we have} \quad c_{app} \geqslant 1$$

and get

$$(m - (f_1 + f_2 + c_{app}), ee[w'_2/x, \text{fix } f(x).ee/f]) \in (\!|\sigma\tau_2|\!)^{\sigma t}_{\varepsilon} \tag{2}$$

Next, we unroll (2) using (†), (††) and $f_r < m - (f_1 + f_2 + c_{app})$ to obtain

i) $\langle T_r, ee[w'_2/x, \text{fix } f(x).ee/f]\rangle \curvearrowright w'_r, T'_r, c'_r$
j) $v_r = L(w'_r) \ \wedge \ v'_r = R(w'_r)$
k) $c'_r \leqslant \sigma t$
l) $(m - (f_1 + f_2 + f_r + c_{app}), w'_r) \in (\!|\sigma\tau_2|\!)_v$

Now, we can conclude as follows:

1. Using a), e) and i)
$$\frac{\begin{array}{c}\langle T_1, R(\delta^\ulcorner e_1 {}^\urcorner)\rangle \curvearrowright \texttt{fix } f(x).ee, T'_1, c'_1 \\ \langle T_2, R(\delta^\ulcorner e_2 {}^\urcorner)\rangle \curvearrowright w'_2, T'_2, c'_2 \\ \langle T_r, ee[w'_2/x, (\texttt{fix } f(x).ee)/f]\rangle \curvearrowright w'_r, T'_r, c'_r \qquad v'_r = V(T'_r)\end{array}}{\begin{array}{c}\langle\langle \_, \texttt{app}(T_1, T_2, T_r)\rangle, R(\delta^\ulcorner e_1 {}^\urcorner)\ R(\delta^\ulcorner e_2 {}^\urcorner)\rangle \curvearrowright \\ w'_r, \langle v'_r, \texttt{app}(T'_1, T'_2, T'_r)\rangle, c'_1 + c'_2 + c'_r\end{array}}\ \textbf{cp-app}$$
2. By j)
3. Using c), g) and k), we get $(c'_1 + c'_2 + c'_r) \leqslant \sigma t_1 + \sigma t_2 + \sigma t$
4. By l)

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash_{C\mathbb{P}} e_1 : \tau_1 \mid t_1 \qquad \Delta; \Phi_a; \Gamma \vdash_{C\mathbb{P}} e_2 : \tau_2 \mid t_2}{\Delta; \Phi_a; \Gamma \vdash_{C\mathbb{P}} \langle e_1, e_2\rangle : \tau_1 \times \tau_2 \mid t_1 + t_2}$ **cp-prod**

Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.
TS: $(m, \langle \delta^\ulcorner e_1 {}^\urcorner, \delta^\ulcorner e_2 {}^\urcorner\rangle) \in (\!|\sigma\tau_1 \times \sigma\tau_2|\!)^{\sigma t_1 + \sigma t_2}_{\varepsilon}$.
Following the definition of $(\!|\cdot|\!)^{\cdot}_{\varepsilon}\cdot$, assume that

$$\frac{L(\delta^\ulcorner e_1 {}^\urcorner) \Downarrow^{f_1} T_1 \ \ (\star) \qquad L(\delta^\ulcorner e_1 {}^\urcorner) \Downarrow^{f_2} T_2 \ \ (\diamond) \qquad v_i = V(T_i)}{\langle L(\delta^\ulcorner e_1 {}^\urcorner), L(\delta^\ulcorner e_2 {}^\urcorner)\rangle \Downarrow^{f_1 + f_2} \langle\langle v_1, v_2\rangle, \langle T_1, T_2\rangle\rangle}\ \textbf{e-pair}$$ and

$$\frac{R(\delta^\ulcorner e_1 {}^\urcorner) \Downarrow^{f'_1} T_1 \ \ (\star\star) \qquad R(\delta^\ulcorner e_1 {}^\urcorner) \Downarrow^{f'_2} T_2 \ \ (\diamond\diamond) \qquad v'_i = V(T'_i)}{\langle R(\delta^\ulcorner e_1 {}^\urcorner), R(\delta^\ulcorner e_2 {}^\urcorner)\rangle \Downarrow^{f'_1 + f'_2} \langle\langle v'_1, v'_2\rangle, \langle T'_1, T'_2\rangle\rangle}\ \textbf{e-pair}$$

and

$f_1 + f_2 < m$.

By IH 1 on the first premise, we get $(m, \delta^{\ulcorner} e_1 {}^{\urcorner}) \in (\!|\sigma\tau_1|\!)_\varepsilon^{\sigma t_1}$. Unrolling its definition with $(\star)$, $(\star\star)$ and $f_1 < m$, we get

a) $\langle T_1, \delta^{\ulcorner} e_1 {}^{\urcorner}\rangle \curvearrowright w_1', T_1', c_1'$
b) $v_1 = L(w_1') \ \wedge \ v_1' = R(w_1')$
c) $c_1' \leqslant \sigma t_1$
d) $(m - f_1, w_1') \in (\!|\sigma\tau_1|\!)_v$

By IH 1 on the second premise, we get $(m, \delta^{\ulcorner} e_2 {}^{\urcorner}) \in (\!|\sigma\tau_2|\!)_\varepsilon^{\sigma t_2}$. Unrolling its definition with $(\diamond)$, $(\diamond\diamond)$ and $f_2 < m$, we get

e) $\langle T_2, \delta^{\ulcorner} e_2 {}^{\urcorner}\rangle \curvearrowright w_2', T_2', c_2'$
f) $v_2 = L(w_2') \ \wedge \ v_2' = R(w_2')$
g) $c_2' \leqslant \sigma t_2$
h) $(m - f_2, w_2') \in (\!|\sigma\tau_2|\!)_v$

We can conclude as follows:

1. Using a) and e)
$$\frac{\langle T_1, \delta^{\ulcorner} e_1 {}^{\urcorner}\rangle \curvearrowright w_1', T_1', c_1' \qquad \langle T_2, \delta^{\ulcorner} e_2 {}^{\urcorner}\rangle \curvearrowright w_2', T_2', c_2' \qquad v_i' = V(T_i')}{\langle\langle \_, \langle T_1, T_2\rangle\rangle, (\delta^{\ulcorner} e_1 {}^{\urcorner}, \delta^{\ulcorner} e_2 {}^{\urcorner})\rangle \curvearrowright (w_1', w_2'), \langle\langle v_1', v_2'\rangle, \langle T_1', T_2'\rangle\rangle, c_1' + c_2'} \textbf{ cp-pair}$$
2. Using b) and f), $\langle v_1, v_2\rangle = L((w_1', w_2')) \ \wedge \ \langle v_1', v_2'\rangle = R((w_1', w_2'))$
3. By using c) and g), we get $c_1' + c_2' \leqslant \sigma t_1 + \sigma t_2$
4. By downward closure (Lemma 33) on d) and h) using

$$m - (f_1 + f_2) \leqslant m - f_1$$

$$m - (f_1 + f_2) \leqslant m - f_2$$

we get $(m - (f_1 + f_2), w_1') \in (\!|\sigma\tau_1|\!)_v$ and $(m - (f_1 + f_2), w_2') \in (\!|\sigma\tau_2|\!)_v$, when combined, gives us $(m - (f_1 + f_2), (w_1', w_2')) \in (\!|\sigma\tau_2|\!)_v$

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau_1 \times \tau_2 \mid t}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \pi_1(e) : \tau_1 \mid t}$ **cp-proj1**
Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m, \pi_1 \delta^\ulcorner e^\urcorner) \in (\!|\sigma\tau_1|\!)^{\sigma t}_\varepsilon$.
Following the definition of $(\!|\cdot|\!)^{\cdot}_\varepsilon \cdot$, assume that

$$\frac{L(\delta^\ulcorner e^\urcorner) \Downarrow^f T \quad (\star) \qquad \langle v_1, v_2\rangle = V(T)}{\pi_1 L(\delta^\ulcorner e^\urcorner) \Downarrow^{f+c_{proj}} \langle v_1, \pi_1\ T\rangle}\ \textbf{e-proj}_1 \text{ and}$$

$$\frac{R(\delta^\ulcorner e^\urcorner) \Downarrow^{f'} T' \quad (\star\star) \qquad \langle v_1', v_2'\rangle = V(T')}{\pi_1 R(\delta^\ulcorner e^\urcorner) \Downarrow^{f'+c_{proj}} \langle v_1', \pi_1\ T'\rangle}\ \textbf{e-proj}_1 \text{ and}$$

$f + c_{proj} < m$.
By IH 1 on the first premise, we get $(m, \delta^\ulcorner e^\urcorner) \in (\!|\sigma\tau_1 \times \sigma\tau_2|\!)^{\sigma t}_\varepsilon$.
Unrolling its definition with $(\star)$, $(\star\star)$ and $f < m$, we get

a) $\langle T, \delta^\ulcorner e^\urcorner\rangle \curvearrowright w', T', c'$ where $w' = (w_1, w_2)$
b) $\langle v_1, v_2\rangle = L((w_1, w_2)) \ \wedge\ \langle v_1', v_2'\rangle = R((w_1, w_2))$
c) $c_1' \leqslant \sigma t_1$
d) $(m - f_1, (w_1, w_2)) \in (\!|\sigma\tau_1 \times \sigma\tau_2|\!)_v$

We can conclude as follows:

1. Using a)
$$\frac{\langle T, \delta^\ulcorner e_1^\urcorner\rangle \curvearrowright (w_1, w_2), T', c' \qquad \langle v_1', v_2'\rangle = V(T')}{\langle \pi_1\ T, \pi_1 \delta^\ulcorner e_1^\urcorner\rangle \curvearrowright w_1, \langle v_1', \pi_1\ T'\rangle, c'}\ \textbf{cp-proj}_1$$
2. Using c), $v_1 = L(w_1) \ \wedge\ v_1' = R(w_1)$
3. By using c)
4. By downward closure (Lemma 33) on d) using

$$m - (f + c_{proj}) \leqslant m - f$$

we get $(m - (f + c_{proj}), w_1) \in (\!|\sigma\tau_1|\!)_v$.

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash_{C\mathbb{P}} e : \tau_1 \mid t \qquad \Delta; \Phi \vdash \tau_2\ \mathtt{wf}}{\Delta; \Phi_a; \Gamma \vdash_{C\mathbb{P}} \textbf{inl}\ e : \tau_1 + \tau_2 \mid t}\ \textbf{cp-inl}$
Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.
TS: $(m, \mathrm{inl}\ (\delta^\ulcorner e^\urcorner)) \in (\!|\sigma\tau_1 + \sigma\tau_2|\!)^{\sigma t}_\varepsilon$.
Following the definition of $(\!|\cdot|\!)^{\cdot}_\varepsilon \cdot$, assume that

$$\frac{L(\delta^\ulcorner e^\urcorner) \Downarrow^f T \quad (\star) \qquad v = V(T)}{\mathrm{inl}\ L(\delta^\ulcorner e^\urcorner) \Downarrow^f \langle \mathrm{inl}\ v, \mathtt{inl}\ T\rangle}\ \textbf{e-inl} \text{ and}$$

$$\frac{R(\delta\ulcorner e\urcorner) \Downarrow^{f'} T' \quad (\star\star) \qquad v' = V(T')}{\mathtt{inl}\ R(\delta\ulcorner e\urcorner) \Downarrow^{f'} \langle \mathtt{inl}\ v', \mathtt{inl}\ T'\rangle}\ \textbf{e-inl}$$ and $f < m$.

By IH 1 on the first premise, we get $(m, \delta\ulcorner e\urcorner) \in (\!|\sigma\tau_1|\!)^{\sigma t}_{\varepsilon}$. Unrolling its definition with $(\star)$, $(\star\star)$ and $f < m$, we get

a) $\langle T, \delta\ulcorner e\urcorner\rangle \curvearrowright w', T', c'$

b) $v = L(w') \ \wedge\ v' = R(w')$

c) $c' \leqslant \sigma t$

d) $(m - f, w') \in (\!|\sigma\tau|\!)_v$

We can conclude as follows:

1. Using a)
$$\frac{\langle T, \delta\ulcorner e\urcorner\rangle \curvearrowright w', T', c' \qquad v' = V(T')}{\langle\langle \_, \mathtt{inl}\ T\rangle, \mathtt{inl}\ \delta\ulcorner e\urcorner\rangle \curvearrowright \mathtt{inl}\ w', \langle \mathtt{inl}\ v', \mathtt{inl}\ T'\rangle, c'}\ \textbf{cp-inl}$$
2. Using b), $\mathtt{inl}\ v = L(\mathtt{inl}\ w) \ \wedge\ \mathtt{inl}\ v' = R(\mathtt{inl}\ w)$
3. By using c)
4. Using d), we can show that $(m - f, \mathtt{inl}\ w) \in (\!|\sigma\tau_1 + \sigma\tau_2|\!)_v$

Case:
$$\frac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau_1 + \tau_2 \mid t \qquad \Delta; \Phi; x : \tau_1, \Gamma \vdash_{\mathbb{CP}} e_1 : \tau \mid t' \qquad \Delta; \Phi; y : \tau_2, \Gamma \vdash_{\mathbb{CP}} e_2 : \tau \mid t'}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \mathbf{case}\ (e, x.e_1, y.e_2) : \tau \mid t + t'}\ \textbf{cp-case}$$

Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m,\ \mathrm{case}\ (\delta\ulcorner e\urcorner, \delta\ulcorner e_1\urcorner, \delta\ulcorner e_2\urcorner)) \in (\!|\sigma\tau|\!)^{\sigma t + \sigma t'}_{\varepsilon}$.

Following the definition of $(\!|\cdot|\!)^{\cdot}_{\varepsilon}$, assume that

$L(\ \mathrm{case}\ (\delta\ulcorner e\urcorner, \delta\ulcorner e_1\urcorner, \delta\ulcorner e_2\urcorner)) \Downarrow^{F} v_r$ and

$R(\ \mathrm{case}\ (\delta\ulcorner e\urcorner, \delta\ulcorner e_1\urcorner, \delta\ulcorner e_2\urcorner)) \Downarrow^{F'} v'_r$ and

$F < m$.

Depending on what $L(\delta\ulcorner e\urcorner)$ and $R(\delta\ulcorner e\urcorner)$ evaluate to, there are four cases:

subcase 1:
$$\frac{\begin{array}{c} L(\delta\ulcorner e\urcorner) \Downarrow^{f} T \quad (\star) \\ \mathtt{inl}\ v = V(T) \qquad L(\delta\ulcorner e_1\urcorner)[v/x] \Downarrow^{f_r} T_r \quad (\diamond) \qquad v_r = V(T_r) \end{array}}{\mathrm{case}\ (L(\delta\ulcorner e\urcorner), x.L(\delta\ulcorner e_1\urcorner), y.L(\delta\ulcorner e_2\urcorner)) \Downarrow^{f+f_r+c_{case}} \langle v_r, \mathtt{case}_{\mathtt{inl}}(T, T_r)\rangle}\ \textbf{ev-case-l}$$
and

$$\frac{\begin{array}{c} R(\delta\ulcorner e\urcorner) \Downarrow^{f'} T' \quad (\star\star) \\ \text{inl } v' = V(T') \qquad R(\delta\ulcorner e_1\urcorner)[v'/x] \Downarrow^{f'_r} T'_r \quad (\diamond\diamond) \qquad v'_r = V(T'_r) \end{array}}{\text{case } (R(\delta\ulcorner e\urcorner), x.R(\delta\ulcorner e_1\urcorner), y.R(\delta\ulcorner e_2\urcorner)) \Downarrow^{f'+f'_r+c_{case}} \langle v'_r, \mathtt{case}_{\mathtt{inl}}(T', T'_r)\rangle}\ \textbf{ev-case-l}$$

and

$F = f + f_r + c_{case} < m$.

By IH 1 on the first premise, we get $(m, \delta\ulcorner e\urcorner) \in (\!|\sigma\tau_1 + \sigma\tau_2|\!)^{\sigma t}_{\varepsilon}$.
Unrolling its definition with $(\star)$, $(\star\star)$ and $f < m$, we get

a) $\langle T, \delta\ulcorner e\urcorner\rangle \curvearrowright w', T', c'$ where $w' = \mathtt{inl}\ w$
b) $\text{inl } v = L(\mathtt{inl}\ w) \ \wedge\ \text{inl } v' = R(\mathtt{inl}\ w)$
c) $c' \leqslant \sigma t$
d) $(m - f, \mathtt{inl}\ w) \in (\!|\sigma\tau_1 + \sigma\tau_2|\!)_v$

By IH 1 on the second premise using $(m - f, \delta[x \mapsto w]) \in \mathcal{G}(\!|\sigma\Gamma, x : \sigma\tau_1|\!)$ obtained by

- $(m - f, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ by downward-closure (lemma 33) on $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ using $m - f \leqslant m$
- $(m - f, w) \in (\!|\sigma\tau_1|\!)_v$ by unfolding e)

we get $(m - f, \delta\ulcorner e_1\urcorner[w/x]) \in (\!|\sigma\tau|\!)^{\sigma t'}_{\varepsilon}$. Unrolling its definition with $(\diamond)$, $(\diamond\diamond)$ and $f_r < m - f$, we get

e) $\langle T_r, \delta\ulcorner e_1\urcorner[w/x]\rangle \curvearrowright w'_r, T'_r, c'_r$
f) $v_r = L(w'_r) \ \wedge\ v'_r = R(w'_r)$
g) $c'_r \leqslant \sigma t'$
h) $(m - f - f_r, w'_r) \in (\!|\sigma\tau'|\!)_v$

We conclude with

1. Using a) and e)
$$\frac{\begin{array}{c} \langle T, \delta\ulcorner e\urcorner\rangle \curvearrowright \mathtt{inl}\ w, T', c' \\ \langle T_r, \delta\ulcorner e_1\urcorner[w/x]\rangle \curvearrowright w'_r, T'_r, c'_r \qquad v'_r = V(T'_r) \end{array}}{\langle \langle \_, \mathtt{case}_{\mathtt{inl}}(T, T_r)\rangle, \mathtt{case}(\delta\ulcorner e\urcorner, x.\delta\ulcorner e_1\urcorner, y, \delta\ulcorner e_2\urcorner)\rangle \curvearrowright w'_r, \langle v'_r, \mathtt{case}_{\mathtt{inl}}(T', T'_r)\rangle, c' + c'_r}\ \textbf{cp-case-inl}$$
2. Using f)
3. By using c) and g), we get $c' + c'_r \leqslant \sigma t + \sigma t'$

4. By downward closure (Lemma 33) on h) using

$$m - (f + f_r + c_{case}) \leqslant m - f - f_r$$

we get $(m - (f + f_r + c_{case}), w'_r) \in (\!|\sigma\tau'|\!)_v$.

subcase 2:
$$\frac{e \Downarrow^{f} T \qquad \mathrm{inr}\, v = V(T) \qquad e_2[v/y] \Downarrow^{f_r} T_r \qquad v_r = V(T_r)}{\mathrm{case}\ (e, x.e_1, y.e_2) \Downarrow^{f+f_r+c_{case}} \langle v_r, \mathtt{case}_{\mathtt{inr}}(T, T_r)\rangle}\ \textbf{ev-case-r}$$
This case is symmetric, hence we skip its proof.

subcase 3:
$$\frac{\begin{array}{c} L(\delta^{\ulcorner} e^{\urcorner}) \Downarrow^{f} T \quad (\star) \\ \mathrm{inl}\, v = V(T) \qquad L(\delta^{\ulcorner} e_1{}^{\urcorner})[v/x] \Downarrow^{f_r} T_r \quad (\diamond) \qquad v_r = V(T_r) \end{array}}{\mathrm{case}\ (L(\delta^{\ulcorner} e^{\urcorner}), x.L(\delta^{\ulcorner} e_1{}^{\urcorner}), y.L(\delta^{\ulcorner} e_2{}^{\urcorner})) \Downarrow^{f+f_r+c_{case}} \langle v_r, \mathtt{case}_{\mathtt{inl}}(T, T_r)\rangle}\ \textbf{ev-case-l}$$
and
$$\frac{\begin{array}{c} R(\delta^{\ulcorner} e^{\urcorner}) \Downarrow^{f'} T' \quad (\star\star) \\ \mathrm{inr}\, v' = V(T') \qquad R(\delta^{\ulcorner} e_2{}^{\urcorner})[v'/y] \Downarrow^{f'_r} T'_r \quad (\diamond\diamond) \qquad v'_r = V(T'_r) \end{array}}{\mathrm{case}\ (R(\delta^{\ulcorner} e^{\urcorner}), x.R(\delta^{\ulcorner} e_1{}^{\urcorner}), y.R(\delta^{\ulcorner} e_2{}^{\urcorner})) \Downarrow^{f'+f'_r+c_{case}} \langle v'_r, \mathtt{case}_{\mathtt{inr}}(T', T'_r)\rangle}\ \textbf{ev-case-r}$$
and
$F = f + f_r + c_{case} < m$.
By IH 1 on the first premise, we get $(m, \delta^{\ulcorner} e^{\urcorner}) \in (\!|\sigma\tau_1 + \sigma\tau_2|\!)^{\sigma t}_{\varepsilon}$.
Unrolling its definition with $(\star)$, $(\star\star)$ and $f < m$, we get

a) $\langle T, \delta^{\ulcorner} e^{\urcorner}\rangle \curvearrowright w', T', c'$
b) $\mathrm{inl}\, v = L(\mathtt{inl}\ w) \ \wedge\ \mathrm{inr}\, v' = R(\mathtt{inl}\ w)$
c) $c' \leqslant \sigma t$
d) $(m - f, w') \in (\!|\sigma\tau_1 + \sigma\tau_2|\!)_v$

However, d) is false since a bi-value with different tags are not related at type $\sigma\tau_1 + \sigma\tau_2$.

subcase 4:
$$\frac{\begin{array}{c} L(\delta^{\ulcorner} e^{\urcorner}) \Downarrow^{f} T \quad (\star) \\ \mathrm{inr}\, v = V(T) \qquad L(\delta^{\ulcorner} e_2{}^{\urcorner})[v/y] \Downarrow^{f_r} T_r \quad (\diamond) \qquad v_r = V(T_r) \end{array}}{\mathrm{case}\ (L(\delta^{\ulcorner} e^{\urcorner}), x.L(\delta^{\ulcorner} e_1{}^{\urcorner}), y.L(\delta^{\ulcorner} e_2{}^{\urcorner})) \Downarrow^{f+f_r+c_{case}} \langle v_r, \mathtt{case}_{\mathtt{inr}}(T, T_r)\rangle}\ \textbf{ev-case-r}$$
and

$$\frac{\begin{array}{c} R(\delta^{\ulcorner} e^{\urcorner}) \Downarrow^{f'} T' \quad (\star\star) \\ \text{inl } v' = V(T') \qquad R(\delta^{\ulcorner} e_1{}^{\urcorner})[v'/x] \Downarrow^{f'_r} T'_r \quad (\diamond\diamond) \qquad v'_r = V(T'_r) \end{array}}{\text{case } (R(\delta^{\ulcorner} e^{\urcorner}), x.R(\delta^{\ulcorner} e_1{}^{\urcorner}), y.R(\delta^{\ulcorner} e_2{}^{\urcorner})) \Downarrow^{f'+f'_r+c_{case}} \langle v'_r, \mathtt{case}_{\mathtt{inl}}(T', T'_r)\rangle}\ \textbf{ev-case-l}$$

and

$F = f + f_r + c_{case} < m$.

By IH 1 on the first premise, we get $(m, \delta^{\ulcorner} e^{\urcorner}) \in (\!|\sigma\tau_1 + \sigma\tau_2|\!)^{\sigma t}_{\varepsilon}$.

Unrolling its definition with $(\star)$, $(\star\star)$ and $f < m$, we get

a) $\langle T, \delta^{\ulcorner} e^{\urcorner}\rangle \curvearrowright w', T', c'$

b) $\text{inr } v = L(\mathtt{inl}\ w) \ \wedge\ \text{inl } v' = R(\mathtt{inl}\ w)$

c) $c' \leqslant \sigma t$

d) $(m - f, w') \in (\!|\sigma\tau_1 + \sigma\tau_2|\!)_v$

However, d) is false since a bi-value with different tags are not related at type $\sigma\tau_1 + \sigma\tau_2$.

Case: $$\frac{i :: S, \Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t \qquad i \notin \mathbf{FIV}(\Phi; \Gamma)}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \Lambda.e : \forall i \overset{\mathbb{CP}(t)}{::} S.\, \tau \mid 0}\ \textbf{cp-iLam}$$

Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m, \Lambda.\delta^{\ulcorner} e^{\urcorner}) \in (\!|\forall i \overset{\mathbb{CP}(\sigma t)}{::} S.\, \sigma\tau|\!)^{0}_{\varepsilon}$.

By lemma 31, STS: $(m, \Lambda.\delta^{\ulcorner} e^{\urcorner}) \in (\!|\forall i \overset{\mathbb{CP}(\sigma t)}{::} S.\, \sigma\tau|\!)_v$ $(\star)$.

There are two cases to show:

subcase 1: By unrolling $(\star)$'s definition, assume that $\vdash I :: S$.

STS: $(m, \delta^{\ulcorner} e^{\urcorner}) \in (\!|\sigma\tau\{I/i\}|\!)^{\sigma t[I/i]}_{\varepsilon}$.

This follows by IH 1 on the premise instantiated with the substitution $\sigma[i \mapsto I] \in \mathcal{D}[\![i :: S, \Delta]\!]$.

subcase 2: STS: $(m, \Lambda.\delta^{\ulcorner} e^{\urcorner}) \in (\!(\, \forall i \overset{\mathbb{FS}(\sigma t)}{::} S.\, |\sigma\tau|\,)\!)_v$ $(\star\star)$.

By unrolling $(\star\star)$'s definition, assume that $\vdash I :: S$.

STS: $(m, \delta^{\ulcorner} e^{\urcorner}) \in (\!|\mathcal{U}\,(|\sigma\tau\{I/i\}|)|\!)^{\infty}_{\varepsilon}$.

This follows by IH 5 on the premise instantiated with the substitution $\sigma[i \mapsto I] \in \mathcal{D}[\![i :: S, \Delta]\!]$.

Case: $\dfrac{\Delta;\Phi_a;\Gamma \vdash_{\mathbb{CP}} e : \forall i \overset{\mathbb{CP}(t')}{::} S.\,\tau \mid t \qquad \Delta \vdash I : S}{\Delta;\Phi_a;\Gamma \vdash_{\mathbb{CP}} e[\,] : \tau\{I/i\} \mid t + t'[I/i]}$ **cp-iApp**

Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m, \delta\ulcorner e \urcorner[\,]) \in (\!|\sigma\tau\{\sigma I/i\}|\!)_\varepsilon^{\sigma t + \sigma t'[\sigma I/i]}$.

Following the definition of $(\!|\cdot|\!)_\varepsilon^{\cdot}$, assume that

$$\frac{L(\delta\ulcorner e\urcorner) \Downarrow^{f} T \;\; (\star) \qquad \Lambda.e' = V(T) \qquad e' \Downarrow^{f_r} T_r \;\; (\diamond) \qquad v_r = V(T_r)}{L(\delta\ulcorner e\urcorner)[\,] \Downarrow^{f+f_r} \langle v_r, \mathtt{iApp}(T, T_r)\rangle}\ \textbf{ev-iApp}$$

and

$$\frac{R(\delta\ulcorner e\urcorner) \Downarrow^{f'} T' \;\; (\star\star) \qquad \Lambda.e'' = V(T') \qquad e'' \Downarrow^{f'_r} T'_r \;\; (\diamond\diamond) \qquad v'_r = V(T'_r)}{R(\delta\ulcorner e\urcorner)[\,] \Downarrow^{f'+f'_r} \langle v'_r, \mathtt{iApp}(T', T'_r)\rangle}\ \textbf{ev-iApp}$$ and

$(f + f_r) < m$.

By IH on the first premise, we get $(m, \delta\ulcorner e\urcorner) \in (\!|\forall i \overset{\mathbb{CP}(\sigma t')}{::} S.\,\sigma\tau|\!)_\varepsilon^{\sigma t}$.

By unrolling its definition with $(\star)$, $(\star\star)$ and $f < m$, we get

a) $\langle T, \delta\ulcorner e\urcorner\rangle \curvearrowright w', T', c'$ where $w' = \Lambda.ee'$

b) $\Lambda.e' = L(w') \;\wedge\; \Lambda.e'' = R(w')$

c) $c' \leqslant \sigma t$

d) $(m - c, \Lambda.ee') \in (\!|\forall i \overset{\mathbb{CP}(\sigma t')}{::} S.\,\sigma\tau|\!)_v$

By lemma 22 on the second premise using $\sigma \in \mathcal{D}[\![\Delta]\!]$, we get

$$\vdash \sigma I :: S \tag{1}$$

By unrolling the definition of e) with (1), we get

$$(m - f, ee') \in (\!|\sigma\tau\{\sigma I/i\}|\!)_\varepsilon^{\sigma t'[\sigma I/i]} \tag{2}$$

By unrolling the definition of (2) with $(\diamond)$, $(\diamond\diamond)$ and $f_r < m - f$, we get

e) $\langle T_r, ee'\rangle \curvearrowright w'_r, T'_r, c'_r$

f) $v_r = L(w'_r) \;\wedge\; v'_r = R(w'_r)$

g) $c'_r \leqslant \sigma t'$

h) $(m - (f + f_r), w'_r) \in (\!|\sigma\tau\{\sigma I/i\}|\!)_v$

We conclude as follows

1. Using a) and e)
$$\frac{\langle T, \delta^{\ulcorner} e^{\urcorner}\rangle \curvearrowright \Lambda.\text{æ}', T', c' \qquad \langle T_r, \text{æ}'\rangle \curvearrowright w'_r, T'_r, c'_r \qquad \nu'_r = V(T'_r)}{\langle\langle \_, \texttt{iApp}(T, T_r)\rangle, \delta^{\ulcorner} e^{\urcorner}[]\rangle \curvearrowright w'_r, \langle \nu'_r, \texttt{iApp}(T', T'_r)\rangle, c' + c'_r}\ \textbf{cp-iApp}$$
2. Using f)
3. By using c) and g), we get $c' + c'_r \leqslant \sigma t + \sigma t'$
4. By h)

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau\{I/i\} \mid t \qquad \Delta \vdash I :: S}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \textbf{pack}\ e : \exists i{::}S.\, \tau \mid t}$ **cp-pack**

Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m, \text{pack}\ \delta^{\ulcorner} e^{\urcorner}) \in (\!|\exists i{::}S.\, \sigma\tau|\!)^{\sigma t}_{\varepsilon}$.

Following the definition of $(\!|\cdot|\!)^{\cdot}_{\varepsilon}$, assume that

$\dfrac{L(\delta^{\ulcorner} e^{\urcorner}) \Downarrow^{f} T \quad (\star) \qquad \nu = V(T)}{\text{pack}\ L(\delta^{\ulcorner} e^{\urcorner}) \Downarrow^{f} \langle \text{pack}\ \nu, \text{pack}\ T\rangle}$ **ev-pack** and

$\dfrac{R(\delta^{\ulcorner} e^{\urcorner}) \Downarrow^{f'} T' \quad (\star\star) \qquad \nu' = V(T')}{\text{pack}\ R(\delta^{\ulcorner} e^{\urcorner}) \Downarrow^{f'} \langle \text{pack}\ \nu', \text{pack}\ T'\rangle}$ **ev-pack** and

$f < m$.

By IH on the first premise, we get $(m, \delta^{\ulcorner} e^{\urcorner}) \in (\!|\sigma\tau\{\sigma I/i\}|\!)^{\sigma t}_{\varepsilon}$.

By unrolling its definition with $(\star)$, $(\star\star)$ and $f < m$, we get

a) $\langle T, \delta^{\ulcorner} e^{\urcorner}\rangle \curvearrowright w', T', c'$

b) $\nu = L(w') \ \wedge\ \nu' = R(w')$

c) $c' \leqslant \sigma t$

d) $(m - f, w) \in (\!|\sigma\tau\{\sigma I/i\}|\!)_{\nu}$

By lemma 22 on the second premise, we get

$$\vdash \sigma I :: S \tag{1}$$

We can conclude as follows

1. Using a)
$$\frac{\langle T, \delta^{\ulcorner} e^{\urcorner}\rangle \curvearrowright w', T', c' \qquad \nu'_r = V(T')}{\langle\langle \_, \text{pack}\ T\rangle, \text{pack}\ \delta^{\ulcorner} e^{\urcorner}\rangle \curvearrowright \text{pack}\ w', \langle \text{pack}\ \nu'_r, \text{pack}\ T'\rangle, c'}\ \textbf{cp-pack}$$
2. Using b), $\text{pack}\ \nu = L(\text{pack}\ w) \ \wedge\ \text{pack}\ \nu' = R(\text{pack}\ w)$

3. By using c)
4. TS: $(m - f, \mathsf{pack}\ w) \in (\!|\exists i{::}S.\ \sigma\tau|\!)_v$
   STS1: $\vdash \sigma I :: S$ follows directly by (1).
   STS2: $(m - f, w) \in (\!|\sigma\tau\{\sigma I/i\}|\!)_v$ follows by d)

Case: $$\frac{\Delta; \Phi_a; \Gamma \vdash_{C\mathbb{P}} e_1 : \exists i{::}S.\ \tau_1 \mid t_1 \qquad i :: S, \Delta; \Phi; x : \tau_1, \Gamma \vdash_{C\mathbb{P}} e_2 : \tau_2 \mid t_2 \qquad i \notin FV(\Phi; \Gamma, \tau_2, t_2)}{\Delta; \Phi_a; \Gamma \vdash_{C\mathbb{P}} \mathbf{unpack}\ e_1\ \mathbf{as}\ x\ \mathbf{in}\ e_2 : \tau_2 \mid t_1 + t_2}\ \textbf{cp-unpack}$$

Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.
TS: $(m, \mathsf{unpack}\ \delta\ulcorner e_1 \urcorner\ \mathsf{as}\ x\ \mathsf{in}\ \delta\ulcorner e_2 \urcorner) \in (\!|\sigma\tau_2|\!)_\varepsilon^{\sigma t_1 + \sigma t_2}$.
Following the definition of $(\!|\cdot|\!)_\varepsilon^{\cdot}$, assume that

$$\frac{L(\delta\ulcorner e_1 \urcorner) \Downarrow^{f_1} T_1\ \ (\star) \qquad \mathsf{pack}\ v = V(T_1) \qquad L(\delta\ulcorner e_2 \urcorner)[v/x] \Downarrow^{f_r} T_r\ \ (\diamond) \qquad v_r = V(T_r)}{\mathsf{unpack}\ L(\delta\ulcorner e_1 \urcorner)\ \mathsf{as}\ x\ \mathsf{in}\ L(\delta\ulcorner e_2 \urcorner) \Downarrow^{f_1 + f_r} \langle v_r, \mathsf{unpack}(T_1, x, T_r)\rangle}\ \textbf{ev-unpack}$$

and

$$\frac{R(\delta\ulcorner e_1 \urcorner) \Downarrow^{f'_1} T'_1\ \ (\star\star) \qquad \mathsf{pack}\ v' = V(T'_1) \qquad R(\delta\ulcorner e_2 \urcorner)[v'/x] \Downarrow^{f'_r} T'_r\ \ (\diamond\diamond) \qquad v'_r = V(T'_r)}{\mathsf{unpack}\ R(\delta\ulcorner e_1 \urcorner)\ \mathsf{as}\ x\ \mathsf{in}\ R(\delta\ulcorner e_2 \urcorner) \Downarrow^{f'_1 + f'_r} \langle v'_r, \mathsf{unpack}(T'_1, x, T'_r)\rangle}\ \textbf{ev-unpack}$$

and

$(f_1 + f_2) < m$.
By IH 1 on the first premise, we get $(m, \delta\ulcorner e_1 \urcorner) \in (\!|\exists i{::}S.\ \sigma\tau_1|\!)_\varepsilon^{\sigma t_1}$.
By unrolling its definition with $(\star)$,$(\star\star)$ and $f_1 < m$, we get

a) $\langle T_1, \delta\ulcorner e_1 \urcorner\rangle \curvearrowright w'_1, T'_1, c'_1$ where $w'_1 = \mathsf{pack}\ w$
b) $\mathsf{pack}\ v = L(\mathsf{pack}\ w)\ \wedge\ \mathsf{pack}\ v' = R(\mathsf{pack}\ w)$
c) $c' \leqslant \sigma t_1$
d) $(m - f_1, \mathsf{pack}\ w) \in (\!|\exists i{::}S.\ \sigma\tau_1|\!)_v$

By unrolling the definition of e), we get

$$\vdash I :: S \tag{1}$$

$$(m - f_1, w) \in (\!|\sigma\tau_1\{I/i\}|\!)_v \tag{2}$$

By downward closure (Lemma 33) on $(m, \delta) \in \mathcal{G}(\!|\Gamma|\!)$, we have

$$(m - f_1, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!) \tag{3}$$

By IH 1 on the second premise using

- $\sigma[i \mapsto I] \in \mathcal{D}[\![i :: S, \Delta]\!]$ using (1)
- $(m - f_1, \delta[x \mapsto w]) \in \mathcal{G}(\!|\sigma[i \mapsto I](\Gamma, x : \tau_1)|\!)$ using (2) and (3)

we get

$$(m - f_1, \delta\ulcorner e_2 \urcorner[w/x]) \in (\!|\sigma\tau_2|\!)_\varepsilon^{\sigma t_2} \tag{4}$$

By unrolling (4)'s definition using ($\diamond$), ($\diamond\diamond$) and $f_2 < m - f_1$, we get

e) $\langle T_r, \delta\ulcorner e_2 \urcorner[w/x]\rangle \curvearrowright w_r', T_r', c_r'$
f) $v_r = L(w_r') \ \wedge \ v_r' = R(w_r')$
g) $c_r' \leqslant \sigma t_2$
h) $(m - f_1 - f_r, w_r') \in (\!|\sigma\tau_2|\!)_v$

We can conclude as follows

1. Using a) and e)
$$\frac{\langle T, \delta\ulcorner e_1 \urcorner\rangle \curvearrowright \mathsf{pack}\ w', T_1', c_1' \qquad \langle T_r, \delta\ulcorner e_2 \urcorner[w'/x]\rangle \curvearrowright w_r', T_r', c_r' \qquad v_r' = V(T_r')}{\langle\langle \_, \mathsf{unpack}(T, x, T_r)\rangle, \mathsf{unpack}\ \delta\ulcorner e_1 \urcorner\ \mathsf{as}\ x\ \mathsf{in}\ \delta\ulcorner e_2 \urcorner\rangle \curvearrowright w_r', \langle v_r', \mathsf{unpack}(T', x, T_r')\rangle, c_1' + c_r'}\ \textbf{cp-unpack}$$
2. Using f)
3. By using c) and g), we get $c' + c_r' \leqslant \sigma t_1 + \sigma t_2$
4. By h)

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_1 : \tau_1 \mid t_1 \qquad \Delta; \Phi; x : \tau_1, \Gamma \vdash_{\mathbb{CP}} e_2 : \tau_2 \mid t_2}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \textbf{let}\ x = e_1\ \textbf{in}\ e_2 : \tau_2 \mid t_1 + t_2}\ \textbf{cp-let}$

Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.
TS: $(m, \mathsf{let}\ x = \delta\ulcorner e_1 \urcorner\ \mathsf{in}\ \delta\ulcorner e_2 \urcorner) \in (\!|\sigma\tau_2|\!)_\varepsilon^{\sigma t_1 + \sigma t_2}$.
Following the definition of $(\!|\cdot|\!)_\varepsilon^{\cdot}$, assume that

$$\frac{\begin{array}{c} L(\delta^\ulcorner e_1 {}^\urcorner) \Downarrow^{f_1} T_1 \quad (\diamond) \\ v_1 = V(T_1) \qquad L(\delta^\ulcorner e_2 {}^\urcorner)[v_1/x] \Downarrow^{f_r} T_r \quad (\dagger) \qquad v_r = V(T_r) \end{array}}{\text{let } x = L(\delta^\ulcorner e_1 {}^\urcorner) \text{ in } L(\delta^\ulcorner e_2 {}^\urcorner) \Downarrow^{f_1+f_r+c_{let}} \langle v_r, \texttt{let}(x, T_1, T_r)\rangle}\ \textbf{ev-let} \text{ and}$$

$$\frac{\begin{array}{c} R(\delta^\ulcorner e_1 {}^\urcorner) \Downarrow^{f'_1} T'_1 \quad (\diamond\diamond) \\ v'_1 = V(T'_1) \qquad R(\delta^\ulcorner e_2 {}^\urcorner)[v'_1/x] \Downarrow^{f'_r} T'_r \quad (\dagger\dagger) \qquad v'_r = V(T'_r) \end{array}}{\text{let } x = R(\delta^\ulcorner e_1 {}^\urcorner) \text{ in } R(\delta^\ulcorner e_2 {}^\urcorner) \Downarrow^{f'_1+f'_r+c_{let}} \langle v'_r, \texttt{let}(x, T'_1, T'_r)\rangle}\ \textbf{ev-let} \text{ and}$$

$(f_1 + f_r + c_{let}) < m$.

By IH 1 on the first premise, we get $(m, \delta^\ulcorner e_1 {}^\urcorner) \in (\!|\sigma\tau_1|\!)^{\sigma t_1}_\varepsilon$. Unrolling its definition with $(\diamond)$, $(\diamond\diamond)$ and $f_1 < m$, we get

a) $\langle T_1, \delta^\ulcorner e_1 {}^\urcorner\rangle \curvearrowright w'_1, T'_1, c'_1$
b) $v_1 = L(w'_1) \ \wedge\ v'_1 = R(w'_1)$
c) $c' \leqslant \sigma t_1$
d) $(m - f_1, w'_1) \in (\!|\sigma\tau_1|\!)_v$

By IH 1 on the second premise using $(m - f_1, \delta[x \mapsto w'_1]) \in \mathcal{G}(\!|\sigma\Gamma, x : \sigma\tau_1|\!)$ obtained by

- $(m - f_1, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ by downward closure (Lemma 33) on $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ using $m - f_1 \leqslant m$
- $(m - f_1, w'_1) \in (\!|\sigma\tau_1|\!)_v$ by e)

we get $(m - f_1, \delta^\ulcorner e_2 {}^\urcorner[w'_1/x]) \in (\!|\sigma\tau_2|\!)^{\sigma t_2}_\varepsilon$. Unrolling its definition with $(\dagger)$, $(\dagger\dagger)$ and $f_r < m - f_1$, we get

e) $\langle T_r, \delta^\ulcorner e_2 {}^\urcorner[w'_1/x]\rangle \curvearrowright w'_r, T'_r, c'_r$
f) $v_r = L(w'_r) \ \wedge\ v'_r = R(w'_r)$
g) $c'_r \leqslant \sigma t_2$
h) $(m - f_1 - f_r, w'_r) \in (\!|\sigma\tau_2|\!)_v$

Now, we can conclude with

1. Using a) and e)
$$\frac{\langle T_1, \delta^\ulcorner e_1 {}^\urcorner\rangle \curvearrowright w'_1, T'_1, c'_1 \qquad \langle T_r, \delta^\ulcorner e_2 {}^\urcorner[w'_1/x]\rangle \curvearrowright w'_r, T'_r, c'_r}{\langle \texttt{let}(x, T_1, T_r), \texttt{let } x = \delta^\ulcorner e_1 {}^\urcorner \texttt{ in } \delta^\ulcorner e_2 {}^\urcorner\rangle \curvearrowright w'_r, \texttt{let}(x, T'_1, T'_r), c'_1 + c'_r}\ \textbf{cp-let}$$
2. Using f)

3. By using c) and g), we get $c' + c'_r \leqslant \sigma t_1 + \sigma t_2$
4. By downward closure (Lemma 33) on h) using $m - f_1 - f_r - c_{let} \leqslant m - f_1 - f_r$, we get $(m - (f_1 + f_r + c_{let}), w'_r) \in (\!|\sigma\tau_2|\!)_\nu$

Case: $\dfrac{\Delta; \Phi; |\Gamma| \vdash_{\mathbb{FS}} e : A \mid t}{\Delta; \Phi; \Gamma \vdash_{\mathbb{CP}} e : U\, A \mid t}$ **cp-switch**

Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m, \delta^{\ulcorner} e^{\urcorner}) \in (\!|U\, \sigma A|\!)^{\sigma t}_{\varepsilon}$.

By lemma 38 on $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$, we get

$$(m, \delta) \in \mathcal{G}(\!|U\, |\sigma\Gamma||\!) \tag{1}$$

Then, we can conclude by IH 4 on the premise using eq. (1).

Case: $\dfrac{\Delta; \Phi \models C \qquad \Delta; \Phi \wedge C; \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : C \,\&\, \tau \mid t}$ **cp-c-andI**

Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m, \delta^{\ulcorner} e^{\urcorner}) \in (\!|\sigma C \,\&\, \sigma\tau|\!)^{\sigma t}_{\varepsilon}$.

Following the definition of $(\!|\cdot|\!)^{\cdot}_{\varepsilon} \cdot$, assume that

a) $L(\delta^{\ulcorner} e^{\urcorner}) \Downarrow^{f} T$
b) $R(\delta^{\ulcorner} e^{\urcorner}) \Downarrow^{f'} T'$
c) $f < m$

By IH 1 on the first premise using

- $\models \sigma(C \wedge \Phi)$ hold by the main assumption $\models \sigma\Phi$ and $\models \sigma C$ $(\star)$ obtained by lemma 22 using the premise $\Delta; \Phi \models C$

we get $(m, \delta^{\ulcorner} e^{\urcorner}) \in (\!|\sigma\tau|\!)^{\sigma t}_{\varepsilon}$. Unrolling its definition with (a-c), we get

a) $\langle T, \delta^{\ulcorner} e^{\urcorner} \rangle \curvearrowright w', T', c'$
b) $\nu = L(w'_1) \;\wedge\; \nu' = R(w'_1)$
c) $c' \leqslant \sigma t$
d) $(m - f, w'_1) \in (\!|\sigma\tau|\!)_\nu$

We can conclude as follows:

1. By a)
2. By b)
3. By c)
4. Using d) and $(\star)$, we can show that $(m - f, w_1') \in (\!|\sigma C \,\&\, \sigma\tau|\!)_v$

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_1 : e_1' \mid t_1 \; C \,\&\, \tau_1 \qquad \Delta; \Phi \wedge C; x : \tau_1, \Gamma \vdash_{\mathbb{CP}} e_2 : e_2' \mid t_2 \; \tau_2}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \textbf{clet } e_1 \textbf{ as } x \textbf{ in } e_2 : \tau_2 \mid t_1 + t_2}$ **cp-c-andE**

Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m, \text{clet } \delta\ulcorner e_1 \urcorner \text{ as } x \text{ in } \delta\ulcorner e_2 \urcorner) \in (\!|\sigma\tau_2|\!)_\varepsilon^{\sigma t_1 + \sigma t_2}$.

Following the definition of $(\!|\cdot|\!)_\varepsilon^{\cdot}$, assume that

$$\frac{L(\delta\ulcorner e_1 \urcorner) \Downarrow^{f_1} T_1 \;\; (\diamond) \qquad \nu_1 = V(T_1) \qquad L(\delta\ulcorner e_2 \urcorner)[\nu_1/x] \Downarrow^{f_r} T_r \;\; (\dagger) \qquad \nu_r = V(T_r)}{\text{clet } L(\delta\ulcorner e_1 \urcorner) \text{ as } x \text{ in } L(\delta\ulcorner e_2 \urcorner) \Downarrow^{f_1 + f_r} \langle \nu_r, \mathtt{clet}_{\mathtt{as}}(x, T_1, T_r)\rangle} \textbf{ ev-clet and}$$

$$\frac{R(\delta\ulcorner e_1 \urcorner) \Downarrow^{f_1'} T_1' \;\; (\diamond\diamond) \qquad \nu_1' = V(T_1') \qquad R(\delta\ulcorner e_2 \urcorner)[\nu_1'/x] \Downarrow^{f_r'} T_r' \;\; (\dagger\dagger) \qquad \nu_r' = V(T_r')}{\text{clet } R(\delta\ulcorner e_1 \urcorner) \text{ as } x \text{ in } R(\delta\ulcorner e_2 \urcorner) \Downarrow^{f_1' + f_r'} \langle \nu_r', \mathtt{clet}_{\mathtt{as}}(x, T_1', T_r')\rangle} \textbf{ ev-clet and}$$

$(f_1 + f_r) < m$.

By IH 1 on the first premise, we get $(m, \delta\ulcorner e \urcorner) \in (\!|\sigma C \,\&\, \sigma\tau_1|\!)_\varepsilon^{\sigma t_1}$.

Unrolling its definition with $(\diamond)$, $(\diamond\diamond)$ and $f_1 < m$, we get

a) $\langle T_1, \delta\ulcorner e_1 \urcorner\rangle \curvearrowright w_1', T_1', c_1'$

b) $\nu_1 = L(w_1') \;\wedge\; \nu_1' = R(w_1')$

c) $c' \leqslant \sigma t_1$

d) $(m - f_1, w_1') \in (\!|\sigma C \,\&\, \sigma\tau_1|\!)_v$

By IH 1 on the second premise using $(m - f_1, \delta[x \mapsto w_1']) \in \mathcal{G}(\!|\sigma\Gamma, x : \sigma\tau_1|\!)$ obtained by

- $\models \sigma(C \wedge \Phi)$ hold by the main assumption $\models \sigma\Phi$ and $\models \sigma C$ obtained by unrolling the definition of b)
- $(m - f_1, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ by downward closure (Lemma 33) on $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ using $m - f_1 \leqslant m$
- $(m - f_1, w_1') \in (\!|\sigma\tau_1|\!)_v$ by unrolling the definition of e)

we get $(m - f_1, \delta^\ulcorner e_2 \urcorner[w_1'/x]) \in (\!|\sigma\tau_2|\!)_\varepsilon^{\sigma t_2}$. Unrolling its definition with (†), (††) and $f_r < m - f_1$, we get

e) $\langle T_r, \delta^\ulcorner e_2 \urcorner[w_1'/x]\rangle \curvearrowright w_r', T_r', c_r'$
f) $v_r = L(w_r') \ \wedge \ v_r' = R(w_r')$
g) $c_r' \leqslant \sigma t_2$
h) $(m - f_1 - f_r, w_r') \in (\!|\sigma\tau_2|\!)_v$

Now, we can conclude with

1. Using a) and e)
$$\frac{\langle T_1, \delta^\ulcorner e_1 \urcorner\rangle \curvearrowright w_1', T_1', c_1' \qquad \langle T_r, \delta^\ulcorner e_2 \urcorner[w_1'/x]\rangle \curvearrowright w_r', T_r', c_r'}{\langle \mathtt{clet}_{\mathtt{as}}(x, T_1, T_r), \mathtt{clet}\ \delta^\ulcorner e_1 \urcorner\ \mathtt{as}\ x\ \mathtt{in}\ \delta^\ulcorner e_2 \urcorner\rangle \curvearrowright w_r', \mathtt{clet}_{\mathtt{as}}(x, T_1', T_r'), c_1' + c_r'}\ \textbf{cp-clet}$$
2. Using f)
3. By using c) and g), we get $c' + c_r' \leqslant \sigma t_1 + \sigma t_2$
4. By h)

Case: $\dfrac{\Delta; \Phi \wedge C; \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : C \supset \tau \mid t}$ **cp-c-impI**

Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.
TS: $(m, \delta^\ulcorner e^\urcorner) \in (\!|\sigma C \ \& \ \sigma\tau|\!)_\varepsilon^{\sigma t}$.
Following the definition of $(\!|\cdot|\!)_\varepsilon^{\cdot}\cdot$, assume that

a) $L(\delta^\ulcorner e^\urcorner) \Downarrow^f T$ where $T = \langle v, D\rangle$
b) $R(\delta^\ulcorner e^\urcorner) \Downarrow^{f'} T'$ where $T' = \langle v', D'\rangle$
c) $f < m$

TS1: $\langle\langle v, D\rangle, \text{æ}\rangle \curvearrowright w', T', c'$
TS2: $v' = R(w') \ \wedge \ v = L(w')$
TS3: $c' \leqslant \sigma t$
TS4: $(m - f, w') \in (\!|U\ \sigma A|\!)_v$
We first show the last statement, the previous ones will be shown later.
TS2: $(m - c, w') \in (\!|\sigma C \supset \sigma\tau|\!)_v$
Assume that $\models \sigma C$ $(\star)$.
STS: $(m - c, w') \in (\!|\sigma\ \tau|\!)_v$
By IH 1 on the first premise using

- $\models \sigma(C \wedge \Phi)$ hold by the main assumption $\models \sigma\Phi$ and $\models \sigma C$ (by $\star$)

we get $(m, \delta\ulcorner e \urcorner) \in (\!|\sigma\tau|\!)^{\sigma t}_{\varepsilon}$. Unrolling its definition with (a-c), we get

a) $\langle T, \delta\ulcorner e \urcorner\rangle \curvearrowright w', T', c'$
b) $v = L(w') \ \wedge \ v' = R(w')$
c) $c' \leqslant \sigma t$
d) $(m - f, w') \in (\!|\sigma\tau|\!)_v$

We can conclude as follows:

1. By a)
2. By b)
3. By c)
4. Using d) and $(\star)$, we can show that $(m - f, w') \in (\!|\sigma C \supset \sigma\tau|\!)_v$

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : C \supset \tau \mid t \qquad \Delta; \Phi \models C}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \mathbf{celim}_{\supset}\ e : \tau \mid t}$ **cp-c-implE**

Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.
TS: $(m, \mathrm{celim}_{\supset}\ \delta\ulcorner e \urcorner) \in (\!|\sigma\tau|\!)^{\sigma t}_{\varepsilon}$.
Following the definition of $(\!|\cdot|\!)^{\cdot}_{\varepsilon}\cdot$, assume that
$\dfrac{L(\delta\ulcorner e \urcorner) \Downarrow^{f} T \quad (\diamond)}{\mathrm{celim}_{\supset}\ L(\delta\ulcorner e \urcorner) \Downarrow^{f} T}$ **ev-celim** and $\dfrac{R(\delta\ulcorner e \urcorner) \Downarrow^{f'} T' \quad (\diamond\diamond)}{\mathrm{celim}_{\supset}\ R(\delta\ulcorner e \urcorner) \Downarrow^{f'} T'}$ **ev-celim**
and $f < m$ $(\star)$.
By IH 1 on the first premise, we get $(m, \delta\ulcorner e \urcorner) \in (\!|\sigma C \supset \sigma\tau|\!)^{\sigma t}_{\varepsilon}$.
Unrolling its definition using $(\diamond)$, $(\diamond\diamond)$ and $(\star)$, we get

a) $\langle T, \delta\ulcorner e \urcorner\rangle \curvearrowright w', T', c'$
b) $v = L(w') \ \wedge \ v' = R(w')$
c) $c' \leqslant \sigma t$
d) $(m - c, w') \in (\!|\sigma C \supset \sigma\tau|\!)_v$

We can conclude as follows:

1. By a)
2. By b)

3. By c)
4. Using d) and $\models \sigma C$ (obtained by lemma 22 on the second premise), we can show that $(m - f, w') \in (\!|\sigma\tau|\!)_v$

Case: $\dfrac{\Upsilon(\zeta) = \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2 \qquad \Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau_1 \mid t'}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \zeta\ e : \tau_2 \mid t + t'}$ **cp-primapp**

Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m, \zeta\ \delta\ulcorner e \urcorner) \in (\!|\sigma\tau_2|\!)_\varepsilon^{\sigma t + \sigma t'}$.

Following the definition of $(\!|\cdot|\!)_\varepsilon^{:}$, assume that

$\dfrac{L(\delta\ulcorner e\urcorner) \Downarrow^{f} T \quad (\star) \qquad v = V(T) \qquad \zeta(v) = (f_r, v_r) \quad (\diamond)}{\zeta\ L(\delta\ulcorner e\urcorner) \Downarrow^{f + f_r + c_{primapp}} \langle v_r, \mathtt{prim}_{\mathtt{app}}(T, \zeta)\rangle}$ **ev-primapp** and

$\dfrac{R(\delta\ulcorner e\urcorner) \Downarrow^{f'} T' \quad (\star\star) \qquad v' = V(T') \qquad \zeta(v') = (f'_r, v'_r) \quad (\diamond\diamond)}{\zeta\ R(\delta\ulcorner e\urcorner) \Downarrow^{f' + f'_r + c'_{primapp}} \langle v'_r, \mathtt{prim}_{\mathtt{app}}(T', \zeta)\rangle}$ **ev-primapp**

and

$(f + f_r + c_{primapp}) < m$.

By IH 1 on the second premise, we get $(m, \delta\ulcorner e\urcorner) \in (\!|\sigma\tau_1|\!)_\varepsilon^{\sigma t'}$.

Unrolling its definition with $(\star)$, $(\star\star)$ and $f < m$, we get

a) $\langle T, \delta\ulcorner e\urcorner\rangle \curvearrowright w', T', c'$
b) $v = L(w') \ \wedge\ v' = R(w')$
c) $c' \leqslant \sigma t$
d) $(m - f, w') \in (\!|\sigma\tau_1|\!)_v$

Next, by Assumption (assumption 44) using $\zeta : \sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2$ (obtained by substitution on the first premise), d) and $(\diamond)$, $(\diamond\diamond)$, we get

e) $f'_r \leqslant \sigma t'$
f) $(m - f - f_r, \mathtt{merge}(v_r, v'_r)) \in (\!|\sigma\tau_2|\!)_v$

Now, we can conclude as follows:

1. Using a) and e)

$\dfrac{\langle T, \delta\ulcorner e\urcorner\rangle \curvearrowright w', T', c' \qquad v' = V(T') \qquad (f'_r, v'_r) = \zeta(v')}{\langle\langle v_r, \mathtt{prim}_{\mathtt{app}}(T, \zeta)\rangle, \zeta\ \delta\ulcorner e\urcorner\rangle \curvearrowright \mathtt{merge}(v_r, v'_r), \langle v'_r, \mathtt{prim}_{\mathtt{app}}(T', \zeta)\rangle, c' + f'_r}$ **cp-prim**

2. By definition $v_r = L(\mathtt{merge}(v_r, v_r')) \ \wedge \ v_r' = R(\mathtt{merge}(v_r, v_r'))$
3. By using c) and e), we get $c' + f_r' \leqslant \sigma t + \sigma t'$
4. By downward closure (Lemma 33) on f) using

$$m - (f + f_r + c_{primapp}) \leqslant m - (f + f_r)$$

we get $(m - (f + f_r + c_{primapp}), \mathtt{merge}(v_r, v_r')) \in (\!|\sigma\tau_2|\!)_v$.

Case: $$\frac{\begin{array}{c}\Delta; \Phi; \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t \\ \forall x \in dom(\Gamma).\ \Delta; \Phi \models \Gamma(x) \sqsubseteq \Box\Gamma(x)\end{array}}{\Delta; \Phi; \Gamma, \Gamma' \vdash_{\mathbb{CP}} e : \Box\tau \mid 0}$$ **cp-nochange**

Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma, \sigma\Gamma'|\!)$ and $\models \sigma\Phi$.
Then, $\delta = \delta_1 \cup \delta_2$ such that $(m, \delta_1) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $(m, \delta_2) \in \mathcal{G}(\!|\sigma\Gamma'|\!)$.
TS: $(m, \delta\ulcorner e \urcorner) \in (\!|\Box\,\sigma\tau|\!)^0_\varepsilon$.
Since e doesn't have any free variables from Γ' by the first premise,
STS: $(m, \delta_1\ulcorner e \urcorner) \in (\!|\Box\,\sigma\tau|\!)^0_\varepsilon$.
Assume that

a) $L(\delta\ulcorner e \urcorner) \Downarrow^f T$
b) $R(\delta\ulcorner e \urcorner) \Downarrow^{f'} T'$
c) $f < m$.

By IH 1 on the first premise using

- $(m, \delta_1) \in \mathcal{G}(\!|\sigma\Gamma|\!)$
- $\models \sigma\Phi$

we get $(m, \delta_1\ulcorner e \urcorner) \in (\!|\sigma\tau|\!)^{\sigma t}_\varepsilon$.
Unfolding its definition with (a-c), we get

a) $\langle T, \delta\ulcorner e \urcorner\rangle \curvearrowright w', T', c'$
b) $v = L(w') \ \wedge \ v' = R(w')$
c) $c' \leqslant \sigma t$
d) $(m - f, w') \in (\!|\sigma\tau|\!)_v$

We can conclude as follows:

1. By a)
2. By b)
3. By lemma 39 using $(m, \delta_1) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and the second premise, we get $(m, \delta_1) \in \mathcal{G}(\!|\Box\,\sigma\Gamma|\!)$. This means that $\forall x \in \mathrm{dom}(\Gamma).\ \mathtt{stable}(\delta(x))$. Therefore, $\mathtt{stable}(\delta\ulcorner e \urcorner)$. Hence, by lemma 35, we have $c' = 0$ and $\mathtt{stable}(w')$ $(\star)$.
4. By d) and $(\star)$ obtained above, we get$(m - c, w') \in (\!|\Box\,\sigma\tau|\!)_v$.

Case: $\dfrac{\Delta; \Phi \wedge C; \Gamma \vdash_{\mathbb{CP}} e_1 : \tau \mid t \qquad \Delta; \Phi \wedge \neg C; \Gamma \vdash_{\mathbb{CP}} e_1 : \tau \mid t \qquad \Delta \vdash C\ \mathtt{wf}}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_1 : \tau \mid t}$ **cp-split**

Assume that $\models \sigma\Phi$ and $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$.

TS: $(m, \delta\ulcorner e \urcorner) \in (\!|\sigma\tau|\!)^{\sigma k}_{\varepsilon}$.

There are two cases:

subcase 1: $\models \sigma\Phi \wedge C$

Follows immediately by IH on the first premise.

subcase 2: $\models \sigma\Phi \wedge \neg C$

Follows immediately by IH on the second premise.

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t \qquad \Delta; \Phi \models \tau \sqsubseteq \tau' \qquad \Delta; \Phi \models t \leqslant t'}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau' \mid t'}$ **cp-$\sqsubseteq$**

Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m, \delta\ulcorner e \urcorner) \in (\!|\sigma\tau'|\!)^{\sigma t'}_{\varepsilon}$.

Following the definition of $(\!|\cdot|\!)^{\cdot}_{\varepsilon}$, assume that

a) $L(\delta\ulcorner e \urcorner) \Downarrow^{f} T$
b) $R(\delta\ulcorner e \urcorner) \Downarrow^{f'} T'$
c) $f < m$

By IH 1 on the first premise using we get $(m, \delta\ulcorner e \urcorner) \in (\!|\sigma\tau|\!)^{\sigma t}_{\varepsilon}$.
Unrolling its definition with (a-c), we get

d) $\langle T, \delta\ulcorner e \urcorner\rangle \curvearrowright w', T', c'$
e) $v = L(w') \ \wedge\ v' = R(w')$
f) $c' \leqslant \sigma t$

g) $(m - f, w') \in (\!|\sigma\tau|\!)_v$

We can conclude as follows:

1. By d)
2. By e)
3. By Assumption (assumption 25) on the third premise, we get $\sigma t \leqslant \sigma t'$. Combining this with f), we get $c' \leqslant \sigma t'$.
4. By lemma 39 on the second premise with g), we get $(m - c, w') \in (\!|\sigma\tau'|\!)_v$

□

Proof of Statement (2). Remember the statement (2) of Theorem 46:
Assume that $\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A \mid t$ and $\sigma \in \mathcal{D}[\![\Delta]\!]$ and $\models \sigma\Phi$ and $(m, \gamma) \in \mathcal{G}[\![\sigma\Omega]\!]$. Then, $(m, \gamma e) \in [\![\sigma A]\!]_\varepsilon^{\sigma t}$.

Proof is by induction on the typing of e. We show a few selected cases.

Case: $\dfrac{\Omega(x) = A}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} x : A \mid 0}$ **fs-var**

Assume that $\models \sigma\Phi$ and $(m, \gamma) \in \mathcal{G}[\![\sigma\Omega]\!]$.
TS: $(m, \gamma(x)) \in [\![\sigma A]\!]_\varepsilon^0$.
By Value Lemma (lemma 31),
STS: $(m, \gamma(x)) \in [\![\sigma A]\!]_v$.
This follows by $\Omega(x) = A$ and $(m, \gamma) \in \mathcal{G}(\!|\sigma\Omega|\!)$

Case: $\dfrac{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e_1 : A \mid t_1 \qquad \Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e_2 : \mathbf{list}[n]\, A \mid t_2}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} \mathbf{cons}(e_1, e_2) : \mathbf{list}[n+1]\, A \mid t_1 + t_2}$ **fs-cons**

Assume that $\models \sigma\Phi$ and $(m, \gamma) \in \mathcal{G}[\![\sigma\Omega]\!]$.
TS: $(m, \mathrm{cons}(\gamma e_1, \gamma e_2)) \in [\![\mathrm{list}[\sigma n + 1]\, \sigma A]\!]_\varepsilon^{\sigma t_1 + \sigma t_2}$.
Following the definition of $[\![\cdot]\!]_\varepsilon^{\cdot}$,
Assume that

$$\frac{\gamma e_1 \Downarrow^{f_1} T_1 \quad (\star) \qquad \gamma e_2 \Downarrow^{f_2} T_2 \quad (\diamond) \qquad v_i = \mathsf{V}(T_i)}{\mathrm{cons}(\gamma e_1, \gamma e_2) \Downarrow^{f_1 + f_2} \langle \mathrm{cons}(v_1, v_2), \mathsf{cons}(T_1, T_2) \rangle}\ \textbf{ev-cons}$$

and $f_1 + f_2 < m$.
By IH 2 on the first premise, we get $(m, \gamma e_1) \in [\![\sigma A]\!]_\varepsilon^{\sigma t_1}$.
Unrolling its definition with $(\star)$ and $f_1 < m$, we get

a) $f_1 \leqslant \sigma t_1$
b) $(m - f_1, v_1) \in [\![\sigma A]\!]_v$

By IH 2 on the second premise,we get $(m, \gamma e_2) \in [\![\mathrm{list}[\sigma n]\ \sigma A]\!]_\varepsilon^{\sigma t_2}$.
Unrolling its definition with $(\diamond)$ and $f_2 < m$, we get

c) $f_2 \leqslant \sigma t_2$
d) $(m - f_2, v_2) \in [\![\mathrm{list}[\sigma n]\ \sigma A]\!]_v$

Now, we can conclude as follows:

1. Using a) and c), we get $(f_1 + f_2) \leqslant \sigma t_1 + \sigma t_2$
2. By downward closure (Lemma 33) on b) using

$$m - (f_1 + f_2) \leqslant m - f_1$$

we get $(m - (f_1 + f_2), v_1) \in [\![\sigma A]\!]_v$.
By downward closure (Lemma 33) on d) using

$$m - (f_1 + f_2) \leqslant m - f_2$$

we get $(m - (f_1 + f_2), v_2) \in [\![\mathrm{list}[\sigma n]\ \sigma A]\!]_v$.
By combining these two statements, we can conclude as $(m - (f_1 + f_2), \mathrm{cons}(v_1, v_2)) \in [\![\mathrm{list}[\sigma n + 1]\ \sigma A]\!]_v$

Case:
$$\frac{\Delta; \Phi; x : A_1, f : A_1 \xrightarrow{\mathbb{FS}(t)} A_2, \Omega \vdash_{\mathbb{FS}} e : A_2 \mid t \qquad \Delta; \Phi \vdash^{A} A_1 \xrightarrow{\mathbb{FS}(t)} A_2\ \mathtt{wf}}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} \mathbf{fix}\ f(x).e : A_1 \xrightarrow{\mathbb{FS}(t)} A_2 \mid 0}\ \textbf{fs-fix}$$

Assume that $\models \sigma\Phi$ and $(m, \gamma) \in \mathcal{G}[\![\sigma\Omega]\!]$.
TS: $(m, \mathrm{fix}\ f(x).\gamma e) \in [\![\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2]\!]_\varepsilon^0$.

By lemma 31, STS: $(m, \text{fix } f(x).\gamma e) \in [\![\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2]\!]_v$.
We prove the more general statement

$$\forall m' \leqslant m.\ (m', \text{fix } f(x).\gamma e) \in [\![\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2]\!]_v$$

by subinduction on m'.
There are two cases:

subcase 1: $m' = 0$
Since there is no non-negative j such that $j < 0$, the goal is vacuously true.

subcase 2: $m' = m'' + 1 \leqslant m$
By sub-IH

$$(m'', \text{fix } f(x).\gamma e) \in [\![\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2]\!]_v \tag{1}$$

STS: $(m'' + 1, \text{fix } f(x).\gamma e) \in [\![\sigma A_1 \xrightarrow{\mathbb{FS}(t)} \sigma A_2]\!]_v$.
Pick $j < m'' + 1$ and assume that $(j, v) \in [\![\sigma A_1]\!]_v$.
STS: $(j, \gamma e[v/x, (\text{fix } f(x).\gamma e)/f]) \in [\![\sigma A_2]\!]_\varepsilon^{\sigma t}$.
This follows by IH on the premise instantiated with

- $(j, \gamma[x \mapsto v, f \mapsto (\text{fix } f(x).\gamma e)]) \in \mathcal{G}[\![\sigma\Omega', x : \sigma A_1, f : \sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2]\!]$ which holds because
  - $(j, \gamma) \in \mathcal{G}[\![\sigma\Omega']\!]$ obtained by downward closure (Lemma 33) on $(m, \gamma) \in \mathcal{G}[\![\sigma\Omega']\!]$ using $j < m'' + 1 \leqslant m$.
  - $(j, v) \in [\![\sigma A_1]\!]_v$, from the assumption above
  - $(j, \text{fix } f(x).\gamma e) \in [\![\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2]\!]_v$, obtained by downward closure (Lemma 33) on (1) using $j \leqslant m''$

Case: $\dfrac{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e_1 : A_1 \xrightarrow{\mathbb{FS}(t)} A_2 \mid t_1 \qquad \Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e_2 : A_1 \mid t_2}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e_1\ e_2 : A_2 \mid t_1 + t_2 + t + c_{app}}$ **fs-app**

Assume that $\models \sigma\Phi$ and $(m, \gamma) \in \mathcal{G}[\![\sigma\Omega]\!]$.
TS: $(m, \gamma e_1\ \gamma e_2) \in [\![\sigma A_2]\!]_\varepsilon^{\sigma t_1 + \sigma t_2 + \sigma t + c_{app}}$.
Following the definition of $[\![\cdot]\!]_\varepsilon^{\cdot}$, there are two cases: Assume that

$$\frac{\gamma e_1 \Downarrow^{f_1} T_1 \quad (\star) \qquad \gamma e_2 \Downarrow^{f_2} T_2 \quad (\diamond) \qquad \text{fix } f(x).e = \mathsf{V}(T_1) \qquad v_2 = \mathsf{V}(T_2) \qquad e[v_2/x, (\text{fix } f(x).e)/f] \Downarrow^{f_r} T_r \quad (\dagger) \qquad v_r = \mathsf{V}(T_r)}{\gamma e_1\ \gamma e_2 \Downarrow^{f_1+f_2+f_r+c_{app}} \langle v_r, \mathsf{app}(T_1, T_2, T_r)\rangle}\ \textbf{ev-app}$$

and $f_1 + f_2 + f_r + c_{app} < m$.
By IH 2 on the first premise, we get $(m, \gamma e_1) \in [\![\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2]\!]_\varepsilon^{\sigma t_1}$.
Unrolling its definition with $(\star)$ and $f_1 < m$, we get

a) $f_1 \leqslant \sigma t_1$
b) $(m - f_1, \text{fix } f(x).e) \in [\![\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2]\!]_v$

By IH 2 on the second premise, we get $(m, \gamma e_2) \in [\![\sigma A_1]\!]_\varepsilon^{\sigma t_2}$. Unrolling its definition with $(\diamond)$ and $f_2 < m$, we get

c) $f_2 \leqslant \sigma t_2$
d) $(m - f_2, v_2) \in [\![\sigma A_1]\!]_v$

By downward closure (Lemma 33) on d) using $m - f_1 - f_2 - c_{app} \leqslant m - f_2$, we get

$$(m - (f_1 + f_2 + c_{app}), v_2) \in [\![\sigma A_1]\!]_v \tag{1}$$

Next, we unroll b) with (1) and $m - (f_1 + f_2 + c_{app}) < m - f_1$ (note that $0 < c_{app}$) to obtain

$$(m - (f_1 + f_2 + c_{app}), e[v_2/x, (\text{fix } f(x).e)]) \in [\![\sigma A_2]\!]_\varepsilon^{\sigma t} \tag{2}$$

By unrolling (2)'s definition using $(\dagger)$ and $f_r < m - (f_1 + f_2 + c_{app})$ (note that $0 < c_{app}$), we get

e) $f_r \leqslant \sigma t$
f) $(m - (f_1 + f_2 + f_r + c_{app}), v_r) \in [\![\sigma A_2]\!]_v$

Now, we can conclude as follows:

1. Using a), c) and e), we get $(f_1 + f_2 + f_r + c_{app}) \leqslant \sigma t_1 + \sigma t_2 + \sigma t + c_{app}$

2. By f)

Case: $\dfrac{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A_1 \mid t \qquad \Delta; \Phi \vdash^{A} A_2\ \mathtt{wf}}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} \mathbf{inl}\ e : A_1 + A_2 \mid t}$ **fs-inl**

Assume that $\models \sigma\Phi$ and $(m, \gamma) \in \mathcal{G}[\![\sigma\Omega]\!]$.

TS: $(m, \mathrm{inl}\ (\gamma e)) \in [\![\sigma A_1 + \sigma A_2]\!]^{\sigma t}_{\varepsilon}$.

Following the definition of $[\![\cdot]\!]^{\cdot}_{\varepsilon}$, assume that

$\dfrac{\gamma e \Downarrow^{f} T \ \ (\star) \qquad v = \mathtt{V}(T)}{\mathrm{inl}\ \gamma e \Downarrow^{f} \langle \mathrm{inl}\ v, \mathtt{inl}\ T \rangle}$ **e-inl** and $f < m$.

By IH 2 on the first premise, we get $(m, \gamma e) \in [\![\sigma A]\!]^{\sigma t}_{\varepsilon}$.

Unrolling its definition with $(\star)$ and $f < m$, we get

a) $f \leqslant \sigma t$

b) $(m - f, v) \in [\![\sigma A]\!]_v$

We can conclude as follows:

1. By a), $f \leqslant \sigma t$
2. By b), we can show that $(m - f, \mathrm{inl}\ v) \in [\![\sigma A_1 + \sigma A_2]\!]_v$

Case: $\dfrac{\begin{array}{c}\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A_1 + A_2 \mid t \\ \Delta; \Phi; x : A_1, \Omega \vdash_{\mathbb{FS}} e_1 : A \mid t' \qquad \Delta; \Phi; y : A_2, \Omega \vdash_{\mathbb{FS}} e_2 : A \mid t'\end{array}}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}}\ \mathbf{case}\ (e, x.e_1, y.e_2) : A \mid t + t' + c_{case}}$ **fs-case**

Assume that $\models \sigma\Phi$ and $(m, \gamma) \in \mathcal{G}[\![\sigma\Omega]\!]$.

TS: $(m,\ \mathrm{case}\ (\gamma e, \gamma e_1, \gamma e_2)) \in [\![\sigma A]\!]^{\sigma t + \sigma t' + c_{case}}_{\varepsilon}$.

Following the definition of $[\![\cdot]\!]^{\cdot}_{\varepsilon}$, assume that

$\dfrac{\gamma e \Downarrow^{f} T \ \ (\star) \qquad \mathrm{inl}\ v = \mathtt{V}(T) \qquad \gamma e_1[v/x] \Downarrow^{f_r} T_r \ \ (\diamond) \qquad v_r = \mathtt{V}(T_r)}{\mathrm{case}\ (\gamma e, x.\gamma e_1, y.\gamma e_2) \Downarrow^{f + f_r + c_{case}} \langle v_r, \mathtt{case}_{\mathtt{inl}}(T, T_r) \rangle}$ **ev-case-l**

and

$f + f_r + c_{case} < m$.

By IH 2 on the first premise, we get $(m, \gamma e) \in [\![\sigma A_1 + \sigma A_2]\!]^{\sigma t}_{\varepsilon}$.

Unrolling second part of its definition with $(\star)$ and $f < m$, we get

a) $f \leqslant \sigma r$

b) $(m - f, \mathrm{inl}\ v) \in [\![\sigma A_1 + \sigma A_2]\!]_v$

By IH 2 on the second premise using $(m - f, \gamma[x \mapsto v]) \in \mathcal{G}[\![\sigma\Omega', x : \sigma A_1]\!]$ obtained by

- $(m-f), \gamma) \in \mathcal{G}[\![\sigma\Omega']\!]$ by downward-closure (lemma 33) on $(m, \gamma) \in \mathcal{G}[\![\sigma\Omega']\!]$ using $m - f \leqslant m$
- $(m - f, v) \in [\![\sigma A_1]\!]_v$ by downward closure (lemma 33) on c), and unfolding its definition

we get

$$(m - f, \gamma e_1[v/x]) \in [\![\sigma A]\!]_\varepsilon^{\sigma t'} \tag{1}$$

By unrolling (1)'s definition using ($\diamond$) and $f_r < m - f$, we get

c) $f_r \leqslant \sigma t'$

d) $(m - (f + f_r), v_r) \in [\![\sigma A]\!]_v$

Now, we can conclude as follows

1. By a) and c) $(f + f_r + c_{case}) \leqslant \sigma t + \sigma t' + c_{case}$
2. By downward closure (Lemma 33) on d) using

$$m - (f + f_r + c_{case}) \leqslant m - (f + f_r)$$

we get $(m - (f + f_r + c_{case}), v_r) \in [\![\sigma A]\!]_v$.

Case:
$$\frac{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : \mathbf{list}[n]\ A \mid t \qquad \Delta; \Phi \wedge n = 0; \Omega \vdash_{\mathbb{FS}} e_1 : A' \mid t' \qquad i, \Delta; \Phi \wedge n = i + 1; h : A, tl : \mathbf{list}[i]\ A, \Omega \vdash_{\mathbb{FS}} e_2 : A' \mid t'}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} \mathbf{case}\ e\ \mathbf{of\ nil}\ \to e_1 \mid h :: tl \to e_2 : A' \mid t + t' + c_{caseL}}\ \textbf{fs-caseL}$$

Assume that $\models \sigma\Phi$ and $(m, \gamma) \in \mathcal{G}[\![\sigma\Omega]\!]$.

TS: $(m,\ \text{case}\ \gamma e\ \text{of nil}\ \to \gamma e_1 \mid h :: tl \to \gamma e_2) \in [\![\sigma A']\!]_\varepsilon^{\sigma t + \sigma t' + c_{caseL}}$.

Following the definition of $[\![\cdot]\!]_\varepsilon^:$, assume that

$\text{case}\ \gamma e\ \text{of nil}\ \to \gamma e_1 \mid h :: tl \to \gamma e_2 \Downarrow^{v_r} F$ and $F < m$.

Depending on what γe evaluates to, there are two cases.

subcase 1:

$$\frac{\gamma e \Downarrow^{f} T \quad (\star) \qquad \gamma e_1 \Downarrow^{f_r} T_1 \quad (\diamond) \qquad \mathsf{nil} = \mathsf{V}(T) \qquad v_r = \mathsf{V}(T_r)}{\mathsf{case}\ \gamma e\ \mathsf{of}\ \mathsf{nil} \to \gamma e_1 \mid h :: tl \to \gamma e_2 \Downarrow^{f+f_r+c_{caseL}} \langle v_r, \mathsf{case}_{\mathsf{nil}}(T, T_r)\rangle}\ \textbf{ev-case-nil}$$

and $F = f + f_r + c_{caseL} < m$.

By IH 2 on the first premise, we get $(m, \gamma e) \in [\![\mathsf{list}[\sigma n]\ \sigma A]\!]^{\sigma t}_{\varepsilon}$.

Unrolling its definition with $(\star)$ and $f < m$, we get

a) $f \leqslant \sigma t$

b) $(m - f, \mathsf{nil}\ ) \in [\![\mathsf{list}[\sigma n]\ \sigma A]\!]_{v}$

By b), $\sigma n = 0$ since $v = \mathsf{nil}$.

Then, we can instantiate IH 2 on the second premise using $\models \sigma\Phi \wedge \sigma n \doteq 0$ obtained by combining $\models \sigma\Phi$ with $\models \sigma n \doteq 0$, we get $(m, \gamma e_1) \in [\![\sigma A']\!]^{\sigma t'}_{\varepsilon}$.

Unrolling its definition using $(\diamond)$ and $f_r < m$, we get

c) $f_r \leqslant \sigma t'$

d) $(m - f_r, v_r) \in [\![\sigma A']\!]_{v}$

We conclude with

1. By a) and c), we get $f + f_r + c_{caseL} \leqslant \sigma t + \sigma t' + c_{caseL}$
2. By downward closure (Lemma 33) on d) using

$$m - (f + f_r + c_{caseL}) \leqslant m - (f + f_r)$$

we get $(m - (f + f_r + c_{caseL}), v_r) \in [\![\sigma A']\!]_{v}$.

subcase 2:

$$\frac{\begin{array}{c}\gamma e \Downarrow^{f} T \quad (\star) \qquad \mathsf{cons}(v_h, v_{tl}) = \mathsf{V}(T) \\ \gamma e_2[v_h/h, v_{tl}/tl] \Downarrow^{f_r} T_r \quad (\diamond\diamond) \qquad v_r = \mathsf{V}(T_r)\end{array}}{\mathsf{case}\ \gamma e\ \mathsf{of}\ \mathsf{nil} \to \gamma e_1 \mid h :: tl \to \gamma e_2 \Downarrow^{f+f_r+c_{caseL}} \langle v_r, \mathsf{case}_{\mathsf{cons}}(T, T_r)\rangle}\ \textbf{ev-case-cons}$$

By IH 2 on the first premise, we get $(m, \gamma e) \in [\![\mathsf{list}[\sigma n]\ \sigma A]\!]^{\sigma t}_{\varepsilon}$.

Unrolling its definition with $(\star)$ and $f < m$, we get

a) $f \leqslant \sigma t$

b) $(m - f, \text{cons}(v_1, v_2)) \in [\![\text{list}[\sigma n]\ \sigma A]\!]_v$

By b), $\sigma n = I + 1$ for some I and we have

$$(m - f, v_1) \in [\![\sigma A]\!]_v \tag{1}$$

$$(m - f, v_2) \in [\![\text{list}[I]\ \sigma A]\!]_v \tag{2}$$

Then, we can instantiate IH 2 on the third premise using

- $\sigma[i \mapsto I] \in \mathcal{D}[\![i :: \mathbb{N}, \Delta]\!]$
- $\models \sigma[i \mapsto I](\Phi \wedge n \doteq i + 1)$ obtained by combining $\models \sigma\Phi$ with $\models \sigma n \doteq I + 1$,
- $(m - f, \gamma[h \mapsto v_1, tl \mapsto v_2]) \in \mathcal{G}[\![\sigma[i \mapsto I](\Omega', x : A, tl : \text{list}[i]\ A)]\!]$ using (1) and (2) and $(m - f, \gamma) \in \mathcal{G}[\![\sigma\Omega']\!]$ (obtained by downward closure (Lemma 33)).

we get $(m, \gamma e_2[v_1/h, v_2/tl]) \in [\![\sigma[i \mapsto I]A]\!]_\varepsilon^{\sigma[i \mapsto I]k'}\ \sigma[i \mapsto I]t'$.
Since, $i \notin FV(k', t', A, A')$, we have
$(m, \gamma e_2[v_1/h, v_2/tl]) \in [\![\sigma A']\!]_\varepsilon^{\sigma t'}$.
Unrolling its definition using ($\diamond\diamond$) and $f_r < m - f$, we get

c) $f_r \leqslant \sigma t'$

d) $(m - f - f_r, v_r) \in [\![\sigma A']\!]_v$

We conclude with

1. By a) and c), we get $f + f_r + c_{caseL} \leqslant \sigma t + \sigma t' + c_{caseL}$
2. By downward closure (Lemma 33) on d) using

$$m - (f + f_r + c_{caseL}) \leqslant m - (f + f_r)$$

we get $(m - (f + f_r + c_{caseL}), v_r) \in (\!|\sigma A'|\!)_v$.

Case: $\dfrac{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e_1 : A_1 \mid t_1 \qquad \Delta; \Phi; x : A_1, \Omega \vdash_{\mathbb{FS}} e_2 : A_2 \mid t_2}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} \textbf{let } x = e_1 \textbf{ in } e_2 : A_2 \mid t_1 + t_2 + c_{let}}$ **fs-let**

Assume that $\models \sigma\Phi$ and $(m, \gamma) \in \mathcal{G}[\![\sigma\Omega]\!]$.

TS: $(m, \text{let } x = \gamma e_1 \text{ in } \gamma e_2) \in [\![\sigma A_2]\!]_\varepsilon^{\sigma t_1 + \sigma t_2 + c_{let}}$.

Following the definition of $[\![\cdot]\!]_\varepsilon$, assume that

$\dfrac{\gamma e_1 \Downarrow^{f_1} T_1 \quad (\star) \qquad v_1 = \mathsf{V}(T_1) \qquad \gamma e_2[v_1/x] \Downarrow^{f_r} T_r \quad (\diamond) \qquad v_r = \mathsf{V}(T_r)}{\text{let } x = \gamma e_1 \text{ in } \gamma e_2 \Downarrow^{f_1 + f_r + c_{let}} \langle v_r, \mathtt{let}(x, T_1, T_r)\rangle}$ **ev-let**

and $f_1 + f_r + c_{let} < m$.

By IH 2 on the first premise, we get $(m, \gamma e_1) \in [\![\sigma A_1]\!]_\varepsilon^{\sigma t_1}$.

Unrolling its definition with $(\star)$ and $f_1 < m$, we get

a) $f_1 \leqslant \sigma t_1$

b) $(m - f_1, v_1) \in [\![\sigma A_1]\!]_v$

By IH 2 on the second premise using $(m - f_1, \gamma[x \mapsto v]) \in \mathcal{G}[\![\sigma\Omega', x : \sigma A_1]\!]$ obtained by

- $(m - f_1, \gamma) \in \mathcal{G}[\![\sigma\Omega']\!]$ by downward closure (Lemma 33) on $(m, \gamma) \in \mathcal{G}[\![\sigma\Omega']\!]$ using $m - f_1 \leqslant m$
- $(m - f_1, v) \in [\![\sigma A_1]\!]_v$ by downward closure (Lemma 33) on c)

we get

$$(m - f_1, \gamma e_1[v/x]) \in [\![\sigma A_2]\!]_\varepsilon^{\sigma t_2} \tag{1}$$

Unrolling (1)'s definition using $(\diamond)$ and $f_r < m - f_1$, we get

c) $f_r \leqslant \sigma t_2$

d) $(m - (f_1 + f_r), v_r) \in [\![\sigma A]\!]_v$

Now, we can conclude as follows

1. By a) and c) $(f_1 + f_r + c_{let}) \leqslant \sigma t_1 + \sigma t_2 + c_{let}$
2. By downward closure (Lemma 33) on d) using

$$m - (f_1 + f_r + c_{let}) \leqslant m - (f_1 + f_r)$$

we get $(m - (f_1 + f_r + c_{let}), v_r) \in (\!|\sigma A|\!)_v$.

Case: $\dfrac{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A\{I/i\} \mid t \qquad \Delta \vdash I :: S}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} \mathbf{pack}\ e : \exists i{::}S.\, A \mid t}$ **fs-pack**

Assume that $\models \sigma\Phi$ and $(m, \gamma) \in \mathcal{G}[\![\sigma\Omega]\!]$.

TS: $(m, \mathsf{pack}\ \gamma e) \in [\![\exists i{::}S.\, A]\!]^{\sigma t}_{\varepsilon}$.

Following the definition of $[\![\cdot]\!]^{\cdot}_{\varepsilon}$, assume that

$\dfrac{\gamma e \Downarrow^{f} T \quad (\star) \qquad v = \mathsf{V}(T)}{\mathsf{pack}\ \gamma e \Downarrow^{f} \langle \mathsf{pack}\ v, \mathsf{pack}\ T \rangle}$ **ev-pack** and $f < m$.

By IH 2 on the first premise, we get $(m, \gamma e) \in [\![\sigma A\{\sigma I/i\}]\!]^{\sigma t}_{\varepsilon}$.

Unrolling its definition with $(\star)$ and $f < m$, we get

a) $f \leqslant \sigma t$

b) $(m - f, v) \in [\![\sigma A\{\sigma I/i\}]\!]_v$

Then we can conclude as follows:

1. By a), $f \leqslant \sigma t$
2. TS: $(m - f, \mathsf{pack}\ v) \in [\![\exists i{::}S.\, A]\!]_v$.
   By lemma 22 on the second premise we know that $\vdash \sigma I :: S$.
   STS: $(m - f, v) \in [\![\sigma A\{\sigma I/i\}]\!]_v$.
   This follows by b).

Case: $\dfrac{\Upsilon(\zeta) = A_1 \xrightarrow{\mathbb{FS}(t)} A_2 \qquad \Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A_1 \mid t'}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} \zeta\ e : A_2 \mid t + t' + c_{primapp}}$ **fs-primapp**

Assume that $\models \sigma\Phi$ and $(m, \gamma) \in \mathcal{G}[\![\sigma\Omega]\!]$.

TS: $(m, \zeta\ \gamma e) \in [\![\sigma A_2]\!]^{\sigma t + \sigma t' + c_{primapp}}_{\varepsilon}$.

Following the definition of $[\![\cdot]\!]^{\cdot}_{\varepsilon}$, assume that

$\dfrac{\gamma e \Downarrow^{f} T \quad (\star) \qquad v = \mathsf{V}(T) \qquad \zeta(v) = (f_r, v_r) \quad (\diamond)}{\zeta\ \gamma e \Downarrow^{f + f_r + c_{primapp}} \langle v_r, \mathsf{prim}_{\mathsf{app}}(T, \zeta) \rangle}$ **ev-primapp**

$f + f_r + c_{primapp} < m$.

By IH 2 on the second premise, we get $(m, \gamma e) \in [\![\sigma A_1]\!]^{\sigma t'}_{\varepsilon}$.

Unrolling its definition with $f < m$, we get

a) $f \leqslant \sigma t'$

b) $(m - f, v) \in [\![\sigma A_1]\!]_v$

Next, by Assumption (assumption 45) using $\zeta : \sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2$ (obtained by substitution on the first premise), ($\diamond$) and (b), we get

c) $f_r \leqslant \sigma t$

d) $(m - f - f_r, v_r) \in [\![\sigma A_2]\!]_v$

Now, we can conclude as follows:

1. Using a) and d), we get $(f + f_r + c_{primapp}) \leqslant \sigma t + \sigma t' + c_{primapp}$
2. By downward closure (Lemma 33) on d) using

$$m - (f + f_r + c_{primapp}) \leqslant m - (f + f_r)$$

we get $(m - (f + f_r + c_{primapp}), v_r) \in (\!|\sigma A_2|\!)_v$.

Case: $\dfrac{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A \mid t \qquad \Delta; \Phi \models A \sqsubseteq A' \qquad \Delta; \Phi \models t \leqslant t'}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A' \mid t'} \sqsubseteq \textbf{exec}$

Assume that $\models \sigma\Phi$ and $(m, \gamma) \in \mathcal{G}[\![\sigma\Omega]\!]$.
TS: $(m, \gamma e) \in [\![\sigma A']\!]^{\sigma t'}_{\varepsilon}$.
Following the definition of $[\![\cdot]\!]^{\cdot}_{\varepsilon}$, assume that

a) $\gamma e \Downarrow^f v$

b) $f < m$.

By IH 2 on the first premise, we get $(m, \gamma e) \in [\![\sigma A]\!]^{\sigma t'}_{\varepsilon}$.
Unrolling its definition with a) and b), we get

c) $f \leqslant \sigma t$

d) $(m - f, v) \in [\![\sigma A]\!]_v$

We can conclude this subcase

1. By Assumption (25) on the third premise, we get $\sigma t' \leqslant \sigma t'$.
   By c) we know $f \leqslant \sigma t$, therefore we get $f \leqslant \sigma t'$

2. By lemma 39 on the second premise using c), we get $(m - f, v) \in [\![\sigma A']\!]_v$

□

Proof of Statement (3). Remember the statement (3) of Theorem 46:
Assume that $\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CIP}} e : \tau \mid t$ and $\sigma \in \mathcal{D}[\![\Delta]\!]$ and $\models \sigma\Phi$ and $(m, \gamma) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$, then $(m, \gamma e) \in [\![|\sigma\tau|]\!]^{\infty}_{\varepsilon}$.

Case: $\dfrac{\Gamma(x) = \tau}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CIP}} x : \tau \mid 0}$ **cp-var**

Assume that $\models \sigma\Phi$ and $(m, \gamma) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$.
TS: $(m, \gamma(x)) \in [\![|\sigma\tau|]\!]^{0,\infty}_{\varepsilon}$.
By lemma 31, STS: $(m, \gamma(x)) \in [\![|\sigma\tau|]\!]_v$.
By $(m, \gamma) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$ and $\Gamma(x) = \tau$, we can conclude that $(m, \gamma(x)) \in [\![|\sigma\tau|]\!]_v$.

Case: $\dfrac{\Delta; \Phi \vdash \tau_1 \xrightarrow{\mathbb{CIP}(t)} \tau_2 \ \mathsf{wf} \qquad \Delta; \Phi; x : \tau_1, f : \tau_1 \xrightarrow{\mathbb{CIP}(t)} \tau_2, \Gamma \vdash_{\mathbb{CIP}} e : \tau_2 \mid t}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CIP}} \textbf{fix}\ f(x).e : \tau_1 \xrightarrow{\mathbb{CIP}(t)} \tau_2 \mid 0}$ **cp-fix**

Assume that $\models \sigma\Phi$ and $(m, \gamma) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$.
TS: $(m, \text{fix } f(x).\gamma e) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]^{\infty}_{\varepsilon}$.
By lemma 31, STS: $(m, \text{fix } f(x).\gamma e) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_v$.
We prove the more general statement

$$\forall m' \leqslant m.\ (m', \text{fix } f(x).\gamma e) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_v$$

by subinduction on m'.
There are two cases:

subcase 1: $m' = 0$

Since there is no non-negative j such that $j < 0$, the goal is vacuously true.

subcase 2: $m' = m'' + 1 \leqslant m$

By sub-IH

$$(m'', \text{fix } f(x).\gamma e) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_v \tag{1}$$

STS: $(m'' + 1, \text{fix } f(x).\gamma e) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_v$.
Pick $j < m'' + 1$ and assume that $(j, v) \in [\![|\sigma\tau_1|]\!]_v$.
STS: $(j, \gamma e[v/x, (\text{fix } f(x).\gamma e)/f]) \in [\![|\sigma\tau_2|]\!]^{\infty}_{\varepsilon}$.
This follows by IH 3 on the premise instantiated with

- $(j, \gamma[x \mapsto v, f \mapsto (\text{fix } f(x).\gamma e)]) \in \mathcal{G}[\![x : |\sigma\tau_1|, f : |\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|, |\sigma\Gamma|]\!]$ which holds because
  - $(j, \gamma) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$ using downward closure (Lemma 33) on $(m, \gamma) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$ using $j < m'' + 1 \leqslant m$.
  - $(j, v) \in [\![|\sigma\tau_1|]\!]_v$, from the assumption above
  - $(j, \text{fix } f(x).\gamma e) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_v$, obtained by downward closure (Lemma 33) on (1) using $j \leqslant m''$

Case:
$$\frac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_1 : \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2 \mid t_1 \qquad \Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_2 : \tau_1 \mid t_2}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_1\ e_2 : \tau_2 \mid t_1 + t_2 + t}\ \textbf{cp-app}$$

Assume that $\models \sigma\Phi$ and $(m, \gamma) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$.
TS: $(m, \gamma e_1\ \gamma e_2) \in [\![|\sigma\tau_2|]\!]^{\infty}_{\varepsilon}$.
Following the definition of $[\![\cdot]\!]^{:}_{\varepsilon}$, assume that

$$\frac{\gamma e_1 \Downarrow^{f_1} T_1\ (\star) \qquad \gamma e_2 \Downarrow^{f_2} T_2\ (\diamond) \qquad \text{fix } f(x).e = V(T_1) \qquad v_2 = V(T_2) \qquad e[v_2/x, (\text{fix } f(x).e)/f] \Downarrow^{f_r} T_r\ (\dagger) \qquad v_r = V(T_r)}{\gamma e_1\ \gamma e_2 \Downarrow^{f_1+f_2+f_r+c_{app}} \langle v_r, \text{app}(T_1, T_2, T_r)\rangle}\ \textbf{ev-app}$$

and

$f_1 + f_2 + f_r + c_{app} < m$.
By IH 3 on the first premise, we get $(m, \gamma e_1) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]^{\infty}_{\varepsilon}$.
Unrolling its definition with $(\star)$ and $f_1 < m$, we get

a) $f_1 \leqslant \infty$
b) $(m - f_1, \text{fix } f(x).e) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_v$

By IH 3 on the second premise, we get $(m, \gamma e_2) \in [\![|\sigma\tau_1|]\!]^{\infty}_{\varepsilon}$.
Unrolling its definition with $(\diamond)$ and $f_2 < m$, we get

c) $f_2 \leqslant \infty$
d) $(m - f_2, v_2) \in [\![|\sigma\tau_1|]\!]_v$

By downward closure (Lemma 33) on d) using $m - f_1 - f_2 - c_{app} \leqslant m - f_2$, we get

$$(m - (f_1 + f_2 + c_{app}), v_2) \in [\![|\sigma\tau_1|]\!]_v \tag{1}$$

Next, we unroll b) with (1) and $m - (f_1 + f_2 + c_{app}) < m - f_1$ to obtain

$$(m - (f_1 + f_2 + c_{app}), e[v_2/x, (\text{fix } f(x).e)]) \in [\![|\sigma\tau_2|]\!]^{\infty}_{\varepsilon} \tag{2}$$

By unrolling (2)'s definition using $(\dagger)$ and $f_r < m - (f_1 + f_2 + c_{app})$, we get

e) $f_r \leqslant \infty$
f) $(m - (f_1 + f_2 + f_r + c_{app}), v_r) \in [\![|\sigma\tau_2|]\!]_v$

Now, we can conclude as follows:

1. We can trivially show $(f_1 + f_2 + f_r + c_{app}) \leqslant \infty$
2. By f)

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_1 : \tau \mid t_1 \qquad \Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_2 : \mathbf{list}[n]^{\alpha} \tau \mid t_2}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \mathbf{cons}(e_1, e_2) : \mathbf{list}[n+1]^{\alpha+1} \tau \mid t_1 + t_2}$ **cp-cons1**

Assume that $\models \sigma\Phi$ and $(m, \gamma) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$.
TS: $(m, \text{cons}(\gamma e_1, \gamma e_2)) \in [\![|\text{list}[\sigma n + 1]^{\sigma\alpha+1} \sigma\tau|]\!]^{0,\infty}_{\varepsilon} \equiv [\![\text{list}[\sigma n + 1] |\sigma\tau|]\!]^{0,\infty}_{\varepsilon}$.
Following the definition of $[\![\cdot]\!]^{\cdot}_{\varepsilon}$, assume that

$$\frac{\gamma e_1 \Downarrow^{f_1} T_1 \quad (\star) \qquad \gamma e_2 \Downarrow^{f_2} T_2 \quad (\diamond) \qquad v_i = V(T_i)}{\text{cons}(\gamma e_1, \gamma e_2) \Downarrow^{f_1+f_2} \langle \text{cons}(v_1, v_2), \text{cons}(T_1, T_2) \rangle} \textbf{ ev-cons}$$

and $f_1 + f_2 < m$.
By IH 3 on the first premise, we get $(m, \gamma e_1) \in [\![|\sigma\tau|]\!]^{\infty}_{\varepsilon}$.
Unrolling its definition with $(\star)$ and $f_1 < m$, we get

a) $f_1 \leqslant \infty$
b) $(m - f_1, v_1) \in [\![|\sigma\tau|]\!]_v$

By IH 3 on the second premise, we get $(m, \gamma e_2) \in [\![|\mathsf{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|]\!]^{\infty}_{\varepsilon}$.
Unrolling its definition with $(\diamond)$ and $f_2 < m$, we get

c) $f_2 \leqslant \infty$
d) $(m - f_2, v_2) \in [\![\mathsf{list}[\sigma n]\,|\sigma\tau|]\!]_v$

Now, we can conclude as follows:

1. We can trivially show that $(f_1 + f_2) \leqslant \infty$
2. By downward closure (Lemma 33) on b) and d), we get $(m - (f_1 + f_2), v_1) \in [\![|\sigma\tau|]\!]_v$ and $(m - (f_1 + f_2), v_2) \in [\![\mathsf{list}[\sigma n]\,|\sigma\tau|]\!]_v$, when combined, gives us
$(m - (f_1 + f_2), \mathsf{cons}(v_1, v_2)) \in [\![\mathsf{list}[\sigma n + 1]\,|\sigma\tau|]\!]_v \equiv [\![|\mathsf{list}[\sigma n + 1]^{\sigma\alpha+1}\ \sigma\tau|]\!]_v$

Case: $$\frac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_1 : \Box\,\tau \mid t_1 \qquad \Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_2 : \mathbf{list}[n]^{\alpha}\,\tau \mid t_2}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \mathbf{cons}(e_1, e_2) : \mathbf{list}[n+1]^{\alpha}\,\tau \mid t_1 + t_2}\ \textbf{cp-cons2}$$
Assume that $\models \sigma\Phi$ and $(m, \gamma) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$..
TS: $(m, \mathsf{cons}(\gamma e_1, \gamma e_2)) \in [\![|\mathsf{list}[\sigma n + 1]^{\sigma\alpha}\ \sigma\tau|]\!]^{0,\infty}_{\varepsilon} \equiv [\![\mathsf{list}[\sigma n + 1]\,|\sigma\tau|]\!]^{0,\infty}_{\varepsilon}$.
Following the definition of $[\![\cdot]\!]^{\cdot}_{\varepsilon}\cdot$, assume that

$$\frac{\gamma e_1 \Downarrow^{f_1} T_1 \quad (\star) \qquad \gamma e_2 \Downarrow^{f_2} T_2 \quad (\diamond) \qquad v_i = \mathsf{V}(T_i)}{\mathsf{cons}(\gamma e_1, \gamma e_2) \Downarrow^{f_1+f_2} \langle \mathsf{cons}(v_1, v_2), \mathsf{cons}(T_1, T_2)\rangle}\ \textbf{ev-cons}$$

and $f_1 + f_2 < m$.
By IH 3 on the first premise, we get $(m, \gamma e_1) \in [\![|\Box\,\sigma\tau|]\!]^{\infty}_{\varepsilon}$. Unrolling its definition with $(\star)$ and $f_1 < m$, we get

a) $f_1 \leqslant \infty$
b) $(m - f_1, v_1) \in [\![|\Box\,\sigma\tau|]\!]_v \equiv [\![|\sigma\tau|]\!]_v$

By IH 3 on the second premise, we get $(m, \gamma e_2) \in [\![|\text{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|]\!]^{\infty}_{\varepsilon}$.
Unrolling its definition with ($\diamond$) and $f_2 < m$, we get

c) $f_2 \leqslant \infty$
d) $(m - f_2, v_2) \in [\![\text{list}[\sigma n]\ |\sigma\tau|]\!]_v$

Now, we can conclude as follows:

1. We can trivially show that $(f_1 + f_2) \leqslant \infty$
2. By downward closure (Lemma 33) on b) and d), we get $(m - (f_1 + f_2), v_1) \in [\![|\sigma\tau|]\!]_v$ and $(m - (f_1 + f_2), v_2) \in [\![\text{list}[\sigma n]\ |\sigma\tau|]\!]_v$, when combined, gives us $(m - (f_1 + f_2), \text{cons}(v_1, v_2)) \in [\![\text{list}[\sigma n + 1]\ |\sigma\tau|]\!]_v \equiv [\![|\text{list}[\sigma n + 1]^{\sigma\alpha}\ \sigma\tau|]\!]_v$

Case: $\dfrac{\Delta; \Phi; |\Gamma| \vdash_{\mathbb{FS}} e : A \mid t}{\Delta; \Phi; \Gamma \vdash_{\mathbb{CP}} e : U\ A \mid t}$ **cp-switch**
Assume that $\models \sigma\Phi$ and $(m, \gamma) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$.
TS: $(m, \gamma e_1) \in [\![|U\ \sigma A|]\!]^{0,\infty}_{\varepsilon} \equiv [\![\sigma A]\!]^{0,\infty}_{\varepsilon}$.
Assume that

a) $\gamma e_1 \Downarrow^{f_r} v_r$
b) $f_r < m$.

By IH 2 on the first premise, we get $(m, \gamma e_1) \in [\![\sigma A]\!]^{\sigma t}_{\varepsilon}$
By unrolling its definition with a) and b), we get

c) $f_r \leqslant \sigma t$
d) $(m - f_r, v_r) \in [\![\sigma A]\!]_v$

We can conclude as follows

1. Trivially, $f_r \leqslant \infty$
2. By d)

□

Proof of Statement (4). Remember the statement (4) of Theorem 46:
Assume that $\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A \mid t$ and $\sigma \in \mathcal{D}[\![\Delta]\!]$ and $\models \sigma\Phi$ and $(m, \delta) \in$

$\mathcal{G}(\!|\mathcal{U}\,\sigma\Omega|\!)$, then $(m, \delta^{\ulcorner} e^{\urcorner}) \in (\!|\mathcal{U}\,\sigma A|\!)^{\sigma t}_{\varepsilon}$.

Case: $\dfrac{\Omega(x) = A}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} x : A \mid 0}$ **fs-var**

Assume that $\models \sigma\Phi$ and $(m, \delta) \in \mathcal{G}(\!|\mathcal{U}\,\sigma\Omega|\!)$.

TS: $(m, \delta(x)) \in (\!|\mathcal{U}\,\sigma A|\!)^{0}_{\varepsilon}$.

By lemma 31, STS: $(m, \delta(x)) \in (\!|\mathcal{U}\,\sigma A|\!)_{v}$.

This follows by $\Omega(x) = A$ and $(m, \delta) \in \mathcal{G}(\!|\mathcal{U}\,\sigma\Omega|\!)$.

Case: $\dfrac{\Delta; \Phi \vdash^{A} A_1 \xrightarrow{\mathbb{FS}(t)} A_2\ \mathsf{wf} \qquad \Delta; \Phi; x : A_1, f : A_1 \xrightarrow{\mathbb{FS}(t)} A_2, \Omega \vdash_{\mathbb{FS}} e : A_2 \mid t}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} \textbf{fix}\ f(x).e : A_1 \xrightarrow{\mathbb{FS}(t)} A_2 \mid 0}$ **fs-fix**

Assume that $\models \sigma\Phi$ and $(m, \delta) \in \mathcal{G}(\!|\mathcal{U}\,\sigma\Omega|\!)$.

TS: $(m, \text{fix } f(x).\delta^{\ulcorner} e^{\urcorner}) \in (\!|\mathcal{U}\,(\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2)|\!)^{0}_{\varepsilon}$.

By lemma 31, STS: $(m, \text{fix } f(x).\delta^{\ulcorner} e^{\urcorner}) \in (\!|\mathcal{U}\,(\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2)|\!)_{v}$.

By definition of $(\!|\mathcal{U}\,\cdot|\!)_{v}$, since fix $f(x).\delta^{\ulcorner} e^{\urcorner} \neq \mathsf{new}(\cdot,\cdot)$,

STS: $(m, \text{fix } f(x).\delta^{\ulcorner} e^{\urcorner}) \in \mathcal{C}\,\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2\,\mathcal{D}_{v}$.

Let $F = \text{fix } f(x).\delta^{\ulcorner} e^{\urcorner}$.

We prove the more general statement

$$\forall m' \leqslant m.\ (m', F) \in \mathcal{C}\,\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2\,\mathcal{D}_{v}$$

by subinduction on m'.

There are three cases:

- STS: $\forall j.(j, L(F)) \in [\![\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2]\!]_{v} \wedge (j, R(F)) \in [\![\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2]\!]_{v}$.

  Pick j.

  We show the left projection only, the right one is similar.

– STS 1: $(j, L(F)) \in [\![\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2]\!]_v$
We prove the more general statement

$$\forall m' \leqslant j.\ (m', L(F)) \in [\![\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2]\!]_v$$

by subinduction on m'.
There are two cases:

* $m' = 0$
Since there is no non-negative j such that $j < 0$, the goal is vacuously true.

* $m' = m'' + 1 \leqslant j$
By sub-IH

$$(m'', \text{fix } f(x).L(\delta^{\ulcorner} e^{\urcorner})) \in [\![\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2]\!]_v \tag{1}$$

STS: $(m'' + 1, \text{fix } f(x).L(\delta^{\ulcorner} e^{\urcorner})) \in [\![\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2]\!]_v$.
Pick $j'' < m'' + 1$ and assume that $(j'', v) \in [\![\sigma A_1]\!]_v$.
STS: $(j'', L(\delta^{\ulcorner} e^{\urcorner})[v/x, L(F)/f]) \in [\![\sigma A_2]\!]^{\sigma t}_{\varepsilon}$.
This follows by IH 2 on the premise instantiated with $(j'', \delta[x \mapsto v, f \mapsto L(F)]) \in \mathcal{G}[\![x : \sigma A_1, f : \sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2, \sigma\Omega]\!]$ which holds because

· $(j'', L(\delta)) \in \mathcal{G}[\![\sigma\Omega]\!]$ using lemma 32 on $(m, \delta) \in \mathcal{G}(\!|\mathcal{U}\ \sigma\Omega|\!)$

· $(j'', v) \in [\![\sigma A_1]\!]_v$, from the assumption above

· $(j'', \text{fix } f(x).L(\delta^{\ulcorner} e^{\urcorner})) \in [\![\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2]\!]_v$, obtained by downward closure (Lemma 33) on (1) using $j'' \leqslant m''$

• $m' = 0$
Since there is no non-negative j such that $j < 0$, the goal is vacuously true.

- $m' = m'' + 1 \leqslant m$
  By sub-IH

$$(m'', \text{fix } f(x).\delta^{\ulcorner} e^{\urcorner}) \in \mathcal{C}\ \sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2\ \mathcal{D}_v \subseteq (\!|\mathsf{U}\ (\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2)|\!)_v \tag{2}$$

  STS: $(m'' + 1, \text{fix } f(x).\delta^{\ulcorner} e^{\urcorner}) \in \mathcal{C}\ \sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2\ \mathcal{D}_v$.
  Pick $j'' < m'' + 1$ and assume that $(j'', w) \in (\!|\mathsf{U}\ \sigma A_1|\!)_v$.
  STS: $(j'', \delta^{\ulcorner} e^{\urcorner}[w/x, F/f]) \in (\!|\mathsf{U}\ \sigma A_2|\!)^{\sigma t}_{\varepsilon}$.
  This follows by IH 4 on the second premise instantiated with $(j'', \delta[x \mapsto w, f \mapsto F]) \in \mathcal{G}(\!|x : \mathsf{U}\ \sigma A_1, f : \mathsf{U}\ (\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2), \mathsf{U}\ \sigma\Omega|\!)$ which holds because

  - $(j'', \delta) \in \mathcal{G}(\!|\mathsf{U}\ \sigma\Omega|\!)$ by downward closure (Lemma 33) on $(m, \delta) \in \mathcal{G}(\!|\mathsf{U}\ \sigma\Omega|\!)$ using $j'' \leqslant m$.
  - $(j'', w) \in (\!|\mathsf{U}\ \sigma A_1|\!)_v$, from the assumption above
  - $(j'', \text{fix } f(x).\delta^{\ulcorner} e^{\urcorner}) \in (\!|\mathsf{U}\ (\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2)|\!)_v$, obtained by downward closure (Lemma 33) on (2) using $j'' \leqslant m''$

This completes the proof of this case.

Case: $\dfrac{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e_1 : A_1 \xrightarrow{\mathbb{FS}(t)} A_2 \mid t_1 \qquad \Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e_2 : A_1 \mid t_2}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e_1\ e_2 : A_2 \mid t_1 + t_2 + t + c_{app}}$ **fs-app**

Assume that $\models \sigma\Phi$ and $(m, \delta) \in \mathcal{G}(\!|\mathsf{U}\ \sigma\Omega|\!)$.
TS: $(m, \delta^{\ulcorner} e_1\ e_2^{\urcorner}) \in (\!|\mathsf{U}\ \sigma A_2|\!)^{\sigma t_1 + \sigma t_2 + \sigma t + c_{app}}_{\varepsilon}$.
Following the definition of $(\!|\cdot|\!)^{\cdot}_{\varepsilon}$, assume that

$$\dfrac{\begin{array}{c} L(\delta^{\ulcorner} e_1^{\urcorner}) \Downarrow^{f_1} T_1 \quad (\star) \\ L(\delta^{\ulcorner} e_2^{\urcorner}) \Downarrow^{f_2} T_2 \quad (\diamond) \qquad \text{fix } f(x).e = V(T_1) \qquad v_2 = V(T_2) \\ e[v_2/x, (\text{fix } f(x).e)/f] \Downarrow^{f_r} T_r \quad (\dagger) \qquad v_r = V(T_r) \end{array}}{L(\delta^{\ulcorner} e_1^{\urcorner})\ L(\delta^{\ulcorner} e_2^{\urcorner}) \Downarrow^{f_1 + f_2 + f_r + c_{app}} \langle v_r, \mathsf{app}(T_1, T_2, T_r)\rangle}\ \textbf{ev-app} \text{ and}$$

$$\dfrac{\begin{array}{c} R(\delta^{\ulcorner} e_1^{\urcorner}) \Downarrow^{f'_1} T'_1 \quad (\star\star) \\ R(\delta^{\ulcorner} e_2^{\urcorner}) \Downarrow^{f'_2} T'_2 \quad (\diamond\diamond) \qquad \text{fix } f(x).e' = V(T'_1) \qquad v'_2 = V(T'_2) \\ e[v'_2/x, (\text{fix } f(x).e')/f] \Downarrow^{f'_r} T'_r \quad (\dagger\dagger) \qquad v'_r = V(T'_r) \end{array}}{R(\delta^{\ulcorner} e_1^{\urcorner})\ R(\delta^{\ulcorner} e_2^{\urcorner}) \Downarrow^{f'_1 + f'_2 + f'_r + c_{app}} \langle v'_r, \mathsf{app}(T'_1, T'_2, T'_r)\rangle}\ \textbf{ev-app} \text{ and}$$

$(f_1 + f_2 + f_r + c_{app}) < m$.

By IH 4 on the first premise, we get $(m, \delta^{\ulcorner} e_1 {}^{\urcorner}) \in (\!| \mathcal{U}\,(\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2) |\!)_{\varepsilon}^{\sigma t_1}$. Unrolling its definition with $(\star)$, $(\star\star)$ and $f_1 < m$, we get

a) $\langle T_1, \delta^{\ulcorner} e_1 {}^{\urcorner} \rangle \curvearrowright w_1', T_1', c_1'$

b) $\text{fix } f(x).e = L(w_1') \ \wedge\ \text{fix } f(x).e' = R(w_1')$

c) $c_1' \leqslant \sigma t_1$

d) $(m - f_1, w_1') \in (\!| \mathcal{U}\,(\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2) |\!)_v$

By IH 4 on the second premise, we get $(m, \delta^{\ulcorner} e_2 {}^{\urcorner}) \in (\!| \mathcal{U}\,\sigma A_1 |\!)_{\varepsilon}^{\sigma t_2}$.
Unrolling its definition with $(\diamond)$ and $(\diamond\diamond)$ and $f_2 < m$, we get

e) $\langle T_2, \delta^{\ulcorner} e_2 {}^{\urcorner} \rangle \curvearrowright w_2', T_2', c_2'$

f) $v_2 = L(w_2') \ \wedge\ v_2' = R(w_2')$

g) $c_2' \leqslant \sigma t_2$

h) $(m - f_2, w_2') \in (\!| \mathcal{U}\,\sigma A_1 |\!)_v$

There are two cases for d)

subcase 1: $w_1' = \mathsf{new}(\text{fix } f(x).e, \text{fix } f(x).e')$

By d), we have $(m - f_1, \mathsf{new}(\text{fix } f(x).e, \text{fix } f(x).e')) \in (\!| \mathcal{U}\,(\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2) |\!)_v$ $(\star)$

Now, we can conclude as follows:

1. Using a), e) and $(\dagger\dagger)$

$$\frac{\begin{array}{c}\langle T_1, R(\delta^{\ulcorner} e_1 {}^{\urcorner}) \rangle \curvearrowright \mathsf{new}(\text{fix } f(x).e, \text{fix } f(x).e'), T_1', c_1' \\ \langle T_2, R(\delta^{\ulcorner} e_2 {}^{\urcorner}) \rangle \curvearrowright w_2', T_2', c_2' \\ e'[R(w_2')/x, (\text{fix } f(x).e')/f] \Downarrow^{f_r'} T_r' \qquad v_r' = V(T_r')\end{array}}{\begin{array}{c}\langle \langle v_r, \mathsf{app}(T_1, T_2, T_r) \rangle, R(\delta^{\ulcorner} e_1 {}^{\urcorner})\ R(\delta^{\ulcorner} e_2 {}^{\urcorner}) \rangle \curvearrowright \\ \mathsf{new}(v_r, v_r'), \langle v_r', \mathsf{app}(T_1', T_2', T_r') \rangle, c_1' + c_2' + f_r' + c_{app}\end{array}}\ \textbf{cp-app-new}$$

2. Trivially, $v_r = L(\mathsf{new}(v_r, v_r')) \ \wedge\ v_r' = R(\mathsf{new}(v_r, v_r'))$

4. TS: $(m - (f_1 + f_2 + f_r + c_{app}), \mathsf{new}(v_r, v_r')) \in (\!| \mathcal{U}\,\sigma A_2 |\!)_v$
STS: $\forall j.(j, v_r) \in [\![\sigma A_2]\!]_v \ \wedge\ (j, v_r') \in [\![\sigma A_2]\!]_v$

Pick j.
TS1: $(j, v_r) \in [\![\sigma A_2]\!]_v$
By unrolling the definition of $(\star)$, we have

$$\forall j.(j, \text{fix } f(x).e) \in [\![\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2]\!]_v \wedge (j, \text{fix } f(x).e') \in [\![\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2]\!]_v \tag{1}$$

By lemma 32 on h), we get

$$\forall j.(j, L(w_2')) \in [\![\sigma A_1]\!]_v \wedge (j, R(w_2')) \in [\![\sigma A_2]\!]_v \tag{2}$$

Next, we instantiate eq. (2) with $j + f_r + 2$ and get

$$(j + f_r + 2, \text{fix } f(x).e) \in [\![\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2]\!]_v \tag{3}$$

Then, we instantiate eq. (2) with $j + f_r + 1$ and get

$$(j + f_r + 1, L(w_2')) \in [\![\sigma A_1]\!]_v \tag{4}$$

Unrolling the definition of eq. (3) using eq. (4) and $j + f_r + 1 < j + f_r + 2$, we get

$$(j + f_r + 1, e[L(w_2')/x, (\text{fix } f(x).e)/f]) \in [\![\sigma A_2]\!]_\varepsilon^{\sigma t} \tag{5}$$

Unrolling the definition of eq. (5) with $(\dagger)$ and $f_r < j + f_r + 1$, we get

i) $f_r \leqslant \sigma t$
j) $(j + 1, v_r) \in [\![\sigma A_2]\!]_v$

Then, we obtain $(j, v_r) \in [\![\sigma A_2]\!]_v$ by downward closure (Lemma 33) on j) using $j \leqslant j + 1$.
This concludes TS1.
Next, we follow similar steps for the right projection as fol-

lows:

We next instantiate eq. (2) with $j + f'_r + 2$ and get

$$(j + f'_r + 2, \text{fix } f(x).e') \in [\![\sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2]\!]_v \tag{6}$$

Then, we instantiate eq. (2) with $j + f'_r + 1$ and get

$$(j + f'_r + 1, R(w'_2)) \in [\![\sigma A_1]\!]_v \tag{7}$$

Unrolling the definition of eq. (6) using eq. (7) and $j + f'_r + 1 < j + f'_r + 2$, we get

$$(j + f'_r + 1, e[R(w'_2)/x, (\text{fix } f(x).e')/f]) \in [\![\sigma A_2]\!]^{\sigma t}_{\varepsilon} \tag{8}$$

Unrolling the definition of eq. (8) with (††) and $f'_r < j + f'_r + 1$, we get

k) $f'_r \leqslant \sigma t$

l) $(j + 1, v'_r) \in [\![\sigma A_2]\!]_v$

Then, we obtain $(j, v'_r) \in [\![\sigma A_2]\!]_v$ by downward closure (Lemma 33) on l) using $j \leqslant j + 1$.

This concludes TS2.

3. Using c), g) and k), we get $(c'_1 + c'_2 + f'_r + c_{app}) \leqslant \sigma t_1 + \sigma t_2 + \sigma t + c_{app}$

subcase 2: $w'_1 = \texttt{fix } f(x).\text{æ}$

Then, by unrolling the definition of d), we have

$$(m - f_1, \texttt{fix } f(x).\text{æ}) \in \mathbb{C}\ \sigma A_1 \xrightarrow{\mathbb{FS}(\sigma t)} \sigma A_2\ \mathbb{D}_v \tag{9}$$

Next, we apply downward-closure (lemma 33) to h) using

$$m - (f_1 + f_2 + c_{app}) \leqslant m - f_2$$

and we get

$$(m - (f_1 + f_2 + c_{app}), w_2') \in (\!|U\ \sigma A_1|\!)_v \tag{10}$$

We unroll eq. (9) using (10) since

$m - (f_1 + f_2 + c_{app}) < m - f_1$ Note that here we have $c_{app} \geqslant 1$

and get

$$(m - (f_1 + f_2 + c_{app}), ee[w_2'/x, \text{fix } f(x).ee/f]) \in (\!|U\ \sigma A_2|\!)_\varepsilon^{\sigma t} \tag{11}$$

Next, we unroll (11) using (†), (††) and $f_r < m - (f_1 + f_2 + c_{app})$ to obtain

i) $\langle T_r, ee[w_2'/x, \text{fix } f(x).ee/f]\rangle \curvearrowright w_r', T_r', c_r'$
j) $v_r = L(w_r') \ \wedge\ v_r' = R(w_r')$
k) $c_r' \leqslant \sigma t$
l) $(m - (f_1 + f_2 + f_r + c_{app}), w_r') \in (\!|U\ \sigma A_2|\!)_v$

Now, we can conclude as follows:

1. Using a), e) and i)
$$\frac{\begin{array}{c}\langle T_1, R(\delta^\ulcorner e_1{}^\urcorner)\rangle \curvearrowright \texttt{fix } f(x).ee, T_1', c_1' \\ \langle T_2, R(\delta^\ulcorner e_2{}^\urcorner)\rangle \curvearrowright w_2', T_2', c_2' \\ \langle T_r, ee[w_2'/x, (\texttt{fix } f(x).ee)/f]\rangle \curvearrowright w_r', T_r', c_r' \qquad v_r' = V(T_r')\end{array}}{\begin{array}{c}\langle\langle \_, \texttt{app}(T_1, T_2, T_r)\rangle, R(\delta^\ulcorner e_1{}^\urcorner)\ R(\delta^\ulcorner e_2{}^\urcorner)\rangle \curvearrowright \\ w_r', \langle v_r', \texttt{app}(T_1', T_2', T_r')\rangle, c_1' + c_2' + c_r'\end{array}}\ \textbf{cp-app}$$
2. By j)
3. Using c), g) and k), we get
$(c_1' + c_2' + c_r') \leqslant \sigma t_1 + \sigma t_2 + \sigma t \leqslant \sigma t_1 + \sigma t_2 + \sigma t + c_{app}$
4. By l)

Case:
$$\frac{\Delta;\Phi_a;\Omega \vdash_{\mathbb{FS}} e : A_1 + A_2 \mid t \qquad \Delta;\Phi;x : A_1,\Omega \vdash_{\mathbb{FS}} e_1 : A \mid t' \qquad \Delta;\Phi;y : A_2,\Omega \vdash_{\mathbb{FS}} e_2 : A \mid t'}{\Delta;\Phi_a;\Omega \vdash_{\mathbb{FS}} \textbf{case}\ (e, x.e_1, y.e_2) : A \mid t + t' + c_{case}}\ \textbf{fs-case}$$

Assume that $\models \sigma\Phi$ and $(m,\delta) \in \mathcal{G}(\!|\mathcal{U}\,\sigma\Omega|\!)$.

TS: $(m,\ \text{case}\ (\delta\ulcorner e\urcorner, \delta\ulcorner e_1\urcorner, \delta\ulcorner e_2\urcorner)) \in (\!|\mathcal{U}\,\sigma A|\!)_\varepsilon^{\sigma t+\sigma t'+c_{case}}$.

Following the definition of $(\!|\cdot|\!)_\varepsilon^{\cdot}$, assume that

$L(\ \text{case}\ (\delta\ulcorner e\urcorner, \delta\ulcorner e_1\urcorner, \delta\ulcorner e_2\urcorner)) \Downarrow^F v_r$ and

$R(\ \text{case}\ (\delta\ulcorner e\urcorner, \delta\ulcorner e_1\urcorner, \delta\ulcorner e_2\urcorner)) \Downarrow^{F'} v'_r$ and

$F < m$.

Depending on what $L(\delta\ulcorner e\urcorner)$ and $R(\delta\ulcorner e\urcorner)$ evaluate to, there are four cases:

subcase 1:

$$\frac{L(\delta\ulcorner e\urcorner) \Downarrow^f T\ \ (\star) \qquad \text{inl}\ v = V(T) \qquad L(\delta\ulcorner e_1\urcorner)[v/x] \Downarrow^{f_r} T_r\ \ (\diamond) \qquad v_r = V(T_r)}{\text{case}\ (L(\delta\ulcorner e\urcorner), x.L(\delta\ulcorner e_1\urcorner), y.L(\delta\ulcorner e_2\urcorner)) \Downarrow^{f+f_r+c_{case}} \langle v_r, \text{case}_{\text{inl}}(T, T_r)\rangle}\ \textbf{ev-case-l}$$

and

$$\frac{R(\delta\ulcorner e\urcorner) \Downarrow^{f'} T'\ \ (\star\star) \qquad \text{inl}\ v' = V(T') \qquad R(\delta\ulcorner e_1\urcorner)[v'/x] \Downarrow^{f'_r} T'_r\ \ (\diamond\diamond) \qquad v'_r = V(T'_r)}{\text{case}\ (R(\delta\ulcorner e\urcorner), x.R(\delta\ulcorner e_1\urcorner), y.R(\delta\ulcorner e_2\urcorner)) \Downarrow^{f'+f'_r+c_{case}} \langle v'_r, \text{case}_{\text{inl}}(T', T'_r)\rangle}\ \textbf{ev-case-l}$$

and

$F = f + f_r + c_{case} < m$.

By IH 4 on the first premise, we get $(m, \delta\ulcorner e\urcorner) \in (\!|\mathcal{U}\,\sigma A_1 + \sigma A_2|\!)_\varepsilon^{\sigma t}$. Unrolling its definition with $(\star)$, $(\star\star)$ and $f < m$, we get

a) $\langle T, \delta\ulcorner e\urcorner\rangle \curvearrowright w', T', c'$

b) $\text{inl}\ v = L(w') \ \wedge\ \text{inl}\ v' = R(w')$

c) $c' \leqslant \sigma t$

d) $(m - f, w') \in (\!|\mathcal{U}\,\sigma A_1 + \sigma A_2|\!)_v$

There are two cases for d):

subsubcase 1: $w' = \mathsf{new}(\mathsf{inl}\ v, \mathsf{inl}\ v')$

By d), we have $(m - f, \mathsf{new}(\mathsf{inl}\ v, \mathsf{inl}\ v')) \in (\!|U\ (\sigma A_1 + \sigma A_2)|\!)_v$.
By unrolling its definition, we have

$$\forall j.(j, \mathsf{inl}\ v) \in [\![\sigma A_1 + \sigma A_2]\!]_v \wedge (j, \mathsf{inl}\ v') \in [\![\sigma A_1 + \sigma A_2]\!]_v \quad (1)$$

Then, by using eq. (1), we can show that

$$\forall j.(j, v) \in [\![A_1]\!]_v \wedge (j, v') \in [\![A_1]\!]_v \quad (2)$$

We conclude as follows:

1. Using a) and ($\diamond\diamond$)
$$\frac{\langle T, \delta\ulcorner e\urcorner\rangle \curvearrowright \mathsf{new}(\_, \mathsf{inl}\ v'), T', c' \qquad R(\delta\ulcorner e_1\urcorner)[v'/x] \Downarrow^{f'_r} T'_r \qquad v'_r = V(T'_r)}{\langle\langle v_r, \mathsf{case}_{\mathsf{inl}}(T, T_r)\rangle, \mathsf{case}(\delta\ulcorner e\urcorner, x.\delta\ulcorner e_1\urcorner, y, \delta\ulcorner e_2\urcorner)\rangle \curvearrowright \mathsf{new}(v_r, v'_r), \langle v'_r, \mathsf{case}_{\mathsf{inl}}(T', T'_r)\rangle, c' + f'_r + c_{case}}\ \textbf{cp-case-inl}_l$$
2. Trivially, $v_r = L(\mathsf{new}(v_r, v'_r)) \wedge v'_r = R(\mathsf{new}(v_r, v'_r))$
4. TS: $(m - (f + f_r + c_{case}), \mathsf{new}(v_r, v'_r)) \in (\!|U\ \sigma A'_2|\!)_v$
STS: $\forall j.(j, v_r) \in [\![\sigma A']\!]_v \wedge (j, v'_r) \in [\![\sigma A']\!]_v$.
Pick j.
TS1: $(j, v_r) \in [\![\sigma A']\!]_v$
By instantiating the definition of eq. (2) with $j + f_r + 1$, we have

$$(j + f_r + 1, v) \in [\![A_1]\!]_v \quad (3)$$

By IH 2 on the second premise using $(j + f_r + 1, L(\delta)[x \mapsto v]) \in \mathcal{G}[\![\Omega, x : \sigma A_1]\!]$ obtained by

- $(j + f_r + 1, L(\delta)) \in \mathcal{G}[\![\sigma\Omega]\!]$ by (lemma 32) on $(m, \delta) \in \mathcal{G}(\!|U\ \sigma\Omega|\!)$
- $(j + f_r + 1, v) \in [\![\sigma A_1]\!]_v$ by eq. (3)

we get $(j + f_r + 1, L(\delta)L(\ulcorner e_1 \urcorner)[\nu/x]) \in [\![\sigma A']\!]^{\sigma t'}_{\varepsilon}$.
Unrolling its definition with $(\diamond)$ and $f_r < j + f_r + 1$, we get

e) $f_r \leqslant \sigma t'$

f) $(j + 1, \nu_r) \in [\![\sigma A']\!]_\nu$

Then, we obtain $(j, \nu_r) \in [\![\sigma A']\!]_\nu$ by downward closure (Lemma 33) on j) using $j \leqslant j + 1$.
Next, we follow similar steps for the right projection as follows:
TS2: $(j, \nu'_r) \in [\![\sigma A']\!]_\nu$
By instantiating the definition of eq. (2) with $j + f'_r + 1$, we have

$$(j + f'_r + 1, \nu') \in [\![A_1]\!]_\nu \tag{4}$$

By IH 2 on the **second premise** using $(j + f'_r + 1, R(\delta)[x \mapsto \nu']) \in \mathcal{G}[\![\Omega, x : \sigma A_1]\!]$ obtained by

- $(j + f'_r + 1, R(\delta)) \in \mathcal{G}[\![\sigma\Omega]\!]$ by (lemma 32) on $(m, \delta) \in \mathcal{G}(\!|\mathcal{U}\,\sigma\Omega|\!)$
- $(j + f'_r + 1, \nu') \in [\![\sigma A_1]\!]_\nu$ by eq. (4)

we get $(j + f'_r + 1, R(\delta)R(\ulcorner e_1 \urcorner)[\nu'/x]) \in [\![\sigma A']\!]^{\sigma t'}_{\varepsilon}$.
Unrolling its definition with $(\diamond\diamond)$ and $f'_r < j + f'_r + 1$, we get

g) $f'_r \leqslant \sigma t'$

h) $(j + 1, \nu'_r) \in [\![\sigma A']\!]_\nu$

Then, we obtain $(j, \nu'_r) \in [\![\sigma A']\!]_\nu$ by downward closure (Lemma 33) on j) using $j \leqslant j + 1$.

3. By using c) and g), we get $c' + f'_r + c_{case} \leqslant \sigma t + \sigma t' + c_{case}$

subsubcase 2: $w' = \texttt{inl}\ w$

By d), we have $(m - f, \texttt{inl}\ w) \in (\!|U\,(\sigma A_1 + \sigma A_2)|\!)_v$.
By unrolling its definition, we have

$$(m - f, w) \in (\!|U\,\sigma A_1|\!)_v \tag{5}$$

By IH 4 on the **second premise** using $(m - f, \delta[x \mapsto w]) \in \mathcal{G}(\!|U\,\sigma\Omega, x : U\,\sigma A_1|\!)$ obtained by

- $(m - f, \delta) \in \mathcal{G}(\!|U\,\sigma\Omega|\!)$ by downward-closure (lemma 33) on $(m, \delta) \in \mathcal{G}(\!|U\,\sigma\Omega|\!)$ using $m - f \leqslant m$
- $(m - f, w) \in (\!|U\,\sigma A_1|\!)_v$ by eq. (5)

we get $(m - f, \delta\ulcorner e_1 \urcorner[w/x]) \in (\!|U\,\sigma A|\!)_\varepsilon^{\sigma t'}$.
Unrolling its definition with ($\diamond$), ($\diamond\diamond$) and $f_r < m - f$, we get

e) $\langle T_r, \delta\ulcorner e_1 \urcorner[w/x]\rangle \curvearrowright w'_r, T'_r, c'_r$
f) $v_r = L(w'_r) \ \wedge\ v'_r = R(w'_r)$
g) $c'_r \leqslant \sigma t'$
h) $(m - f - f_r, w'_r) \in (\!|U\,\sigma A'|\!)_v$

We conclude with

1. Using a) and e)
$$\frac{\langle T, \delta\ulcorner e\urcorner\rangle \curvearrowright \texttt{inl}\ w, T', c' \qquad \langle T_r, \delta\ulcorner e_1\urcorner[w/x]\rangle \curvearrowright w'_r, T'_r, c'_r \qquad v'_r = V(T'_r)}{\langle\langle \_, \texttt{case}_{\texttt{inl}}(T, T_r)\rangle, \texttt{case}(\delta\ulcorner e\urcorner, x.\delta\ulcorner e_1\urcorner, y, \delta\ulcorner e_2\urcorner)\rangle \curvearrowright w'_r, \langle v'_r, \texttt{case}_{\texttt{inl}}(T', T'_r)\rangle, c' + c'_r}\ \textbf{cp-case-inl}$$
2. Using f)
3. By using c) and g), we get
$c' + c'_r \leqslant \sigma t + \sigma t' \leqslant \sigma t + \sigma t' + c_{case}$
4. By downward closure (Lemma 33) on h) using
$$m - (f + f_r + c_{case}) \leqslant m - f - f_r$$

we get $(m - (f + f_r + c_{case}), w'_r) \in (\!|tchs\sigma A'|\!)_v$.

subcase 2: $$\frac{e \Downarrow^{f} T \qquad \text{inr } v = V(T) \qquad e_2[v/y] \Downarrow^{f_r} T_r \qquad v_r = V(T_r)}{\text{case } (e, x.e_1, y.e_2) \Downarrow^{f+f_r+c_{case}} \langle v_r, \text{case}_{\text{inr}}(T, T_r)\rangle}\ \textbf{ev-case-r}$$

This case is symmetric, hence we skip its proof.

subcase 3:

$$\frac{\begin{array}{c} L(\delta^{\ulcorner} e^{\urcorner}) \Downarrow^{f} T \ \ (\star) \\ \text{inl } v = V(T) \qquad L(\delta^{\ulcorner} e_1{}^{\urcorner})[v/x] \Downarrow^{f_r} T_r \ \ (\diamond) \qquad v_r = V(T_r) \end{array}}{\text{case } (L(\delta^{\ulcorner} e^{\urcorner}), x.L(\delta^{\ulcorner} e_1{}^{\urcorner}), y.L(\delta^{\ulcorner} e_2{}^{\urcorner})) \Downarrow^{f+f_r+c_{case}} \langle v_r, \text{case}_{\text{inl}}(T, T_r)\rangle}\ \textbf{ev-case-l}$$

and

$$\frac{\begin{array}{c} R(\delta^{\ulcorner} e^{\urcorner}) \Downarrow^{f'} T' \ \ (\star\star) \\ \text{inr } v' = V(T') \qquad R(\delta^{\ulcorner} e_2{}^{\urcorner})[v'/y] \Downarrow^{f'_r} T'_r \ \ (\diamond\diamond) \qquad v'_r = V(T'_r) \end{array}}{\text{case } (R(\delta^{\ulcorner} e^{\urcorner}), x.R(\delta^{\ulcorner} e_1{}^{\urcorner}), y.R(\delta^{\ulcorner} e_2{}^{\urcorner})) \Downarrow^{f'+f'_r+c_{case}} \langle v'_r, \text{case}_{\text{inr}}(T', T'_r)\rangle}\ \textbf{ev-case-r}$$

and

$F = f + f_r + c_{case} < m$.

By IH 4 on the first premise, we get $(m, \delta^{\ulcorner} e^{\urcorner}) \in (\!|U\,\sigma A_1 + \sigma A_2|\!)^{\sigma t}_{\varepsilon}$. Unrolling its definition with $(\star)$, $(\star\star)$ and $f < m$, we get

a) $\langle T, \delta^{\ulcorner} e^{\urcorner}\rangle \curvearrowright w', T', c'$

b) $\text{inl } v = L(w') \ \wedge\ \text{inr } v' = R(w')$

c) $c' \leqslant \sigma t$

d) $(m - f, w') \in (\!|U\,\sigma A_1 + \sigma A_2|\!)_v$

There are two cases for d):

subsubcase 1: $w' = \text{new}(\text{inl } v, \text{inr } v')$

By d), we have $(m - f, \text{new}(\text{inl } v, \text{inr } v')) \in (\!|U\,(\sigma A_1 + \sigma A_2)|\!)_v$.

By unrolling its definition, we have

$$\forall j.(j, \text{inl } v) \in [\![\sigma A_1 + \sigma A_2]\!]_v \wedge (j, \text{inr } v') \in [\![\sigma A_1 + \sigma A_2]\!]_v \quad (6)$$

Then, by using eq. (6), we can show that

$$\forall j.(j, v) \in [\![A_1]\!]_v \ \wedge\ (j, v') \in [\![A_2]\!]_v \quad (7)$$

We conclude as follows:

1. Using a) and ($\diamond\diamond$)

$$\frac{\langle T, \delta\ulcorner e\urcorner\rangle \curvearrowright \mathsf{new}(\_, \mathrm{inr}\ v'), T', c' \qquad R(\delta\ulcorner e_2\urcorner)[v'/x] \Downarrow^{f'_r} T'_r \qquad v'_r = V(T'_r)}{\langle\langle v_r, \mathsf{case}_{\mathsf{inl}}(T, T_r)\rangle, \mathsf{case}(\delta\ulcorner e\urcorner, x.\delta\ulcorner e_1\urcorner, y, \delta\ulcorner e_2\urcorner)\rangle \curvearrowright \mathsf{new}(v_r, v'_r), \langle v'_r, \mathsf{case}_{\mathsf{inr}}(T', T'_r)\rangle, c' + f'_r + c_{case}}\ \textbf{cp-case-inl}_r$$

2. Trivially, $v_r = L(\mathsf{new}(v_r, v'_r)) \ \wedge \ v'_r = R(\mathsf{new}(v_r, v'_r))$

4. TS: $(m - (f + f_r + c_{case}), \mathsf{new}(v_r, v'_r)) \in (\!|U\ \sigma A'_2|\!)_v$
   STS: $\forall j.(j, v_r) \in [\![\sigma A']\!]_v \ \wedge \ (j, v'_r) \in [\![\sigma A']\!]_v$.
   Pick j.
   TS1: $(j, v_r) \in [\![\sigma A']\!]_v$
   By instantiating the definition of eq. (7) with $j + f_r + 1$, we have

$$(j + f_r + 1, v) \in [\![A_1]\!]_v \tag{8}$$

   By IH 2 on the **second premise** using $(j + f_r + 1, L(\delta)[x \mapsto v]) \in \mathcal{G}[\![\Omega, x : \sigma A_1]\!]$ obtained by

   - $(j + f_r + 1, L(\delta)) \in \mathcal{G}[\![\sigma\Omega]\!]$ by (lemma 32) on $(m, \delta) \in \mathcal{G}(\!|U\ \sigma\Omega|\!)$
   - $(j + f_r + 1, v) \in [\![\sigma A_1]\!]_v$ by eq. (8)

   we get $(j + f_r + 1, L(\delta)L(\ulcorner e_1\urcorner)[v/x]) \in [\![\sigma A']\!]^{\sigma t'}_{\varepsilon}$.
   Unrolling its definition with ($\diamond$) and $f_r < j + f_r + 1$, we get

   e) $f_r \leqslant \sigma t'$
   f) $(j + 1, v_r) \in [\![\sigma A']\!]_v$

   Then, we obtain $(j, v_r) \in [\![\sigma A']\!]_v$ by downward closure (Lemma 33) on j) using $j \leqslant j + 1$.
   Next, we follow similar steps for the right projection as follows:
   TS2: $(j, v'_r) \in [\![\sigma A']\!]_v$

By instantiating the definition of eq. (7) with $j + f'_r + 1$, we have

$$(j + f'_r + 1, v') \in [\![A_2]\!]_v \tag{9}$$

By IH 2 on the **third premise** using $(j + f'_r + 1, R(\delta)[x \mapsto v']) \in \mathcal{G}[\![\Omega, x : \sigma A_2]\!]$ obtained by

- $(j + f'_r + 1, R(\delta)) \in \mathcal{G}[\![\sigma\Omega]\!]$ by (lemma 32) on $(m, \delta) \in \mathcal{G}(\!|\mathcal{U}\ \sigma\Omega|\!)$
- $(j + f'_r + 1, v') \in [\![\sigma A_2]\!]_v$ by eq. (9)

we get $(j + f'_r + 1, R(\delta)R(\ulcorner e_2 \urcorner)[v'/x]) \in [\![\sigma A']\!]^{\sigma t'}_{\varepsilon}$.
Unrolling its definition with ($\diamond\diamond$) and $f'_r < j + f'_r + 1$, we get

g) $f'_r \leqslant \sigma t'$

h) $(j + 1, v'_r) \in [\![\sigma A']\!]_v$

Then, we obtain $(j, v'_r) \in [\![\sigma A']\!]_v$ by downward closure (Lemma 33) on j) using $j \leqslant j + 1$.

3. By using c) and g), we get $c' + f'_r + c_{case} \leqslant \sigma t + \sigma t' + c_{case}$

subsubcase 2: $w' = \texttt{inl}\ w \vee \texttt{inr}\ w$

This case is impossible due to $\text{inl}\ v = L(w') \wedge \text{inr}\ v' = R(w')$ (obtained in b))

subcase 4:

$$\frac{\text{inr}\ v = V(T) \qquad L(\delta\ulcorner e\urcorner) \Downarrow^{f}\ T\ \ (\star) \qquad L(\delta\ulcorner e_2\urcorner)[v/y] \Downarrow^{f_r}\ T_r\ \ (\diamond) \qquad v_r = V(T_r)}{\text{case}\ (L(\delta\ulcorner e\urcorner), x.L(\delta\ulcorner e_1\urcorner), y.L(\delta\ulcorner e_2\urcorner)) \Downarrow^{f+f_r+c_{case}} \langle v_r, \texttt{case}_{\texttt{inr}}(T, T_r)\rangle}\ \textbf{ev-case-r}$$

and

$$\frac{\text{inl}\ v' = V(T') \qquad R(\delta\ulcorner e\urcorner) \Downarrow^{f'}\ T'\ \ (\star\star) \qquad R(\delta\ulcorner e_1\urcorner)[v'/x] \Downarrow^{f'_r}\ T'_r\ \ (\diamond\diamond) \qquad v'_r = V(T'_r)}{\text{case}\ (R(\delta\ulcorner e\urcorner), x.R(\delta\ulcorner e_1\urcorner), y.R(\delta\ulcorner e_2\urcorner)) \Downarrow^{f'+f'_r+c_{case}} \langle v'_r, \texttt{case}_{\texttt{inl}}(T', T'_r)\rangle}\ \textbf{ev-case-l}$$

and

$F = f + f_r + c_{case} < m$.

This case is symmetric to above case, hence we skip its proof.

Case: $\dfrac{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A \mid t \qquad \Delta; \Phi \models A \sqsubseteq A' \qquad \Delta; \Phi \models t \leqslant t'}{\Delta; \Phi_a; \Omega \vdash_{\mathbb{FS}} e : A' \mid t'} \sqsubseteq \textbf{exec}$

Assume that $\models \sigma\Phi$ and $(m, \delta) \in \mathcal{G}(\!|\mathcal{U}\, \sigma\Omega|\!)$..

TS: $(m, \delta^{\ulcorner} e^{\urcorner}) \in (\!|\mathcal{U}\, \sigma A'|\!)^{\sigma t'}_{\varepsilon}$.

Following the definition of $(\!|\cdot|\!)^{\cdot}_{\varepsilon}$, assume that

a) $L(\delta^{\ulcorner} e^{\urcorner}) \Downarrow^{f} T$

b) $R(\delta^{\ulcorner} e^{\urcorner}) \Downarrow^{f'} T'$

c) $f < m$

By IH 1 on the first premise using we get $(m, \delta^{\ulcorner} e^{\urcorner}) \in (\!|\mathcal{U}\, \sigma A|\!)^{\sigma t}_{\varepsilon}$.
Unrolling its definition with (a-c), we get

d) $\langle T, \delta^{\ulcorner} e^{\urcorner} \rangle \curvearrowright w', T', c'$

e) $v = L(w') \ \wedge \ v' = R(w')$

f) $c' \leqslant \sigma t$

g) $(m - f, w') \in (\!|\mathcal{U}\, \sigma A|\!)_{v}$

We can conclude as follows:

1. By d)
2. By e)
3. By assumption 25 on the third premise, we get $\sigma t \leqslant \sigma t'$. Combining this with f), we get $c' \leqslant \sigma t'$.
4. By lemma 39 (clause 8) on the second premise with g), we get $(m - c, w') \in (\!|\mathcal{U}\, \sigma A'|\!)_{v}$

□

Proof of Statement (5). Remember the statement (5) of Theorem 46:
Assume that $\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CIP}} e : \tau \mid t$ and $\sigma \in \mathcal{D}[\![\Delta]\!]$ and $\models \sigma\Phi$ and $(m, \delta) \in \mathcal{G}(\!|\mathcal{U}\, |\sigma\Gamma|\!)$, then $(m, \delta^{\ulcorner} e^{\urcorner}) \in (\!|\mathcal{U}\, |\sigma\tau|\!)^{\infty}_{\varepsilon}$.

Case: $\dfrac{\Gamma(x) = \tau}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} x : \tau \mid 0}$ **cp-var**

Assume that $\models \sigma\Phi$ and $(m, \delta) \in \mathcal{G}(\!|\mathcal{U}\,|\sigma\Gamma|\!|)$.

TS: $(m, \delta(x)) \in (\!|\mathcal{U}\,|\sigma\tau|\!|)^{\infty}_{\varepsilon}$.

Instead, we first show

$$(m, \delta(x)) \in (\!|\mathcal{U}\,|\sigma\tau|\!|)^{0}_{\varepsilon} \tag{1}$$

By lemma 31, STS: $(m, \delta(x)) \in (\!|\mathcal{U}\,|\sigma\tau|\!|)_{v}$.

By $(m, \delta) \in \mathcal{G}(\!|\mathcal{U}\,|\sigma\Gamma|\!|)$ and $\Gamma(x) = \tau$, we obtain $(m, \delta(x)) \in (\!|\mathcal{U}\,|\sigma\tau|\!|)_{v}$.

We conclude by lemma 39 on eq. (1) using $0 \leqslant \infty$.

Case: $\dfrac{\Delta; \Phi \vdash \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2 \ \mathsf{wf} \qquad \Delta; \Phi; x : \tau_1, f : \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2, \Gamma \vdash_{\mathbb{CP}} e : \tau_2 \mid t}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} \textbf{fix}\ f(x).e : \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2 \mid 0}$ **cp-fix**

Assume that $\models \sigma\Phi$ and $(m, \delta) \in \mathcal{G}(\!|\mathcal{U}\,|\sigma\Gamma|\!|)$.

TS: $(m, \text{fix } f(x).\delta^{\ulcorner} e^{\urcorner}) \in (\!|\mathcal{U}\,|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!|)^{\infty}_{\varepsilon}$.

By lemma 31 and lemma 39 using $0 \leqslant \infty$,

STS: $(m, \text{fix } f(x).\delta^{\ulcorner} e^{\urcorner}) \in (\!|\mathcal{U}\,|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!|)_{v} = (\!|\mathcal{U}\,(|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|)|\!|)_{v}$.

Let $F = \text{fix } f(x).\delta^{\ulcorner} e^{\urcorner}$.

By definition of $(\!|\mathcal{U}\,\cdot|\!|)_{v}$, since fix $f(x).\delta^{\ulcorner} e^{\urcorner} \neq \mathsf{new}(\cdot, \cdot)$,

STS: $(m, \text{fix } f(x).\delta^{\ulcorner} e^{\urcorner}) \in \mathcal{C}\,|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|\,\mathcal{D}_{v}$.

Let $F = \text{fix } f(x).\delta^{\ulcorner} e^{\urcorner}$.

We prove the more general statement

$$\forall m' \leqslant m.\ (m', F) \in \mathcal{C}\,|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|\,\mathcal{D}_{v}$$

by subinduction on m'.

There are three cases:

- STS: $\forall j.(j, L(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_{v} \ \wedge\ (j, R(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_{v}$.

Pick j.

We show the left projection only, the right one is similar.

- STS 1: $(j, L(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$

  We prove the more general statement

  $$\forall m' \leqslant j.\ (m', L(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$$

  by subinduction on m'.

  There are two cases:

  * $m' = 0$

    Since there is no non-negative j such that $j < 0$, the goal is vacuously true.

  * $m' = m'' + 1 \leqslant j$

    By sub-IH

    $$(m'', \text{fix } f(x).L(\delta^\ulcorner e^\urcorner)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu \qquad (1)$$

    STS: $(m'' + 1, \text{fix } f(x).L(\delta^\ulcorner e^\urcorner)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$.

    Pick $j'' < m'' + 1$ and assume that $(j'', \nu) \in [\![|\sigma\tau_1|]\!]_\nu$.

    STS: $(j'', L(\delta^\ulcorner e^\urcorner)[\nu/x, L(F)/f]) \in [\![\sigma A_2]\!]^\infty_\varepsilon$.

    This follows by IH 3 on the second premise instantiated with $(j'', \delta[x \mapsto \nu, f \mapsto L(F)]) \in \mathcal{G}[\![x : \sigma A_1, f : |\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|, |\sigma\Gamma|]\!]$ which holds because

    · $(j'', L(\delta)) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$ using lemma 32 on $(m, \delta) \in \mathcal{G}(\!|U\,|\sigma\Gamma|)\!)$

    · $(j'', \nu) \in [\![|\sigma\tau_1|]\!]_\nu$, from the assumption above

    · $(j'', \text{fix } f(x).L(\delta^\ulcorner e^\urcorner)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$, obtained by downward closure (Lemma 33) on (1) using $j'' \leqslant m''$

- $m' = 0$

  Since there is no non-negative j such that $j < 0$, the goal is vacuously true.

- $m' = m'' + 1 \leqslant m$
  By sub-IH

$$(m'', \text{fix } f(x).\delta^{\ulcorner} e^{\urcorner}) \in \mathcal{C}\,|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|\,\mathcal{D}_\nu \subseteq (\!|\mathcal{U}\,(|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|)|\!)_\nu \tag{2}$$

  STS: $(m'' + 1, \text{fix } f(x).\delta^{\ulcorner} e^{\urcorner}) \in \mathcal{C}\,|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|\,\mathcal{D}_\nu$.
  Pick $j'' < m'' + 1$ and assume that $(j'', w) \in (\!|\mathcal{U}\,|\sigma\tau_1|\!|)_\nu$.
  STS: $(j'', \delta^{\ulcorner} e^{\urcorner}[w/x, F/f]) \in (\!|\mathcal{U}\,|\sigma\tau_2|\!|)^{\infty}_{\varepsilon}$.
  This follows by IH 5 on the second premise instantiated with $(j'', \delta[x \mapsto w, f \mapsto F]) \in \mathcal{G}(\!|x : \mathcal{U}\,|\sigma\tau_1|, f : \mathcal{U}\,(|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|), \mathcal{U}\,|\sigma\Gamma|\!|)$ which holds because

  - $(j'', \delta) \in \mathcal{G}(\!|\mathcal{U}\,|\sigma\Gamma|\!|)$ by downward closure (Lemma 33) on $(m, \delta) \in \mathcal{G}(\!|\mathcal{U}\,|\sigma\Gamma|\!|)$ using $j'' \leqslant m$.
  - $(j'', w) \in (\!|\mathcal{U}\,|\sigma\tau_1|\!|)_\nu$, from the assumption above
  - $(j'', \text{fix } f(x).\delta^{\ulcorner} e^{\urcorner}) \in (\!|\mathcal{U}\,(|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|)|\!)_\nu$, obtained by downward closure (Lemma 33) on (2) using $j'' \leqslant m''$

This completes the proof of this case.

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_1 : \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2 \mid t_1 \qquad \Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_2 : \tau_1 \mid t_2}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e_1\ e_2 : \tau_2 \mid t_1 + t_2 + t}$ **cp-app**

Assume that $\models \sigma\Phi$ and $(m, \delta) \in \mathcal{G}(\!|\mathcal{U}\,|\sigma\Gamma|\!|)$.
TS: $(m, \delta^{\ulcorner} e_1\ e_2^{\urcorner}) \in (\!|\mathcal{U}\,\sigma\tau_2|\!|)^{\infty}_{\varepsilon}$.
Following the definition of $(\!|\cdot|\!|^{\cdot}_{\varepsilon}$, assume that

$$\dfrac{\begin{array}{c} L(\delta^{\ulcorner} e_1^{\urcorner}) \Downarrow^{f_1} T_1 \quad (\star) \\ L(\delta^{\ulcorner} e_2^{\urcorner}) \Downarrow^{f_2} T_2 \quad (\diamond) \qquad \text{fix } f(x).e = V(T_1) \qquad v_2 = V(T_2) \\ e[v_2/x, (\text{fix } f(x).e)/f] \Downarrow^{f_r} T_r \quad (\dagger) \qquad v_r = V(T_r) \end{array}}{L(\delta^{\ulcorner} e_1^{\urcorner})\ L(\delta^{\ulcorner} e_2^{\urcorner}) \Downarrow^{f_1+f_2+f_r+c_{app}} \langle v_r, \text{app}(T_1, T_2, T_r)\rangle}\ \textbf{ev-app} \text{ and}$$

$$\dfrac{\begin{array}{c} R(\delta^{\ulcorner} e_1^{\urcorner}) \Downarrow^{f'_1} T'_1 \quad (\star\star) \\ R(\delta^{\ulcorner} e_2^{\urcorner}) \Downarrow^{f'_2} T'_2 \quad (\diamond\diamond) \qquad \text{fix } f(x).e' = V(T'_1) \qquad v'_2 = V(T'_2) \\ e[v'_2/x, (\text{fix } f(x).e')/f] \Downarrow^{f'_r} T'_r \quad (\dagger\dagger) \qquad v'_r = V(T'_r) \end{array}}{R(\delta^{\ulcorner} e_1^{\urcorner})\ R(\delta^{\ulcorner} e_2^{\urcorner}) \Downarrow^{f'_1+f'_2+f'_r+c_{app}} \langle v'_r, \text{app}(T'_1, T'_2, T'_r)\rangle}\ \textbf{ev-app} \text{ and}$$

$(f_1 + f_2 + f_r + c_{app}) < m$.

By IH 5 on the first premise, we get $(m, \delta^\ulcorner e_1 {}^\urcorner) \in (\!|\mathcal{U}\,|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)^{\sigma t_1}_{\varepsilon}$. Unrolling its definition with $(\star)$, $(\star\star)$ and $f_1 < m$, we get

a) $\langle T_1, \delta^\ulcorner e_1 {}^\urcorner\rangle \curvearrowright w_1', T_1', c_1'$

b) $w_1' = L(\text{fix } f(x).æ) \ \wedge\ \text{fix } f(x).e' = R(\text{fix } f(x).æ)$

c) $c_1' \leqslant \infty$

d) $(m - f_1, w_1') \in (\!|\mathcal{U}\,|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)_v$

By IH 5 on the second premise, we get $(m, \delta^\ulcorner e_2 {}^\urcorner) \in (\!|\mathcal{U}\,|\sigma\tau_1|\!)^{\sigma t_2}_{\varepsilon}$. Unrolling its definition with $(\diamond)$ and $(\diamond\diamond)$ and $f_2 < m$, we get

e) $\langle T_2, \delta^\ulcorner e_2 {}^\urcorner\rangle \curvearrowright w_2', T_2', c_2'$

f) $v_2 = L(w_2') \ \wedge\ v_2' = R(w_2')$

g) $c_2' \leqslant \infty$

h) $(m - f_2, w_2') \in (\!|\mathcal{U}\,|\sigma\tau_1|\!)_v$

There are two cases for d)

subcase 1: $w_1' = \mathsf{new}(\text{fix } f(x).e, \text{fix } f(x).e')$

By d), we have $(m - f_1, \mathsf{new}(\text{fix } f(x).e, \text{fix } f(x).e')) \in (\!|\mathcal{U}\,(|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|)|\!)_v$ $(\star)$

Now, we can conclude as follows:

1. Using a), e) and $(\dagger\dagger)$

$$\frac{\langle T_1, R(\delta^\ulcorner e_1 {}^\urcorner)\rangle \curvearrowright \mathsf{new}(\text{fix } f(x).e, \text{fix } f(x).e'), T_1', c_1' \qquad \langle T_2, R(\delta^\ulcorner e_2 {}^\urcorner)\rangle \curvearrowright w_2', T_2', c_2' \qquad e'[R(w_2')/x, (\text{fix } f(x).e')/f] \Downarrow^{f_r'} T_r' \qquad v_r' = V(T_r')}{\langle\langle v_r, \mathsf{app}(T_1, T_2, T_r)\rangle, R(\delta^\ulcorner e_1 {}^\urcorner)\ R(\delta^\ulcorner e_2 {}^\urcorner)\rangle \curvearrowright \mathsf{new}(v_r, v_r'), \langle v_r', \mathsf{app}(T_1', T_2', T_r')\rangle, c_1' + c_2' + f_r' + c_{app}}\ \textbf{cp-app-new}$$

2. Trivially, $v_r = L(\mathsf{new}(v_r, v_r')) \ \wedge\ v_r' = R(\mathsf{new}(v_r, v_r'))$

4. TS: $(m - (f_1 + f_2 + f_r + c_{app}), \mathsf{new}(v_r, v_r')) \in (\!|\mathcal{U}\,\sigma\tau_2|\!)_v$

STS: $\forall j.(j, v_r) \in [\![\sigma\tau_2]\!]_v \ \wedge\ (j, v_r') \in [\![\sigma\tau_2]\!]_v$ Pick j.

TS1: $(j, v_r) \in [\![\sigma\tau_2]\!]_v$
By unrolling the definition of $(\star)$, we have

$$\forall j.(j, \text{fix } f(x).e) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} \sigma|\tau_2|]\!]_v \wedge (j, \text{fix } f(x).e') \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_v \tag{1}$$

By lemma 32 on h), we get

$$\forall j.(j, L(w_2')) \in [\![U\,|\sigma\tau_1|]\!]_v \wedge (j, R(w_2')) \in [\![U\,|\sigma\tau_2|]\!]_v \tag{2}$$

Next, we instantiate eq. (1) with $j + f_r + 2$ and get

$$(j + f_r + 2, \text{fix } f(x).e) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_v \tag{3}$$

Then, we instantiate eq. (2) with $j + f_r + 1$ and get

$$(j + f_r + 1, L(w_2')) \in [\![U\,|\sigma\tau_1|]\!]_v \tag{4}$$

Unrolling the definition of eq. (3) using eq. (4) and $j + f_r + 1 < j + f_r + 2$, we get

$$(j + f_r + 1, e[L(w_2')/x, (\text{fix } f(x).e)/f]) \in [\![U\,|\sigma\tau_2|]\!]^{\infty}_{\varepsilon} \tag{5}$$

Unrolling the definition of eq. (5) with $(\dagger)$ and $f_r < j + f_r + 1$, we get

i) $f_r \leqslant \infty$
j) $(j + 1, v_r) \in [\![U\,|\sigma\tau_2|]\!]_v$

Then, we obtain $(j, v_r) \in [\![U\,|\sigma\tau_2|]\!]_v$ by downward closure (Lemma 33) on j) using $j \leqslant j + 1$.
This concludes TS1.
Next, we follow similar steps for the right projection as fol-

lows:
We next instantiate eq. (2) with $j + f'_r + 2$ and get

$$(j + f'_r + 2, \text{fix } f(x).e') \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_v \tag{6}$$

Then, we instantiate eq. (2) with $j + f'_r + 1$ and get

$$(j + f'_r + 1, R(w'_2)) \in [\![\mathcal{U}\,|\sigma\tau_1|]\!]_v \tag{7}$$

Unrolling the definition of eq. (6) using eq. (7) and $j + f'_r + 1 < j + f'_r + 2$, we get

$$(j + f'_r + 1, e[R(w'_2)/x, (\text{fix } f(x).e')/f]) \in [\![\mathcal{U}\,|\sigma\tau_2|]\!]^{\infty}_{\varepsilon} \tag{8}$$

Unrolling the definition of eq. (8) with (††) and
$f'_r < j + f'_r + 1$, we get

k) $f'_r \leqslant \infty$
l) $(j + 1, v'_r) \in [\![\mathcal{U}\,|\sigma\tau_2|]\!]_v$

Then, we obtain $(j, v'_r) \in [\![\mathcal{U}\,|\sigma\tau_2|]\!]_v$ by downward closure (Lemma 33) on l) using $j \leqslant j + 1$.
This concludes TS2.

3. Using c), g) and k), we get $(c'_1 + c'_2 + f'_r + c_{app}) \leqslant \infty$

subcase 2: $w'_1 = \texttt{fix } f(x).\text{ee}$
Then, by unrolling the definition of d), we have

$$(m - f_1, \texttt{fix } f(x).\text{ee}) \in \mathcal{C}\,|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|\,\mathcal{D}_v \tag{9}$$

Next, we apply downward-closure (lemma 33) to h) using

$$m - (f_1 + f_2 + c_{app}) \leqslant m - f_2$$

and we get

$$(m - (f_1 + f_2 + c_{app}), w_2') \in (\!|\mathcal{U}\,|\sigma\tau_1|\!|)_\nu \tag{10}$$

We unroll eq. (9) using (10) since

$m - (f_1 + f_2 + c_{app}) < m - f_1$ Note that here we have $c_{app} \geqslant 1$

and get

$$(m - (f_1 + f_2 + c_{app}), ee[w_2'/x, \text{fix } f(x).ee/f]) \in (\!|\mathcal{U}\,|\sigma\tau_2|\!|)_\varepsilon^{\sigma t} \tag{11}$$

Next, we unroll (11) using (†), (††) and $f_r < m - (f_1 + f_2 + c_{app})$ to obtain

i) $\langle T_r, ee[w_2'/x, \text{fix } f(x).ee/f]\rangle \curvearrowright w_r', T_r', c_r'$
j) $\nu_r = L(w_r') \ \wedge \ \nu_r' = R(w_r')$
k) $c_r' \leqslant \infty$
l) $(m - (f_1 + f_2 + f_r + c_{app}), w_r') \in (\!|\mathcal{U}\,|\sigma\tau_2|\!|)_\nu$

Now, we can conclude as follows:

1. Using a), e) and i)

$$\frac{\begin{array}{c}\langle T_1, R(\delta^\ulcorner e_1{}^\urcorner)\rangle \curvearrowright \texttt{fix } f(x).ee, T_1', c_1' \\ \langle T_2, R(\delta^\ulcorner e_2{}^\urcorner)\rangle \curvearrowright w_2', T_2', c_2' \\ \langle T_r, ee[w_2'/x, (\texttt{fix } f(x).ee)/f]\rangle \curvearrowright w_r', T_r', c_r' \qquad \nu_r' = V(T_r')\end{array}}{\begin{array}{c}\langle\langle \_, \texttt{app}(T_1, T_2, T_r)\rangle, R(\delta^\ulcorner e_1{}^\urcorner)\ R(\delta^\ulcorner e_2{}^\urcorner)\rangle \curvearrowright \\ w_r', \langle \nu_r', \texttt{app}(T_1', T_2', T_r')\rangle, c_1' + c_2' + c_r'\end{array}}\ \textbf{cp-app}$$

2. By j)
3. Using c), g) and k), we get $(c_1' + c_2' + c_r') \leqslant \infty$
4. By l)

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t \qquad \Delta; \Phi \models \tau \sqsubseteq \tau' \qquad \Delta; \Phi \models t \leqslant t'}{\Delta; \Phi_a; \Gamma \vdash_{\mathbb{CP}} e : \tau' \mid t'}\ \textbf{cp-}\sqsubseteq$

Assume that $\models \sigma\Phi$ and $(m, \delta) \in \mathcal{G}(\!|\mathcal{U}\,|\sigma\Gamma|\!|)$.

TS: $(m, \delta\ulcorner e \urcorner) \in (\!|\, U\,|\sigma\tau'|\,|\!)^{\sigma t'}_{\varepsilon}$.
Following the definition of $(\!|\cdot|\!)^{\cdot}_{\varepsilon}$, assume that

a) $L(\delta\ulcorner e \urcorner) \Downarrow^{f} T$
b) $R(\delta\ulcorner e \urcorner) \Downarrow^{f'} T'$
c) $f < m$

By IH 1 on the first premise using we get $(m, \delta\ulcorner e \urcorner) \in (\!|\sigma\tau|\!)^{\sigma t}_{\varepsilon}$.
Unrolling its definition with (a-c), we get

d) $\langle T, \delta\ulcorner e \urcorner\rangle \curvearrowright w', T', c'$
e) $v = L(w') \ \wedge \ v' = R(w')$
f) $c' \leqslant \sigma t$
g) $(m - f, w') \in (\!|\, U\,|\sigma\tau|\,|\!)_{v}$

We can conclude as follows:

1. By d)
2. By e)
3. By assumption 25 on the third premise, we get $\sigma t \leqslant \sigma t'$. Combining this with f), we get $c' \leqslant \sigma t'$.
4. By lemma 39(clause 11) on the second premise with g), we get $(m - c, w') \in (\!|\, U\,|\sigma\tau'|\,|\!)_{v}$

□

□

Theorem 47 (Fundamental theorem for bivalues/biexpressions)**.** *The following holds.*

1. *Assume that* $\Delta; \Phi_a; \Gamma \vdash \mathtt{w} \gg \tau$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$. *Then,* $(m, \delta\mathtt{w}) \in (\!|\sigma\tau|\!)_v$.

2. *Assume that* $\Delta; \Phi_a; \Gamma \vdash \mathtt{w} \gg \tau$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, \gamma) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$. *Then,* $(m, L(\delta\mathtt{w})) \in (\!|\sigma\tau|\!)_v \wedge (m, R(\delta\mathtt{w})) \in (\!|\sigma\tau|\!)_v$.

3. *Assume that* $\Delta; \Phi_a; \Gamma \vdash \mathtt{w} \gg \tau$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, \delta) \in \mathcal{G}(\!|\mathsf{U}\,|\sigma\Gamma|\!|)$. *Then,* $(m, \delta\mathtt{w}) \in (\!|\mathsf{U}\,|\sigma\tau|\!|)_v$.

4. *Assume that* $\Delta; \Phi_a; \Gamma \vdash \mathtt{ee} \gg \tau \mid \mathtt{t}$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$. *Then,* $(m, \delta\mathtt{ee}) \in (\!|\sigma\tau|\!)^{\sigma\mathtt{t}}_{\varepsilon}$.

5. *Assume that* $\Delta; \Phi_a; \Gamma \vdash \mathtt{ee} \gg \tau \mid \mathtt{t}$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, \gamma) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$. *Then,* $(m, L(\gamma\mathtt{ee})) \in (\!|\sigma\tau|\!)^{\infty}_{\varepsilon} \wedge (m, R(\gamma\mathtt{ee})) \in (\!|\sigma\tau|\!)^{\infty}_{\varepsilon}$.

6. *Assume that* $\Delta; \Phi_a; \Gamma \vdash \mathtt{ee} \gg \tau \mid \mathtt{t}$ *and* $\sigma \in \mathcal{D}[\![\Delta]\!]$ *and* $\models \sigma\Phi$ *and* $(m, \delta) \in \mathcal{G}(\!|\mathsf{U}\,|\sigma\Gamma|\!|)$. *Then,* $(m, \delta\mathtt{ee}) \in (\!|\mathsf{U}\,|\sigma\tau|\!|)^{\infty}_{\varepsilon}$.

Proof. Proofs are by induction on typing derivations with a sub-induction on step-indices for recursive functions. We show select cases of each statement separately.

Proof of Statement (1). We proceed by induction on the bi-value typing derivation. We show the most important cases below.

Case: $\dfrac{}{\Delta; \Phi; \Gamma \vdash \mathsf{keep}(\mathsf{n}) \gg \mathbf{int}_r}$ **bi-keep**

Assume that $\models \sigma\Phi$ and $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$.

TS: $(m, \delta(\mathsf{keep}(\mathsf{n}))) \in (\!|\mathrm{int}_r|\!)_v$.

This immediately follows from the definition of $(\!|\mathrm{int}_r|\!)_v$.

Case: $\dfrac{\Delta; \Phi; \cdot \vdash_{\mathbb{FS}} v : A \mid \mathtt{t} \qquad \Delta; \Phi; \cdot \vdash_{\mathbb{FS}} v' : A \mid \mathtt{t}'}{\Delta; \Phi; \Gamma \vdash \mathsf{new}(v, v') \gg \mathsf{U}\,A}$ **bi-new**

Assume that $\models \sigma\Phi$ and $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$.

TS: $(m, \delta(\mathsf{new}(v, v'))) \in (\!|\mathsf{U}\ \sigma A|\!)_v$.
STS: $\forall j.(j, v) \in [\![\sigma A]\!]_v \ \wedge\ (j, v') \in [\![\sigma A]\!]_v$.
Pick j.
RTS1: $(j, v) \in [\![\sigma A]\!]_v$
RTS2: $(j, v') \in [\![\sigma A]\!]_v$.
Next, we will instantiate theorem 46 (second clause) on the first and second premises.
For the first premise, we know that $(j+1, \cdot) \in \mathcal{G}(\!|\cdot|\!)$ (by definition).
Hence, by instantiating theorem 46 (second clause) on the first premise with $(j+1, \cdot) \in \mathcal{G}(\!|\cdot|\!)$, we get $(j+1, v) \in [\![\sigma A]\!]^{\sigma t}_{\varepsilon}$.
To unroll its definition we use

a) Since v is a value, by **ev-value** rule, we have $v \Downarrow^0 \langle v, v\rangle$.
b) $0 < j+1$

Therefore, we get

$$(j+1, v) \in [\![\sigma A]\!]_v \tag{1}$$

Next, we obtain the first statement $(j, v) \in [\![\sigma A]\!]_v$ by downward closure (Lemma 33) on Equation (1) using $j \leqslant j+1$.
Similarly, we instantiate theorem 46 (second clause) on the second premises and we get $(j+1, v') \in [\![\sigma A]\!]^{\sigma t'}_{\varepsilon}$.
To unroll its definition we use

a) Since v' is a value, by **ev-value** rule, we have $v' \Downarrow^0 \langle v', v'\rangle$.
b) $0 < j+1$

Therefore, we get

$$(j+1, v') \in [\![\sigma A]\!]_v \tag{2}$$

Hence, we obtain the second statement $(j, v') \in [\![\sigma A]\!]_v$ by downward closure (Lemma 33) on Equation (2) using $j \leqslant j+1$.

Case: $\dfrac{\Delta;\Phi;\Gamma \vdash w_1 \gg \tau \qquad \Delta;\Phi;\Gamma \vdash w_2 \gg \mathbf{list}[n]^{\alpha}\,\tau}{\Delta;\Phi;\Gamma \vdash \mathtt{cons}(w_1,w_2) \gg \mathbf{list}[n+1]^{\alpha+1}\,\tau}$ **bi-cons**

Assume that $(m,\delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m, \mathtt{cons}(\delta w_1, \delta w_2)) \in (\!|\mathrm{list}[\sigma n+1]^{\sigma\alpha+1}\,\sigma\tau|\!)_v$.

By IH 1 on the first premise, we get $(m,\delta w_1) \in (\!|\sigma\tau|\!)_v$ $(\star)$.

By IH 1 on the second premise, we get $(m,\delta w_2) \in (\!|\mathrm{list}[\sigma n]^{\sigma\alpha}\,\sigma\tau|\!)_v$ $(\diamond)$.

By using $(\star)$ and $(\diamond)$, we can conclude as follows:

$(m, \mathtt{cons}(\delta w_1, \delta w_2)) \in (\!|\mathrm{list}[\sigma n+1]^{\sigma\alpha+1}\,\sigma\tau|\!)_v$.

Case: $\dfrac{\Delta;\Phi;\Gamma \vdash w_1 \gg \Box\tau \qquad \Delta;\Phi;\Gamma \vdash w_2 \gg \mathbf{list}[n]^{\alpha}\,\tau}{\Delta;\Phi;\Gamma \vdash \mathtt{cons}(w_1,w_2) \gg \mathbf{list}[n+1]^{\alpha}\,\tau}$ **bi-cons-□**

Assume that $(m,\delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m, \mathtt{cons}(\delta w_1, \delta w_2)) \in (\!|\mathrm{list}[\sigma n+1]^{\sigma\alpha}\,\sigma\tau|\!)_v$.

By IH 1 on the first premise, we get $(m,\delta w_1) \in (\!|\Box\,\sigma\tau|\!)_v$ $(\star)$.

By IH 1 on the second premise, we get $(m,\delta w_2) \in (\!|\mathrm{list}[\sigma n]^{\sigma\alpha}\,\sigma\tau|\!)_v$ $(\diamond)$.

By using $(\star)$ and $(\diamond)$, we can conclude as follows:

$(m, \mathtt{cons}(\delta w_1, \delta w_2)) \in (\!|\mathrm{list}[\sigma n+1]^{\sigma\alpha}\,\sigma\tau|\!)_v$.

Case: $\dfrac{\Delta;\Phi;x:\tau_1,f:\tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2,\Gamma \vdash ee \gg \tau_2 \mid t}{\Delta;\Phi;\Gamma \vdash \mathtt{fix}\ f(x).\,ee \gg \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2}$ **bi-fix**

Assume that $(m,\delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m, \mathrm{fix}\ f(x).\delta ee) \in (\!|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)_v$.

Let $F = \mathrm{fix}\ f(x).\delta ee$.

We prove the more general statement

$$\forall\, m' \leqslant m.\ (m', F) \in (\!|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)_v$$

by subinduction on m'.

There are two parts to show:

subcase 1: $m' = 0$

By the definition of the interpretation of function types, there are two parts to show:

subsubcase 1: $\forall j < m' = 0 \cdots$

Since there is no non-negative j such that $j < 0$, the goal is vacuously true.

subsubcase 2: TS: $(0, F) \in \mathbb{C}\, |\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|\, \mathbb{D}_\nu$

As above, since there is no $j < 0$,

RTS: $\forall j.(j, L(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu \wedge (j, R(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$.

Pick j.

We show the left projection only, the right one is similar.

- STS 1: $(j, L(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$

  We prove the more general statement

  $$\forall m' \leqslant j.\ (m', L(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$$

  by subinduction on m'.

  There are two cases:

  - $m' = 0$

    Since there is no non-negative j such that $j < 0$, the goal is vacuously true.

  - $m' = m'' + 1 \leqslant m$

    By sub-IH

    $$(m'', \text{fix } f(x).L(\delta^\ulcorner e^\urcorner)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu \quad (1)$$

    STS: $(m'' + 1, \text{fix } f(x).L(\delta^\ulcorner e^\urcorner)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$.

    Pick $j'' < m'' + 1$ and assume that $(j'', \nu) \in [\![|\sigma\tau_1|]\!]_\nu$.

    STS: $(j'', L(\delta^\ulcorner e^\urcorner)[\nu/x, L(F)/f]) \in [\![|\sigma\tau_2|]\!]^\infty_\varepsilon$.

    This follows by IH 5 on the premise instantiated with $(j'', \delta[x \mapsto \nu, f \mapsto L(F)]) \in \mathcal{G}[\![x : |\sigma\tau_1|, f : |\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|, |\sigma\Gamma|]\!]$ which holds because

    * $(j'', \delta) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$ using lemma 32 on $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$
    * $(j'', \nu) \in [\![|\sigma\tau_1|]\!]_\nu$, from the assumption above

* $(j'', \text{fix } f(x).L(\delta^\ulcorner e^\urcorner)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$, obtained by downward closure (Lemma 33) on (1) using $j'' \leqslant m''$

subcase 2: $m' = m'' + 1 \leqslant m$

By sub-IH

$$(m'', F) \in (\!|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)_\nu \tag{2}$$

TS: $(m'' + 1, \text{fix } f(x).\delta\text{ee}) \in (\!|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)_\nu$

There are two cases to show:

subsubcase 1: Pick $j < m'' + 1$ and assume that $(j, w) \in (\!|\sigma\tau_1|\!)_\nu$.

STS: $(j, \delta^\ulcorner e^\urcorner[w/x, F/f]) \in (\!|\sigma\tau_2|\!)^{\sigma t}_\varepsilon$.

This follows by IH 4 on the second premise instantiated with $(j, \delta[x \mapsto w, f \mapsto F]) \in \mathcal{G}(\!|\sigma\Gamma, x : \sigma\tau_1, f : \sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)$ which holds because

- $(j, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ obtained by downward closure (lemma 33) using $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $j < m' \leqslant m$.
- $(j, w) \in (\!|\sigma\tau_1|\!)_\nu$, from the assumption above
- $(j, F) \in (\!|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)_\nu$, obtained by downward closure (Lemma 33) on (2) using $j \leqslant m''$

subsubcase 2: STS: $(m'' + 1, \text{fix } f(x).\delta^\ulcorner e^\urcorner) \in \mathbb{C}\,|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|\,\mathbb{D}_\nu$

There are also two cases to show here.

- Pick $j < m'' + 1$ and assume that $(j, w) \in (\!|\mathbb{U}\,|\sigma\tau_1|\!|)_\nu$.
  STS: $(j, \delta^\ulcorner e^\urcorner[w/x, F/f]) \in (\!|\mathbb{U}\,\sigma\tau_2|\!|)^\infty_\varepsilon$.
  This follows by IH 6 on the second premise instantiated with
  $(j, \delta[x \mapsto w, f \mapsto F]) \in \mathcal{G}(\!|\mathbb{U}\,|\sigma\Gamma|, x : \mathbb{U}\,|\sigma\tau_1|, f : \mathbb{U}\,(|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|)|\!)$ which holds because
  - $(j, \delta) \in \mathcal{G}(\!|\mathbb{U}\,|\sigma\Gamma|\!|)$ obtained by downward closure (Lemma 33) on $(m, \delta) \in \mathcal{G}(\!|\mathbb{U}\,|\sigma\Gamma|\!|)$ (obtained by inclusion lemma on $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$) and $j < m' \leqslant m$.

- $(j, \mathbf{w}) \in (\!|\mathcal{U}\,|\sigma\tau_1|\,|\!)_\nu$, from the assumption above
- $(j, F) \in (\!|\mathcal{U}\,(|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|)|\!)_\nu$, obtained by downward closure (Lemma 33) on (2) using $j \leqslant m''$

• STS: $\forall j.(j, L(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu \wedge (j, R(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$.

The proof is same as above subcase where $m' = 0$.

This completes the proof of this case.

Case: $\dfrac{\Delta; \Phi; x : \tau_1, f : \Box\,(\tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2), \Gamma \vdash æ \gg\!> \tau_2 \mid t \quad \forall x \in \Gamma.\ \Delta; \Phi \models \Gamma(x) \sqsubseteq \Box\,\Gamma(x) \qquad \texttt{stable}(æ)}{\Delta; \Phi; \Gamma, \Gamma' \vdash \texttt{fix}\ f(x).æ \gg\!> \Box\,(\tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2)}$ **bi-fix-NC**

Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma, \sigma\Gamma'|\!)$ and $\models \sigma\Phi$.

Then, $\delta = \delta_1 \cup \delta_2$ such that $(m, \delta_1) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $(m, \delta_2) \in \mathcal{G}(\!|\sigma\Gamma'|\!)$.

TS: $(m, \text{fix } f(x).\delta\ulcorner e\urcorner) \in (\!|\Box\,(\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2)|\!)^{\emptyset}_{\varepsilon}$.

Since e doesn't have any free variables from Γ' by the second premise,

TS: $(m, \text{fix } f(x).\delta_1\ulcorner e\urcorner) \in (\!|\Box\,(\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2)|\!)^{\emptyset}_{\varepsilon}$.

By lemma 31, STS: $(m, \text{fix } f(x).\delta_1\ulcorner e\urcorner) \in (\!|\Box\,(\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2)|\!)_\nu$.

By lemma 39 using $(m, \delta_1) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and the third premise, we get $(m, \delta_1) \in \mathcal{G}(\!|\Box\,\sigma\Gamma|\!)$, i.e. $\forall x \in \text{dom}(\Gamma).\texttt{stable}(\delta_1(x))$.

We also know that by definition, $\texttt{stable}(\ulcorner e\urcorner)$.

Hence, $\texttt{stable}(\text{fix } f(x).\delta_1\ulcorner e\urcorner)$.

Therefore, STS: $(m, \text{fix } f(x).\delta_1\ulcorner e\urcorner) \in (\!|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)_\nu$.

Let $F = \text{fix } f(x).\delta_1 e$.

We prove the more general statement

$$\forall\, m' \leqslant m.\ (m', F) \in (\!|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)_\nu$$

by subinduction on m'.

There are two parts to show:

subcase 1: $m' = 0$

By the definition of the interpretation of function types, there are two parts to show:

subsubcase 1: $\forall j < m' = 0 \cdots$
Since there is no non-negative j such that $j < 0$, the goal is vacuously true.

subsubcase 2: TS: $(0, F) \in (\!| |\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2| |\!)_\nu$
As above, since there is no $j < 0$,
RTS: $\forall j.(j, L(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu \wedge (j, R(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$.
Pick j.
We show the left projection only, the right one is similar.

- STS 1: $(j, L(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$
  We prove the more general statement

$$\forall m' \leqslant j.\ (m', L(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$$

  by subinduction on m'.
  There are two cases:

  - $m' = 0$
    Since there is no non-negative j such that $j < 0$, the goal is vacuously true.
  - $m' = m'' + 1 \leqslant m$
    By sub-IH

$$(m'', \text{fix } f(x).L(\delta_1 \ulcorner e \urcorner)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu \quad (1)$$

    STS: $(m'' + 1, \text{fix } f(x).L(\delta_1 \ulcorner e \urcorner)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$.
    Pick $j'' < m'' + 1$ and assume that $(j'', \nu) \in [\![|\sigma\tau_1|]\!]_\nu$.
    STS: $(j'', L(\delta_1 \ulcorner e \urcorner)[\nu/x, L(F)/f]) \in [\![|\sigma\tau_2|]\!]^\infty_\varepsilon$.
    This follows by IH 5 on the premise instantiated with $(j'', \delta_1[x \mapsto \nu, f \mapsto L(F)]) \in \mathcal{G}[\![x : |\sigma\tau_1|, f : |\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|, |\sigma\Gamma|]\!]$ which holds because

    * $(j'', \delta_1) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$ using lemma 32 on $(m, \delta_1) \in \mathcal{G}(\!|\sigma\Gamma|\!)$
    * $(j'', \nu) \in [\![|\sigma\tau_1|]\!]_\nu$, from the assumption above

* $(j'', \text{fix } f(x).L(\delta_1\ulcorner e\urcorner)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_v$, obtained by downward closure (Lemma 33) on (1) using $j'' \leqslant m''$

subcase 2: $m' = m'' + 1 \leqslant m$

By sub-IH

$$(m'', F) \in (\!|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)_v \tag{2}$$

TS: $(m'' + 1, \text{fix } f(x).\delta_1\ulcorner e\urcorner) \in (\!|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)_v$

There are two cases to show:

subsubcase 1: Pick $j < m'' + 1$ and assume that $(j, w) \in (\!|\sigma\tau_1|\!)_v$.

STS: $(j, \delta_1\ulcorner e\urcorner[w/x, F/f]) \in (\!|\sigma\tau_2|\!)^{\sigma t}_{\varepsilon}$.

This follows by IH 4 on the second premise instantiated with $(j, \delta_1[x \mapsto w, f \mapsto F]) \in \mathcal{G}(\!|\sigma\Gamma, x : \sigma\tau_1, f : \square(\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2)|\!)$ which holds because

- $(j, \delta_1) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ obtained by downward closure (lemma 33) using $(m, \delta_1) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $j < m' \leqslant m$.
- $(j, w) \in (\!|\sigma\tau_1|\!)_v$, from the assumption above
- $(j, F) \in (\!|\square(\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2)|\!)_v$, obtained by downward closure (Lemma 33) on (2) using $j \leqslant m''$ and also by $\texttt{stable}(F)$

subsubcase 2: STS: $(m'' + 1, \text{fix } f(x).\delta_1\ulcorner e\urcorner) \in \mathbb{C}\,|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|\,\mathbb{D}_v$

There are also two cases to show here.

- Pick $j < m'' + 1$ and assume that $(j, w) \in (\!|\mathcal{U}\,|\sigma\tau_1|\!|)_v$.
  STS: $(j, \delta_1\ulcorner e\urcorner[w/x, F/f]) \in (\!|\|\mathcal{U}\,\sigma\tau_2|\!|)^{\infty}_{\varepsilon}$.
  This follows by IH 6 on the second premise instantiated with
  $(j, \delta_1[x \mapsto w, f \mapsto F]) \in \mathcal{G}(\!|\mathcal{U}\,|\sigma\Gamma|, x : \mathcal{U}\,|\sigma\tau_1|, f : \mathcal{U}\,(|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|)|\!)$ which holds because
  - $(j, \delta_1) \in \mathcal{G}(\!|\mathcal{U}\,|\sigma\Gamma|\!|)$ obtained by downward closure (Lemma 33) on $(m, \delta_1) \in \mathcal{G}(\!|\mathcal{U}\,|\sigma\Gamma|\!|)$ (obtained by inclusion lemma on $(m, \delta_1) \in \mathcal{G}(\!|\sigma\Gamma|\!)$) and $j < m' \leqslant m$.

– $(j, w) \in (\!|\mathsf{U}\,|\sigma\tau_1|\!|)_\nu$, from the assumption above

– $(j, F) \in (\!|\mathsf{U}\,(|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|)|\!|)_\nu$, obtained by downward closure (Lemma 33) on (2) using $j \leqslant m''$

- STS: $\forall j.(j, \mathrm{L}(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu \wedge (j, \mathrm{R}(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$.
  The proof is same as above subcase where $m' = 0$.

This completes the proof of this case.

Case: $\dfrac{\Delta; \Phi; \Gamma \vdash w \gg \tau \qquad \forall x \in \Gamma.\ \Delta; \Phi \models \Gamma(x) \sqsubseteq \Box\Gamma(x) \qquad \mathtt{stable}(w)}{\Delta; \Phi; \Gamma, \Gamma' \vdash w \gg \Box\tau}$ **bi-nochange**

Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma, \sigma\Gamma'|\!)$ and $\models \sigma\Phi$.

Then, $\delta = \delta_1 \cup \delta_2$ such that $(m, \delta_1) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $(m, \delta_2) \in \mathcal{G}(\!|\sigma\Gamma'|\!)$.

TS: $(m, \delta w) \in (\!|\Box\,\sigma\tau|\!)_\nu$.

Since w doesn't have any free variables from Γ' by the first premise,

STS: $(m, \delta_1 w) \in (\!|\Box\,\sigma\tau|\!)_\nu$.

RTS1: $(m, \delta_1 w) \in (\!|\sigma\tau|\!)_\nu$.

RTS2: $\mathtt{stable}(\delta_1 w)$.

The first part can be shown by By IH 1 on the first premise.

The second part can be shown by lemma 39 using $(m, \delta_1) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and the second premise, i.e. $(m, \delta_1) \in \mathcal{G}(\!|\Box\,\sigma\Gamma|\!)$. This means that $\forall x \in \mathrm{dom}(\Gamma).\ \mathtt{stable}(\delta_1(x))$.

Therefore, since $\mathtt{stable}(w)$, we have $\mathtt{stable}(\delta_1 w)$.

Case: $\dfrac{\Delta; \Phi; \Gamma \vdash w \gg \tau \qquad \Delta; \Phi \models \tau \sqsubseteq \tau'}{\Delta; \Phi; \Gamma \vdash w \gg \tau'}$ **bi-$\sqsubseteq$**

Assume that $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$ and $\models \sigma\Phi$.

TS: $(m, \delta w) \in (\!|\sigma\tau'|\!)_\nu$.

By IH 1 on the first premise, we have $(m, \delta w) \in (\!|\sigma\tau|\!)_\nu$ $(\star)$.

By lemma 39 on the second premise with $(\star)$, we get $(m, \delta w) \in (\!|\sigma\tau'|\!)_\nu$.

□

Proof of Statement (2). We proceed by induction on the bi-value typing derivation. For brevity, we show the most important cases of the left projection below; the right one can be obtained similarly.

Case: $\dfrac{}{\Delta;\Phi;\Gamma \vdash \mathsf{keep}(n) \gg \mathbf{int}_r}$ **bi-keep**

Assume that $\models \sigma\Phi$ and $(m,\gamma) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$.

TS: $(m, \mathrm{L}((\gamma\mathsf{keep}(n)))) \in (\!|\mathbf{int}|\!)_\nu$.

This immediately follows from the definition of $(\!|\mathbf{int}|\!)_\nu$.

Case: $\dfrac{\Delta;\Phi;\cdot \vdash_{\mathbb{FS}} v : A \mid t \qquad \Delta;\Phi;\cdot \vdash_{\mathbb{FS}} v' : A \mid t'}{\Delta;\Phi;\Gamma \vdash \mathsf{new}(v,v') \gg \mathrm{U}\, A}$ **bi-new**

Assume that $\models \sigma\Phi$ and $(m,\gamma) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$.

TS: $(m, \gamma\mathrm{L}(\mathsf{new}(v,v'))) \in [\![|\mathrm{U}\,\sigma A|]\!]_\nu = [\![\sigma A]\!]_\nu$.

STS: $(m,v) \in [\![\sigma A]\!]_\nu$.

Next, we will instantiate theorem 46 (second clause) on the first premise.

For the first premise, we know that $(m+1,\cdot) \in \mathcal{G}(\!|\cdot|\!)$ (by definition).

Hence, by instantiating theorem 46 (second clause) on the first premise with $(m+1,\cdot) \in \mathcal{G}(\!|\cdot|\!)$, we get $(m+1,v) \in [\![\sigma A]\!]^{\mathsf{ot}}_\varepsilon$.

To unroll its definition we use

a) Since v is a value, by **ev-value** rule, we have $v \Downarrow^0 \langle v,v\rangle$.

b) $0 < m+1$

Therefore, we get

$$(m+1,v) \in [\![\sigma A]\!]_\nu \tag{1}$$

Next, we obtain the first statement $(m,v) \in [\![\sigma A]\!]_\nu$ by downward closure (Lemma 33) on Equation (1) using $m \leqslant m+1$.

Case: $\dfrac{\Delta;\Phi;\Gamma \vdash w_1 \gg \tau \qquad \Delta;\Phi;\Gamma \vdash w_2 \gg \mathbf{list}[n]^\alpha\,\tau}{\Delta;\Phi;\Gamma \vdash \mathsf{cons}(w_1,w_2) \gg \mathbf{list}[n+1]^{\alpha+1}\,\tau}$ **bi-cons**

Assume that $\models \sigma\Phi$ and $(m,\gamma) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$.

TS: $(m, \mathsf{cons}(L(\gamma w_1), L(\gamma w_2))) \in [\![|\mathrm{list}[\sigma n + 1]^{\sigma\alpha+1}\ \sigma\tau|]\!]_v \equiv [\![\mathrm{list}[\sigma n + 1]\,|\sigma\tau|]\!]_v$.

By IH 2 on the first premise, we get $(m, L(\gamma w_1)) \in [\![|\sigma\tau|]\!]_v$ $(\star)$.

By IH 2 on the second premise, we get $(m, L(\gamma w_2)) \in [\![|\mathrm{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|]\!]_v \equiv [\![\mathrm{list}[\sigma n]\,|\sigma\tau|]\!]_v$ $(\diamond)$.

By using $(\star)$ and $(\diamond)$, we can conclude as follows:

$(m, \mathsf{cons}(L(\gamma w_1), L(\gamma w_2))) \in [\![\mathrm{list}[\sigma n + 1]\,|\sigma\tau|]\!]_v \equiv [\![|\mathrm{list}[\sigma n + 1]^{\sigma\alpha+1}\ \sigma\tau|]\!]_v$.

Case: $\dfrac{\Delta; \Phi; \Gamma \vdash w_1 \gg \Box\tau \qquad \Delta; \Phi; \Gamma \vdash w_2 \gg \mathbf{list}[n]^{\alpha}\,\tau}{\Delta; \Phi; \Gamma \vdash \mathsf{cons}(w_1, w_2) \gg \mathbf{list}[n+1]^{\alpha}\,\tau}$ **bi-cons-□**

Assume that $\models \sigma\Phi$ and $(m, \gamma) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$.

TS: $(m, \mathsf{cons}(L(\gamma w_1), L(\gamma w_2))) \in [\![|\mathrm{list}[\sigma n + 1]^{\sigma\alpha}\ \sigma\tau|]\!]_v \equiv [\![\mathrm{list}[\sigma n + 1]\,|\sigma\tau|]\!]_v$.

By IH 2 on the first premise, we get $(m, L(\gamma w_1)) \in [\![|\Box\,\sigma\tau|]\!]_v$ $(\star)$.

By IH 2 on the second premise, we get $(m, L(\gamma w_2)) \in [\![|\mathrm{list}[\sigma n]^{\sigma\alpha}\ \sigma\tau|]\!]_v \equiv [\![\mathrm{list}[\sigma n]\,|\sigma\tau|]\!]_v$ $(\diamond)$.

By using $(\star)$ and $(\diamond)$, we can conclude as follows:

$(m, \mathsf{cons}(L(\gamma w_1), L(\gamma w_2))) \in [\![\mathrm{list}[\sigma n + 1]\,|\sigma\tau|]\!]_v \equiv [\![|\mathrm{list}[\sigma n + 1]^{\sigma\alpha}\ \sigma\tau|]\!]_v$.

Case: $\dfrac{\Delta; \Phi; x : \tau_1, f : \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2, \Gamma \vdash ee \gg \tau_2 \mid t}{\Delta; \Phi; \Gamma \vdash \mathsf{fix}\ f(x).\,ee \gg \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2}$ **bi-fix**

Assume that $\models \sigma\Phi$ and $(m, \gamma) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$.

TS: $(m, \mathrm{fix}\ f(x).L(\gamma ee)) \in [\![|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|]\!]_v = [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_v$.

We prove the more general statement

$$\forall m' \leqslant m.\ (m', \mathrm{fix}\ f(x).L(\gamma ee)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_v$$

by subinduction on m'.

There are two cases:

subcase 1: $m' = 0$

Since there is no non-negative j such that $j < 0$, the goal is vacuously true.

subcase 2: $m' = m'' + 1 \leqslant m$

By sub-IH

$$(m'', \text{fix } f(x).L(\gamma @ @)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu \tag{1}$$

STS: $(m'' + 1, \text{fix } f(x).L(\gamma @ @)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$.
Pick $j < m'' + 1$ and assume that $(j, \nu) \in [\![|\sigma\tau_1|]\!]_\nu$.
STS: $(j, L(\gamma @ @)[\nu/x, (\text{fix } f(x).L(\gamma @ @))/f]) \in [\![|\sigma\tau_2|]\!]^\infty_\varepsilon$.
This follows by IH 4 on the premise instantiated with

- $(j, \gamma[x \mapsto \nu, f \mapsto (\text{fix } f(x).L(\gamma @ @))]) \in \mathcal{G}[\![x : |\sigma\tau_1|, f : |\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|, |\sigma\Gamma|]\!]$ which holds because
  - $(j, \gamma) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$ using downward closure (Lemma 33) on $(m, \gamma) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$ using $j < m'' + 1 \leqslant m$.
  - $(j, \nu) \in [\![|\sigma\tau_1|]\!]_\nu$, from the assumption above
  - $(j, \text{fix } f(x).L(\gamma @ @)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$, obtained by downward closure (Lemma 33) on (1) using $j \leqslant m''$

Case: $\dfrac{\Delta; \Phi; \Gamma \vdash w \gg \tau \qquad \forall x \in \Gamma.\ \Delta; \Phi \models \Gamma(x) \sqsubseteq \square\Gamma(x) \qquad \texttt{stable}(w)}{\Delta; \Phi; \Gamma, \Gamma' \vdash w \gg \square\tau}$ **bi-nochange**

Assume that $\models \sigma\Phi$ and $(m, \gamma) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$.
Then, $\gamma = \gamma_1 \cup \gamma_2$ such that $(m, \gamma_1) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$ and $(m, \gamma_2) \in \mathcal{G}[\![|\sigma\Gamma|']\!]$.
TS: $(m, L(\gamma w)) \in [\![|\square\,\sigma\tau|]\!]_\nu$.
Since w doesn't have any free variables from Γ' by the first premise,
STS: $(m, \gamma_1 w) \in [\![|\square\,\sigma\tau|]\!]_\nu = [\![|\sigma\tau|]\!]_\nu$.
This follows by IH 2 on the first premise.

□

Proof of Statement (3). Remember that we are trying to prove:
Assume that $\Delta; \Phi_a; \Gamma \vdash w \gg \tau$ and $\sigma \in \mathcal{D}[\![\Delta]\!]$ and $\models \sigma\Phi$ and $(m, \delta) \in \mathcal{G}(\!|\mathcal{U}\,|\sigma\Gamma|)\!)$. Then, $(m, \delta w) \in (\!|\mathcal{U}\,|\sigma\tau|)\!)_\nu$.
Proof is by induction on the bi-value typing.

Case: $\dfrac{\Delta; \Phi; x : \tau_1, f : \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2, \Gamma \vdash \text{æ} \gg \tau_2 \mid t}{\Delta; \Phi; \Gamma \vdash \texttt{fix}\ f(x).\,\text{æ} \gg \tau_1 \xrightarrow{\mathbb{CP}(t)} \tau_2}$ **bi-fix**

Assume that $\models \sigma\Phi$ and $(m, \delta) \in \mathcal{G}(\!|\mathcal{U}\,|\sigma\Gamma|\!)$.

TS: $(m, \text{fix}\ f(x).\delta\text{æ}) \in (\!|\mathcal{U}\,|\sigma\tau_1 \xrightarrow{\mathbb{CP}(\sigma t)} \sigma\tau_2|\!)_\nu = (\!|\mathcal{U}\,(|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|)|\!)_\nu$.

Let $F = \text{fix}\ f(x).\delta\text{æ}$.

By definition of $(\!|\mathcal{U}\,\cdot|\!)_\nu$, since $\text{fix}\ f(x).\delta\text{æ} \neq \texttt{new}(\cdot,\cdot)$,

STS: $(m, \text{fix}\ f(x).\delta\text{æ}) \in \mathcal{C}\,|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|\,\mathcal{D}_\nu$.

Let $F = \text{fix}\ f(x).\delta\text{æ}$.

We prove the more general statement

$$\forall m' \leqslant m.\ (m', F) \in \mathcal{C}\,|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|\,\mathcal{D}_\nu$$

by subinduction on m'.

There are three cases:

- STS: $\forall j.(j, L(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu \ \wedge\ (j, R(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$.

  Pick j.

  We show the left projection only, the right one is similar.

  - STS 1: $(j, L(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$

    We prove the more general statement

    $$\forall m' \leqslant j.\ (m', L(F)) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu$$

    by subinduction on m'.

    There are two cases:

    * $m' = 0$

      Since there is no non-negative j such that $j < 0$, the goal is vacuously true.

    * $m' = m'' + 1 \leqslant j$

      By sub-IH

      $$(m'', \text{fix}\ f(x).L(\delta\text{æ})) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_\nu \qquad (1)$$

STS: $(m'' + 1, \text{fix } f(x).L(\delta \text{ee})) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_v$.
Pick $j'' < m'' + 1$ and assume that $(j'', v) \in [\![|\sigma\tau_1|]\!]_v$.
STS: $(j'', L(\delta \text{ee})[v/x, L(F)/f]) \in [\![\sigma A_2]\!]^{\infty}_{\varepsilon}$.
This follows by IH 5 on the second premise instantiated with $(j'', \delta[x \mapsto v, f \mapsto L(F)]) \in \mathcal{G}[\![x : \sigma A_1, f : |\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|, |\sigma\Gamma|]\!]$ which holds because

- $(j'', L(\delta)) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$ using lemma 32 on $(m, \delta) \in \mathcal{G}(\!|\mathcal{U}\,|\sigma\Gamma|)\!)$
- $(j'', v) \in [\![|\sigma\tau_1|]\!]_v$, from the assumption above
- $(j'', \text{fix } f(x).L(\delta \text{ee})) \in [\![|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|]\!]_v$, obtained by downward closure (Lemma 33) on (1) using $j'' \leqslant m''$

- $m' = 0$
  Since there is no non-negative j such that $j < 0$, the goal is vacuously true.
- $m' = m'' + 1 \leqslant m$
  By sub-IH

$$(m'', \text{fix } f(x).\delta \text{ee}) \in \mathcal{C}\,|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|\,\mathcal{D}_v \subseteq (\!|\mathcal{U}\,(|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|)|\!)_v \tag{2}$$

STS: $(m'' + 1, \text{fix } f(x).\delta \text{ee}) \in \mathcal{C}\,|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|\,\mathcal{D}_v$.
Pick $j'' < m'' + 1$ and assume that $(j'', w) \in (\!|\mathcal{U}\,|\sigma\tau_1|)\!)_v$.
STS: $(j'', \delta \text{ee}[w/x, F/f]) \in (\!|\mathcal{U}\,|\sigma\tau_2|)\!)^{\infty}_{\varepsilon}$.
This follows by IH 6 on the second premise instantiated with $(j'', \delta[x \mapsto w, f \mapsto F]) \in \mathcal{G}(\!|x : \mathcal{U}\,|\sigma\tau_1|, f : \mathcal{U}\,(|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|), \mathcal{U}\,|\sigma\Gamma|)\!)$ which holds because

- $(j'', \delta) \in \mathcal{G}(\!|\mathcal{U}\,|\sigma\Gamma|)\!)$ by downward closure (Lemma 33) on $(m, \delta) \in \mathcal{G}(\!|\mathcal{U}\,|\sigma\Gamma|)\!)$ using $j'' \leqslant m$.
- $(j'', w) \in (\!|\mathcal{U}\,|\sigma\tau_1|)\!)_v$, from the assumption above
- $(j'', \text{fix } f(x).\delta \text{ee}) \in (\!|\mathcal{U}\,(|\sigma\tau_1| \xrightarrow{\mathbb{FS}(\infty)} |\sigma\tau_2|)|\!)_v$, obtained by downward closure (Lemma 33) on (2) using $j'' \leqslant m''$

This completes the proof of this case.

□

Proof of Statement (4). There is only one case.

Case: $\dfrac{\overline{\Delta; \Phi; \Gamma \vdash w_i \gg \tau_i} \qquad \Delta; \Phi; \overline{x_i : \tau_i}, \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t}{\Delta; \Phi; \Gamma \vdash \ulcorner e \urcorner[\overline{w_i/x_i}] \gg \tau \mid t}$ **bi-expr**

Assume that $\models \sigma\Phi$ and $(m, \gamma) \in \mathcal{G}[\![\sigma\Gamma]\!]$.
TS: $(m, \delta^{\ulcorner} e^{\urcorner}[\overline{\delta(w_i)/x_i}]) \in (\!|\sigma\tau|\!)^{\sigma t}_{\varepsilon}$ $(*)$.
By IH 1 on premise $\Delta; \Phi; \Gamma \vdash w_i \gg \tau_i$, we get

$$(m, \delta(w_i)) \in (\!|\sigma\tau_i|\!)_{\nu} \tag{1}$$

By (Fundamental Theorem) Theorem 46 (first clause) on the second premise using

- $\sigma \in \mathcal{D}[\![\Delta]\!]$,
- $(m, \delta[x_i \mapsto \delta(w_i)]) \in \mathcal{G}(\!|x_i : \sigma\tau_i, \sigma\Gamma|\!)$ (by eq. (1) and $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!)$)
- $\models \sigma\Phi$,

we get $(m, \delta[x_i \mapsto \delta(w_i)]^{\ulcorner} e^{\urcorner}) \in (\!|\sigma\tau|\!)^{\sigma t}_{\varepsilon}$ which is the same as $(*)$.

□

Proof of Statement (5). There is only one case. We only show the left projection; the right one is similar.

Case: $\dfrac{\overline{\Delta; \Phi; \Gamma \vdash w_i \gg \tau_i} \qquad \Delta; \Phi; \overline{x_i : \tau_i}, \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t}{\Delta; \Phi; \Gamma \vdash \ulcorner e \urcorner[\overline{w_i/x_i}] \gg \tau \mid t}$ **bi-expr**

Assume that $\models \sigma\Phi$ and $(m, \gamma) \in \mathcal{G}[\![|\sigma\Gamma|]\!]$.
TS: $(m, L(\delta^{\ulcorner} e^{\urcorner}[\overline{\delta(w_i)/x_i}])) \in [\![|\sigma\tau|]\!]^{\infty}_{\varepsilon}$ $(*)$.
By IH 2 on first premise $\Delta; \Phi; \Gamma \vdash w_i \gg \tau_i$, we get

$$(m, L(\delta(w_i))) \in [\![|\sigma\tau_i|]\!]_{\nu} \tag{1}$$

By (Fundamental Theorem) Theorem 46 (third clause) on the second premise using

- $\sigma \in \mathcal{D}[\![\Delta]\!]$,

- $(m, \delta[x_i \mapsto L(\delta(w_i))]) \in \mathcal{G}(\!|x_i : |\sigma\tau_i|, |\sigma\Gamma|\!|)$ (by eq. (1) and $(m, \delta) \in \mathcal{G}(\!|\sigma\Gamma|\!|)$
- $\models \sigma\Phi$,

we get $(m, \delta[x_i \mapsto L(\delta(w_i))]L(\ulcorner e \urcorner)) \in (\!|\sigma\tau|\!|)^{\infty}_{\varepsilon}$ which is the same as $(*)$.

□

Proof of Statement (6). There is only one case.

Case: $\dfrac{\overline{\Delta; \Phi; \Gamma \vdash w_i \gg \tau_i} \qquad \Delta; \Phi; \overline{x_i : \tau_i}, \Gamma \vdash_{\mathbb{CP}} e : \tau \mid t}{\Delta; \Phi; \Gamma \vdash \ulcorner e \urcorner[\overline{w_i/x_i}] \gg \tau \mid t}$ **bi-expr**

Assume that $\models \sigma\Phi$ and $(m, \gamma) \in \mathcal{G}(\!|U\,|\sigma\Gamma|\!|)$.
TS: $(m, \delta\ulcorner e \urcorner[\overline{\delta(w_i)/x_i}]) \in (\!|U\,|\sigma\tau|\!|)^{\infty}_{\varepsilon}$ $(*)$.
By IH 3 on the first premise $\Delta; \Phi; \Gamma \vdash w_i \gg \tau_i$, we get

$$(m, \delta(w_i)) \in (\!|U\,|\sigma\tau|_i|\!|)_v \tag{1}$$

By (Fundamental Theorem) Theorem 46 (fifth clause) on the second premise using

- $\sigma \in \mathcal{D}[\![\Delta]\!]$,
- $(m, \delta[x_i \mapsto \delta(w_i)]) \in \mathcal{G}(\!|x_i : U\,|\sigma\tau|_i, U\,|\sigma\Gamma|\!|)$ (by eq. (1) and $(m, \delta) \in \mathcal{G}(\!|U\,|\sigma|\Gamma|\!|)$
- $\models \sigma\Phi$,

we get $(m, \delta[x_i \mapsto \delta(w_i)]\ulcorner e \urcorner) \in (\!|U\,|\sigma\tau|\!|)^{\sigma t}_{\varepsilon}$ which is the same as $(*)$.

□

□

C

APPENDIX FOR BIRELCOST

In this chapter, we first describe the necessary definitions, lemmas and theorems for proving the soundness and completeness of the BiRelCost's unary and binary (relational) typing with respect to the algorithmic system.

C.1 BIRELCOST LEMMAS

Lemma 48 (Embedding of Binary Subtyping). *If* $\Delta;\Phi \models \tau \sqsubseteq \tau'$ *then* $\exists e \in$ *RelCost Core such that* $\Delta;\Phi;\cdot \vdash e \ominus e \lesssim 0 :^{\mathbf{c}} \tau \xrightarrow{diff(0)} \tau'$.

Proof. Proof is by induction on the subtyping derivation. We denote the witness e of type $\tau \xrightarrow{\mathrm{diff}(0)} \tau'$ as $\mathtt{coerce}_{\tau,\tau'}$ for clarity.

Case: $\dfrac{\Delta;\Phi_a \models \tau_1' \sqsubseteq \tau_1 \quad (\star) \qquad \Delta;\Phi_a \models \tau_2 \sqsubseteq \tau_2' \quad (\diamond) \qquad \Delta;\Phi_a \models t \leqslant t'}{\Delta;\Phi_a \models \tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2 \sqsubseteq \tau_1' \xrightarrow{\mathbf{diff}(t')} \tau_2'}\ \mathbf{r\text{-}\rightarrow}$

diff

By IH on $(\star)$, we get $\exists \mathtt{coerce}_{\tau_1',\tau_1}$.

$\Delta;\Phi;\cdot \vdash \mathtt{coerce}_{\tau_1',\tau_1} \ominus \mathtt{coerce}_{\tau_1',\tau_1} \lesssim 0 :^{\mathbf{c}} \tau_1' \xrightarrow{\mathrm{diff}(0)} \tau_1$

By IH on $(\diamond)$, we get $\exists \mathtt{coerce}_{\tau_2,\tau_2'}$.

$\Delta;\Phi;\cdot \vdash \mathtt{coerce}_{\tau_2,\tau_2'} \ominus \mathtt{coerce}_{\tau_2,\tau_2'} \lesssim 0 :^{\mathbf{c}} \tau_2 \xrightarrow{\mathrm{diff}(0)} \tau_2'$

Then, using these two statements and $\Delta;\Phi \models t \leqslant t'$ with binary subeffecting rule (rule **c-r-≡** in Figure 40), we can construct the following derivation where

$e = \lambda x.\lambda y.\mathtt{coerce}_{\tau_2,\tau_2'}\ (x\ (\mathtt{coerce}_{\tau_1',\tau_1}\ y))$

$\Delta;\Phi;\cdot \vdash e \ominus e \lesssim 0 :^{\mathbf{c}} (\tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2) \xrightarrow{\mathrm{diff}(0)} \tau_1' \xrightarrow{\mathrm{diff}(t')} \tau_2'$

Case: $\dfrac{}{\Delta;\Phi \models \mathbf{unit}_r \sqsubseteq \Box\,\mathbf{unit}_r}$ **r-unit**

Then, we can immediately construct the derivation using the rule **c-nochange** in Figure 40.

$$\frac{}{\Delta;\Phi;\cdot \vdash \lambda x.\mathtt{NC}\ () \ominus \lambda x.\mathtt{NC}\ () \lesssim 0 :^{\mathbf{c}} \mathrm{unit}_r \xrightarrow{\mathrm{diff}(0)} \Box\,\mathrm{unit}_r}$$

Case: $\dfrac{}{\Delta;\Phi \models \mathbf{int}_r \sqsubseteq \Box\,\mathbf{int}_r}$ **r-int-$\Box$**

Then, we can construct the derivation using the primitive function $\mathrm{box}_{\mathrm{int}} : \mathrm{int}_r \xrightarrow{\mathrm{diff}(0)} \Box\,\mathrm{int}_r$

$$\frac{}{\Delta;\Phi;\cdot \vdash \lambda x.\mathrm{box}_{\mathrm{int}} x \ominus \lambda x.\mathrm{box}_{\mathrm{int}} x \lesssim 0 :^{\mathbf{c}} \mathrm{int}_r \xrightarrow{\mathrm{diff}(0)} \Box\,\mathrm{int}_r}$$

Case: $\dfrac{}{\Delta;\Phi \models \Box\,\mathcal{U}\,(\mathbf{int},\mathbf{int}) \sqsubseteq \mathbf{int}_r}$ **r-$\Box$U-int**

Then, we can construct the derivation using the primitive function $\mathrm{box}_{\mathcal{U}} : \Box\,\mathcal{U}\,(\mathrm{int},\mathrm{int}) \xrightarrow{\mathrm{diff}(0)} \mathrm{int}_r$

$$\frac{}{\Delta;\Phi;\cdot \vdash \lambda x.\mathrm{box}_{\mathcal{U}} x \ominus \lambda x.\mathrm{box}_{\mathcal{U}} x \lesssim 0 :^{\mathbf{c}} \Box\,\mathcal{U}\,(\mathrm{int},\mathrm{int}) \xrightarrow{\mathrm{diff}(0)} \mathrm{int}_r}$$

Case: $\dfrac{}{\Delta;\Phi \models \Box\,\tau \sqsubseteq \tau}$ **T**

Then, we can immediately construct the derivation using the rule **c-der** in Figure 40.

$$\frac{}{\Delta;\Phi;\cdot \vdash \lambda x.\mathtt{der}\ x \ominus \lambda x.\mathtt{der}\ x \lesssim 0 :^{\mathbf{c}} \Box\,\tau \xrightarrow{\mathrm{diff}(0)} \tau}$$

Case: $\dfrac{}{\Delta;\Phi \models \Box\tau \sqsubseteq \Box\Box\tau}$ **D**

Then, we can immediately construct the derivation using the rule **c-nochange** in Figure 40.

$$\frac{}{\Delta;\Phi;\cdot \vdash \lambda x.\mathtt{NC}\ x \ominus \lambda x.\mathtt{NC}\ x \lesssim 0 :^{\mathbf{c}} \Box\tau \xrightarrow{\mathrm{diff}(0)} \Box\Box\tau}$$

Case: $\dfrac{\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_2 \ (\star)}{\Delta;\Phi_a \models \Box\tau_1 \sqsubseteq \Box\tau_2}$ **B-$\Box$**

By IH on $(\star)$, $\exists \mathtt{coerce}_{\tau_1,\tau_2}$.
$\Delta;\Phi;\cdot \vdash \mathtt{coerce}_{\tau_1,\tau_2} \ominus \mathtt{coerce}_{\tau_1,\tau_2} \lesssim 0 :^{\mathbf{c}} \tau_1 \xrightarrow{\mathrm{diff}(0)} \tau_2$

Then, using $(\star)$ and the rules **c-der** and **c-nochange** in Figure 40, we can construct the derivation

$$\frac{}{\Delta;\Phi;\cdot \vdash e \ominus e \lesssim 0 :^{\mathbf{c}} \Box\tau_1 \xrightarrow{\mathrm{diff}(0)} \Box\tau_2}$$

where $e = \lambda x.\mathtt{NC}\ (\mathtt{coerce}_{\tau_1,\tau_2}\ (\mathtt{der}\ x))$

Case: $\dfrac{}{\Delta;\Phi \models \tau \sqsubseteq \mathsf{U}\ (|\tau|_1, |\tau|_2)}$ **W**

Then, we can immediately construct the derivation using the rule **c-switch** in Figure 40.

$$\frac{}{\Delta;\Phi;\cdot \vdash \lambda x.\mathtt{switch}\ x \ominus \lambda x.\mathtt{switch}\ x \lesssim 0 :^{\mathbf{c}} \tau \xrightarrow{\mathrm{diff}(0)} \mathsf{U}\ (|\tau|_1, |\tau|_2)}$$

Case: $\dfrac{}{\Delta;\Phi \models \tau \sqsubseteq \tau}$ **r-refl**

Then, we can immediately construct the derivation

$$\frac{}{\Delta;\Phi;\cdot \vdash \lambda x.x \ominus \lambda x.x \lesssim 0 :^{\mathbf{c}} \tau \xrightarrow{\mathrm{diff}(0)} \tau}$$

Case: $\dfrac{\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_2 \quad (\star) \qquad \Delta;\Phi_a \models \tau_2 \sqsubseteq \tau_3 \quad (\diamond)}{\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_3}$ **r-trans**

By IH on $(\star)$, $\exists \mathtt{coerce}_{\tau_1,\tau_2}$.
$i :: S, \Delta; \Phi; \cdot \vdash \mathtt{coerce}_{\tau_1,\tau_2} \ominus \mathtt{coerce}_{\tau_1,\tau_2} \lesssim 0 :^{\mathbf{c}} \tau_1 \xrightarrow{\mathbf{diff}(0)} \tau_2$
By IH on $(\diamond)$, $\exists \mathtt{coerce}_{\tau_2,\tau_3}$.
$i :: S, \Delta; \Phi; \cdot \vdash \mathtt{coerce}_{\tau_2,\tau_3} \ominus \mathtt{coerce}_{\tau_2,\tau_3} \lesssim 0 :^{\mathbf{c}} \tau_2 \xrightarrow{\mathbf{diff}(0)} \tau_3$

Then, using $(\star)$ and $(\diamond)$, we can construct the derivation simply by function composition

$$\frac{}{\Delta;\Phi;\cdot \vdash e \ominus e \lesssim 0 :^{\mathbf{c}} \tau_1 \xrightarrow{\mathbf{diff}(0)} \tau_3}$$

where $e = \lambda x.\mathtt{coerce}_{\tau_2,\tau_3}\ (\mathtt{coerce}_{\tau_1,\tau_2}\ x)$

Case: $\dfrac{}{\Delta;\Phi \models \Box(\tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2) \sqsubseteq \Box\tau_1 \xrightarrow{\mathbf{diff}(0)} \Box\tau_2}$ **r-$\to$ $\Box_{\mathbf{diff}}$**

Then, we can immediately construct the derivation where
$e = \lambda x.\lambda y.\mathtt{NC}\ (\mathtt{der}\ x)\ (\mathtt{der}\ y))$
$\Delta;\Phi;\cdot \vdash e \ominus e \lesssim 0 :^{\mathbf{c}} \Box(\tau_1 \xrightarrow{\mathbf{diff}(k)} \tau_2) \xrightarrow{\mathbf{diff}(0)} \Box\tau_1 \xrightarrow{\mathbf{diff}(0)} \Box\tau_2$

Case: $\dfrac{}{\Delta;\Phi \models U(A_1 \xrightarrow{\mathbf{exec}(k,t)} A_2, A_1') \xrightarrow{\mathbf{exec}(k',t')} A_2') \sqsubseteq U(A_1, A_1') \xrightarrow{\mathbf{diff}(t-k')} U(A_2, A_2')}$ **r-$\to$ execdiff**

Then, we can immediately construct the following derivation where $e = \lambda x.\lambda y.\mathtt{switch}\ (x\ y)$ using the **c-switch** and **c-app** rules.
$\Delta;\Phi;\cdot \vdash e \ominus e \lesssim 0 :^{\mathbf{c}} \tau$
where $\tau = (U(A_1 \xrightarrow{\mathrm{exec}(k,t)} A_2, A_1') \xrightarrow{\mathrm{exec}(k',t')} A_2')) \xrightarrow{\mathbf{diff}(0)} U(A_1, A_1') \xrightarrow{\mathbf{diff}(t-k')} U(A_2, A_2')$

Case: $\dfrac{i :: S, \Delta; \Phi_a \models \tau \sqsubseteq \tau' \quad (\star) \qquad i :: S, \Delta; \Phi_a \models t \leqslant t' \qquad i \notin FV(\Phi_a)}{\Delta;\Phi_a \models \forall i \overset{\mathbf{diff}(t)}{::} S.\tau \sqsubseteq \forall i \overset{\mathbf{diff}(t')}{::} S.\tau'}$ **r-$\forall_{\mathbf{diff}}$**

By IH on $(\star)$, $\exists \mathtt{coerce}_{\tau,\tau'}$. $i :: S, \Delta; \Phi; \cdot \vdash \mathtt{coerce}_{\tau,\tau'} \ominus \mathtt{coerce}_{\tau,\tau'} \lesssim 0 :^{\mathbf{c}} \tau \xrightarrow{\mathbf{diff}(0)} \tau'$

Then, using this, the second premise and the **c-r-iLam** and **c-r-iApp** rules in RelCost Core, we can construct the following derivation:

$$\Delta;\Phi;\cdot \vdash \lambda x.\Lambda i.\mathtt{coerce}_{\tau,\tau'}\ (x\ [i]) \ominus \lambda x.\Lambda i.\mathtt{coerce}_{\tau,\tau'}\ (x\ [i]) \lesssim 0 :^{\mathbf{c}} \tau_2$$
$$\text{where } \tau_r = (\forall i \overset{\mathbf{diff}(t)}{::} S.\tau) \xrightarrow{\mathbf{diff}(0)} \forall i \overset{\mathbf{diff}(t')}{::} S.\tau'$$

Case: $\dfrac{}{\Delta;\Phi \models \Box(\forall i \overset{\mathbf{diff}(t)}{::} S.\tau) \sqsubseteq \forall i \overset{\mathbf{diff}(0)}{::} S.\Box\tau}$ **r-**$\forall\,\Box$

Then, we can immediately construct the following derivation using the **c-der**, **c-nochange**, **c-r-iLam** and **c-r-iApp** rules in Figures 40 and 42.

$$\Delta;\Phi;\cdot \vdash \lambda x.\Lambda i.\mathtt{NC}\ ((\mathtt{der}\ x)\ [i]) \ominus \lambda x.\Lambda i.\mathtt{NC}\ ((\mathtt{der}\ x)\ [i]) \lesssim 0 :^{\mathbf{c}} \tau_r$$
$$\text{where } \tau_r = \Box(\forall i \overset{\mathbf{diff}(t)}{::} S.\tau) \xrightarrow{\mathbf{diff}(0)} \forall i \overset{\mathbf{diff}(t')}{::} S.\Box\tau$$

Case: $\dfrac{}{\Delta;\Phi \models U(\forall i \overset{\mathbf{exec}(k,t)}{::} S.A, \forall i \overset{\mathbf{exec}(k',t')}{::} S.A') \sqsubseteq \forall i \overset{\mathbf{diff}(t-k')}{::} S.U(A,A')}$ **r-**$\forall$**U**

Then, we can immediately construct the following derivation where $e = \lambda x.\Lambda i.\mathtt{switch}\ (x\ [i])$ using the **c-switch** and **c-iApp** rules in Figures 40 and 42.

$$\Delta;\Phi;\cdot \vdash e \ominus e \lesssim 0 :^{\mathbf{c}} (U(\forall i \overset{\mathbf{exec}(k,t)}{::} S.A, \forall i \overset{\mathbf{exec}(k',t')}{::} S.A')) \xrightarrow{\mathbf{diff}(0)} \forall i \overset{\mathbf{diff}(t-k')}{::} S.U(A,A')$$

Case: $\dfrac{\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_1'\ \ (\star) \qquad \Delta;\Phi_a \models \tau_2 \sqsubseteq \tau_2'\ \ (\diamond)}{\Delta;\Phi_a \models \tau_1 \times \tau_2 \sqsubseteq \tau_1' \times \tau_2'}$ **r-**$\times$

By IH on $(\star)$, $\exists \mathtt{coerce}_{\tau_1,\tau_1'}\ .\ \Delta;\Phi;\cdot \vdash \mathtt{coerce}_{\tau_1,\tau_1'} \ominus \mathtt{coerce}_{\tau_1,\tau_1'} \lesssim 0 :^{\mathbf{c}} \tau_1 \xrightarrow{\mathbf{diff}(0)} \tau_1'$

By IH on $(\diamond)$, $\exists \mathtt{coerce}_{\tau_2,\tau_2'}\ .\ \Delta;\Phi;\cdot \vdash \mathtt{coerce}_{\tau_2,\tau_2'} \ominus \mathtt{coerce}_{\tau_2,\tau_2'} \lesssim 0 :^{\mathbf{c}} \tau_2 \xrightarrow{\mathbf{diff}(0)} \tau_2'$

Then, using these two statements and the rules **c-prod** and **c-proj**$_i$ in Figure 41, we can show the following derivation where

$e = \lambda x.\langle \mathtt{coerce}_{\tau_1,\tau_1'}\ (\pi_1 x), \mathtt{coerce}_{\tau_2,\tau_2'}\ (\pi_2 x)\rangle$

$$\Delta;\Phi;\cdot \vdash e \ominus e \lesssim 0 :^{\mathbf{c}} (\tau_1 \times \tau_2) \xrightarrow{\mathbf{diff}(0)} \tau_1' \times \tau_2'$$

Case: $\dfrac{}{\Delta;\Phi \models \Box\tau_1 \times \Box\tau_2 \equiv \Box(\tau_1 \times \tau_2)}$ **r-×□**

We show the direction from right-to-left using the rules **c-der**, **c-nochange**, **c-r-proj$_i$**, **c-r-let** and **c-r-prod** in Figures 40 to 42 where the expression

$e = \lambda x.\text{let } a = \pi_1 x \text{ in}$

$\text{let } b = \pi_2 x \text{ in } \mathtt{NC}\ (\langle \mathtt{der}\ a, \mathtt{der}\ b\rangle).$

$\Delta;\Phi;\cdot \vdash e \ominus e \lesssim 0 :^{\mathbf{c}} \Box\tau_1 \times \Box\tau_2 \xrightarrow{\text{diff}(0)} \Box(\tau_1 \times \tau_2)$

Case: $\dfrac{}{\Delta;\Phi \models \mathrm{U}(A_1 \times A_2, A_1' \times A_2') \sqsubseteq \mathrm{U}(A_1, A_1') \times \mathrm{U}(A_2, A_2')}$ **r-×U**

Then, we can immediately construct the following derivation where $e = \lambda x.(\langle \mathtt{switch}\ \pi_1 x, \mathtt{switch}\ \pi_2 x\rangle)$ using the **c-switch**, **c-r-prod** and **c-r-proj$_i$** rules in Figures 40 and 41.

$\Delta;\Phi;\cdot \vdash e \ominus e \lesssim 0 :^{\mathbf{c}} (\mathrm{U}(A_1 \times A_2, A_1' \times A_2')) \xrightarrow{\text{diff}(0)} \mathrm{U}(A_1, A_1') \times \mathrm{U}(A_2, A_2')$

Case: $\dfrac{}{\Delta;\Phi \models \Box\tau_1 + \Box\tau_2 \sqsubseteq \Box(\tau_1 + \tau_2)}$ **r-+□**

We can construct the following derivation by using the rules **c-der**, **c-nochange**, **c-r-case**, **c-r-inl** and **c-r-inr** in Figure 40 where the expression

$e = \lambda x.\ \text{case}\ (x, a.\mathtt{NC}\ (\text{inl}\ \mathtt{der}\ a), b.\mathtt{NC}\ (\text{inr}\ \mathtt{der}\ b)).$

$\Delta;\Phi;\cdot \vdash e \ominus e \lesssim 0 :^{\mathbf{c}} \Box\tau_1 + \Box\tau_2 \xrightarrow{\text{diff}(0)} \Box(\tau_1 + \tau_2)$

Case: $\dfrac{\Delta;\Phi_a \models n \doteq n' \ \ (\star) \qquad \Delta;\Phi_a \models \alpha \leqslant \alpha' \ \ (\diamond) \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau' \ \ (\dagger)}{\Delta;\Phi_a \models \mathbf{list}[n]^{\alpha}\tau \sqsubseteq \mathbf{list}[n']^{\alpha'}\tau'}$ **r-l1**

By IH on $(\dagger)$, $\exists \mathtt{coerce}_{\tau,\tau'}$.

$\Delta;\Phi;\cdot \vdash \mathtt{coerce}_{\tau,\tau'} \ominus \mathtt{coerce}_{\tau,\tau'} \lesssim 0 :^{\mathbf{c}} \tau \xrightarrow{\text{diff}(0)} \tau'$

We first construct the more generic term for type:

$$\begin{aligned} &\text{unit}_\tau \xrightarrow{\text{diff}(0)} \forall n{::}\mathbb{N}.\,\forall n'{::}\mathbb{N}.\,\forall \alpha{::}\mathbb{N}.\,\forall \alpha'{::}\mathbb{N}. \\ &((n = n' \wedge \alpha \leqslant \alpha')\ \&\ \text{list}[n]^{\alpha}\tau) \xrightarrow{\text{diff}(0)} \text{list}[n']^{\alpha'}\tau' \end{aligned} \tag{1}$$

and then instantiate the term for eq. (1) later.
It can be shown that such a derivation can be constructed for expression
$e' = \text{fix fList}(\_).\Lambda n.\Lambda n'.\Lambda\alpha.\Lambda\alpha'.\lambda x.\text{clet } x \text{ as } e \text{ in}$
$\text{case } e \text{ of}$
$\text{nil} \ \to \text{nil}$
$\mid h ::_N tl \ \to \text{let } r = \text{fList } () \ [n-1] \ [n'-1] \ [\alpha] \ [\alpha'] \ tl \text{ in}$
$\text{cons}_{NC}(\texttt{NC}\ (\texttt{coerce}_{\tau,\tau'}\ \ \texttt{der}\ h), r)$
$\mid h ::_C tl \ \to \text{let } r = \text{fList } () \ [n-1] \ [n'-1] \ [\alpha-1] \ [\alpha'-1] \ tl \text{ in}$
$\text{cons}_C(\texttt{coerce}_{\tau,\tau'}\ \ h, r)$

Then, we can instantiate fList using $(\star)$ and $(\diamond)$ as follows where
$e'' = \lambda x.\text{fList } () \ [n][n'][\alpha][\alpha'] \ x$

$$\Delta; \Phi; \cdot \vdash e'' \ominus e'' \lesssim 0 :^{\mathbf{c}} \text{list}[n]^{\alpha}\,\tau \xrightarrow{\text{diff}(0)} \text{list}[n']^{\alpha'}\,\tau'$$

Case: $\dfrac{}{\Delta; \Phi \models \mathbf{list}[n]^{\alpha}\,\Box\tau \sqsubseteq \Box\,(\mathbf{list}[n]^{\alpha}\,\tau)}$ **r-l$\Box$**
We first construct the more generic term for type

$$\text{unit}_r \xrightarrow{\text{diff}(0)} \forall n{::}\mathbb{N}.\,\forall\alpha{::}\mathbb{N}.\,\text{list}[n]^{\alpha}\,\Box\tau \xrightarrow{\text{diff}(0)} \Box\,(\text{list}[n]^{\alpha}\,\tau) \qquad (1)$$

and then instantiate the term for eq. (1) later. It can be shown that such a derivation can be constructed for expression
$e' = \text{fix fList}(\_).\Lambda n.\Lambda\alpha.\lambda x.$
$\text{case } e \text{ of}$
$\text{nil} \ \to \texttt{NC}\ (\text{nil}\ )$
$\mid h ::_N tl \ \to \text{let } r = \text{fList } () \ [n-1] \ [\alpha] \ tl \text{ in } \texttt{NC}\ (\text{cons}_{NC}(\texttt{der}\ h, \texttt{der}\ r))$
$\mid h ::_C tl \ \to \text{let } r = \text{fList } () \ [n-1] \ [\alpha-1] \ tl \text{ in } \texttt{NC}\ (\text{cons}_C(\texttt{der}\ h, \texttt{der}\ r))$

Then, we can instantiate fList with a concrete n and α as follows where $e'' = \lambda x.\text{fList } () \ [n][\alpha] \ x$

$$\Delta; \Phi; \cdot \vdash e'' \ominus e'' \lesssim 0 :^{\mathbf{c}} \text{list}[n]^{\alpha}\,\Box\tau \xrightarrow{\text{diff}(0)} \Box\,(\text{list}[n]^{\alpha}\,\tau)$$

Case: $\dfrac{\Delta;\Phi \models \alpha \doteq 0}{\Delta;\Phi \models \mathbf{list}[n]^{\alpha}\,\tau \sqsubseteq \mathbf{list}[n]^{\alpha}\,\Box\,\tau}$ **r-l2**

We first construct the more generic term for type

$$\mathrm{unit}_r \xrightarrow{\mathrm{diff}(0)} \forall n{::}\mathbb{N}.\,\forall \alpha{::}\mathbb{N}.\,(\alpha = 0\ \&\ \mathrm{list}[n]^{\alpha}\,\tau) \xrightarrow{\mathrm{diff}(0)} \mathrm{list}[n]^{\alpha}\,\Box\,\tau \quad (1)$$

and then instantiate the term for eq. (1) later. It can be shown that such a derivation can be constructed for expression

$e' = \mathrm{fix}\ \mathrm{fList}(\_).\Lambda n.\Lambda\alpha.\lambda x.\mathrm{clet}\ x\ \mathrm{as}\ e\ \mathrm{in}$

$\ \mathrm{case}\ e\ \mathrm{of}$

$\mathrm{nil}\ \to \mathrm{nil}$

$\mid h ::_N tl \to \mathrm{let}\ r = \mathrm{fList}\ ()\ [n-1][\alpha]\ tl\ \mathrm{in}\ \mathrm{cons}_{NC}(\mathtt{NC}\ h, r)$

$\mid h ::_C tl \to \mathtt{contra}$

Then, we can instantiate fList with a concrete n and α (note the premise $\alpha = 0$) as follows where $e'' = \lambda x.\mathrm{fList}\ ()\ [n][\alpha]\ x$

$\Delta;\Phi;\cdot \vdash e'' \ominus e'' \lesssim 0 :^{\mathbf{c}} \mathrm{list}[n]^{0}\,\tau \xrightarrow{\mathrm{diff}(0)} \mathrm{list}[n]^{0}\,\Box\,\tau$

Case: $\dfrac{i :: S, \Delta;\Phi_a \models \tau \sqsubseteq \tau' \quad (\star) \qquad i \notin FV(\Phi_a)}{\Delta;\Phi_a \models \exists i{::}S.\,\tau \sqsubseteq \exists i{::}S.\,\tau'}$ **r-∃**

By IH on $(\star)$, $\exists \mathtt{coerce}_{\tau,\tau'}$.

$i :: S, \Delta;\Phi;\cdot \vdash \mathtt{coerce}_{\tau,\tau'} \ominus \mathtt{coerce}_{\tau,\tau'} \lesssim 0 :^{\mathbf{c}} \tau \xrightarrow{\mathrm{diff}(0)} \tau'$

Then, using this and the **c-r-pack** and **c-r-unpack** rules in RelCost Core in Figure 42, we can construct the following derivation where

$e = \lambda x.\mathrm{unpack}\ x\ \mathrm{as}\ (y, i)\ \mathrm{in}\ \mathrm{pack}\ (\mathtt{coerce}_{\tau,\tau'}\ \ y)\ \mathrm{with}\ i$

$\Delta;\Phi;\cdot \vdash e \ominus e \lesssim 0 :^{\mathbf{c}} (\exists i{::}S.\,\tau) \xrightarrow{\mathrm{diff}(0)} \exists i{::}S.\,\tau'$

Case: $\dfrac{}{\Delta;\Phi \models \exists i{::}S.\,\Box\,\tau \sqsubseteq \Box\,(\exists i{::}S.\,\tau)}$ **r-∃□**

Then, we can immediately construct the following derivation using the **c-der**, **c-nochange**, **c-r-pack** and **c-r-unpack** rules in in Figures 40

and 42 where $e = \lambda x.\text{unpack } x \text{ as } (y, i) \text{ in } \texttt{NC}\ (\text{pack } \texttt{der}\ y \text{ with } i)$.

$$\Delta; \Phi; \cdot \vdash e \ominus e \lesssim 0 :^{c} (\exists i{::}S.\, \Box \tau) \xrightarrow{\text{diff}(0)} \Box\, (\exists i{::}S.\, \tau)$$

Case: $$\frac{\Delta; \Phi_a \wedge C' \models C \quad (\star) \qquad \Delta; \Phi_a \models \tau \sqsubseteq \tau' \quad (\diamond)}{\Delta; \Phi_a \models C \supset \tau \sqsubseteq C' \supset \tau'}\ \textbf{r-c-impl}$$

By IH on $(\diamond)$, $\exists \texttt{coerce}_{\tau,\tau'}$.

$$\Delta; \Phi; \cdot \vdash \texttt{coerce}_{\tau,\tau'} \ominus \texttt{coerce}_{\tau,\tau'} \lesssim 0 :^{c} \tau \xrightarrow{\text{diff}(0)} \tau'$$

Then, using this and the premise $(\star)$ along with the **c-r-c-implI** and **c-r-c-implE** rules in Figure 42, we can construct the following derivation where

$e = \lambda x.\texttt{coerce}_{\tau,\tau'}\ (\text{celim}_{\supset}\ x)$

$$\Delta; \Phi; \cdot \vdash e \ominus e \lesssim 0 :^{c} (C \supset \tau) \xrightarrow{\text{diff}(0)} C' \supset \tau'$$

Case: $$\frac{}{\Delta; \Phi \models \Box\, (C \supset \tau) \sqsubseteq C \supset \Box \tau}\ \textbf{r-c-impl-}\Box$$

Then, we can immediately construct the following derivation using the **c-der**, **c-nochange** and **c-r-c-implE** rules in RelCost Core where $e = \lambda x.\texttt{NC}\ (\text{celim}_{\supset}\ \texttt{der}\ x)$.

$$\Delta; \Phi; \cdot \vdash e \ominus e \lesssim 0 :^{c} \Box\, (C \supset \tau) \xrightarrow{\text{diff}(0)} (C \supset \Box \tau)$$

Case: $$\frac{\Delta; \Phi_a \wedge C \models C' \quad (\star) \qquad \Delta; \Phi_a \models \tau \sqsubseteq \tau' \quad (\diamond)}{\Delta; \Phi_a \models C \,\&\, \tau \sqsubseteq C' \,\&\, \tau'}\ \textbf{r-c-and}$$

By IH on $(\diamond)$, $\exists \texttt{coerce}_{\tau,\tau'}$.

$$\Delta; \Phi; \cdot \vdash \texttt{coerce}_{\tau,\tau'} \ominus \texttt{coerce}_{\tau,\tau'} \lesssim 0 :^{c} \tau \xrightarrow{\text{diff}(0)} \tau'$$

Then, using this and the premise $(\star)$ along with the **c-r-c-prodI** and **c-r-c-prodE** rules in Figure 41, we can construct the following deriva-

tion where
$e = \lambda x.\text{clet } x \text{ as } y \text{ in } \texttt{coerce}_{\tau,\tau'}\ y$

$$\Delta; \Phi; \cdot \vdash e \ominus e \lesssim 0 :^{\mathfrak{c}} (C \,\&\, \tau) \xrightarrow{\text{diff}(0)} C' \,\&\, \tau'$$

Case: $\dfrac{}{\Delta; \Phi \models C \,\&\, \Box\tau \sqsubseteq \Box(C \,\&\, \tau)}$ **r-c-and-□**

Then, we can immediately construct the following derivation using the **c-der**, **c-nochange**, **c-r-c-prodI** and **c-r-c-prodE** rules in Figures 40 to 42 where $e = \lambda x.\text{clet } x \text{ as } y \text{ in } \texttt{NC}\ (\texttt{der}\ y)$.

$$\Delta; \Phi; \cdot \vdash e \ominus e \lesssim 0 :^{\mathfrak{c}} (C \,\&\, \Box\tau) \xrightarrow{\text{diff}(0)} \Box(C \,\&\, \tau)$$

□

Lemma 49 (Reflexivity of Algorithmic Binary Type Equivalence). *$\Delta; \psi_a; \Phi_a \models \tau \equiv \tau \Rightarrow \Phi$ and $\Delta; \psi_a; \Phi_a \models \Phi$.*

Proof. By induction on the binary type. □

Lemma 50 (Reflexivity of Unary Algorithmic Subtyping). *$\Delta; \Phi_a \models^{A} A \sqsubseteq A \Rightarrow \Phi$ and $\Delta; \Phi_a \models \Phi$.*

Proof. By induction on the unary type. □

Lemma 51 (Transitivity of Unary Algorithmic Subtyping). *If $\Delta; \Phi_a \models^{A} A_1 \sqsubseteq A_2 \Rightarrow \Phi_1$ and $\Delta; \Phi_a \models^{A} A_2 \sqsubseteq A_3 \Rightarrow \Phi_2$ and $\Delta; \Phi_a \models \Phi_1 \wedge \Phi_2$, then $\Delta; \Phi_a \models^{A} A_1 \sqsubseteq A_3 \Rightarrow \Phi_3$ for some Φ_3 such that $\Delta; \Phi_a \models \Phi_3$.*

Proof. By induction on the first subtyping derivation. □

Theorem 52 (Soundness of the Algorithmic Unary Subtyping). *Assume that*

1. *$\Delta; \psi_a; \Phi_a \models^{A} A' \sqsubseteq A \Rightarrow \Phi$*
2. *$FIV(\Phi_a, A, A') \subseteq \Delta, \psi_a$*

3. $\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$ *is provable s.t* $\Delta \rhd \theta_a : \psi_a$ *is derivable.*

Then $\Delta; \Phi_a[\theta_a] \models^{A} A'[\theta_a] \sqsubseteq A[\theta_a]$.

Proof. By induction on the algorithmic unary subtyping derivation. □

Theorem 53 (Completeness of the Unary Algorithmic Subtyping)**.** *Assume that* $\Delta; \Phi_a \models^{A} A' \sqsubseteq A$. *Then* $\exists\, \Phi$. *such that* $\Delta; \Phi_a \models^{A} A' \sqsubseteq A \Rightarrow \Phi$ *and* $\Delta; \Phi_a \models \Phi$.

Proof. By induction on the unary subtyping derivation. □

Theorem 54 (Soundness of the Algorithmic Binary Type Equality)**.** *Assume that*

1. $\Delta; \psi_a; \Phi_a \models \tau' \equiv \tau \Rightarrow \Phi$
2. $FIV(\Phi_a, \tau, \tau') \subseteq \Delta, \psi_a$
3. $\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$ *is provable s.t* $\Delta \rhd \theta_a : \psi_a$ *is derivable.*

Then $\Delta; \Phi_a[\theta_a] \models \tau'[\theta_a] \equiv \tau[\theta_a]$.

Proof. By induction on the algorithmic binary type equivalence derivation. □

Theorem 55 (Completeness of the Binary Algorithmic Type Equivalence)**.** *Assume that* $\Delta; \Phi_a \models \tau' \equiv \tau$. *Then* $\exists\, \Phi$. *such that* $\Delta; \Phi_a \models \tau' \equiv \tau \Rightarrow \Phi$ *and* $\Delta; \Phi_a \models \Phi$.

Proof. By induction on the binary subtyping derivation. □

Theorem 56 (Soundness of RelCost Core & Type Preservation of Embedding)**.** *The following holds.*

1. *If* $\Delta; \Phi_a; \Omega \vdash^{t}_{k} e \rightsquigarrow e^* : A$, *then* $\Delta; \Phi_a; \Omega \vdash^{t}_{k} e^* :^{c} A$ *and* $\Delta; \Phi_a; \Omega \vdash^{t}_{k} e : A$.
2. *If* $\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \rightsquigarrow e_1^* \ominus e_2^* \lesssim t : \tau$, *then* $\Delta; \Phi_a; \Gamma \vdash e_1^* \ominus e_2^* \lesssim t :^{c} \tau$ *and*
   $\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau$.

Proof. Proof is by simultaneous induction on the embedding derivations. The proof follows from the embedding rules presented in Figures 40 to 42. We show a few representative cases.

Proof of Theorem 56.2:

Case:
$$\frac{\Delta;\Phi_a;|\Gamma|_1 \vdash^{t_1}_{k_1} e_1 \rightsquigarrow e_1^* : A_1 \quad (\star) \qquad \Delta;\Phi_a;|\Gamma|_2 \vdash^{t_2}_{k_2} e_2 \rightsquigarrow e_2^* : A_2 \quad (\diamond) \qquad E = \texttt{switch}\ e_1^* \qquad E' = \texttt{switch}\ e_2^*}{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \rightsquigarrow E \ominus E' \lesssim t_1 - k_2 : U\,(A_1, A_2)}\ \textbf{e-switch}$$

By Theorem 56.1 on $(\star)$, we get $\Delta;\Phi_a;\Omega \vdash^{t_1}_{k_1} e_1^* :^{c} A_1 \quad (\star\star)$.

By Theorem 56.1 on $(\diamond)$, we get $\Delta;\Phi_a;\Omega \vdash^{t_2}_{k_2} e_2^* :^{c} A_2 \quad (\diamond\diamond)$.

Then, we conclude as follows:

$$\frac{\Delta;\Phi_a;|\Gamma|_1 \vdash^{t_1}_{k_1} e_1 :^{c} A_1 \qquad \Delta;\Phi_a;|\Gamma|_2 \vdash^{t_2}_{k_2} e_2 :^{c} A_2}{\Delta;\Phi_a;\Gamma \vdash \texttt{switch}\ e_1 \ominus \texttt{switch}\ e_2 \lesssim t_1 - k_2 :^{c} U\,(A_1, A_2)}\ \textbf{c-switch}$$

Case:
$$\frac{\Delta;\Phi_a;\Gamma \vdash e \ominus e \rightsquigarrow e^* \ominus e^* \lesssim t : \tau \quad (\star) \qquad \forall x_i \in \mathrm{dom}(\Gamma),\ e_i = \texttt{coerce}_{\Gamma(x_i),\Box\Gamma(x_i)} \quad (\diamond) \qquad e' = \textbf{let}\ \overline{y_i = e_i\ x_i}\ \textbf{in}\ \texttt{NC}\ e^*[\overline{y_i/x_i}]}{\Delta;\Phi_a;\Gamma,\Gamma' \vdash e \ominus e \rightsquigarrow e' \ominus e' \lesssim 0 : \Box\tau}\ \textbf{e-nochange}$$

By Theorem 56.2 on $(\star)$, we get $\Delta;\Phi_a;\Gamma \vdash e^* \ominus e^* \lesssim t :^{c} \tau \quad (\star\star)$.

By Lemma 48 using $(\diamond)$, we know that

$\Delta;\Phi_a;\cdot \vdash e_i \ominus e_i \lesssim 0 :^{c} \Gamma(x_i) \xrightarrow{\mathrm{diff}(0)} \Box\Gamma(x_i) \quad (\diamond\diamond)$.

By applying **c-r-var** rule in Figure 40, we get

$\Delta;\Phi_a;\Gamma \vdash x_i \ominus x_i \lesssim t :^{c} \Gamma(x_i) \quad (\spadesuit)$.

By applying **c-r-app** rule in Figure 41 to $(\diamond\diamond)$ and $(\spadesuit)$, we get

$\Delta;\Phi_a;\Gamma \vdash e_i\ x_i \ominus e_i\ x_i \lesssim t :^{c} \Box\Gamma(x_i) \quad (\spadesuit\spadesuit)$.

By substituting in $(\star\star)$, we get

$\Delta;\Phi_a;\overline{y_i : \Box\Gamma(x_i)} \vdash e^*[\overline{y_i/x_i}] \ominus e^*[\overline{y_i/x_i}] \lesssim t :^{c} \tau \quad (\dagger)$.

By applying **c-nochange** rule in Figure 40 to $(\dagger)$, we get

$\Delta;\Phi_a;\overline{y_i : \Box\Gamma(x_i)},\Gamma,\Gamma' \vdash \texttt{NC}\ e^*[\overline{y_i/x_i}] \ominus \texttt{NC}\ e^*[\overline{y_i/x_i}] \lesssim t :^{c} \Box\tau \quad (\dagger\dagger)$.

By applying **c-r-let** rule in Figure 42 to $(\spadesuit\spadesuit)$ and $(\dagger\dagger)$, we can conclude as follows

$\Delta;\Phi_a;\Gamma,\Gamma' \vdash \text{let}\ \overline{y_i = e_i\ x_i}\ \text{in}\ \texttt{NC}\ e^*[\overline{y_i/x_i}] \ominus \text{let}\ \overline{y_i = e_i\ x_i}\ \text{in}\ \texttt{NC}\ e^*[\overline{y_i/x_i}] \lesssim t :^{c} \Box\tau.$

Case:
$$\frac{\begin{array}{c}\Delta;\Phi_a \vdash \tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2\ \mathtt{wf} \\ \Delta;\Phi_a;x:\tau_1, f:\square(\tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2), \Gamma \vdash e \ominus e \rightsquigarrow e^* \ominus e^* \lesssim t:\tau_2 \quad (\star) \\ \forall x_i \in \mathrm{dom}(\Gamma),\ \ e_i = \mathtt{coerce}_{\Gamma(x_i),\square\Gamma(x_i)} \quad (\diamond) \\ e^{**} = \mathbf{let}\ \overline{y_i = e_i\ x_i}\ \mathbf{in}\ \mathbf{fix}_{\mathrm{NC}}\ f(x).e^*[\overline{y_i/x_i}]\end{array}}{\Delta;\Phi_a;\Gamma \vdash \mathbf{fix}\ f(x).e \ominus \mathbf{fix}\ f(x).e \rightsquigarrow e^{**} \ominus e^{**} \lesssim 0 : \square(\tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2)}\ \textbf{e-r-fixNC}$$

By Theorem 56.2 on $(\star)$, we get

$\Delta;\Phi_a;x:\tau_1, f:\square(\tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2), \Gamma \vdash e^* \ominus e^* \lesssim t :^{\mathbf{c}} \tau_2 \quad (\star\star)$.

By Lemma 48 using $(\diamond)$, we know that

$\Delta;\Phi_a;\cdot \vdash e_i \ominus e_i \lesssim 0 :^{\mathbf{c}} \Gamma(x_i) \xrightarrow{\mathrm{diff}(0)} \square\Gamma(x_i) \quad (\diamond\diamond)$.

By applying **c-r-var** rule in Figure 40, we get

$\Delta;\Phi_a;\Gamma \vdash x_i \ominus x_i \lesssim t :^{\mathbf{c}} \Gamma(x_i) \quad (\spadesuit)$.

By applying **c-r-app** rule in Figure 41 to $(\diamond\diamond)$ and $(\spadesuit)$, we get

$\Delta;\Phi_a;\Gamma \vdash e_i\ x_i \ominus e_i\ x_i \lesssim t :^{\mathbf{c}} \square\Gamma(x_i) \quad (\spadesuit\spadesuit)$.

By substituting in $(\star\star)$, we get

$\Delta;\Phi_a;x:\tau_1, f:\square(\tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2), \overline{y_i:\square\Gamma(x_i)} \vdash e^*[\overline{y_i/x_i}] \ominus e^*[\overline{y_i/x_i}] \lesssim t :^{\mathbf{c}} \tau_2 \quad (\dagger)$.

By applying **c-r-fixNC** rule in Figure 40 to $(\dagger)$, we get

$\Delta;\Phi_a;\overline{y_i:\square\Gamma(x_i)}, \Gamma, \Gamma' \vdash \mathrm{fix}_{\mathrm{NC}}\ f(x).e^*[\overline{y_i/x_i}] \ominus \mathrm{fix}_{\mathrm{NC}}\ f(x).e^*[\overline{y_i/x_i}] \lesssim t :^{\mathbf{c}} \square(\tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2)$ $(\dagger\dagger)$.

By applying **c-r-let** rule in Figure 42 to $(\spadesuit\spadesuit)$ and $(\dagger\dagger)$, we can conclude as follows

$\Delta;\Phi_a;\Gamma, \Gamma' \vdash e^{**} \ominus e^{**} \lesssim t :^{\mathbf{c}} \square(\tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2)$

where $e^{**} = \mathrm{let}\ \overline{y_i = e_i\ x_i}\ \mathrm{in}\ \mathrm{fix}_{\mathrm{NC}}\ f(x).e^*[\overline{y_i/x_i}]$.

Case:
$$\frac{\begin{array}{c}\Delta;\Phi_a \vdash \tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2\ \mathtt{wf} \\ \Delta;\Phi_a;x:\tau_1, f:\tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2, \Gamma \vdash e_1 \ominus e_2 \rightsquigarrow {e_1}^* \ominus {e_2}^* \lesssim t:\tau_2 \quad (\star)\end{array}}{\Delta;\Phi_a;\Gamma \vdash \mathbf{fix}\ f(x).e_1 \ominus \mathbf{fix}\ f(x).e_2 \rightsquigarrow \mathbf{fix}\ f(x).{e_1}^* \ominus \mathbf{fix}\ f(x).{e_2}^* \lesssim 0 : \tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2}\ \textbf{e-r-fix}$$

By Theorem 56.2 on $(\star)$, we get

$\Delta;\Phi_a;x:\tau_1, f:\tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2, \Gamma \vdash e_1^* \ominus e_2^* \lesssim t :^{\mathbf{c}} \tau_2 \quad (\star\star)$.

By applying **c-r-fix** rule in Figure 40 to $(\star\star)$, we conclude as follows we get

$\Delta;\Phi_a;\Gamma \vdash \mathrm{fix}\ f(x).e_1^* \ominus \mathrm{fix}\ f(x).e_2^* \lesssim t :^{\mathbf{c}} \tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2$.

Case:
$$\frac{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \rightsquigarrow {e_1}^* \ominus {e_2}^* \lesssim t : \tau \quad (\star) \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau' \quad (\diamond) \qquad e' = \mathtt{coerce}_{\tau,\tau'} \quad (\dagger) \qquad \Delta;\Phi_a \models t \leqslant t'}{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \rightsquigarrow e'\ e_1^* \ominus e'\ e_2^* \lesssim t' : \tau'}\ \textbf{e-r-}\sqsubseteq$$

By Theorem 56.2 on $(\star)$, we get $\Delta;\Phi_a;\Gamma \vdash e_1^* \ominus e_2^* \lesssim t :^{\mathbf{c}} \tau \quad (\star\star)$.

By Lemma 48 using $(\diamond)$, we know that $\Delta;\Phi_a;\cdot \vdash e' \ominus e' \lesssim 0 :^{\mathbf{c}} \tau \xrightarrow{\mathrm{diff}(0)} \tau' \quad (\diamond\diamond)$.

By applying **c-r-app** rule in Figure 41 to $(\star\star)$ and $(\diamond\diamond)$, we get $\Delta;\Phi_a;\Gamma \vdash e'\ e_1^* \ominus e'\ e_2^* \lesssim t :^{\mathbf{c}} \tau' \quad (\spadesuit)$.

By reflexivity of binary type equivalence, we know $\Delta;\Phi_a \models \tau' \equiv \tau' \quad (\spadesuit\spadesuit)$.

Then, we conclude as follows:

$$\frac{\Delta;\Phi_a;\Gamma \vdash e'\ e_1^* \ominus e'\ e_2^* \lesssim t :^{\mathbf{c}} \tau' \quad (\spadesuit) \qquad \Delta;\Phi_a \models \tau' \equiv \tau' \quad (\spadesuit\spadesuit) \qquad \Delta;\Phi_a \models t \leqslant t' \quad (\dagger)}{\Delta;\Phi_a;\Gamma \vdash e'\ e_1^* \ominus e'\ e_2^* \lesssim t' :^{\mathbf{c}} \tau'}\ \textbf{c-r-}\sqsubseteq$$

Case:
$$\frac{\Delta;C \wedge \Phi_a;\Gamma \vdash e_1 \ominus e_2 \rightsquigarrow {e_1}^* \ominus {e_2}^* \lesssim t : \tau \quad (\star) \qquad \Delta;\neg C \wedge \Phi_a;\Gamma \vdash e_1 \ominus e_2 \rightsquigarrow e_1^{**} \ominus e_2^{**} \lesssim t : \tau \quad (\diamond) \qquad \Delta \vdash C\ \mathtt{wf} \qquad E = \mathtt{split}\ (e_1^*, e_1^{**})\ \mathbf{with}\ C \qquad E' = \mathtt{split}\ (e_2^*, e_2^{**})\ \mathbf{with}\ C}{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \rightsquigarrow E \ominus E' \lesssim t : \tau}\ \textbf{e-r-split}$$

By Theorem 56.2 on $(\star)$, we get $\Delta;C \wedge \Phi_a;\Gamma \vdash e_1^* \ominus e_2^* \lesssim t :^{\mathbf{c}} \tau \quad (\star\star)$.

By Theorem 56.2 on $(\diamond)$, we get $\Delta;\neg C \wedge \Phi_a;\Gamma \vdash e_1^{**} \ominus e_2^{**} \lesssim t :^{\mathbf{c}} \tau \quad (\diamond\diamond)$.

By applying **c-r-split** rule in Figure 40 to $(\star\star)$ and $(\diamond\diamond)$, we get

Then, we conclude as follows:

$$\frac{\Delta;\Phi_a \wedge C;\Gamma \vdash e_1^* \ominus e_2^* \lesssim t :^{\mathbf{c}} \tau \quad (\star\star) \qquad \Delta;\Phi_a \wedge \neg C;\Gamma \vdash e_1^{**} \ominus e_2^{**} \lesssim t :^{\mathbf{c}} \tau \quad (\diamond\diamond)}{\Delta;\Phi_a;\Gamma \vdash \mathtt{split}\ (e_1^*, e_1^{**})\ \mathtt{with}\ C \ominus \mathtt{split}\ (e_2^*, e_2^{**})\ \mathtt{with}\ C \lesssim t :^{\mathbf{c}} \tau}\ \textbf{c-r-split}$$

Case:
$$\frac{\Delta;\Phi_a \models \bot \quad (\star) \qquad \Delta;\Phi_a \vdash \Gamma \ \mathtt{wf} \quad (\diamond)}{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \rightsquigarrow \mathtt{contra}\ e_1 \ominus \mathtt{contra}\ e_2 \lesssim t : \tau}\ \textbf{e-r-contra}$$

By applying **c-r-contra** rule in Figure 40 to $(\star)$, we get

Then, we conclude as follows:

$$\frac{\Delta;\Phi_a \models \bot \quad (\star) \qquad (\diamond)}{\Delta;\Phi_a;\Gamma \vdash \mathtt{contra}\ e_1 \ominus \mathtt{contra}\ e_2 \lesssim t :^{c} \tau}\ \textbf{c-r-contra}$$

Case:
$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash e \ominus e' \rightsquigarrow e^* \ominus e'^* \lesssim t : \mathbf{list}[n]^{\alpha}\tau \quad (\star)\\ \Delta;\Phi_a \wedge n = 0;\Gamma \vdash e_1 \ominus e_1' \rightsquigarrow {e_1}^* \ominus e_1'^* \lesssim t' : \tau' \quad (\diamond)\\ \Phi_a' = \Phi_a \wedge n = i+1\\ i,\Delta;\Phi_a';h : \square\tau, tl : \mathbf{list}[i]^{\alpha}\tau,\Gamma \vdash e_2 \ominus e_2' \rightsquigarrow {e_2}^* \ominus e_2'^* \lesssim t' : \tau' \quad (\dagger)\\ \Phi_a'' = \Phi_a \wedge n = i+1 \wedge \alpha = \beta+1\\ i,\beta,\Delta;\Phi_a'';h : \tau, tl : \mathbf{list}[i]^{\beta}\tau,\Gamma \vdash e_2 \ominus e_2' \rightsquigarrow e_3^* \ominus e_3'^* \lesssim t' : \tau' \quad (\spadesuit)\end{array}}{\begin{array}{l}\Delta;\Phi_a;\Gamma \vdash \begin{array}{l}\mathbf{case}\ e^*\ \mathbf{of\ nil}\ \to e_1^*\\ \quad \mid h ::_{NC} tl \to e_2^*\\ \quad \mid h ::_{C} tl \to e_3^*\end{array} \ominus \begin{array}{l}\mathbf{case}\ e'^*\ \mathbf{of\ nil}\ \to e_1'^*\\ \quad \mid h ::_{NC} tl \to e_2'^*\\ \quad \mid h ::_{C} tl \to e_3'^*\end{array}\\ \qquad \rightsquigarrow E \ominus E' \lesssim t + t' : \tau'\end{array}}\ \textbf{e-r-caseL}$$

By Theorem 56.2 on $(\star)$, we get $\Delta;\Phi_a;\Gamma \vdash e^* \ominus e'^* \lesssim t :^{c} \mathrm{list}[n]^{\alpha}\tau \quad (\star\star)$.

By Theorem 56.2 on $(\diamond)$, we get $\Delta;\Phi_a \wedge n = 0;\Gamma \vdash e_1^* \ominus e_1'^* \rightsquigarrow e_1^{**} \ominus e_1'^{**} \lesssim t' : \tau' \quad (\diamond\diamond)$

By Theorem 56.2 on $(\dagger)$, we get

$i,\Delta;\Phi_a';h : \square\tau, tl : \mathrm{list}[i]^{\alpha}\tau,\Gamma \vdash e_2^* \ominus e_2'^* \rightsquigarrow e_2^{**} \ominus e_2'^{**} \lesssim t' : \tau' \quad (\dagger\dagger)$ By Theorem 56.2 on $(\spadesuit)$, we get

$i,\beta,\Delta;\Phi_a'';h : \tau, tl : \mathrm{list}[i]^{\beta}\tau,\Gamma \vdash e_3^* \ominus e_3'^* \rightsquigarrow e_3^{**} \ominus e_3'^{**} \lesssim t' : \tau' \quad (\spadesuit\spadesuit)$.

Then we conclude by applying **c-r-caseL** rule in Figure 40 to $(\star\star)$, $(\diamond,\diamond)$, $(\dagger\dagger)$, $(\spadesuit,\spadesuit)$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash e^* \ominus e'^* \lesssim t :^{c} \mathrm{list}[n]^{\alpha}\tau\\ \Delta;\Phi_a \wedge n = 0;\Gamma \vdash e_1^* \ominus e_1'^* \lesssim t' :^{c} \tau' \qquad \Phi_a' = \Phi_a \wedge n = i+1\\ i,\Delta;\Phi_a';h : \square\tau, tl : \mathrm{list}[i]^{\alpha}\tau,\Gamma \vdash e_2^* \ominus e_2'^* \lesssim t' :^{c} \tau'\\ \Phi_a'' = \Phi_a \wedge n = i+1 \wedge \alpha = \beta+1\\ i,\beta,\Delta;\Phi_a'';h' : \tau, tl : \mathrm{list}[i]^{\beta}\tau,\Gamma \vdash e_3^* \ominus e_3'^* \lesssim t' :^{c} \tau'\end{array}}{\begin{array}{l}\Delta;\Phi_a;\Gamma \vdash \begin{array}{l}\mathrm{case}\ e\ \mathrm{of\ nil}\ \to e_1^*\\ \quad \mid h ::_{N} tl \to e_2^*\\ \quad \mid h' ::_{C} tl' \to e_3^*\end{array} \ominus \begin{array}{l}\mathrm{case}\ e\ \mathrm{of\ nil}\ \to e_1'^*\\ \quad \mid h ::_{N} tl \to e_2'^*\\ \quad \mid h' ::_{C} tl' \to e_3'^*\end{array}\\ \qquad \lesssim t + t' :^{c} \tau'\end{array}}\ \textbf{c-r-caseL}$$

□

Theorem 57 (Completeness of RelCost Core)**.** *The following holds.*

1. *If* $\Delta;\Phi;\Omega \vdash^{t}_{k} e : A$ *then,* $\exists\, e^*$ *such that* $\Delta;\Phi;\Omega \vdash^{t}_{k} e \rightsquigarrow e^* : A$.
2. *If* $\Delta;\Phi;\Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau$ *then,* $\exists\, e_1^*$ *and* $\exists\, e_2^*$ *such that* $\Delta;\Phi;\Gamma \vdash e_1 \ominus e_2 \rightsquigarrow e_1^* \ominus e_2^* \lesssim t : \tau$.

Proof. Proof is by simultaneous induction on the typing derivations. The proof follows from the embedding rules presented in Figures Figures 46 to 49. We show a few representative cases.

Proof of Theorem 57.1:

Case:
$$\frac{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e : A \quad (\star) \qquad \Delta;\Phi_a \models A \sqsubseteq A' \quad (\diamond) \qquad \Delta;\Phi_a \models k' \leqslant k \quad (\dagger) \qquad \Delta;\Phi_a \models t \leqslant t' \quad (\dagger\dagger)}{\Delta;\Phi_a;\Omega \vdash^{t'}_{k'} e : A'}\ \sqsubseteq \textbf{ exec}$$

By Theorem 57.1 on $(\star)$, we get $\exists\, e^*$ such that $\Delta;\Phi;\Omega \vdash^{t}_{k} e \rightsquigarrow e^* : A \quad (\star\star)$.
By **e-u-**$\sqsubseteq$ rule using $(\star\star)$, $(\diamond)$, $(\dagger)$ and $(\dagger\dagger)$, we conclude as follows

$$\frac{\Delta;\Phi_a;\Omega \vdash^{t}_{k} e \rightsquigarrow e^* : A \qquad \Delta;\Phi_a \models^{A} A \sqsubseteq A' \qquad \Delta;\Phi_a \models k' \leqslant k \qquad \Delta;\Phi_a \models t \leqslant t'}{\Delta;\Phi_a;\Omega \vdash^{t'}_{k'} e \rightsquigarrow e^* : A'}\ \textbf{e-u-}\sqsubseteq$$

Proof of Theorem 57.2:

Case:
$$\frac{\Delta;\Phi_a;|\Gamma|_1 \vdash^{t_1}_{k_1} e_1 : A_1 \quad (\star) \qquad \Delta;\Phi_a;|\Gamma|_2 \vdash^{t_2}_{k_2} e_2 : A_2 \quad (\diamond)}{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \lesssim t_1 - k_2 : U\,(A_1, A_2)}\ \textbf{switch}$$

By Theorem 57.1 on $(\star)$, we get $\exists\, e_1^*$ such that $\Delta;\Phi;\Omega \vdash^{t_1}_{k_1} e_1 \rightsquigarrow {e_1}^* : A_1 \quad (\star\star)$.
By Theorem 57.1 on $(\diamond)$, we get $\exists\, e_2^*$ such that $\Delta;\Phi;\Omega \vdash^{t_2}_{k_2} e_2 \rightsquigarrow {e_2}^* : A_2 \quad (\diamond\diamond)$.
By **e-switch** embedding rule using $(\star\star)$ and $(\diamond\diamond)$, we can conclude as follows:

$$\frac{\Delta;\Phi_a;|\Gamma|_1 \vdash^{t_1}_{k_1} e_1 \rightsquigarrow e_1^* : A_1 \quad (\star\star) \qquad \Delta;\Phi_a;|\Gamma|_2 \vdash^{t_2}_{k_2} e_2 \rightsquigarrow e_2^* : A_2 \quad (\diamond\diamond) \qquad E = \texttt{switch}\ e_1^* \qquad E' = \texttt{switch}\ e_2^*}{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \rightsquigarrow E \ominus E' \lesssim t_1 - k_2 : U\,(A_1, A_2)}\ \textbf{e-switch}.$$

Case:
$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash e \ominus e \lesssim t : \tau \quad (\star)\\ \forall x \in \mathrm{dom}(\Gamma).\ \ \Delta;\Phi_a \models \Gamma(x) \sqsubseteq \Box\Gamma(x) \quad (\diamond)\end{array}}{\Delta;\Phi_a;\Gamma,\Gamma';\Omega \vdash e \ominus e \lesssim 0 : \Box\tau}\ \textbf{nochange}$$

By Theorem 57.2 on $(\star)$, we get $\exists\, e^*$ such that
$\Delta;\Phi;\Gamma \vdash e \ominus e \rightsquigarrow e^* \ominus e^* \lesssim t : \tau \quad (\dagger)$.
By Lemma 48 on $(\diamond)$, we get
$\exists e_i = \mathtt{coerce}_{\Gamma(x_i),\Box(\Gamma(x_i)}$ for all $x_i \in \mathrm{dom}(\Gamma)$ $(\dagger\dagger)$.
By **e-nochange** embedding rule using $(\dagger)$ and $(\dagger\dagger)$, we can conclude as follows:

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash e \ominus e \rightsquigarrow e^* \ominus e^* \lesssim t : \tau \quad (\dagger)\\ \forall x_i \in \mathrm{dom}(\Gamma),\ \ e_i = \mathtt{coerce}_{\Gamma(x_i),\Box\Gamma(x_i)} \quad (\dagger\dagger)\\ e' = \mathtt{let}\ \overline{y_i = e_i\ x_i}\ \mathtt{in}\ \mathtt{NC}\ e^*[\overline{y_i/x_i}]\end{array}}{\Delta;\Phi_a;\Gamma,\Gamma' \vdash e \ominus e \rightsquigarrow e' \ominus e' \lesssim 0 : \Box\tau}\ \textbf{e-nochange}.$$

Case:
$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash e \ominus e' \lesssim t : \mathbf{list}[n]^\alpha\,\tau \quad (\star)\\ \Delta;\Phi_a \wedge n = 0;\Gamma \vdash e_1 \ominus e_1' \lesssim t' : \tau' \quad (\diamond)\\ i,\Delta;\Phi_a \wedge n = i+1;h : \Box\tau, tl : \mathbf{list}[i]^\alpha\,\tau,\Gamma \vdash e_2 \ominus e_2' \lesssim t' : \tau' \quad (\dagger)\\ i,\beta,\Delta;\Phi_a \wedge n = i+1 \wedge \alpha = \beta+1;h : \tau, tl : \mathbf{list}[i]^\beta\,\tau,\Gamma \vdash e_2 \ominus e_2' \lesssim t' : \tau' \quad (\spadesuit)\end{array}}{\Delta;\Phi_a;\Gamma \vdash \begin{array}{l}\mathbf{case}\ e\ \mathbf{of\ nil} \rightarrow e_1\\ \mid h :: tl \rightarrow e_2\end{array} \ominus \begin{array}{l}\mathbf{case}\ e'\ \mathbf{of\ nil} \rightarrow e_1'\\ \mid h :: tl \rightarrow e_2'\end{array} \lesssim t + t' : \tau'}\ \textbf{r-caseL}$$

By Theorem 57.2 on $(\star)$, we get $\exists\, e^*$ and $\exists\, e'^*$ s.t.
$\Delta;\Phi;\Gamma \vdash e \ominus e' \rightsquigarrow e^* \ominus e'^* \lesssim t : \mathrm{list}[n]^\alpha\,\tau \quad (\star\star)$.
By Theorem 57.2 on $(\diamond)$, we get $\exists\, e_1^*$ and $\exists\, e_1'^*$ s.t.
$\Delta; n \doteq 0 \wedge \Phi;\Gamma \vdash e_1 \ominus e_1' \rightsquigarrow e_1^* \ominus e_1'^* \lesssim t : \tau' \quad (\diamond\diamond)$.
By Theorem 57.2 on $(\dagger)$, we get $\exists\, e_2^*$ and $\exists\, e_2'^*$ s.t.
$i :: S,\Delta;\Phi_a';h : \Box\tau, tl : \mathrm{list}[i]^\alpha\,\tau,\Gamma \vdash e_2 \ominus e_2' \rightsquigarrow e_2^* \ominus e_2'^* \lesssim t : \tau' \quad (\dagger\dagger)$
where $\Phi_a' = \Phi_a \wedge n = i+1$.
By Theorem 57.2 on $(\spadesuit)$, we get $\exists\, e_3^*$ and $\exists\, e_3'^*$ s.t.
$i :: S, \beta :: S,\Delta;\Phi_a'';h : \tau, tl : \mathrm{list}[i]^\beta\,\tau,\Gamma \vdash e_2 \ominus e_2' \rightsquigarrow e_3^* \ominus e_3'^* \lesssim t : \tau' \quad (\spadesuit\spadesuit)$.
where $\Phi_a'' = \Phi_a \wedge n = i+1 \wedge \alpha = \beta+1$.
By **e-caseL** embedding rule using $(\star\star)$, $(\diamond\diamond)$, and $(\spadesuit\spadesuit)$, we can conclude as follows

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma\vdash e\ominus e'\rightsquigarrow e^*\ominus e'^*\lesssim t:\mathrm{list}[n]^{\alpha}\,\tau\\ \Delta;\Phi_a\wedge n=0;\Gamma\vdash e_1\ominus e_1'\rightsquigarrow {e_1}^*\ominus {e_1'}^*\lesssim t':\tau'\\ \Phi_a'=\Phi_a\wedge n=i+1\\ i,\Delta;\Phi_a';h:\square\tau,tl:\mathrm{list}[i]^{\alpha}\,\tau,\Gamma\vdash e_2\ominus e_2'\rightsquigarrow {e_2}^*\ominus {e_2'}^*\lesssim t':\tau'\\ \Phi_a''=\Phi_a\wedge n=i+1\wedge\alpha=\beta+1\\ i,\beta,\Delta;\Phi_a'';h:\tau,tl:\mathrm{list}[i]^{\beta}\,\tau,\Gamma\vdash e_2\ominus e_2'\rightsquigarrow e_3^*\ominus e_3'^*\lesssim t':\tau'\\ \begin{array}{ll}\text{case } e^* \text{ of nil } \to e_1^* & \text{case } e'^* \text{ of nil } \to e_1'^*\\ E=\mid h ::_{NC} tl\to e_2^* & E'=\mid h ::_{NC} tl\to e_2'^*\\ \quad\mid h ::_{C} tl\to e_3^* & \quad\mid h ::_{C} tl\to e_3'^*\end{array}\end{array}}{\Delta;\Phi_a;\Gamma\vdash \begin{array}{l}\text{case } e \text{ of nil } \to e_1\\ \mid h :: tl\to e_2\end{array}\ominus\begin{array}{l}\text{case } e \text{ of nil } \to e_1'\\ \mid h :: tl\to e_2'\end{array}\rightsquigarrow E\ominus E'\lesssim t+t':\tau'}\ \textbf{e-r-caseL}$$

Case:
$$\frac{\begin{array}{c}\Delta;\Phi_a\vdash\tau_1\xrightarrow{\mathbf{diff}(t)}\tau_2\ \mathtt{wf}\\ \Delta;\Phi_a;x:\tau_1,f:\square(\tau_1\xrightarrow{\mathbf{diff}(t)}\tau_2),\Gamma\vdash e\ominus e\lesssim t:\tau_2\quad(\star)\\ \forall x\in\mathrm{dom}(\Gamma).\ \ \Delta;\Phi_a\models\Gamma(x)\sqsubseteq\square\Gamma(x)\quad(\diamond)\end{array}}{\Delta;\Phi_a;\Gamma\vdash \mathbf{fix}\ f(x).e\ominus\mathbf{fix}\ f(x).e\lesssim 0:\square(\tau_1\xrightarrow{\mathbf{diff}(t)}\tau_2)}\ \textbf{r-fixNC}$$

By Theorem 57.2 on $(\star)$, we get $\exists\, e^*$ such that $\Delta;\Phi;x:\tau_1,f:\square(\tau_1\xrightarrow{\mathrm{diff}(t)}\tau_2),\Gamma\vdash e\ominus e\rightsquigarrow e^*\ominus e^*\lesssim t:\tau_2\quad(\star\star)$.
By Lemma 48 on $(\diamond)$, we get $\exists e_i=\mathtt{coerce}_{\Gamma(x_i),\square(\Gamma(x_i)}$ for all $x_i\in\mathrm{dom}(\Gamma)\quad(\diamond\diamond)$.

By **e-fixNC** embedding rule using $(\star\star)$ and $(\diamond\diamond)$, we can conclude as follows:

$$\frac{\begin{array}{c}\Delta;\Phi_a\vdash\tau_1\xrightarrow{\mathrm{diff}(t)}\tau_2\ \mathtt{wf}\\ \Delta;\Phi_a;x:\tau_1,f:\square(\tau_1\xrightarrow{\mathrm{diff}(t)}\tau_2),\Gamma\vdash e\ominus e\rightsquigarrow e^*\ominus e^*\lesssim t:\tau_2\\ \forall x_i\in\mathrm{dom}(\Gamma),\quad e_i=\mathtt{coerce}_{\Gamma(x_i),\square\Gamma(x_i)}\\ e^{**}=\mathrm{let}\ \overline{y_i=e_i\ x_i}\ \mathrm{in}\ \mathrm{fix}_{NC}\ f(x).e^*[\overline{y_i/x_i}]\end{array}}{\Delta;\Phi_a;\Gamma\vdash \mathrm{fix}\ f(x).e\ominus\mathrm{fix}\ f(x).e\rightsquigarrow e^{**}\ominus e^{**}\lesssim 0:\square(\tau_1\xrightarrow{\mathrm{diff}(t)}\tau_2)}\ \textbf{e-r-fixNC}.$$

Case:
$$\frac{i::S,\Delta;\Phi_a;\Gamma\vdash e\ominus e'\lesssim t:\tau\quad(\star)\qquad i\notin\mathbf{FIV}(\Phi_a;\Gamma)}{\Delta;\Phi_a;\Gamma\vdash\Lambda.e\ominus\Lambda.e'\lesssim 0:\forall i\overset{\mathbf{diff}(t)}{::}S.\,\tau}\ \textbf{r-iLam}$$

By Theorem 57.2 on $(\star)$, we get $\exists\, e^*$ and $\exists\, e'^*$ such that $i::S,\Delta;\Phi;\Gamma\vdash e\ominus e'\rightsquigarrow e^*\ominus e'^*\lesssim t:\tau\quad(\star\star)$.

By **e-iLam** embedding rule using $(\star\star)$, we can conclude as follows:

$$\frac{i :: S, \Delta; \Phi_a; \Gamma \vdash e \ominus e' \rightsquigarrow e^* \ominus e'^* \lesssim t : \tau \qquad i \notin FIV(\Phi_a; \Gamma)}{\Delta; \Phi_a; \Gamma \vdash \Lambda.e \ominus \Lambda.e' \rightsquigarrow \Lambda i.e^* \ominus \Lambda i.e'^* \lesssim 0 : \forall i \overset{diff(t)}{::} S.\, \tau}\ \textbf{e-r-iLam}.$$

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t : \forall i \overset{\mathbf{diff}(t')}{::} S.\, \tau \quad (\star) \qquad \Delta \vdash I : S \quad (\diamond)}{\Delta; \Phi_a; \Gamma \vdash e[] \ominus e'[] \lesssim t + t'[I/i] : \tau\{I/i\}}\ \textbf{r-iApp}$

By Theorem 57.2 on $(\star)$, we get $\exists\, e^*$ such that

$\Delta; \Phi; \Gamma \vdash e \ominus e' \rightsquigarrow e^* \ominus e'^* \lesssim t : \forall i \overset{exec(t',\tau)}{::} S. \quad (\star\star)$.

By **e-iApp** embedding rule using $(\star\star)$ and $(\diamond)$, we can conclude as follows:

$$\frac{\Delta; \Phi_a; \Gamma \vdash e \ominus e' \rightsquigarrow e^* \ominus e'^* \lesssim t : \forall i \overset{diff(t')}{::} S.\, \tau \qquad \Delta \vdash I : S}{\Delta; \Phi_a; \Gamma \vdash e[] \ominus e'[] \rightsquigarrow e^*[I] \ominus e'^*[I] \lesssim t + t'[I/i] : \tau\{I/i\}}\ \textbf{e-r-iApp}.$$

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t : \tau\{I/i\} \quad (\star) \qquad \Delta \vdash I :: S \quad (\diamond)}{\Delta; \Phi_a; \Gamma \vdash \mathbf{pack}\ e \ominus \mathbf{pack}\ e' \lesssim t : \exists i{::}S.\, \tau}\ \textbf{r-pack}$

By Theorem 57.2 on $(\star)$, we get $\exists\, e^*$ and $\exists\, e'^*$ such that

$\Delta; \Phi; \Gamma \vdash e \ominus e' \rightsquigarrow e^* \ominus e'^* \lesssim t : \tau\{I/i\} \quad (\star\star)$.

By **e-pack** embedding rule using $(\star\star)$ and $(\diamond)$, we can conclude as follows:

$$\frac{\begin{array}{c}\Delta; \Phi_a; \Gamma \vdash e \ominus e' \rightsquigarrow e^* \ominus e'^* \lesssim t : \tau\{I/i\} \\ \Delta \vdash I :: S \qquad E = \text{pack } e^* \text{ with } I \qquad E' = \text{pack } e'^* \text{ with } I\end{array}}{\Delta; \Phi_a; \Gamma \vdash \text{pack } e \ominus \text{pack } e' \rightsquigarrow E \ominus E' \lesssim t : \exists i{::}S.\, \tau}\ \textbf{e-r-pack}.$$

Case: $\dfrac{\begin{array}{c}\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_1' \lesssim t_1 : \exists i{::}S.\, \tau_1 \quad (\star) \\ i :: S, \Delta; \Phi_a; x : \tau_1, \Gamma \vdash e_2 \ominus e_2' \lesssim t_2 : \tau_2 \quad (\diamond) \\ i \notin FV(\Phi_a; \Gamma, \tau_2, t_2)\end{array}}{\Delta; \Phi_a; \Gamma \vdash \mathbf{unpack}\ e_1\ \mathbf{as}\ x\ \mathbf{in}\ e_2 \ominus \mathbf{unpack}\ e_1'\ \mathbf{as}\ x\ \mathbf{in}\ e_2' \lesssim t_1 + t_2 : \tau_2}\ \textbf{r-unpack}$

By Theorem 57.2 on $(\star)$, we get $\exists\, e_1^*$ and $\exists\, e_1'^*$ such that

$\Delta; \Phi; \Gamma \vdash e_1 \ominus e_1' \rightsquigarrow e_1^* \ominus e_1'^* \lesssim t : \exists i{::}S.\, \tau_1 \quad (\star\star)$.

By Theorem 57.2 on $(\diamond)$, we get $\exists\, e_2^*$ and $\exists\, e_2'^*$ such that

$i :: S, \Delta; \Phi; x : \tau_1, \Gamma \vdash e_2 \ominus e_2' \rightsquigarrow e_2^* \ominus e_2'^* \lesssim t : \tau_2 \quad (\diamond\diamond)$.

By **e-unpack** embedding rule using $(\star\star)$ and $(\diamond\diamond)$, we can conclude

as follows:

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_1' \rightsquigarrow {e_1}^* \ominus {e_1'}^* \lesssim t_1 : \exists i{::}S.\,\tau_1 \\ i :: S, \Delta;\Phi_a;x:\tau_1,\Gamma \vdash e_2 \ominus e_2' \rightsquigarrow {e_2}^* \ominus {e_2'}^* \lesssim t_2 : \tau_2 \\ i \notin FV(\Phi_a;\Gamma,\tau_2,t_2) \\ E = \text{unpack } e_1^* \text{ as } (x,i) \text{ in } e_2^* \qquad E' = \text{unpack } e_1'^* \text{ as } (x,i) \text{ in } e_2'^*\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{unpack } e_1 \text{ as } x \text{ in } e_2 \ominus \text{unpack } e_1' \text{ as } x \text{ in } e_2' \rightsquigarrow E \ominus E' \lesssim t_1 + t_2 : \tau_2}\ \textbf{e-r-unpac}$$

Case: $\dfrac{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau \quad (\star) \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau' \quad (\diamond) \qquad \Delta;\Phi_a \models t \leqslant t' \quad \dagger}{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \lesssim t' : \tau'}\ \textbf{r-}\sqsubseteq$

By Theorem 57.2 on $(\star)$, we get $\exists\, e_1^*, e_2^*$ such that
$\Delta;\Phi;\Gamma \vdash e_1 \ominus e_2 \rightsquigarrow {e_1}^* \ominus {e_2}^* \lesssim t : \tau \quad (\star\star)$.
By Lemma 48 on $(\diamond)$, we can show that $\exists e' = \mathtt{coerce}_{\tau,\tau'} \quad (\diamond\diamond)$.
By **e-r-**$\sqsubseteq$ rule using $(\star\star)$, $(\diamond\diamond)$ and $(\dagger)$, we conclude as follows

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \rightsquigarrow {e_1}^* \ominus {e_2}^* \lesssim t : \tau \\ \Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad e' = \mathtt{coerce}_{\tau,\tau'} \qquad \Delta;\Phi_a \models t \leqslant t'\end{array}}{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \rightsquigarrow e'\ {e_1}^* \ominus e'\ {e_2}^* \lesssim t' : \tau'}\ \textbf{e-r-}\sqsubseteq$$

□

Theorem 58 (Invariant of the Algorithmic Typechecking)**.** *We have the following.*

1. *Assume that* $\Delta;\psi_a;\Phi_a;\Omega \vdash e \downarrow A,k,t \Rightarrow \Phi$ *and* $FIV(\Phi_a,\Omega;A,k,t) \subseteq \mathrm{dom}(\Delta,\psi_a)$. *Then* $FIV(\Phi) \subseteq \mathrm{dom}(\Delta;\psi_a)$.

2. *Assume that* $\Delta;\psi_a;\Phi_a;\Omega \vdash e \uparrow A \Rightarrow [\psi],k,t,\Phi$ *and* $FIV(\Phi_a,\Omega) \subseteq \mathrm{dom}(\Delta,\psi_a)$. *Then* $FIV(A,k,t,\Phi) \subseteq \mathrm{dom}(\Delta,\psi;\psi_a)$.

3. *Assume that* $\Delta;\psi_a;\Phi_a;\Gamma \vdash e \ominus e' \downarrow \tau,t \Rightarrow \Phi$ *and* $FIV(\Phi_a,\Gamma,\tau,t) \subseteq \mathrm{dom}(\Delta,\psi_a)$.*Then* $FIV(\Phi) \subseteq \mathrm{dom}(\Delta;\psi_a)$.

4. *Assume that* $\Delta;\psi_a;\Phi_a;\Gamma \vdash e \ominus e' \uparrow \tau \Rightarrow [\psi],t,\Phi$ *and* $FIV(\Phi_a,\Gamma) \subseteq \mathrm{dom}(\Delta,\psi_a)$.*Then* $FIV(\tau,t,\Phi) \subseteq \mathrm{dom}(\Delta;\psi;\psi_a)$.

C.2 BIRELCOST THEOREMS

Theorem 59 (Soundness of the Algorithmic Typechecking)**.** *We have the following.*

1. *Assume that* $\Delta; \psi_a; \Phi_a; \Omega \vdash e \downarrow A, k, t \Rightarrow \Phi$ *and*
   a) $FIV(\Phi_a, \Omega, A, k, t) \subseteq dom(\Delta, \psi_a)$
   b) $\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$ *is provable for some* θ_a *such that* $\Delta \rhd \theta_a : \psi_a$ *is derivable*

   Then $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t[\theta_a]}_{k[\theta_a]} |e| :^{c} A[\theta_a]$.

2. *Assume that* $\Delta; \psi_a; \Phi_a; \Omega \vdash e \uparrow A \Rightarrow [\psi], k, t, \Phi$ *and*
   a) $FIV(\Phi_a, \Omega) \subseteq dom(\Delta, \psi_a)$
   b) $\forall \theta\ \forall \theta_a.\ \Delta; \Phi_a[\theta_a] \models \Phi[\theta\,\theta_a]$ *is provable s.t* $\Delta \rhd \theta : \psi$ *and* $\Delta \rhd \theta_a : \psi_a$ *are derivable*

   Then $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t[\theta\,\theta_a]}_{k[\theta\,\theta_a]} |e| :^{c} A[\theta\,\theta_a]$.

3. *Assume that* $\Delta; \psi_a; \Phi_a; \Gamma \vdash e \ominus e' \downarrow \tau, t \Rightarrow \Phi$ *and*
   a) $FIV(\Phi_a, \Gamma, \tau, t) \subseteq dom(\Delta, \psi_a)$
   b) $\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$ *is provable for some* θ_a *such that* $\Delta \rhd \theta_a : \psi_a$ *is derivable*

   Then $\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |e| \ominus |e'| \lesssim t[\theta_a] :^{c} \tau[\theta_a]$.

4. *Assume that* $\Delta; \psi_a; \Phi_a; \Gamma \vdash e \ominus e' \uparrow \tau \Rightarrow [\psi], t, \Phi$ *and*
   a) $FIV(\Phi_a, \Gamma) \subseteq dom(\Delta, \psi_a)$
   b) $\forall \theta\ \forall \theta_a.\ \Delta; \Phi_a[\theta_a] \models \Phi[\theta\,\theta_a]$ *is provable s.t* $\Delta \rhd \theta : \psi$ *and* $\Delta \rhd \theta_a : \psi_a$ *are derivable*

   Then $\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |e| \ominus |e'| \lesssim t[\theta\,\theta_a] :^{c} \tau[\theta\,\theta_a]$.

Proof. Statements (1—4) follow from simultaneous structural induction on the algorithmic typing derivations. We present several cases below.

Proof of Theorem 59.1:

Case:

$$\frac{k_1, t_1, k_2, t_2 \in \textbf{fresh}(\mathbb{R}) \qquad \Delta; k_1, t_1, \psi_a; \Phi_a; \Omega \vdash e_1 \downarrow A_1, k_1, t_1 \Rightarrow \Phi_1 \qquad \Delta; k_2, t_2, \psi_a; \Phi_a; \Omega \vdash e_2 \downarrow A_1, k_2, t_2 \Rightarrow \Phi_2 \qquad \Phi = \exists k_1, t_1 :: \mathbb{R}.\Phi_1 \wedge \exists k_2, t_2 :: \mathbb{R}.\Phi_2 \wedge t_1 + t_2 \doteq t \wedge k \doteq k_1 + k_2}{\Delta; \psi_a; \Phi_a; \Omega \vdash \langle e_1, e_2 \rangle \downarrow A_1 \times A_2, k, t \Rightarrow \Phi}\ \textbf{alg-u-prod-}\downarrow$$

TS: $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t[\theta_a]}_{k[\theta_a]} \langle |e_1|, |e_2| \rangle :^{c} A_1[\theta_a] \times A_2[\theta_a]$.

By the main assumptions, we have

$\mathrm{FIV}(\Phi_a, \Omega, A, k, t) \subseteq \mathrm{dom}(\Delta, \psi_a)$ $(\star)$

$\Delta; \Phi_a[\theta_a] \models$

$(\exists k_1, t_1 :: \mathbb{R}.\Phi_1 \wedge \exists k_2, t_2 :: \mathbb{R}.\Phi_2 \wedge (t_1 + t_2) \doteq t \wedge (k_1 + k_2) \doteq k)[\theta_a]$ $(\star\star)$

Using $(\star)$, $(\star\star)$'s derivation must be in a form such that we have

a) $\Delta \vdash K_1 :: \mathbb{R}$ and $\Delta \vdash T_1 :: \mathbb{R}$

b) $\Delta \vdash K_2 :: \mathbb{R}$ and $\Delta \vdash T_2 :: \mathbb{R}$

c) $\Delta; \Phi_a[\theta_a] \models \Phi_1[\theta_a, k_1 \mapsto K_1, t_1 \mapsto T_1]$

d) $\Delta; \Phi_a[\theta_a] \models \Phi_2[\theta_a, k_2 \mapsto K_2, t_2 \mapsto T_2]$

e) $\Delta; \Phi_a[\theta_a] \models (T_1 + T_2) \doteq t[\theta_a] \wedge (K_1 + K_2) \doteq k[\theta_a]$

By Theorem 59.1 on the first premise using $(\star)$ and c), we can show that

$$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{T_1}_{K_1} |e_1| :^{c} A_1[\theta_a] \tag{1}$$

By Theorem 59.1 on the second premise using $(\star)$ and d), we can show that

$$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{T_2}_{K_2} |e_2| :^{c} A_2[\theta_a] \tag{2}$$

Combining eqs. (1) and (2) with **c-prod** rule, we get

$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{T_1+T_2}_{K_1+K_2} \langle |e_1|, |e_2| \rangle :^{c} A_1[\theta_a] \times A_2[\theta_a]$.

Then, by using e) with the **c-$\sqsubseteq$** exec rule , we can conclude that

$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t[\theta_a]}_{k[\theta_a]} \langle |e_1|, |e_2| \rangle :^{c} A_1[\theta_a] \times A_2[\theta_a]$.

Case:
$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Omega \vdash e \uparrow A' \Rightarrow [\psi], k', t', \Phi_1 \\ \Delta;\psi,\psi_a;\Phi_a \models^{A} A' \sqsubseteq A \Rightarrow \Phi_2\end{array}}{\Delta;\psi_a;\Phi_a;\Omega \vdash e \downarrow A, k, t \Rightarrow \exists(\psi).\Phi_1 \wedge \Phi_2 \wedge t' \leqslant t \wedge k \leqslant k'}\ \textbf{alg-}\uparrow\downarrow$$

TS: $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{t[\theta_a]}_{k[\theta_a]} |e| :^{c} A[\theta_a]$.

By the main assumptions, we have

$\mathrm{FIV}(\Phi_a,\Omega,A,k,t) \subseteq \mathrm{dom}(\Delta,\psi_a)$ $(\star)$ and

$\Delta;\Phi_a[\theta_a] \models (\exists(\psi).\Phi_1 \wedge \Phi_2 \wedge t' \leqslant t \wedge k \leqslant k')[\theta_a]$ $(\star\star)$

By Theorem 58 using $(\star)$ and the first premise, we get

$\mathrm{FIV}(A',k',t',\Phi_1) \subseteq \mathrm{dom}(\Delta,\psi;\psi_a)$ $(\diamond)$.

Using $(\star)$ and $(\diamond)$, $(\star\star)$'s derivation must be in a form such that we have

a) $\Delta \rhd \theta_a : \psi_a$

b) $\Delta;\Phi_a[\theta_a] \models \Phi_1[\theta_a,\theta_a]$

c) $\Delta;\Phi_a[\theta_a] \models \Phi_2[\theta_a,\theta_a]$

d) $\Delta;\Phi_a[\theta_a] \models t'[\theta\,\theta_a] \leqslant t[\theta_a] \wedge k[\theta_a] \leqslant k'[\theta\,\theta_a]$

By Theorem 59.2 on the first premise using $(\star)$, a) and b), we can show that

$$\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{t'[\theta\,\theta_a]}_{k'[\theta\,\theta_a]} |e| :^{c} A'[\theta\,\theta_a] \tag{1}$$

By Theorem 52 using the second premise and c), we obtain

$$\Delta;\Phi_a[\theta_a] \models^{A} A'[\theta\,\theta_a] \sqsubseteq A[\theta\,\theta_a] \tag{2}$$

Note that due to $(\star)$, we have $A[\theta\,\theta_a] = A[\theta_a]$.

Then we can conclude by the **c-⊑** exec rule using eqs. (1) and (2) and (d) that $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{t[\theta_a]}_{k[\theta_a]} |e| :^{c} A[\theta_a]$.

Case:
$$\frac{\Delta;\psi_a;\Phi_a;f : A_1 \xrightarrow{\mathbf{exec}(k',t')} A_2, x : A_1, \Omega \vdash e \downarrow A_2, k', t' \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Omega \vdash \mathbf{fix}\ f(x).e \downarrow A_1 \xrightarrow{\mathbf{exec}(k',t')} A_2, k, t \Rightarrow \Phi \wedge k \doteq 0 \wedge 0 \doteq t}\ \textbf{alg-u-fix-}\downarrow$$

TS: $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{t[\theta_a]}_{k[\theta_a]} \mathrm{fix}\ f(x).|e| :^{c} A_1[\theta_a] \xrightarrow{\mathrm{exec}(k'[\theta_a],t'[\theta_a])} A_2[\theta_a]$.

By the main assumptions, we have

$\mathrm{FIV}(\Phi_a, \Omega, A_1 \xrightarrow{\mathrm{exec}(k',t')}, k, t) \subseteq \mathrm{dom}(\Delta, \psi_a)$ $(\star)$ and
$\Delta; \Phi_a[\theta_a] \models \Phi \wedge 0 \doteq k \wedge 0 \doteq t'[\theta_a]$ $(\star\star)$
Using $(\star)$, we can show that

a) $\mathrm{FIV}(\Phi_a, \Omega, A_1, A_1 \xrightarrow{\mathrm{exec}(k',t')}, A_2, k', t') \subseteq \mathrm{dom}(\Delta, \psi_a)$.

We also can show that $(\star\star)$'s derivation must be in a form such that we have

b) $\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$
c) $\Delta; \Phi_a[\theta_a] \models 0 \doteq k[\theta_a]$
d) $\Delta; \Phi_a[\theta_a] \models 0 \doteq t[\theta_a]$

By Theorem 59.1 on the first premise using a) and b), we can show that

$$\Delta; \Phi_a[\theta_a]; x : A_1[\theta_a], f : A_1[\theta_a] \xrightarrow{\mathrm{exec}(k'[\theta_a],t'[\theta_a])} A_2[\theta_a], \Omega[\theta_a] \vdash^{t'[\theta_a]}_{k'[\theta_a]} |e| :^{c} A_2[\theta_a] \tag{1}$$

By the **c-fix** rule using eq. (1), we obtain
$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{0}_{0} \mathrm{fix}\ f(x).|e| :^{c} A_1[\theta_a] \xrightarrow{\mathrm{exec}(k'[\theta_a],t'[\theta_a])} A_2[\theta_a]$.
By **c-$\sqsubseteq$** exec rule using (c) and (d), we obtain
$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t[\theta_a]}_{k[\theta_a]} \mathrm{fix}\ f(x).|e| :^{c} A_1[\theta_a] \xrightarrow{\mathrm{exec}(k'[\theta_a],t'[\theta_a])} A_2[\theta_a]$.

Case:
$$\frac{i :: S, \Delta; \psi_a; \Phi_a; \Omega \vdash e \downarrow A, k_e, t_e \Rightarrow \Phi \qquad \Phi' = (\forall i :: S.\Phi) \wedge k \doteq 0 \wedge 0 \doteq t}{\Delta; \psi_a; \Phi_a; \Omega \vdash \Lambda i.e \downarrow \forall i \overset{\mathrm{exec}(k_e,t_e)}{::} S.A, k, t \Rightarrow \Phi'}\ \textbf{alg-u-iLam-}\downarrow$$

TS: $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t[\theta_a]}_{k[\theta_a]} \Lambda i.|e| :^{c} \forall i \overset{\mathrm{exec}(k_e[\theta_a],t_e[\theta_a])}{::} S.A[\theta_a]$.
By the main assumptions, we have
$\mathrm{FIV}(\Phi_a, \Omega, \forall i \overset{\mathrm{exec}(k_e,t_e)}{::} S.A, k, t) \subseteq \mathrm{dom}(\Delta, \psi_a)$ $(\star)$ and
$\Delta; \Phi_a[\theta_a] \models ((\forall i :: S.\Phi) \wedge 0 \doteq k \wedge 0 \doteq t)[\theta_a]$ $(\star\star)$
Using $(\star)$, we can show that

a) $\mathrm{FIV}(\Phi_a, \Omega, A, k_e, t_e) \subseteq i, \mathrm{dom}(\Delta, \psi_a)$.

We can also show that $(\star\star)$'s derivation must be in a form such that we have

b) $i :: S, \Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$
c) $\Delta; \Phi_a[\theta_a] \models 0 \doteq k[\theta_a]$
d) $\Delta; \Phi_a[\theta_a] \models 0 \doteq t[\theta_a]$

By Theorem 59.1 on the premise using a) and b), we can show that

$$i :: S, \Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t_e[\theta_a]}_{k_e[\theta_a]} |e| :^{c} A[\theta_a] \tag{1}$$

By the **c-iLam** rule using eq. (1), we obtain
$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{0}_{0} \Lambda i.|e| :^{c} \forall i \overset{\text{exec}(k_e[\theta_a], t_e[\theta_a])}{::} S. A[\theta_a]$.
By **c-⊑** exec rule using (c) and (d), we obtain
$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t[\theta_a]}_{k[\theta_a]} \Lambda i.|e| :^{c} \forall i \overset{\text{exec}(k_e[\theta_a], t_e[\theta_a])}{::} S. A[\theta_a]$.

Case: $\dfrac{\Delta; \psi_a; \Phi_a; \Omega \vdash e \downarrow A\{I/i\}, k, t \Rightarrow \Phi \qquad \Delta \vdash I :: S}{\Delta; \psi_a; \Phi_a; \Omega \vdash \textbf{pack } e \textbf{ with } I \downarrow \exists i{::}S. A, k, t \Rightarrow \Phi}$ **alg-u-pack-↓**

TS: $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t[\theta_a]}_{k[\theta_a]} \text{pack } |e| \text{ with } I :^{c} \exists i{::}S. A[\theta_a]$.
By the main assumptions, we have
$\text{FIV}(\Phi_a, \Omega, \exists i{::}S. A, k, t) \subseteq \text{dom}(\Delta, \psi_a)$ $(\star)$ and
$\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$ $(\star\star)$
Using $(\star)$ and the second premise, we can show that

a) $\text{FIV}(\Phi_a, \Omega, A\{I/i\}, k, t) \subseteq \text{dom}(\Delta, \psi_a)$.

By Theorem 59.1 on the premise using a) and $(\star\star)$, we can show that

$$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t[\theta_a]}_{k[\theta_a]} |e| :^{c} A[\theta_a]\{I/i\} \tag{1}$$

By the **c-pack** rule using eq. (1) and the second premise, we obtain
$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t[\theta_a]}_{k[\theta_a]} \text{pack } |e| \text{ with } I :^{c} \exists i{::}S. A[\theta_a]$.

Case:
$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Omega \vdash e_1 \uparrow \exists i{::}S.\, A_1 \Rightarrow [\psi], k_1, t_1, \Phi_1 \\ k_2, t_2 \in \mathbf{fresh}(\mathbb{R}) \\ i :: S, \Delta; k_2, t_2, \psi, \psi_a; \Phi_a; x : A_1, \Omega \vdash e_2 \downarrow A_2, k_2, t_2 \Rightarrow \Phi_2 \\ i \notin FV(\Phi_a; \Omega, A_2, k_2, t_2) \\ \Phi_c = k \doteq k_1 + k_2 + c_{unp} \wedge t_1 + t_2 + c_{unp} \doteq t \\ \Phi = \Phi_1 \wedge \exists k_2, t_2 :: \mathbb{R}. \forall i :: S. \Phi_2 \wedge \Phi_c\end{array}}{\Delta;\psi_a;\Phi_a;\Omega \vdash \mathbf{unpack}\ e_1\ \mathbf{as}\ (x,i)\ \mathbf{in}\ e_2 \downarrow A_2, k, t \Rightarrow \exists(\psi).\Phi}\ \textbf{alg-u-unpack-}\downarrow$$

TS: $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{t[\theta_a]}_{k[\theta_a]} \text{unpack } |e_1| \text{ with } (x,i) \text{ in } |e_2| :^{c} A_2[\theta_a]$.

By the main assumptions, we have

$FIV(\Phi_a, \Omega, A_2, k, t) \subseteq dom(\Delta, \psi_a)$ $(\star)$ and

$\Delta;\Phi_a[\theta_a] \models (\exists(\psi).(\Phi_1 \wedge \exists k_2, t_2 :: \mathbb{R}. \forall i :: S. \Phi_2 \wedge \Phi_c))[\theta_a]$ $(\star\star)$

where $\Phi_c = (t_1 + t_2 + c_{unp}) \doteq t \wedge (k_1 + k_2 + c_{unp}) \doteq k$.

By Theorem 58 using the first premise and $(\star)$, we get

$FIV(A_1, k_1, t_1, \Phi_1) \subseteq dom(\Delta, \psi; \psi_a)$ $(\diamond)$.

Using $(\star)$, $(\diamond)$ and the 4th premise, $(\star\star)$'s derivation must be in a form such that we have

a) $\Delta \rhd \theta : \psi$

b) $\Delta;\Phi_a[\theta_a] \models \Phi_1[\theta\,\theta_a]$

c) $i :: S, \Delta;\Phi_a[\theta_a] \models \Phi_2[\theta_a, \theta, k_2 \mapsto K_2, t_2 \mapsto T_2]$

d) $\Delta;\Phi_a[\theta_a] \models t_1[\theta\,\theta_a] + T_2 + c_{unp} \doteq t[\theta_a]$

e) $\Delta;\Phi_a[\theta_a] \models k[\theta_a] \doteq k_1[\theta\,\theta_a] + K_2 + c_{unp}$

By Theorem 59.2 on the first premise using $(\star)$, a) and b), we can show that

$$\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{t_1[\theta\,\theta_a]}_{k_1[\theta\,\theta_a]} |e_1| :^{c} \exists i{::}S.\, A_1[\theta\,\theta_a] \tag{1}$$

From $(\star)$ and $(\diamond)$, we can show that

f) $FIV(\Phi_a, A_1, \Omega, A_2, k_2, t_2) \subseteq i, k_2, t_2, dom(\Delta, \psi, \psi_a)$

By Theorem 59.1 on the second premise using c), f), $(\star)$ and $(\diamond)$, we obtain

$$i :: S, \Delta; \Phi_a[\theta_a]; x : A_1[\theta\,\theta_a], \Omega[\theta_a] \vdash^{T_2}_{K_2} |e_2| :^{c} A_2[\theta\,\theta_a] \tag{2}$$

Note that due to $(\star)$, we have $A_2[\theta\,\theta_a] = A_2[\theta_a]$. Then by the **c-unpack** rule using eqs. (1) and (2), we can show that
$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t_1[\theta\,\theta_a]+T_2+c_{unp}}_{k_1[\theta\,\theta_a]+K_1+c_{unp}}$ unpack $|e_1|$ with (x, i) in $|e_2| :^{c} A_2[\theta_a]$.
By $\sqsubseteq$ exec rule using (d) and (e), we obtain
$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t[\theta_a]}_{k[\theta_a]}$ unpack $|e_1|$ with (x, i) in $|e_2| :^{c} A_2[\theta_a]$.

Case:
$$\frac{\begin{array}{c}\Delta; \psi_a; C \wedge \Phi_a; \Omega \vdash e_1 \downarrow A, k, t \Rightarrow \Phi_1 \\ \Delta; \psi_a; \neg C \wedge \Phi_a; \Omega \vdash e_2 \downarrow A, k, t \Rightarrow \Phi_2 \qquad \Delta \vdash C\ \mathtt{wf} \\ \Phi = C \to \Phi_1 \wedge \neg C \to \Phi_2\end{array}}{\Delta; \psi_a; \Phi_a; \Omega \vdash \mathtt{split}\ (e_1, e_2)\ \textbf{with}\ C \downarrow A, k, t \Rightarrow \Phi}\ \textbf{alg-u-split-}\downarrow$$

TS: $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t[\theta_a]}_{k[\theta_a]} \mathtt{split}\ (|e_1|, |e_2|)$ with $C :^{c} A[\theta_a]$.
By the main assumptions, we have
$\mathrm{FIV}(\Phi_a, \Omega, A, k, t) \subseteq \mathrm{dom}(\Delta, \psi_a)$ $(\star)$ and
$\Delta; \Phi_a[\theta_a] \models (C \to \Phi_1 \wedge \neg C \to \Phi_2)[\theta_a]$ $(\star\star)$
Using $(\star)$ and the third premise, we can show that

a) $\mathrm{FIV}(C \wedge \Phi_a, \Omega, A, k, t) \subseteq \mathrm{dom}(\Delta, \psi_a)$.
b) $\mathrm{FIV}(\neg C \wedge \Phi_a, \Omega, A, k, t) \subseteq \mathrm{dom}(\Delta, \psi_a)$.

Using $(\star\star)$ and the third premise, we can show that

c) $\Delta; C \wedge \Phi_a[\theta_a] \models \Phi_1[\theta_a]$
d) $\Delta; \neg C \wedge \Phi_a[\theta_a] \models \Phi_2[\theta_a]$

By Theorem 59.1 on the first premise using $(\star)$ and c), we can show that

$$\Delta; C \wedge \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t[\theta_a]}_{k[\theta_a]} |e_1| :^{c} A[\theta_a]\{I[\theta_a]/i\} \tag{1}$$

By Theorem 59.1 on the second premise using $(\star)$ and d), we can show that

$$\Delta; \neg C \wedge \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t[\theta_a]}_{k[\theta_a]} |e_2| :^{c} A[\theta_a]\{I[\theta_a]/i\} \tag{2}$$

By the **c-split** rule using eqs. (1) and (2) and the third premise, we obtain
$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t[\theta_a]}_{k[\theta_a]} \texttt{split}\ (|e_1|, |e_2|)$ with $C :^{c} A[\theta_a]$.

Case: $$\frac{\Delta; \Phi \wedge C; \Omega \vdash e \downarrow A, k, t \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a; \Omega \vdash e \downarrow C \supset A, k, t \Rightarrow C \to \Phi}\ \textbf{alg-u-c-impI-}\downarrow$$

TS: $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t[\theta_a]}_{k[\theta_a]} |e| :^{c} C[\theta_a] \supset A[\theta_a]$.
By the main assumptions, we have
$\mathrm{FIV}(\Phi_a, \Omega, C \supset A, k, t) \subseteq \mathrm{dom}(\Delta, \psi_a)$ $(\star)$ and
$\Delta; \Phi_a[\theta_a] \models (C \to \Phi)[\theta_a]$ $(\star\star)$
Using $(\star)$, we can show that

a) $\mathrm{FIV}(C \wedge \Phi_a, \Omega, A, k, t) \subseteq \mathrm{dom}(\Delta, \psi_a)$.

By Theorem 59.1 on the premise using $(\star)$ and a), we can show that

$$\Delta; C[\theta_a] \wedge \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t[\theta_a]}_{k[\theta_a]} |e| :^{c} A[\theta_a] \tag{1}$$

By the **c-cimpI** rule using eq. (1), we obtain
$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t[\theta_a]}_{k[\theta_a]} |e| :^{c} C[\theta_a] \supset A[\theta_a]$.

Proof of Theorem 59.2:

Case: $$\frac{\Delta; \psi_a; \Phi_a; \Omega \vdash e \downarrow A, k, t \Rightarrow \Phi \qquad \Delta; \Phi_a \vdash^{A} A\ \texttt{wf} \qquad \textbf{FIV}(A, k, t) \in \Delta}{\Delta; \psi_a; \Phi_a; \Omega \vdash (e : A, k, t) \uparrow A \Rightarrow [\cdot], k, t, \Phi}\ \textbf{alg-u-anno-}\uparrow$$

TS: $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t[\theta_a]}_{k[\theta_a]} |(e : A, k, t)| :^{c} A[\theta_a]$.
Since by definition, $\forall e.\ |(e : \_, \_, \_)| = |e|$, STS:
$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t[\theta_a]}_{k[\theta_a]} |e| :^{c} A[\theta_a]$.
By the main assumptions, we have $\mathrm{FIV}(\Phi_a, \Omega) \subseteq \mathrm{dom}(\Delta, \psi_a)$ $(\star)$

and $\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$ $(\star\star)$
Using the third premise, we can show that

a) $\mathrm{FIV}(\Phi_a, \Omega, A, k, t) \subseteq \mathrm{dom}(\Delta, \psi_a)$.

By Theorem 59.1 on the first premise using $(\star\star)$ and a), we can conclude that
$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t[\theta_a]}_{k[\theta_a]} |e| :^{c} A[\theta_a]$.

Case: $\dfrac{\begin{array}{c}\Delta; \psi_a; \Phi_a; \Omega \vdash e_1 \uparrow A_1 \xrightarrow{\mathbf{exec}(k_e, t_e)} A_2 \Rightarrow [\psi], k_1, t_1, \Phi_1 \\ k_2, t_2 \in \mathbf{fresh}(\mathbb{R}) \qquad \Delta; k_2, t_2, \psi, \psi_a; \Phi_a; \Omega \vdash e_2 \downarrow A_1, k_2, t_2 \Rightarrow \Phi_2 \\ k = k_1 + k_2 + k_e + c_{app} \qquad t = t_1 + t_2 + t_e + c_{app}\end{array}}{\Delta; \psi_a; \Phi_a; \Omega \vdash e_1\ e_2 \uparrow A_2 \Rightarrow [k_2, t_2, \psi], k, t, \Phi_1 \wedge \Phi_2}$ **alg-u-app-↑**

TS: $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{(t_1+t_2+t_e+c_{app})[\theta_a,\theta_2]}_{(k_1+k_2+k_e+c_{app})[\theta_a,\theta_2]} |e_1|\ |e_2| :^{c} A_2[\theta_a, \theta_2]$.
By the main assumptions, we have $\mathrm{FIV}(\Phi_a, \Omega) \subseteq \mathrm{dom}(\Delta, \psi_a)$ $(\star)$
and
$\Delta; \Phi_a[\theta_a] \models (\Phi_1 \wedge \Phi_2)[\theta_a, \theta_2]$ $(\star\star)$ such that $\Delta \rhd \theta_2 : k_2, t_2, \psi$ $(\diamond)$
and $\Delta \rhd \theta_a : \psi_a$ are derivable.
By $(\diamond)$, we can show that $\theta_2 = k_2, t_2, \theta$ such that

a) $\theta(k_2) = K_2$ and $\theta(t_2) = T_2$ for some K_2 and T_2.
b) $\Delta \rhd \theta : \psi$

By Theorem 59.2 on the first premise using $(\star)$ and (b), we obtain

$$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t_1[\theta\,\theta_a]}_{k_1[\theta\,\theta_a]} |e_1| :^{c} A_1[\theta\,\theta_a] \xrightarrow{\mathrm{exec}(k_e[\theta\,\theta_a], t_e[\theta\,\theta_a])} A_2[\theta\,\theta_a] \quad (1)$$

By Theorem 58.2 on the first premise and $(\star)$, we get

c) $\mathrm{FIV}(A_1 \xrightarrow{\mathrm{exec}(k_e, t_e)} A_2, k_1, t_1, \Phi_1) \subseteq \mathrm{dom}(\Delta, \psi, \psi_a)$.

By $(\star)$ and c), we get

d) $\mathrm{FIV}(\Phi_a, \Omega, A_2, k_2, t_2) \subseteq k_2, t_2, \mathrm{dom}(\Delta, \psi, \psi_a)$.

By Theorem 59.2 on the third premise using (c), (d) and $(\star\star)$, we obtain

$$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{T_2}_{K_2} |e_2| :^{c} A_1[\theta\,\theta_a] \tag{2}$$

Then, by using **c-app** rule using eqs. (1) and (2), we can show that $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t_1[\theta\,\theta_a]+t_e[\theta\,\theta_a]+T_2}_{k_1[\theta\,\theta_a]+k_e[\theta\,\theta_a]+K_2} |e_1|\,|e_2| :^{c} A_2[\theta_a, \theta_2]$.
Note that we have $k_2[\theta_a, \theta_2] = K_2$ and $k_2[\theta_a, \theta_2] = K_2$.
Moreover, $t_1[\theta_a, \theta_2] = t_1[\theta\,\theta_a]$ and $k_1[\theta_a, \theta_2] = k_1[\theta\,\theta_a]$ (similarly for k_e and t_e) since k_2, t_2 are fresh variables.

Case: $\dfrac{\Delta; \psi_a; \Phi_a; \Omega \vdash e \uparrow \forall i \overset{exec(k_e,t_e)}{::} S.\, A' \Rightarrow [\psi], k, t, \Phi \qquad \Delta \vdash I :: S}{\Delta; \psi_a; \Phi_a; \Omega \vdash e\,[I] \uparrow A'\{I/i\} \Rightarrow [\psi], k + k_e[I/i], t + t_e[I/i], \Phi}$ **alg-u-iApp-↑**

TS: $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{(t+t_e[I/i])[\theta\,\theta_a]}_{(k+k_e[I/i])[\theta\,\theta_a]} |e|\,[I] :^{c} (A'\{I/i\})[\theta\,\theta_a]$.
By the main assumptions, we have

- $FIV(\Phi_a, \Omega) \subseteq dom(\Delta, \psi_a)$ $(\star)$ and
- $\Delta; \Phi_a[\theta_a] \models \Phi_2[\theta\,\theta_a]$ $(\star\star)$ such that the following are derivable
  - $\Delta \rhd \theta : \psi$ $(\diamond)$
  - $\Delta \rhd \theta_a : \psi_a$

By Theorem 59.2 on the first premise using $(\star)$ and $(\diamond)$, we obtain

$$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t[\theta\,\theta_a]}_{k[\theta\,\theta_a]} |e| :^{c} \forall i \overset{exec(k_e[\theta\,\theta_a],t_e[\theta\,\theta_a])}{::} S.\, A'[\theta\,\theta_a] \tag{1}$$

Then, by **c-iApp** rule using eq. (1) and the second premise, we can conclude that
$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{t[\theta\,\theta_a]+t_e[\theta\,\theta_a][I/i]}_{k[\theta\,\theta_a]+k_e[\theta\,\theta_a][I/i]} |e|\,[I] :^{c} A'[\theta\,\theta_a]\{I/i\}$.

Proof of Theorem 59.3:

Case: $\dfrac{t' \in \mathbf{fresh}(\mathbb{R}) \qquad \Delta; t', \psi_a; \Phi_a; \square\,\Gamma \vdash e \ominus e \downarrow \tau, t' \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a; \Gamma', \square\,\Gamma \vdash \mathtt{NC}\ e \ominus \mathtt{NC}\ e \downarrow \square\tau, t \Rightarrow 0 \doteq t \wedge (\exists t' :: \mathbb{R}.\Phi)}$ **alg-r-nochange-↓**

TS: $\Delta; \Phi_a[\theta_a]; \Gamma'[\theta_a], \Box\Gamma[\theta_a] \vdash \mathtt{NC}\ e \ominus \mathtt{NC}\ e \lesssim t[\theta_a] : \Box\tau[\theta_a]$.
By the main assumptions, we have

- $\mathrm{FIV}(\Phi_a, \Gamma', \Box\Gamma, \tau, t) \subseteq \mathrm{dom}(\Delta, \psi_a)$ $(\star)$
- $\Delta; \Phi_a[\theta_a] \models (0 \doteq t \wedge \exists t' :: \mathbb{R}.\Phi)[\theta_a]$ $(\star\star)$

Using $(\star)$ and the first premise, we can show that

a) $\mathrm{FIV}(\Phi_a, \Gamma, \tau, t') \subseteq t', \mathrm{dom}(\Delta, \psi_a)$.

Using $(\star)$, $(\star\star)$'s derivation must be in a form such that we have

b) $\Delta; \Phi_a[\theta_a] \models 0 \doteq t[\theta_a]$
c) $\Delta \vdash T' :: \mathbb{R}$ for some T'
d) $\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a, t' \mapsto T']$

By Theorem 59.3 on the premise using a), d) and $(\star)$, we can show that

$$\Delta; \Phi_a[\theta_a]; \Box\Gamma[\theta_a] \vdash |e| \ominus |e| \lesssim T' : \tau[\theta_a] \tag{1}$$

By the **c-nochange** rule using eq. (1), we obtain
$\Delta; \Phi_a[\theta_a]; \Gamma'[\theta_a], \Box\Gamma[\theta_a] \vdash \mathtt{NC}\ |e| \ominus \mathtt{NC}\ |e| \lesssim 0 : \Box\tau[\theta_a]$.
By the **c-r-$\sqsubseteq$** rule using this and b), we obtain
$\Delta; \Phi_a[\theta_a]; \Gamma'[\theta_a], \Box\Gamma[\theta_a] \vdash \mathtt{NC}\ e \ominus \mathtt{NC}\ e \lesssim t[\theta_a] : \Box\tau[\theta_a]$.

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Gamma \vdash e \ominus e' \uparrow \mathbf{list}[n]^{\alpha}\,\tau \Rightarrow [\psi], t_1, \Phi_e \\ t_2 \in \mathbf{fresh}(\mathbb{R}) \qquad \Delta; t_2, \psi, \psi_a; n \doteq 0 \wedge \Phi_a; \Gamma \vdash e_1 \ominus e_1' \downarrow \tau', t_2 \Rightarrow \Phi_1 \\ \Phi_a' = n \doteq i+1 \wedge \Phi_a \\ i, \Delta; t_2, \psi, \psi_a; \Phi_a'; h : \square\tau, tl : \mathbf{list}[i]^{\alpha}\,\tau, \Gamma \vdash e_2 \ominus e_2' \downarrow \tau', t_2 \Rightarrow \Phi_2 \\ \Phi_a'' = n \doteq i+1 \wedge \alpha \doteq \beta + 1 \wedge \Phi_a \\ i, \beta, \Delta; t_2, \psi, \psi_a; \Phi_a''; h : \tau, tl : \mathbf{list}[i]^{\beta}\,\tau, \Gamma \vdash e_3 \ominus e_3' \downarrow \tau', t_2 \Rightarrow \Phi_3 \\ \Phi_{cons} = \forall i :: \mathbb{N}.(n \doteq i+1) \to (\Phi_2 \wedge \forall \beta :: \mathbb{N}.(\alpha \doteq \beta + 1) \to \Phi_3) \\ \Phi = \exists(\psi).(\Phi_e \wedge \exists t_2 :: \mathbb{R}.((n \doteq 0 \to \Phi_1) \wedge \Phi_{cons} \wedge t_1 + t_2 \doteq t)) \end{array}}{\begin{array}{l}\Delta;\psi_a;\Phi_a;\Gamma \vdash \left|\begin{array}{l} \mathbf{case}\ e\ \mathbf{of\ nil} \ \to e_1 \\ \quad | \ h ::_{NC} tl \ \to e_2 \\ \quad | \ h ::_{C} tl \ \to e_3 \end{array}\right. \ominus \left|\begin{array}{l} \mathbf{case}\ e'\ \mathbf{of\ nil} \ \to e_1' \\ \quad | \ h ::_{NC} tl \ \to e_2' \\ \quad | \ h ::_{C} tl \ \to e_3' \end{array}\right. \downarrow \tau', t \Rightarrow \Phi \end{array}}\ \textbf{alg-r-caseL-}\downarrow$$

TS:

$$\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash \left|\begin{array}{l} \mathbf{case}\ e\ \mathbf{of\ nil} \ \to |e_1| \\ \quad | \ h ::_{N} tl \ \to |e_2| \\ \quad | \ h ::_{C} tl \ \to |e_3| \end{array}\right. \ominus \left|\begin{array}{l} \mathbf{case}\ e'\ \mathbf{of\ nil} \ \to |e_1'| \\ \quad | \ h ::_{N} tl \ \to |e_2'| \\ \quad | \ h ::_{C} tl \ \to |e_3'| \end{array}\right. \lesssim t[\theta_a] : \tau'[\theta_a]$$

By the main assumptions, we have

- $\mathrm{FIV}(\Phi_a, \Gamma, \tau', t) \subseteq \mathrm{dom}(\Delta, \psi_a)$ $(\star)$
- $\Delta;\Phi_a[\theta_a] \models (\exists(\psi).\Phi_e \wedge \exists t_2 :: \mathbb{R}.\Phi_{body})[\theta_a]$ $(\star\star)$

By Theorem 58 using the first premise and $(\star)$, we get
$\mathrm{FIV}(\mathbf{list}[n]^{\alpha}\,\tau, t_1, \Phi_e) \subseteq \mathrm{dom}(\Delta, \psi; \psi_a)$ $(\diamond)$.
Using $(\star)$ and $(\diamond)$, $(\star\star)$'s derivation must be in a form such that we have

a) $\Delta \ \triangleright \ \theta : \psi$
b) $\Delta;\Phi_a[\theta_a] \models \Phi_e[\theta\,\theta_a]$
c) $\Delta \vdash T_2 :: \mathbb{R}$
d) $\Delta; n[\theta\,\theta_a] \doteq 0 \wedge \Phi_a[\theta_a] \models \Phi_1[\theta\,\theta_a]$
e) $i :: S, \Delta; n[\theta\,\theta_a] \doteq i+1 \wedge \Phi_a[\theta_a] \models \Phi_2[\theta_a, \theta, t_2 \mapsto T_2]$
f) $i :: S, \beta :: S, \Delta; n[\theta\,\theta_a] \doteq i+1 \wedge \alpha[\theta\,\theta_a] \doteq \beta + 1 \wedge \Phi_a[\theta_a] \models \Phi_3[\theta_a, \theta, t_2 \mapsto T_2]$
g) $\Delta;\Phi_a[\theta_a] \models t_1[\theta\,\theta_a] + T_2 \doteq t[\theta_a]$

By Theorem 59.4 on the first premise using b) and $(\star)$, we can show that

$$\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |e| \ominus |e'| \lesssim t_1[\theta\ \theta_a] : \mathrm{list}[n[\theta\ \theta_a]]^{\alpha[\theta\ \theta_a]}\ \tau[\theta\ \theta_a] \quad (1)$$

By Theorem 59.3 on the second premise using d) and $(\star)$, we can show that

$$\Delta; n[\theta\ \theta_a] \doteq 0 \wedge \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |e_1| \ominus |e_1'| \lesssim T_2 : \tau'[\theta\ \theta_a] \quad (2)$$

By Theorem 59.3 on the third premise using e) and $(\star)$, we can show that

$$i :: S, \Delta; n[\theta\ \theta_a] \doteq i + 1 \wedge \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |e_2| \ominus |e_2'| \lesssim T_2 : \tau'[\theta\ \theta_a] \quad (3)$$

By Theorem 59.3 on the fourth premise using f) and $(\star)$, we can show that

$$i :: S, \beta :: S, \Delta; \Phi_a'; \Gamma[\theta_a] \vdash |e_3| \ominus |e_3'| \lesssim T_2 : \tau'[\theta\ \theta_a] \quad (4)$$

where $\Phi_a' = n[\theta\ \theta_a] \doteq i + 1 \wedge \alpha[\theta\ \theta_a] \doteq \beta + 1 \wedge \Phi_a[\theta_a]$.
Then by **c-r-caseL** rule using eqs. (1) to (4), we can show that

$$\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash \begin{array}{l} \text{case } e \text{ of nil } \to |e_1| \\ \quad | \ h ::_N tl \to |e_2| \\ \quad | \ h ::_C tl \to |e_3| \end{array} \ominus \begin{array}{l} \text{case } e' \text{ of nil } \to |e_1'| \\ \quad | \ h ::_N tl \to |e_2'| \\ \quad | \ h ::_C tl \to |e_3'| \end{array} \lesssim t_1[\theta\ \theta_a] + T_2 : \tau'[\theta_a]$$

We conclude by applying **c-r-$\sqsubseteq$** rule to this using g).

Case:
$$\frac{\begin{array}{c} t_1, t_2 \in \mathbf{fresh}(\mathbb{R}) \qquad i \in \mathbf{fresh}(\mathbb{N}) \\ \Delta; t_1, \psi_a; \Phi_a; \Gamma \vdash e_1 \ominus e_1' \downarrow \Box\tau, t_1 \Rightarrow \Phi_1 \\ \Delta; i, t_2, \psi_a; \Phi_a; \Gamma \vdash e_2 \ominus e_2' \downarrow \mathbf{list}[i]^{\alpha}\,\tau, t_2 \Rightarrow \Phi_2 \\ \Phi_2' = \Phi_2 \wedge n \doteq (i+1) \wedge t_1 + t_2 \doteq t \\ \Phi = \exists t_1 :: \mathbb{R}.(\Phi_1 \wedge \exists t_2 :: \mathbb{R}.\exists i :: \mathbb{N}.\Phi_2') \end{array}}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \mathbf{cons}_{NC}(e_1, e_2) \ominus \mathbf{cons}_{NC}(e_1', e_2') \downarrow \mathbf{list}[n]^{\alpha}\,\tau, t \Rightarrow \Phi}\ \textbf{alg-r-consNC-}\downarrow$$

TS:
$\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash \mathrm{cons}_{NC}(|e_1|, |e_2|) \ominus \mathrm{cons}_{NC}(|e_1'|, |e_2'|) \lesssim t[\theta_a] : \mathrm{list}[n[\theta_a]]^{\alpha[\theta_a]}\,\tau[\theta_a]$.
By the main assumptions, we have
$\mathrm{FIV}(\Phi_a, \Gamma, \mathrm{list}[n]^{\alpha}\,\tau, t) \subseteq \mathrm{dom}(\Delta, \psi_a)$ $(\star)$ and
$\Delta; \Phi_a[\theta_a] \models (\exists t_1 :: \mathbb{R}.\Phi_1 \wedge \exists t_2 :: \mathbb{R}.\exists i :: \mathbb{N}.\Phi_2')[\theta_a]$ $(\star\star)$
Using $(\star)$, $(\star\star)$'s derivation must be in a form such that we have

a) $\Delta \vdash T_1 :: \mathbb{R}$
b) $\Delta \vdash T_2 :: \mathbb{R}$
c) $\Delta; \Phi_a[\theta_a] \models \Phi_1[\theta_a, t_1 \mapsto T_1]$
d) $\Delta \vdash I :: \mathbb{N}$
e) $\Delta; \Phi_a[\theta_a] \models \Phi_2[\theta_a, t_2 \mapsto T_2, i \mapsto I]$
f) $\Delta; \Phi_a[\theta_a] \models (I+1) \doteq n[\theta_a]$
g) $\Delta; \Phi_a[\theta_a] \models (T_1 + T_2) \doteq t[\theta_a]$

By Theorem 59.3 on the third premise using $(\star)$ and c), we can show that

$$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash |e_1| \ominus |e_1'| \lesssim T_1 : \Box\tau[\theta_a] \tag{1}$$

By Theorem 59.3 on the fourth premise using $(\star)$ and e), we can show that

$$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash |e_2| \ominus |e_2'| \lesssim T_2 : \mathrm{list}[I]^{\alpha[\theta_a]}\,\tau[\theta_a] \tag{2}$$

By **c-r-cons1** typing rule using eqs. (1) and (2), we obtain
$\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash \mathrm{cons}_{NC}(|e_1|, |e_2|) \ominus \mathrm{cons}_{NC}(|e_1'|, |e_2'|) \lesssim T_1 + T_2 : \mathrm{list}[I+1]^{\alpha[\theta_a]}\,\tau[\theta_a]$.
We conclude by applying **c-r-$\sqsubseteq$** rule to this using f) and g).

Proof of Theorem 59.4:

Case: $$\frac{\Delta; \psi_a; \Phi_a; \Gamma \vdash e \ominus e' \downarrow \tau, t \Rightarrow \Phi \qquad \Delta; \Phi_a \vdash \tau \ \mathtt{wf} \qquad \Delta \vdash t :: \mathbb{R}}{\Delta; \psi_a; \Phi_a; \Gamma \vdash (e : \tau, t) \ominus (e' : \tau, t) \uparrow \tau \Rightarrow [\cdot], t, \Phi}\ \textbf{alg-r-anno-}\uparrow$$

TS: $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash |(e : \tau, t)| \ominus |(e' : \tau, t)| \lesssim t[\theta_a] : \tau[\theta_a]$.

Since by definition, $\forall e.\ |(e : \_, \_)| = |e|$, STS:

$\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |e| \ominus |e'| \lesssim t[\theta_a] : \tau[\theta_a]$.

By the main assumptions, we have $\mathrm{FIV}(\Phi_a, \Gamma) \subseteq \mathrm{dom}(\Delta, \psi_a)$ $(\star)$ and $\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$ $(\star\star)$

Using the third premise, we can show that

a) $\mathrm{FIV}(\Phi_a, \Gamma, \tau, k, t) \subseteq \mathrm{dom}(\Delta, \psi_a)$.

By Theorem 59.4 on the first premise using $(\star\star)$ and a), we can conclude that

$\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |e| \ominus |e'| \lesssim t[\theta_a] : \tau[\theta_a]$.

Case: $$\frac{\Delta; \psi_a; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \uparrow \Box\tau \Rightarrow [\psi], t, \Phi}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \mathtt{der}\ e_1 \ominus \mathtt{der}\ e_2 \uparrow \tau \Rightarrow [\psi], t, \Phi}\ \textbf{alg-r-der-}\uparrow$$

TS: $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{|e|}_{t[\theta\,\theta_a]} |e'| :^{c} \tau[\theta\,\theta_a]$.

By the main assumptions, we have $\mathrm{FIV}(\Phi_a, \Gamma) \subseteq \mathrm{dom}(\Delta, \psi_a)$ $(\star)$ and $\Delta; \Phi_a[\theta_a] \models \Phi[\theta\,\theta_a]$ $(\star\star)$ such that $\Delta \rhd \theta : \psi$ $(\diamond)$ and $\Delta \rhd \theta_a : \psi_a$ are derivable.

By Theorem 59.4 on the first premise using $(\star)$ and $(\diamond)$, we obtain

$$\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |e| \ominus |e'| \lesssim t[\theta\,\theta_a] : \Box\tau[\theta\,\theta_a] \tag{1}$$

Then, by **c-der** rule using eq. (1) and the second premise, we can conclude that

$\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |e| \ominus |e'| \lesssim t[\theta\,\theta_a] : \tau[\theta\,\theta_a]$.

□

Theorem 60 (Completeness of the Algorithmic Typechecking)**.** *We have the following.*

1. *Assume that* $\Delta; \Phi_a; \Omega \vdash^{t}_{k} e :^{c} A$. *Then,* $\exists e'$ *such that*
   a) $\Delta; \cdot; \Phi_a; \Omega \vdash e' \downarrow A, k, t \Rightarrow \Phi$
   b) $\Delta; \Phi_a \models \Phi$
   c) $|e'| = e$
2. *Assume that* $\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t :^{c} \tau$. *Then,* $\exists e'_1, e'_2$ *such that*
   a) $\Delta; \cdot; \Phi_a; \Gamma \vdash e'_1 \ominus e'_2 \downarrow \tau, t \Rightarrow \Phi$
   b) $\Delta; \Phi_a \models \Phi$
   c) $|e'_1| = e_1$ *and* $|e'_2| = e_2$

Proof. Proof is by simultaneous induction on the RelCost Core typing derivations.

Proof of Theorem 60.1:

Case: $\dfrac{\Omega(x) = A}{\Delta; \Phi_a; \Omega \vdash^{0}_{0} x :^{c} A}$ **c-var**

We can conclude as follows

$$\dfrac{\dfrac{\Omega(x) = A}{\Delta; \cdot; \Phi_a; \Omega \vdash x \uparrow A \Rightarrow [.], 0, 0, \top}\ \textbf{alg-u-var-}\uparrow \qquad \Delta; \Phi_a \models^{A} A \sqsubseteq A \Rightarrow \Phi \ \text{ by lemma 50}}{\Delta; \psi_a; \Phi_a; \Omega \vdash x \downarrow A, 0, 0 \Rightarrow \top}\ \textbf{alg-r-}\uparrow\downarrow$$

Case: $\dfrac{\Delta; \Phi_a; \Omega \vdash^{t_1}_{k_1} e_1 :^{c} A \qquad \Delta; \Phi_a; \Omega \vdash^{t_2}_{k_2} e_2 :^{c} \textbf{list}[n]\, A}{\Delta; \Phi_a; \Omega \vdash^{t_1+t_2}_{k_1+k_2} \textbf{cons}_C(e_1, e_2) :^{c} \textbf{list}[n+1]\, A}$ **c-cons**

By Theorem 60.2 on the first premise, $\exists e'_1$ such that

a) $\Delta; \cdot; \Phi_a; \Omega \vdash e'_1 \downarrow A, k_1, t_1 \Rightarrow \Phi_1$
b) $\Delta; \Phi_a \models \Phi_1$
c) $|e'_1| = e_1$

By a), we can show that for $k_1', t_1' \in \text{fresh}(\mathbb{R})$ where
$\Phi_1' = \Phi_1 \wedge k_1 \doteq k_1' \wedge t_1 \doteq t_1'$

$$\Delta; k_1', t_1'; \Phi_a; \Omega \vdash e_1' \downarrow A, k_1', t_1' \Rightarrow \Phi_1' \tag{1}$$

By Theorem 60.2 on the second premise, $\exists e_2'$ such that

d) $\Delta; \cdot; \Phi_a; \Omega \vdash e_2' \downarrow \text{list}[n]\, A, k_2, t_2 \Rightarrow \Phi_2$
e) $\Delta; \Phi_a \models \Phi_2$
f) $|e_2'| = e_2$

By a), we can show that for $i, k_2', t_2' \in \text{fresh}(\mathbb{R})$ where
$\Phi_2' = \Phi_2 \wedge k_2 \doteq k_2' \wedge t_2 \doteq t_2' \wedge i \doteq n$

$$\Delta; i, k_2', t_2'; \Phi_a; \Omega \vdash e_2' \downarrow \text{list}[i]\, A, k_2', t_2' \Rightarrow \Phi_2' \tag{2}$$

Then, we can conclude as follows

1.
$$\frac{\begin{array}{c} k_1', t_1', k_2', t_2' \in \text{fresh}(\mathbb{R}) \qquad i \in \text{fresh}(\mathbb{N}) \\ \Delta; k_1', t_1', \psi_a; \Phi_a; \Omega \vdash e_1' \downarrow A, k_1', t_1' \Rightarrow \Phi_1' \;\; \text{eq. (1)} \\ \Delta; i, k_2', t_2', \psi_a; \Phi_a; \Omega \vdash e_2' \downarrow \text{list}[i]\, A, k_2', t_2' \Rightarrow \Phi_2' \;\; \text{eq. (2)} \\ \Phi_2'' = (\Phi_2' \wedge n + 1 \doteq (i+1) \wedge k_1 + k_2 \doteq k_1' + k_2' \wedge t_1' + t_2' \doteq t_1 + t_2) \\ \Phi = \exists k_1', t_1' :: \mathbb{R}.(\Phi_1' \wedge \exists k_2', t_2' :: \mathbb{R}.\exists i :: \mathbb{N}.\Phi_2'') \end{array}}{\Delta; \psi_a; \Phi_a; \Omega \vdash \text{cons}_C(e_1', e_2') \downarrow \text{list}[n+1]\, A, k_1 + k_2, t_1 + t_2 \Rightarrow \Phi}\ \textbf{alg-u-cons-}\downarrow$$

2. Using b) and e) for the substitutions $k_i' = k_i$ and $t_i' = t_i$ for the fresh costs and $i = n$ for the size of the tail.
3. Using c) and f), $|\text{cons}_C(e_1', e_2')| = \text{cons}_C(e_1, e_2)$

Proof of Theorem 60.2:

Case: $\dfrac{\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t :^{c} \square\tau}{\Delta; \Phi_a; \Gamma \vdash \texttt{der}\ e_1 \ominus \texttt{der}\ e_2 \lesssim t :^{c} \tau}$ **c-der**

By Theorem 60.2 on the premise, $\exists e_1', e_2'$ such that

a) $\Delta; \cdot; \Phi_a; \Gamma \vdash e_1' \ominus e_2' \downarrow \tau, t \Rightarrow \Phi$

b) $\Delta; \Phi_a \models \Phi$

c) $|e_1'| = e_1$ and $|e_2'| = e_2$

Then, we can conclude by using a), b) and c) as follows:

$$\frac{\Delta; \cdot; \Phi_a; \Gamma \vdash e_1' \ominus e_2' \downarrow \Box\tau, t \Rightarrow \Phi}{\Delta; \cdot; \Phi_a; \Gamma \vdash \mathtt{der}\ e_1' \ominus \mathtt{der}\ e_2' \downarrow \tau, t \Rightarrow \Phi}\ \textbf{alg-r-der-}\downarrow \text{ and}$$

1.

$$\frac{\dfrac{\dfrac{\Delta; \cdot; \Phi_a; \Gamma \vdash e_1' \ominus e_2' \downarrow \Box\tau, t \Rightarrow \Phi}{\Delta; \cdot; \Phi_a; \Gamma \vdash \mathtt{der}\ e_1' \ominus \mathtt{der}\ e_2' \downarrow \tau, t \Rightarrow \Phi}\ \textbf{alg-r-der-}\downarrow}{\Delta; \cdot; \Phi_a; \Gamma \vdash (\mathtt{der}\ e_1' : \tau, t) \ominus (\mathtt{der}\ e_2' : \tau, t) \uparrow \tau \Rightarrow [\cdot], t, \Phi}\ \textbf{alg-r-anno-}\uparrow \qquad \Delta; \Phi_a \models \tau \equiv \tau \Rightarrow \Phi' \text{ by Lemma 49}}{\Delta; \cdot; \Phi_a; \Gamma \vdash (\mathtt{der}\ e_1' : \tau, t) \ominus (\mathtt{der}\ e_2' : \tau, t) \downarrow \tau, t \Rightarrow \Phi \wedge \Phi' \wedge t \leqslant t}\ \textbf{alg-r-}\uparrow\downarrow$$

2. By c), $|(\mathtt{der}\ e_i' : \tau, t)| = \mathtt{der}\ |e_i'|$.
3. By b) and Lemma 49.

Case: $\dfrac{\Delta; \Phi_a; |\Gamma|_1 \vdash_{k_1}^{t_1} e_1 :^{c} A_1 \qquad \Delta; \Phi_a; |\Gamma|_2 \vdash_{k_2}^{t_2} e_2 :^{c} A_2}{\Delta; \Phi_a; \Gamma \vdash \mathtt{switch}\ e_1 \ominus \mathtt{switch}\ e_2 \lesssim t_1 - k_2 :^{c} U\,(A_1, A_2)}\ \textbf{c-switch}$

By Theorem 60.1 on the first premise , $\exists e_1'$ such that

a) $\Delta; \cdot; \Phi_a; |\Gamma|_1 \vdash e_1' \downarrow A_1, k_1, t_1 \Rightarrow \Phi_1$

b) $\Delta; \Phi_a \models \Phi_1$

c) $|e_1'| = e_1$

By a), we can show that for $k_1', t_1' \in \text{fresh}(\mathbb{R})$ where
$\Phi_1' = \Phi_1 \wedge k_1 \doteq k_1' \wedge t_1 \doteq t_1'$

$$\Delta; k_1', t_1'; \Phi_a; |\Gamma|_1 \vdash e_1' \downarrow A_1, k_1', t_1' \Rightarrow \Phi_1' \tag{1}$$

By Theorem 60.1 on the second premise , $\exists e_2'$ such that

d) $\Delta; \cdot; \Phi_a; |\Gamma|_2 \vdash e_2' \downarrow A_2, k_2, t_2 \Rightarrow \Phi_2$

e) $\Delta; \Phi_a \models \Phi_2$

f) $|e_2'| = e_2$

By d), we can show that for $k_2', t_2' \in \text{fresh}(\mathbb{R})$ where $\Phi_2' = \Phi_2 \wedge k_2 \doteq k_2' \wedge t_2 \doteq t_2'$

$$\Delta; k_2', t_2'; \Phi_a; |\Gamma|_2 \vdash e_2' \downarrow A_2, k_2', t_2' \Rightarrow \Phi_2' \tag{2}$$

Then, we can conclude as follows

1. By using eqs. (1) and (2):
$$\frac{\begin{array}{c} k_1', t_1', k_2', t_2' \in \text{fresh}(\mathbb{R}) \\ \Delta; k_1', t_1', \psi_a; \Phi; |\Gamma|_1 \vdash e_1' \downarrow A_1, k_1', t_1' \Rightarrow \Phi_1' \\ \Delta; k_2', t_2', \psi_a; \Phi; |\Gamma|_2 \vdash e_2' \downarrow A_2, k_2', t_2' \Rightarrow \Phi_2' \\ \exists k_1, t_1 :: \mathbb{R}.(\Phi_1' \wedge \exists k_2', t_2' :: \mathbb{R}.\Phi_2' \wedge t_1' - k_2' \doteq t) \end{array}}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \texttt{switch}\ e_1' \ominus \texttt{switch}\ e_2' \downarrow t, \mathsf{U}(A_1, A_2) \Rightarrow \Phi}\ \textbf{alg-r-switch-}\downarrow.$$
2. By using b) and e) and the substitutions $k_i' = k_i$ and $t_i' = t_i$ for the fresh costs where $t = t_1 - k_2$.
3. By c) and f), we get $|\texttt{switch}\ e_i'| = \texttt{switch}\ |e_i|$

Case: $$\frac{\Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t :^{\mathbf{c}} \tau \qquad \Delta; \Phi_a \models \tau \equiv \tau' \qquad \Delta; \Phi_a \models t \leqslant t'}{\Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t' :^{\mathbf{c}} \tau'}\ \textbf{c-r-}\equiv$$

By Theorem 60.2 on the first premise, $\exists e_1', e_2'$ such that

a) $\Delta; \cdot; \Phi_a; \Gamma \vdash e_1' \ominus e_2' \downarrow \tau, t \Rightarrow \Phi_1$
b) $\Delta; \Phi_a \models \Phi_1$
c) $|e_1'| = e$ and $|e_2'| = e'$

By Theorem 55 on the second premise,

d) $\Delta; \Phi_a \models \tau \equiv \tau' \Rightarrow \Phi_2$
e) $\Delta; \Phi_a \models \Phi_2$.

Then, we can conclude as follows

1. By using a) and d)

$$\dfrac{\dfrac{\Delta;\cdot;\Phi_a;\Gamma \vdash e_1' \ominus e_2' \downarrow \tau, t \Rightarrow \Phi_1}{\Delta;\cdot;\Phi_a;\Gamma \vdash (e_1' : \tau, t) \ominus (e_2' : \tau, t) \uparrow \tau \Rightarrow [\cdot], t, \Phi_1}\ \textbf{alg-r-anno-}\uparrow \qquad \Delta;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi_2}{\Delta;\cdot;\Phi_a;\Gamma \vdash (e_1' : \tau, t) \ominus (e_2' : \tau, t) \downarrow \tau', t' \Rightarrow \Phi_1 \wedge \Phi_2 \wedge t \leqslant t'}\ \textbf{alg-r-}\uparrow\downarrow$$

2. By using b), e) and the third premise, we can show that $\Delta;\Phi_a \models \Phi_1 \wedge \Phi_2 \wedge t \leqslant t'$
3. By c), $|(e_1' : \tau, t)| = e$ and $(e_2' : \tau, t) = e'$

Case: $\dfrac{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_1' \lesssim t_1 :^{c} \tau_1 \xrightarrow{\mathbf{diff}(t)} \tau_2 \qquad \Delta;\Phi_a;\Gamma \vdash e_2 \ominus e_2' \lesssim t_2 :^{c} \tau_1}{\Delta;\Phi_a;\Gamma \vdash e_1\ e_2 \ominus e_1'\ e_2' \lesssim t_1 + t_2 + t :^{c} \tau_2}\ \textbf{c-r-app}$

By Theorem 60.2 on the first premise, $\exists \overline{e}_1, \overline{e}_1'$ such that

a) $\Delta;\cdot;\Phi_a;\Gamma \vdash \overline{e}_1 \ominus \overline{e}_1' \downarrow \tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2, t_1 \Rightarrow \Phi_1$
b) $\Delta;\Phi_a \models \Phi_1$
c) $|\overline{e}_1| = e_1$ and $|\overline{e}_1'| = e_1'$

By Theorem 60.2 on the second premise, $\exists \overline{e}_2, \overline{e}_2'$ such that

d) $\Delta;\cdot;\Phi_a;\Gamma \vdash \overline{e}_2 \ominus \overline{e}_2' \downarrow \tau_1, t_2 \Rightarrow \Phi_2$
e) $\Delta;\Phi_a \models \Phi_2$
f) $|\overline{e}_2| = e_2$ and $|\overline{e}_2'| = e_2'$

By d), we can show that for $t_2' \in \mathrm{fresh}(\mathbb{R})$ where $\Phi_2' = \Phi_2 \wedge t_2 \doteq t_2'$

$$\Delta; t_2'; \Phi_a; \Gamma \vdash \overline{e}_2 \ominus \overline{e}_2' \downarrow \tau_1, t_2' \Rightarrow \Phi_2' \tag{1}$$

Then, we can conclude as follows

1.

$$\dfrac{\dfrac{\dfrac{\Delta;\cdot;\Phi_a;\Gamma \vdash \overline{e}_1 \ominus \overline{e}_1' \downarrow \tau_1 \xrightarrow{\text{diff}(t)} \tau_2, t_1 \Rightarrow \Phi_1}{\Delta;\cdot;\Phi_a;\Gamma \vdash E_1 \ominus E_1' \uparrow \tau_1 \xrightarrow{\text{diff}(t)} \tau_2 \Rightarrow [\cdot], t_1, \Phi_1}\ \textbf{alg-r-anno-}\uparrow \quad t_2' \in \text{fresh}(\mathbb{R}) \qquad \Delta; t_2'; \Phi_a; \Gamma \vdash \overline{e}_2 \ominus \overline{e}_2' \downarrow \tau_1, t_2' \Rightarrow \Phi_2'}{\Delta;\cdot;\Phi_a;\Gamma \vdash E_{1,2} \ominus E_{1,2}' \uparrow \tau_2 \Rightarrow [t_2'], t_1 + t + t_2', \Phi_1 \wedge \Phi_2'}\ \textbf{c-r-app}}{\Phi = \exists t_2' :: \mathbb{R}.\Phi_1 \wedge \Phi_2' \wedge t_1 + t + t_2' \leqslant t_1 + t + t_2 \qquad \Delta;\cdot;\Phi_a;\Gamma \vdash E_{1,2} \ominus E_{1,2}' \downarrow \tau_2, t_1 + t + t_2 \Rightarrow \Phi}\ \textbf{alg-r-}\uparrow\downarrow$$

where $E_{1,2} = (\overline{e}_1 : \tau_1 \xrightarrow{\text{diff}(t)} \tau_2, t_1)\ \overline{e}_2$ and
$E_1 = (\overline{e}_1 : \tau_1 \xrightarrow{\text{diff}(t)} \tau_2, t_1)$ and
$E_1' = (\overline{e}_1' : \tau_1 \xrightarrow{\text{diff}(t)} \tau_2, t_1)$ and $E_{1,2}' = (\overline{e}_1' : \tau_1 \xrightarrow{\text{diff}(t)} \tau_2, t_1)\ \overline{e}_2'$.

2. By using b) and e) and the substitution $t_2' = t_2$ for the fresh cost.
3. Using c) and f), we get
   $|(\overline{e}_1 : \tau_1 \xrightarrow{\text{diff}(t)} \tau_2, t_1)\ \overline{e}_2| = e_1\ e_2$ and
   $|(\overline{e}_1' : \tau_1 \xrightarrow{\text{diff}(t)} \tau_2, t_1)\ \overline{e}_2'| = e_1'\ e_2'$.

Case:

$$\dfrac{\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_1' \lesssim t_1 :^{c} \exists i::S.\tau_1 \qquad i :: S, \Delta;\Phi_a;x:\tau_1,\Gamma \vdash e_2 \ominus e_2' \lesssim t_2 :^{c} \tau_2 \qquad i \notin FV(\Phi_a;\Gamma,\tau_2,t_2)}{\Delta;\Phi_a;\Gamma \vdash \textbf{unpack}\ e_1\ \textbf{as}\ (x,i)\ \textbf{in}\ e_2 \ominus \textbf{unpack}\ e_1'\ \textbf{as}\ (x,i)\ \textbf{in}\ e_2' \lesssim t_1 + t_2 :^{c} \tau_2}\ \textbf{c-r-unpack}$$

By Theorem 60.2 on the first premise, $\exists \overline{e}_1, \overline{e}_1'$ such that

a) $\Delta;\cdot;\Phi_a;\Gamma \vdash \overline{e}_1 \ominus \overline{e}_1' \downarrow \exists i::S.\tau_1, t_1 \Rightarrow \Phi_1$
b) $\Delta;\Phi_a \models \Phi_1$
c) $|\overline{e}_1| = e_1$ and $|\overline{e}_1'| = e_1'$

By Theorem 60.2 on the second premise, $\exists \overline{e}_2, \overline{e}_2'$ such that

d) $i :: S, \Delta;\cdot;\Phi_a;x:\tau_1,\Gamma \vdash \overline{e}_2 \ominus \overline{e}_2' \downarrow \tau_2, t_2 \Rightarrow \Phi_2$
e) $i :: S, \Delta;\Phi_a \models \Phi_2$
f) $|\overline{e}_2| = e_2$ and $|\overline{e}_2'| = e_2'$

By d), we can show that for $t_2' \in \text{fresh}(\mathbb{R})$ where $\Phi_2' = \Phi_2 \wedge t_2 \doteq t_2'$

$$i :: S, \Delta; t_2'; \Phi_a; x : \tau_1, \Gamma \vdash \overline{e}_2 \ominus \overline{e}_2' \downarrow \tau_2, t_2' \Rightarrow \Phi_2' \tag{1}$$

Then, we can conclude as follows

1.
$$\dfrac{\dfrac{\Delta; \cdot; \Phi_a; \Gamma \vdash \overline{e}_1 \ominus \overline{e}_1' \downarrow \exists i::S.\, \tau_1, t_1 \Rightarrow \Phi_1 \quad (a)}{\Delta; \cdot; \Phi_a; \Gamma \vdash E_1 \ominus E_1' \uparrow \exists i::S.\, \tau_1 \Rightarrow [\cdot], t_1, \Phi_1}\ \textbf{alg-r-anno-}\uparrow \quad i :: S, \Delta; t_2'; \Phi_a; x : \tau_1, \Gamma \vdash \overline{e}_2 \ominus \overline{e}_2' \downarrow \tau_2, t_2' \Rightarrow \Phi_2' \quad (\text{eq. } (1)) \quad \Phi' = \exists t_2' :: \mathbb{R}.\Phi_1 \wedge \Phi_2' \wedge t_1 + t_2' \doteq t_1 + t_2}{\Delta; \cdot; \Phi_a; \Gamma \vdash \text{unpack } E_1 \text{ as } (x, i) \text{ in } \overline{e}_2 \ominus \text{unpack } E_1' \text{ as } (x, i) \text{ in } \overline{e}_2' \downarrow \tau_2, t_1 + t_2 \Rightarrow \Phi'}$$

   where $E_1 = (\overline{e}_1 : \exists i::S.\, \tau_1, t_1,)$ and $E_2 = (\overline{e}_1' : \exists i::S.\, \tau_1, t_1,)$
2. By using b) and e) and the substitution $t_2' = t_2$ for the fresh cost.
3. Using c) and f), we get
   $|\text{unpack } (\overline{e}_1 : \exists i::S.\, \tau_1, t_1) \text{ as } (x, i) \text{ in } \overline{e}_2| = \text{unpack } e_1 \text{ as } (x, i) \text{ in } e_2$
   and
   $|\text{unpack } (\overline{e}_1' : \exists i::S.\, \tau_1, t_1) \text{ as } (x, i) \text{ in } \overline{e}_2'| = \text{unpack } e_1' \text{ as } (x, i) \text{ in } e_2'$.

□

D

APPENDIX FOR CASE STUDIES

In this chapter, we present additional material for our implementation and case studies. Appendix D.1 presents the lemmas necessary to type two of our examples: `msort` and `bfold`. Finally, Appendix D presents all the examples.

D.1 SOME ARITHMETIC PROPERTIES FOR DIVIDE AND CONQUER PROGRAMS

We prove some arithmetic properties of summations and the log function that are needed to type divide and conquer examples.

Lemma 61. *For* $n > 1$, $\lceil \log_2(n) \rceil = 1 + \lceil \log_2(\lceil \frac{n}{2} \rceil) \rceil$.

Proof. We split cases on the parity of n.

Case: n even.

Let $n = 2k$. Then, $k = \lceil \frac{n}{2} \rceil$ and

$$\begin{aligned} \lceil \log_2(n) \rceil &= \lceil \log_2(2k) \rceil \\ &= \lceil 1 + \log_2(k) \rceil \\ &= 1 + \lceil \log_2(k) \rceil \\ &= 1 + \lceil \log_2(\lceil \frac{n}{2} \rceil) \rceil \end{aligned}$$

Case: n odd.

Then, $\lceil \frac{n}{2} \rceil = \frac{n+1}{2}$.

Let $k \geqslant 1$ be such that $2^{k-1} < \frac{n+1}{2} \leqslant 2^k$ (note that $n \geqslant 3$, so such a k exists).

Then, $2^k - 1 < n \leqslant 2^{k+1} - 1$.

Since n is odd, this forces $2^k + 1 \leqslant n \leqslant 2^{k+1} - 1$.

Hence, $k < \log_2(n) < k + 1$, so $\lceil \log_2(n) \rceil = k + 1$.

Clearly, $\lceil \log_2(\frac{n+1}{2}) \rceil = k$.

Hence, $\lceil \log_2(n) \rceil = k + 1 = \lceil \log_2(\frac{n+1}{2}) \rceil + 1 = \lceil \log_2(\lceil \frac{n}{2} \rceil) \rceil + 1$. □

Lemma 62. *Let* f *be a monotonic function and* $n > 1$, $\alpha > 0$ *and* $\alpha_1 + \alpha_2 = \alpha$. *Then,*

$$\left\lceil \sum_{i=0}^{\lceil \log_2(\lceil \frac{n}{2} \rceil)\rceil} f(2^i) \cdot \min\left(\alpha_1, 2^{\lceil \log_2(\lceil \frac{n}{2} \rceil)\rceil - i}\right) \right\rceil + \left\lceil \sum_{i=0}^{\lceil \log_2(\lfloor \frac{n}{2} \rfloor)\rceil} f(2^i) \cdot \min\left(\alpha_2, 2^{\lceil \log_2(\lfloor \frac{n}{2} \rfloor)\rceil - i}\right) \right\rceil + f(n)$$

$$\leqslant \sum_{i=0}^{\lceil \log_2(n)\rceil} f(2^i) \cdot \min\left(\alpha, 2^{\lceil \log_2(n)\rceil - i}\right)$$

Proof. We prove the inequality through a series of transformations. In each step, we highlight the changed subexpressions in red.

$$\left\lceil \sum_{i=0}^{\lceil \log_2(\lceil \frac{n}{2} \rceil)\rceil} f(2^i) \cdot \min\left(\alpha_1, 2^{\lceil \log_2(\lceil \frac{n}{2} \rceil)\rceil - i}\right) \right\rceil + \left\lceil \sum_{i=0}^{\lceil \log_2(\lfloor \frac{n}{2} \rfloor)\rceil} f(2^i) \cdot \min\left(\alpha_2, 2^{\lceil \log_2(\lfloor \frac{n}{2} \rfloor)\rceil - i}\right) \right\rceil + f(n)$$

$$\leqslant \left\lceil \sum_{i=0}^{\lceil \log_2(\lceil \frac{n}{2} \rceil)\rceil} f(2^i) \cdot \min\left(\alpha_1, 2^{\lceil \log_2(\lceil \frac{n}{2} \rceil)\rceil - i}\right) \right\rceil + \left\lceil \sum_{i=0}^{\lceil \log_2(\lceil \frac{n}{2} \rceil)\rceil} f(2^i) \cdot \min\left(\alpha_2, 2^{\lceil \log_2(\lceil \frac{n}{2} \rceil)\rceil - i}\right) \right\rceil + f(n)$$

$$= \left\lceil \sum_{i=0}^{\lceil \log_2(\lceil \frac{n}{2} \rceil)\rceil} f(2^i) \cdot \left[\min\left(\alpha_1, 2^{\lceil \log_2(\lceil \frac{n}{2} \rceil)\rceil - i}\right) + \min\left(\alpha_2, 2^{\lceil \log_2(\lceil \frac{n}{2} \rceil)\rceil - i}\right)\right] \right\rceil + f(n)$$

$$\leqslant \left\lceil \sum_{i=0}^{\lceil \log_2(\lceil \frac{n}{2} \rceil)\rceil} f(2^i) \cdot \min\left(\alpha_1 + \alpha_2, 2 \cdot 2^{\lceil \log_2(\lceil \frac{n}{2} \rceil)\rceil - i}\right) \right\rceil + f(n)$$

(Using the inequality: $\mathtt{min}(a,c)+\mathtt{min}(b,c) \leqslant \mathtt{min}(a+b,2c)$)

$$= \left\lceil \sum_{i=0}^{\lceil \log_2(\lceil \frac{n}{2} \rceil)\rceil} f(2^i) \cdot \mathtt{min}\left(\alpha, 2^{1+\lceil \log_2(\lceil \frac{n}{2} \rceil)\rceil - i}\right) \right\rceil + f(n)$$

$$= \left\lceil \sum_{i=0}^{\lceil \log_2(\lceil \frac{n}{2} \rceil)\rceil} f(2^i) \cdot \mathtt{min}\left(\alpha, 2^{\lceil \log_2(n)\rceil - i}\right) \right\rceil + f(n)$$

(by Lemma 61)

$$\leqslant \left\lceil \sum_{i=0}^{\lceil \log_2(\lceil \frac{n}{2} \rceil)\rceil} f(2^i) \cdot \mathtt{min}\left(\alpha, 2^{\lceil \log_2(n)\rceil - i}\right) \right\rceil + f(2^{\lceil \log_2(n)\rceil})$$

($n \leqslant 2^{\lceil \log_2(n)\rceil}$ and f is monotone)

$$= \left\lceil \sum_{i=0}^{\lceil \log_2(\lceil \frac{n}{2} \rceil)\rceil} f(2^i) \cdot \mathtt{min}\left(\alpha, 2^{\lceil \log_2(n)\rceil - i}\right) \right\rceil + f(2^{\lceil \log_2(n)\rceil}) \cdot \mathtt{min}(\alpha, 1)$$

(because $\alpha > 0$, $\mathtt{min}(\alpha,1) = 1$)

$$= \left\lceil \sum_{i=0}^{\lceil \log_2(\lceil \frac{n}{2} \rceil)\rceil} f(2^i) \cdot \mathtt{min}\left(\alpha, 2^{\lceil \log_2(n)\rceil - i}\right) \right\rceil + f(2^{\lceil \log_2(n)\rceil}) \cdot \mathtt{min}\left(\alpha, 2^{(\lceil \log_2(n)\rceil - \lceil \log_2(n)\rceil)}\right)$$

$$= \sum_{i=0}^{\lceil \log_2(n)\rceil} f(2^i) \cdot \mathtt{min}\left(\alpha, 2^{\lceil \log_2(n)\rceil - i}\right)$$

(by Lemma 61, $\lceil \log_2(n)\rceil = 1 + \lceil \log_2(\lceil \frac{n}{2} \rceil)\rceil$)

□

Lemma 63 (Balanced fold complexity). *Let* $P(n,\alpha) = \sum_{i=0}^{\lceil \log_2(n)\rceil} h(2^i) \cdot \mathtt{min}(\alpha, 2^{\lceil \log_2(n)\rceil - i})$.
Then $P(n,\alpha) \in O(\kappa \cdot (\alpha + \alpha \cdot \log_2(n/\alpha)))$.
Specifically, for constant κ, $P(n,\alpha,\kappa) \in O(\alpha + \alpha \cdot \log_2(n/\alpha))$.

Proof. We proceed by splitting cases on i in the summation in $P(n, \alpha)$.

Case: $i > \lceil \log_2(n) \rceil - \lceil \log_2(\alpha) \rceil$

Then, $\lceil \log_2(n) \rceil - i < \lceil \log_2(\alpha) \rceil$.
Hence, $\lceil \log_2(n) \rceil - i \leqslant \lfloor \log_2(\alpha) \rfloor \leqslant \log_2(\alpha)$.
So, $2^{\lceil \log_2(n) \rceil - i} \leqslant \alpha$.
Therefore, $\min(\alpha, 2^{\lceil \log_2(n) \rceil - i}) = 2^{\lceil \log_2(n) \rceil - i}$.

Case: $i \leqslant \lceil \log_2(n) \rceil - \lceil \log_2(\alpha) \rceil$
Then $\lceil \log_2(n) \rceil - i \geqslant \lceil \log_2(\alpha) \rceil$ and $2^{\lceil \log_2(n) \rceil - i} \geqslant \alpha$.
Therefore, $\min(\alpha, 2^{\lceil \log_2(n) \rceil - i}) = \alpha$.

It follows that

$$
\begin{aligned}
P(n, \alpha) &= \sum_{i=0}^{\lceil \log_2(n) \rceil} \kappa \cdot \min(\alpha, 2^{\lceil \log_2(n) \rceil - i}) \\
&= \sum_{i=0}^{\lceil \log_2(n) \rceil - \lceil \log_2(\alpha) \rceil} \kappa \cdot \min(\alpha, 2^{\lceil \log_2(n) \rceil - i}) + \sum_{i=\lceil \log_2(n) \rceil - \lceil \log_2(\alpha) \rceil + 1}^{\lceil \log_2(n) \rceil} \kappa \cdot \min(\alpha, 2^{\lceil \log_2(n) \rceil - i}) \\
&= \kappa \cdot \alpha \cdot (\lceil \log_2(n) \rceil - \lceil \log_2(\alpha) \rceil + 1) + \kappa \cdot (2^{\lceil \log_2(\alpha) \rceil - 1} + \ldots + 2^0) \\
&= \kappa \cdot \alpha \cdot (\lceil \log_2(n) \rceil - \lceil \log_2(\alpha) \rceil + 1) + \kappa \cdot (2^{\lceil \log_2(\alpha) \rceil} - 1) \\
&\in O(\kappa \cdot \alpha \cdot \log_2(n/\alpha)) + O(\kappa \cdot \alpha) \\
&= O(\kappa \cdot (\alpha + \alpha \cdot \log_2(n/\alpha)))
\end{aligned}
$$

□

Lemma 64 (Mergesort complexity)**.** *Assume that* h *is a linear, monotonic function.*
Let $Q(n, \alpha) = \sum_{i=0}^{\lceil \log_2(n) \rceil} h(2^i) \cdot \min(\alpha, 2^{\lceil \log_2(n) \rceil - i})$
Then, $Q(n, \alpha) \in O(n \cdot (1 + \log_2(\alpha)))$.

Proof. We proceed by splitting cases on "i" in the summation in $Q(n, \alpha)$.

Case: $i > \lceil \log_2(n) \rceil - \lceil \log_2(\alpha) \rceil$
Then, $\lceil \log_2(n) \rceil - i < \lceil \log_2(\alpha) \rceil$ and, hence,

$\lceil \log_2(n) \rceil - i \leqslant \lfloor \log_2(\alpha) \rfloor \leqslant \log_2(\alpha)$.
So, $2^{\lceil \log_2(n) \rceil - i} \leqslant \alpha$.
Therefore, $\min(\alpha, 2^{\lceil \log_2(n) \rceil - i}) = 2^{\lceil \log_2(n) \rceil - i}$.

Case: $i \leqslant \lceil \log_2(n) \rceil - \lceil \log_2(\alpha) \rceil$
Then $\lceil \log_2(n) \rceil - i \geqslant \lceil \log_2(\alpha) \rceil$ and $2^{\lceil \log_2(n) \rceil - i} \geqslant \alpha$.
Therefore, $\min(\alpha, 2^{\lceil \log_2(n) \rceil - i}) = \alpha$.

It follows that

$$
\begin{aligned}
Q(n, \alpha) &= \sum_{i=0}^{\lceil \log_2(n) \rceil} h(2^i) \cdot \min(\alpha, 2^{\lceil \log_2(n) \rceil - i}) \\
&= \sum_{i=0}^{\lceil \log_2(n) \rceil - \lceil \log_2(\alpha) \rceil} h(2^i) \cdot \min(\alpha, 2^{\lceil \log_2(n) \rceil - i}) + \sum_{i=\lceil \log_2(n) \rceil - \lceil \log_2(\alpha) \rceil + 1}^{\lceil \log_2(n) \rceil} h(2^i) \cdot \min(\alpha, 2^{\lceil \log_2(n) \rceil - i}) \\
&= \sum_{i=0}^{\lceil \log_2(n) \rceil - \lceil \log_2(\alpha) \rceil} h(2^i) \cdot \alpha + \sum_{i=\lceil \log_2(n) \rceil - \lceil \log_2(\alpha) \rceil + 1}^{\lceil \log_2(n) \rceil} h(2^i) \cdot 2^{\lceil \log_2(n) \rceil - i} \\
&\quad \text{(since h is linear, i.e. } h(x) = a \cdot x + b\text{)} \\
&= \sum_{i=0}^{\lceil \log_2(n) \rceil - \lceil \log_2(\alpha) \rceil} (\alpha \cdot a \cdot 2^i + \alpha \cdot b) + \sum_{i=\lceil \log_2(n) \rceil - \lceil \log_2(\alpha) \rceil + 1}^{\lceil \log_2(n) \rceil} a \cdot 2^i \cdot 2^{\lceil \log_2(n) \rceil - i} + b \cdot 2^{\lceil \log_2(n) \rceil - i} \\
&\quad \text{(since h is linear, i.e. } h(x) = a \cdot x + b\text{)} \\
&= \alpha \cdot a \cdot (2^0 + \ldots + 2^{\lceil \log_2(n) \rceil - \lceil \log_2(\alpha) \rceil}) + \alpha \cdot b \cdot (\lceil \log_2(n) \rceil - \lceil \log_2(\alpha) \rceil) + \\
&\quad a \cdot 2^{\lceil \log_2(n) \rceil} \cdot \lceil \log_2(\alpha) \rceil + b \cdot (2^{\lceil \log_2(\alpha) \rceil - 1} + \ldots + 2^0) \\
&= \alpha \cdot a \cdot (2^{\lceil \log_2(n) \rceil - \lceil \log_2(\alpha) \rceil + 1} - 1) + \alpha \cdot b \cdot (\lceil \log_2(n) \rceil - \lceil \log_2(\alpha) \rceil) + \\
&\quad a \cdot 2^{\lceil \log_2(n) \rceil} \cdot \lceil \log_2(\alpha) \rceil + b \cdot (2^{\lceil \log_2(\alpha) \rceil} - 1) \\
&\in O(n) + O(n \cdot \log_2(\alpha)) \\
&= O(n \cdot (1 + \log_2(\alpha)))
\end{aligned}
$$

□

D.2 EXAMPLE PROGRAMS

In this section, we present a list of example programs that we have type-checked with BiRelCost.

D.2.1 *List operations*

LIST APPEND We describe how the standard list append program can be typed in BiRelCost.

$$
\begin{array}{l}
\text{fix } \texttt{append}(\_).\Lambda.\Lambda.\Lambda.\Lambda.\lambda l_1.\lambda l_2. \\
\quad \text{case } l_1 \text{ of nil } \to l_2 \\
\qquad \mid h :: tl \to \text{cons}(h, \texttt{append}\ ()\ []\ []\ []\ []\ tl\ l_2)
\end{array}
$$

$$
\vdash \texttt{append} \ominus \texttt{append} \lesssim 0 : \ \text{unit}_r \to \forall i, j, \alpha, \beta. \\
\text{list}[i]^{\alpha}\,\tau \to \text{list}[j]^{\beta}\,\tau \xrightarrow{\text{diff}(0)} \text{list}[i+j]^{\alpha+\beta}\,\tau
$$

LIST FILTER We describe how the standard filter program can be typed in BiRelCost.

$$
\begin{array}{l}
\Lambda.\text{fix } \texttt{filter}(f).\Lambda.\Lambda.\lambda l. \\
\quad \text{case } l \text{ of nil } \to \text{pack nil} \\
\qquad \mid h :: tl \to \text{let } r = \texttt{filter}\ f\ []\ []\ tl \text{ in} \\
\qquad\qquad \text{let } b = f\ h \text{ in} \\
\qquad\qquad \text{unpack } r \text{ as } r' \text{ in} \\
\qquad\qquad \text{if } f \text{ then pack cons}(h, r') \text{ else pack } r'
\end{array}
$$

$$
\vdash \texttt{filter} \ominus \texttt{filter} \lesssim 0 : \ \forall t.(\square\,(U\,\text{int} \xrightarrow{\text{diff}(t)} U\,\text{bool})) \to \forall n, \alpha. \\
\text{list}[n]^{\alpha}\,U\,\text{int} \xrightarrow{\text{diff}(t\cdot\alpha)} U\,(\exists j.\text{list}[j]\,\text{int})
$$

LIST ZIP We describe how the standard list zip function can be typed in BiRelCost.

$$
\begin{array}{l}
\text{fix } \texttt{zip}(\_).\Lambda.\Lambda.\Lambda.\lambda l_1.\texttt{case } l_1 \texttt{ of} \\
\quad \text{nil } \to \text{nil} \\
\mid h_1 :: tl_1 \ \to \texttt{case } l_2 \texttt{ of} \\
\qquad\qquad \text{nil } \to \texttt{contra} \\
\qquad\quad \mid h_2 :: tl_2 \ \to \text{let } r = \texttt{zip}\ ()[]\ []\ []\ tl_1\ tl_2 \text{ in} \\
\qquad\qquad\qquad\qquad \text{cons}(\langle h_1, h_2\rangle, r)
\end{array}
$$

$$
\vdash \texttt{zip} \ominus \texttt{zip} \lesssim 0 : \ \text{unit}_r \to \forall n, \alpha, \beta. \\
\text{list}[n]^{\alpha}\,\tau_1 \to \text{list}[n]^{\beta}\,\tau_2 \xrightarrow{\text{diff}(0)} \text{list}[n]^{\min(n,\alpha+\beta)}\,\tau
$$

LIST REV We describe how the standard list reverse program can be typed in BiRelCost.

$$\begin{array}{l}\text{fix } \texttt{rev}(\_).\Lambda.\Lambda.\Lambda.\Lambda.\lambda l.\lambda acc.\\ \quad \text{case } l \text{ of nil } \to acc\\ \qquad \mid h :: tl \to \texttt{rev}\ ()\ []\ []\ []\ []\ \ tl\ \text{cons}(h, acc)\end{array}$$

$$\vdash \texttt{rev} \ominus \texttt{rev} \lesssim 0 : \ \text{unit}_r \to \forall i, j, \alpha, \beta. \\ \text{list}[i]^{\alpha}\,\tau \to \text{list}[j]^{\beta}\,\tau \xrightarrow{\text{diff}(0)} \text{list}[i+j]^{\alpha\beta}\,\tau$$

LIST SHUFFLE We describe how following program that shuffles the elements of a list can be typed in BiRelCost. The program shuffles a list by reversing its tail at each recursive call.

$$\begin{array}{l}\text{fix } \texttt{shuffle}(\_).\Lambda.\Lambda.\lambda l.\\ \quad \text{case } l \text{ of nil } \to \text{nil}\\ \qquad \mid h :: tl \to \text{cons}(h, \texttt{shuffle}\ ()[]\ []\ (\texttt{rev}\ ()[]\ []\ \ []\ \ tl\ \text{nil}\ ))\end{array}$$

$$\vdash \texttt{shuffle} \ominus \texttt{shuffle} \lesssim 0 : \ \text{unit}_r \to \forall n, \alpha. \\ \text{list}[n]^{\alpha}\,U\,\text{int} \xrightarrow{\text{diff}(0)} \text{list}[n]^{\alpha}\,U\,\text{int}$$

LIST FOLD (COMPARISON) We describe how following program that compares the relative cost of the standard `foldr` and `foldl` functions can be typed in BiRelCost.

$$\begin{array}{l}\text{fix } \texttt{foldr}(f).\Lambda.\Lambda.\lambda l.\lambda acc\\ \quad \text{case } l \text{ of nil } \to acc\\ \qquad \mid h :: tl \to \text{let } r = \texttt{foldr}\ f[]\ []\ tl\ acc\ \text{in } f\ h\ r\end{array}$$

$$\begin{array}{l}\text{fix } \texttt{foldl}(f).\Lambda.\Lambda.\lambda l.\lambda acc\\ \quad \text{case } l \text{ of nil } \to acc\\ \qquad \mid h :: tl \to \texttt{foldl}\ f[]\ []\ tl\ (f\ h\ acc)\end{array}$$

$$\vdash \texttt{foldr} \ominus \texttt{foldl} \lesssim 0 : \ \forall t.(U\,(\text{int} \to \text{bool} \xrightarrow{\text{exec}(t,t)} \text{bool})) \to \forall n, \alpha. \\ \text{list}[n]^{\alpha}\,U\,\text{int} \xrightarrow{\text{diff}(0)} U\,\text{bool}$$

LIST FLATTEN We describe how the standard list flatten program can be typed in BiRelCost. Assume that we know the type of the append function as derived above.

$$
\begin{array}{l}
\text{fix } \texttt{flatten}(\_).\Lambda.\Lambda.\Lambda.\Lambda.\lambda M. \\
\text{case } M \text{ of nil } \to \text{nil} \\
\qquad \mid l :: M' \to \text{let } r = \texttt{flatten}\ ()\ []\ []\ []\ []\ M' \text{ in} \\
\qquad\qquad \texttt{append}\ []\ []\ []\ []\ l\ r
\end{array}
$$

$$
\vdash \texttt{flatten} \ominus \texttt{flatten} \lesssim 0 : \text{unit}_r \to \forall i, j, \alpha, \beta. \\
\text{list}[i]^{\alpha}\,\tau \to \text{list}[j]^{\beta}\,\tau \xrightarrow{\text{diff}(0)} \text{list}[i \cdot j]^{\alpha \cdot \beta}\,\tau
$$

D.2.2 *Example programs from RelCost*

We have already all but one example in the thesis.

LOOP UNSWITCHING Next, we consider a compiler optimization technique known as *loop unswitching* that moves a conditional inside a loop to the outside. For simplicity, we consider a variant in which the else branch just returns a unit. Consider the function `loop` that iterates over a list `l`.

$$
\begin{array}{l}
\text{fix } \texttt{loop}(l).\texttt{case } l \texttt{ of} \\
\quad \text{nil } \to () \\
\mid h :: tl \ \to \ \text{if } b \text{ then } \boxed{\text{let } \_ = f\ h \text{ in } \texttt{loop}\ tl} \text{ else } ().
\end{array}
$$

This program can be transformed to a version that pulls out the conditional from the loop body as follows:

$$
\begin{array}{l}
\texttt{loopOp} = \text{if } b \text{ then} \\
\qquad\qquad \text{fix } \texttt{loop}'(l).\texttt{case } l \texttt{ of} \\
\qquad\qquad\quad \text{nil } \to () \\
\qquad\qquad \mid h :: tl \ \to \ \boxed{\text{let } \_ = f\ h \text{ in } \texttt{loop}'\ tl} \\
\text{else } \lambda l.()
\end{array}
$$

Suppose that the list `l` has type $\text{list}[n]^{0}\,\text{int}_r$, i.e. it is substituted by the same list for two programs, and the function `f` has type $\text{int}_r \xrightarrow{\mathbb{CP}(0)} \text{int}_r$, i.e. given related integers, it returns related integers with 0 execution cost difference. Assuming that the boolean input `b` is equal between two runs, what can we say about the relative cost of these two programs? Intuitively, `loopOp` is an optimization: rather than checking `b` at each iteration, it only checks it once outside of the function definition. Here, we do a more fine-grained

cost counting and assume all elimination forms to have a unit cost. Then, intuitively one would expect that the execution cost difference between these two programs is n.

If we resort to the non-relational execution cost analysis, using the switch rule we have introduced in Example 1 from the paper, we can establish the following type

$$\vdash \lambda b.\texttt{loop} \ominus \lambda b.\texttt{loop0p} \lesssim 0 : \quad U\,((\mathsf{bool} \to \forall n{::}\mathbb{N}.\ \mathsf{list}[n]\,\mathsf{int} \xrightarrow{\mathsf{exec}(5\cdot n+1,1)} \mathsf{unit}), .)$$

by typing the two programs independently. Then, via subtyping $U\,(A_1 \xrightarrow{\mathsf{exec}(k,t)} A_2) \subseteq U\,A_1 \xrightarrow{\mathbb{CP}(t-k)} U\,A_2$, we can establish a relative execution cost difference of $5 \cdot n$ for these two functions. However, this bound is not precise enough: it is 5 times more than what we expected, because it completely ignores the fact that b doesn't change between the two programs.

Instead, we can obtain a more precise bound using relational analysis. To achieve this, we make use of asynchronous rules that allows us to compare programs with different structure. For instance, we can compare an arbitrary expression e to an if statement as follows:

$$\frac{|\Gamma|_2 \vdash^{t}_{k} e' : \mathsf{bool} \qquad \Gamma \vdash e \ominus e'_1 \lesssim t' : \tau \qquad \Gamma \vdash e \ominus e'_2 \lesssim t' : \tau}{\Gamma \vdash e \ominus (\mathsf{if}\ e'\ \mathsf{then}\ e'_1\ \mathsf{else}\ e'_2) \lesssim t' - k - 1 : \tau}\ \textbf{e-if}$$

In this rule we relate e to the branches of the conditional and separately establish lower and upper bounds on the execution cost of the guard of the conditional. This rule allows us to compare `loop` to the inner recursive function `loop'` in `loop0p`. Similarly, using a symmetric variant of **e-if** rule, we can compare the inner conditional branch of `loop` to the body of `loop'` (shown in shaded boxes above). Note that, in the latter, we want to avoid comparing the "`else`" branch $()$ to let $\_ = f\ h$ in `loop' tl`. This can be taken care of by refining the boolean type with its value as follows: $\mathsf{bool}_r[B]$. [1] Then, we can type these two programs with a more precise relative cost n

$$\vdash \lambda b.\texttt{loop} \ominus \lambda b.\texttt{loop0p} \lesssim 0 : \ \forall B :: \{\mathsf{true}, \mathsf{false}\}.\ \mathsf{bool}[B] \xrightarrow{\mathbb{CP}(-1)} \forall n{::}\mathbb{N}.\ \mathsf{list}[n]^0\,\mathsf{int}_r \xrightarrow{\mathbb{CP}(n)} \mathsf{unit}_r.$$

1 Although we do not consider indexed booleans in this paper, they can be easily simulated by lists.

The negative cost 1 comes from the fact that the optimized version incurs a unit cost for the outer "`if`" statement and the expected cost n comes from the fact that the conditional elimination incurs a unit cost for each recursive call.

D.2.3 *Additional examples*

SELECTION SORT Consider the standard selection sort algorithm that finds the smallest element in a list and then sorts the remaining list recursively. In RelCost, we can show that `ssort` is a constant time algorithm, i.e. its relative cost is 0.

We briefly explain its typing. The first ingredient is the function `select` that takes an element x and a list of size n and returns the minimum among $x :: l$ and the rest of the list.

$$
\begin{array}{l}
\texttt{fix select}(x).\lambda l.\texttt{case l of} \\
\quad \texttt{nil} \ \rightarrow \langle x, \texttt{nil} \rangle \\
\mid h :: tl \ \rightarrow \texttt{let } (small, big) = \texttt{if } x < h \texttt{ then } \langle x, h \rangle \texttt{ else } \langle h, x \rangle \\
\qquad\qquad \texttt{let } (smaller, rest) = \texttt{select } small \ tl \texttt{ in} \\
\qquad\qquad \langle smaller, \texttt{cons}(big, rest) \rangle
\end{array}
$$

It can be given the following relational type:

$$
\begin{array}{l}
\vdash \texttt{select} \ominus \texttt{select} \lesssim 0 : \ U \, \texttt{int} \xrightarrow{\mathbb{CP}(0)} \forall n, \alpha{::}\mathbb{N}. \\
\qquad\qquad \texttt{list}[n]^{\alpha} \, U \, \texttt{int} \xrightarrow{\mathbb{CP}(0)} \exists \beta{::}\mathbb{N}. \, (U \, \texttt{int} \times \texttt{list}[n]^{\beta} \, U \, \texttt{int})
\end{array}
$$

The selection sort function `ssort` first finds the minimum element and the rest of the list members and then returns the minimum element appended to the rest of the sorted list.

$$
\begin{array}{l}
\texttt{fix ssort}(l).\texttt{case l of} \\
\quad \texttt{nil} \ \rightarrow \texttt{nil} \\
\mid h :: tl \ \rightarrow \texttt{let } (smallest, rest) = \texttt{select } h \ tl \texttt{ in} \\
\qquad\qquad \texttt{cons}(smallest, \texttt{ssort } rest)
\end{array}
$$

Then, we can relationally show that `ssort` has zero relative cost with respect to two lists that differ by α elements.

$$
\begin{array}{l}
\vdash \texttt{ssort} \ominus \texttt{ssort} \lesssim 0 : \ \texttt{unit}_r \xrightarrow{\mathbb{CP}(0)} \forall n, \alpha{::}\mathbb{N}. \\
\qquad\qquad\qquad\qquad \texttt{list}[n]^{\alpha} \, U \, \texttt{int} \xrightarrow{\mathbb{CP}(0)} \exists \beta{::}\mathbb{N}. \, .\texttt{list}[n]^{\beta} \, U \, \texttt{int}.
\end{array}
$$

We briefly explain how we derived this type. We focus on the part where the list has at least one element. From the type above, we know that `select`'s relative cost is 0 and "cons"'ing is constant time. In addition, we assumed that recursively, `ssort` incurs 0 cost. Hence we can conclude that relative cost of `ssort` is 0.

D.2.4 *Approximate sum*

The next example is from the approximate computing domain in which one often runs an approximate version of the program to save resources. For instance, consider two implementations of a calculation that computes the mean of a list of numbers. The first function computes the sum of a list of numbers and divides the sum by the length of the list whereas the second function (its approximate version) only computes the sum of the half of the elements, divides this sum by the total length of the list and then doubles the result afterwards. The first version could be operating over precise numbers whereas the second–approximate–version could be operating over approximate numbers. How can we type these two implementations in RelCost?

We first show the two helper functions `sum` and `sumAppr` that correspond to precise and approximate summation over a list of numbers.

```
fix sum(acc).λl.case l of
  nil  → acc
| h :: tl  → case tl of
               nil  → h + acc
             | h′ :: tl′  →  sum (h + h′+ acc ) tl′
```

```
fix sumAppr(acc).λl.case l of
  nil  → acc
| h :: tl  → case tl of
               nil  → h + acc
             | h′ :: tl′  →  sum ( h′+ acc ) tl′
```

Assume that addition and division operations are constant time and the helper function `length` can be given the following type

$$\vdash \texttt{length} \ominus \texttt{length} \lesssim 0 : \forall n{::}\mathbb{N}.\, \mathrm{list}[n]^{\alpha}\, \mathrm{U}\, \mathrm{int} \xrightarrow{\mathrm{diff}(0)} \mathrm{int}_r$$

Then, we can show that the two helper functions `sum` and `sumAppr` can be given the following relational type with relative cost n.

$$\vdash \texttt{sum} \ominus \texttt{sumAppr} \lesssim 0 : \mathrm{U}\,\mathrm{int} \xrightarrow{\mathrm{diff}(0)} \forall n{::}\mathbb{N}.\, \mathrm{list}[n]^{\alpha}\,\mathrm{U}\,\mathrm{int} \xrightarrow{\mathrm{diff}(n)} \mathrm{U}\,\mathrm{int}\ .$$

Intuitively, these two functions only differ by an addition operation for each recursive call, therefore we obtain n difference cost in their execution time: for each recursive call (which goes down in size by 2), a unit cost for the addition and a unit cost for the primitive application.

Then we can type these two functions as follows:

$$\vdash \left(\lambda \mathtt{l}.\ \frac{\texttt{sum 0 l}}{\texttt{length l}}\right) \ominus \left(\lambda \mathtt{l}.\ 2 \cdot \frac{\texttt{sumAppr 0 l}}{\texttt{length l}}\right) \lesssim 0 : \\ \forall n{::}\mathbb{N}.\, \mathrm{list}[n]^{\alpha}\,\mathrm{U}\,\mathrm{int} \xrightarrow{\mathrm{diff}(n-2)} \mathrm{U}\,\mathrm{int}.$$

Since the approximate version performs an additional multiplication operation, we use the symmetric version of the rule **r-let-e** and subtract two unit costs: one for the cost of the multiplication and one for the cost of the application of the primitive application.

BIBLIOGRAPHY

[1] *A Refinement Type by Any Other Name*. 2015. URL: http://www.weaselhat.com/2015/03/16/a-refinement-type-by-any-other-name/.

[2] Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Riecke. "A Core Calculus of Dependency." In: *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '99. New York, NY, USA: ACM, 1999, pp. 147–160.

[3] Umut A. Acar. "Self-Adjusting Computation." PhD thesis. Department of Computer Science, Carnegie Mellon University, 2005.

[4] Umut A. Acar, Amal Ahmed, and Matthias Blume. "Imperative Self-adjusting Computation." In: *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '08. San Francisco, California, USA: ACM, 2008.

[5] Umut A. Acar, Guy E. Blelloch, and Robert Harper. "Adaptive Functional Programming." In: *ACM Trans. Program. Lang. Syst.* 28.6 (2006).

[6] Umut A. Acar, Guy E. Blelloch, Matthias Blume, Robert Harper, and Kanat Tangwongsan. "An Experimental Analysis of Self-adjusting Computation." In: *ACM Trans. Program. Lang. Syst.* 32.1 (2009), 3:1–3:53.

[7] Amal Ahmed. "Step-Indexed Syntactic Logical Relations for Recursive and Quantified Types." In: *Proceedings of the 15th European Conference on Programming Languages and Systems*. ESOP'06. Vienna, Austria, 2006, pp. 69–83.

[8] Marco Gaboardi Pierre-Yves Strub Alejandro Aguirre Gilles Barthe and Deepak Garg. "A Relational Logic for Higher-Order Programs." In: *Proceedings of the 22nd International Conference on Functional Programming*. ICFP '17. Oxford, UK, 2017.

[9] Arthur Azevedo de Amorim, Marco Gaboardi, Emilio Jesús Gallego Arias, and Justin Hsu. "Really Natural Linear Indexed Type Checking." In: *Proceedings of the 26th 2014 International Symposium on Implementation and Application of Functional Languages, IFL '14, Boston, MA, USA, October 1-3, 2014*. 2014, 5:1–5:12.

[10] Torben Amtoft, Sruthi Bandhakavi, and Anindya Banerjee. "A logic for information flow in object-oriented programs." In: *Proceedings of the 33th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'06*. Ed. by J. Gregory Morrisett and Simon L. Peyton Jones. 2006, pp. 91–102.

[11] Andrew W. Appel and David A. McAllester. "An indexed model of recursive types for foundational proof-carrying code." In: *ACM Trans. Program. Lang. Syst.* 23.5 (2001), pp. 657–683.

[12] Lennart Augustsson. "Cayenne&Mdash;a Language with Dependent Types." In: *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming*. ICFP '98. New York, NY, USA: ACM, 1998, pp. 239–250.

[13] Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. "Probabilistic Relational Reasoning for Differential Privacy." In: *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '12. ACM, 2012.

[14] Gilles Barthe, Cédric Fournet, Benjamin Grégoire, Pierre-Yves Strub, Nikhil Swamy, and Santiago Zanella Béguelin. "Probabilistic relational verification for cryptographic implementations." In: *Proceedings of the 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'14*. Ed. by Suresh Jagannathan and Peter Sewell. 2014, pp. 193–206.

[15] Gilles Barthe, Marco Gaboardi, Emilio Jesús Gallego Arias, Justin Hsu, Aaron Roth, and Pierre-Yves Strub. "Higher-Order Approximate Relational Refinement Types for Mechanism Design and Differential Privacy." In: *Proceedings of the 42nd Annual ACM SIGPLAN-*

SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015. 2015, pp. 55–68.

[16] Gilles Barthe, Marco Gaboardi, Emilio Jesús Gallego Arias, Justin Hsu, Aaron Roth, and Pierre-Yves Strub. "Higher-order approximate relational refinement types for mechanism design and differential privacy." In: *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*. Ed. by Sriram K. Rajamani and David Walker. 2015, pp. 55–68.

[17] Nick Benton. "Simple Relational Correctness Proofs for Static Analyses and Program Transformations." In: *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '04. New York, NY, USA: ACM, 2004, pp. 14–25.

[18] Nick Benton. "Simple relational correctness proofs for static analyses and program transformations." In: *Proceedings of the 31th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'04*. Ed. by Neil D. Jones and Xavier Leroy. 2004, pp. 14–25.

[19] Yves Bertot and Pierre Castran. *Interactive Theorem Proving and Program Development: Coq'Art The Calculus of Inductive Constructions*. Springer Publishing Company, Incorporated, 2010.

[20] François Bobot, Sylvain Conchon, E Contejean, Mohamed Iguernelala, Stéphane Lescuyer, and Alain Mebsout. *The Alt-Ergo automated theorem prover, 2008*. 2013.

[21] Guillaume Bonfante, Jean-Yves Marion, and Jean-Yves Moyen. "Quasi-interpretations a way to control resources." In: *Theor. Comput. Sci.* 412.25 (2011), pp. 2776–2796.

[22] Ana Bove and Peter Dybjer. "Language Engineering and Rigorous Software Development." In: Springer-Verlag, 2009. Chap. Dependent Types at Work, pp. 57–99.

[23] Edwin C. Brady. "Idris: General Purpose Programming with Dependent Types." In: *Proceedings of the 7th Workshop on Programming Languages Meets Program Verification*. PLPV '13. 2013.

[24] Val Breazu-Tannen, Thierry Coquand, Carl A. Gunter, and Andre Scedrov. "Inheritance As Implicit Coercion." In: *Inf. Comput.* 93.1 (July 1991), pp. 172–221.

[25] Jacob Burnim and Koushik Sen. "Asserting and checking determinism for multithreaded programs." In: *Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2009, Amsterdam, The Netherlands, August 24-28, 2009*. Ed. by Hans van Vliet and Valérie Issarny. 2009, pp. 3–12.

[26] Yufei Cai, Paolo G. Giarrusso, Tillmann Rendel, and Klaus Ostermann. "A Theory of Changes for Higher-order Languages: Incrementalizing Λ-calculi by Static Differentiation." In: *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '14. Edinburgh, United Kingdom, 2014, pp. 145–155.

[27] Michael Carbin, Sasa Misailovic, and Martin C. Rinard. "Verifying quantitative reliability for programs that execute on unreliable hardware." In: *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, OOPSLA 2013, part of SPLASH 2013, Indianapolis, IN, USA, October 26-31, 2013*. Ed. by Antony L. Hosking, Patrick Th. Eugster, and Cristina V. Lopes. 2013, pp. 33–52.

[28] Michael Carbin, Deokhwan Kim, Sasa Misailovic, and Martin C. Rinard. "Proving acceptability properties of relaxed nondeterministic approximate programs." In: *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '12, Beijing, China - June 11 - 16, 2012*. Ed. by Jan Vitek, Haibo Lin, and Frank Tip. 2012, pp. 169–180.

[29] Magnus Carlsson. "Monads for Incremental Computing." In: *Proceedings of the 7th International Conference on Functional Programming*. ICFP '02. Pittsburgh, PA, USA, 2002, pp. 26–35.

[30] Swarat Chaudhuri, Sumit Gulwani, and Roberto Lublinerman. "Continuity Analysis of Programs." In: *Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '10. Madrid, Spain, 2010, pp. 57–70.

[31] Swarat Chaudhuri, Sumit Gulwani, and Roberto Lublinerman. "Continuity and robustness of programs." In: *Communications of the ACM* 55.8 (2012), pp. 107–115.

[32] Yan Chen, Joshua Dunfield, and Umut A. Acar. "Type-directed Automatic Incrementalization." In: *Proceedings of the 33rd Conference on Programming Language Design and Implementation*. PLDI '12. Beijing, China, 2012, pp. 299–310.

[33] Yan Chen, Joshua Dunfield, Matthew A. Hammer, and Umut A. Acar. "Implicit Self-Adjusting Computation for Purely Functional Programs." In: *J. Functional Programming* 24.1 (2014), pp. 56–112.

[34] Ezgi Çiçek, Deepak Garg, and Umut A. Acar. "Refinement Types for Incremental Computational Complexity." In: *Programming Languages and Systems - 24th European Symposium on Programming, ESOP 2015, London, UK, April 11-18, 2015. Proceedings*. 2015, pp. 406–431.

[35] Ezgi Çiçek, Zoe Paraskevopoulou, and Deepak Garg. "A Type Theory for Incremental Computational Complexity With Control Flow Changes." In: *Proceedings of the 21st International Conference on Functional Programming*. ICFP '16. Nara, Japan, 2016.

[36] Thierry Coquand. "An algorithm for type-checking dependent types." In: *Science of Computer Programming* 26.1 (1996), pp. 167 –177.

[37] Karl Crary. "Typed Compilation of Inclusive Subtyping." In: *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming*. ICFP '00. ACM, 2000, pp. 68–81. ISBN: 1-58113-202-6.

[38] Karl Crary and Stephnie Weirich. "Resource Bound Certification." In: *Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '00. Boston, MA, USA: ACM, 2000, pp. 184–198. ISBN: 1-58113-125-9.

[39] Ugo Dal Lago and Marco Gaboardi. "Linear Dependent Types and Relative Completeness." In: *Proceedings of the 2011 IEEE 26th Annual Symposium on Logic in Computer Science*. LICS '11. 2011, pp. 133–142.

[40] Ugo Dal Lago and Barbara Petit. "Linear Dependent Types in a Call-by-value Scenario." In: *Sci. Comput. Program.* 84 (May 2014), pp. 77–100. ISSN: 0167-6423.

[41] Ugo Dal lago and Barbara Petit. "The Geometry of Types." In: *Proceedings of the 40th Annual Symposium on Principles of Programming Languages*. POPL '13. Rome, Italy, 2013, pp. 167–178.

[42] Nils Anders Danielsson. "Lightweight semiformal time complexity analysis for purely functional data structures." In: *ACM SIGPLAN Notices*. Vol. 43. 1. ACM. 2008, pp. 133–144.

[43] Norman Danner, Daniel R. Licata, and Ramyaa Ramyaa. "Denotational Cost Semantics for Functional Languages with Inductive Types." In: *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming*. ICFP 2015. Vancouver, BC, Canada, 2015, pp. 140–151.

[44] Rowan Davies and Frank Pfenning. "Intersection Types and Computational Effects." In: *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming*. ICFP '00. 2000, pp. 198–208.

[45] Vincent Dornic, Pierre Jouvelot, and David K. Gifford. "Polymorphic Time Systems for Estimating Program Complexity." In: *ACM Lett. Program. Lang. Syst.* 1.1 (Mar. 1992), pp. 33–45.

[46] Joshua Dunfield and Neelakantan R. Krishnaswami. "Complete and Easy Bidirectional Typechecking for Higher-Rank Polymorphism." In: *International Conference on Functional Programming*. Sept. 2013.

[47] Joshua Dunfield and Neelakantan R. Krishnaswami. "Sound and Complete Bidirectional Typechecking for Higher-Rank Polymorphism with Existentials and Indexed Types." In: *CoRR* abs/1601.05106 (2016).

[48] Joshua Dunfield and Frank Pfenning. "Type Assignment for Intersections and Unions in Call-by-value Languages." In: *Proceedings of the 6th International Conference on Foundations of Software Science and Computation Structures and Joint European Conference on Theory and Practice of Software*. FOSSACS'03/ETAPS'03. Springer-Verlag, 2003, pp. 250–266.

[49] Dennis Felsing, Sarah Grebing, Vladimir Klebanov, Philipp Rümmer, and Mattias Ulbrich. "Automating regression verification." In: *ACM/IEEE International Conference on Automated Software Engineering, ASE '14, Vasteras, Sweden - September 15 - 19, 2014*. Ed. by Ivica Crnkovic, Marsha Chechik, and Paul Grünbacher. 2014, pp. 349–360.

[50] Jean-Christophe Filliâtre and Andrei Paskevich. "Why3: Where Programs Meet Provers." In: *Proceedings of the 22Nd European Conference on Programming Languages and Systems*. ESOP'13. Springer-Verlag, 2013, pp. 125–128.

[51] Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C. Pierce. "Linear Dependent Types for Differential Privacy." In: *Proceedings of the 40th Annual Symposium on Principles of Programming Languages*. POPL '13. Rome, Italy, 2013, pp. 357–370.

[52] Benny Godlin and Ofer Strichman. "Regression verification: proving the equivalence of similar programs." In: *Softw. Test., Verif. Reliab.* 23.3 (2013), pp. 241–258.

[53] Bernd Grobauer. "Cost Recurrences for DML Programs." In: *Proceedings of the Sixth ACM SIGPLAN International Conference on Functional Programming*. ICFP '01. ACM, 2001, pp. 253–264.

[54] Sumit Gulwani, Krishna K. Mehra, and Trishul Chilimbi. "SPEED: Precise and Efficient Static Estimation of Program Computational Complexity." In: *Proceedings of the 36th Annual Symposium on Princi-*

ples of Programming Languages. POPL '09. Savannah, GA, USA, 2009, pp. 127–139.

[55] Matthew A. Hammer, Umut A. Acar, and Yan Chen. "CEAL: A C-based Language for Self-adjusting Computation." In: *Proceedings of the 2009 Conference on Programming Language Design and Implementation*. PLDI '09. 2009, pp. 25–37.

[56] Matthew A. Hammer, Khoo Yit Phang, Michael Hicks, and Jeffrey S. Foster. "Adapton: Composable, Demand-driven Incremental Computation." In: *Proceedings of the 35th Conference on Programming Language Design and Implementation*. PLDI '14. Edinburgh, United Kingdom, 2014, pp. 156–166.

[57] Chris Hawblitzel, Shuvendu K. Lahiri, Kshama Pawar, Hammad Hashmi, Sedar Gokbulut, Lakshan Fernando, Dave Detlefs, and Scott Wadsworth. "Will you still compile me tomorrow? static cross-version compiler validation." In: *Proceedings of the Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13*. Ed. by Bertrand Meyer, Luciano Baresi, and Mira Mezini. 2013, pp. 191–201.

[58] Nevin Heintze and Jon G. Riecke. "The SLam Calculus: Programming with Secrecy and Integrity." In: *Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '98. New York, NY, USA: ACM, 1998, pp. 365–377.

[59] Allan Heydon, Roy Levin, and Yuan Yu. "Caching Function Calls Using Precise Dependencies." In: *Proceedings of the Conference on Programming Language Design and Implementation*. PLDI '00. Vancouver, British Columbia, Canada, 2000, pp. 311–320.

[60] Jan Hoffmann, Klaus Aehlig, and Martin Hofmann. "Multivariate Amortized Resource Analysis." In: *Proceedings of the 38th Annual Symposium on Principles of Programming Languages*. POPL '11. Austin, Texas, USA, 2011, pp. 357–370.

[61] Jan Hoffmann and Martin Hofmann. "Amortized Resource Analysis with Polynomial Potential: A Static Inference of Polynomial Bounds for Functional Programs." In: *Proceedings of the 19th European Conference on Programming Languages and Systems*. ESOP'10. Paphos, Cyprus: Springer-Verlag, 2010, pp. 287–306.

[62] Jan Hoffmann and Zhong Shao. "Automatic Static Cost Analysis for Parallel Programs." In: *Proceedings of the 24th European Symposium on Programming on Programming Languages and Systems*. New York, NY, USA: Springer-Verlag New York, Inc., 2015, pp. 132–157.

[63] Steffen Jost, Hans-Wolfgang Loid, Kevin Hammond, and Martin Hofmann. "Static Determination of Quantitative Resource Usage for Higher-Order Programs." In: *Symp. on Principles of Prog. Langs. (POPL '10)*. Madrid, Spain: ACM, Jan. 2010, pp. 223–236. ISBN: 978-1-60558-479-9.

[64] Shin-ya Katsumata. "Parametric Effect Monads and Semantics of Effect Systems." In: *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '14. ACM, 2014, pp. 633–645.

[65] Oleg Kiselyov, Amr Sabry, and Cameron Swords. "Extensible Effects: An Alternative to Monad Transformers." In: *Proceedings of the 2013 ACM SIGPLAN Symposium on Haskell*. Haskell '13. Boston, Massachusetts, USA: ACM, 2013, pp. 59–70. ISBN: 978-1-4503-2383-3.

[66] Christoph Koch. "Incremental Query Evaluation in a Ring of Databases." In: *Proceedings of the Twenty-ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. PODS '10. Indianapolis, Indiana, USA, 2010, pp. 87–98.

[67] Ugo Dal Lago and Marco Gaboardi. "Linear Dependent Types and Relative Completeness." In: vol. 8. 4. Logical Methods in Computer Science Association, 2012.

[68] Shuvendu K. Lahiri, Kapil Vaswani, and C. A. R. Hoare. "Differential static analysis: opportunities, applications, and challenges." In: *Proceedings of the Workshop on Future of Software Engineering Research, FoSER 2010, at the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2010, Santa Fe, NM, USA, November 7-11, 2010*. Ed. by Gruia-Catalin Roman and Kevin J. Sullivan. 2010, pp. 201–204.

[69] Shuvendu K. Lahiri, Chris Hawblitzel, Ming Kawaguchi, and Henrique Rebêlo. "SYMDIFF: A Language-Agnostic Semantic Diff Tool for Imperative Programs." In: *Computer Aided Verification - 24th International Conference, CAV 2012*. Ed. by P. Madhusudan and Sanjit A. Seshia. Vol. 7358. Lecture Notes in Computer Science. 2012, pp. 712–717.

[70] Shuvendu K. Lahiri, Kenneth L. McMillan, Rahul Sharma, and Chris Hawblitzel. "Differential assertion checking." In: *Proceedings of the Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13*. Ed. by Bertrand Meyer, Luciano Baresi, and Mira Mezini. 2013, pp. 345–355.

[71] Daniel Le Métayer. "ACE: An Automatic Complexity Evaluator." In: *ACM Trans. Program. Lang. Syst.* 10.2 (Apr. 1988), pp. 248–266. ISSN: 0164-0925.

[72] Paul Blain Levy. "Call-By-Push-Value." PhD thesis. Queen Mary and Westfield College, University of London, 2001. URL: http://www.cs.bham.ac.uk/~pbl/papers/thesisqmwphd.pdf.

[73] Ruy Ley-Wild, Umut A. Acar, and Matthew Fluet. "A Cost Semantics for Self-adjusting Computation." In: *Proceedings of the 36th Annual Symposium on Principles of Programming Languages*. POPL '09. Savannah, GA, USA, 2009, pp. 186–199.

[74] Sam Lindley, Conor McBride, and Craig McLaughlin. "Do Be Do Be Do." In: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*. POPL 2017. ACM, 2017, pp. 500–514.

[75] Yanhong A. Liu and Tim Teitelbaum. "Systematic Derivation of Incremental Programs." In: *Science of Computer Programming* 24.1 (1995), pp. 1–39.

[76] Francesco Logozzo, Shuvendu K. Lahiri, Manuel Fähndrich, and Sam Blackshear. "Verification modulo versions: towards usable verification." In: *Proceedings of 2014 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI'14*. Ed. by Michael F. P. O'Boyle and Keshav Pingali. 2014, p. 32.

[77] Conor McBride and James McKinna. "The view from the left." In: *Journal of Functional Programming* 14.1 (2004).

[78] Aleksandar Nanevski, Anindya Banerjee, and Deepak Garg. "Verification of Information Flow and Access Control Policies with Dependent Types." In: *32nd IEEE Symposium on Security and Privacy, S&P 2011, 22-25 May 2011, Berkeley, California, USA*. 2011, pp. 165–179.

[79] Aleksandar Nanevski, Anindya Banerjee, and Deepak Garg. "Dependent Type Theory for Verification of Information Flow and Access Control Policies." In: *ACM Trans. Program. Lang. Syst.* 35.2 (July 2013).

[80] V. C. Ngo, M. Dehesa-Azuara, M. Fredrikson, and J. Hoffmann. "Verifying and Synthesizing Constant-Resource Implementations with Types." In: *2017 IEEE Symposium on Security and Privacy (SP)*. 2017, pp. 710–728.

[81] Flemming Nielson and HanneRiis Nielson. "Type and Effect Systems." In: *Correct System Design*. Vol. 1710. Lecture Notes in Computer Science. 1999, pp. 114–136.

[82] Ulf Norell. "Towards a practical programming language based on dependent type theory." PhD thesis. SE-412 96 Göteborg, Sweden: Department of Computer Science and Engineering, Chalmers University of Technology, 2007.

[83] Chris Okasaki. *Purely Functional Data Structures*. New York, NY, USA: Cambridge University Press, 1998. ISBN: 0-521-63124-6.

[84] Robert Paige and Shaye Koenig. "Finite Differencing of Computable Expressions." In: *ACM Trans. Program. Lang. Syst.* 4.3 (1982), pp. 402–454.

[85] Zoe Paraskevopoulou. "Self-Adjusting Computation for CostIt." MA thesis. École Normale Supérieure de Cachan, France, 2016.

[86] Nimrod Partush and Eran Yahav. "Abstract semantic differencing via speculative correlation." In: *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications, OOPSLA 2014, part of SPLASH 2014, Portland, OR, USA, October 20-24, 2014*. Ed. by Andrew P. Black and Todd D. Millstein. 2014, pp. 811–828.

[87] Suzette Person, Matthew B. Dwyer, Sebastian G. Elbaum, and Corina S. Pasareanu. "Differential symbolic execution." In: *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2008, Atlanta, Georgia, USA, November 9-14, 2008*. Ed. by Mary Jean Harrold and Gail C. Murphy. 2008, pp. 226–237.

[88] Suzette Person, Guowei Yang, Neha Rungta, and Sarfraz Khurshid. "Directed incremental symbolic execution." In: *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*. Ed. by Mary W. Hall and David A. Padua. 2011, pp. 504–515.

[89] Simon Peyton Jones, Dimitrios Vytiniotis, Stephanie Weirich, and Mark Shields. "Practical Type Inference for Arbitrary-rank Types." In: *J. Funct. Program.* 17.1 (Jan. 2007), pp. 1–82.

[90] Brigitte Pientka. "A Type-theoretic Foundation for Programming with Higher-order Abstract Syntax and First-class Substitutions." In: *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '08. 2008, pp. 371–382.

[91] Benjamin C. Pierce and David N. Turner. "Local Type Inference." In: *ACM Trans. Program. Lang. Syst.* 22.1 (Jan. 2000), pp. 1–44. ISSN: 0164-0925.

[92] Gordon D. Plotkin. *Lambda-Definability and Logical Relations*. Memorandum SAI–RM–4. Edinburgh, Scotland: University of Edinburgh, 1973.

[93] François Pottier and Vincent Simonet. "Information Flow Inference for ML." In: *ACM Trans. Prog. Lang. Sys.* 25.1 (Jan. 2003), pp. 117–158.

[94] Jason Reed and Benjamin C. Pierce. "Distance Makes the Types Grow Stronger: A Calculus for Differential Privacy." In: *Proceedings of the 15th International Conference on Functional Programming*. ICFP '10. Baltimore, Maryland, USA: ACM, 2010, pp. 157–168.

[95] Jason Reed and Benjamin C. Pierce. "Distance makes the types grow stronger: a calculus for differential privacy." In: *Proceeding of the 15th ACM SIGPLAN international conference on Functional programming, ICFP 2010, Baltimore, Maryland, USA, September 27-29, 2010*. 2010, pp. 157–168.

[96] Brian Reistad and David K. Gifford. "Static Dependent Costs for Estimating Execution Time." In: *Proceedings of the 1994 ACM Conference on LISP and Functional Programming*. LFP '94. Orlando, Florida, USA, 1994, pp. 65–78.

[97] John C. Reynolds. "Types, Abstraction and Parametric Polymorphism." In: *IFIP Congress*. 1983, pp. 513–523.

[98] Mads Rosendahl. "Automatic Complexity Analysis." In: *Proceedings of the Fourth International Conference on Functional Programming Languages and Computer Architecture*. FPCA '89. Imperial College, London, United Kingdom: ACM, 1989, pp. 144–156. ISBN: 0-89791-328-0.

[99] David Sands. "Total Correctness by Local Improvement in Program Transformation." In: *Proceedings of the 22Nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. 1995, pp. 221–232.

[100] Tim Sheard. "Type-level Computation Using Narrowing in Î©mega." In: *Electronic Notes in Theoretical Computer Science* 174.7 (2007). Proceedings of the Programming Languages meets Program Verification (PLPV 2006), pp. 105 –128.

[101] Matías Toro and Éric Tanter. "Customizable Gradual Polymorphic Effects for Scala." In: *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*. OOPSLA 2015. New York, NY, USA: ACM, 2015, pp. 935–953.

[102] Phil Trinder, Murray Cole, Kevin Hammond, Hans-Wolfgang Loidl, and Greg Michaelson. "Resource analyses for parallel and distributed coordination." In: 25 (Mar. 2013).

[103] Pedro Vasconcelos. "Space cost analysis using sized types." PhD thesis. School of Computer Science, University of St Andrews, 2008.

[104] Niki Vazou, Eric L. Seidel, and Ranjit Jhala. "LiquidHaskell: experience with refinement types in the real world." In: *Proceedings of the 2014 ACM SIGPLAN symposium on Haskell, Gothenburg, Sweden, September 4-5, 2014*. 2014, pp. 39–51.

[105] Hongwei Xi. available as `https://www.cs.cmu.edu/~rwh/theses/xi.pdf`. PhD thesis. Carnegie Mellon University, 1998.

[106] Hongwei Xi and Frank Pfenning. "Dependent Types in Practical Programming." In: *Proceedings of the 26th Symposium on Principles of Programming Languages*. POPL '99. San Antonio, Texas, USA, 1999, pp. 214–227.

[107] Guowei Yang, Matthew B. Dwyer, and Gregg Rothermel. "Regression model checking." In: *25th IEEE International Conference on Software Maintenance (ICSM 2009), September 20-26, 2009, Edmonton, Alberta, Canada*. 2009, pp. 115–124.

[108] Hongseok Yang. "Relational Separation Logic." In: *Theor. Comput. Sci.* 375.1-3 (Apr. 2007), pp. 308–334.

[109] Hongseok Yang. "Relational separation logic." In: 375.1-3 (2007), pp. 308–334.

[110] Ezgi Çiçek, Gilles Barthe, Marco Gaboardi, Deepak Garg, and Jan Hoffmann. "Relational Cost Analysis." In: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*. POPL 2017. ACM, 2017, pp. 316–329.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede.