An Approximation and Refinement Approach to First-Order Automated Reasoning

Dissertation

zur Erlangung des Grades des Doktors der Naturwissenschaften
an der Fakultät Mathematik und Informatik
der Universität des Saarlandes

vorgelegt von Andreas Teucke

Saarbrücken, Oktober, 2017

Tag des Kolloquiums:	16.5.2018
Dekan:	Prof. Dr. Frank-Olaf Schreyer
Prüfungsausschuss	
Berichterstatter:	Dr. Konstantin Korovin
	Prof. Dr. Christoph Weidenbach
Vorsitzender:	Prof.Dr. Gert Smolka
Akademischer Mitarbeiter:	Dr. Sophie Tourret

Abstract

With the goal of lifting model-based guidance from the propositional setting to first-order logic, I have developed an approximation theorem proving approach based on counterexample-guided abstraction refinement. A given clause set is transformed into a simplified form where satisfiability is decidable. This approximation extends the signature and preserves unsatisfiability: if the simplified clause set is satisfiable, so is the original clause set. A resolution refutation generated by a decision procedure on the simplified clause set can then either be lifted to a refutation in the original clause set, or it guides a refinement excluding the previously found unliftable refutation. This way the approach is refutationally complete.

The monadic shallow linear Horn fragment, which is the initial target of the approximation, is well-known to be decidable. It was a long standing open problem how to extend the fragment to the non-Horn case, preserving decidability, that would, e.g., enable to express non-determinism in protocols. I have now proven decidability of the non-Horn monadic shallow linear fragment via ordered resolution.

I further extend the clause language with a new type of constraints, called straight dismatching constraints. The extended clause language is motivated by an improved refinement of the approximation-refinement framework. All needed operations on straight dismatching constraints take linear or linear logarithmic time in the size of the constraint. Ordered resolution with straight dismatching constraints is sound and complete and the monadic shallow linear fragment with straight dismatching constraints is decidable.

I have implemented my approach based on the SPASS theorem prover. On certain satisfiable problems, the implementation shows the ability to beat established provers such as SPASS, iProver, and Vampire.

Zusammenfassung

Mit dem Ziel die Modell-basierten Methoden der Aussagenlogik auf die Logik erster Stufe anzuwenden habe ich ein approximations Beweis-System entwickelt, das auf der Idee der 'Gegenbeispiel-gelenkten Abstraktions-Verfeinerung' beruht.

Eine gegebene Klausel-Menge wird zunächst in eine vereinfachte Form übersetzt, in der die Erfüllbarkeit entscheidbar ist. Diese sogenannte Approximation erweitert die Signatur, aber erhält Unerfüllbarkeit: Falls die approximierte Klauseln erfüllbar sind, so ist es auch die ursprüngliche Menge. Ein Resolutions-Beweis, der von einer Entscheidungs-Prozedur auf der Approximation erzeugt wurde, kann dann entweder als Basis eines Unerfüllbarkeits Beweises der ursprünglichen Menge dienen oder aber eine Verfeinerung der Approximation aufzeigen, welche den gefundenen Beweis davon ausschließt noch einmal gefunden zu werden. Damit ist der Ansatz widerlegungs vollständig.

Das monadisch flache lineare Horn Fragment, das als anfängliches Ziel der Approximation dient, ist bereits seit längerem als entscheidbar bekannt. Es war ein lange offenes Problem, wie man das Fragment auf den nicht-Horn Fall erweitern kann ohne Entscheidbarkeit zu verlieren. Damit lässen sich unter anderem nicht-deterministische Protokolle ausgedrücken. Ich habe nun die Entscheidbarkeit des nicht-Horn monadisch flachen linearen Fragments mittels geordneter Resolution bewiesen.

Zusätzlich habe ich die Klausel-Sprache durch eine neue Art von Constraints erweitert, die ich als 'straight dismatching constraints' bezeichne. Diese Erweiterung ist dadurch motiviert dass sie eine Verbesserung der Approximations-Verfeinerung des vorgestellten Systems erlaubt. Alle benötigten Operationen auf diesen Constraints nehmen lediglich lineare oder linear-logarithmische Zeit und Platz in Anspruch. Ich zeige, dass geordnete Resolution mit Constraints korrekt und vollständig ist und dass das monadische flache lineare Fragment mit Constraints entscheidbar ist.

Ich habe meinen Ansatz auf dem Theorem-Beweiser SPASS basierend implementiert. Auf bestimmten erfüllbaren Problem schlägt meine Implementierung sogar etablierte Beweiser wie SPASS, iProver und Vampire.

Acknowledgments

First and foremost, I thank Christoph Weidenbach, who supported me throughout the different stages of my research. Despite his always tight schedule, he put a lot of time into our reviews and especially polishing my submissions to conferences. Thanks to this, most of them were accepted on the first try. Furthermore, he always knew to point me in the right direction when I was out of ideas or which direction to focus on when I had too many.

I want to thank Gert Smolka for introducing me to computational logic in his lectures and suggesting Christoph's group to me for my master's and Ph.D.

Further thanks to Uwe Waldmann's dedication to his deepening lecture on first-order logic – despite the circumstance that the number of students could be counted on one hand – and his continued helping hand throughout my masters and Ph.D.

I would also like to thank all the colleagues of the Automation of Logic group for a friendly and fun atmosphere, and especially my office-mates Gábor Alagi and Martin Bromberger.

I am grateful to the university for providing such a high quality computer science faculty so close to my home, allowing me to stay near to my supportive family.

Moreover, I want to thank the anonymous reviewers of the publications underlying this thesis for their valuable input.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Inst-Gen calculus by Ganzinger and Korovin [27] and its implementation in iProver has shown to be very successful. The calculus is based on an under-approximation - instantiation refinement loop. A given first-order clause set is under-approximated by finite grounding and afterwards a SAT-solver is used to test unsatisfiability. If the ground clause set is unsatisfiable then a refutation for the original clause set is found. If it is satisfiable, the model generated by the SAT-solver is typically not a model for the original clause set. If it is not, it is used to instantiate the original clause such that the found model is ruled out for the future.

In this thesis, I define an approximation-based first-order theorem proving approach based on counterexample-guided abstraction refinement that is dual to the Inst-Gen calculus. A given first-order clause set N is step by step transformed into an over-approximation N_k in a decidable fragment of first-order logic. That means if N_k is satisfiable so is N. However, if N_k is unsatisfiable, then it is not known whether N in unsatisfiable, in general. In that case, the approximation provides a lifting terminology for the found refutation. Each step of the transformation is considered separately to attempt to transform the proof of unsatisfiability for N_k to a proof of unsatisfiability for N. If this fails, the cause is analysed to refine the original clause set such that the found refutation is ruled out for the future and the procedure repeats.

As a starting point of the approach, I consider first-order logic without equality and use the Monadic Shallow Linear Horn theory [45] (MSLH) as the decidable fragment of the approximation. The fragment consists of first-order clauses of the form $\Gamma \rightarrow P(t)$ where all predicates are monadic and t is either a variable or a linear term $f(x_1, \ldots, x_n)$. The atoms in Γ, denoting negative literals, are not subject to any restriction. The transformation into this fragment is polynomial in the size of N and constitutes an over-approximation. However, using the MSLH fragment, the lifting and refinement were quite expensive. Therefore, I developed two new decidable fragments based on MSLH that alleviate the problems. The first improvement over MSLH is the removal of the restriction to Horn clauses, which lead to the MSL fragment. This avoids an exponential lifting step caused by the transformation into Horn clauses. The second change is the introduction of straight dismatching constraints to MSL, called the MSL(SDC) fragment. With MSL(SDC) the refinement phase is simplified from creating quadratically many instantiations to just two instead. The MSLH fragment and its successors properly include first-order ground logic, but are also expressive enough to represent minimal infinite Herbrand models.

In addition to developing a new proof method for first-order logic this constitutes my second motivation for studying the new calculus and its approximation. It is meanwhile accepted that a model-based guidance can significantly improve an automated reasoning calculus. The propositional CDCL calculus [32] is one prominent example for this insight. In first-order logic, (partial) model operators typically generate inductive models for which almost all interesting properties become undecidable, in general. One way out of this problem is to generate a model for an approximated clause set, such that important properties with respect to the original clause set are preserved. In the case of my calculus and approximation, a found model can be effectively translated into a model for the original clause set. So my result is also a first step towards model-based guidance in first-order logic automated reasoning.

1.1 The Approximation-Refinement Loop

The central method introduced in this thesis is the approximation-refinement loop, where the four steps of approximating, solving, lifting, and refining are repeated until either the approximation is satisfiable or an unsatisfiability proof of the original set is found.

Starting from a set N of clauses, first, the clauses are transformed to a set N': $N \Rightarrow^{*}_{\mathrm{App}} N'$. For the loop, the transformation requires the properties that N' belongs to a decidable fragment and if N' is satisfiable so is N.

If N' is unsatisfiable, then it is not known whether N in unsatisfiable, in general. However, there necessarily exists a resolution refutation for N', which I can potentially lift into a proof of unsatisfiability of N by following $\Rightarrow^{*}_{\mathrm{App}}$ in reverse.

To avoid the complex DAG structure of resolution refutations, I instead introduce the notion of a *conflicting core* of a clause set N. Given an unsatisfiable clause set N a conflicting core of N is a clause set $N^{\perp}$ such that for each clause $C' \in N^{\perp}$ there is a clause $C \in N$ with $C\sigma = C'$ for some σ and $N^{\perp}\tau$ is unsatisfiable for any substitution τ. Thus, conflicting cores are finite, unsatisfiable clause sets where the remaining variables are used as global parameters such that any instantiation preserves unsatisfiability. Conflicting cores can be effectively generated out of a resolution refutation.

Now given an approximation $N = N_0 \Rightarrow^{k}_{\mathrm{App}} N_k$ and a conflicting core $N^{\perp}_{k}$ of N_k I proceed as follows: if the conflicting core $N^{\perp}_{k}$ can be lifted to a conflicting core $N^{\perp}_{k-1}$ of N_{k-1} by undoing the approximation step $N_{k-1} \Rightarrow_{\mathrm{App}} N_k$, I continue with $N = N_0 \Rightarrow^{k-1}_{\mathrm{App}} N_{k-1}$. If I arrive this way at $N = N_0$, unsatisfiability of N

is shown. If the approximation step fails, then I refine N and eventually N_{k-1} via instantiation, such that the unliftable conflicting core can no longer be generated. I show that the approach is sound and complete: a lifted conflicting core obviously shows unsatisfiability of N and if N has a conflicting core, then it can be found and lifted by the procedure.

The overall approach is parametric in the actual reasoning procedure for the decidable fragment the clause set N_k eventually belongs to. It relies on the properties of the approximation relation $\Rightarrow_{\mathrm{App}}$.

For example, consider the first-order Horn clauses, here written as an implication with a conjunction of negative literals on the left and a disjunction of positive literals on the right,

$$\begin{array}{rcl} S(x) & \rightarrow & P(x, g(x)) \\ & \rightarrow & S(a) \\ & \rightarrow & S(b) \\ & \rightarrow & S(g(x)) \\ P(a, g(b)) & \rightarrow & \\ P(g(x), g(g(x))) & \rightarrow & \end{array}$$

that are approximated into the MSLH theory

$$\begin{array}{rcl} S(x), R(y) & \rightarrow & T(f_P(x, y)) \\ S(x) & \rightarrow & R(g(x)) \\ & \rightarrow & S(a) \\ & \rightarrow & S(b) \\ & \rightarrow & S(g(x)) \\ T(f_P(a, g(b))) & \rightarrow & \\ T(f_P(g(x), g(g(x)))) & \rightarrow & \end{array}$$

where the relation P is encoded by the function f_P and the non-shallow sub-term $g(x)$ in the first clause is extracted by the introduction of the additional predicate R. The approximated clause set has two possible refutations: one using $\neg T(f_P(a, g(b)))$ and the second using $\neg T(f_P(g(x), g(g(x))))$ plus the rest of the clauses, respectively.

For the first refutation the conflicting core is

$$\begin{array}{rcl} S(a), R(g(b)) & \rightarrow & T(f_P(a, g(b))) \\ S(b) & \rightarrow & R(g(b)) \\ & \rightarrow & S(a) \\ & \rightarrow & S(b) \\ T(f_P(a, g(b))) & \rightarrow & \end{array}$$

In this case lifting fails because where in the original clause $S(x) \rightarrow P(x, g(x))$ the variable x is used, the conflicting core clause $S(a), R(g(b)) \rightarrow T(f_P(a, g(b)))$ instantiates both a and b. Therefore the original clause set is refined by replacing the first clause with

$$\begin{array}{rcl} S(a) & \rightarrow & P(a, g(a)) \\ S(b) & \rightarrow & P(b, g(b)) \text{ and} \\ S(g(x)) & \rightarrow & P(g(x), g(g(x))). \end{array}$$

The approximation of the refined clause set

$$\begin{array}{rcl} S(a) & \rightarrow & P(a, g(a)) \\ S(b) & \rightarrow & P(b, g(b)) \\ S(g(x)) & \rightarrow & P(g(x), g(g(x))) \\ S(g(x)), R(y) & \rightarrow & T(f_P(g(x), y)) \\ R'(y) & \rightarrow & R(g(y)) \\ S(g(x)) & \rightarrow & R'(g(x)) \\ & \rightarrow & S(a) \\ & \rightarrow & S(b) \\ & \rightarrow & S(g(x)) \\ T(f_P(a, g(b))) & \rightarrow & \\ T(f_P(g(x), g(g(x)))) & \rightarrow & \end{array}$$

does no longer enable a refutation using $\neg T(f_P(a, g(b)))$. Therefore, the refutation using $\neg T(f_P(g(x), g(g(x))))$ is found and lifted instead.

1.2 Related Work

The initial starting point of this work was based on [45] which shows decidability of the MSLH fragment, referred to as a "monadic Horn theory where all positive literals are linear and shallow". To avoid exponential and quadratic blow-ups in the lifting and refinement phases, the proofs in Sections 5.1 and 6.2 extend this result to non-Horn clauses combined with the notion of dismatching constraints, which are a restricted version of the dismatching constraints introduced in [1].

A fragment equivalent to MSLH, called $\mathcal{H}_1$, was independently described and proven decidable in [31]. A monadic Horn clause $P_1(t_1), \ldots, P_n(t_n) \rightarrow E$ is in $\mathcal{H}_1$ if E is linear and for any two variables x and y that appear in E and together in some $P_i(t_i)$, x and y are arguments of the same subterm in E. While $\mathcal{H}_1$ is not positive shallow, the condition on its variables guarantees that an $\mathcal{H}_1$-clause can be transformed into an equivalent set of MSLH clauses using Shallow transformation. Their approach is based on automata transformations as opposed to my superposition approach. On one hand, this allowed $\mathcal{H}_1$ to be successfully extended with simple, path and homomorphism disequalities in [38, 39, 35], respectively. On the other hand, the superposition approach allowed the inclusion of non-Horn clause and dismatching constraints into the fragment. Neither of these extension are compatible in an apparent way with the respective other approach.

In [19], the decidability of $\mathcal{H}_1$ is also shown using superposition in a very similar way to [45]. [19] achieves single exponential time complexity as compared to the double exponential complexity of [45] using an additional rule, called ϵ-splitting, that replaces a clause $P_1(x), \ldots, P_n(x), \Gamma \rightarrow E$ with $P_1(x), \ldots, P_n(x) \rightarrow P$ and

$P, \Gamma \to E$ where P is a fresh predicate and x does not appear in $\Gamma \to E$. In practice, however, the first-order prover SPASS [46] is able to apply general splitting internally, which includes ϵ-splitting.

In "A theory of abstractions" [18] Giunchiglia and Walsh define a general framework to classify and compare approximations, which are here called abstractions. They informally define abstractions as "the process of mapping a representations of a problem" that "helps deal with the problem in the original search space by preserving certain desirable properties" and "is simpler to handle". In their framework an abstraction is a mapping between formal systems, i.e., a triple of a language, axioms and deduction rules, which satisfy one of the following conditions: An increasing abstraction (TI) f maps theorems only to theorems, i.e., if α is a theorem, then $f(\alpha)$ is also a theorem, while a decreasing abstraction (TD) maps only theorems to theorems, i.e., if $f(\alpha)$ is a theorem, then α was also a theorem. Furthermore, they define dual definitions for refutations, where not theorems but formulas that make a formal system inconsistent are considered. An increasing abstraction (NTI) then maps inconsistent formulas only to inconsistent formulas and vice versa for decreasing abstractions (NTD). With respect to their notions the approximation described in this paper is an abstraction where the desirable property is the over-approximation and the decidability of the fragment makes it simpler to handle. More specifically $\Rightarrow_{\mathrm{AP}}$ is an NTI abstraction for refutation systems, i.e., it is an abstraction that preserves inconsistency of the original.

In [18], they list several examples of abstractions such as ABSTRIPS by Sacerdoti [37], a GPS planning method by Newell and Simon [30], Plaisted's theory of abstractions [34], propositional abstractions exemplified by Giunchiglia [17], predicate abstractions by Plaisted [34] and Tenenberg [42], domain abstractions by Hobbs [22] and Imielinski [23] and ground abstractions introduced by Plaisted [34]. In [34], three classes of abstractions are defined. The first two are ordinary and weak abstractions, which share the condition that if C subsumes D then every abstraction of D is subsumed by some abstraction of C. However, my approximations fall in neither class as they violate this condition via the Shallow transformation. For example $Q(x) \to P$ subsumes $Q(x) \to Q(f(f(x))), P$; but the shallow approximations $S(x') \to Q(f(x')), P$ and $Q(x) \to S(f(x))$ of $Q(x) \to Q(f(f(x))), P$ are not subsumed by any approximation of $Q(x) \to P$. The third class are generalization functions, which do not change the problem but abstract the resolution rule of inference.

Another approximation framework is defined by Kautz and Selman [25], consisting of a tuple $\langle \mathcal{L}, \models, \mathcal{L}_S, \mathcal{L}_T, \mathcal{L}_Q, f_L, f_U \rangle$ called a *knowledge compilation system*. $\mathcal{L}$ is the main logic, which is first-order logic without equality in my case. f_L and f_U, which they call lower and upper bounds, are over- and under-approximations that translate sets in the input fragment $\mathcal{L}_S$ into series of approximations with increasing precision in the target fragment $\mathcal{L}_T$. For a query α in the query language $\mathcal{L}_Q$ and set $N \in \mathcal{L}_S$, it holds that if $f_L(N) \not\models \alpha$ then $N \not\models \alpha$ and if $f_U(N) \models \alpha$ then $N \models \alpha$. My approximation-refinement approach can be considered a knowledge compilation system where $\mathcal{L}_T$ is the MSL fragment, $\mathcal{L}_Q$ contains just the empty clause $\Box$, and f_L is the approximation $\Rightarrow_{\text{AP}}$ with successive refinements. While I do not use one, I can, following the example in [25], also remove all non-MSL clauses from a given input to create an under-approximation.

An abstraction-refinement framework intended for large theories is introduced by Hernandez and Korovin in [21]. In the context of this framework, $\Rightarrow_{\text{App}}$ would be called a *strengthening abstract function*. While they also include under-approximation, their variation on lifting and refinement using a *concretisation function* is weaker than mine. It has to refine all abstraction clauses involved in the proof since it is unable to extract a singular cause.

Abstractions close to $\Rightarrow_{\text{AP}}$ are mentioned in [20] and [16]. Goubault-Larrecq and Parrennes model C code with Horn clauses [20]. By using rules similar to $\Rightarrow_{\text{LI}}$ and $\Rightarrow_{\text{SH}}$, they abstract these clauses into $\mathcal{H}_1$ and then use automated reasoning to check desirable properties. Since most model clauses are already in $\mathcal{H}_1$, the abstraction is reasonably close to the original problem. Fruhwith et al. use approximation to type check logic programs [16]. A Horn clause $P_1(t_1), \ldots, P_m(t_m) \rightarrow P(t)$ representing a logic rule is approximated by clauses $X_1(x_1^1), \ldots, X_k(x_k^l) \rightarrow \text{type}(f_p(\tilde{t}))$ and $\text{type}(f_{p_1}(t_1)), \ldots, \text{type}(f_{p_m}(t_m)) \rightarrow X_i(x_i)$, where $x_i^1, \ldots, x_i^n$ are the occurrences of variable x_i in $P(t)$ and $\tilde{t}$ is the linearisation of t replacing each x_i by the respective x_i^j. This is approximation closely resembles a combination of Monadic and Linear transformation, where the approximation is again in $\mathcal{H}_1$.

My approach of using an approximation-refinement loop, where the refinement is based on the unsatisfiability proof of the approximation, closely resembles the counterexample-guided abstraction refinement framework (CEGAR) described in [12]. There, CEGAR is applied to CTL based automata, where the automaton is the synthesis of a model and a negated property. An accepting trace of this automaton then constitutes a counterexample of the property. Since, in general, the automaton cannot be efficiently checked, a simplified abstraction is used instead. These, however, can introduce "spurious" counterexamples that are then targeted by refinement. Counterexamples correspond to conflicting cores in my framework and spurious counterexamples are the equivalent of unliftable conflicting cores. The CEGAR framework has been applied to numerous problems, however, generally only in the context of model checking [7, 8, 15, 28].

An application of CEGAR close to this paper is found in the theorem prover iProver. iProver uses the Inst-Gen [27] method, where a first-order problem is abstracted with a SAT problem by replacing every variable with a (fresh) constant c. The abstraction is solved by a SAT solver and a model is lifted to the original clause set by equating abstracted terms with the set they represent, e.g., if $P(c)$ is true in a model returned by the SAT solver, then all instantiations of the original $P(x)$ are considered true as well. Inst-Gen abstracts using an under-approximation of the original clause set. In case the lifting of the satisfying model is inconsistent, the clash is resolved by appropriately instantiating the involved clauses, which mimics an inference step. This is the dual of my method with the roles of satisfiability and unsatisfiability interchanged. A further difference, however, is that Inst-Gen only finds finite models after approximation, while my approximation also discovers infinite Herbrand models. For example the simple problem

$$\begin{array}{rcl} & \rightarrow & P(a) \\ P(f(a)) & \rightarrow & \\ P(x) & \rightarrow & P(f(f(x))) \\ P(f(f(x))) & \rightarrow & P(x) \end{array}$$

has the satisfying Herbrand model where P is the set of even numbers. However, iProver's approximation can never return such a model as any $P(f^n(c))$ will necessarily abstract both true and false atoms and therefore instantiate new clauses infinitely. My method on the other hand will produce the approximation

$$\begin{array}{rcl} & \rightarrow & P(a) \\ P(f(a)) & \rightarrow & \\ S(y) & \rightarrow & P(f(y)) \\ P(x) & \rightarrow & S(f(x)) \\ P(f(f(x))) & \rightarrow & P(x) \end{array}$$

which is saturated after inferring $\neg S(a)$ from clauses two and three. However, Inst-Gen can easily solve clause sets such as $\{P(y, g(y)) \rightarrow;\ \rightarrow P(x, x)\}$ that my method struggles with (see Section 6.5.1).

1.3 Main Contributions

This thesis contributes to both theory and practice of automated first-order reasoning. The main contributions can be summarized as follows:

Chapter 3: First-Order Over-Approximations I examine a number of transformations on first-order clauses and their properties; especially with respect to model approximation.

Chapter 4: Approximation-Refinement A novel approximation approach for first-order theorem proving based on counter-example guided abstraction refinement. I show that this approach can be applied to create a sound and complete calculus. I further incrementally improve this calculus to address practical concerns.

Chapter 5: Decidability of the Monadic Shallow Linear Fragment As an improvement in the lifting phase of the Approximation-Refinement, I define a new decidable fragment of first-order logic by extending the decision procedure of the monadic, shallow, linear, Horn fragment to non-Horn clauses.

Chapter 6: Straight Dismatching Constraints I introduce a new subset of dismatching constraints. These straight dismatching constraints are compatible with the decision procedure of the MSL fragment and they further have a polynomial-time satisfiability check as opposed to the exponential-time cost of general dismatching constraints. Nevertheless, they still allow an improvement of the Approximation-Refinement's refinement phase from a quadratic to a constant size operation.

Chapter 7: Implementation I present a prototype implementation which incorporates all three of my theoretical contributions. Although it cannot compete with existing (portfolio) solvers, there are certain problem types only my implementation can solve.

Chapter 2

Preliminaries

I consider a standard first-order language where letters v, w, x, y, z denote variables, f, g, h functions, a, b, c constants, s, t terms, p, q, r positions and Greek letters $\sigma, \tau, \rho, \delta$ are used for substitutions. S, P, Q, R denote predicates, $\approx$ denotes equality, A, B atoms, E, L literals, C, D clauses, and N clause sets. $\overline{L}$ is the complement of L. The signature $\Sigma = (\mathcal{F}, \mathcal{P})$ consists of two disjoint, non-empty, in general infinite sets of function and predicate symbols $\mathcal{F}$ and $\mathcal{P}$, respectively. The set of all *terms* over the set of variables $\mathcal{V}$ is $\mathcal{T}(\mathcal{F}, \mathcal{V})$. I use $f(\bar{t})$ and $f(\overline{x})$ as shorthand for the terms $f(t_1, \ldots, t_n)$ and $f(x_1, \ldots, x_n)$ where n is the arity of f. If there are no variables, then terms, literals and clauses are called *ground*, respectively. A *substitution* σ is denoted by pairs $\{x \mapsto t\}$ and its update at x by $\sigma[x \mapsto t]$. A substitution σ is a *grounding* substitution for a set of variables $\mathcal{V}$ if $x\sigma$ is ground for every variable $x \in \mathcal{V}$. The set of all ground clauses of a clause set N is defined as $\mathcal{G}(N) := \{C\sigma \mid C \in N \text{ and } C\sigma \text{ is ground}\}$.

The set of *free* variables of an atom A (term t) denoted by vars(A) (vars(t)). A *position* is a sequence of positive integers, where ε denotes the empty position. As usual $t|_p = s$ denotes the subterm s of t at position p, which I also write as $t[s]_p$, and $t[p/s']$ then denotes the replacement of s with s' in t at position p. These notions are extended to literals and multiple positions.

A predicate with exactly one argument is called *monadic*. A term is *complex* if it is not a variable and *shallow* if it has at most depth one. It is called *linear* if there are no duplicate variable occurrences. A literal, where every argument term is shallow, is also called *shallow*. A term $f(s_1, \ldots, s_n)$ is called *straight*, if $f(s_1, \ldots, s_n)$ is linear and all arguments are variables except for at most one straight argument term s_i. For example, the terms $f(x, f(a, y))$ and $f(x, f(y, z))$ are straight, while $f(x, f(a, b))$ is not. The *term skeleton* skt(t) of a term t is recursively defined by the following two rules: (1) $\text{skt}(x) = x'$, where x' is a fresh variable and (2) $\text{skt}(f(s_1, \ldots, s_n)) = f(\text{skt}(s_1), \ldots, \text{skt}(s_n))$. Informally, the skt($t$) is the linearisation of t, e.g., $\text{skt}(f(x, g(x, x))) = f(x, g(y, z))$.

A *clause* is a multiset of literals which I write as an implication $\Gamma \to \Delta$ where the atoms in the multiset Δ (the *succedent*) denote the positive literals and the atoms in the multiset Γ (the *antecedent*) the negative literals. In this notation, a clause $E_1 \lor \neg E_2 \lor \neg E_3 \lor E_4 \lor E_5$, in disjunctive normal form, is written as $E_2, E_3 \rightarrow E_1, E_4, E_5$. I write $\square$ for the empty clause. If two clauses C and D are equal up to α-renaming, D is called a *variant* of C and vice-versa. If Γ is empty I omit $\rightarrow$, e.g., I can write $P(x)$ as an alternative of $\rightarrow P(x)$ whereas if Δ is empty $\rightarrow$ is always shown. I abbreviate disjoint set union with sequencing, for example, I write $\Gamma, \Gamma' \rightarrow \Delta, L$ instead of $\Gamma \cup \Gamma' \rightarrow \Delta \cup \{L\}$. A clause $E, E, \Gamma \rightarrow \Delta$ is equivalent to $E, \Gamma \rightarrow \Delta$ and I call them equal *modulo duplicate literal elimination*. If every term in Δ is shallow, the clause is called *positive shallow*. If all atoms in Δ are linear and variable disjoint, the clause is called *positive linear*. A clause $\Gamma \rightarrow \Delta$ is called an *MSL* clause, if it is (i) positive shallow and linear, (ii) all occurring predicates are monadic, (iii) no equations occur in Δ, and (iv) no equations occur in Γ or $\Gamma = \{s \approx t\}$ and Δ is empty where s and t are not unifiable. The first-order fragment consisting of MSL clauses I call *MSL*. Clauses $\Gamma, s \approx t \rightarrow \Delta$ where Γ, Δ are non-empty and s, t are not unifiable could be added to the MSL fragment without changing any of the results. Considering a superposition calculus, it would select $s \approx t$. Since the two terms are not unifiable, no inference will take place on such a clause and the clause will not contribute to the model operator or resolution refutation. In this sense such clauses do not increase the expressiveness of the fragment.

An *atom ordering* $\prec$ is an irreflexive, well-founded, total ordering on ground atoms. Any atom ordering $\prec$ is lifted to literals by representing A and $\neg A$ as multisets $\{A\}$ and $\{A, A\}$, respectively. The multiset extension of the literal ordering induces an ordering on ground clauses. The clause ordering is compatible with the atom ordering; if the maximal atom in C is greater than the maximal atom in D then $D \prec C$. I use $\prec$ simultaneously to denote an atom ordering and its multiset, literal, and clause extensions. For a ground clause set N and clause C, the set $N^{\prec C} = \{D \in N \mid D \prec C\}$ denotes the clauses of N smaller than C. A literal A is called *[strictly] maximal* in a clause $C \lor A$ if and only if there exists a grounding substitution σ such that for all literals B in C, $B\sigma \preceq A\sigma$ $[B\sigma \prec A\sigma]$.

A *precedence* $\prec$ is a strict total ordering on the symbols in $\mathcal{F} \cup \mathcal{P}$. The *lexicographic path ordering* $\prec_{\text{lpo}}$ (LPO) is a term ordering induced by a precedence $\prec$ and defined by: $t \prec_{\text{lpo}} s$ iff (1) $t \in \mathit{var}(s)$ and $t \neq s$, or (2) $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$, and (a) $t \prec_{\text{lpo}} s_i$ for some i, or (b) $g \prec f$ and $t_j \prec_{\text{lpo}} s$ for all j, or (c) $f = g$, $t_j \prec_{\text{lpo}} S$ for all j, and $(s_1, \ldots, s_m)(\prec_{\text{lpo}})_{lex}(t_1, \ldots, t_n)$. The LPO extends to an atom ordering $\prec_{\text{lpo}}$ in the usual way.

A *Herbrand interpretation* $\mathcal{I}$ is a - possibly infinite - set of ground atoms. A ground atom A is called *true* in $\mathcal{I}$ if $A \in \mathcal{I}$ and *false*, otherwise. $\mathcal{I}$ is said to *satisfy* a ground clause $C = \Gamma \rightarrow \Delta$, denoted by $\mathcal{I} \models C$, if $\Delta \cap \mathcal{I} \neq \emptyset$ or $\Gamma \not\subseteq \mathcal{I}$. A non-ground clause C is satisfied by $\mathcal{I}$ if $\mathcal{I} \models C\sigma$ for every grounding substitution σ. A Herbrand interpretation $\mathcal{I}$ is called a *Herbrand model* of N, $\mathcal{I} \models N$, if $\mathcal{I} \models C$ for every $C \in N$. A Herbrand model $\mathcal{I}$ of N is considered *minimal* with respect to set inclusion, i.e., if there is no Herbrand model $\mathcal{I}'$ with $\mathcal{I}' \subset \mathcal{I}$ and $\mathcal{I}' \models N$, then $\mathcal{I}$ is minimal. A set of clauses N is *satisfiable*, if there exists a model that satisfies N.

Otherwise, the set is *unsatisfiable* and in that case, any finite subset of $\mathcal{G}(N)$ that is unsatisfiable is called an unsatisfiability core. An unsatisfiability core is considered *minimal* if none of its strict subsets are unsatisfiable.

The *partial model* $\mathcal{I}_N^{\prec}$ of a clause set N under a given ordering $\prec$ is a specific Herbrand interpretation defined by the rules in Definition 2.0.1[46]. The smallest ground clause $C \in \mathcal{G}(N)$ such that $\mathcal{I}_N^{\prec} \not\models C$, if it exists, is called the minimal false clause of N.

Definition 2.0.1 (Partial Model Construction)**.** *Given a clause set N and an ordering $\prec$, the partial model $\mathcal{I}_N^{\prec}$ for N is constructed inductively as follows:*

$$\mathcal{I}_C^{\prec} := \bigcup_{D \in \mathcal{G}(N)^{\prec C}} \delta_D$$

$$\delta_D := \begin{cases} \{A\} & \text{if } D = \Gamma \rightarrow \Delta, A; \ A \text{ strictly maximal; and } \mathcal{I}_D^{\prec} \not\models D \\ \emptyset & \text{otherwise} \end{cases}$$

$$\mathcal{I}_N^{\prec} := \bigcup_{C \in \mathcal{G}(N)} \delta_C$$

Clauses C with $\delta_C \neq \emptyset$ are called productive.

A clause C is called *redundant* in a clause set N if for every $D \in \mathcal{G}(C)$, there exist $D_1, \ldots, D_n$ in $\mathcal{G}(N)^{\prec D}$ such that $D_1, \ldots, D_n \models D$. A clause C' is called a *condensation* of C if $C' \subset C$ and there exists a substitution σ such that, for all $L \in C$ there is an $L' \in C'$ with $L\sigma = L'$.

Ordered Resolution with selection is the calculus defined by the resolution and factoring inferences (Definitions 2.0.2 and 2.0.3). A *selection function* sel assigns to a clause $\Gamma \rightarrow \Delta$ a possibly empty subset of Γ. For a clause C and selection function sel, the literals in $\text{sel}(C)$ are called selected.

Definition 2.0.2 (Ordered Resolution with Selection)**.**

$$\frac{\Gamma_1 \rightarrow \Delta_1, A \qquad \Gamma_2, B \rightarrow \Delta_2}{(\Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2)\sigma} \ , \textit{if}$$

1. $\sigma = \text{mgu}(A, B)$*;*

2. $A\sigma$ *is strictly maximal in* $\Gamma_1 \rightarrow \Delta_1, A)\sigma$ *and* $\text{sel}(\Gamma_1 \rightarrow \Delta_1, A) = \emptyset$*;*

3. $B \in \text{sel}(\Gamma_2, B \rightarrow \Delta_2)$
   or $\text{sel}(\Gamma_2, B \rightarrow \Delta_2) = \emptyset$ *and* $\neg B\sigma$ *maximal in* $(\Gamma_2, B \rightarrow \Delta_2)\sigma$*.*

Definition 2.0.3 (Ordered Factoring with Selection)**.**

$$\frac{\Gamma \rightarrow \Delta, A, B}{(\Gamma \rightarrow \Delta, A)\sigma} \ , \textit{if}$$

1. $\sigma = \text{mgu}(A, B)$*;*

2. $\mathrm{sel}(\Gamma \to \Delta, A, B) = \emptyset$;

3. *$A\sigma$ is maximal in* $(\Gamma \to \Delta, A, B)\sigma$.

A clause set N is called *saturated up to redundancy* (by ordered resolution with selection), if for every inference between clauses in N the result R is either redundant in N or $\mathcal{G}(R) \subseteq \mathcal{G}(N)$. Ordered Resolution with selection is sound and complete [2], i.e., A clause set N, that is saturated up to redundancy, is unsatisfiable, if and only if $\square \in N$. As a consequence, if N is unsatisfiable there exists a *resolution refutation* of N that produces the empty clause.

Definition 2.0.4 (Resolution Refutation). *A resolution refutation $\mathcal{R}$ of a clause set N is a list of clauses, where the last clause is the empty clause $\square$. Each clause C in $\mathcal{R}$ is annotated as either an input, factoring, or resolution clause. Furthermore, a resolution refutation $\mathcal{R}$ satisfies all of the following three properties:*

(R1) *If $\mathcal{R} = \mathcal{R}_1, C^{Input}, \mathcal{R}_2$, then C is a variant of a clause in N.*

(R2) *If $\mathcal{R} = \mathcal{R}_1, C^{Fac}, \mathcal{R}_2$, then C is a factor of a clause in $\mathcal{R}_1$.*

(R3) *If $\mathcal{R} = \mathcal{R}_1, C^{Res}, \mathcal{R}_2$, then C is a resolvent of two clauses in $\mathcal{R}_1$.*

Note that as a consequence of (R1)-(R3), $N \models C$ for any clause C in a resolution refutation $\mathcal{R}$ of N, including the empty clause at the end of $\mathcal{R}$. The list starting with the clauses in N and extended with every inference made by a resolution solver is a resolution refutation. Additionally, a resolution refutation can be represented as a *resolution refutation tree*. The empty clause is the root, input clauses are leaves, and the factoring and resolution clauses are internal nodes with their parent clauses as children.

Consider as an example the resolution refutation

$$\begin{array}{rrl}
[1 : \mathit{Input}] & & \to P(x,x) \\
[2 : \mathit{Input}] & P(a,y) & \to Q(y,b) \\
[3 : \mathit{Input}] & Q(a,x) & \to \\
[4 : \mathit{Res} : 1,2] & & \to Q(a,b) \\
[5 : \mathit{Res} : 4,3] & & \square
\end{array}$$

The corresponding resolution refutation tree is written as

$$\dfrac{\dfrac{\to P(x,x) \qquad P(a,y) \to Q(y,b)}{\to Q(a,b)} \qquad Q(a,x) \to}{\square}$$

Chapter 3

Approximations on Clauses

In this chapter I introduce several transformations on first-order clauses and discuss their properties. In particular, how the transformations affect satisfiability and models of the clause set after transformation. I also call transformations with the following desired properties approximations.

Definition 3.0.5 (Approximation)**.** *Let $N \Rightarrow N'$ be a relation on the clause sets,*

(1) $\Rightarrow$ *is called an* over-approximation
if satisfiability of N' implies satisfiability of N,

(2) $\Rightarrow$ *is called an* under-approximation
if unsatisfiability of N' implies unsatisfiability of N.

(3) $\Rightarrow$ *is called* satisfiability equivalent
if $\Rightarrow$ is both an over- and under-approximation.

In either case N' is called an approximation of N.

In the following sections, most transformations are defined with rules of the following form where one clause from the original set N is replaced by one or more new clauses.

Rule $\quad N \,\dot{\cup}\, \{C\} \;\Rightarrow_{\mathrm{R}}\; N \cup \{C_1, \ldots, C_n\}$
with some conditions on C and $C_1; \ldots; C_n$.

In this case, $C_1; \ldots; C_n$ are called the *approximation* clauses of the *approximated* or *parent* clause C. In case of a series of transformations $N_0 \Rightarrow_{\mathrm{R}}^{k} N_k$, the transitive closure of the parent relation between approximation and approximated clauses defines the so-called *ancestor* relation, where each approximation clause in N_k has a unique ancestor in N_0.

3.1 Monadic Transformation

The first transformation is called the Monadic transformation. As its name suggests it creates a monadic clause set by removing every occurrence of non-monadic predicates from the clause set. This can be achieved in several ways, for example, simply removing all but one argument for each non-monadic predicate. I, however, take an approach using term-encoding, which is satisfiability equivalent.

Starting from a clause set N the Monadic transformation is parametrized by a single monadic projection predicate T fresh to N and for each non-monadic predicate P a separate projection function f_P which is also fresh to N.

Definition 3.1.1 (Monadic Projection)**.** *Given a non-monadic predicate P, a monadic projection predicate T, and a projection function f_P, define the injective function μ_P^T on atoms as $\mu_P^T(P(\bar{t})) := T(f_p(\bar{t}))$ and $\mu_P^T(Q(\bar{s})) := Q(\bar{s})$ for $P \neq Q$. The function is extended to clauses, clause sets and interpretations. Given a signature Σ with non-monadic predicates $P_1, \ldots, P_n$, define $\mu_\Sigma^T(N) := \mu_{P_1}^T(\ldots(\mu_{P_n}^T(N))\ldots)$ and $\mu_\Sigma^T(\mathcal{I}) := \mu_{P_1}^T(\ldots(\mu_{P_n}^T(\mathcal{I}))\ldots)$.*

Using Definition 3.1.1, the following rule defines the Monadic transformation.

Monadic $\qquad N \Rightarrow_{\mathrm{MO}} \mu_P^T(N)$
provided P is a non-monadic predicate in the signature of N.

For a clause $C \in N$, the clause $\mu_P^T(C) \in \mu_P^T(N)$ is its approximation clause. The first important property of a transformation is *termination*. Termination determines whether it is possible to fully transform a given clause set such that all clauses have a desired property. In this case, the Monadic transformation terminates and a clause set that it cannot be applied on is monadic.

Lemma 3.1.2 (Termination)**.** $\Rightarrow_{\mathrm{MO}}^*$ *is terminating.*

Proof. Let N_0 be a clause set with a signature containing the n non-monadic predicates $P_1, \ldots, P_n$. Then for $N_0 \Rightarrow_{\mathrm{MO}} N_1 = \mu_{P_1}^T(N_0)$, the signature of N_1 contains the $n-1$ non-monadic predicates $P_2, \ldots, P_n$. After n transformations, $N_0 \Rightarrow_{\mathrm{MO}}^n N_n$, the signature of N_n is monadic and the Monadic transformation is finished. □

Next is the question which category of approximation (Definition 3.0.5), if any, the transformation falls into. As mentioned, the Monadic transformation is both an over- and under-approximation and therefore satisfiability equivalent.

Lemma 3.1.3 (Equivalence)**.** $\Rightarrow_{\mathrm{MO}}$ *is satisfiability equivalent.*

Proof. Let $N_0 \Rightarrow_{\mathrm{MO}} N_1 = \mu_P(N_0)$. Then, $N_0 = \mu_P^{-1}(N_1)$. Let $\mathcal{I}$ be a model of N_1 and $C \in N_0$. Since $\mu_P(C) \in N_1$, $\mathcal{I} \models \mu_P(C)$ and thus, $\mu_P^{-1}(\mathcal{I}) \models C$. Hence, $\mu_P^{-1}(\mathcal{I})$ is a model of N_0. Therefore, the Monadic transformation is an over-approximation.

Let $\mathcal{I}$ be a model of N_0 and $D \in N_1$. By construction, there is a $C \in N_0$ such that $\mu_P(C) = D$ Since $\mathcal{I} \models C$, $\mu_P(\mathcal{I}) \models \mu_P(C) = D$. Hence, $\mu_P(\mathcal{I})$ is a model of N_1. Therefore, the Monadic transformation is an under-approximation. □

Lastly, I compare the partial models (Definition 2.0.1) of a clause set and its approximation. Since the partial model depends on the term-ordering, the term-ordering of the approximation has to agree with the one of the original. However, whenever a transformation introduces fresh symbols to the signature, there is some freedom in choosing the term-ordering of the approximation.

In the following, I show that, under certain conditions, the partial model of the Monadic transformation is the 'same' as the partial model of the original.

Definition 3.1.4. *Given an atom ordering $\prec$ and a non-monadic predicate P. Define $\prec^P$ as an ordering that extends $\prec$ such that if $A \prec P(t_1, \ldots, t_n) \prec B$, then $\mu_P(A) \prec^P T(f_p(t_1, \ldots, t_n)) \prec^P \mu_P(B)$.*

For example, in an LPO give T the lowest precedence while f_p inherits the precedence of P.

Lemma 3.1.5. *Given a Monadic transformation $N_0 \Rightarrow_{\mathrm{MO}} N_1 = \mu_P(N_0)$ and an ordering $\prec$. Then, $\mu_P(\mathcal{I}^{\prec}_{N_0}) = \mathcal{I}^{\prec^P}_{N_1}$.*

Proof. Assume $\mu_P(\mathcal{I}^{\prec}_{N_0}) \neq \mathcal{I}^{\prec}_{N_1}$.
Let Q be the minimal atom distinguishing $\mu_P(\mathcal{I}^{\prec}_{N_0})$ and $\mathcal{I}^{\prec}_{N_1}$.

Let $Q \in \mathcal{I}^{\prec}_{N_0}$ and $\mu_P(Q) \notin \mathcal{I}^{\prec}_{N_1}$. By definition, there is a productive ground clause $C \vee Q$ in N and hence, Q is strictly maximal and $\mathcal{I}^{\prec}_{N_0} \not\models C$. By construction, $\mu_P(C \vee Q)$ is a ground clause in N_1 with $\mu_P(Q)$ also strictly maximal, which is not productive as $\mu_P(Q) \notin \mathcal{I}^{\prec}_{N_1}$. Hence $\mathcal{I}^{\prec}_{N_1} \models \mu_P(C)$ and there is a literal $R \prec Q$ such that $\mathcal{I}^{\prec}_{N_1} \models \mu_P(R)$, but $\mathcal{I}^{\prec}_{N_0} \not\models R$. This contradicts with Q being minimal.

Let $Q \in \mathcal{I}^{\prec}_{N_1}$ and $\mu_P^{-1}(Q) \notin \mathcal{I}^{\prec}_{N_0}$. By definition, there is a productive ground clause $C'\sigma = C \vee Q$ in N_1 and hence, Q is strictly maximal and $\mathcal{I}^{\prec}_{N_1} \not\models C$. By construction, $\mu_P^{-1}(C')$ is a clause in N_0.
Assume $\mu_P^{-1}(C')\sigma$ is not a ground clause in N_0. This is only possible if a term substituted by σ contains f_p. However, T is the only predicate or function with an argument of the S_p sort and it always appears by construction with a f_p-term as argument.
Hence, $\mu_P^{-1}(C')\sigma$ is a ground clause in N_0, which is not productive as $\mu_P^{-1}(Q) \notin \mathcal{I}^{\prec}_{N_0}$. Hence $\mathcal{I}^{\prec}_{N_0} \models \mu_P^{-1}(C)$ and there is a literal $R \prec Q$ such that $\mathcal{I}^{\prec}_{N_0} \models \mu_P^{-1}(R)$, but $\mathcal{I}^{\prec}_{N_1} \not\models R$. This contradicts with Q being minimal. □

Using the result of Lemma 3.1.3, I will for simplicity in the following chapters sometimes use non-monadic predicates in a context where only monadic predicates are expected. In that case, a non-monadic atom such as $P(x, y)$ implicitly represents the monadic atom $T(f_P(x, y))$.

3.2 Linear Transformation

The second transformation is called the Linear transformation. Its purpose is to linearise the succedent of each clause in a given clause set by replacing duplicate variables with fresh ones. The Linear transformation is obviously not satisfiability equivalent. For example, the satisfiable clause set $\{\rightarrow P(x,x);\ P(a,b) \rightarrow\}$ is transformed into the unsatisfiable set $\{\rightarrow P(x,x');\ P(a,b) \rightarrow\}$. To avoid this when possible, I additionally duplicate antecedent literals that contain a replaced variable. For example, the satisfiable set $\{\rightarrow S(a);\ S(x) \rightarrow P(x,x);\ P(a,b) \rightarrow\}$ can be approximated with the still satisfiable set $\{\rightarrow S(a);\ S(x'), S(x) \rightarrow P(x,x');\ P(a,b) \rightarrow\}$ instead of the unsatisfiable set $\{\rightarrow S(a);\ S(x) \rightarrow P(x,x');\ P(a,b) \rightarrow\}$.

Linear 1 $\quad N \dot\cup \{\Gamma \rightarrow \Delta, E'[x]_p, E[x]_q\} \Rightarrow_{\mathrm{LI}} N \cup \{\Gamma\sigma, \Gamma \rightarrow \Delta, E'[x]_p, E[q/x']\}$

provided x' is fresh and $\sigma = \{x \mapsto x'\}$.

Linear 2 $\quad N \dot\cup \{\Gamma \rightarrow \Delta, E[x]_{p,q}\} \Rightarrow_{\mathrm{LI}} N \cup \{\Gamma\sigma, \Gamma \rightarrow \Delta, E[q/x']\}$

provided x' is fresh, $p \neq q$ and $\sigma = \{x \mapsto x'\}$.

I consider Linear 1 and Linear 2 as two cases of the same Linear transformation rule. Their only difference is whether the two occurrences of x are in the same or different literals. Further, the duplication of literals in Γ is not needed if x does not occur in them.

Duplicating the antecedent literals can in the worst-case lead to an exponential blow-up of the size of the approximations clause. For example, the clause

$$P(x,y) \rightarrow Q(x,x,x,y,y)$$

is approximated by

$$P(x,y), P(x,y'), P(x',y), P(x',y'), P(x'',y), P(x'',y'), \rightarrow Q(x,x',x'',y,y').$$

In Section 7.1.2, I introduce an optional preprocessing transformation that mitigates this effect.

Further, consider a Linear transformation $N \cup \{C\} \Rightarrow_{\mathrm{LI}} N \cup \{C'\}$, where a fresh variable x' replaces an occurrence of a non-linear variable x in C. Then, $C'\{x' \mapsto x\}$ is equal to C modulo duplicate literal elimination.

As with the Monadic transformation, the Linear transformation terminates. However, as shown by the example, the Linear transformation is only an over-approximation.

Lemma 3.2.1 (Linear Termination). $\Rightarrow^*_{\mathrm{LI}}$ *is terminating.*

Proof. The application of the Linear transformation strictly reduces the number of duplicate variable occurrences in the succedent of a clause. When there are no more duplicate variables, the entire clause set is linear. □

Lemma 3.2.2. $\Rightarrow_{\mathrm{LI}}$ *is an over-approximation.*

Proof. Let $N_0 = N \cup \{\Gamma \rightarrow \Delta[x]_{p,q}\} \Rightarrow_{\mathrm{LI}} N_1 = N \cup \{\Gamma\{x \mapsto x'\}, \Gamma \rightarrow \Delta[x']_q\}$. Let $\mathcal{I}$ be a model of N_1 and C be a ground instance of a clause in N_0. If C is an instance of a clause in N, then $\mathcal{I} \models C$. Otherwise $C = (\Gamma \rightarrow \Delta[x]_{p,q})\sigma$ for some ground substitution σ. Then, $(\Gamma\{x \mapsto x'\}, \Gamma \rightarrow \Delta[x']_q)\{x' \mapsto x\}\sigma = (\Gamma, \Gamma \rightarrow \Delta[x]_q)\sigma$, which equals C modulo duplicate literal elimination. Thus, $\mathcal{I} \models C$ because $\mathcal{I} \models (\Gamma\{x \mapsto x'\}, \Gamma \rightarrow \Delta[x']_q)\{x' \mapsto x\}\sigma$. Hence $\mathcal{I} \models N_0$. Therefore, the Linear transformation is an over-approximation. □

Next, consider the clause set $N := \{\rightarrow P(x,x); \rightarrow P(a,b), Q(b,b)\}$. With an LPO $\prec_{\mathrm{lpo}}$ induced by the precedence $a \prec b \prec P \prec Q$ on the signature, N has the partial model $\mathcal{I}_N^{\prec_{\mathrm{lpo}}} = \{P(a,a), P(b,b), Q(b,b)\}$. The linear approximation $N' = \{\rightarrow P(x,x'); \rightarrow P(a,b), Q(b,b)\}$ has $\mathcal{I}_{N'}^{\prec_{\mathrm{lpo}}} = \{P(a,a), P(a,b), P(b,a), P(b,b)\}$. This example shows that in general $\mathcal{I}_N^{\prec} \not\subseteq \mathcal{I}_{N'}^{\prec}$. However, for Horn clause sets the model inclusion property holds.

Lemma 3.2.3. *Given Horn clause sets N and N′ and an ordering $\prec$. If $N' \Rightarrow_{\mathrm{LI}} N$, then $\mathcal{I}_N^{\prec} \subseteq \mathcal{I}_{N'}^{\prec}$.*

Proof. Assume $\mathcal{I}_N^{\prec} \not\subseteq \mathcal{I}_{N'}^{\prec}$. Then, there exists a minimal atom $A \in \mathcal{I}_N^{\prec}$ with $A \notin \mathcal{I}_{N'}^{\prec}$. By definition, there is a productive ground clause $C\sigma = \Gamma \rightarrow A$ in N and hence, A is strictly maximal in C and $\Gamma \subseteq \mathcal{I}_N^{\prec}$. Let $\sigma' = \sigma[x' \rightarrow x\sigma]$. By construction, there is a clause $C' \in N'$ such that $C'\sigma' \equiv C\sigma$ is a ground clause in N'. If C was linearised, $\Gamma'\{x \rightarrow x'\}\sigma' \subseteq \Gamma$. Otherwise, $C = C'$. As $A \notin \mathcal{I}_{N'}^{\prec}$, $C'\sigma'$ can not be productive. Therefore, there has to be a $B \in \Gamma$ with $B \notin \mathcal{I}_{N'}^{\prec}$. However, since $B \prec A$, this contradicts with A being minimal. □

3.3 Shallow Transformation

As its basic function, the Shallow transformation flattens positive literals by extracting complex subterms and encoding them with a fresh sort predicate and a new clause.

$$\begin{aligned} &N \,\dot\cup\, \{\Gamma \rightarrow E[s]_p, \Delta\} \Rightarrow_{\mathrm{SH}} \\ &N \cup \{S(x), \Gamma \rightarrow E[p/x], \Delta\} \cup \{\Gamma \rightarrow S(s)\} \end{aligned}$$

provided x and S are fresh, s is complex, E is monadic, and $|p| = 2$.

The rule assumes that the clause set N is monadic. Alternatively, one could use an additional rule requiring that $|p| = 1$ for non-monadic literals E. However, this is not necessary by simply applying the Monadic transformation first.

The two clauses $S(x), \Gamma \rightarrow E[p/x], \Delta$ and $\Gamma \rightarrow S(s)$ are respectively called the *left* and *right* approximation clauses named after the side of the S-literals. Their resolvent R on the S-atoms subsumes the original clause, i.e., $R \models \Gamma \rightarrow E[s], \Delta$; though not the other way around. E.g., $\rightarrow P(x, f(x))$ has the approximations

$S(y) \to P(x,y)$ and $\to S(f(x))$. Their resolvent $\to P(x,f(y))$ subsumes $\to P(x,f(x))$ but $\{\to P(x,f(x))\} \not\models \{\to P(x,f(y))\}$.

Since the predicate S is fresh, $\Gamma \to S(s)$ is the only clause where S occurs in a positive literal. Therefore, the variable x in $S(x), \Gamma \to E[p/x], \Delta$ is always instantiated with an instance of s in any resolution proof.

I further generalize the Shallow transformation by allowing the original Γ and Δ to be split between the two approximation clauses.

$$N \dot\cup \{\Gamma \to E[s]_p, \Delta\} \Rightarrow_{\mathrm{SH}}$$
$$N \cup \{S(x), \Gamma_l \to E[p/x], \Delta_l\} \cup \{\Gamma_r \to S(s), \Delta_r\}$$

provided x and S are fresh, s is complex, E is monadic, and $|p| = 2$,
$\Gamma_l \cup \Gamma_r = \Gamma$, $\Delta_l \cup \Delta_r = \Delta$,
$\{Q(y) \in \Gamma \mid y \in \mathrm{vars}(E[p/x], \Delta_l)\} \subseteq \Gamma_l$,
$\{Q(y) \in \Gamma \mid y \in \mathrm{vars}(s, \Delta_r)\} \subseteq \Gamma_r$.

Literals in Γ and Δ can be put in either or both of the approximation clauses, as long as they appear in at least one of them. However, if the original clause is the result of a Shallow transformation itself, the transformation has to keep the literals containing the fresh variable x together. Otherwise, the property that x is an instance of s would be lost.

This free distribution of literals allows me to some extend to minimize and choose the variables shared between the approximated clauses, i.e.,

$$\mathrm{vars}(\Gamma_l, \Delta_l, E[p/x]) \cap \mathrm{vars}(\Gamma_r, \Delta_r, S(s)).$$

Note that if there are no shared variables, the resolvent of the approximated clauses is exactly the original clause which means in this case the approximation is satisfiability equivalent (see Section 3.3).

Lastly, the extraction term s could occur in Γ. In this case, these occurrences can also be extracted at the same time, though, only if the containing literal is put into the approximation clause with the extraction variable. This constitutes the full version of the Shallow transformation rule.

Shallow $\quad N \dot\cup \{\Gamma \to E[s]_p, \Delta\} \Rightarrow_{\mathrm{SH}}$
$$N \cup \{S(x), \Gamma_l \to E[p/x], \Delta_l\} \cup \{\Gamma_r \to S(s), \Delta_r\}$$

provided x and S are fresh, s is complex, E is monadic, and $|p| = 2$,
$\Gamma_l\{x \mapsto s\} \cup \Gamma_r = \Gamma$, $\Delta_l \cup \Delta_r = \Delta$,
$\{Q(y) \in \Gamma \mid y \in \mathrm{vars}(E[p/x], \Delta_l)\} \subseteq \Gamma_l$,
$\{Q(y) \in \Gamma \mid y \in \mathrm{vars}(s, \Delta_r)\} \subseteq \Gamma_r$.

For example, let $N = \{Q(f(x), y) \to P(g(f(x), y))\}$. The simple Shallow transformation $\{S(x'), Q(f(x), y) \to P(g(x', y)); \to S(f(x))\}$ is not satisfiability equivalent – nor with any alternative partition of Γ. However, by replacing the occurrence of the extraction term $f(x)$ in $Q(f(x), y)$ with the fresh variable x', the approximation $\{S(x'), Q(x', y) \to P(g(x', y)); \to S(f(x))\}$ is satisfiability equivalent.

As with the Linear transformation, the Shallow transformation is also a terminating over-approximation.

Lemma 3.3.1 (Shallow Termination). *$\Rightarrow^*_{\mathrm{SH}}$ is terminating.*

Proof. An application of the Shallow transformation strictly reduces the multiset of term depths of the newly introduced clauses compared to the removed parent clause. When there are no more complex terms at depth two, the entire clause set is shallow. □

Lemma 3.3.2. *Shallow transformation is an over-approximation.*

Proof. Let N be a clause set with $C := \Gamma \rightarrow E[s]_p, \Delta$ where the complex term s is extracted. The left and right approximation clauses in the approximation N' are $S(x), \Gamma_l \rightarrow E[p/x], \Delta_l$ and $\Gamma_r \rightarrow S(s), \Delta_r$.

Let I be a Herbrand model of N' and $C\sigma$ be a ground clause of N. Then, I satisfies $(\Gamma_r \rightarrow S(s), \Delta_r)\sigma$ and either $I \not\models \Gamma_r\sigma$, $I \models \Delta_r\sigma$, or $S(s)\sigma \in I$. In the first two cases, $I \models C\sigma$ follows by construction. If $S(s)\sigma \in I$, then because I also satisfies $(S(x), \Gamma_l \rightarrow E[p/x], \Delta_l)\{x \mapsto s\}\sigma$, either $I \not\models \Gamma_l\{x \mapsto s\}\sigma$ or $I \models E[s]_p\sigma, \Delta_l\sigma$. Again, $I \models C\sigma$ follows by construction. Hence, $I \models N$ and therefore, the Shallow transformation is an over-approximation. □

The General Shallow Transformation is a satisfiability equivalent alternative to the Shallow transformation (see Lemma 3.3.3). It keeps track of the shared variables by adding them as additional arguments to the predicate S. Because a Shallow transformation without shared variables is the same as a General Shallow transformation, it shows that in this special case the Shallow transformation is also satisfiability equivalent.

GenShallow $\quad N \mathbin{\dot\cup} \{\Gamma \rightarrow E[s]_p, \Delta\} \Rightarrow_{\mathrm{SH}}$
$\quad N \cup \{S(x, y_1, \ldots, y_m), \Gamma_l \rightarrow E[p/x], \Delta_l\}$
$\quad\quad \cup \{\Gamma_r \rightarrow S(s, y_1, \ldots, y_m), \Delta_r\}$

provided x and S are fresh, s is complex, E is monadic, $|p| \geq 2$,
$\Gamma_l\{x \mapsto s\} \cup \Gamma_r = \Gamma$ and $\Delta_l \cup \Delta_r = \Delta$,
$\{y_1, \ldots, y_m\} := \mathrm{vars}(\Gamma_r, \Delta_r, s) \cap \mathrm{vars}(\Gamma_l, \Delta_l, E[p/x])$,

Note that in general $S(s, y_1, \ldots, y_m)$ is not monadic. The term s is then under an implicit Monadic transformation at depth two and therefore applicable for the Shallow transformation, again. Hence, the General Shallow transformation does not terminate because it makes no progress in flattening the clause set if s was already at depth two.

Lemma 3.3.3. *General Shallow transformation is satisfiability equivalent.*

Proof. Analogous to Lemma 3.3.2, the General Shallow transformation is an over-approximation. Let N be a clause set with $C := \Gamma \rightarrow E[s]_p, \Delta$ where the complex term s is extracted. The left and right approximation clauses in the approximation N' are $C_1 := S(x, y_1, \ldots, y_m), \Gamma_l \rightarrow E[p/x], \Delta_l$ and $C_r := \Gamma_r \rightarrow S(s, y_1, \ldots, y_m), \Delta_r$.

Assume N' is unsatisfiable and let $\mathcal{I}$ be an arbitrary interpretation. Then, there exist ground clauses $C \in \mathcal{G}(N')$ such that $\mathcal{I} \not\models C$. If such a C is not an instance of C_l or C_r, then $C \in \mathcal{G}(N)$ and thus, $\mathcal{I} \not\models N$.

Otherwise, let $C_l\tau_1, \ldots, C_l\tau_m$ and $C_r\rho_1, \ldots, C_r\rho_n$ be all ground clauses of N' false under $\mathcal{I}$. Let

$$\begin{aligned}\mathcal{I}' := \mathcal{I} \setminus \{&S(x, y_1, \ldots, y_m)\tau_1, \ldots, S(x, y_1, \ldots, y_m)\tau_m\}\\ &\cup \{S(s, y_1, \ldots, y_m)\rho_1, \ldots, S(s, y_1, \ldots, y_m)\rho_n\},\end{aligned}$$

i.e., change the truth value for S-atoms such that the clauses unsatisfied under $\mathcal{I}$ are satisfied under $\mathcal{I}'$. Because $\mathcal{I}$ and $\mathcal{I}'$ only differ on S-atoms, there exists a ground clause D of N' that is false under $\mathcal{I}'$ and contains an S-atom. Without loss of generality, let $D = C_l\sigma'$. The proof for $D = C_r\sigma'$ is analogous. Since $\mathcal{I} \models D$, $S(x, y_1, \ldots, y_m)\sigma'$ was added to $\mathcal{I}'$ by some clause $C_r\rho_j$, where $S(s, y_1, \ldots, y_m)\rho_j = S(x, y_1, \ldots, y_m)\sigma'$. Let R be the resolvent of $C_r\rho_j$ and $C_l\sigma'$ on $S(s, y_1, \ldots, y_m)\rho_j$ and $S(x, y_1, \ldots, y_m)\sigma'$. Then, $\mathcal{I} \not\models R$ because $\mathcal{I} \not\models C_r\rho_j$ and $\mathcal{I} \cup \{S(s, y_1, \ldots, y_m)\rho_j\} \not\models C_l\sigma'$. Because R is by construction a ground instance of $C \in N, \mathcal{I} \not\models N$.

Therefore, the General Shallow transformation is an under-approximation. □

The Partial Model Property Just as the Linear transformation, the Shallow transformation also has the property that for Horn clause sets the partial model of the original clause set is a subset of the partial model of the approximation clause set.

Consider the clause set $N = \{\rightarrow P(x);\ P(x) \rightarrow Q(x, f(x));\ \rightarrow Q(a, f(b)), R\}$. Under the lexicographic path ordering $\prec_{\text{lpo}}$ induced by the precedence $a \prec b \prec f \prec P \prec Q \prec R$, $\mathcal{I}_N^{\prec_{\text{lpo}}} = \{R,\ Q(t, f(t)),\ P(t) \mid t \text{ is a term}\}$. The shallow approximation $N' = \{\rightarrow P(x);\ P(x) \rightarrow S(f(x));\ S(x'), P(x) \rightarrow Q(x, x');\ \rightarrow Q(a, f(b)), R\}$ has under $\prec'_{\text{lpo}}$ induced by the precedence $a \prec' b \prec' f \prec' P \prec' S \prec' Q \prec' R$, the partial model $\mathcal{I}_{N'}^{\prec'_{\text{lpo}}} = \{P(t),\ S(f(t)),\ Q(s, f(t))) \mid s, t \text{ are terms}\}$. Hence, for general clause sets and orderings, $\mathcal{I}_N^{\prec} \not\subseteq \mathcal{I}_{N'}^{\prec}$.

As with the Monadic transformation, the extension of the signature by a fresh predicate symbol S requires a corresponding extension of the applied ordering.

Definition 3.3.4. *Given a Horn clause set N and an ordering $\prec$. Let $\Gamma \rightarrow E[s]_p$ be a ground clause in N with $E[s]_p$ maximal. If the clause from which $\Gamma \rightarrow E[s]_p$ is instantiated is transformed by Shallow transformation with extraction term s, then define $\prec'$ as an extension of $\prec$ such that $\Gamma \prec' S(s) \prec' E[s]_p$.*

Lemma 3.3.5. *Given a Horn clause set N and an atom ordering $\prec$. For $N \Rightarrow_{\text{SH}} N'$, $\mathcal{I}_N^{\prec} \subseteq \mathcal{I}_{N'}^{\prec'}$.*

Proof. Assume $\mathcal{I}_N^{\prec} \not\subseteq \mathcal{I}_{N'}^{\prec'}$. Then, there exists a minimal atom $A \in \mathcal{I}_N^{\prec}$ with $A \notin \mathcal{I}_{N'}^{\prec'}$. By definition, there is a productive ground clause $C\sigma = \Gamma \rightarrow A$ in N and hence, A is strictly maximal in $C\sigma$ and $\Gamma \subseteq \mathcal{I}_N^{\prec}$. As $\Gamma \prec A$ and A is the minimal counterexample, $\Gamma \subseteq \mathcal{I}_{N'}^{\prec'}$.
If C is not transformed, then $C\sigma$ is also a ground clause in N' and $C\sigma$ is productive in N', which contradicts $A \notin \mathcal{I}_{N'}^{\prec'}$.
Otherwise, $C\sigma = \Gamma \rightarrow E[s]_p$ with $A = E[s]_p$ and by construction $S(s), \Gamma \rightarrow E[p/s]$ and $\Gamma \rightarrow S(s)$ are ground clauses in N'. As $S(s)$ is strictly maximal in $\Gamma \rightarrow S(s)$ and $\Gamma \subseteq \mathcal{I}_{N'}^{\prec'}$, it is either productive or $S(s)$ was already in $\mathcal{I}_{N'}$. Then also, $S(s), \Gamma \rightarrow E[p/s]$ is productive, which contradicts with $E[p/s] = A \notin \mathcal{I}_{N'}^{\prec'}$. □

3.4 Horn Transformation

The Horn transformation approximate a given non-Horn clause $\Gamma \rightarrow E_1, \ldots, E_n$ with potentially several Horn clauses for each positive literal E_i. For an over-approximation, at least one positive literal needs to be chosen.

Horn $\quad N \,\dot\cup\, \{\Gamma \rightarrow E_1, \ldots, E_n\} \;\Rightarrow_{\mathrm{HO}}\; N \cup \{\Gamma \rightarrow E_{i_1}; \ldots; \Gamma \rightarrow E_{i_k}\}$
provided $n > 1$, $k > 0$ and $1 \leq i_j \leq n$

Lemma 3.4.1 (Horn Termination). *$\Rightarrow_{\mathrm{HO}}^*$ is terminating.*

Proof. An application of the Horn transformation strictly reduces the number of non-Horn clauses. When there are no more non-Horn clauses, the entire clause set is Horn. □

Lemma 3.4.2. *The Horn transformation is an over-approximation.*

Proof. Let N be a clause set containing $C = \Gamma \rightarrow E_1, \ldots, E_n$ and a single step Horn transformation is applied on C returning the set N'. Let $\mathcal{I}$ be a model of N' and $C\sigma$ a ground clause. There is at least one clause $C' = \Gamma \rightarrow E_i \in N'$ with $1 \leq i \leq n$. Since $C'\sigma$ subsumes $C\sigma$, $\mathcal{I} \models C\sigma$. Hence, the Horn transformation is an over-approximation. □

If the Horn transformation is additionally restricted such that it produces approximation clauses for at least each maximal positive literal, the partial model property also holds. I call this the strict Horn transformation.

stHorn $\quad N \,\dot\cup\, \{\Gamma \rightarrow E_1, \ldots, E_n\} \;\Rightarrow_{\mathrm{HO}}\; N \cup \{\Gamma \rightarrow E_{i_1}; \ldots; \Gamma \rightarrow E_{i_k}\}$
provided $n > 1$, $k > 0$, $1 \leq i_j \leq n$, and
if E_i is maximal in $\Gamma \rightarrow E_1, \ldots, E_n$, then $E_i \in \{E_{i_1} \ldots, E_{i_k}\}$

However, consider the example $N = \{\rightarrow P;\ \rightarrow P, Q;\ \rightarrow Q, R\}$ with atom ordering $P \prec Q \prec R$ and $\mathcal{I}_N^{\prec} = \{P, R\}$. For a single Horn transformation step producing $N' = \{\rightarrow P; \rightarrow Q; \rightarrow Q, R\}$, the partial model $\mathcal{I}_{N'}^{\prec} = \{P, Q\}$ is not a superset of

$\mathcal{I}_N^{\prec}$. But, after fully exhausting the strict Horn transformation, $N'' = \{\to P;\ \to R\}$ has the partial model $\mathcal{I}_{N''}^{\prec} = \{P, Q, R\}$. From here on, if not otherwise noted, the Horn transformation will be exhaustive.

Lemma 3.4.3. *Given a clause set N, its strict Horn approximation N' and an ordering $\prec$. Then, $\mathcal{I}_N^{\prec} \subseteq \mathcal{I}_{N'}^{\prec}$.*

Proof. Assume $\mathcal{I}_N^{\prec} \not\subseteq \mathcal{I}_{N'}^{\prec}$. Then, there exists a minimal atom $A \in \mathcal{I}_N^{\prec}$ with $A \notin \mathcal{I}_{N'}^{\prec}$. By definition, there is a productive ground clause $C\sigma = \Gamma \to \Delta, A$ in N and hence, A is strictly maximal in $C\sigma$ and $\Gamma \subseteq \mathcal{I}_{C\sigma}$. By construction, $C'\sigma = \Gamma \to A$ is a ground clause in N' with A also strictly maximal. As $A \notin \mathcal{I}_{N'}^{\prec}$, $C'\sigma$ can not be productive. Therefore, there has to be an atom $B \in \Gamma$ with $B \notin \mathcal{I}_{N'}^{\prec}$. However, $B \prec A$ and $B \in \mathcal{I}_N^{\prec}$, which contradicts with A being minimal. □

Additionally, if only maximal literals are used in the strict Horn transformation, then the partial model of the approximation is the most precise.

Lemma 3.4.4. *Given a clause set N, an ordering $\prec$ and two strict Horn approximations N' and N'' of N where N' only approximates maximal literals. Then, $\mathcal{I}_{N'}^{\prec} \subseteq \mathcal{I}_{N''}^{\prec}$.*

Proof. Assume $\mathcal{I}_{N'}^{\prec} \not\subseteq \mathcal{I}_{N''}^{\prec}$. Then, there exists a minimal atom $A \in \mathcal{I}_{N'}^{\prec}$ with $A \notin \mathcal{I}_{N''}^{\prec}$. By definition, there is a productive ground clause $C\sigma = \Gamma \to A$ in N' and hence, A is strictly maximal in $C\sigma$ and $\Gamma \subseteq \mathcal{I}_{C\sigma}^{\prec}$. By construction, $C\sigma = \Gamma \to A$ is also a ground clause in N''. As $A \notin \mathcal{I}_{N''}^{\prec}$, $C\sigma$ can not be productive. Therefore, there has to be a $B \in \Gamma$ with $B \notin \mathcal{I}_{N''}^{\prec}$. However, $B \prec A$ and $B \in \mathcal{I}_{N'}^{\prec}$, which contradicts with A being minimal. □

Splitting as a Horn Transformation

In some cases splitting can be used as an alternative to the Horn transformation. As opposed to the previous transformations, splitting creates branches that are individually solved. If either of them is satisfiable, the original is satisfiable as well.

Definition 3.4.5 (Splitting). $\dfrac{\Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2}{\Gamma_1 \to \Delta_1 \mid \Gamma_2 \to \Delta_2}$
where $\mathrm{vars}(\Gamma_1 \to \Delta_1) \cap \mathrm{vars}(\Gamma_2 \to \Delta_2) = \emptyset$ *and* $\Delta_1 \neq \emptyset, \Delta_2 \neq \emptyset$.

Lemma 3.4.6. *Splitting is satisfiability equivalent.*

Proof. Let N be a clause set with $C = \Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2 \in N$, which can be split into branches N_1 and N_2.

Let N_1 be satisfiable. As $\Gamma_1 \to \Delta_1$ subsumes C, any satisfying model of N_1 also satisfies N.

Let N_1 and N_2 be unsatisfiable. Then for any model I there are σ_1 and σ_2 such that $(\Gamma_1 \to \Delta_1)\sigma_1$ and $(\Gamma_2 \to \Delta_2)\sigma_2$ is unsatisfied. Therefore I does not satisfy the ground clause $\Gamma_1\sigma_1, \Gamma_2\sigma_2 \to \Delta_1\sigma_1, \Delta_2\sigma_2 = C\sigma_1\sigma_2$ and by extension N. □

3.5 Under-Approximations

While I focus on over-approximation transformations in this thesis, I will briefly discuss possible under-approximations in this section; particularly the deletion and instantiation approximations, where non-MSL clauses are either deleted or replaced with ground instantiations (See also [21]).

Although both transformations trivially preserve satisfiability, the partial model of an approximation is not generally a subset of the original, the property dual to the over-approximation case. This is demonstrated by the following two examples.

Example 3.5.1. *Let $N = \{\rightarrow P, Q\}$ and $N' = \{\rightarrow P, Q; \ \rightarrow P\}$. $\mathcal{I}_N^{\prec} = \{Q\}$ and $\mathcal{I}_{N'}^{\prec} = \{P\}$ with the atom ordering $P \prec Q$. So, $N \subseteq N'$, but $\mathcal{I}_N^{\prec} \nsubseteq \mathcal{I}_{N'}^{\prec}$.*

Example 3.5.2. *Let $N = \{\rightarrow P(x); \ \rightarrow P(a), Q(b); \ \rightarrow P(b), Q(b)\}$. With the atom ordering $\prec_{\text{lpo}}$ induced by the precedence $a \prec b \prec P \prec Q$, $\mathcal{I}_N^{\prec_{\text{lpo}}} = \{P(a), P(b)\}$. Let $N_a = \{\rightarrow P(a); \rightarrow P(a), Q(b); \rightarrow P(b), Q(b)\}$ and $N_b = \{\rightarrow P(b); \rightarrow P(a), Q(b); \rightarrow P(b), Q(b)\}$. Then $\mathcal{I}_{N_a}^{\prec_{\text{lpo}}} = \{P(a), Q(b)\}$ and $\mathcal{I}_{N_b}^{\prec_{\text{lpo}}} = \{P(b), Q(b)\}$. Although N_a and N_b are instantiations of N, $\mathcal{I}_N^{\prec} \nsubseteq \mathcal{I}_{N_a}^{\prec} \nsubseteq \mathcal{I}_N^{\prec}$ and $\mathcal{I}_N^{\prec} \nsubseteq \mathcal{I}_{N_b}^{\prec} \nsubseteq \mathcal{I}_N^{\prec}$.*

When selecting a subset of a clause set as an approximation, the partial model property can be achieved if all productive clauses in the subset are Horn.

Lemma 3.5.3. *Given a clause set N and an ordering $\prec$. Let N' be a clause set such that $\mathcal{G}_\Sigma(N') \subseteq \mathcal{G}_\Sigma(N)$ and every ground clause $C \in \mathcal{G}_\Sigma(N')$ with a positive maximal literal is Horn. Then, $\mathcal{I}_{N'}^{\prec} \subseteq \mathcal{I}_N^{\prec}$.*

Proof. Assume $\mathcal{I}_{N'}^{\prec} \nsubseteq \mathcal{I}_N^{\prec}$. Then, there exists a minimal atom $A \in \mathcal{I}_{N'}^{\prec}$ with $A \notin \mathcal{I}_N^{\prec}$. By definition, there is a productive ground clause $C\sigma = \Gamma \rightarrow \Delta, A$ in N'. As A is strictly maximal in $C\sigma$, Δ is empty and $\Gamma \subseteq \mathcal{I}_{N'}^{\prec}$. By construction, $C\sigma$ is also a ground clause in N with A also strictly maximal. As $A \notin \mathcal{I}_N^{\prec}$, $C\sigma$ can not be productive. Therefore, there has to be a $B \in \Gamma$ with $B \notin \mathcal{I}_N^{\prec}$. However, $B \prec A$ and $B \in \mathcal{I}_{N'}^{\prec}$, which contradicts with A being minimal. □

For example, if all non-Horn clauses with a maximal positive literal are deleted from a clause set N, then for the resulting set N', $\mathcal{I}_{N'}^{\prec} \subseteq \mathcal{I}_N^{\prec}$.

Next, when creating an under-approximation via ground instantiations, the partial model property is generally preserved if all ground clauses are enumerated by the given ordering $\prec$ up to an arbitrary point, i.e. given a clause set N, a ground clause D, and an ordering $\prec$, then $\mathcal{I}_{N'}^{\prec} \subseteq \mathcal{I}_N^{\prec}$ for $N' = \mathcal{G}(N)^{\prec D}$.This can be further refined by starting from an arbitrary ground under-approximation and taking all atoms that could be produced. Then, recursively add all smaller productive ground clauses that could affect the productivity of these atoms.

Definition 3.5.4. *Given clause sets N and N' with $\mathcal{G}_\Sigma(N') \subseteq \mathcal{G}_\Sigma(N)$ and an atom ordering $\prec$. An atom A is called locked if for every clause $\Gamma \rightarrow \Delta, A \in \mathcal{G}_\Sigma(N)$, where A is maximal, $\Gamma \rightarrow \Delta, A \in \mathcal{G}_\Sigma(N')$ and every $B \in \Gamma \cup \Delta$ is locked.*

Lemma 3.5.5. *Given clause sets N and N' with $\mathcal{G}_\Sigma(N') \subseteq \mathcal{G}_\Sigma(N)$ and an ordering $\prec$. If atom A is locked, then $A \in \mathcal{I}_N^\prec$ if and only if $A \in \mathcal{I}_{N'}^\prec$.*

Proof. By induction on $\prec$.

Let $A \in \mathcal{I}_{N'}^\prec$. By construction, there is a clause $\Gamma \to \Delta, A \in \mathcal{G}_\Sigma(N')$ that produced A. As $\mathcal{G}_\Sigma(N') \subseteq \mathcal{G}_\Sigma(N)$, $\Gamma \to \Delta, A \in \mathcal{G}_\Sigma(N)$. By definition, every $B \in \Gamma \cup \Delta$ is locked and $B \prec A$. Hence, by the inductive hypothesis, $B \in \mathcal{I}_N^\prec$ if and only if $B \in \mathcal{I}_{N'}^\prec$, for every $B \in \Gamma \cup \Delta$. Therefore, as $\Gamma \to \Delta, A$ is productive in N', either $\Gamma \to \Delta, A$ is also productive in N or A is already in $\mathcal{I}_N^\prec$.

Let $A \in \mathcal{I}_N^\prec$. By construction, there is a clause $\Gamma \to \Delta, A \in \mathcal{G}_\Sigma(N)$ that produced A. By definition, $\Gamma \to \Delta, A \in \mathcal{G}_\Sigma(N)$ and every $B \in \Gamma \cup \Delta$ is locked and $B \prec A$. Hence, by the inductive hypothesis $B \in \mathcal{I}_N^\prec \leftrightarrow B \in \mathcal{I}_{N'}^\prec$ for every $B \in \Gamma \cup \Delta$. Therefore, as $\Gamma \to \Delta, A$ is productive in N, either $\Gamma \to \Delta, A$ is also productive in N' or A is already in $\mathcal{I}_N^\prec$. □

Lemma 3.5.6. *Given an ordering $\prec$ and clause sets N and N' where $\mathcal{G}_\Sigma(N') \subseteq \mathcal{G}_\Sigma(N)$ and for every clause $\Gamma \to \Delta, A \in \mathcal{G}_\Sigma(N')$ with A maximal, every $B \in \Delta$ is locked. Then, $\mathcal{I}_{N'}^\prec \subseteq \mathcal{I}_N^\prec$.*

Proof. Assume $\mathcal{I}_{N'}^\prec \not\subseteq \mathcal{I}_N^\prec$. Then, there exists a minimal atom $A \in \mathcal{I}_{N'}^\prec$ with $A \notin \mathcal{I}_N^\prec$. By definition, there is a productive clause $C := \Gamma \to \Delta, A$ in $\mathcal{G}_\Sigma(N')$. Hence, A is maximal in C, $\Delta \cap \mathcal{I}_{N'}^\prec$ is empty and $\Gamma \subseteq \mathcal{I}_{N'}^\prec$. As $\mathcal{G}_\Sigma(N') \subseteq \mathcal{G}_\Sigma(N)$ and $A \notin \mathcal{I}_N^\prec$, $C \in \mathcal{G}_\Sigma(N)$ is not productive. Therefore, there has to be either a $B \in \Gamma$ with $B \notin \mathcal{I}_N^\prec$ or a $B \in \Delta$ with $B \in \mathcal{I}_N^\prec$. Let $B \in \Gamma$ with $B \notin \mathcal{I}_N^\prec$. Then, $B \prec A$ and $B \in \mathcal{I}_{N'}^\prec$, which contradicts with A being minimal. Let $B \in \Delta$ with $B \in \mathcal{I}_N^\prec$. As B is locked, $B \in \mathcal{I}_{N'}^\prec$ by Lemma 3.5.5. This contradicts $\Delta \cap \mathcal{I}_{N'}^\prec = \emptyset$. □

3.6 Avoiding Redundant Inferences

Saturation based superposition solvers generate many redundant clauses during their search. Most of its time a solver spends on removing these redundant clauses and still, the clause set tends to grow too large, slowing the search down and eventually running out of space. A method to identify non-redundant inferences is missing.

Obviously, superposition based theorem provers are unable to avoid redundant inferences, but this is also true for other calculi. For example, the instantiation based iProver which uses the Inst-Gen method, where a first order problem is approximated with a SAT problem by replacing every variable by a constant c. The approximation is solved by a SAT solver and its answer is lifted to the original. In case the approximation's satisfying model cannot be lifted, the clash is resolved by appropriately instantiating the involved clauses, which mimics an inference step.

Consider the clause set $N = \{P(x, y) \lor Q(y), \neg P(a, z) \lor R(z), Q(a), R(b)\}$ which has the approximation $N' = \{P(c, c) \lor Q(c),\ \neg P(a, c) \lor R(c),\ Q(a),\ R(b)\}$. A possible satisfying model returned by a SAT solver is $P(c, c)$, $\neg P(a, c)$, $Q(a)$, $R(b)$. This assignment clashes in N as it suggests for example $P(a, a)$ and $\neg P(a, a)$ and hence the implied inference is between $P(x, y) \vee Q(y)$ and $\neg P(a, z) \vee R(z)$. The resolvent $Q(y) \vee R(y)$ is redundant under the signature $\{a, b\}$ as $\{Q(a), R(b)\} \models Q(y) \vee R(y)$.

Theoretically, an inference with the minimal false clause under the partial model is not redundant. Finding this minimal false clause is however undecidable. Therefore, I intended to use an approximation where its partial model includes the partial model of the original clause set. In a decidable fragment, finding the minimal false clause of a clause set N is also decidable. If N is satisfiable there is no need to look for a minimal false clause. Otherwise, enumerate the ground clauses of N by the given ordering and build their partial model until the minimal false clause is found. If this clause is false under the original partial model as well, then the minimal false clause of the original problem can be identified.

An appropriate approximation is defined by the following four rules that were introduced in the previous sections.

Monadic $\quad N \Rightarrow_{\text{MO}} \mu_P^T(N)$

provided P is a non-monadic predicate in the signature of N

stHorn $\quad N \dot\cup \{\Gamma \to E_1, \ldots, E_n\} \Rightarrow_{\text{HO}} N \cup \{\Gamma \to E_{i_1}; \ldots; \Gamma \to E_{i_k}\}$

provided $n > 1$, $k > 0$, $1 \leq i_j \leq n$, and
if E_i is maximal in $\Gamma \to E_1, \ldots, E_n$, then $E_i \in \{E_{i_1} \ldots, E_{i_k}\}$

Shallow $\quad N \dot\cup \{\Gamma \to E[s]_p\} \Rightarrow_{\text{SH}}$
$N \cup \{S(x), \Gamma_l \to E[p/x]; \Gamma_r \to S(s)\}$

provided s is a complex term, $p \neq \varepsilon$, x and S fresh, $\Gamma_l\{x \mapsto s\} \cup \Gamma_r = \Gamma$, $\{Q(y) \in \Gamma \mid y \in \text{vars}(E[p/x])\} \subseteq \Gamma_l$, $\{Q(y) \in \Gamma \mid y \in \text{vars}(S(s))\} \subseteq \Gamma_r$

Linear $\quad N \dot\cup \{\Gamma \to E[x]_{p,q}\} \Rightarrow_{\text{LI}} N \cup \{\Gamma\{x \mapsto x'\}, \Gamma \to E[q/x']\}$

provided x' is fresh and $p \neq q$

Definition 3.6.1 ($\Rightarrow_{\text{pm}}$). *Define $\Rightarrow_{\text{pm}}$ as the priority rewrite system [3] consisting of $\Rightarrow_{\text{MO}}$, $\Rightarrow_{\text{HO}}$, $\Rightarrow_{\text{SH}}$ and $\Rightarrow_{\text{LI}}$ with priority $\Rightarrow_{\text{MO}} > \Rightarrow_{\text{HO}} > \Rightarrow_{\text{SH}} > \Rightarrow_{\text{LI}}$.*

Lemma 3.6.2. *$\Rightarrow_{\text{pm}}$ is a terminating over-approximation and for an exhaustive transformation $N \Rightarrow_{\text{pm}}^* N'$ and term ordering $\prec$, $\mathcal{I}_N^{\prec} \subseteq \mathcal{I}_{N'}^{\prec'}$.*

Proof. Termination follows from Lemmas 3.1.2, 3.2.1, 3.3.1, and 3.4.1 and the fact that neither transformation creates clauses that a higher priority rule could be applied on. $\Rightarrow_{\text{pm}}$ is an over-approximation by Lemmas 3.1.3, 3.2.2, 3.3.2, and 3.4.2 and $\mathcal{I}_N^{\prec}\prec \subseteq \mathcal{I}_{N'}^{\prec'}$ follows from Lemmas 3.1.5, 3.2.3, 3.3.5, and 3.4.3. □

Although I could not find a practical way to apply this method, from an unsuccessful attempt, I derived the idea for the lifting in the approximation-refinement calculi described in the following chapters.

Chapter 4

Monadic Shallow Linear Horn Approximation-Refinement

My first complete calculus is based on approximating into the monadic shallow linear Horn fragment (MSLH) which has a decision procedure given in [45]. If the approximation is satisfiable, so is the original clause set. Otherwise, I take the unsatisfiability proof the approximation and attempt to construct a corresponding proof of the original or refine the approximation and repeating the process.

4.1 Approximation $\Rightarrow_{\text{APH}}$

I introduce the concrete over-approximations $\Rightarrow_{\text{APH}}$ that maps a clause set N to an MSLH clause set N'. These rules of $\Rightarrow_{\text{APH}}$ were previously introduced in Chapter 3 and are adjusted to produce the MSLH fragment.

Monadic $\quad N \ \Rightarrow_{\text{MO}} \ \mu_P^T(N)$
provided P is a non-monadic predicate in the signature of N

Horn $\quad N \,\dot\cup\, \{\Gamma \to E_1, \ldots, E_n\} \ \Rightarrow_{\text{HO}} \ N \cup \{\Gamma \to E_i\}$
provided $n > 1$ and $1 \leq i \leq n$

Shallow $\quad N \,\dot\cup\, \{\Gamma \to E[s]_p\} \ \Rightarrow_{\text{SH}}$
$\quad N \cup \{S(x), \Gamma_l \to E[p/x]; \Gamma_r \to S(s)\}$
provided s is a complex term, $p \neq \varepsilon$, x and S fresh, $\Gamma_l\{x \mapsto s\} \cup \Gamma_r = \Gamma$, $\{Q(y) \in \Gamma \mid y \in \text{vars}(E[p/x])\} \subseteq \Gamma_l$, $\{Q(y) \in \Gamma \mid y \in \text{vars}(S(s))\} \subseteq \Gamma_r$

Linear $\quad N \,\dot\cup\, \{\Gamma \to E[x]_{p,q}\} \ \Rightarrow_{\text{LI}} \ N \cup \{\Gamma\{x \mapsto x'\}, \Gamma \to E[q/x']\}$
provided x' is fresh and $p \neq q$

Definition 4.1.1 ($\Rightarrow_{\text{APH}}$)**.** *Define* $\Rightarrow_{\text{APH}}$ *as the priority rewrite system [3] consisting of* $\Rightarrow_{\text{MO}}$, $\Rightarrow_{\text{HO}}$, $\Rightarrow_{\text{SH}}$ *and* $\Rightarrow_{\text{LI}}$ *with priority* $\Rightarrow_{\text{MO}} > \Rightarrow_{\text{HO}} > \Rightarrow_{\text{SH}} > \Rightarrow_{\text{LI}}$.

Lemma 4.1.2. $\Rightarrow_{\text{APH}}$ *is a terminating over-approximation.*

Proof. Termination follows from Lemmas 3.1.2, 3.2.1, 3.3.1, and 3.4.1 and the fact that neither transformation creates clauses that a higher priority rule could be applied on. $\Rightarrow_{\text{APH}}$ is an over-approximation by Lemmas 3.1.3, 3.2.2, 3.3.2, and 3.4.2. □

In addition to being an over-approximation, if a minimal Herbrand model $\mathcal{I}$ of the approximation exists, it also preserves the skeleton term structure of the original clause set, i.e., if $P(s) \in \mathcal{I}$, then there is an original clause $\Gamma \rightarrow \Delta, P(t)$ such that s is an instance of $\text{skt}(t)$. This property ensures that the refinement always makes progress towards finding a model or a refutation of the original clause set by instantiating its clauses (see Section 4.3).

Lemma 4.1.3. *Let N_0 be a monadic clause set and $N_0 \Rightarrow_{\text{APH}} N_k$ be its approximation. Let N_k be satisfiable and $\mathcal{I}$ be a minimal Herbrand model for N_k. If $P(s) \in \mathcal{I}$ and P is a predicate in N_0, then there exists a clause $C = \Gamma \rightarrow \Delta, P(t) \in N_0$ and a substitution σ such that $s = \text{skt}(t)\sigma$ and for each variable x and predicate S with $C = S(x), \Gamma' \rightarrow \Delta, P(t[x]_p)$ and $s|_p = s''$, $S(s'') \in \mathcal{I}$.*

Proof. By induction on the length of the approximation $N_0 \Rightarrow^*_{\text{APH}} N_k$.

For the base $N_k = N_0$, assume there is no $C \in N_k$ with $C\sigma = \Gamma \rightarrow \Delta, P(s)$ where for each variable x and predicate S with $C = S(x), \Gamma' \rightarrow \Delta, P(t[x]_p)$ and $s|_p = s''$, $S(s'') \in \mathcal{I}$. Then $\mathcal{I} \setminus \{P(s)\}$ is still a model of N_k and therefore $\mathcal{I}$ was not minimal. A contradiction.

Let $N_0 \Rightarrow_{\text{APH}} N_1 \Rightarrow^*_{\text{APH}} N_k$, $P(s) \in \mathcal{I}$ and P is a predicate in N_0 and hence also in N_1. By the induction hypothesis on $N_1 \Rightarrow^*_{\text{APH}} N_k$, there exist a clause $C = \Gamma \rightarrow \Delta, P(t) \in N_1$ and a substitution σ such that $s = \text{skt}(t)\sigma$ and $S(s_2) \in \mathcal{I}$ for each variable y and predicate S with $C = S(y), \Gamma' \rightarrow \Delta, P(t[y]_p)$ and $s|_p = s_2$. The first approximation rule application is either a Linear, a Shallow or a Horn transformation, considered below by case analysis.

Horn Case. Let $\Rightarrow_{\text{APH}}$ be a Horn transformation that replaces $\Gamma'' \rightarrow \Delta', Q(t')$ with $\Gamma'' \rightarrow Q(t')$. If $C \neq \Gamma'' \rightarrow Q(t')$, then $C \in N_0$ already fulfils the claim. Otherwise, $\Gamma \rightarrow \Delta, P(t) = \Gamma'' \rightarrow Q(t')$ and hence $P(t) = Q(t')$ and $\Gamma = \Gamma''$. For $\Gamma'' \rightarrow \Delta', Q(t') \in N_0$, $s = \text{skt}(t)\sigma = skt(t')\sigma$ and $S(s'') \in \mathcal{I}$ for each variable x and predicate S with $S(x) \in \Gamma''$, $Q(t') = Q(t'[x]_p)$ and $s|_p = s_2$.

Linear Case. Let $\Rightarrow_{\text{APH}}$ be a Linear transformation where $C_0 = \Gamma'' \rightarrow \Delta'[x]_{p,q}$ is approximated with $C_1 = \Gamma'', \Gamma''\{x \mapsto x'\} \rightarrow \Delta'[q/x']$. If $C \neq C_1$, then $C \in N_0$ fulfils the claim. Otherwise, $C_0 = \Gamma'' \rightarrow \Delta, P(t)\{x' \mapsto x\} \in N_0$ fulfils the claim because $s = \text{skt}(t)\sigma = \text{skt}(t\{x' \mapsto x\})\sigma$ and $\Gamma'' \subseteq \Gamma'', \Gamma''\{x \mapsto x'\}$.

Shallow Case.
Let $\Rightarrow_{\text{APH}}$ be a Shallow transformation where $C_0 = \Gamma'' \rightarrow E[s_1]_p, \Delta$ is approximated with $C_l = S(x), \Gamma_l \rightarrow E[p/x], \Delta_l$ and $C_r = \Gamma_r \rightarrow S(s_1), \Delta_r$. If $C \neq C_l$ and $C \neq C_2$, then $C \in N_0$ fulfils the claim. If $C = C_2$, then $P(t) \in \Delta_r$ because S is fresh and thus $C_0 \in N_0$ fulfils the claim. Otherwise, $C = C_l$. If $P(t) \neq Q(t'[p/x])$, then $P(t) \in \Delta_l$ and $C_0 \in N_0$ fulfils the claim. Otherwise $C = S(x), \Gamma_l \rightarrow Q(t'[x]_p), \Delta_l$ with $P(t) = Q(t'[p/x])$, $s = \text{skt}(t[x]_p)\sigma$ and $S(s_2) \in \mathcal{I}$ for $s|_p = s_2$. Then by

the induction hypothesis, there exist a clause $C_S = \Gamma_S \rightarrow \Delta_S, S(t_S) \in N_1$ and a substitution σ_S such that $s_2 = \mathrm{skt}(t_S)\sigma_S$ and for each variable y and predicate S' with $C_S = S'(y), \Gamma'_S \rightarrow \Delta_S, P(t_S[y]_q)$ and $s_2|_q = s_3$, $S'(s_3) \in \mathcal{I}$. By construction, $C_S = C_r$. Thus, $s_2 = \mathrm{skt}(s_1)\sigma_S$ and $s = \mathrm{skt}(t[x]_p)\sigma$ imply there exists a σ'' such that $s = \mathrm{skt}(t[p/s_1])\sigma''$. Furthermore, because $\Gamma_l\{x \mapsto s_1\} \cup \Gamma_r = \Gamma''$, if $C_0 = S'(y), \Gamma''' \rightarrow \Delta, P(t[p/s_1])[y]_q$, then either $S'(y) \in \Gamma_l$ and thus $S'(s_4) \in \mathcal{I}_N$, where $s|_q = s_4$, or $S'(y) \in \Gamma_r$ and thus $S'(s_4) \in \mathcal{I}$, where $(s[p/s_2])|_q = s_4$. Hence, $C_0 \in N_0$ fulfils the claim. □

Note that for Lemma 4.1.3 the clause set N_k may contain predicates that have been introduced by the Shallow transformation and are therefore not contained in N_0.

Lemma 4.1.4. *Let $N \Rightarrow^*_{\mathrm{APH}} N'$ such that N' is satisfiable, and $\mathcal{I}$ be a minimal Herbrand model of N'. If $\mu^T_\Sigma(P(s_1, \ldots, s_n)) \in \mathcal{I}$ and P is a predicate in N, then there is a clause $\Gamma \rightarrow \Delta, P(t_1, \ldots, t_n) \in N$ and a substitution σ such that $s_i = \mathrm{skt}(t_i)\sigma$ for all $1 \leq i \leq n$.*

Proof. Because of the rule priority of $\Rightarrow_{\mathrm{APH}}$, $N \Rightarrow^*_{\mathrm{MO}} \mu_\Sigma(N) \Rightarrow^*_{\mathrm{APH}} N'$.

Let $P(s) \in \mathcal{I}$ and P be a monadic predicate in N. Since P is monadic, P is a predicate in $\mu_\Sigma(N)$. Hence by Lemma 4.1.3, there exists a $\Gamma \rightarrow \Delta, P(t) \in \mu_\Sigma(N)$ and a substitution σ such that $s = \mathrm{skt}(t)\sigma$. Then, $\mu_\Sigma^{-1}(\Gamma \rightarrow \Delta, P(t)) \in N$ fulfils the claim.

Let $T(f_p(s_1, \ldots, s_n)) \in \mathcal{I}$. T is monadic and a predicate in $\mu_\Sigma(N)$. Hence by Lemma 4.1.3, there exists a clause $\Gamma \rightarrow \Delta, T(t) \in \mu_\Sigma(N)$ and a substitution σ such that $f_p(s_1, \ldots, s_n) = \mathrm{skt}(t)\sigma$. Therefore, $t = f_p(t_1, \ldots, t_n)$ with $s_i = \mathrm{skt}(t_i)\sigma$ for all i. Then, $\mu_\Sigma^{-1}(\Gamma \rightarrow \Delta, T(f_p(t_1, \ldots, t_n))) \in N$ fulfils the claim. □

Lemma 4.1.4 also holds if satisfiability of N' is dropped and $\mathcal{I}$ is replaced by the superposition partial minimal model operator.

4.2 Lifting of $\Rightarrow_{\mathrm{APH}}$

As the combined transformation $\Rightarrow_{\mathrm{APH}}$ is an over-approximation, if the approximated clause set is satisfiable, so is the original set. However, if it is unsatisfiable I cannot directly infer properties of the original clause set. Instead, I attempt to use the unsatisfiability proof of the approximation to construct an unsatisfiability proof of the original.

Such a proof is generated by the decision procedure in form of a resolution refutation tree or DAG, where each leaf is an input clause, the root is the empty clause, and the internal nodes are the inferred clauses with the premisses of the inference attached as children to the node. Since manipulating such structures can be quite complex, I do not use the resolution refutation directly, but instead first create an unsatisfiable clause set called the conflicting core which is based on unsatisfiability cores.

Given an approximation $N \Rightarrow^*_{\text{AP}} N_k$ and a conflicting core $N_k^\perp$ of N_k, using the lifting lemmas provided in this section I attempt to lift $N_k^\perp$ step-wise to a conflicting core of N. In case of success this shows the unsatisfiability of N. Otherwise, there is an approximation step that cannot be lifted. Then, I refine the approximation, as explained in Section 4.3. Because the lifting follows the steps of the approximation, I show how each transformation rule is lifted, individually.

4.2.1 Conflicting Core

Definition 4.2.1 (Conflicting Core). *A finite set of clauses $N^\perp$ is a conflicting core if $N^\perp\sigma$ is unsatisfiable for all ground substitutions σ of $N^\perp$. A conflicting core $N^\perp$ is a conflicting core of a clause set N if for every $D \in N^\perp$ there is a clause $C \in N$ such that $D = C\sigma$ for some substitution σ. C is then called the* instantiated *clause of D in $N^\perp$. A conflicting core $N^\perp$ of N is minimal if for any strict subset $M \subsetneq N^\perp$, M is not a conflicting core of N.*

For example, let $N = \{\rightarrow P(x, x'); P(y, a), P(z, b) \rightarrow\}$. N is unsatisfiable and a possible ground refutation is resolving $P(b, a), P(a, b) \rightarrow$ with $P(b, a)$ and $P(a, b)$. This corresponds to the conflicting core $N^\perp = \{P(b, a); P(a, b); P(b, a), P(a, b) \rightarrow\}$. Alternatively, ordered resolution might resolve $\rightarrow P(x, x')$ and $P(y, a), P(z, b) \rightarrow$ with substitution $\{x \mapsto y; x' \mapsto a\}$ and then the resolvent and $\rightarrow P(x, x')$ with substitution $\{x \mapsto z; x' \mapsto b\}$. Here, the conflicting core is $N'^\perp = \{P(y, a); P(z, b); P(y, a), P(z, b) \rightarrow\}$. Note that $N'^\perp$ contains the global variables y and z and is more general than $N^\perp$ since $N'^\perp\{y \mapsto b; z \mapsto a\} = N^\perp$.

In other words, a ground conflicting core $N^\perp$ is an unsatisfiability core while a general conflicting core $N^\perp$ is a parametrized unsatisfiability core using globally shared variables. Their advantage is that a single conflicting core can represent an infinite number of unsatisfiability cores and can be further specified using instantiation. The conflict clauses and their global variables in $N^\perp$ are separate from their instantiated clauses in N.

Generating the Conflicting Core

Next, I explain how the conflicting core is generated from a resolution refutation. I assume here that the resolution refutation only uses factorization and resolution inferences. Although in a Horn clause set the factorization rule is never applied, I will here not make use of this restriction and instead describe a general procedure.

Definition 4.2.2 (Generalized Resolution Refutation). *A generalized resolution refutation $\mathcal{R}$ of a clause set N is a list of variable disjoint clauses, where the last clause is the empty clause $\square$. Each clause C in $\mathcal{R}$ is annotated with one of four labels in the following way:*
C^N, $C^{C',\sigma}$, $C^{C_1,C_2,\sigma}$, and C^M which are called input, factor, resolution, and derived clauses, respectively. Additionally, C^* is used as a placeholder for an arbitrary label. Furthermore, a generalized resolution refutation $\mathcal{R}$ satisfies the following

four properties, where (R1)-(R3) correspond to the default properties of a resolution refutation while (R4) is an additional invariant on generalized resolution refutations used in Lemma 4.2.3.

(R1) *If $\mathcal{R} = \mathcal{R}_1, C^N, \mathcal{R}_2$, then C is a variant of a clause in N.*

(R2) *If $\mathcal{R} = \mathcal{R}_1, C^{C',\sigma}, \mathcal{R}_2$, then $C'^* \in \mathcal{R}_1$, and C is a factor of C' using unifier σ.*

(R3) *If $\mathcal{R} = \mathcal{R}_1, C^{C_1,C_2,\sigma}, \mathcal{R}_2$, then $C_1^* \in \mathcal{R}_1$ and $C_2^* \in \mathcal{R}_1$, and C is the resolvent of C_1 and C_2 using unifier σ.*

(R4) *If $\mathcal{R} = \mathcal{R}_1, C^M, \mathcal{R}_2$, then (i) for all $D \in M$, D is an instance of a clause in N, and (ii) for all grounding substitutions δ over* $\mathrm{vars}(N) \cup \mathrm{vars}(C)$*, $M\delta \models C\delta$.*

Note that the clauses in $\mathcal{R}$ are implicitly considered variable disjoint because ordered resolution treats the clauses in N as implicitly variable disjoint. This separates the input clauses in $\mathcal{R}$ from their counterparts in N.

A generalized resolution refutation $\mathcal{R}$ returned by the decision procedure initially contains only input, factor and resolution clauses, which represent each node of the resolution refutation. Derived clauses on the other hand represent nodes that are already processed by the procedure generating the conflicting core. I use the following rules $\Rightarrow_{\mathcal{R}}$ on generalized resolution refutations to replace all clauses in $\mathcal{R}$ with derived clauses:

Input $\qquad \mathcal{R}_1, C^N, \mathcal{R}_2 \;\Rightarrow_{\mathcal{R}}\; \mathcal{R}_1, C^{\{C\}}, \mathcal{R}_2$

Factoring $\qquad \mathcal{R}_1, C^{C',\sigma}, \mathcal{R}_2 \;\Rightarrow_{\mathcal{R}}\; \mathcal{R}_1, C^{M\sigma}, \mathcal{R}_2$
if $C'^M \in \mathcal{R}_1$.

Resolution $\qquad \mathcal{R}_1, C^{C_1,C_2,\sigma}, \mathcal{R}_2 \;\Rightarrow_{\mathcal{R}}\; \mathcal{R}_1, C^{M_1\sigma \cup M_2\sigma}, \mathcal{R}_2$
if $C_1^{M_1} \in \mathcal{R}_1$ and $C_2^{M_2} \in \mathcal{R}_1$.

Lemma 4.2.3. *If $\mathcal{R}$ is a generalized resolution refutation and $\mathcal{R} \Rightarrow_{\mathcal{R}} \mathcal{R}'$, then $\mathcal{R}'$ is also a generalized resolution refutation.*

Proof. Let $\mathcal{R}$ be a generalized resolution refutation for a constrained clause set N.

Let $\mathcal{R} = \mathcal{R}_1, C^N, \mathcal{R}_2 \Rightarrow_{\mathcal{R}} \mathcal{R}_1, C^{\{C\}}, \mathcal{R}_2 = \mathcal{R}'$. We only need to show that $C^{\{C\}}$ satisfies property R4. Trivially, (i) C is a variant of a clause in N (R2) and (ii) $C\delta \models C\delta$ for any δ. Thus $\mathcal{R}'$ is a generalized resolution refutation of N.

Let $\mathcal{R} = \mathcal{R}_1, C^{C_1,C_2,\sigma}, \mathcal{R}_2 \Rightarrow_{\mathcal{R}} \mathcal{R}_1, C^{M_1\sigma \cup M_2\sigma}, \mathcal{R}_2 = \mathcal{R}'$ with $C_1^{M_1}$ and $C_2^{M_2}$ in $\mathcal{R}_1$. We only need to show that $C^{M_1\sigma \cup M_2\sigma}$ satisfies property R4. By property R4, for $i = 1, 2$ (i) for all $D \in M_i$ there exists a $D' \in N$ such that D is an instance of D' and (ii) for all substitution δ_i, $M_i\delta_i \models C_i\delta_i$.

(i) W.l.o.g., let $D \in M_1\sigma$. Then, $D = D_1\sigma$ for some $D_1 \in M_1$ and thus, there is a $D' \in N$ such that D_1 is an instance of D'. Then $D = D_1\sigma$ is also an instance of

D'. (ii) Let δ be a grounding substitution. Then, $M_1\sigma\delta \models C_1\sigma\delta$ and $M_2\sigma\delta \models C_2\sigma\delta$. Therefore, $(M_1\sigma \cup M_2\sigma)\delta \models C$, because $C_1\sigma, C_2\sigma \models C$ by R3. Hence, $\mathcal{R}'$ is a generalized resolution refutation of N.

For $\mathcal{R} = \mathcal{R}_1, C^{C',\sigma}, \mathcal{R}_2 \Rightarrow_{\mathcal{R}} \mathcal{R}_1, C^{M\sigma}, \mathcal{R}_2 = \mathcal{R}'$, the proof works analogously to the previous case. □

Since each step replaces one non-derived clause, $\Rightarrow^*_{\mathcal{R}}$ terminates. Furthermore, one of the rules can always be applied on the left-most non-derived clause in $\mathcal{R}$. Therefore, the end result contains only derived clauses. For the last clause $\square^M$, $M = N^\perp$ is then a conflicting core of N as a direct consequence of property R4.

Note that each clause in $N^\perp$ can be traced back to a unique input clause in the original $\mathcal{R}$. Each input clause is a variant of exactly one clause in N, otherwise N would have contained redundant variants of the same clause. In practice, this clause is the instantiated clause of the conflict clause in $N^\perp$.

Because the lifting is done step-wise on the applied approximation, I show in the following how each individual transformation rule is lifted.

4.2.2 Lifting the Linear Transformation

In order to lift a Linear transformation the remaining and the newly introduced variable need to be instantiated with the same term. For example, consider N and its Linear transformation N'.

$$\begin{array}{rcl} & \rightarrow & P(x,x) \\ P(y,a), P(z,b) & \rightarrow & \end{array} \quad \Rightarrow_{\mathrm{LI}} \quad \begin{array}{rcl} & \rightarrow & P(x,x') \\ P(y,a), P(z,b) & \rightarrow & \end{array}$$

A ground conflicting core of N' is $N^\perp$.

$$\begin{array}{rcl} & \rightarrow & P(a,a) \\ & \rightarrow & P(b,b) \\ P(a,a), P(b,b) & \rightarrow & \end{array}$$

Since $P(a,a)$ and $P(b,b)$ are instances of $P(x,x)$ lifting succeeds and $N^\perp$ is also a unsatisfiability core of N.

Lemma 4.2.4 (Lifting the Linear Transformation). *Let $N_k \Rightarrow_{\mathrm{LI}} N_{k+1}$ be a Linear transformation where $N_k = N \cup \{C\}$, $N_{k+1} = N \cup \{C'\}$, $C = \Gamma \rightarrow E[x]_{p,q}$, and $C' = \Gamma\{x \mapsto x'\}, \Gamma \rightarrow E[q/x']$. Let $N^\perp_{k+1}$ be a conflicting core of N_{k+1} and $C'\sigma_1, \ldots, C'\sigma_m$ be all clauses in $N^\perp_{k+1}$ that are instances of C'. If $x\sigma_j = x'\sigma_j$ for $1 \leq j \leq m$, then $N^\perp_{k+1} \setminus \{C'\sigma_1, \ldots, C'\sigma_m\} \cup \{C\sigma_1, \ldots, C\sigma_m\} = N^\perp_k$ is a conflicting core of N_k.*

Proof. Let σ be any grounding substitution and $\mathcal{I}$ be any interpretation. Then, $\mathcal{I} \not\models N^\perp_{k+1}\sigma$ and there exists a clause $C^\perp \in N^\perp_{k+1}\sigma$ such that $\mathcal{I} \not\models C^\perp$. If $C^\perp$ is not an instance of C', then $C^\perp$ is a clause in $N^\perp_k\sigma$. Thus, $\mathcal{I} \not\models N^\perp_k\sigma$. If $C^\perp$ is an instance of C', then $C^\perp = C'\sigma_j\sigma$ for some $1 \leq j \leq m$. Because $x\sigma_j = x'\sigma_j$, $C'\sigma_j\sigma$ and $C\sigma_j\sigma$ are equal modulo duplicate literal elimination. Thus, $\mathcal{I} \not\models N^\perp_k\sigma$. Therefore, $N^\perp_k$ is a conflicting core of N_k. □

4.2.3 Lifting the Shallow Transformation

A Shallow transformation introduces a new predicate S, which is removed in the lifting step. I take all clauses with S-atoms in the conflicting core and generate any possible resolutions on these S-atoms. The resolvents, which do not contain the S-atom any more, then replace their parent clauses in the core. Lifting succeeds if all resolvents are instances of the original clause in the Shallow transformation.

For example, consider N and its Shallow transformation N'.

$$\begin{array}{rcl}
P(x), Q(y) & \rightarrow & R(g(x, f(y))) \\
 & & \\
R(g(a, f(b))) & \rightarrow & \\
 & \rightarrow & P(a) \\
 & \rightarrow & Q(b)
\end{array}
\quad \Rightarrow_{\mathrm{SH}} \quad
\begin{array}{rcl}
S(x'), P(x) & \rightarrow & R(g(x, x')) \\
Q(y) & \rightarrow & S(f(y)) \\
R(g(a, f(b))) & \rightarrow & \\
 & \rightarrow & P(a) \\
 & \rightarrow & Q(b)
\end{array}$$

A conflicting core of N' is $N'^{\perp}$.

$$\begin{array}{rcl}
S(f(b)), P(a) & \rightarrow & R(g(a, f(b))) \\
Q(b) & \rightarrow & S(f(b)) \\
R(g(a, f(b))) & \rightarrow & \\
 & \rightarrow & P(a) \\
 & \rightarrow & Q(b)
\end{array}$$

By replacing $S(f(b)), P(a) \rightarrow R(g(a, f(b)))$ and $Q(b) \rightarrow S(f(b))$ with their resolvent, $N'^{\perp}$ lifts to a conflicting core of N.

$$\begin{array}{rcl}
P(a), Q(b) & \rightarrow & R(g(a, f(b))) \\
R(g(a, f(b))) & \rightarrow & \\
 & \rightarrow & P(a) \\
 & \rightarrow & Q(b)
\end{array}$$

Lemma 4.2.5 (Lifting the Shallow Transformation)**.** *Let $N_k \Rightarrow_{\mathrm{SH}} N_{k+1}$ be a Shallow transformation where $N_k = N \cup \{C\}$, $N_{k+1} = N \cup \{C_l, C_r\}$, $C = \Gamma \rightarrow E[s]_p$, $C_l = S(x), \Gamma_l \rightarrow E[p/x]$ and $C_r = \Gamma_r \rightarrow S(s)$. Let $N_{k+1}^{\perp}$ be a conflicting core of N_{k+1} and N_S be the set of all resolvents of clauses in $N_{k+1}^{\perp}$ on the S-atom. If all clauses in N_S are instances of C modulo duplicate literal elimination, then $N_k^{\perp} = \{D \in N_{k+1}^{\perp} \mid D$ not an instance of C_l or $C_r\} \cup N_S$ is a conflicting core of N_k.*

Proof. Let σ be any grounding substitution and $\mathcal{I}$ be any interpretation. Then, $\mathcal{I} \not\models N_{k+1}^{\perp}\sigma$ and there exists $C^{\perp} \in N_{k+1}^{\perp}\sigma$ such that $\mathcal{I} \not\models C^{\perp}$. If $C^{\perp}$ is not an instance of C_l or C_r, then $C^{\perp} \in N_k^{\perp}\sigma$. Thus, $\mathcal{I} \not\models N_k^{\perp}\sigma$. Otherwise, assume $C_l\tau_1 \ldots, C_l\tau_m$ and $C_r\rho_1, \ldots, C_r\rho_n$ are the only clauses in $N_k^{\perp}\sigma$ false under $\mathcal{I}$. Let $\mathcal{I}' := \mathcal{I} \setminus \{S(x)\tau_1, \ldots, S(x)\tau_m\} \cup \{S(s)\rho_1, \ldots, S(s)\rho_n\}$, i.e., change the truth value for S-atoms such that the clauses unsatisfied under $\mathcal{I}$ are satisfied under $\mathcal{I}'$. Because $\mathcal{I}$ and $\mathcal{I}'$ only differ on S-atoms, there exists a clause $D \in N_k^{\perp}\sigma$ that is false under $\mathcal{I}'$ and contains an S-atom. Let $D = C_l\sigma'$. Since $\mathcal{I} \models D$, $S(x)\sigma'$ was added to $\mathcal{I}'$ by some clause $C_r\rho_j$, where $S(s)\rho_j = S(x)\sigma'$. Let R be the resolvent of $C_r\rho_j$ and $C_l\sigma'$

on $S(s)\rho_j$ and $S(x)\sigma'$. Then, $\mathcal{I} \not\models R$ because $\mathcal{I} \not\models C_r\rho_j$ and $\mathcal{I} \cup \{S(s)\rho_j\} \not\models C_l\sigma'$. Thus, $\mathcal{I} \not\models N_k^\perp\sigma$. For $D = C_r\sigma'$, the proof is analogous. Therefore, $N_k^\perp$ is a conflicting core of N_k. □

4.2.4 Lifting the Horn Transformation

Because the Horn transformation only keeps a single literal per non-Horn clause, a single conflicting core corresponds to an incomplete proof of the original problem. For example, if the clause $\rightarrow E_1, E_2$ is approximated as E_1 and a proof uses $E_1\sigma$ and $E_1\tau$, then repeating the refutation with $\rightarrow E_1, E_2$ leads to the clause $\rightarrow E_2\sigma, E_2\tau$ instead of $\perp$.

Without loss of generality and to ease the below arguments, I assume that any non-Horn clause has exactly two positive literals: $\Gamma \rightarrow E_1, E_2$. Any clause with more positive literals is preprocessed by renaming pairs of positive literals using fresh predicates.

For the lifting, I require that $\Gamma \rightarrow E_1, E_2$ was approximated both ways resulting in an approximation with $\Gamma \rightarrow E_1$ and another with $\Gamma \rightarrow E_2$. If neither approximation is satisfiable, there exist conflicting cores for both. From each core, all instances $(\Gamma \rightarrow E_1)\tau_i$ and $(\Gamma \rightarrow E_2)\rho_j$ are replaced by a sort of Cartesian product: $\Gamma\tau_i, \Gamma\rho_j \rightarrow E_1\tau_i, E_2\rho_j$. Lifting succeeds if all products are instances of the original clause $\Gamma \rightarrow E_1, E_2$ modulo duplicate literal elimination.

For example, consider N

$$\begin{array}{rcl} & \rightarrow & P(x), Q(y) \\ Q(a), Q(b) & \rightarrow & \\ P(a) & \rightarrow & \end{array}$$

and its two Horn transformations N_1 and N_2.

$$\begin{array}{rcl|rcl} & \rightarrow & P(x) & & \rightarrow & Q(y) \\ Q(a), Q(b) & \rightarrow & & Q(a), Q(b) & \rightarrow & \\ P(a) & \rightarrow & & P(a) & \rightarrow & \end{array}$$

Their respective conflicting cores are $N_1^\perp$ and $N_2^\perp$.

$$\begin{array}{rcl|rcl} & \rightarrow & P(a) & & \rightarrow & Q(a) \\ P(a) & \rightarrow & & & \rightarrow & Q(b) \\ & & & Q(a), Q(b) & \rightarrow & \end{array}$$

By combining them, we get $N^\perp$,

$$\begin{array}{rcl} & \rightarrow & P(a), Q(a) \\ & \rightarrow & P(a), Q(b) \\ Q(a), Q(b) & \rightarrow & \\ P(a) & \rightarrow & \end{array}$$

a conflicting core of N.

Lemma 4.2.6 (Lifting the Horn Transformation)**.** *Let $N_k = N \cup \{C\} \Rightarrow_{\mathrm{HO}} N \cup \{C_1\}$ and $N_k \Rightarrow_{\mathrm{HO}} N \cup \{C_2\}$ where $C = \Gamma \rightarrow E_1, E_2$, $C_1 = \Gamma \rightarrow E_1$ and $C_2 = \Gamma \rightarrow E_2$. Respectively, let $N_1^\perp$ and $N_2^\perp$ be their conflicting cores, and $C_1\tau_1, \ldots, C_1\tau_m$ and $C_2\rho_1, \ldots, C_2\rho_n$ be all clauses in $N_1^\perp$ and $N_2^\perp$ that are instances of C_1 and C_2. Define $N_H = \{\Gamma\tau_i, \Gamma\rho_j \rightarrow E_1\tau_i, E_2\rho_j \mid 1 \leq i \leq m, 1 \leq j \leq n\}$. If every clause in N_H is an instance of C modulo duplicate literal elimination, then $N_k^\perp = N_1^\perp \setminus \{C_1\tau_1, \ldots, C_1\tau_m\} \cup N_2^\perp \setminus \{C_2\rho_1, \ldots, C_2\rho_n\} \cup N_H$ is a conflicting core of N_k.*

Proof. Let σ be any grounding substitution and $\mathcal{I}$ be any interpretation. Then, $\mathcal{I} \not\models N_1^\perp\sigma$ and $\mathcal{I} \not\models N_2^\perp\sigma$. There exist $C_1^\perp \in N_1^\perp$ and $C_2^\perp \in N_2^\perp$ such that $\mathcal{I} \not\models C_1^\perp$ and $\mathcal{I} \not\models C_2^\perp$. If $C_l^\perp$ is not an instance of C_l for some l, then $C_l^\perp \in N_k^\perp\sigma$. Thus, $\mathcal{I} \not\models N_k^\perp\sigma$. Otherwise, $C_1^\perp = C_1\tau_i\sigma$ and $C_2^\perp = C_2\rho_j\sigma$ for some i and j. Then, $\mathcal{I} \not\models (\Gamma\tau_i, \Gamma\rho_j \rightarrow E_1\tau_i, E_2\rho_j)\sigma$, which is in $N_H\sigma$. Thus, $\mathcal{I} \not\models N_k^\perp\sigma$. Therefore, $N_k^\perp$ is a conflicting core of N_k. □

As an alternative to the preprocessing, Lemma 4.2.6 can be generalized such that for a clause $\Gamma \rightarrow E_1, \ldots, E_n$, the cores $N_1^\perp$ to $N_n^\perp$ are combined.

Lastly, there exists a special case, where a single core is sufficient. If the core is a conflicting core for each transformation case, then each instance only needs to be combined with itself, which guarantees a successful lifting. For example, let $N = \{P(a,b) \rightarrow; P(x,b), P(a,y)\}$. Then $N_1 = \{P(a,b) \rightarrow; P(x,b)\}$ and $N_2 = \{P(a,b) \rightarrow; P(a,y)\}$ are Horn transformations of N. $N^\perp = \{P(a,b) \rightarrow; P(a,b)\}$ is a conflicting core of both N_1 and N_2. The lifting to $N^\perp = \{P(a,b) \rightarrow; P(a,b), P(a,b)\}$ is a conflicting core of N.

4.2.5 Lifting the Monadic Transformation

Since the Monadic transformation is satisfiability equivalent, lifting always succeeds by replacing any $T(f_P(t_1, \ldots, t_n))$ atoms in the core by $P(t_1, \ldots, t_n)$. For example, consider N and its Monadic transformation N'.

$$\begin{array}{rlcrl} & \rightarrow P(x,x') & \Rightarrow_{\mathrm{MO}} & & \rightarrow T(f_P(x,x')) \\ P(y,a), P(z,b) & \rightarrow & \Rightarrow_{\mathrm{MO}} & T(f_P(y,a)), T(f_P(z,b)) & \rightarrow \end{array}$$

A conflicting core of N' is $N'^\perp$.

$$\begin{array}{rcl} & \rightarrow & T(f_P(y,a)) \\ & \rightarrow & T(f_P(z,b)) \\ T(f_P(y,a)), T(f_P(z,b)) & \rightarrow & \end{array}$$

Reverting the atoms gives $N^\perp$,

$$\begin{array}{rcl} & \rightarrow & P(y,a) \\ & \rightarrow & P(z,b) \\ P(y,a), P(z,b) & \rightarrow & \end{array}$$

a conflicting core of N.

Lemma 4.2.7 (Lifting the Monadic Transformation)**.** *Let $N \Rightarrow_{\mathrm{MO}} \mu_P(N)$ where P is a non-monadic predicate in N. If $N^{\perp}$ is a conflicting core of $\mu_P(N)$ then $\mu_P^{-1}(N^{\perp})$ is a conflicting core of N.*

Lifting with Instantiation

By definition, if $N^{\perp}$ is a conflicting core of N, then $N^{\perp}\tau$ is also a conflicting core of N for any τ. Sometimes a conflicting core, where no lifting lemma applies, can be instantiated into a core, where one does. This then still implies a successful lifting.

For example, consider N and its Linear transformation N'.

$$\begin{array}{rcl} & \rightarrow & P(x,x) \\ P(y,a), P(z,b) & \rightarrow & \end{array} \Rightarrow_{\mathrm{LI}} \begin{array}{rcl} & \rightarrow & P(x,x') \\ P(y,a), P(z,b) & \rightarrow & \end{array}$$

A possible conflicting core of N' is $N'^{\perp}$.

$$\begin{array}{rcl} & \rightarrow & P(y,a) \\ & \rightarrow & P(b,b) \\ P(y,a), P(b,b) & \rightarrow & \end{array}$$

Because $P(y,a)$ is not an instance of $P(x,x)$, Lemma 4.2.4 is not applicable. However, Lemma 4.2.4 can be applied on $N'^{\perp}\{y \mapsto a\}$.

$$\begin{array}{rcl} & \rightarrow & P(a,a) \\ & \rightarrow & P(b,b) \\ P(a,a), P(b,b) & \rightarrow & \end{array}$$

In practice, this means lifting a conflicting core can fail because some conflict clause C_c is not an instance of its parent clause C, even though C_c and C are unifiable. In this case, I can instantiate the conflicting core with their most general unifier σ. Then, $C_c\sigma$ is an instance of C and no longer causes lifting to fail.

4.3 Approximation-Refinement

In the previous section, I have presented the lifting process for the case that the conflicting core can actually be lifted. If, however, in one of the lifting steps the conditions of the corresponding lemma cannot be met, lifting fails. A failed lifting lemma always means that some clause in the conflicting core, the so-called *lift-conflict*, is not an instance of the parent clause in the respective transformation step. Because the clauses also have overlapping skeleton term structures, there exists at least one variable in the original clause that the approximation instantiates twice in a non-unifiable way. This results in two so-called *conflicting instantiations*.

For example, consider again N and its Linear transformation N'.

$$\begin{array}{rcl} & \rightarrow & P(x,x) \\ P(y,a), P(z,b) & \rightarrow & \end{array} \Rightarrow_{\mathrm{LI}} \begin{array}{rcl} & \rightarrow & P(x,x') \\ P(y,a), P(z,b) & \rightarrow & \end{array}$$

A possible ground conflicting core of N' is

$$\begin{array}{rcl} & \rightarrow & P(a,a) \\ & \rightarrow & P(a,b) \\ P(a,a), P(a,b) & \rightarrow & \end{array}$$

Because $P(x,x)$ and $P(a,b)$ cannot be unified, lifting fails. $P(a,b)$ is the lift-conflict and $\{x \mapsto a\}$ and $\{x \mapsto b\}$ are the conflicting instantiations.

The refinement then always replaces the original clause in N with a set of specific instances, which are determined by the conflicting instantiations.

Definition 4.3.1 (Specific Instances)**.** *Let C be a clause and σ_1 and σ_2 be two substitutions such that the respective literals in $C\sigma_1$ and $C\sigma_2$ cannot be simultaneously unified. Then, the* specific instances *of C with respect to σ_1 and σ_2 are clauses $C\tau_1, \dots, C\tau_n$ such that (i) any ground instance of C is an instance of some $C\tau_i$ and (ii) no $C\tau_i$ shares ground instances with both $C\sigma_1$ and $C\sigma_2$.*

The existence of a finite set of specific instances is guaranteed for conflicting substitutions [29]. The first property ensures that if C is replaced by specific instances, the models of N stay the same. The second property ensures that the approximation of the refined set no longer produces the same conflict because by Lemma 4.1.4 the changes to the skeleton term structure carry over to the approximation.

Continuing from the previous example. The specific instances of $P(x,x)$ are $P(a,a)$ and $P(b,b)$. In the approximation of the refinement N'',

$$\begin{array}{rcl} & \rightarrow & P(a,a) \\ & \rightarrow & P(b,b) \\ P(y,a), P(z,b) & \rightarrow & \end{array}$$

the lift-conflict $P(a,b)$ is not an instance any more. Hence, the previous conflicting core cannot be found again. The refinement loop then restarts with the refined clause set.

Since lifting fails at some step except for Monadic transformation, I describe the refinement for Linear, Shallow and Horn transformation separately.

A Linear transformation enables more instantiations of the approximation clause than the original clause, that is, two variables that were the same can now be instantiated differently.

Definition 4.3.2 (Linear Approximation-Refinement)**.** *Let $N \Rightarrow^k_{\mathrm{APH}} N_k \Rightarrow_{\mathrm{LI}} N_{k+1}$ where $C = \Gamma \rightarrow E[x]_{p,q}$ is approximated by $C_1 = \Gamma\{x \mapsto x'\}, \Gamma \rightarrow E[q/x']$. Let $N^{\perp}_{k+1}$ be a ground conflicting core of N_{k+1} with some lift-conflict $C_1\sigma \in N^{\perp}_{k+1}$ such that $x\sigma$ and $x'\sigma$ cannot be unified. Let $C \in N$ be the ancestor of $C' \in N_{k+1}$. $N \setminus \{C\} \cup \{C\tau_1, \dots, C\tau_n\}$ is the* linear approximation-refinement *of N, where the $C\tau_i$ are the specific instances of C with respect to the conflicting instantiations $\{x \mapsto x\sigma\}$ and $\{x \mapsto x'\sigma\}$.*

The Shallow transformation is similar to the Linear transformation, because it may also result in cases where the same variable is instantiated differently. As mentioned before, the Shallow transformation can always be lifted if the set of shared variables $\text{vars}(\Gamma_2, s) \cap \text{vars}(\Gamma_1, E[p/x])$ is empty. Otherwise, each variable in the intersection potentially introduces instantiations that are not unifiable.

For example, consider N and its Shallow transformation N'.

$$\begin{array}{rclcrcl} & \rightarrow & P(f(x,g(x))) & \Rightarrow_{\text{SH}} & S(z) & \rightarrow & P(f(x,z)) \\ & & & & Q(y) & \rightarrow & S(g(y)) \\ P(f(a,g(b)) & \rightarrow & & & P(f(a,g(b)) & \rightarrow & \end{array}$$

A conflicting core of N' is $N'^{\perp}$.

$$\begin{array}{rcl} S(g(b)) & \rightarrow & P(f(a,g(b))) \\ & \rightarrow & S(g(b)) \\ P(f(a,g(b)) & \rightarrow & \end{array}$$

Because $P(f(a,g(b)))$ and $P(f(x,g(x)))$ cannot be unified, lifting fails. $P(f(a,g(b)))$ is the lift-conflict and $\{x \mapsto a\}$ and $\{x \mapsto b\}$ are the conflicting instantiations. The specific instances of $P(f(x,g(x)))$ are

$$\begin{array}{rcl} & \rightarrow & P(f(f(x,y),g(f(x,y)))) \\ & \rightarrow & P(f(g(x),g(g(x)))) \\ & \rightarrow & P(f(a,g(a))) \\ \text{and} & \rightarrow & P(f(b,g(b))) \end{array}$$

In the approximation of the refinement, the lift-conflict $P(f(a,g(b)))$ can no longer appear in a conflicting core.

Definition 4.3.3 (Shallow Approximation-Refinement)**.** *Let $N \Rightarrow^k_{\text{APH}} N_k \Rightarrow_{\text{SH}} N_{k+1}$ where a clause $C = \Gamma \rightarrow E[s]_p$ is approximated by $C_l = S(x), \Gamma_l \rightarrow E[p/x]$ and $C_r = \Gamma_r \rightarrow S(s)$. Let $N^{\perp}_{k+1}$ be a conflicting core of N_{k+1} with $C_l\sigma_l \in N^{\perp}_{k+1}$ and $C_r\sigma_r \in N^{\perp}_{k+1}$ such that their resolvent C_R is not unifiable with C. Let $C \in N$ be the ancestor of C_l. $N \setminus \{C\} \cup \{C\tau_1, \ldots, C\tau_n\}$ is the* shallow approximation-refinement *of N, where the $C\tau_i$ are the specific instances of C with respect to the substitutions σ_l and σ_r.*

The Horn approximation-refinement works in an analogous way to the Shallow approximation-refinement with the difference that the two clauses come from two conflicting cores instead of one. Similarly, if the set of common variables $\text{vars}(E_1) \cap \text{vars}(E_2)$ is not empty, each variable in the intersection potentially introduces instantiations that are not liftable.

For example, consider N

$$\begin{array}{rcl} & \rightarrow & P(x), Q(x) \\ Q(a), Q(b) & \rightarrow & \\ P(a) & \rightarrow & \\ P(b) & \rightarrow & \end{array}$$

and its two Horn transformations N_1 and N_2.

$$\begin{array}{rcl|rcl} & \rightarrow & P(x) & & \rightarrow & Q(x) \\ Q(a), Q(b) & \rightarrow & & Q(a), Q(b) & \rightarrow & \\ P(a) & \rightarrow & & P(a) & \rightarrow & \\ P(b) & \rightarrow & & P(b) & \rightarrow & \end{array}$$

Their respective conflicting cores are $N_1^\perp$ and $N_2^\perp$.

$$\begin{array}{rcl|rcl} & \rightarrow & P(a) & & \rightarrow & Q(a) \\ P(a) & \rightarrow & & & \rightarrow & Q(b) \\ & & & Q(a), Q(b) & \rightarrow & \end{array}$$

Because the product $\rightarrow P(a), Q(b)$ cannot be unified with $\rightarrow P(x), Q(x)$, lifting fails. $\rightarrow P(a), Q(b)$ is the lift-conflict and $\{x \mapsto a\}$ and $\{x \mapsto b\}$ are the conflicting instantiations. The specific instances of $\rightarrow P(x), Q(x)$ are $\rightarrow P(a), Q(a)$ and $\rightarrow P(b), Q(b)$. In the approximation of the refinement, the lift-conflict $\rightarrow P(a), Q(b)$ is not a possible product any more.

Definition 4.3.4 (Horn Approximation-Refinement). *Let $N \Rightarrow^k_{\text{APH}} N_k = N' \cup \{C\}$, $N_k \Rightarrow_{\text{HO}} N' \cup \{C_1\}$ and $N_k \Rightarrow_{\text{HO}} N' \cup \{C_2\}$ where $C = \Gamma \rightarrow E_1, E_2$; $C_1 = \Gamma \rightarrow E_1$; and $C_2 = \Gamma \rightarrow E_2$. Let $N_1^\perp$ and $N_2^\perp$ be conflicting cores of each case respectively with $C_1\sigma_1 \in N_1^\perp$ and $C_2\sigma_2 \in N_2^\perp$ such that their product is not unifiable with C. Let $C \in N$ be the ancestor of C_1 and C_2. The* Horn approximation-refinement *of N is the clause set $N \setminus \{C\} \cup \{C\tau_1, \ldots, C\tau_n\}$ where the $C\tau_i$ are the specific instances of C with respect to the substitutions σ_1 and σ_2.*

Lastly, there is a possibility that lifting fails but the respective refinement does not apply. This is the case when a lifting lemma fails because some lift-conflict C_c is not an instance of its parent clause C, but C and C_c are nevertheless unifiable. In this case, I apply lifting by instantiation by applying the most general unifier σ of C and C_c to the conflicting core $N^\perp$. In the new conflicting core $N^\perp\sigma$, $C_c\sigma$ is no longer a lift-conflict.

Theorem 4.3.5 (Static Completeness). *Let N_0 be an unsatisfiable clause set and N_k its MSLH approximation. Then, there exists a conflicting core of N_k that can be lifted to N_0.*

Proof. by induction on the number k of approximation steps. The case $k = 0$ is obvious. For $k > 0$, let $N_0 \Rightarrow^{k-1}_{\text{APH}} N_{k-1} \Rightarrow_{\text{AP}} N_k$. By the inductive hypothesis, there is a conflicting core $N_{k-1}^\perp$ of N_{k-1} which can be lifted to N_0. The final approximation rule application is either a Linear, a Shallow, a Horn or a Monadic transformation, considered below by case analysis.

Linear Case. Let $N_{k-1} = N' \cup \{C\} \Rightarrow_{\text{LI}} N_k = N' \cup \{C'\}$ with $C = \Gamma \rightarrow E[x]_{p,q}$ and $C' = \Gamma\{x \mapsto x'\}, \Gamma \rightarrow E[q/x']$. Let $C\sigma_1, \ldots, C\sigma_n$ be the instances of C in $N_{k-1}^\perp$. $N_{k-1}^\perp \setminus \{C\sigma_1, \ldots, C\sigma_n\} \cup \{C'\{x' \mapsto x\}\sigma_j \mid 1 \leq j \leq n\}$ is a conflicting core of N_k. By

Lemma 4.2.4, it can be lifted to $N^{\perp}_{k-1}$ and by the inductive hypothesis, can be lifted to a conflicting core of N_0.

Shallow Case. Let $N \Rightarrow^{*}_{\text{AP}} N_{k-1} = N' \cup \{C\} \Rightarrow_{\text{SH}} N_k = N' \cup \{C_l, C_r\}$ with $C = \Gamma \rightarrow E[s]_p$, $C_l = S(x), \Gamma_l \rightarrow E[p/x]$ and $C_r = \Gamma_r \rightarrow S(s)$. Assume $C\sigma$ is the only instance of C in $N^{\perp}_{k-1}$. $N^{\perp}_{k} = N^{\perp}_{k-1} \setminus \{C\sigma\} \cup \{C_l\{x \mapsto s\}\sigma, C_r\sigma\}$ is a conflicting core of N_k. By Lemma 4.2.5, $N^{\perp}_{k}$ can be lifted to $N^{\perp}_{k-1}$. Now, let $C\sigma_1, \ldots, C\sigma_n$ be the instances of C in $N^{\perp}_{k-1}$ with $n > 1$. Let $C_0 \in N_0$ be the ancestor of C and $N'_0 = N_0 \setminus \{C_0\} \cup \{C_0\sigma_1, \ldots, C_0\sigma_n\}$. $N^{\perp}_{k-1}$ is also a conflicting core for the corresponding approximation $N'_k = N' \cup \{C\sigma_1, \ldots, C\sigma_n\}$. For each $C\sigma_i$, $N^{\perp}_{k-1}$ contains only one instance such that the above case applies. Thus, there is a core for approximation of each $C\sigma_i$ that can be lifted to $N^{\perp}_{k-1}$.

Horn Case. Let $N_{k-1} = N' \cup \{C\}$ with $C = \Gamma \rightarrow E_1, E_2$. Assume $C\sigma$ is the only instances of C in $N^{\perp}_{k-1}$. $N^{\perp}_{k_1} = N^{\perp}_{k-1} \setminus \{C\sigma\} \cup \{(\Gamma \rightarrow E_1)\sigma\}$ is a conflicting core of $N_{k_1} = N' \cup \{(\Gamma \rightarrow E_1)\}$. $N^{\perp}_{k_2}$, constructed analogously for $\Gamma \rightarrow E_2$, is a conflicting core of N_{k_2}. By Lemma 4.2.6, $N^{\perp}_{k_1}$ and $N^{\perp}_{k_2}$ can be lifted to $N^{\perp}_{k-1}$. Now, let $C\sigma_1, \ldots, C\sigma_n$ be the instances of C in $N^{\perp}_{k-1}$ with $n > 1$. Let $C_0 \in N_0$ be the ancestor of C and $N'_0 = N_0 \setminus \{C_0\} \cup \{C_0\sigma_1, \ldots, C_0\sigma_n\}$. $N^{\perp}_{k-1}$ is also a conflicting core for the corresponding approximation $N'_k = N' \cup \{C\sigma_1, \ldots, C\sigma_n\}$. For each $C\sigma_i$, $N^{\perp}_{k-1}$ contains only one instance such that the above case applies. Thus, there are cores for each possible approximation of the $C\sigma_i$ such that they can be lifted to $N^{\perp}_{k-1}$.

Monadic Case. Let $N_{k-1} \Rightarrow_{\text{MO}} N_k = \mu^{T}_{P}(N_{k-1})$ where P is a non-monadic predicate in N_{k-1}. $N^{\perp}_{k} = \mu^{T}_{P}(N^{\perp}_{k-1})$ is a conflicting core of N_k. By Lemma 4.2.7, $N^{\perp}_{k}$ can be lifted to $N^{\perp}_{k-1}$. □

The above lemma considers static completeness, i.e., it does not tell how the conflicting core that can eventually be lifted is found. One way is to enumerate all resolution refutations of N_k in a fair way. A straightforward fairness criterion is to enumerate the refutations by increasing term depth of the clauses used in the refutation. Since the decision procedure on the MSLH fragment generates only finitely many different non-redundant clauses not exceeding a concrete term depth with respect to renaming, eventually the liftable refutation will be generated.

Chapter 5

Monadic Shallow Linear Approximation-Refinement

Motivated by the automatic analysis of security protocols, the monadic shallow linear Horn (MSLH) fragment was shown to be decidable in [45]. The fragment can be finitely saturated by superposition (ordered resolution with selection) where negative literals with non-variable arguments are always selected.

Although from a complexity point of view, the difference between Horn clause fragments and the respective non-Horn clause fragments is typically reflected by membership in the deterministic vs. the non-deterministic respective complexity class, for monadic shallow linear clauses so far there was no decidability result for the non-Horn case. From a security protocol application point of view, non-Horn clauses enable a natural representation of non-determinism.

Furthermore, in the approximation-refinement calculus based on the MSLH fragment, lifting of the Horn transformation requires conflicting cores for each positive literal in the approximated clause (see Lemma 4.2.6). This leads to a worst-case exponential blow-up in the number of non-Horn clauses in the original clause set.

For example consider the non-Horn clauses set $N = \{\rightarrow P_1, P_2, P_3;\ \rightarrow Q_1, Q_2;\ P_1, Q_1 \rightarrow;\ P_1, Q_2 \rightarrow;\ P_2, Q_1 \rightarrow;\ P_2, Q_2 \rightarrow;\ P_3, Q_1 \rightarrow;\ P_3, Q_2 \rightarrow;\ \}$. As shown in Figure 5.1, just these two clauses create six unique approximations. Since in this example each set is its own minimal conflicting core, the lifting requires the conflicting cores of each approximation to create a conflicting core of N.

The results of this chapter close the gap. I show that the monadic shallow linear non-Horn (MSL) fragment is also decidable by ordered resolution with selection. This drastically reduces the complexity of the approximation-refinement calculus, where the non-determinism is more efficiently handled by the superposition solver.

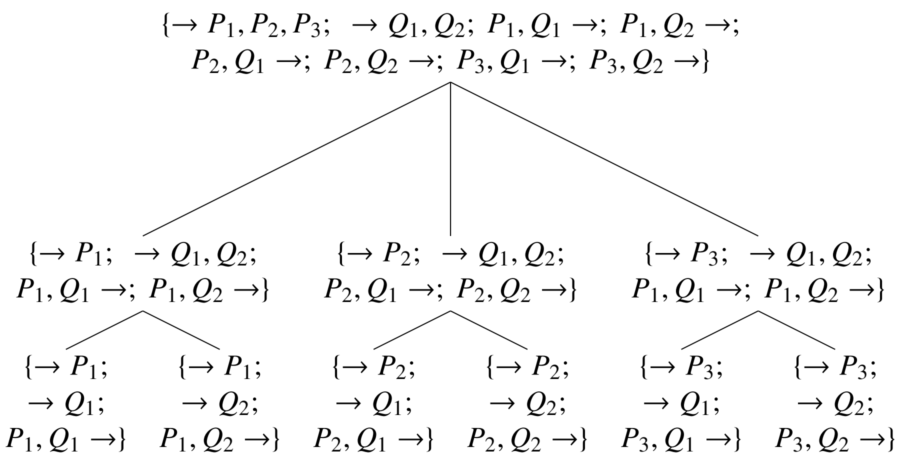

$\{\rightarrow P_1, P_2, P_3;\ \rightarrow Q_1, Q_2;\ P_1, Q_1 \rightarrow;\ P_1, Q_2 \rightarrow;$
$P_2, Q_1 \rightarrow;\ P_2, Q_2 \rightarrow;\ P_3, Q_1 \rightarrow;\ P_3, Q_2 \rightarrow\}$
$\{\rightarrow P_1;\ \rightarrow Q_1, Q_2;\ P_1, Q_1 \rightarrow;\ P_1, Q_2 \rightarrow\}$
$\{\rightarrow P_2;\ \rightarrow Q_1, Q_2;\ P_2, Q_1 \rightarrow;\ P_2, Q_2 \rightarrow\}$
$\{\rightarrow P_3;\ \rightarrow Q_1, Q_2;\ P_1, Q_1 \rightarrow;\ P_1, Q_2 \rightarrow\}$
$\{\rightarrow P_1;\ \rightarrow Q_1;\ P_1, Q_1 \rightarrow\}$
$\{\rightarrow P_1;\ \rightarrow Q_2;\ P_1, Q_2 \rightarrow\}$
$\{\rightarrow P_2;\ \rightarrow Q_1;\ P_2, Q_1 \rightarrow\}$
$\{\rightarrow P_2;\ \rightarrow Q_2;\ P_2, Q_2 \rightarrow\}$
$\{\rightarrow P_3;\ \rightarrow Q_1;\ P_3, Q_1 \rightarrow\}$
$\{\rightarrow P_3;\ \rightarrow Q_2;\ P_3, Q_2 \rightarrow\}$

Figure 5.1: Horn transformation example.

5.1 Decidability of the MSL Fragment

In this section, I generalize the decidability of MSLH to the MSL fragment. Since MSLH is a special case of MSL, the new decision procedure works for both fragments. The procedure in question is ordered resolution with selection [46].

The ordering has to be chosen in such a way that a literal $\neg Q(s)$ is smaller than a literal $P(t[s]_p)$, where p is not the top position. For example, an LPO with a precedence where functions are larger than predicates has this property.

As the selection function, I define a specific function sel.

Definition 5.1.1 (sel)**.** *Let $C = S_1(t_1), \ldots, S_n(t_n) \rightarrow P_1(s_1), \ldots, P_m(s_m)$ be an MSL clause. The Superposition Selection function* sel *is defined by $S_i(t_i) \in \mathrm{sel}(C)$ if*

(1) *t_i is not a variable or*

(2) *$t_1, \ldots, t_n$ are variables and $t_i \notin vars(s_1, \ldots, s_m)$ or*

(3) *$\{t_1, \ldots, t_n\} \subseteq vars(s_1, \ldots, s_m)$ and for some $1 \leq j \leq m$, $s_j = t_i$.*

The selection function sel (Definition 5.1.1) ensures that a clause $\Gamma \rightarrow \Delta$ can only be resolved on a positive literal if Γ contains only variables, which also appear in Δ at a non-top position. For example, in the clause

$$P(f(x)), P(x), Q(z) \rightarrow Q(x), R(f(y))$$

sel selects $P(f(x))$ because $f(x)$ is not a variable. In

$$P(x), Q(z) \rightarrow Q(x), R(f(y))$$

sel selects $Q(z)$ because rule (1) does not apply and $z \notin \{x, y\}$. In

$$P(x), Q(y) \rightarrow Q(x), R(f(y))$$

sel selects $P(x)$ because rules (1) and (2) do not apply and the argument term of $P(x)$ is also the argument of $Q(x)$. Then in

$$P(x), Q(y) \to Q(f(x)), R(f(y))$$

nothing is selected as no rule applies.

Furthermore, if no negative literal is selected, then only positive literals are maximal (Lemma 5.1.2).

Lemma 5.1.2. *Let $C = S_1(t_1), \ldots, S_n(t_n) \to P_1(s_1), \ldots, P_m(s_m)$ be an MSL clause. If some $S_i(t_i)$ is maximal in C, then at least one literal in C is selected.*

Proof. Assume the literal $S_i(t_i)$ is maximal in C, but no literal is selected by sel. Then, by Definition 5.1.1 the term t_i is a variable (1), which appears in some s_j (2) and $s_j \neq t_i$ (3). Hence, $s_j = f(\ldots, t_i, \ldots)$ for some function f and therefore $S_i(t_i) \prec P_j(f(\ldots, t_i, \ldots))$. A contradiction. □

The following lemmas show that saturating an MSL clause set via ordered resolution with selection function sel terminates. This consists of two parts. First, any result of a resolution inference either has lower term depths than the parent clauses or has a certain form that signifies a base case (Lemma 5.1.5). Second, this base case is finite under a certain notion of redundancy (Lemma 5.1.4).

Definition 5.1.3 (Closed Set)**.** *A clause set N is called closed under renaming, if for every clause $C \in N$ and non-trivial variable renaming τ, $C\tau \notin N$. N is called closed under condensation, if for every clause $C \in N$, no condensation exists.*

Lemma 5.1.4. *Let N be a set of MSL clauses, where for every clause $C \in N$,*

$$(1)\ \ C = S_1(x_1), \ldots, S_n(x_n), S'_1(t), \ldots, S'_m(t) \to \Delta \quad \textit{or}$$

$$(2)\ \ C = S_1(x_1), \ldots, S_n(x_n), S'_1(t), \ldots, S'_m(t) \to \Delta, S(t)$$

with $\mathrm{vars}(t) \cap \mathrm{vars}(\Delta) = \emptyset$ *and t shallow and linear. If N is closed under renaming and condensation, then N is finite.*

Proof. Let $C = \Gamma \to \Delta \in N$. Since $\mathrm{vars}(t) \cap \mathrm{vars}(\Delta) = \emptyset$ and all top level terms in Γ are either a variable or t, C can be partitioned into variable disjoint parts:

$$\begin{array}{lr}
(A) & S_1(x), \ldots, S_k(x) \to S(x) \\
(B) & S_1(x_1), \ldots, S_k(x_k), S'_1(t), \ldots, S'_n(t) \to S(t) \\
(C) & S_1(x_1), \ldots, S_k(x_k), S'_1(t), \ldots, S'_n(t) \to
\end{array}$$

where $t = f(y_1, \ldots, y_m)$ is linear and $\{x_1, \ldots, x_k\} \subseteq \{y_1, \ldots, y_m\}$.

Since the number of predicates, function symbols, and their ranks is finite, the number of possible $S(f(y_1, \ldots, y_m))$ different up to renaming is finite. For a given t, there exist only finitely many clauses of the form $S_1(t), \ldots, S_n(t) \to S(t)$ or $S_1(t), \ldots, S_n(t) \to$ modulo condensation. For a fixed set of variables $x_1, \ldots, x_k$, there exist only finitely many clauses of the form $S_1(x_1), \dots, S_k(x_k) \to$ modulo condensation. Therefore, for each form (A)-(C) there are only finitely many distinct clauses modulo renaming and condensation.

Lastly, under renaming and condensation there are only finite possible combinations to construct distinct clauses of type (1) and (2) out of parts (A)-(C). □

Lemma 5.1.5 (Saturation)**.** *Let N be a monadic positive shallow linear first-order clause set. Then N can be finitely saturated under subsumption and condensation by ordered resolution with selection function* sel.

Proof. Let $C = \Gamma \to S(t), \Delta$ and $D = S(t'), \Gamma' \to \Delta'$ be clauses in N where ordered resolution with selection function sel can be applied on $S(t)$ and $S(t')$ respectively. Since N is positive shallow linear, all terms in Δ and Δ' are shallow and all variables are linear. For C this implies that no literal in Γ is selected and hence C can match only one of two patterns:

(A) $C = S_1(x_1), \dots, S_n(x_n) \to \Delta, S(f(y_1, \dots, y_k))$
where $x_1, \dots, x_n$ are variables in $\{y_1, \dots, y_k\} \cup \mathrm{vars}(\Delta)$.

(B) $C = S_1(x_1), \dots, S_k(x_k) \to \Delta, S(y)$
where $x_1, \dots, x_n$ are variables in $\mathrm{vars}(\Delta)$, i.e., y occurs only once.

In D, by Lemma 5.1.2 $S(t')$ is selected by sel and therefore D can match only one of three patterns.

(1) $D = S(f(t_1, \dots, t_k)), \Gamma' \to \Delta'$.

(2) $D = S(y'), \Gamma' \to \Delta'$
where Γ' has no function terms and $y \notin \mathrm{vars}(\Delta')$.

(3) $D = S(y'), \Gamma' \to S'(y'), \Delta'$
where Γ' has no function terms.

This means that any application of ordered resolution with selection sel results in one of the following six resolvents R:

$(A1)$ $R = S_1(x_1)\sigma, \dots, S_n(x_n)\sigma, \Gamma' \to \Delta, \Delta'$
with $\sigma = \{y_1 \mapsto t_1, \dots\}$ and $\Delta\sigma = \Delta$ because $\{y_1, \dots, y_k\} \cap \mathrm{vars}(\Delta) = \emptyset$.

$(B1)$ $R = S_1(x_1), \dots, S_n(x_n), \Gamma' \to \Delta, \Delta'$
The substitution $\{y \mapsto f(t_1, \dots, t_k)\}$ is irrelevant since $y \neq x_i$ and $y \notin \mathrm{vars}(\Delta)$

$(A2)$ $R = S_1(x_1), \dots, S_n(x_n), \Gamma'\tau \to \Delta, \Delta'$
with $\tau = \{y' \mapsto f(y_1, \dots, y_k)\}$ and $\Delta'\tau = \Delta'$ because $y' \notin \mathrm{vars}(\Delta')$.

$(B2)$ $R = S_1(x_1), \dots, S_n(x_n), \Gamma' \to \Delta, \Delta'$.

$(A3)$ $R = S_1(x_1), \dots, S_n(x_n), \Gamma'\tau \to S'(f(y_1, \dots, y_k)), \Delta, \Delta'$
with $\tau = \{y \mapsto f(y_1, \dots, y_k)\}$ and $\Delta'\tau = \Delta'$ because $y' \notin \mathrm{vars}(\Delta')$.

$(B3)\quad R = S_1(x_1), \dots, S_n(x_n), \Gamma' \rightarrow S'(y'), \Delta, \Delta'.$

In each case the resolvent is again monadic positive shallow linear. Furthermore, in the (A2) to (B3) cases

$$R = S_1(x_1), \dots, S_l(x_l), S'_1(t), \dots, S'_m(t) \rightarrow \Delta \quad \text{or}$$

$$R = S_1(x_1), \dots, S_l(x_l), S'_1(t), \dots, S'_m(t) \rightarrow \Delta, S(t)$$

with $t = f(y_1, \dots, y_k)$. By Lemma 5.1.4, there are only finitely many such clauses modulo renaming and condensation. Therefore, these four cases can only be applied finitely many times by ordered resolution.

In the (A1) and (B2) cases, the multiset of term depths of the resolvent's negative literals is strictly smaller than for the right parent. In both, the Γ is the same between the right parent and the resolvent. Only the $f(t_1, \dots, t_k)$ term is replaced by $x_1\sigma, \dots, x_n\sigma$ and $x_1, \dots, x_n$ respectively. In the first case, the depth of the $x_i\sigma$ is either zero if $x_i \notin \{y_1, \dots, y_k\}$ or at least one less than $f(t_1, \dots, t_k)$ since $x_i\sigma = t_i$. In the second case, the x_i have depth zero which is strictly smaller than the depth of $f(t_1, \dots, t_k)$. Since the multiset ordering on natural numbers is terminating, the first and second case can only be applied finitely many times by ordered resolution, as well.

Let $\Gamma \rightarrow \Delta, S(t), S(t')$ be a clause in N where ordered factoring with selection function sel applies with $\sigma = \text{mgu}(S(t), S(t'))$ and $R = (\Gamma \rightarrow \Delta, S(t))\sigma$. Because in $\Gamma \rightarrow \Delta, S(t), S(t')$ no literal is selected, $\Gamma \rightarrow \Delta, S(t), S(t')$ and $(\Gamma \rightarrow \Delta, S(t))\sigma$ can only match one of three patterns.

$(A)\ S_1(x_1), \dots, S_n(x_n) \rightarrow S(f(y_1, \dots, y_k)), S(f(z_1, \dots, z_l)), \Delta$

where $\{x_1, \dots, x_n\} \subseteq \{y_1, \dots, y_k\} \cup \{z_1, \dots, z_l\} \cup \text{vars}(\Delta)$, $t = f(y_1, \dots, y_k)$, and $t' = f(z_1, \dots, z_k)$. The result is

$$S_1(x_1)\sigma, \dots, S_n(x_n)\sigma \rightarrow S(f(y_1, \dots, y_k)), \Delta \text{ with } \sigma = \{z_1 \mapsto y_1, \dots\}.$$

$(B)\ S_1(x_1), \dots, S_n(x_n) \rightarrow S(f(y_1, \dots, y_k)), S(z), \Delta$

where $t = f(y_1, \dots, y_k)$, $t' = z$ and $\{x_1, \dots, x_n\} \subseteq \{y_1, \dots, y_k\} \cup \text{vars}(\Delta)$, i.e., z occurs only once. The result is

$$S_1(x_1), \dots, S_n(x_n) \rightarrow S(f(y_1, \dots, y_k)), \Delta.$$

$(C)\ S_1(x_1), \dots, S_n(x_n) \rightarrow S(y), S(z), \Delta$

where $t = y$, $t' = z$ and $\{x_1, \dots, x_n\} \subseteq \text{vars}(\Delta)$, i.e., y and z occur only once. The result is

$$S_1(x_1), \dots, S_n(x_n) \rightarrow S(y), \Delta.$$

In each case, the resolvent is again an MSL clause.

Furthermore, in each case the clause is of the form $S_1(x_1), \ldots, S_l(x_l) \rightarrow \Delta$. By Lemma 5.1.4, there are only finitely many such clauses after condensation and removal of variants. Therefore, these three cases can apply only finitely many times during saturation. □

Theorem 5.1.6 (Non-Horn Decidability)**.** *Satisfiability of MSL first-order clause sets is decidable.*

Proof. Follows from Lemma 5.1.5, as well as the soundness and completeness of ordered resolution with selection. □

5.2 Approximation $\Rightarrow_{\mathrm{AP}}$

I introduce the concrete over-approximation $\Rightarrow_{\mathrm{AP}}$ that maps a clause set N to an MSL clause set N'. The main difference between $\Rightarrow_{\mathrm{AP}}$ and $\Rightarrow_{\mathrm{APH}}$ is the removal of the Horn transformation rule. As a consequence, the shallow and linear rules are modified to handle non-Horn clauses as well.

Monadic $\quad N \Rightarrow_{\mathrm{MO}} \mu_P^T(N)$

provided P is a non-monadic predicate in the signature of N

Shallow $\quad N \,\dot\cup\, \{\Gamma \rightarrow E[s]_p, \Delta\} \Rightarrow_{\mathrm{SH}}$
$N \cup \{S(x), \Gamma_l \rightarrow E[p/x], \Delta_l\} \cup \{\Gamma_r \rightarrow S(s), \Delta_r\}$

provided s is a complex term, $p \neq \varepsilon$, x and S fresh,
$\Gamma_l\{x \mapsto s\} \cup \Gamma_r = \Gamma$, $\Delta_l \cup \Delta_r = \Delta$,
$\{Q(y) \in \Gamma \mid y \in \mathrm{vars}(E[p/x], \Delta_l)\} \subseteq \Gamma_l$,
$\{Q(y) \in \Gamma \mid y \in \mathrm{vars}(S(s), \Delta_r)\} \subseteq \Gamma_r$;

Linear 1 $\quad N \,\dot\cup\, \{\Gamma \rightarrow \Delta, E'[x]_p, E[x]_q\} \Rightarrow_{\mathrm{LI}}$
$N \cup \{\Gamma\sigma, \Gamma \rightarrow \Delta, E'[x]_p, E[q/x']\}$

provided x' is fresh and $\sigma = \{x \mapsto x'\}$.

Linear 2 $\quad N \,\dot\cup\, \{\Gamma \rightarrow \Delta, E[x]_{p,q}\} \Rightarrow_{\mathrm{LI}}$
$N \cup \{\Gamma\sigma, \Gamma \rightarrow \Delta, E[q/x']\}$

provided x' is fresh, $p \neq q$ and $\sigma = \{x \mapsto x'\}$.

Definition 5.2.1 ($\Rightarrow_{\mathrm{AP}}$)**.** *Define $\Rightarrow_{\mathrm{AP}}$ as the priority rewrite system [3] consisting of $\Rightarrow_{\mathrm{MO}}$, $\Rightarrow_{\mathrm{SH}}$ and $\Rightarrow_{\mathrm{LI}}$ with priority $\Rightarrow_{\mathrm{MO}} > \Rightarrow_{\mathrm{SH}} > \Rightarrow_{\mathrm{LI}}$.*

Lemma 5.2.2 ($\Rightarrow_{\mathrm{AP}}$ is a Terminating Over-Approximations)**.** *$\Rightarrow_{\mathrm{AP}}$ is a terminating over-approximation.*

Proof. Termination follows from Lemmas 3.1.2, 3.2.1, 3.3.1, and 3.4.1 and the fact that neither transformation creates clauses that a higher priority rule could be applied on. $\Rightarrow_{\text{APH}}$ is an over-approximation by Lemmas 3.1.3, 3.2.2, 3.3.2, and 3.4.2. □

Corollary 5.2.3. *If $N \Rightarrow^*_{\text{AP}} N'$ and N' is satisfied by a model $\mathcal{I}$, then $\mu^{-1}_{\Sigma}(\mathcal{I})$ is a model of N.*

Proof. Follows from Lemma 5.2.2. □

As in Chapter 4, the minimal Herbrand model of the eventual approximation generated by $\Rightarrow_{\text{AP}}$ preserves the skeleton term structure of the original clause set, if it exists.

Lemma 5.2.4. *Let N_0 be a monadic clause set and N_k be its approximation via $\Rightarrow_{\text{AP}}$. Let N_k be satisfiable and $\mathcal{I}$ be a minimal Herbrand model for N_k. If $P(s) \in \mathcal{I}$ and P is a predicate in N_0, then there exists a clause $C = \Gamma \rightarrow \Delta, P(t) \in N_0$ and a substitution σ such that $s = \text{skt}(t)\sigma$ and for each variable x and predicate S with $C = S(x), \Gamma' \rightarrow \Delta, P(t[x]_p)$ and $s|_p = s''$, $S(s'') \in \mathcal{I}$.*

Proof. By induction on the length of the approximation $N_0 \Rightarrow^*_{\text{AP}} N_k$.

For the base $N_k = N_0$, assume there is no $C \in N_k$ with $C\sigma = \Gamma \rightarrow \Delta, P(s)$ where for each variable x and predicate S with $C = S(x), \Gamma' \rightarrow \Delta, P(t[x]_p)$ and $s|_p = s''$, $S(s'') \in \mathcal{I}$. Then $\mathcal{I} \setminus \{P(s)\}$ is still a model of N_k and therefore $\mathcal{I}$ was not minimal. A contradiction.

Let $N_0 \Rightarrow_{\text{AP}} N_1 \Rightarrow^*_{\text{AP}} N_k$, $P(s) \in \mathcal{I}$ and P is a predicate in N_0 and hence also in N_1. By the induction hypothesis on $N_1 \Rightarrow^*_{\text{AP}} N_k$, there exist a clause $C = \Gamma \rightarrow \Delta, P(t) \in N_1$ and a substitution σ such that $s = \text{skt}(t)\sigma$ and $S(s_2) \in \mathcal{I}$ for each variable y and predicate S with $C = S(y), \Gamma' \rightarrow \Delta, P(t[y]_p)$ and $s|_p = s_2$. The first approximation rule application is either a Linear or a Shallow transformation, considered below by case analysis.

Linear Case. Let $\Rightarrow_{\text{AP}}$ be a Linear transformation where $\Gamma'' \rightarrow \Delta'[x]_{p,q} = C_0$ is approximated with $C_1 = \Gamma'', \Gamma''\{x \mapsto x'\} \rightarrow \Delta'[q/x']$. If $C \neq C_1$, then $C \in N_0$ fulfils the claim. Otherwise, $C_0 = \Gamma'' \rightarrow \Delta, P(t)\{x' \mapsto x\} \in N_0$ fulfils the claim because $s = \text{skt}(t)\sigma = \text{skt}(t\{x' \mapsto x\})\sigma$ and $\Gamma'' \subseteq \Gamma'', \Gamma''\{x \mapsto x'\}$.

Shallow Case. Let $\Rightarrow_{\text{AP}}$ be a Shallow transformation where $\Gamma'' \rightarrow Q(t'[s_1]_p), \Delta = C_0$ is approximated with $C_l = S(x), \Gamma_l \rightarrow Q(t'[p/x]), \Delta_l$ and $C_r = \Gamma_r \rightarrow S(s_1), \Delta_r$. If $C \neq C_l$ and $C \neq C_r$, then $C \in N_0$ fulfils the claim. If $C = C_r$, then $P(t) \in \Delta_r$ because S is fresh and thus $C_0 \in N_0$ fulfils the claim. Otherwise, $C = C_l$. If $P(t) \neq Q(t'[p/x])$, then $P(t) \in \Delta_l$ and $C_0 \in N_0$ fulfils the claim. Otherwise $C = S(x), \Gamma_l \rightarrow Q(t'[x]_p), \Delta_l$ with $P(t) = Q(t'[p/x])$, $s = \text{skt}(t[x]_p)\sigma$ and $S(s_2) \in \mathcal{I}$ for $s|_p = s_2$. Then by the induction hypothesis, there exist a clause $C_S = \Gamma_S \rightarrow \Delta_S, S(t_S)$ in N_1 and a substitution σ_S such that $s_2 = \text{skt}(t_S)\sigma_S$ and for each variable y and predicate S' with $C_S = S'(y), \Gamma'_S \rightarrow \Delta_S, P(t_S[y]_q)$ and $s_2|_q = s_3$, $S'(s_3) \in \mathcal{I}$. By construction, $C_S = C_r$. Thus, $s_2 = \text{skt}(s_1)\sigma_S$ and $s = \text{skt}(t[x]_p)\sigma$ imply there exists a σ'' such that $s = \text{skt}(t[s_1]_p)\sigma''$. Furthermore, because $\Gamma_l\{x \mapsto s_1\} \cup \Gamma_r = \Gamma''$, if

$C_0 = S'(y), \Gamma''' \rightarrow \Delta, P(t[s_1]_p)[y]_q$, then either $S'(y) \in \Gamma_l$ and thus $S'(s_4) \in \mathcal{I}_N$, where $s|_q = s_4$, or $S'(y) \in \Gamma_r$ and thus $S'(s_4) \in \mathcal{I}$, where $(s[s_2]_p)|_q = s_4$. Hence, $C_0 \in N_0$ fulfils the claim. □

Lemma 5.2.5. *Let $N \Rightarrow^*_{\text{AP}} N'$, where N' is a normal form, and $\mathcal{I}$ be a minimal Herbrand model of N'. If $\mu^T_\Sigma(P(s_1, \ldots, s_n)) \in \mathcal{I}$ and P is a predicate in N, then there is a clause $\Gamma \rightarrow \Delta, P(t_1, \ldots, t_n) \in N$ and a substitution σ such that $s_i = \text{skt}(t_i)\sigma$ for all i.*

Proof. Because of the rule priority of $\Rightarrow_{\text{AP}}$, $N \Rightarrow^*_{\text{MO}} \mu_\Sigma(N) \Rightarrow^*_{\text{AP}} N'$.

Let $P(s) \in \mathcal{I}$ and P be a monadic predicate in N. Since P is monadic, P is a predicate in $\mu_\Sigma(N)$. Hence by Lemma 5.2.4, there exists a $\Gamma \rightarrow \Delta, P(t) \in \mu_\Sigma(N)$ and a substitution σ such that $s = \text{skt}(t)\sigma$. Then, $\mu^{-1}_\Sigma(\Gamma \rightarrow \Delta, P(t)) \in N$ fulfils the claim.

Let $T(f_p(s_1, \ldots, s_n)) \in \mathcal{I}$. T is monadic and a predicate in $\mu_\Sigma(N)$. Hence by Lemma 5.2.4, there exists a clause $\Gamma \rightarrow \Delta, T(t) \in \mu_\Sigma(N)$ and a substitution σ such that $f_p(s_1, \ldots, s_n) = \text{skt}(t)\sigma$. Therefore, $t = f_p(t_1, \ldots, t_n)$ with $s_i = \text{skt}(t_i)\sigma$ for all i. Then, $\mu^{-1}_\Sigma(\Gamma \rightarrow \Delta, T(f_p(t_1, \ldots, t_n))) \in N$ fulfils the claim. □

5.3 Lifting of $\Rightarrow_{\text{AP}}$ and Approximation-Refinement

Lifting and refinement follow the same idea as in Chapter 5 with the exception of the missing Horn case and the slight changes to the Linear and Shallow transformation rules. Definition and generation of the conflicting core is also unchanged but now actually requires the factorization case.

Lemma 5.3.1 (Lifting the Linear Transformation)**.** *Let $N_k \Rightarrow_{\text{LI}} N_{k+1}$ be a Linear transformation where $N_k = N \cup \{C\}$, $N_{k+1} = N \cup \{C'\}$, $C = \Gamma \rightarrow \Delta[x]_{p,q}$ and $C' = \Gamma\{x \mapsto x'\}, \Gamma \rightarrow \Delta[q/x']$. Let $N^\perp_{k+1}$ be a conflicting core of N_{k+1} and $C'\sigma_1, \ldots, C'\sigma_m$ be all clauses in $N^\perp_{k+1}$ that are instances of C'. If $x\sigma_j = x'\sigma_j$ for $1 \leq j \leq m$, then $N^\perp_{k+1} \setminus \{C'\sigma_1, \ldots, C'\sigma_m\} \cup \{C\sigma_1, \ldots, C\sigma_m\} = N^\perp_k$ is a conflicting core of N_k.*

Proof. Let σ be any grounding substitution and $\mathcal{I}$ be any interpretation. Then, $\mathcal{I} \not\models N^\perp_{k+1}\sigma$ and there exists a clause $C^\perp \in N^\perp_{k+1}\sigma$ such that $\mathcal{I} \not\models C^\perp$. If $C^\perp$ is not an instance of C', then $C^\perp$ is a clause in $N^\perp_k\sigma$. Thus, $\mathcal{I} \not\models N^\perp_k\sigma$. If $C^\perp$ is an instance of C', then $C^\perp = C'\sigma_j\sigma$ for some $1 \leq j \leq m$. Because $x\sigma_j = x'\sigma_j$, $C'\sigma_j\sigma$ and $C\sigma_j\sigma$ are equal modulo duplicate literal elimination. Thus, $\mathcal{I} \not\models N^\perp_k\sigma$. Therefore, $N^\perp_k$ is a conflicting core of N_k. □

Lemma 5.3.2 (Lifting the Shallow Transformation)**.** *Let $N_k \Rightarrow_{\text{SH}} N_{k+1}$ be a Shallow transformation where $N_k = N \cup \{C\}$, $N_{k+1} = N \cup \{C_l, C_r\}$, $C = \Gamma \rightarrow E[s]_p, \Delta$; $C_l = S(x), \Gamma_l \rightarrow E[p/x], \Delta_l$ and $C_r = \Gamma_r \rightarrow S(s), \Delta_r$. Let $N^\perp_{k+1}$ be a conflicting core of N_{k+1} and N_S be the set of all resolvents of clauses in $N^\perp_{k+1}$ on the S-atom. If all clauses in N_S are instances of C modulo duplicate literal elimination, then $N^\perp_k = \{D \in N^\perp_{k+1} \mid D \text{ not an instance of } C_l \text{ or } C_r\} \cup N_S$ is a conflicting core of N_k.*

Proof. Let σ be any grounding substitution and $\mathcal{I}$ be any interpretation. Then, $\mathcal{I} \not\models N^{\perp}_{k+1}\sigma$ and there exists $C^{\perp} \in N^{\perp}_{k+1}\sigma$ such that $\mathcal{I} \not\models C^{\perp}$. If $C^{\perp}$ is not an instance of C_l or C_r, then $C^{\perp} \in N^{\perp}_{k}\sigma$. Thus, $\mathcal{I} \not\models N^{\perp}_{k}\sigma$. Otherwise, assume $C_l\tau_l \ldots, C_l\tau_m$ and $C_r\rho_l, \ldots, C_r\rho_n$ are the only clauses in $N^{\perp}_{k}\sigma$ false under $\mathcal{I}$. Let $\mathcal{I}' := \mathcal{I} \setminus \{S(x)\tau_l, \ldots, S(x)\tau_m\} \cup \{S(s)\rho_l, \ldots, S(s)\rho_n\}$, i.e., change the truth value for S-atoms such that the clauses unsatisfied under $\mathcal{I}$ are satisfied under $\mathcal{I}'$. Because $\mathcal{I}$ and $\mathcal{I}'$ only differ on S-atoms, there exists a clause $D \in N^{\perp}_{k}\sigma$ that is false under $\mathcal{I}'$ and contains an S-atom. Let $D = C_l\sigma'$. Since $\mathcal{I} \models D$, $S(x)\sigma'$ was added to $\mathcal{I}'$ by some clause $C_r\rho_j$, where $S(s)\rho_j = S(x)\sigma'$. Let R be the resolvent of $C_r\rho_j$ and $C_l\sigma'$ on $S(s)\rho_j$ and $S(x)\sigma'$. Then, $\mathcal{I} \not\models R$ because $\mathcal{I} \not\models C_r\rho_j$ and $\mathcal{I} \cup \{S(s)\rho_j\} \not\models C_l\sigma'$. Thus, $\mathcal{I} \not\models N^{\perp}_{k}\sigma$. For $D = C_r\sigma'$, the proof is analogous. Therefore, $N^{\perp}_{k}$ is a conflicting core of N_k. □

Definition 5.3.3 (Linear Approximation-Refinement). *Let $N \Rightarrow^{k}_{\text{APH}} N_k \Rightarrow_{\text{LI}} N_{k+1}$ where $C = \Gamma \rightarrow \Delta[x]_{p,q}$ is approximated by $C_1 = \Gamma\{x \mapsto x'\}, \Gamma \rightarrow \Delta[q/x']$. Let $N^{\perp}_{k+1}$ be a ground conflicting core of N_{k+1} with some lift-conflict $C_1\sigma \in N^{\perp}_{k+1}$ such that $x\sigma$ and $x'\sigma$ cannot be unified. Let $C \in N$ be the ancestor of $C' \in N_{k+1}$. $N \setminus \{C\} \cup \{C\tau_1, \ldots, C\tau_n\}$ is the* linear approximation-refinement *of N, where the $C\tau_i$ are the specific instances of C with respect to the conflicting instantiations $\{x \mapsto x\sigma\}$ and $\{x \mapsto x'\sigma\}$.*

Definition 5.3.4 (Shallow Approximation-Refinement). *Let $N \Rightarrow^{k}_{\text{AP}} N_k \Rightarrow_{\text{SH}} N_{k+1}$ where $C = \Gamma \rightarrow \Delta, E[s]_p$ is approximated by $C_l = S(x), \Gamma_l \rightarrow E[p/x], \Delta_l$ and $C_r = \Gamma_r \rightarrow S(s), \Delta_r$. Let $N^{\perp}_{k+1}$ be a ground conflicting core of N_{k+1} with $C_l\sigma_l \in N^{\perp}_{k+1}$ and $C_r\sigma_r \in N^{\perp}_{k+1}$ such that their resolvent C_R is not unifiable with C. Let $C \in N$ be the ancestor of C_l. $N \setminus \{C\} \cup \{C\tau_l, \ldots, C\tau_n\}$ is the* shallow approximation-refinement *of N, where the $C\tau_i$ are the specific instances of C with respect to the substitutions σ_l and σ_r.*

Theorem 5.3.5 (Static Completeness). *Let N_0 be an unsatisfiable clause set and N_k its MSL approximation. Then, there exists a conflicting core of N_k that can be lifted to N_0.*

Proof. by induction on the number k of approximation steps. The case $k = 0$ is obvious. For $k > 0$, let $N_0 \Rightarrow^{k-1}_{\text{AP}} N_{k-1} \Rightarrow_{\text{AP}} N_k$. By the inductive hypothesis, there is a conflicting core $N^{\perp}_{k-1}$ of N_{k-1} which can be lifted to N_0. The final approximation rule application is either a Linear, a Shallow, or a Monadic transformation, considered below by case analysis.

Linear Case. Let $N_{k-1} = N' \cup \{C\} \Rightarrow_{\text{LI}} N_k = N' \cup \{C'\}$ with $C = \Gamma \rightarrow \Delta[x]_{p,q}$ and $C' = \Gamma\{x \mapsto x'\}, \Gamma \rightarrow \Delta[q/x']$. Let $C\sigma_1, \ldots, C\sigma_n$ be the instances of C in $N^{\perp}_{k-1}$. $N^{\perp}_{k-1} \setminus \{C\sigma_1, \ldots, C\sigma_n\} \cup \{C'\{x' \mapsto x\}\sigma_j \mid 1 \leq j \leq n\}$ is a conflicting core of N_k. By Lemma 5.3.1, it can be lifted to $N^{\perp}_{k-1}$ and by the inductive hypothesis, can be lifted to a conflicting core of N_0.

Shallow Case. Let $N_{k-1} = N' \cup \{C\} \Rightarrow_{\text{SH}} N_k = N' \cup \{C_l, C_r\}$ with $C = \Gamma \rightarrow E[s]_p, \Delta_l$; $C_l = S(x), \Gamma_l \rightarrow E[p/x]$; and $C_r = \Gamma_r \rightarrow S(s), \Delta_r$. Assume $C\sigma$

is the only instances of C in $N^{\perp}_{k-1}$. $N^{\perp}_{k} = N^{\perp}_{k-1} \setminus \{C\sigma\} \cup \{C_l\{x \mapsto s\}\sigma, C_r\sigma\}$ is a conflicting core of N_k. By Lemma 5.3.2, $N^{\perp}_{k}$ can be lifted to $N^{\perp}_{k-1}$. Now, let $C\sigma_l, \ldots, C\sigma_n$ be the instances of C in $N^{\perp}_{k-1}$ with $n > 1$. Let $C_0 \in N_0$ be the ancestor of C and $N'_0 = N_0 \setminus \{C_0\} \cup \{C_0\sigma_l, \ldots, C_0\sigma_n\}$. $N^{\perp}_{k-1}$ is also a conflicting core for the corresponding approximation $N'_k = N' \cup \{C\sigma_l, \ldots, C\sigma_n\}$. For each $C\sigma_i$, $N^{\perp}_{k-1}$ contains only one instance such that the above case applies. Thus, there is a core for approximation of each $C\sigma_i$ that can be lifted to $N^{\perp}_{k-1}$.

Monadic Case. Let $N_{k-1} \Rightarrow_{\mathrm{MO}} N_k = \mu_P(N_{k-1})$ where P is a non-monadic predicate in N_{k-1} $N^{\perp}_{k} = \mu_P(N^{\perp}_{k-1})$ is a conflicting core of N_k. By Lemma 4.2.7, $N^{\perp}_{k}$ can be lifted to $N^{\perp}_{k-1}$. □

Chapter 6

Straight Dismatching Constraint Approximation-Refinement

In the previous chapter, I presented the decidability of the MSL fragment. This allowed me to change the approximation target for SPASS-AR from the MSLH to the MSL fragment and thereby avoid the worst-case exponential blow-up caused by the Horn transformation during lifting.

The next biggest problem that becomes apparent lies in the refinement: The number of specific instances, which the refinement replaces an original clause with, is in the worst-case quadratic (see Definitions 4.3.1, 5.3.3 and 5.3.4).
For example, consider the first-order Horn clauses

$$\begin{array}{rcl} Q(x) & \rightarrow & P(g(x,f(x))) \\ & \rightarrow & Q(f(g(a,a))) \\ & \rightarrow & Q(f(g(b,b))) \\ P(g(f(g(a,a)),f(f(g(b,b))))) & \rightarrow & \end{array}$$

under signature $\{a/0, b/0, f/1, g/2\}$ that are approximated into

$$\begin{array}{rcl} S(y), Q(x) & \rightarrow & P(g(x,y)) \\ Q(z) & \rightarrow & S(f(z)) \\ & \rightarrow & Q(f(g(a,a))) \\ & \rightarrow & Q(f(g(b,b))) \\ P(g(f(g(a,a),f(f(g(b,b)))) & \rightarrow & \end{array}$$

via linearisation of $g(x, f(x))$ to $g(x, f(z))$ and then deep variable term extraction of $f(z)$ through the introduction of a fresh predicate S [43]. The approximated clause set has a refutation and the corresponding conflicting core, a minimal unsatisfiable set of instances from the above clauses generating this refutation, is

$$\begin{array}{rcl} S(f(f(g(b,b)))), Q(f(g(a,a))) & \rightarrow & P(g(f(g(a,a)),f(f(g(b,b))))) \\ Q(f(g(b,b))) & \rightarrow & S(f(f(g(b,b)))) \\ & \rightarrow & Q(f(g(a,a))) \\ & \rightarrow & Q(f(g(b,b))) \\ P(g(f(g(a,a),f(f(g(b,b)))) & \rightarrow & \end{array}$$

Lifting the conflicting core to the original clause set fails, because the resolvent of the first two conflict clauses, eliminating the introduced S predicate

$$Q(f(g(b,b))), Q(f(g(a,a))) \quad\rightarrow\quad P(g(f(g(a,a)), f(f(g(b,b)))))$$

is not an instance of the original clause $Q(x) \rightarrow P(g(x, f(x)))$, modulo duplicate literal elimination. A refinement step replaces $Q(x) \rightarrow P(g(x, f(x)))$ by the instance $Q(f(g(a,y))) \rightarrow P(g(f(g(a,y)), f(f(g(a,y)))))$, and instances representing $Q(x) \rightarrow P(g(x, f(x)))$, where x is not instantiated with $f(g(a,y))$. The former clause contains the ground instance

$$Q(f(g(a,a))) \rightarrow P(g(f(g(a,a)), f(f(g(a,a)))))$$

and the latter clauses include the ground instance

$$Q(f(g(b,b))) \rightarrow P(g(f(g(b,b)), f(f(g(b,b))))) :$$

$$\begin{aligned}
Q(a) &\rightarrow P(g(a, f(a)))\\
Q(b) &\rightarrow P(g(b, f(b)))\\
Q(g(x,y)) &\rightarrow P(g(g(x,y), f(g(x,y))))\\
Q(f(a)) &\rightarrow P(g(f(a), f(f(a))))\\
Q(f(b)) &\rightarrow P(g(f(b), f(f(b))))\\
Q(f(f(x))) &\rightarrow P(g(f(f(x)), f(f(f(x)))))\\
Q(f(g(b,y))) &\rightarrow P(g(f(g(b,y)), f(f(g(b,y)))))\\
Q(f(g(f(x),y))) &\rightarrow P(g(f(g(f(x),y)), f(f(g(f(x),y)))))\\
Q(f(g(g(x,y),z))) &\rightarrow P(g(f(g(g(x,y),z)), f(f(g(g(x,y),z)))))
\end{aligned}$$

Then, the approximation of the above nine clauses, via linearisation and deep variable term extraction, excludes the previously found refutation. Actually, the refined approximated clause set yields a finite saturation and therefore shows satisfiability of the original clause set.

Note that this list of instantiations depends on the signature and the conflicting instantiations. Specifically in this case, the number of specific instances is the product of $|\{a, b, f, g\}| - 1$ and the depth of the term $f(g(a,a))$.

As a solution, I introduce in this chapter the notion of *straight dismatching constraints* (SDC) which extend the clause language to allow expressing "C, where x is not instantiated with straight[1] term s " as the single constraint clause $(C; x \neq s)$.

This enables a refinement where a clause is replaced by exactly two new clauses. For the example, they are

$$\begin{aligned}
&Q(f(g(a,y))) \rightarrow P(g(f(g(a,y)), f(f(g(a,y)))))\\
\text{and}\quad &Q(x) \rightarrow P(g(x, f(x)))\ ;\ x \neq f(g(a,v)).
\end{aligned}$$

This results in the refined approximation

$$S_1(y), S_2(x), Q(x) \rightarrow\ P(g(x,y))^* \tag{6.1}$$

[1]see Definition 6.1.1

$$
\begin{aligned}
S_2(x), Q(x) &\rightarrow S_1(f(x))^* && (6.2)\\
S_3(x) &\rightarrow S_2(f(x))^* && (6.3)\\
&\rightarrow S_3(g(a,y))^* && (6.4)\\
S_4(y), Q(x) &\rightarrow P(g(x,y))^*;\ x \neq f(g(a,v)) && (6.5)\\
Q(x) &\rightarrow S_4(f(x))^*;\ x \neq f(g(a,v)) && (6.6)\\
&\rightarrow Q(f(g(a,a)))^* && (6.7)\\
&\rightarrow Q(f(g(b,b)))^* && (6.8)\\
P(g(f(g(a,a)), f(f(g(b,b)))))^+ &\rightarrow && (6.9)
\end{aligned}
$$

where under the $\prec_{\text{lpo}}$ ordering induced by the precedence $P \prec Q \prec S_4 \prec S_3 \prec S_2 \prec S_1 \prec a \prec b \prec f \prec g$ and selection of the first complex negative literal, the only inferences are

$$
\begin{aligned}
&[10 : Res : 1,\ 9] && S_1(f(f(g(b,b))))^+, S_2(f(g(a,a))), Q(f(g(a,a))) \rightarrow\\
&[11 : Res : 2, 10] && S_2(f(g(b,b)))^+, Q(f(g(b,b))), S_2(f(g(a,a))), Q(f(g(a,a))) \rightarrow\\
&[12 : Res : 3, 11] && S_3(g(b,b))^+, Q(f(g(b,b))), S_2(f(g(a,a))), Q(f(g(a,a))) \rightarrow
\end{aligned}
$$

A resolution between the fifth and ninth clause has the tautologous result

$$(S(f(f(g(b,b)))), Q(f(g(a,a))) \rightarrow;\ f(g(a,a)) \neq f(g(a,v))).$$

Clauses with straight dismatching constraints are closed under resolution and factoring. Repeated refinement steps and resolution inferences can add further atomic constraints to a clause, which are interpreted as conjunctions, for example, $(C; x \neq t \wedge y \neq s)$ represents instances of C where neither x is an instance of t nor y is an instance of s. Straight dismatching constraints can be efficiently encoded in amortized constant space through structure sharing once the input constraint terms are stored. Any straight term generated by the calculus is a subterm of an input straight term. Relevant operations (substitution, intersection, solvability and subset tests) take linear time in the size of the constraints, once they are ordered. Ordering can be done as usual in linear logarithmic time. The ordered resolution calculus with straight dismatching constraints is sound and complete, enables an abstract redundancy criterion and follows the saturation principle: if all non-redundant inferences from a clause set are performed and the empty clause is not generated then the resulting saturated clause set has a model computed by a model operator.

Note that straight dismatching constraints are incomparable to the disequality constraints with which $\mathcal{H}_1$ is extended in [38, 39].

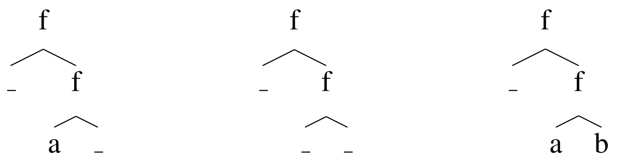
For the improved approximation-refinement approach (FO-AR) presented here, any refinement step results in just two clauses, see Section 6.5. The additional expressiveness of constraint clauses comes almost for free, because necessary computations, like, e.g., checking emptiness of SDCs, can all be done in linear or linear-logarithmic time, see Section 6.1.

6.1 Dismatching Constraints

In the following I extend the notion of clauses by adding straight dismatching constraints. The constraints are used to restrict which grounding substitutions can be applied to a clause.

Definition 6.1.1 (Straight Terms). *A term $s = f(s_1, \ldots, s_n)$ is called straight, if s is linear and all arguments are variables except for at most one straight argument s_i.*

For example, the $f(x, f(a, y))$ and $f(x, f(y, z))$ are straight, while $f(x, f(a, b))$ is not. Straight terms can be identified visually in their tree form, if there is exactly one branch with function symbols on each node.



Note that for two different ground terms t_1 and t_2, there always exists a position in both where they have different function symbols. The path from the root to this position in t_1 defines a straight term t that has only t_1 as an instance but not t_2. Thus, a straight term is sufficient to isolate two ground instances.

Definition 6.1.2 (Straight Dismatching Constraint). *A straight dismatching constraint π is of the form*

$$\bigwedge_{i \in \mathrm{I}} t_i \neq s_i$$

where I *is a finite set of indices, the t_i are arbitrary terms and the s_i are straight terms. Given a substitution σ, $\pi\sigma = \bigwedge_{i \in \mathrm{I}} t_i\sigma \neq s_i$.*

Further extend the set of constraints with the constants $\top, \bot$ representing the empty and the unsolvable constraint, respectively. An atomic *constraint $t \neq s$ occurring in π is called a* sub-constraint *of π. The length $|\pi|$ is defined as the number of sub-constraints, $|\mathrm{I}|$. The depth of a constraint $\bigwedge_{i \in \mathrm{I}} t_i \neq s_i$ is the maximal term depth of the s_i.*

Definition 6.1.3 (Solutions). *Let π be a straight dismatching constraint, Σ a signature and $\mathcal{V}$ a set of variables with* $\mathrm{lvar}(\pi) \subseteq \mathcal{V}$*. A* solution of π over $\mathcal{V}$ *is a grounding substitution δ over $\mathcal{V}$ such that for all $i \in \mathrm{I}$, $s_i\delta$ is not an instance of t_i. $\mathcal{D}^{\mathcal{V}}(\pi) := \{\delta \mid \delta \text{ is a solution of } \pi \text{ over } \mathcal{V}\}$. A straight dismatching constraint is solvable if it has a solution and unsolvable, otherwise. Two constraints are equivalent if they have the same solutions over all $\mathcal{V}$. In particular, all grounding substitutions are solutions of $\top$, and $\bot$ has no solution.*

For example, consider the constraint $\pi = x \neq b \wedge x \neq f(f(u)) \wedge x \neq g(v, w)$ with the signature $\mathcal{F} = \{a/0, b/0, f/1, g/2\}$. π is solvable and has the set of solutions $\mathcal{D}^{\mathcal{V}}(\pi) = \{\{x \mapsto a\}, \{x \mapsto f(a)\}, \{x \mapsto f(b)\}, \{x \mapsto f(g(s, t))\} \mid s, t \in \mathcal{T}(\mathcal{F}, \emptyset)\}$ over $\{x\}$.

Note, that atomic constraints are not symmetric. For example $x \neq a$ can have a solution while $a \neq x$ is unsolvable. Furthermore, the right-hand variables in a constraint can be arbitrarily renamed while the left-hand variables are fixed. For example, the constraints $x \neq f(v)$ and $x \neq f(w)$ are equivalent but $x \neq f(v)$ and $y \neq f(v)$ are not. For an atom $P(g(x, y))$ and a constraint $x \neq f(y)$, the two y variables are not connected since for $\sigma = \{y \mapsto t\}$, $P(g(x, y))\sigma = P(g(x, t))$ but $(x \neq f(y))\sigma = (x \neq f(y))$. To avoid confusion, I generally rename right-hand variables to be unique in a given context.

Furthermore, if δ is a solution of $\pi \wedge \pi'$, then δ is a solution of π and if δ is a solution of $\pi\sigma$, then $\sigma\delta$ is a solution of π. These two properties of solutions follow directly from the definition and I will use them frequently without pointing them out specifically. Further, I will ignore $\mathcal{V}$ if it is clear in a given context. For example, if π restricts the clause C, $\mathcal{V}$ is $\text{vars}(C) \cup \text{lvar}(\pi)$.

Definition 6.1.4 (Constraint Variant)**.** *A constraint $\pi = \bigwedge_{i \in I} t_i \neq s_i$ is called a variant of π' if there is a variable renaming ρ with $\pi' = \bigwedge_{i \in I} t_i \neq s_i\rho$.*

Lemma 6.1.5. *If π is a variant of π', then π and π' are equivalent.*

Proof. Let δ be a solution of $\pi = \bigwedge_{i \in I} t_i \neq s_i$. Then $t_i\delta$ is not an instance of s_i if and only if $t_i\delta$ is not an instance of $s_i\rho$. Hence δ is a solution of $\pi = \bigwedge_{i \in I} t_i \neq s_i$ if and only if it is a solution of $\pi' = \bigwedge_{i \in I} t_i \neq s_i\rho$. □

Definition 6.1.6 (Constraint Normalization)**.** *Define* constraint normalization $\pi{\downarrow}$ *as the normal form of the following rewriting rules over constraints.*

1	$\pi \wedge f(\bar{t}) \neq y$	$\Rightarrow$	$\bot$	
2	$\pi \wedge f(\bar{t}) \neq f(\bar{y})$	$\Rightarrow$	$\bot$	
3	$\pi \wedge f(\bar{t}) \neq f(\bar{s})$	$\Rightarrow$	$\pi \wedge t_i \neq s_i$	*if s_i is complex*
4	$\pi \wedge f(\bar{t}) \neq g(\bar{s})$	$\Rightarrow$	π	*if $f \neq g$*
5	$\pi \wedge x \neq s \wedge x \neq s\sigma$	$\Rightarrow$	$\pi \wedge x \neq s$	

Note that the depth of $\pi{\downarrow}$ is less or equal to the depth of π.

Definition 6.1.7 (Constraint Normal Form)**.** *A straight dismatching constraint $\pi = \bigwedge_{i \in I} t_i \neq s_i$ is called* normal*, if all t_i are variables and if $t_i = t_j$ with $i \neq j$, then s_i is not an instance of s_j.*

Lemma 6.1.8. *$\pi{\downarrow}$ is a normal constraint and equivalent to π.*

Proof. In the first and second rule, $f(t_1, \ldots, t_n)$ is an instance of y and $f(y_1, \ldots, y_n)$, respectively. Hence, there are no solutions, which is equivalent to the unsatisfiable constraint $\bot$. These steps also terminating. In the third rule, a grounding substitution δ is a solution of $f(t_1, \ldots, t_n) \neq f(s_1, \ldots, s_n)$, if and only if each $t_j\delta$ is not an instance of s_j. However, since $f(s_1, \ldots, s_n)$ is straight there is exactly one complex s_i or the second rule applies. Thus, the $t_j\delta$ are instances of s_j for

$j \neq i$ and the sub-constraint is equivalent to $t_i \neq s_i$. In this case, the depth of the sub-constraint decreases by one. In the forth rule, $f(t_1, \ldots, t_n)$ can never be an instance of $g(s_1, \ldots, s_m)$ for any grounding substitution. Hence, all solutions of π are solutions of $\pi \wedge f(t_1, \ldots, t_n) \neq g(s_1, \ldots, s_m)$. In the last rule, let δ be a solution of $\pi \wedge x \neq t$. Then, $x\delta$ is not an instance of t and hence, not an instance of $t\sigma$ as well. Therefore, δ is a solution of $\pi \wedge x \neq t \wedge x \neq t\sigma$. Both rules reduces the number of sub-constraints. All rules together terminate and cover every case where the left-hand side of a sub-constraint is not a variable or there are two constraints $x \neq t$ and $x \neq s$ where s is an instance of t. Hence, applying the five rules exhaustively results in a normal form that is a normal dismatching constraint. □

Definition 6.1.9 (Constrained Terms). *A pair of a term and a constraint* $(t; \pi)$ *is called a* constrained term. *For a given signature* Σ, *the* set of ground instances of $(t; \pi)$ *is* $\mathcal{G}((t; \pi)) = \{t\delta \mid \delta \in \mathcal{D}^{\mathcal{V}}(\pi), \mathcal{V} = \text{vars}(t) \cup \text{lvar}(\pi)\}$.

Definition 6.1.10 (Constrained Clauses). *A pair of a clause and a constraint* $(C; \pi)$ *is called a* constrained clause. *A constrained clause is normal, if* π *is normal and* $\text{lvar}(\pi) \subseteq \text{vars}(C)$. *For a given signature* Σ, *the* set of ground instances of $(C; \pi)$ *is* $\mathcal{G}((C; \pi)) = \{C\delta \mid \delta \in \mathcal{D}^{\mathcal{V}}(\pi), \mathcal{V} = \text{vars}(C) \cup \text{lvar}(\pi)\}$. $(C; \pi)$ *is called an instance of* $(C'; \pi')$ *if* $\mathcal{G}((C; \pi)) \subseteq \mathcal{G}((C'; \pi'))$. *The notion extends to sets of constrained clauses. Two constrained clauses are equivalent if they have the same ground instances. A Herbrand interpretation* $\mathcal{I}$ *satisfies* $(C; \pi)$, *if* $\mathcal{I} \models \mathcal{G}((C; \pi))$.

Note that in the context of a constrained clause $(C; \pi)$, a solution δ of π is implicitly over $\mathcal{V} = \text{vars}(C) \cup \text{lvar}(\pi)$ such that $C\delta$ is always in $\mathcal{G}((C; \pi))$.

Definition 6.1.11 (Clause Variant). *A clause* $(C; \pi)$ *is called a* variant *of* $(D; \pi')$ *if there is a variable renaming* ρ *such that* $C\rho = D$ *and* $\pi\rho$ *is a variant of* π'.

Lemma 6.1.12. *If* $(C; \pi)$ *is a variant of* $(D; \pi')$, *then* $(C; \pi)$ *and* $(D; \pi')$ *are equivalent.*

Proof. Let $D\delta \in \mathcal{G}((D; \pi'))$ and ρ be a variable renaming such that $C\rho = D$ and $\pi\rho$ is a variant of π'. Then $\rho\delta$ is a solution of π and $D\delta = C\rho\delta \in \mathcal{G}((C; \pi))$. The reverse direction is analogous. □

Lemma 6.1.13 (Clause Simplification). *A constrained clause* $(C; \pi \wedge \bigwedge_{i \in \mathrm{I}} x \neq s_i)$ *is equivalent to* $(C; \pi)$, *if* π *is normal,* $x \notin \text{vars}(C) \cup \text{lvar}(\pi)$ *and* $\bigwedge_{i \in \mathrm{I}} x \neq s_i$ *is solvable.*

Proof. Let $C\delta \in \mathcal{G}(C; \pi)$, where δ is a solution of π over $\mathcal{V}$. Because $\bigwedge_{i \in \mathrm{I}} x \neq s_i$ is solvable, there is some solution $\{x \mapsto t\}$ over $\{x\}$. Then, $\delta[x \mapsto t]$ is a solution of $\bigwedge_{i \in \mathrm{I}} x \neq s_i \wedge \pi$ over $\mathcal{V} \cup \{x\}$. Hence, $C\delta[x \mapsto t] \in \mathcal{G}(C; \pi \wedge x \bigwedge_{i \in \mathrm{I}} x \neq s_i)$. The reverse direction is trivial. □

Lemmas 6.1.8 and 6.1.13 show that any constrained clause can be replaced with an equivalent normal clause. In the following I will assume any constrained clause to be normal. For readability, I also often write clauses with the empty constraint $(C; \top)$ simply as C.

Definition 6.1.14 (Constrained Redundancy)**.** *For a given ordering on ground clauses $\prec$, a constrained clause $(C;\pi)$ is called* redundant *in N if for all $D \in \mathcal{G}((C;\pi))$, there exist $D_1,\dots,D_n$ in $\mathcal{G}(N)^{\prec D}$ such that $D_1,\dots,D_n \models D$.*

Note that a constrained clause $(C;\pi)$ with a tautology C or an unsolvable constraint π is redundant for any N.

Definition 6.1.15 (Constrained Condensation)**.** *A constrained clause $(C';\pi')$ is called a condensation of $(C;\pi)$ if $C' \subset C$ and there exists a substitution σ such that, $\pi\sigma = \pi'$, $\pi' \subseteq \pi$, and for all $L \in C$ there is a $L' \in C'$ with $L\sigma = L'$.*

Lemma 6.1.16. *Let constrained clause $(C';\pi')$ be a condensation of constrained clause $(C;\pi)$. Then, (i)$(C;\pi) \models (C';\pi')$ and (ii)$(C;\pi)$ is redundant in $\{(C';\pi')\}$.*

Proof. Let σ be a substitution such that $C' \subset C$, $\pi\sigma = \pi'$, $\pi' \subseteq \pi$, and for all $L \in C$ there is a $L' \in C'$ with $L\sigma = L'$.

(i) Let $C'\delta \in \mathcal{G}((C';\pi'))$. Then $\sigma\delta$ is a solution of π and hence $C\sigma\delta \in \mathcal{G}((C;\pi))$. Let $\mathcal{I} \models C\sigma\delta$. Hence, there is a $L\sigma\delta \in \mathcal{I}$ for some $L \in C$ and thus $L'\delta \in \mathcal{I}$ for some $L' \in C'$ with $L\sigma = L'$. Therefore, $\mathcal{I} \models C'\delta$. Since $\mathcal{I}$ and $C'\delta$ were arbitrary, $(C;\pi) \models (C';\pi')$.

(ii) Let $C\delta \in \mathcal{G}((C;\pi))$. Because $\pi' \subseteq \pi$, δ is a solution of π' and hence, $C'\delta \in \mathcal{G}((C';\pi'))$. Therefore, since $C'\delta \subset C\delta$, $C'\delta \in \mathcal{G}(\{(C';\pi')\})^{\prec C\delta}$ and $C'\delta \models C\delta$. □

Lemma 6.1.17 (Refinement)**.** *Let $N \cup \{(C;\pi)\}$ be a constrained clause set, $x \in$ vars(C) and s a straight term with* vars$(s) \cap$ vars$(C) = \emptyset$*. Then $N \cup \{(C;\pi)\}$ and the* refinement *$N \cup \{(C;\pi \wedge x \neq s), (C;\pi)\{x \mapsto s\}\}$ are satisfiability equivalent.*

Proof. Let $C\delta \in \mathcal{G}((C;\pi))$ be an arbitrary clause. If $x\delta$ is not an instance of s, then δ is a solution of $\pi \wedge x \neq s$ and $C\delta \in \mathcal{G}((C;\pi \wedge x \neq s))$. Otherwise, $\delta = \{x \mapsto s\}\delta'$ for some substitution δ'. Then, δ' is a solution of $\pi\{x \mapsto s\}$ and therefore, $C\delta = C\{x \mapsto s\}\delta' \in \mathcal{G}((C\{x \mapsto s\};\pi\{x \mapsto s\}))$. Hence, $\mathcal{G}((C;\pi)) \subseteq \mathcal{G}((C;\pi \wedge x \neq s)) \cup \mathcal{G}((C;\pi)\{x \mapsto s\})$. Therefore, if $\mathcal{I}$ is a model of $N \cup \{(C;\pi \wedge x \neq s), (C;\pi)\{x \mapsto s\}\}$, then $\mathcal{I}$ is also a model of $N \cup \{(C;\pi)\}$.

Let $D \in \mathcal{G}((C;\pi \wedge x \neq s)) \cup \mathcal{G}((C;\pi)\{x \mapsto s\})$ be an arbitrary clause. If $D = C\delta \in \mathcal{G}((C;\pi \wedge x \neq s))$, then δ is also a solution of π and therefore $D \in \mathcal{G}((C;\pi))$. If $D = C\{x \mapsto s\}\delta \in \mathcal{G}((C;\pi)\{x \mapsto s\})$, then $\{x \mapsto s\}\delta$ is a solution of π and therefore $D \in \mathcal{G}((C;\pi))$. Hence, $\mathcal{G}((C;\pi \wedge x \neq s)) \cup \mathcal{G}((C;\pi)\{x \mapsto s\}) \subseteq \mathcal{G}((C;\pi))$. Therefore, if $\mathcal{I}$ is a model of $N \cup \{(C;\pi)\}$, $\mathcal{I}$ is also a model of $N \cup \{(C;\pi \wedge x \neq s), (C;\pi)\{x \mapsto s\}\}$. □

Consider again the introductory example. Lifting the conflicting core of the approximation failed because $D = Q(f(f(b))), Q(f(f(a))) \rightarrow P(g(f(f(a)), f(f(b))))$ is not an instance of the original clause $Q(x) \rightarrow P(g(x, f(x)))$. Specifically, because x cannot be instantiated with an instance of both $f(f(a))$ and $f(f(b))$. Therefore, I refine the original clause with the instantiation $\{x \mapsto f(f(a))\}$ and the opposing constraint $x \neq f(f(a))$, resulting in $C_1 = Q(f(f(a))) \rightarrow P(g(f(f(a)), f(f(f(a)))))$ and $C_2 = (Q(x) \rightarrow P(g(x, f(x)))\ ;\ x \neq f(f(a)))$. D differs from C_1 because both occurrences of x are already instantiated with $f(f(a))$, excluding any approximations, where the second x is instantiated with $f(f(b))$. Since $x \neq f(f(a))$ has no solutions where x is instantiated with $f(f(a))$ and the approximation preserves this fact by duplicating constraints, D can not be inferred from the approximation of C_2. Therefore, the clause D can not appear in the refined approximation again.

6.1.1 Emptiness Check

In this section, I show that solvability of a constraint is decidable. Note that not all unsolvable constraints trivially normalize to $\bot$. A normal constraint can actually be both solvable and unsolvable depending on the signature. For example, $x \neq a$ is a normal constraint that is solvable for signature $\{a/0, b/0\}$ but unsolvable for signature $\{a/0\}$.

Definition 6.1.18. *For a constraint $\pi = \bigwedge_{i \in \mathrm{I}} t_i \neq s_i$ and variable x, define the* x-sub-constraint π_x *of π as*

$$\pi_x := \bigwedge_{i \in \mathrm{I}\ \wedge\ t_i = x} x \neq s_i.$$

Note that any non-trivial normal constraint π with $\mathrm{lvar}(\pi) = \{x_1, \dots, x_n\}$ can be partitioned into n x_i-sub-constraints $\pi_{x_1} \wedge \ldots \wedge \pi_{x_n}$.

Lemma 6.1.19. *A constraint π has no solution if and only if $\pi{\downarrow} = \bot$ or there exists an x-sub-constraint π_x of $\pi{\downarrow}$ without solutions.*

Proof. If $\pi{\downarrow} = \bot$, π has trivially no solutions. Let $\pi{\downarrow} = \pi_{x_1} \wedge \ldots \wedge \pi_{x_n}$ with $\mathrm{lvar}(\pi{\downarrow}) = \{x_1, \dots, x_n\}$.
"$\Rightarrow$". Assume there is no x-sub-constraint π_x of $\pi{\downarrow}$ without solutions. Thus, there are solutions $\delta_j = \{x_j \mapsto t_j\}$ for each π_{x_j}. Then, $\delta = \delta_1 \cup \ldots \cup \delta_m$ is a solution of $\pi{\downarrow}$, which contradicts the assumption.
"$\Leftarrow$". Assume δ is a solution of $\pi{\downarrow}$. Then, δ is also a solution of any x-sub-constraint π_x of $\pi{\downarrow}$, which contradicts the assumption. $\square$

Definition 6.1.20. *Define the set of* shallow instantiations *of x under signature Σ as the set of substitutions $\Theta_x^\Sigma = \{\{x \mapsto f(y_1, \dots, y_n)\} \mid f \in \mathcal{F}$ and $y_1, \dots, y_n$ linear$\}$.*

Note that if the set of function symbols $\mathcal{F}$ is infinite, Θ_x^Σ is also infinite, but then any constraint is trivially solvable if its normal form is different from $\bot$. In the context of ordered resolution, the function symbols appearing in the input clause set constitute a finite signature.

Lemma 6.1.21. *A constraint $\pi_x = \bigwedge_{i \in \mathrm{I}} x \neq s_i$ has no solution under signature Σ if and only if $\pi_x\sigma$ has no solution for every $\sigma \in \Theta_x^\Sigma$.*

Proof. "⇒". Assume δ is a solution of $\pi_x\sigma$ for some shallow instantiation σ. Then, $\sigma\delta$ is a solution of π_x, which contradicts the assumption.

"⇐". Assume δ is a solution of π_x. Then, $x\delta$ is ground and hence, $x\delta = f(y_1, \dots, y_n)\sigma$ for some function $f \in \mathcal{F}$ and substitution σ. Thus, σ is a solution of $\pi_x\{x \mapsto f(y_1, \dots, y_n)\} = \bigwedge_{i \in I} f(y_1, \dots, y_n) \neq s_i$, which contradicts the assumption. □

Theorem 6.1.22. *Emptiness of a straight dismatching constraint π is decidable.*

Proof. If $\mathcal{F}$ is not finite, a constraint π has no solution if and only if $\pi{\downarrow} = \bot$. Let $\pi{\downarrow} = \pi_{x_1} \wedge \dots \wedge \pi_{x_n}$ with $\text{lvar}(\pi{\downarrow}) = \{x_1, \dots, x_n\}$. Because each π_{x_i} has only finite sub-constraints, by the pigeon hole principle, there exists for each x_i a shallow instantiation $\sigma_i \in \Theta^{\Sigma}_{x_i}$ such that σ_i is a solution of π_{x_i}. Then, $\sigma_1 \cup \dots \cup \sigma_n$ is a solution of π.

If $\mathcal{F}$ is finite, using Lemma 6.1.19 and 6.1.21 recursively to check emptiness of π is sound and complete. In Lemma 6.1.19, the constraints π_x of the recursive calls have a depth smaller or equal than π. In Lemma 6.1.21, the $|\mathcal{F}|$ recursive calls on the constraints $\pi_x\sigma{\downarrow}$ with $\sigma \in \Theta^{\Sigma}_x$ have a depth strictly smaller than π_x. Therefore, the recursion terminates. □

Note that emptiness can be decided for straight dismatching constraints in $O(|\mathcal{F}| \cdot \text{size}(\pi))$ (See Section 6.1.2).

Corollary 6.1.23. *A constraint π is equivalent to $\pi \wedge x \neq t$, if $\pi\{x \mapsto t\}$ is empty.*

Proof. $\mathcal{D}^{\mathcal{V}}(\pi \wedge x \neq t) \subseteq \mathcal{D}^{\mathcal{V}}(\pi)$ holds trivially and $\mathcal{D}^{\mathcal{V}}(\pi) \subseteq \mathcal{D}^{\mathcal{V}}(\pi \wedge x \neq t)$ follows from Lemma 6.1.24. □

6.1.2 Operations on Constraints

In this section I consider implementation aspects of constraints including the representation of constraints, the solvability test and further operations on constraints.

Structure Sharing Looking again at constrained resolution, factoring, and normalization, note that the constraint-terms of the inference are all subterms of the constraints of the parent clauses. This means that throughout a saturation, every straight constraint term is a subterm of the constraints in the input set. Therefore, an implementation can take advantage of structure sharing. By storing the input constraints separately, an atomic constraint is representable as just a variable and a pointer to the respective subterm.

Furthermore, by adding for each n-ary function symbol f the unary functions $f_1, \dots, f_n$ and a constant symbol f_0, straight terms can be encoded and stored as an array of function symbols terminated by a constant symbol. For example, $f(x, f(a, y))$ and $f(x, f(y, z))$ become f_2, f_1, a_0 and f_2, f_0, respectively.

Sorting All of the operations on constraints can be applied on arbitrary constraints, but most of them become significantly faster if the constraints are sorted. Given total orderings $<_{\mathcal{X}}$ on $\mathcal{X}$ and $<_{\mathcal{F}}$ on $\mathcal{F}$, sort normal atomic constraints by the ordering defined by the following rules:

$$\begin{array}{lll}
x \neq [l] \quad < y \neq [r] & \text{if} & x <_{\mathcal{X}} y \\
x \neq f_i, [l] < x \neq g_j, [r] & \text{if} & f <_{\mathcal{F}} g \\
x \neq f_i, [l] < x \neq f_j, [r] & \text{if} & i <_{\mathbb{N}} j \\
x \neq f_i, [l] < x \neq f_i, [r] & \text{if} & 0 <_{\mathbb{N}} i \text{ and } x \neq [l] < x \neq [r]
\end{array}$$

The first operation that becomes faster after sorting is the fifth normalization rule (Definition 6.1.6). In a sorted constraint, $x \neq t$ will be immediately followed by any $x \neq t\sigma_1, \dots, x \neq t\sigma_n$, because any straight instance of a straight term $f_i, \dots, g_0$ has the form $f_i, \dots, g_j, \dots, h_0$.

While initially sorting a constraint takes linear-logarithmic time, sorting new constraints created from already sorted constraints is generally faster. Adding a fresh constraint and intersection require only a linear insert and merge operation, respectively.

Substitution and normalization, $\pi\{x \mapsto t\}{\downarrow}$, preserve the sorting if t is linear and variable disjoint from the left-hand variables of π. Otherwise, I can first bucket sort the constraint depending on the left-hand variables without changing the sorting of the sub-constraints with the same variables. Then, I can merge the individual partially sorted intervals of sub-constraints. For example,

$$\{x \neq f(t_1, z) \wedge \dots \wedge x \neq f(t_n, z) \wedge x \neq f(z, s_1) \wedge \dots \wedge x \neq f(z, s_m)\}\{x \mapsto f(y, y)\}$$

normalizes to $\{y \neq t_1 \wedge \dots \wedge y \neq t_n \wedge y \neq s_1 \wedge \dots \wedge y \neq s_m\}$, where $y \neq t_1$ to $y \neq t_n$ and $y \neq s_1$ to $y \neq s_m$ are still sorted and only need to be linearly merged to sort the normalized constraint.

Solvability Check Since clauses with an unsolvable constraint are tautologies, checking constraint solvability is an important means to avoid unnecessary inferences. Fortunately, solvability of straight dismatching constraints can be tested in linear time.

This is in contrast to general dismatching constraints [13] where solvability is NP-hard [14]. To illustrate this, SAT can be encoded as solvability of dismatching constraints with only ground terms on the right-hand side. For example, the SAT clause $C_i = x \vee \neg y \vee z$ is encoded by the atomic dismatching constraint $\pi_i = f_i(x, y, z) \neq f_i(\text{false}, \text{true}, \text{false})$. Then the general dismatching constraint $\pi_1 \wedge \cdots \wedge \pi_n$ as the result of the encoding of a clause set $\{C_1, \dots, C_n\}$ with signature $\Sigma = \{\text{true}, \text{false}\}$ is solvable if and only if the clause set $\{C_1, \dots, C_n\}$ is satisfiable.

Considering again Lemma 6.1.22, note that for any shallow instantiation $\sigma \in \Theta_x^\Sigma$ an atomic constraint $(x \neq t)\sigma$ normalizes to $\top$ except for exactly the one σ where both $x\sigma$ and t have the same top function symbol. In that case, normalization reduces the size of t by removing the top symbol of t. This means that in total every non-variable position in the straight right-hand terms of the constraint is considered exactly once for each $\sigma \in \Theta_x^\Sigma$. Solvability can therefore be decided in $O(|\mathcal{F}| \cdot \text{size}(\pi))$.

If the sub-constraints are sorted, the solvability check can be computed in-place, because each recursive call on some π_x and $\pi\sigma\downarrow$ applies to non-overlapping and consecutive sections in the sorted constraint. Recall that right hand sides of constraints do not share any variables. Now, looking at the overall recursive derivation tree generated along the proof of Lemma 6.1.22, the following invariant holds: Each subterm of the right hand sides of the initial constraint occurs at most once as a top level term on a right hand side constraint in the overall derivation tree. Solvability of a sorted constraint π can therefore be decided independently of the size of $\mathcal{F}$ in $O(\text{size}(\pi))$.

Furthermore, I can use intermediate results of the solvability check to simplify the constraint. If $\pi_x\{x \mapsto t\}$ has no solutions for some straight term t but $\pi_x\{x \mapsto t\}\downarrow \neq \bot$, I can add $x \neq t$ to π (Corollary 6.1.23) which replaces the sub-constraints in $\pi_x\{x \mapsto t\}\downarrow$ by Definition 6.1.6.5.

Example Consider the constrained clause set N consisting of

$$(P(x,x) \rightarrow \quad ; \; x \neq f(a)) \quad \text{and}$$
$$(P(f(y),z) \quad ; \; y \neq f(f(v)) \wedge z \neq f(f(a))).$$

Without the constraints, N is unsatisfiable because $\square$ can be inferred using the unifier

$$\sigma = \text{mgu}(P(x,x), P(f(y),z)) = \{x \mapsto f(y), z \mapsto f(y)\}.$$

Instead, use σ to analyze the constraints.

$$\begin{array}{rl} & y\sigma \neq f(f(v)) \wedge z\sigma \neq f(f(a)) \wedge x\sigma \neq f(a) \\ = & y \neq f(f(v)) \wedge f(y) \neq f(f(a)) \wedge f(y) \neq f(a) \\ \downarrow & y \neq f(f(v)) \wedge y \neq f(a) \wedge y \neq a \\ \text{sorted} & y \neq a \wedge y \neq f(a) \wedge y \neq f(f(v)) \end{array}$$

For the solvability check, use the signature $\{a/0, f/1\}$ given by the input.

$$\begin{array}{rl} & (y \neq a)\{y \mapsto a\} \text{ or } (y \neq f(a) \wedge y \neq f(f(v)))\{y \mapsto f(y)\}. \\ \Rightarrow & a \neq a \text{ or } y \neq a \wedge y \neq f(v). \\ \Rightarrow & \bot \text{ or } [(y \neq a)\{y \mapsto a\} \text{ or } (y \neq f(v))\{y \mapsto f(y)\}]. \\ \Rightarrow & \bot \text{ or } [a \neq a \text{ or } f(y) \neq f(v)]. \\ \Rightarrow & \bot \text{ or } [\bot \text{ or } \bot]. \\ \Rightarrow & \bot. \end{array}$$

Since the constraint is unsolvable, no resolution is performed and N is saturated. If the constraint were solvable, for example, without the $y \neq a$ constraint, I could replace $y \neq f(a) \wedge y \neq f(f(v))$ with $y \neq f(v)$ since $(y \neq f(a) \wedge y \neq f(f(v)))\{y \mapsto f(y)\}$ is unsolvable.

Many-sorted Signature I can further improve the chance to identify unsolvable constraints by using many-sorted signatures. Then, the shallow instantiations θ_x^Σ

only range over the function symbols in the sort of variable x. In array-theories, for example, data, indices, and arrays are usually modelled to be distinct from each other. A constraint on an index-variable is therefore already unsolvable if just all ground terms of the index-sort are excluded. Without sort information, such a constraint is still solvable.

An algorithm to extract a many-sorted signature with a maximal number of sorts was introduced in [10]. At first, all function and predicate symbols start with different sorts. Then, compute equivalence classes of sorts that should be equal for the problem to be well-sorted: Wherever a function symbol is the argument of another function or predicate, their result and argument sorts are the same and for each variable in each clause, the sorts of its occurrences are also the same. Using union-find, the whole algorithm has linear complexity.

For first-order logic without equality, the resulting sorts are always *monotone* [11], which means that the unsorted and many-sorted versions of the same problem are satisfiability equivalent. As a consequence, the resolution calculus can continue to use the unsorted signature while the constraint operations use the corresponding many-sorted signature.

Subset Check Another important property of straight dismatching constraints is the ability to efficiently check whether the solutions of one constraint are a subset of the solutions of another constraint. The subset check allows common redundancy eliminations such as subsumption, condensation, and subsumption resolution to account for constraints. E.g., $(P(x); x \neq a)$ subsumes $(P(x), Q(y); x \neq a \wedge y \neq b)$, while $(Q(x); x \neq a)$ does not.

If $\pi\sigma{\downarrow} = \top$ the subset condition on the solutions is trivially fulfilled, but even for general constraints, I can decide the subset relation.

Lemma 6.1.24. *Let π and π' be solvable normal constraints. Then $\mathcal{D}^{\mathcal{V}}(\pi) \subseteq \mathcal{D}^{\mathcal{V}}(\pi')$ if and only if $\pi\{x \mapsto t\}$ has no solutions for every sub-constraint $x \neq t$ in π'.*

Proof. "$\Rightarrow$". Assume there exists a sub-constraint $x \neq t$ in π', such that δ is a solution of $\pi\{x \mapsto t\}$. Then, $\{x \mapsto t\}\delta$ is a solution of π. Because $x\{x \mapsto t\}\delta = t\delta$ is an instance of t, $\{x \mapsto t\}\delta$ is not a solution of π'. This contradicts the assumption that $\mathcal{D}^{\mathcal{V}}(\pi) \subseteq \mathcal{D}^{\mathcal{V}}(\pi')$.

"$\Leftarrow$". Assume $\mathcal{D}^{\mathcal{V}}(\pi) \nsubseteq \mathcal{D}^{\mathcal{V}}(\pi')$. Let δ be a solution of π, but not of π'. There exists a sub-constraint $x \neq t$ of π' such that $x\delta$ is an instance of t. Hence, there is a substitution σ such that $\delta = \{x \mapsto t\}\sigma$. Then, σ is a solution of $\pi\{x \mapsto t\}$. This contradicts the assumption that $\pi\{x \mapsto t\}$ has no solutions. □

Note that if π was fully simplified according to Corollary 6.1.23, then $\pi\{x \mapsto t\}$ has no solutions if and only if $\pi\{x \mapsto t\}{\downarrow} = \bot$. This means $\pi\{x \mapsto t\}$ is unsolvable if there exists a constraint $x \neq s \in \pi$ such that t is an instance of s. If both π and π' are also sorted, I can therefore implement the subset check similar to a merge in merge-sort in $O(\text{size}(\pi) + \text{size}(\pi'))$.

Using this subset test, I can check whether a constrained clause $(C\sigma; \pi_1)$ is an instance of $(C; \pi_2)$. If this is the case for two clauses in a clause set N, I can remove $(C; \pi)$ from N without changing the ground instances of N.

Corollary 6.1.25. *$\mathcal{G}((C\sigma; \pi_1)) \subseteq \mathcal{G}((C; \pi_2))$ if and only if $\mathcal{D}^{\mathcal{V}}(\pi_1) \subseteq \mathcal{D}^{\mathcal{V}}(\pi_2\sigma)$.*

Proof. "$\Rightarrow$". If $\mathcal{D}^{\mathcal{V}}(\pi_1)$ is empty, then $\mathcal{D}^{\mathcal{V}}(\pi_1) \subseteq \mathcal{D}^{\mathcal{V}}(\pi_2\sigma)$ is trivial. Otherwise, let $\delta \in \mathcal{D}^{\mathcal{V}}(\pi_1)$ be arbitrary. Then, $C\sigma\delta \in \mathcal{G}((C\sigma; \pi_1))$ and thus, $C\sigma\delta \in \mathcal{G}((C; \pi_2))$. Hence, $\sigma\delta$ is a solution of π_2 and δ is a solution of $\pi_2\sigma$. Thus, $\mathcal{D}^{\mathcal{V}}(\pi_1) \subseteq \mathcal{D}^{\mathcal{V}}(\pi_2\sigma)$.

"$\Leftarrow$". If $\mathcal{G}((C\sigma; \pi_1))$ is empty, then $\mathcal{G}((C\sigma; \pi_1)) \subseteq \mathcal{G}((C; \pi_2))$ is trivial. Otherwise, let $C\sigma\delta \in \mathcal{G}((C\sigma; \pi_1))$ be arbitrary. Then, $\delta \in \mathcal{D}^{\mathcal{V}}(\pi_1)$ and thus, $\delta \in \mathcal{D}^{\mathcal{V}}(\pi_2\sigma)$. Hence, $\sigma\delta$ is a solution of π_2 and $C\sigma\delta \in \mathcal{G}((C; \pi_2))$. Thus, $\mathcal{G}((C\sigma; \pi_1)) \subseteq \mathcal{G}((C; \pi_2))$. □

Definition 6.1.26. *A clause $(C; \pi_1)$ subsumes $(D; \pi_2)$ if there is a substitution σ such that $C\sigma \subset D$ and $\mathcal{D}^{\mathcal{V}}(\pi_2) \subseteq \mathcal{D}^{\mathcal{V}}(\pi_1\sigma)$.*

Lemma 6.1.27. *If $(C; \pi_1)$ subsumes $(D; \pi_2)$, $(D; \pi_2)$ is redundant in $N \cup \{(C; \pi_1)\}$.*

Proof. Let $D\delta \in \mathcal{G}((D; \pi_2))$. δ is a solution of $\pi_1\sigma$ and hence, $C\sigma\delta \in \mathcal{G}((C; \pi))$. Since $C\sigma\delta \subset D\delta$, $C\sigma\delta \in \mathcal{G}((C; \pi_1))^{\prec D\delta}$ and $C\sigma\delta \models D\delta$. Therefore, $(D; \pi_2)$ is redundant in $N \cup \{(C; \pi_1)\}$. □

Ordering In ordered resolution with selection the ordering on atoms holds a key role in the efficiency of the calculus. Just as their unconstrained counterparts, the constrained resolution and factoring rules (Definitions 6.2.3 and 6.2.4) are only applied if the unified literals are (strictly) maximal. This means that redundant inferences can be avoided if the literals are not maximal under the constraint.

In general, any traditional ordering can be used by ignoring the constraints. If an atom is not (strictly) maximal in the clausal part, it is not (strictly) maximal for any ground instance including the ones solving the constraint. On the other hand, the constraint could be excluding all ground instances where the atom is (strictly) maximal. For example, the lexicographic path ordering (LPO) can be extended with straight dismatching constraints. Similar extensions are possible for the RPO and KBO.

Definition 6.1.28 (LPO with Constraints)**.** *Let s, t be terms, π a constraint and $\mathcal{F}$ finite. The constrained lexicographic path ordering is defined by the transition system on pairs $(s \prec t; \pi)$ given in Figure 6.1*

Lemma 6.1.29. *Constrained lexicographic path ordering is well-defined.*

Proof. The first to third rule overlap with LPO: In each, the depth of s or t decreases, while the constraint remains the same. The fourth and last rules are trivially terminating. For the fifth and sixth rule, size$(\pi\sigma\!\downarrow) <$ size(π). Therefore, the constrained lexicographic path ordering is well-defined. □

$$
\begin{aligned}
(f(\overline{s}) \prec g(t_1,\dots,t_m);\pi) &\Rightarrow \bigvee_{1\le i\le m} (f(\overline{s}) \preceq t_i;\pi) && \text{if } g \prec f\\
(f(s_1,\dots,s_n) \prec g(\bar{t});\pi) &\Rightarrow \bigwedge_{1\le i\le n} (s_i \prec g(\bar{t});\pi) && \text{if } f \prec g\\
(f(t_1,..,t_{i-1},s_i,..,s_n) \prec f(\bar{t});\pi) &\Rightarrow \bigwedge_{i<j\le n} (s_j \prec f(\bar{t});\pi) && \wedge\, (s_i \preceq t_i;\pi)\\
(x \prec g(\bar{t});\pi) &\Rightarrow \top && \text{if } x \in \mathrm{vars}(g(\bar{t}))\\
(s \prec x;\pi) &\Rightarrow \bigwedge_{\sigma\in\theta^\Sigma_x} (s\sigma \prec x\sigma;\pi\sigma{\downarrow}) && \text{if } x \in \mathrm{lvar}(\pi)\\
(x \prec t;\pi) &\Rightarrow \bigwedge_{\sigma\in\theta^\Sigma_x} (x\sigma \prec t\sigma;\pi\sigma{\downarrow}) && \text{if } x \in \mathrm{lvar}(\pi)\\
(s \prec t;\bot) &\Rightarrow \top
\end{aligned}
$$

Figure 6.1: The LPO transition system for constrained terms.

Note that the first four rules directly follow the definition of $\prec_{\text{lpo}}$ and therefore $(s \prec t;\top)$ is the same as computing $s \prec_{\text{lpo}} t$. The fifth and sixth rule reuse the principle idea of the solvability check to instantiate π until it normalizes to $\top$ or $\bot$. In the latter case, the last rule applies to remove instantiations that are excluded by the constraint. For example, consider $(a \prec x\ ;\ x \neq a)$ with precedence $a \prec b \prec f \prec g$. The only ground case where $a \not\prec_{\text{lpo}} x$ is $(a \prec a\ ;\ a \neq a)$, but the constraint $a \neq a$ is unsolvable. Therefore, $a \prec_{\text{lpo}} x$ under constraint $x \neq a$.

Furthermore, note that $(a \prec b\ ;\ b \neq a) \Rightarrow^* \top$ directly implies that also $(a \prec f(x)\ ;\ f(x) \neq a) \Rightarrow^* \top$ and $(a \prec g(x,y)\ ;\ g(x,y) \neq a) \Rightarrow^* \top$. In general, only the "minimal" solution of π_x needs to be checked in the fifth rule. Analogously, the "maximal" solution of π_x suffices for the sixth rule, if it exists. A solution δ of π_x is minimal (maximal) if for every solution δ' of π_x, $x\delta \preceq_{\text{lpo}} x\delta'$ ($x\delta \succeq_{\text{lpo}} x\delta'$). For example, the constraint $x \neq a \wedge x \neq f(b) \wedge x \neq f(f(u))$ has under signature $a \prec b \prec f$ the minimal solution $\{x \mapsto b\}$ and maximal solution $\{x \mapsto f(a)\}$, but no maximal solution exists under signature $a \prec b \prec f \prec g$. If there is no maximal solution, it means that arbitrarily large terms are solutions for π_x. Then, I can immediately conclude $(x \prec t;\pi) \Rightarrow \bot$ if $x \notin \mathrm{vars}(t)$. Therefore, I can refine these rules to

$$
\begin{aligned}
(s \prec x;\pi) &\Rightarrow (s\delta \prec x\delta;\pi\delta{\downarrow}) && \text{if } x \in \mathrm{lvar}(\pi) \text{ and } \delta \text{ is the min. solution of } \pi_x\\
(x \prec t\,;\pi) &\Rightarrow (x\delta \prec t\delta;\pi\delta{\downarrow}) && \text{if } x \in \mathrm{lvar}(\pi) \text{ and } \delta \text{ is the max. solution of } \pi_x
\end{aligned}
$$

Just like the solvability check, minimal and maximal solutions can be computed in linear time. Unless an earlier case allows an early termination, the previous rules would eventually generate all cases for solutions of x including the minimal and maximal. Generating the cases alone corresponds to the work required to find the minimal or maximal solution.

Lemma 6.1.30. *$s\delta \prec_{\text{lpo}} t\delta$ for every solution δ of π iff $(s \prec t;\pi) \Rightarrow^* \top$.*

Proof. "$\Rightarrow$": By induction on the derivation $(s \prec t;\pi) \Rightarrow^* \top$.

The first four rules follow directly from the definition of $\prec_{\text{lpo}}$. For example, consider the first rule. Let $(f(\overline{s}) \prec g(t_1,\ldots,t_m);\pi) \Rightarrow \bigvee_{1\leq i\leq m}(f(\overline{s}) \preceq t_i;\pi) \Rightarrow^* \top$. Then, $(f(\overline{s}) \preceq t_i;\pi) \Rightarrow^* \top$ for at least one $1 \leq i \leq m$. By the inductive hypothesis, $f(\overline{s})\delta \prec_{\text{lpo}} t_i\delta$ for every solution δ of π. Therefore by the definition of $\prec_{\text{lpo}}$, $f(\overline{s})\delta \prec_{\text{lpo}} g(t_1,\ldots,t_m)\delta$ for every solution δ of π.

For the fifth rule, let $(s \prec x;\pi) \Rightarrow \bigwedge_{\sigma\in\theta_x^\Sigma}(s\sigma \prec x\sigma;\pi\sigma) \Rightarrow^* \top$. Assume there is a solution δ of π such that $s\delta \not\prec_{\text{lpo}} x\delta$. Then, $x\delta = f(y_1,\ldots,y_n)\delta'$ for some substitution δ' and there is a $\sigma = \{x \mapsto f(y_1,\ldots,y_n)\} \in \Theta_x^\Sigma$. Since δ' is a solution of $\pi\sigma$, $s\sigma\delta' \not\prec_{\text{lpo}} x\sigma\delta'$ contradicts the inductive hypothesis on $(s\sigma \prec x\sigma;\pi\sigma) \Rightarrow^* \top$. The sixth rule is analogous. The last rule is trivial as $\bot$ has no solutions.

"$\Leftarrow$": Let $(s \prec t;\pi) \not\Rightarrow^* \top$, i.e., there is a normal form $(s \prec t;\pi) \Rightarrow^* F$, where F is either $\bot$ or a formula consisting of conjunctions and disjunctions of pairs $(s \prec t;\pi)$ where no rule can be applied. I prove by induction on the derivation $(s \prec t;\pi) \Rightarrow^* F$, that there is a solution δ of π such that $s\delta \not\prec_{\text{lpo}} t\delta$.

Again, the first four rules follow directly from the definition of $\prec_{\text{lpo}}$. E.g., consider the second rule. Let $(f(s_1,\ldots,s_n) \prec g(\overline{t});\pi) \Rightarrow \bigwedge_{1\leq i\leq n}(s_i \preceq g(\overline{t});\pi) \not\Rightarrow^* \top$. Then, $(f(s_1,\ldots,s_n) \preceq t_i;\pi) \not\Rightarrow^* \top$ for at least one $1 \leq i \leq n$. By the inductive hypothesis, $s_i\delta \not\prec_{\text{lpo}} g(\overline{t})\delta$ for a solution δ of π. Therefore, by the definition of $\prec_{\text{lpo}}$, $f(s_1,\ldots,s_n)\delta \not\prec_{\text{lpo}} g(\overline{t})\delta$ for the solution δ of π.

For the fifth rule, let $(s \prec x;\pi) \Rightarrow \bigwedge_{\sigma\in\theta_x^\Sigma}(s\sigma \prec x\sigma;\pi\sigma) \not\Rightarrow^* \top$. Then, $(s\sigma \prec x\sigma;\pi\sigma) \not\Rightarrow^* \top$ for at least one $\sigma \in \Theta_x^\Sigma$. By the inductive hypothesis, $s\sigma\delta \not\prec_{\text{lpo}} x\sigma\delta$ for a solution δ of $\pi\sigma$. Therefore, $s\sigma\delta \not\prec_{\text{lpo}} x\sigma\delta$ for the solution $\sigma\delta$ of π. The sixth rule is analogous and the last rule contradicts $(s \prec t;\pi) \not\Rightarrow^* \top$. □

Constrained *LPO* extends to atoms and literals in the usual way.

Corollary 6.1.31. *Let $(C \vee E;\pi)$ be a constrained clause. If $(E \prec L;\pi) \Rightarrow^* \top$ for some literal L in C, then the literal E is not maximal in $(C \vee E;\pi)$ under $\prec_{\text{lpo}}$. If $(E \preceq L;\pi) \Rightarrow^* \top$ for some literal L in C, then the literal E is not strictly maximal in $(C \vee E;\pi)$ under $\prec_{\text{lpo}}$.*

Union While intersection of two constraints is as straightforward as building their conjunction, their union can generally not be expressed as a single constraint. For example, consider the constrained clauses $(P(x,y);x \neq a)$ and $(P(x,y);y \neq a)$ with the signature $\{a,b\}$. Together they have $P(a,b)$, $P(b,a)$, and $P(b,b)$ as ground instances, but there is no constrained clause $(P(x,y);\pi)$ with the same ground clause set.

For an x-sub-constraint π_x, the emptiness check can also produce the complement. Every time the algorithm reaches a point where the emptiness check

(Lemma 6.1.21) returns non-empty, i.e, $\pi_x\{x \mapsto t\}$ normalizes to $\top$, add instead $x \neq t$ to the complement and continue. From De Morgan's Law then follows the union of x-sub-constraints and subtraction $\pi \setminus \pi_x$.

While general unification is not possible, I can, however, use the subset test to simplify constraints in some cases.

Lemma 6.1.32 (Union Simplification)**.** *Let $\pi_1 \wedge x \neq t$ and π_2 be solvable constraints. If $\mathcal{D}^{\mathcal{V}}(\pi_1\{x \mapsto t\}) \subseteq \mathcal{D}^{\mathcal{V}}(\pi_2\{x \mapsto t\})$, then $\mathcal{D}^{\mathcal{V}}(\pi_1 \wedge x \neq t) \cup \mathcal{D}^{\mathcal{V}}(\pi_2) = \mathcal{D}^{\mathcal{V}}(\pi_1) \cup \mathcal{D}^{\mathcal{V}}(\pi_2)$.*

Proof. The part for $\mathcal{D}^{\mathcal{V}}(\pi_1 \wedge x \neq t) \cup \mathcal{D}^{\mathcal{V}}(\pi_2) \subseteq \mathcal{D}^{\mathcal{V}}(\pi_1) \cup \mathcal{D}^{\mathcal{V}}(\pi_2)$ is trivial. Assume $\mathcal{D}^{\mathcal{V}}(\pi_1) \cup \mathcal{D}^{\mathcal{V}}(\pi_2) \nsubseteq \mathcal{D}^{\mathcal{V}}(\pi_1 \wedge x \neq t) \cup \mathcal{D}^{\mathcal{V}}(\pi_2)$. Let δ be a solution of π_1 but not of either $\pi_1 \wedge x \neq t$ or π_2. Then $x\delta$ is an instance of t and thus, $x\delta = x\{x \mapsto t\}\sigma$ for some grounding substitution σ. σ is a solution of $\pi_1\{x \mapsto t\}$ and by assumption, a solution of $\pi_2\{x \mapsto t\}$. Therefore, δ is a solution of π_2 which contradicts the assumption. □

Constraint Removal Lastly, any set of constraint clauses can be transformed into an equivalent set of unconstrained clauses.

Definition 6.1.33. *Let N be a set of constrained clauses and $\mathcal{F}$ finite. Define the specific instantiations $\mathcal{S}(N)$ of N under Σ as the normal form of N under the following transformation rules:*

$$N \,\dot\cup\, \{(C; \bot)\} \Rightarrow N$$
$$N \,\dot\cup\, \{(C; \pi)\} \Rightarrow N \cup \{(C\sigma; \pi\sigma{\downarrow}) \mid \sigma \in \Theta_x^\Sigma\}, \text{ where } x \in \mathrm{lvar}(\pi).$$

Lemma 6.1.34. *$\mathcal{S}(N)$ is equivalent to N and for every $(C; \pi) \in \mathcal{S}(N)$, $\pi = \top$.*

Proof. First, prove that the transformation rules are satisfiability equivalent. Let $C\delta \in \mathcal{G}((C; \pi))$ and $x\delta = f(y_1, \ldots, y_n)\delta'$. Then, $\sigma = \{x \mapsto f(y_1, \ldots, y_n)\}$ is in Θ_x^Σ and δ' is a solution of $\pi\sigma$. Therefore, $C\delta \in \mathcal{G}((C\sigma; \pi\sigma))$ which is a subset of $N \cup \{(C\sigma; \pi\sigma) \mid \sigma \in \Theta_x^\Sigma$ and $\pi\sigma$ is solvable$\}$. The remaining cases are trivial.

The first rule, reduces the number of clause, while in the second rule for each $(C\sigma; \pi\sigma)$, the multiset of sub-constraint depths in $\pi\sigma{\downarrow}$ is smaller than in π. Therefore, exhaustive transformation terminates. Since $(C; \top) \in N$ is the only case not matched by any rule, the normal form can only contain such clauses.

□

Corollary 6.1.35. *$\mathcal{G}((C; \pi)) = \emptyset$ if and only if $\mathcal{S}(\{(C; \pi)\}) = \emptyset$.*

6.1.3 Matching Constraints

In this section, I will briefly explore the logical extension of straight dismatching constraints with their dual matching constraints. As opposed to dismatching constraints, a matching constraint $x \doteq t$ has a solutions δ if $x\delta$ is an instance of t.

Refinement with constraints (see Section 6.5) segments a clause $(C; \pi)$ into $(C; \pi \wedge x \neq s)$ and $(C; \pi)\{x \mapsto s\}$. Because the first segment does not change the clausal part, the approximation for $(C; \pi \wedge x \neq s)$ is analogous to $(C; \pi)$. On the other hand, after instantiating x with s, $(C; \pi)\{x \mapsto s\}$ likely requires additional approximation steps on s and its subterms. This increases the size of the approximation and adds more predicates, putting an additional burden on the solver. Instead, I could use $(C; \pi \wedge x \doteq s)$ with a matching constraint which again avoids these additional approximation steps.

Additionally, consider the clause $Q(f(x)) \rightarrow P(f(f(x)))$ which requires a Shallow transformation to extract the subterm $f(x)$. In its place, the equivalent clause $(Q(y) \rightarrow P(f(y)); y \doteq f(x))$ is already shallow. Specifically, this always works for subterms that are variable disjoint from the rest of the clause.

Note that a constrained clause $(C; x \doteq t)$ is trivially equivalent to $C\{x \mapsto t\}$ assuming C and t are variable disjoint. Hence, the use of matching constraints is, aside from the above mentioned cases, rather limited. For the sake of simplicity, they will therefore not be considered in the theory outside this section.

Matching constraints, solutions, and normalization are defined analogously to Section 6.1.

Definition 6.1.36 (Matching Constraint). *A matching constraint ψ is of the form*

$$\bigwedge_{i \in \mathrm{I}} s_i \doteq t_i$$

where I *is a finite set of indices and each t_i is linear.*

Definition 6.1.37 (Solutions). *Let ψ be a matching constraint, Σ a signature and* $\mathrm{lvar}(\psi) \subseteq \mathcal{V}$. *A solution of ψ over $\mathcal{V}$ is a grounding substitution δ over $\mathcal{V}$ such that for all $i \in \mathrm{I}$, $s_i\delta$ is an instance of t_i.*

Definition 6.1.38 (Constraint Normalization). *Define constraint normalization $\psi\!\downarrow$ as the normal form of the following rewriting rules over constraints.*

1	$\psi \wedge f(s_1, \ldots, s_n) \doteq y \Rightarrow \psi$		
2	$\psi \wedge f(s_1, \ldots, s_n) \doteq f(t_1, \ldots, t_n) \Rightarrow \psi \wedge \bigwedge_{1 \leq i \leq n} s_i \doteq t_i$		
3	$\psi \wedge f(s_1, \ldots, s_n) \doteq g(t_1, \ldots, t_m) \Rightarrow \bot$	*if* $f \neq g$	
4	$\psi \wedge x \doteq t \wedge x \doteq s \Rightarrow \psi \wedge x \doteq t\sigma$	*if* $\sigma = \mathrm{mgu}(s, t)$	
5	$\psi \wedge x \doteq t \wedge x \doteq s \Rightarrow \bot$	*if* $\neg\exists\sigma = \mathrm{mgu}(s, t)$	

Definition 6.1.39 (Constraint Normal Form). *A matching constraint $\bigwedge_{i \in \mathrm{I}} x_i \doteq t_i$ is called normal if the x_i are pairwise distinct variables.*

Lemma 6.1.40. *$\psi\!\downarrow$ is a normal matching constraint and equivalent to ψ.*

Proof. In the first rule, any solution satisfies $f(s_1, \ldots, s_n) \doteq y$ since $f(s_1, \ldots, s_n)$ is already an instance of y. Hence, it can be dropped reducing the number of sub-constraints. In the second rule, a grounding substitution δ is a solution of $f(s_1, \ldots, s_n) \neq f(t_1, \ldots, t_n)$, if and only if each $s_i\delta$ is an instance of t_i. Thus, the sub-constraint is equivalent to $s_1 \doteq t_1 \wedge \ldots \wedge s_n \doteq t_n$. In this case, the maximal depth of the new sub-constraints decreases by one compared to $f(s_1, \ldots, s_n) \neq f(t_1, \ldots, t_n)$. In the third rule, $f(s_1, \ldots, s_n)$ can never be an instance of $g(t_1, \ldots, t_m)$ for any grounding substitution. Hence, $\psi \wedge f(s_1, \ldots, s_n) \doteq g(t_1, \ldots, t_m)$ has no solutions, which is equivalent to $\bot$ and terminates the normalization. Let δ be a solution of $\psi \wedge x \doteq t \wedge x \doteq s$. Then, $x\delta$ is an instance of both t and s. If s and t have no unifier, this is impossible and hence, $\psi \wedge x \doteq t \wedge x \doteq s$ is equivalent to $\bot$ and terminates the normalization. Otherwise, let $\sigma = \mathrm{mgu}(s, t)$.

Then, $x\delta$ is an instance of $t\sigma$ and hence, δ is a solution of $\psi \wedge x \doteq t\sigma$. This step reduces the number of sub-constraints, while $\mathrm{depth}(t\sigma) = \max(\mathrm{depth}(s), \mathrm{depth}(t))$. Let δ be a solution of $\psi \wedge x \doteq t\sigma$ with $\sigma = \mathrm{mgu}(s, t)$. Then, $x\delta$ is an instance of $t\sigma$ and thus, also of both t and s. Hence, δ is a solution of $\psi \wedge x \doteq t \wedge x \doteq s$.

All rules together terminate and cover every case where the left-hand side of a sub-constraint is not a variable or two left-hand side variables are the same. Hence, applying the five rules exhaustively results in a normal form that is a normal matching constraint. $\square$

Note that there are two significant differences to dismatching constraints. The first is that in an atomic constraint $x \doteq t$, t is not required to be straight, only linear. The second difference is that a normal matching constraint ψ contains at most one sub-constraint per variable. This allows combing normal matching and dismatching constraints with little overhead compared to just dismatching constraints. I call a combination of matching and dismatching constraints $\psi \wedge \pi$ a mixed constraint.

Definition 6.1.41 (Constrained Clauses)**.** *A clause with mixed constraints* $(C; \psi \wedge \pi)$ *is called a constrained clause. A constrained clause is normal, if* ψ *and* π *are normal and* $\mathrm{lvar}(\psi) \cup \mathrm{lvar}(\pi) \subseteq \mathrm{vars}(C)$*. For a given signature* Σ, $\mathcal{G}((C; \pi)) = \{C\delta \mid \delta \in \mathcal{D}^{\mathcal{V}}(\psi), \delta \in \mathcal{D}^{\mathcal{V}}(\pi)\}$ *is called the set of ground instances of* $(C; \psi \wedge \pi)$*.*

Also, I can simplify mixed constraints even further.

Lemma 6.1.42 (Mixed Simplification)**.** *Let* $\psi' \wedge x \doteq s \wedge x \neq t \wedge \pi' = \psi \wedge \pi$ *be a mixed constraint. If* $\mathrm{mgu}(s, t)$ *does not exist, then* $\psi \wedge \pi$ *is equivalent to* $\psi \wedge \pi'$*. If* s *is an instance of* t*, then* $\psi \wedge \pi$ *is equivalent to* $\bot$*.*

Proof. Let δ be a solution of $\psi \wedge \pi'$ and $\mathrm{mgu}(s, t)$ does not exist. Then, $x\delta$ is an instance of s. Since δ is not a unifier of s and t, $x\delta$ is not an instance of t. Thus, δ is a solution of $\psi \wedge x \neq t \wedge \pi' = x \neq t \wedge$. The reverse direction is trivial.

Assume δ is a solution of $\psi \wedge \pi$. Then, $x\delta$ is an instance of s and $x\delta$ is not an instance of t. This contradicts the assumption that s is an instance of t. $\square$

Lemma 6.1.43. *A mixed normal constraint* $\psi \wedge x \doteq t \wedge \pi$ *has no solutions if and only if* $\psi \wedge \pi\{x \mapsto t\}$ *has no solutions.*

Proof. Let δ be a solution of $\psi \wedge x \doteq t \wedge \pi$. Then, $x\delta$ is an instance of t and thus, $\delta = \{x \mapsto t\}\delta'$ for some substitution δ'. Since $\psi \wedge x \doteq t$ is normal, $x \notin \mathrm{lvar}(\psi)$. Therefore, δ' is a solution of ψ and a solution of $\pi\{x \mapsto t\}$.

Let δ be a solution of $\psi \wedge \pi\{x \mapsto t\}$. Since $x \notin \mathrm{lvar}(\psi)$, $\{x \mapsto t\}\delta$ is a solution of ψ and π. Further, $x\{x \mapsto t\}\delta = t\delta$ is an instance of t. Therefore, $\{x \mapsto t\}\delta$ is a solution of $\psi \wedge x \doteq t \wedge \pi$. □

Via Lemma 6.1.43, the emptiness check (Lemma 6.1.22) can be reused for mixed constraints. Decidability (Lemma 6.2.8) and refinement (Lemma 6.5.6) can also be straightforwardly extended to matching constraints. Everything else never uses constraints directly and is therefore unaffected by the addition of matching constraints.

Lastly, this leads to Lemma 6.1.44 which enables the two improvements mentioned at the beginning of this section.

Lemma 6.1.44 (Match Extraction). *The clauses $(C; \psi \wedge \pi)$ and $(C'; x \doteq t \wedge \psi' \wedge \pi')$ are equivalent if there exists a term s such that the term t is an instance of s and $(C'; x \doteq t \wedge \psi' \wedge \pi')\{x \mapsto s\} = (C; \psi \wedge \pi)$.*

Proof. Let $C'\delta$ be a ground instance of $(C'; x \doteq t \wedge \psi' \wedge \pi')$. Then $x\delta$ is an instance of t and by assumption, also an instance of s. Thus, $\delta = \{x \mapsto s\}\delta'$ for some grounding substitution δ'. Then, δ' is a solution of $(x \doteq t \wedge \psi' \wedge \pi')\{x \mapsto s\}$. Therefore, $C'\{x \mapsto s\}\delta' = C\delta'$ is a ground instance of $(C'\{x \mapsto s\}; (x \doteq t \wedge \psi' \wedge \pi')\{x \mapsto s\}) = (C; \psi \wedge \pi)$. The reverse direction is trivial, because $(C; \psi \wedge \pi)$ is a instance of $(C'; x \doteq t \wedge \psi' \wedge \pi')$. □

Lemma 6.1.44 shows that $(C; \pi)\{x \mapsto t\}$ and $(C; \pi \wedge x \doteq t)$ are equivalent, if t is linear and variable disjoint from C. Furthermore, for example, the clauses

$$(Q(g(x,y)) \to P(f(g(x,y))); \;\; x \doteq f(v) \wedge y \neq a)$$
$$(Q(z) \to P(f(z)); \;\; z \doteq g(f(v),w) \wedge z \neq g(v,a))$$

are equivalent because instantiating the latter with $\{z \mapsto g(x,y)\}$ yields the former.

6.2 Decidability of the MSL(SDC) Fragment

In the following I will show that the satisfiability of the MSL(SDC) fragment is decidable. For this purpose I will define ordered resolution with selection on constrained clauses and show that with an appropriate ordering and selection function, saturation of an MSL(SDC) clause set terminates.

For the remainder of this section I assume an atom ordering $\prec$ such that a literal $\neg Q(s)$ is not greater than a literal $P(t[s]_p)$, where $p \neq \varepsilon$. For example, an LPO with a precedence where functions are larger than predicates or a KBO where all symbols have weight one have this property.

Definition 6.2.1 (sel)**.** *Let* $(C;\pi) = (S_1(t_1),\ldots,S_n(t_n) \rightarrow P_1(s_1),\ldots,P_m(s_m);\pi)$ *be an MSL(SDC) clause. The Superposition Selection function* sel *is defined by* $S_i(t_i) \in \mathrm{sel}((C;\pi))$ *if*

(1) t_i *is not a variable or*

(2) $t_1,\ldots,t_n$ *are variables and* $t_i \notin vars(s_1,\ldots,s_m)$ *or*

(3) $\{t_1,\ldots,t_n\} \subseteq vars(s_1,\ldots,s_m)$ *and for some* $1 \leq j \leq m$*,* $s_j = t_i$*.*

The selection function sel (Definition 6.2.1) functions the same as sel defined in Definition 5.1.1.

Definition 6.2.2. *A literal A is called* [strictly] maximal *in a constrained clause* $(C \vee A;\pi)$ *if and only if there exists a solution* δ *of* π *such that for all literals B in C,* $B\delta \preceq A\delta$ $[B\delta \prec A\delta]$.

Definition 6.2.3 (Ordered SDC-Resolution with Selection)**.**

$$\frac{(\Gamma_1 \rightarrow \Delta_1, A\ ;\ \pi_1) \qquad (\Gamma_2, B \rightarrow \Delta_2\ ;\ \pi_2)}{((\Gamma_1,\Gamma_2 \rightarrow \Delta_1,\Delta_2)\sigma\ ;\ (\pi_1 \wedge \pi_2)\sigma\!\downarrow)}\ ,\textit{if}$$

1. $\sigma = \mathrm{mgu}(A,B)$*;*
2. $A\sigma$ *is strictly maximal in* $(\Gamma_1 \rightarrow \Delta_1, A;\pi_1)\sigma$ *and* $\mathrm{sel}(\Gamma_1 \rightarrow \Delta_1, A) = \emptyset$*;*
3. $B \in \mathrm{sel}(\Gamma_2, B \rightarrow \Delta_2)$
   or $\mathrm{sel}(\Gamma_2, B \rightarrow \Delta_2) = \emptyset$ *and* $\neg B\sigma$ *maximal in* $(\Gamma_2, B \rightarrow \Delta_2;\pi_2)\sigma$*;*
4. $(\pi_1 \wedge \pi_2)\sigma\!\downarrow$ *is solvable.*

Definition 6.2.4 (Ordered SDC-Factoring with Selection)**.**

$$\frac{(\Gamma \rightarrow \Delta, A, B\ ;\ \pi)}{((\Gamma \rightarrow \Delta, A)\sigma;\pi\sigma\!\downarrow)}\ ,\textit{if}$$

1. $\sigma = \mathrm{mgu}(A,B)$*;*
2. $\mathrm{sel}(\Gamma \rightarrow \Delta, A, B) = \emptyset$*;*
3. $A\sigma$ *is maximal in* $(\Gamma \rightarrow \Delta, A, B;\pi)\sigma$
4. $\pi\sigma\!\downarrow$ *is solvable.*

Note that while the above rules do not operate on equations, they actually allow unit clauses that consist of non-unifiable disequations, i.e., clauses $s \approx t \rightarrow$ where s and t are not unifiable. There are no potential superposition inferences on such clauses as long as there are no positive equations. So resolution and factoring suffice for completeness. Nevertheless, clauses such as $s \approx t \rightarrow$ affect the models of satisfiable problems. Constrained Resolution and Factoring are sound.

Lemma 6.2.5 (Soundness)**.** *Ordered SDC-Resolution and -Factoring with Selection are sound.*

Proof. Let the clause $(\Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2)\sigma\delta$ be a ground instance of the resolvent $((\Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2)\sigma; (\pi_1 \wedge \pi_2)\sigma)$. Then, δ is a solution of $(\pi_1 \wedge \pi_2)\sigma$ and $\sigma\delta$ is a solution of π_1 and π_2. Hence, $(\Gamma_1 \rightarrow \Delta_1, A)\sigma\delta$ and $(\Gamma_2, B \rightarrow \Delta_2)\sigma\delta$ are ground instances of $(\Gamma_1 \rightarrow \Delta_1, A; \pi_1)$ and $(\Gamma_2, B \rightarrow \Delta_2; \pi_2)$, respectively. Because $A\sigma\delta = B\sigma\delta$, if $(\Gamma_1 \rightarrow \Delta_1, A)\sigma\delta$ and $(\Gamma_2, B \rightarrow \Delta_2)\sigma\delta$ are satisfied, then $(\Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2)\sigma\delta$ is also satisfied. Therefore, Ordered SDC-Resolution with Selection is sound.

Let the clause $(\Gamma \rightarrow \Delta, A)\sigma\delta$ be a ground instance of $((\Gamma \rightarrow \Delta, A)\sigma; \pi\sigma)$. Then, δ is a solution of $\pi\sigma$ and $\sigma\delta$ is a solution of π. Hence, $(\Gamma \rightarrow \Delta, A, B)\sigma\delta$ is a ground instance of $(\Gamma \rightarrow \Delta, A, B; \pi)$. Because $A\sigma\delta = B\sigma\delta$, if $(\Gamma \rightarrow \Delta, A, B)\sigma\delta$ is satisfied, then $(\Gamma \rightarrow \Delta, A)\sigma\delta$ is also satisfied. Therefore, Ordered SDC-Factoring with Selection is sound. $\square$

Definition 6.2.6 (Saturation)**.** *A constrained clause set N is called saturated up to redundancy, if for every inference between clauses in N the result $(R; \pi)$ is either redundant in N or $\mathcal{G}((R; \pi)) \subseteq \mathcal{G}(N)$.*

Note that the redundancy notion includes condensation and that the condition $\mathcal{G}((R; \pi)) \subseteq \mathcal{G}(N)$ allows ignoring variants of clauses.

Lemma 6.2.7 (Ordered SDC Resolution Completeness)**.** *Let N be a constrained clause set saturated up to redundancy by ordered SDC-resolution with selection. Then N is unsatisfiable, if and only if $\square \in \mathcal{G}(N)$. If $\square \notin \mathcal{G}(N)$ then $\mathcal{I}_N \models N$.*

Proof. Assume N is unsatisfiable but $\square \notin \mathcal{G}(N)$. For the partial model $\mathcal{I}_N$, there exists a minimal false clause $C\sigma \in \mathcal{G}((C; \pi))$ for some $(C; \pi) \in N$.

$C\sigma$ is not productive, because otherwise $\mathcal{I}_N \models C\sigma$. Hence, either $\operatorname{sel}(C) \neq \emptyset$ or no positive literal in $C\sigma$ is strictly maximal. Assume $C = \Gamma_2, B \rightarrow \Delta_2$ with $B \in \operatorname{sel}(C)$ or $\neg B\sigma$ maximal. Then, $B\sigma \in \mathcal{I}_{C\sigma}$ and there exists a ground instance $(\Gamma_1 \rightarrow \Delta_1, A)\tau = D\tau \prec C\sigma$ of some clause $(D; \pi') \in N$, which produces $A\tau = B\sigma$. Therefore, there exists a $\rho = \operatorname{mgu}(A, B)$ and ground substitution δ such that $C\sigma = C\rho\delta$, $D\tau = D\rho\delta$. Since $\rho\delta = \sigma$ is a solution of π and π', δ is a solution of $(\pi \wedge \pi')\rho$. Under these conditions, SDC-Resolution can be applied to $(\Gamma_1 \rightarrow \Delta_1, A; \pi')$ and $(\Gamma_2, B \rightarrow \Delta_2; \pi)$. Their resolvent $(R; \pi_R) = ((\Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2)\rho; (\pi \wedge \pi')\rho)$ is either redundant in N or $\mathcal{G}((R; \pi_R)) \subseteq \mathcal{G}(N)$. Its ground instance $R\delta$ is false in $\mathcal{I}_N$ and $R\delta \prec C\sigma$. If $(R; \pi_R)$ is redundant in N, there exist $C_1, \ldots, C_n$ in $\mathcal{G}(N)^{\prec R\delta}$ with $C_1, \ldots, C_n \models R\delta$. Because $C_i \prec R\delta \prec C\sigma$, $\mathcal{I}_N \models C_i$ and hence $\mathcal{I}_N \models R\delta$, which contradicts $\mathcal{I}_N \not\models R\delta$. Otherwise, if $\mathcal{G}((R; \pi_R)) \subseteq \mathcal{G}(N)$, then $R\delta \in \mathcal{G}(N)$, which contradicts $C\sigma$ being minimal false.

Now, assume $\operatorname{sel}(C) = \emptyset$ and $C = \Gamma \rightarrow \Delta, B$ with $B\sigma$ maximal. Then, $C = \Gamma \rightarrow \Delta', A, B$ with $A\sigma = B\sigma$. Therefore, there exists a $\rho = \operatorname{mgu}(A, B)$ and ground substitution δ such that $C\sigma = C\rho\delta$ and $\rho\delta$ is a solution of π. Hence, δ is a solution of $\pi\rho$. Under these conditions, SDC-Factoring can be applied to

$(\Gamma \rightarrow \Delta', A, B; \pi)$. The result $(R; \pi_R) = ((\Gamma \rightarrow \Delta', A)\rho; \pi\rho)$ is either redundant in N or $\mathcal{G}((R; \pi_R)) \subseteq \mathcal{G}(N)$. Its ground instance $R\delta$ is false in $\mathcal{I}_N$ and $R\delta \prec C\sigma$. If $(R; \pi_R)$ is redundant in N, there exist $C_1, \ldots, C_n$ in $\mathcal{G}(N)^{\prec R\delta}$ with $C_1, \ldots, C_n \models R\delta$. Because $C_i \prec R\delta \prec C\sigma$, $\mathcal{I}_N \models C_i$ and hence $\mathcal{I}_N \models R\delta$, which contradicts $\mathcal{I}_N \not\models R\delta$. Otherwise, if $\mathcal{G}((R; \pi_R)) \subseteq \mathcal{G}(N)$, then $R\delta \in \mathcal{G}(N)$, which contradicts $C\sigma$ being minimal false.

Therefore, if $\square \notin \mathcal{G}(N)$, no minimal false clause exists and $\mathcal{I}_N \models N$. $\square$

Lemma 6.2.8. *Let N be a set of MSL(SDC) clauses without variants or condensations and Σ a finite signature. N is finite if there exists an integer d such that for every $(C; \pi) \in N$, depth(π)$\leq d$ and*
(1) $C = S_1(x_1), \ldots, S_n(x_n), S'_1(t), \ldots, S'_m(t) \rightarrow \Delta$ or
(2) $C = S_1(x_1), \ldots, S_n(x_n), S'_1(t), \ldots, S'_m(t) \rightarrow S(t), \Delta$
with t shallow and linear, and $\text{vars}(t) \cap \text{vars}(\Delta) = \emptyset$.

Proof. Let $(C; \pi) \in N$. $(C; \pi)$ can be separated into variable disjoint components $(\Gamma_1, \ldots, \Gamma_n \rightarrow \Delta_1, \ldots, \Delta_n; \pi_1 \wedge \ldots \wedge \pi_n)$, where $|\Delta_i| \leq 1$ and $\text{lvar}(\pi_i) \subseteq \text{vars}(\Gamma_i \rightarrow \Delta_i)$. For each positive literal $P(s) \in \Delta$ there is a fragment

$$(A) \ \ (S_1(x_1), \ldots, S_k(x_k) \rightarrow P(s); \pi')$$

with $\{x_1, \ldots, x_k\} \subseteq \text{vars}(s)$. If $m > 0$, there is another fragment

$$(B) \ \ (S_1(x_1), \ldots, S_k(x_k), S'_1(t), \ldots, S'_m(t) \rightarrow; \pi')$$

or

$$(C) \ \ (S_1(x_1), \ldots, S_k(x_k), S'_1(t), \ldots, S'_m(t) \rightarrow S(t); \pi')$$

with $\{x_1, \ldots, x_k\} \subseteq \text{vars}(t)$, respectively. Lastly, for each variable $x \in \text{vars}(C)$ with $x \notin \text{vars}(t) \cup \text{vars}(\Delta)$ there is a fragment

$$(D) \ \ (S_1(x), \ldots, S_k(x) \rightarrow; \pi').$$

Since there are only finitely many terms s with depth(s)$\leq d$ modulo renaming, there are only finitely many atomic constraints $x \neq s$ for a given variable x different up to renaming s. Thus, a normal constraint can only contain finitely many combinations of sub-constraints $\bigwedge_{i \in I} \ x \neq s_i$ without some s_i being an instance of another s_j. Therefore, for a fixed set of variables $x_1, \ldots, x_k$, there are only finitely many constraints $\pi = \bigwedge_{i \in I} \ z_i \neq s_i$ with $\text{lvar}(\pi) \subseteq \{x_1, \ldots, x_k\}$ up to variants.

Since the number of predicates, function symbols, and their ranks is finite, the number of possible shallow and linear atoms $S(t)$ different up to variants is finite. For a given shallow and linear t, there exist only finitely many clauses of the form $(S_1(t), \ldots, S_n(t) \rightarrow S(t); \pi)$ or $(S_1(t), \ldots, S_n(t) \ \rightarrow; \pi)$ with $\text{lvar}(\pi) \subseteq \text{vars}(t)$ modulo condensation and variants. For a fixed set of variables $x_1, \ldots, x_k$, there exist only finitely many clauses of the form $(S_1(y_1), \ldots, S_k(y_l) \rightarrow; \pi)$ modulo condensation and variants where $\mathrm{lvar}(\pi) \subseteq \{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\}$. Therefore, there are only finitely many distinct clauses of each form (A)-(D) without variants or condensations.

If in the clause $(C; \pi) = (\Gamma_1, \ldots, \Gamma_n \rightarrow \Delta_1, \ldots, \Delta_n; \pi_1 \wedge \ldots \wedge \pi_n)$ for some $i \neq j$, $(\Gamma_i \rightarrow \Delta_i; \pi_i)$ is a variant of $(\Gamma_j \rightarrow \Delta_j; \pi_j)$, then $(C; \pi)$ has a condensation and is therefore not part of N. Hence, there can be only finitely many different $(C; \pi)$ without variants or condensations and thus N is finite. □

Lemma 6.2.9 (Finite Saturation). *Let N be an MSL(SDC) clause set. Then N can be finitely saturated up to redundancy by sdc-resolution with selection function* sel.

Proof. The general idea is that given the way sel is defined the clauses involved in constrained resolution and factoring can only fall into certain patterns. Any result of such inferences then is either strictly smaller than one of its parents by some terminating measure or falls into a set of clauses that is bounded by Lemma 6.2.8. Thus, there can be only finitely many inferences before N is saturated.

Let d be an upper bound on the depth of constraints found in N and Σ be the finite signature consisting of the function and predicate symbols occurring in N. Let $(\Gamma_1 \rightarrow \Delta_1, S(t); \pi_1)$ and $(\Gamma_2, S(t') \rightarrow \Delta_2; \pi_2)$ be clauses in N where sdc-resolution applies with the most general unifier σ of $S(t)$ and $S(t')$ and resolvent $R = ((\Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2)\sigma; (\pi_1 \wedge \pi_2)\sigma\!\downarrow)$.

Because no literal is selected by sel, $\Gamma_1 \rightarrow \Delta_1, S(t)$ can match only one of two patterns:

$$(A)\;\; S_1(x_1), \ldots, S_n(x_n) \rightarrow S(f(y_1, \ldots, y_k)), \Delta$$

where $t = f(y_1, \ldots, y_k)$ and $\{x_1, \ldots, x_n\} \subseteq \{y_1, \ldots, y_k\} \cup \mathrm{vars}(\Delta)$.

$$(B)\;\; S_1(x_1), \ldots, S_n(x_n) \rightarrow S(y), \Delta$$

where $t = y$ and $x_1, \ldots, x_n$ are variables in $\mathrm{vars}(\Delta)$, i.e., y occurs only once.

The literal $S(t')$ is selected by sel in $\Gamma_2, S(t') \rightarrow \Delta_2$, and therefore $\Gamma_2, S(t') \rightarrow \Delta_2$ can match only one of the following three patterns:

(1) $S(f(t_1, \ldots, t_k)), \Gamma' \rightarrow \Delta'$

(2) $S(y'), \Gamma' \rightarrow \Delta'$ where Γ' has no function terms and $y \notin \mathrm{vars}(\Delta')$.

(3) $S(y'), \Gamma' \rightarrow S'(y'), \Delta'$ where Γ' has no function terms.

This means that the clausal part $(\Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2)\sigma$ of R has one of six forms:

$(A1)$ $S_1(x_1)\sigma, \dots, S_n(x_n)\sigma, \Gamma' \rightarrow \Delta, \Delta'$ with $\sigma = \{y_1 \mapsto t_1, \dots\}$.

$\Delta\sigma = \Delta$ because $S(f(y_1, \dots, y_k))$ and Δ do not share variables.

$(B1)$ $S_1(x_1), \dots, S_n(x_n), \Gamma' \rightarrow \Delta, \Delta'$.

The substitution $\{y \mapsto f(t_1, \dots, t_k)\}$ is irrelevant since $S(y)$ is the only literal with variable y.

$(A2)$ $S_1(x_1), \dots, S_n(x_n), \Gamma'\tau \rightarrow \Delta, \Delta'$ with $\tau = \{y' \mapsto f(y_1, \dots, y_k)\}$.

$\Delta'\tau = \Delta'$ because $y' \notin \text{vars}(\Delta')$.

$(B2)$ $S_1(x_1), \dots, S_n(x_n), \Gamma' \rightarrow \Delta, \Delta'$.
$(A3)$ $S_1(x_1), \dots, S_n(x_n), \Gamma'\tau \rightarrow S'(f(y_1, \dots, y_k)), \Delta, \Delta'$ with $\tau = \{y \mapsto f(y_1, \dots, y_k)\}$.

$\Delta'\tau = \Delta'$ because $y' \notin \text{vars}(\Delta')$.

$(B3)$ $S_1(x_1), \dots, S_n(x_n), \Gamma' \rightarrow S'(y'), \Delta, \Delta'$.

In the constraint $(\pi_1 \wedge \pi_2)\sigma\!\downarrow$ the maximal depth of the sub-constraints is less or equal to the maximal depth of π_1 or π_2. Hence, d is also an upper bound on the constraint of the resolvent. In each case, the resolvent is again an MSL(SDC) clause.

In the first and second case, the multiset of term depths of the negative literals in R is strictly smaller than for the right parent. In both, the Γ is the same between the right parent and the resolvent. Only the $f(t_1, \dots, t_k)$ term is replaced by $x_1\sigma, \dots, x_n\sigma$ and $x_1, \dots, x_n$ respectively. In the first case, the depth of the $x_i\sigma$ is either zero if $x_i \notin \{y_1, \dots, y_k\}$ or at least one less than $f(t_1, \dots, t_k)$ since $x_i\sigma = t_i$. In the second case, the x_i have depth zero which is strictly smaller than the depth of $f(t_1, \dots, t_k)$. Since the multiset ordering on natural numbers is terminating, the first and second case can only be applied finitely many times by constrained resolution.

In the third to sixth cases R is either $(S_1(x_1), \dots, S_l(x_l), S'_1(t), \dots, S'_m(t) \rightarrow \Delta; \pi)$ or $(S_1(x_1), \dots, S_l(x_l), S'_1(t), \dots, S'_m(t) \rightarrow S(t)), \Delta; \pi)$ with $t = f(y_1, \dots, y_k)$. By Lemma 6.2.8, there are only finitely many such clauses after condensation and removal of variants. Therefore, these four cases can apply only finitely many times during saturation.

Let $(\Gamma \rightarrow \Delta, S(t), S(t'); \pi)$ be a clause in N where sdc-factoring applies with $\sigma = \text{mgu}(S(t), S(t'))$ and $R = ((\Gamma \rightarrow \Delta, S(t))\sigma; \pi\sigma\!\downarrow)$. Because in $\Gamma \rightarrow \Delta, S(t), S(t')$ no literal is selected, $\Gamma \rightarrow \Delta, S(t), S(t')$ and $(\Gamma \rightarrow \Delta, S(t))\sigma$ can only match one of three patterns.

$(A)\ S_1(x_1), \dots, S_n(x_n) \rightarrow S(f(y_1, \dots, y_k)), S(f(z_1, \dots, z_k)), \Delta$

where $t = f(y_1, \dots, y_k)$, $t' = f(z_1, \dots, z_k)$, and
$\{x_1, \dots, x_n\} \subseteq \{y_1, \dots, y_k, z_1, \dots, z_k\} \cup \text{vars}(\Delta)$. The result is

$$S_1(x_1)\sigma, \dots, S_n(x_n)\sigma \rightarrow S(f(y_1, \dots, y_k)), \Delta \text{ with } \sigma = \{z_1 \mapsto y_1, \dots\}.$$

$(B)\ S_1(x_1), \dots, S_n(x_n) \rightarrow S(f(y_1, \dots, y_k)), S(z), \Delta$

where $t = f(y_1, \dots, y_k)$, $t' = z$ and $\{x_1, \dots, x_n\} \subseteq \{y_1, \dots, y_k\} \cup \text{vars}(\Delta)$,
i.e., z occurs only once. The result is

$$S_1(x_1), \dots, S_n(x_n) \rightarrow S(f(y_1, \dots, y_k)), \Delta.$$

$(C)\ S_1(x_1), \dots, S_n(x_n) \rightarrow S(y), S(z), \Delta$

where $t = y$, $t' = z$ and $\{x_1, \dots, x_n\} \subseteq \text{vars}(\Delta)$, i.e., y and z occur only once. The result is

$$S_1(x_1), \dots, S_n(x_n) \rightarrow S(y), \Delta.$$

In the new constraint $\pi\sigma\!\downarrow$ the maximal depth of the sub-constraints is less or equal to the maximal depth of π. Hence d is also an upper bound on the constraint of the resolvent. In each case, the resolvent is again an MSL(SDC) clause.

Furthermore, in each case the clause is of the form $(S_1(x_1), \dots, S_l(x_l) \rightarrow \Delta; \pi)$. By Lemma 6.2.8, there are only finitely many such clauses after condensation and removal of variants. Therefore, these three cases can apply only finitely many times during saturation. □

Theorem 6.2.10 (MSL(SDC) Decidability)**.** *Satisfiability of the MSL(SDC) first-order fragment is decidable.*

Proof. Follows from Lemma 6.2.7 and 6.2.9. □

6.3 Approximation $\Rightarrow_{\text{APC}}$

The approximation consists of the same transformation rules as in Chapter 5 except further modified to handle constrained clauses. In most cases constraints are simply copied from the approximated clause. For the Linear transformations the fresh variable receives copies of the original variable's constraints. Further, since the refinement is now more rigid compared to before, it is now treated as an additional transformation rule.

Monadic $\quad N \Rightarrow_{\text{MO}} \mu_P^T(N)$

provided P is a non-monadic predicate in the signature of N.

Shallow $\quad N \mathbin{\dot\cup} \{(\Gamma \rightarrow E[s]_p, \Delta; \pi)\} \Rightarrow_{\text{SH}}$
$N \cup \{(S(x), \Gamma_l \rightarrow E[p/x], \Delta_l; \pi); (\Gamma_r \rightarrow S(s), \Delta_r; \pi)\}$

provided s is complex, $|p| = 2$, x and S fresh,
$\Gamma_l\{x \mapsto s\} \cup \Gamma_r = \Gamma$, $\Delta_l \cup \Delta_r = \Delta$,
$\{Q(y) \in \Gamma \mid y \in \text{vars}(E[p/x], \Delta_l)\} \subseteq \Gamma_l$,
$\{Q(y) \in \Gamma \mid y \in \text{vars}(s, \Delta_r)\} \subseteq \Gamma_r$.

Linear 1 $\quad N \mathbin{\dot\cup} \{(\Gamma \rightarrow \Delta, E'[x]_p, E[x]_q; \pi)\} \Rightarrow_{\text{LI}}$
$N \cup \{(\Gamma\sigma, \Gamma \rightarrow \Delta, E'[x]_p, E[q/x']; \pi \wedge \pi\sigma)\}$

provided x' is fresh and $\sigma = \{x \mapsto x'\}$.

Linear 2 $\quad N \mathbin{\dot\cup} \{(\Gamma \rightarrow \Delta, E[x]_{p,q}; \pi)\} \Rightarrow_{\text{LI}}$
$N \cup \{(\Gamma\sigma, \Gamma \rightarrow \Delta, E[q/x']; \pi \wedge \pi\sigma)\}$

provided x' is fresh, $p \neq q$ and $\sigma = \{x \mapsto x'\}$.

Refinement $\quad N \mathbin{\dot\cup} \{(C, \pi)\} \Rightarrow_{\text{Ref}} N \cup \{(C; \pi \wedge x \neq t), (C; \pi)\{x \mapsto t\}\}$

provided $x \in \text{vars}(C)$, t straight and $\text{vars}(t) \cap \text{vars}((C, \pi)) = \emptyset$.

To reach the MSL(SDC) fragment the refinement transformation is strictly optional. The satisfiability equivalenttransformation $N \Rightarrow_{\text{Ref}} N'$ is used to achieve a more fine-grained over-approximation of N.

Note that the constraints of approximation clauses are implicitly simplified causing redundant constraints to be deleted.

Definition 6.3.1 ($\Rightarrow_{\text{APC}}$). *Define $\Rightarrow_{\text{APC}}$ as the priority rewrite system [3] consisting of $\Rightarrow_{\text{Ref}}$, $\Rightarrow_{\text{MO}}$, $\Rightarrow_{\text{SH}}$ and $\Rightarrow_{\text{LI}}$ with priority $\Rightarrow_{\text{Ref}} > \Rightarrow_{\text{MO}} > \Rightarrow_{\text{SH}} > \Rightarrow_{\text{LI}}$, where $\Rightarrow_{\text{Ref}}$ is only applied finitely many times.*

Lemma 6.3.2 ($\Rightarrow_{\text{APC}}$ is a Terminating Over-Approximation). *The approximation rules are terminating over-approximations: (i) $\Rightarrow_{\text{APC}}$ terminates, (ii) the Linear transformation is an over-approximation, (iii) the Shallow transformation is an over-approximation, (iv) the Monadic transformation is an over-approximation, (v) the refinement transformation is an over-approximation,*

Proof. (i) The transformations can be considered sequentially, because of the imposed rule priority. There are, by definition, only finitely many refinements at the beginning of an approximation $\Rightarrow^*_{\text{APC}}$. The Monadic transformation strictly reduces the number of non-monadic atoms. The Shallow transformation strictly reduces the multiset of term depths of the newly introduced clauses compared to the removed parent clause. The Linear transformation strictly reduces the number of duplicate variable occurrences in positive literals. Hence $\Rightarrow_{\text{APC}}$ terminates.

(ii) Let $N \cup \{(C;\pi)\} \Rightarrow_{\mathrm{LI}} N \cup \{(C_a;\pi_a)\}$ where an occurrence of a variable x in $(C;\pi)$ is replaced by a fresh x'. As $(C_a;\pi_a)\{x' \mapsto x\}$ is equal to $(C;\pi)$ modulo duplicate literal elimination, $\mathcal{I} \models (C;\pi)$ if $\mathcal{I} \models (C_a;\pi_a)$. Therefore, the Linear transformation is an over-approximation.

(iii) Let $N \cup \{(C;\pi)\} \Rightarrow_{\mathrm{SH}} N \cup \{(C_l;\pi_l),(C_r;\pi_r)\}$ and $(C_a;\pi_a)$ be the shallow ρ-resolvent. As $(C_a;\pi_a)\rho^{-1}$ equals $(C;\pi)$ modulo duplicate literal elimination, $\mathcal{I} \models (C;\pi)$ if $\mathcal{I} \models (C_l;\pi_l),(C_r;\pi_r)$. Therefore, the Shallow transformation is an over-approximation.

(iv) Let $N \Rightarrow_{\mathrm{MO}} \mu_P(N) = N'$. Then, $N = \mu_P^{-1}(N')$. Let $\mathcal{I}$ be a model of N' and $(C;\pi) \in N$. Since $\mu_P((C;\pi)) \in N'$, $\mathcal{I} \models \mu_P((C;\pi))$ and thus, $\mu_P^{-1}(\mathcal{I}) \models (C;\pi)$. Hence, $\mu_P^{-1}(\mathcal{I})$ is a model of N. Therefore, the Monadic transformation is an over-approximation. Actually, it is a satisfiability equivalent transformation.

(v) Let $N \cup \{(C;\pi)\} \Rightarrow_{\mathrm{Ref}} N \cup \{(C;\pi \wedge x \neq t),(C;\pi)\{x \mapsto t\}\}$. Let $C\delta \in \mathcal{G}((C;\pi))$. If $x\delta$ is not an instance of t, then δ is a solution of $\pi \wedge x \neq t$ and $C\delta \in \mathcal{G}((C;\pi \wedge x \neq t))$. Otherwise, $\delta = \{x \mapsto t\}\delta'$ for some substitution δ'. Then, δ is a solution of $\pi\{x \mapsto t\}$ and thus, $C\delta = C\{x \mapsto t\}\delta' \in \mathcal{G}((C\{x \mapsto t\};\pi\{x \mapsto t\}))$. Hence, $\mathcal{G}((C;\pi)) \subseteq \mathcal{G}((C;\pi \wedge x \neq t)) \cup \mathcal{G}((C;\pi)\{x \mapsto t\})$. Therefore, if $\mathcal{I}$ is a model of $N \cup \{(C;\pi \wedge x \neq t),(C;\pi)\{x \mapsto t\}\}$, then $\mathcal{I}$ is also a model of $N \cup \{(C;\pi)\}$. □

As before, the transformation also preserves, besides unsatisfiability, the structure of terms in a model. However, since the addition of constraints increased the complexity of this property, I explicitly define an ancestor relation $\Rightarrow_{\mathrm{A}}$ on the basis of $\Rightarrow_{\mathrm{APC}}$ that relates clauses, literal occurrences and variables with respect to the approximation. This relation is also used to determine the exact clause, literal, and variable used in the refinement.

Definition 6.3.3 (The Shallow Resolvent). *Let* $N \cup \{(C;\pi)\} \Rightarrow_{\mathrm{SH}} N \cup \{(C_l;\pi),(C_r;\pi)\}$ *with* $C = \Gamma \rightarrow E[s]_p,\Delta$, $C_l = S(x),\Gamma_l \rightarrow E[p/x],\Delta_l$ *and* $C_r = \Gamma_r \rightarrow S(s),\Delta_r$. *Let* $x_1,\dots,x_n$ *be the shared variables of* C_l *and* C_r *and* $\rho = \{x_1 \mapsto x'_1,\dots,x_n \mapsto x'_n\}$ *be a variable renaming with* $x'_1,\dots,x'_n$ *fresh in* C_l *and* C_r. *Define the* shallow ρ-resolvent *as* $(\Gamma_l\{x \mapsto s\rho\},\Gamma_r\rho \rightarrow E[p/s\rho],\Delta_l,\Delta_r\rho;\pi \wedge \pi\rho)$.

Let $(C_a;\pi_a)$ be the shallow ρ-resolvent of $N \cup \{(C;\pi)\} \Rightarrow_{\mathrm{SH}} N \cup \{(C_l;\pi),(C_r;\pi)\}$. Note that for any two ground instances $C_l\delta_l$ and $C_r\delta_r$, their resolvent is a ground instance of $(C_a;\pi_a)$. Further, using the substitution $\rho^{-1} = \{x'_1 \mapsto x_1,\dots,x'_n \mapsto x_n\}$, $(C_a;\pi_a)\rho^{-1} = (\Gamma_l\{x \mapsto s\},\Gamma_r \rightarrow E[s]_p,\Delta_l,\Delta_r;\pi \wedge \pi)$ is equal to $(C;\pi)$ modulo duplicate literal elimination. This is because, $\Delta_l \cup \Delta_r = \Delta$ and $\Gamma_l\{x \mapsto s\} \cup \Gamma_r = \Gamma$ by definition of $\Rightarrow_{\mathrm{SH}}$ and $\pi \wedge \pi$ is equivalent to π.

Next, I establish parent relations that link approximation and approximated clauses, as well as their variables and literals. Together the parent, variable and literal relations allow me to not only trace any approximation clause back to their origin, but also predict what consequences changes to the original set will have on its approximations.

For the following definitions, I assume that clause and literal sets are lists and that μ_P^T and substitutions act as mappings. This means I can uniquely identify clauses and literals by their position in those lists. Further, for every Shallow transformation $N \Rightarrow_{\text{SH}} N'$, I will also include the shallow resolvent in the parent relation as if it were a member of N'.

Definition 6.3.4 (Parent Clause). *For an approximation step $N \Rightarrow_{\text{AP}} N'$ and two clauses $(C;\pi) \in N$ and $(C';\pi') \in N'$, define $[(C;\pi), N] \Rightarrow_{\text{A}} [(C';\pi'), N']$ expressing that $(C;\pi)$ in N is the parent clause of $(C';\pi')$ in N':*
If $N \Rightarrow_{\text{MO}} \mu_P^T(N)$, then
$[(C;\pi), N] \Rightarrow_{\text{A}} [\mu_P^T((C;\pi)), \mu_P^T(N)]$ *for all* $(C;\pi) \in N$.
If $N = N'' \cup \{(C;\pi)\} \Rightarrow_{\text{SH}} N'' \cup \{(C_l;\pi_l), (C_r;\pi_r)\} = N'$, then
$[(D,\pi'), N] \Rightarrow_{\text{A}} [(D,\pi'), N']$ *for all* $(D,\pi') \in N''$ *and*
$[(C,\pi), N] \Rightarrow_{\text{A}} [(C_l;\pi_l), N']$,
$[(C,\pi), N] \Rightarrow_{\text{A}} [(C_r;\pi_r), N']$ *and*
$[(C,\pi), N] \Rightarrow_{\text{A}} [(C_a;\pi_a), N']$ *for any shallow resolvent* $(C_a;\pi_a)$.
If $N = N'' \cup \{(C;\pi)\} \Rightarrow_{\text{LI}} N'' \cup \{(C_a;\pi_a)\} = N'$, then
$[(D,\pi'), N] \Rightarrow_{\text{A}} [(D,\pi'), N']$ *for all* $(D,\pi') \in N''$ *and*
$[(C,\pi), N] \Rightarrow_{\text{A}} [(C_a,\pi_a), N']$.
If $N = N'' \cup \{(C;\pi)\} \Rightarrow_{\text{Ref}} N'' \cup \{(C;\pi \wedge x \neq t), (C;\pi)\{x \mapsto t\}\} = N'$, then
$[(D,\pi'), N] \Rightarrow_{\text{A}} [(D,\pi'), N']$ *for all* $(D,\pi') \in N''$,
$[(C,\pi), N] \Rightarrow_{\text{A}} [(C;\pi \wedge x \neq t), N']$ *and*
$[(C,\pi), N] \Rightarrow_{\text{A}} [(C;\pi)\{x \mapsto t\}, N']$.

Definition 6.3.5 (Parent Variable). *Let $N \Rightarrow_{\text{AP}} N'$ be an approximation step and $[(C;\pi), N] \Rightarrow_{\text{A}} [(C';\pi'), N']$. For two variables x and y, define $[x, (C;\pi), N] \Rightarrow_{\text{A}} [y, (C';\pi'), N']$ expressing that $x \in \text{vars}(C)$ is the parent variable of $y \in \text{vars}(C')$:*
If $x \in \text{vars}((C;\pi)) \cap \text{vars}((C';\pi'))$, then
$[x, (C;\pi), N] \Rightarrow_{\text{A}} [x, (C';\pi'), N']$.
If $N \Rightarrow_{\text{SH}} N'$ and (C',π') is the shallow ρ-resolvent,
$[x_i, (C;\pi), N] \Rightarrow_{\text{A}} [x_i\rho, (C';\pi'), N']$ *for each x_i in the domain of ρ.*
If $N \Rightarrow_{\text{LI}} N'$, $C = \Gamma \rightarrow \Delta[x]_{p,q}$ and $C' = \Gamma\{x \mapsto x'\}, \Gamma \rightarrow \Delta[q/x']$, then
$[x, (C;\pi), N] \Rightarrow_{\text{A}} [x', (C';\pi'), N']$.

Note that if $N \Rightarrow_{\text{SH}} N'$ and x is the fresh extraction variable in $(C_l;\pi_l)$, then x has no parent variable. For literals, I actually further specify the relation on the positions within literals of a clause $(C;\pi)$ using pairs (L,r) of literals and positions. I write $(L,r) \in C$ to denote that (L,r) is a literal position in $(C;\pi)$ if $L \in C$ and $r \in \text{pos}(L)$. Note that a literal position (L,r) in $(C;\pi)$ corresponds to the term $L|_r$.

Definition 6.3.6 (Parent literal position). *Let $N \Rightarrow_{\text{AP}} N'$ be an approximation step and $[(C;\pi), N] \Rightarrow_{\text{A}} [(C';\pi'), N']$. For two literal positions (L,r) and (L',r'), define $[r, L, (C;\pi), N] \Rightarrow_{\text{A}} [r', L', (C';\pi'), N']$ expressing that (L,r) in $(C;\pi)$ is the parent literal position of (L',r') in $(C';\pi')$:*
If $(C;\pi) = (C';\pi')$, then
$[r, L, (C;\pi), N] \Rightarrow_{\text{A}} [r, L, (C';\pi'), N']$ *for all* $(L,r) \in C$.
If $N \Rightarrow_{\text{Ref}} N'$ and $(C',\pi') = (C;\pi \wedge x \neq t)$, then

$[r, L, (C;\pi), N] \Rightarrow_{\mathrm{A}} [r, L, (C';\pi'), N']$ *for all* $(L, r) \in C$.

If $N \Rightarrow_{\mathrm{Ref}} N'$ *and* $(C', \pi') = (C;\pi)\{x \mapsto t\}$, *then*

$[r, L, (C;\pi), N] \Rightarrow_{\mathrm{A}} [r, L\{x \mapsto t\}, (C';\pi'), N']$ *for all* $(L, r) \in C$.

If $N \Rightarrow_{\mathrm{MO}} \mu_P^T(N) = N'$, *then*

$[\varepsilon, P(\bar{t}), (C;\pi), N] \Rightarrow_{\mathrm{A}} [\varepsilon, T(f_p(\bar{t})), (C';\pi'), N']$ *for all* $P(\bar{t}) \in C$ *and*

$[r, P(\bar{t}), (C;\pi), N] \Rightarrow_{\mathrm{A}} [1.r, T(f_p(\bar{t})), (C';\pi'), N']$ *for all* $(P(\bar{t}), r) \in C$.

If $N \Rightarrow_{\mathrm{SH}} N'$, $C = \Gamma \rightarrow E[s]_p, \Delta$ *and* $C' = S(x), \Gamma_l \rightarrow E[p/x], \Delta_l$, *then*

$[r, E[s]_p, (C;\pi), N] \Rightarrow_{\mathrm{A}} [r, E[p/x], (C';\pi'), N']$ *for all* $r \in \mathrm{pos}(E[p/x])$,

$[p, E[s]_p, (C;\pi), N] \Rightarrow_{\mathrm{A}} [r, S(x), (C';\pi'), N']$ *for all* $r \in \mathrm{pos}(S(x))$,

$[r, L\{x \mapsto s\}, (C;\pi), N] \Rightarrow_{\mathrm{A}} [r, L, (C';\pi'), N']$ *for all* $(L, r) \in \Gamma_l$,

$[r, L, (C;\pi), N] \Rightarrow_{\mathrm{A}} [r, L, (C';\pi'), N']$ *for all* $(L, r) \in \Delta_l$.

If $N \Rightarrow_{\mathrm{SH}} N'$, $C = \Gamma \rightarrow E[s]_p, \Delta$ *and* $C' = \Gamma_r \rightarrow S(s), \Delta_r$, *then*

$[p, E[s]_p, (C;\pi), N] \Rightarrow_{\mathrm{A}} [\varepsilon, S(s), (C';\pi'), N']$,

$[pr, E[s]_p, (C;\pi), N] \Rightarrow_{\mathrm{A}} [1.r, S(s), (C';\pi'), N']$ *for all* $r \in \mathrm{pos}(s)$, *and*

$[r, L, (C;\pi), N] \Rightarrow_{\mathrm{A}} [r, L, (C';\pi'), N']$ *for all* $(L, r) \in \Gamma_r \cup \Delta_r$.

If $N \Rightarrow_{\mathrm{SH}} N'$, $C = \Gamma \rightarrow E[s]_p, \Delta$ *and* (C', π') *is the shallow* ρ*-resolvent, then*

$[r, E[s]_p, (C;\pi), N] \Rightarrow_{\mathrm{A}} [r, E[p/s\rho], (C';\pi'), N']$ *for all* $r \in \mathrm{pos}(E[p/s\rho])$,

$[r, L\{x \mapsto s\}, (C;\pi), N] \Rightarrow_{\mathrm{A}} [r, L\{x \mapsto s\rho\}, (C';\pi'), N']$ *for all* $(L, r) \in \Gamma_l$,

$[r, L, (C;\pi), N] \Rightarrow_{\mathrm{A}} [r, L\rho, (C';\pi'), N']$ *for all* $(L, r) \in \Gamma_r \cup \Delta_r$, *and*

$[r, L, (C;\pi), N] \Rightarrow_{\mathrm{A}} [r, L, (C';\pi'), N']$ *for all* $(L, r) \in \Delta_l$.

If $N \Rightarrow_{\mathrm{LI}} N'$, $C = \Gamma \rightarrow \Delta, E'[x]_p, E[x]_q$ *and* $C' = \Gamma\{x \mapsto x'\}, \Gamma \rightarrow \Delta, E'[x]_p, E[q/x']$,

$[r, E'[x]_p, (C;\pi), N] \Rightarrow_{\mathrm{A}} [r, E'[x]_p, (C';\pi'), N']$ *for all* $r \in \mathrm{pos}(E'[x]_p)$,

$[r, E[x]_q, (C;\pi), N] \Rightarrow_{\mathrm{A}} [r, E[q/x'], (C';\pi'), N']$ *for all* $r \in \mathrm{pos}(E[q/x'])$,,

$[r, L, (C;\pi), N] \Rightarrow_{\mathrm{A}} [r, L\{x \mapsto x'\}, (C';\pi'), N']$ *for all* $(L, r) \in \Gamma$,

$[r, L, (C;\pi), N] \Rightarrow_{\mathrm{A}} [r, L, (C';\pi'), N']$ *for all* $(L, r) \in \Gamma$, *and*

$[r, L, (C;\pi), N] \Rightarrow_{\mathrm{A}} [r, L, (C';\pi'), N']$ *for all* $(L, r) \in \Delta$.

If $N \Rightarrow_{\mathrm{LI}} N'$, $C = \Gamma \rightarrow \Delta, E[x]_{p,q}$ *and* $C' = \Gamma\{x \mapsto x'\}, \Gamma \rightarrow \Delta, E[q/x']$, *then*

$[r, E[x]_{p,q}, (C;\pi), N] \Rightarrow_{\mathrm{A}} [r, E[q/x'], (C';\pi'), N']$ *for all* $r \in \mathrm{pos}(E[q/x'])$,

$[r, L, (C;\pi), N] \Rightarrow_{\mathrm{A}} [r, L\{x \mapsto x'\}, (C';\pi'), N']$ *for all* $(L, r) \in \Gamma$,

$[r, L, (C;\pi), N] \Rightarrow_{\mathrm{A}} [r, L, (C';\pi'), N']$ *for all* $(L, r) \in \Gamma$, *and*

$[r, L, (C;\pi), N] \Rightarrow_{\mathrm{A}} [r, L, (C';\pi'), N']$ *for all* $(L, r) \in \Delta$.

The transitive closures of each parent relation are called ancestor relations.

6.4 Lifting of $\Rightarrow_{\mathrm{APC}}$

As with the approximation, the definitions of the conflicting core (Definition 4.2.1) and generalized resolution refutation (Definition 4.2.2) also change with the addition of straight dismatching constraints.

Definition 6.4.1 (Conflicting Core). *A finite set of unconstrained clauses and a solvable constraint* $(N^\perp;\pi)$ *are a conflicting core if* $N^\perp\delta$ *is unsatisfiable for all solutions* δ *of* π *over* $\mathrm{vars}(N^\perp) \cup \mathrm{lvar}(\pi)$. *A conflicting core* $(N^\perp;\pi)$ *is a conflicting*

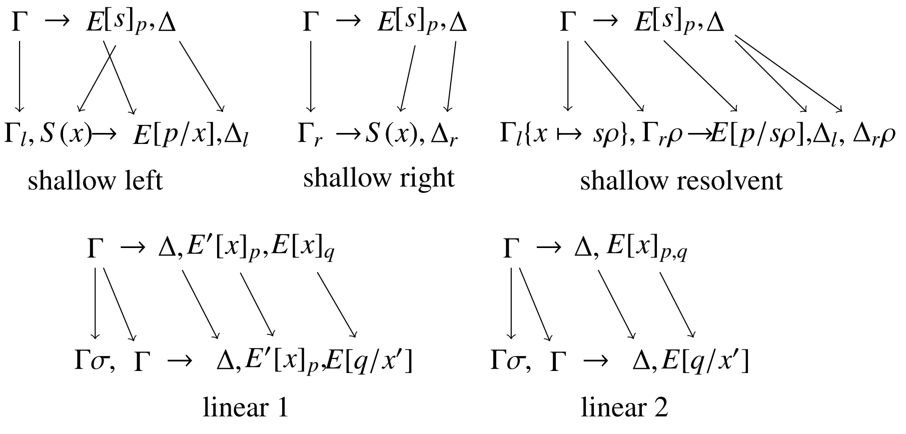

$\Gamma \rightarrow E[s]_p, \Delta$
$\Gamma_l, S(x) \rightarrow E[p/x], \Delta_l$
shallow left
$\Gamma \rightarrow E[s]_p, \Delta$
$\Gamma_r \rightarrow S(x), \Delta_r$
shallow right
$\Gamma \rightarrow E[s]_p, \Delta$
$\Gamma_l\{x \mapsto s\rho\}, \Gamma_r\rho \rightarrow E[p/s\rho], \Delta_l, \Delta_r\rho$
shallow resolvent
$\Gamma \rightarrow \Delta, E'[x]_p, E[x]_q$
$\Gamma\sigma, \Gamma \rightarrow \Delta, E'[x]_p, E[q/x']$
linear 1
$\Gamma \rightarrow \Delta, E[x]_{p,q}$
$\Gamma\sigma, \Gamma \rightarrow \Delta, E[q/x']$
linear 2

Figure 6.2: Visual representation of the parent literal position relation (Definition 6.3.6)

core of the constrained clause set N if for every $C \in N^\perp$ there is a clause $(C', \pi') \in N$ such that $(C; \pi)$ is an instance of $(C'; \pi')$ modulo duplicate literal elimination. $(C'; \pi')$ is then called the instantiated clause of $(C; \pi)$ in $(N^\perp; \pi)$. A conflicting core $N^\perp$ of N is minimal if for any strict subset $M \subsetneq N^\perp$, M is not a conflicting core of N.

Definition 6.4.2 (Generalized Resolution Refutation)**.** *A generalized resolution refutation $\mathcal{R}$ of a constrained clause set N is a list of variable disjoint constrained clauses, where the last clause is the empty clause $(\square; \top)$. Each clause in $\mathcal{R}$ is annotated by one of the following four labels:*
$(C, \pi)^N$, $(C; \pi)^{(C';\pi'),\sigma}$, $(C; \pi)^{(C_1;\pi_1),(C_2;\pi_2),\sigma}$, and $(C; \pi)^{(N^\perp;\pi_c)}$ which are called input, factor, resolution, and derived clauses, respectively. Additionally, C^* is used as a placeholder for an arbitrary label. Furthermore, a generalized resolution refutation $\mathcal{R}$ satisfies the following four properties, where (R1)-(R3) correspond to the default properties of a resolution refutation while (R4) is an additional invariant on generalized resolution refutations

(R1) *If $\mathcal{R} = \mathcal{R}_1, (C, \pi)^N, \mathcal{R}_2$, then (C, π) is a variant of a clause $(C', \pi') \in N$.*

(R2) *If $\mathcal{R} = \mathcal{R}_1, (C; \pi)^{(C';\pi'),\sigma}, \mathcal{R}_2$, then $(C'; \pi')^* \in \mathcal{R}_1$, and (C, π) is a factor of $(C'; \pi')$ using unifier σ.*

(R3) *If $\mathcal{R} = \mathcal{R}_1, (C; \pi)^{(C_1;\pi_1),(C_2;\pi_2),\sigma}, \mathcal{R}_2$, then $(C_1; \pi_1)^* \in \mathcal{R}_1$, $(C_2; \pi_2)^* \in \mathcal{R}_1$, and (C, π) is a resolvent of $(C_1; \pi_1)$ and $(C_2; \pi_2)$ using unifier σ.*

(R4) *If $\mathcal{R} = \mathcal{R}_1, (C; \pi)^{(N^\perp;\pi_c)}, \mathcal{R}_2$, then (i) for all substitutions τ, $\pi_c\tau$ is solvable if $\pi\tau$ is solvable, (ii) for all $D \in N^\perp$, $(D; \pi_c)$ is an instance of a clause in N, and (iii) for all solutions δ of π_c over* $\mathrm{lvar}(\pi_c) \cup \mathrm{vars}(N) \cup \mathrm{lvar}(\pi) \cup \mathrm{vars}(C)$*, δ is a solution of π and $N^\perp\delta \models C\delta$.*

Input $\mathcal{R}_1, (C;\pi)^N, \mathcal{R}_2 \Rightarrow_{\mathcal{R}} \mathcal{R}_1, (C;\pi)^{(\{C\};\pi)}, \mathcal{R}_2$

Factoring $\mathcal{R}_1, (C;\pi)^{(C';\pi'),\sigma}, \mathcal{R}_2 \Rightarrow_{\mathcal{R}} \mathcal{R}_1, (C;\pi)^{(N^\perp;\pi_c)\sigma}, \mathcal{R}_2$
if $(C';\pi')^{(N^\perp;\pi_c)} \in \mathcal{R}_1$.

Resolution $\mathcal{R}_1, (C,\pi)^{(C_1;\pi_1),(C_2;\pi_2),\sigma}, \mathcal{R}_2 \Rightarrow_{\mathcal{R}} \mathcal{R}_1, (C;\pi)^{(N_1^\perp \cup N_2^\perp;\pi_{c1}\wedge\pi_{c2})\sigma}, \mathcal{R}_2$
if $(C_1;\pi_1)^{(N_1^\perp,\pi_{c1})} \in \mathcal{R}_1$ and $(C_2;\pi_2)^{(N_2^\perp,\pi_{c2})} \in \mathcal{R}_1$.

Lemma 6.4.3. *If $\mathcal{R}$ is a resolution refutation and $\mathcal{R} \Rightarrow_{\mathcal{R}} \mathcal{R}'$, then $\mathcal{R}'$ is also a resolution refutation.*

Proof. Let $\mathcal{R}$ be a resolution refutation for a constrained clause set N.

Let $\mathcal{R} = \mathcal{R}_1, (C;\pi)^N, \mathcal{R}_2 \Rightarrow_{\mathcal{R}} \mathcal{R}_1, (C;\pi)^{(\{C\};\pi)}, \mathcal{R}_2 = \mathcal{R}'$. We only need to show that $(C;\pi)^{(\{C\};\pi)}$ satisfies property R4. Trivially, (i) $\pi = \pi_c$, (ii) $(C;\pi)$ is a variant of a clause in N (R2) and (iii) $C\delta \models C\delta$ for any solution δ of π. Hence, $\mathcal{R}'$ is a resolution refutation for N.

Let $\mathcal{R} = \mathcal{R}_1, (C;\pi)^{(C_1;\pi_1),(C_2;\pi_2),\sigma}, \mathcal{R}_2 \Rightarrow_{\mathcal{R}} \mathcal{R}_1, (C;\pi)^{(N_1^\perp \cup N_2^\perp;\pi_{c1}\wedge\pi_{c2})\sigma}, \mathcal{R}_2 = \mathcal{R}'$ with $(C_1;\pi_1')^{(N_1^\perp,\pi_{c1})}$ and $(C_2;\pi_2')^{(N_2^\perp,\pi_{c2})}$ in $\mathcal{R}_1$. Showing that $(C;\pi)^{(N_1^\perp \cup N_2^\perp;\pi_{c1}\wedge\pi_{c2})\sigma}$ satisfies property R4 is sufficient. By property R4, for $i = 1, 2$ (i) for all substitutions τ, $\pi_{ci}\tau$ is solvable if $\pi_i\tau$ is solvable, (ii) for all $D \in N_i^\perp$ there exists a $(D';\pi') \in N$ such that $(D;\pi_{ci})$ is an instance of $(D';\pi')$ and (iii) for all solutions δ_i of π_i, δ_i is a solution of π_i' and $N_i^\perp\delta_i \models C_i\delta_i$. (i) Let $\pi\tau$ be solvable. Since $\pi = (\pi_1 \wedge \pi_2)\sigma$, $\pi_1\sigma\tau$ and $\pi_2\sigma\tau$ are solvable. Then, $\pi_{c1}\sigma\tau$ and $\pi_{c2}\sigma\tau$ are solvable as well and therefore, $(\pi_{c1} \wedge \pi_{c2})\sigma\tau$, too. (ii) W.l.o.g., let $D \in N_1^\perp\sigma$. Then, $D = D_1\sigma$ for some $D_1 \in N_1^\perp$ and thus, there is a $(D';\pi') \in N$ such that $(D_1;\pi_{c1})$ is an instance of $(D';\pi')$. Since $(D_1\sigma;\pi_{c1} \wedge \pi_{c2})$ is an instance of $(D_1;\pi_{c1})$, $(D;\pi_{c1} \wedge \pi_{c2})$ is also an instance of $(D';\pi')$. (iii) Let δ be a solution of $\pi_{c1}\sigma \wedge \pi_{c2}\sigma$. Then, $\sigma\delta$ is a solution of both π_{c1} and π_{c2} and thus, also of both π_1 and π_2. Therefore, δ is a solution of $(\pi_1 \wedge \pi_2)\sigma = \pi$. Then, $N_1^\perp\sigma\delta \models C_1\sigma\delta$ and $N_2^\perp\sigma\delta \models C_2\sigma\delta$. Therefore, $(N_1^\perp\sigma \cup N_2^\perp\sigma)\delta \models C$, because $C_1\sigma, C_2\sigma \models C$ by R3 and Lemma 6.2.5. Hence, $\mathcal{R}'$ is a resolution refutation for N.

Let $\mathcal{R} = \mathcal{R}_1, (C;\pi)^{(C';\pi'),\sigma}, \mathcal{R}_2 \Rightarrow_{\mathcal{R}} \mathcal{R}_1, (C;\pi)^{(N^\perp;\pi_c)\sigma}, \mathcal{R}_2 = \mathcal{R}'$. The proof works analogously to the previous case. □

For the last clause $(\square, \top)^{(N^\perp,\pi_c)}$, $(N^\perp, \pi_c)$ is then a conflicting core of N as a direct consequence of property R4.

Note that each clause in $N^\perp$ can be traced back to a unique input clause in the original $\mathcal{R}$. Each input clause is a variant of exactly one clause in N, otherwise N would have contained redundant variants of the same clause. In practice, I set this clause as the instantiated clause of the conflict clause in $(N^\perp, \pi_c)$.

As in Chapters 4 and 5, given an approximation $N \Rightarrow^*_{\text{AP}} N_k$ and a conflicting core $(N_k^\perp;\pi)$ of N_k, using the lifting lemmas provided in this section I attempt to lift $(N_k^\perp;\pi)$ step-wise to a conflicting core of N.

Lemma 6.4.4 (Linear Lifting). *Let $N \cup \{(C;\pi)\} \Rightarrow_{\mathrm{LI}} N \cup \{(C_a;\pi_a)\}$ and $(N^\perp;\pi_c)$ be a conflicting core of $N \cup \{(C_a;\pi_a)\}$. Let $C_1,\dots,C_m$ be all clauses in $N^\perp$ with instantiated clause $(C_a;\pi_a)$. If for each C_i, $(C_i;\pi_c)$ is an instance of $(C;\pi)$ modulo duplicate literal elimination, then $(N^\perp;\pi_c)$ is a conflicting core of $N \cup \{(C;\pi)\}$.*

Proof. Follows trivially from the definition of conflicting cores, i.e., $(N^\perp;\pi_c)$ is already a conflicting core and for any clause $C^\perp \in N^\perp$ that is not among $C_1,\dots,C_m$, there is a clause $(D,\pi') \in N$ such that $(C^\perp,\pi_c)$ is an instance of (D,π') modulo duplicate literal elimination. □

Lemma 6.4.5 (Shallow Lifting). *Let $N \cup \{(C;\pi)\} \Rightarrow_{\mathrm{SH}} N \cup \{(C_l;\pi_l),(C_r;\pi_r)\}$ with the fresh predicate S and $(N^\perp;\pi_c)$ be a conflicting core of $N \cup \{(C_l;\pi_l),(C_r;\pi_r)\}$. Let N_S be the set of all resolvents of clauses in $N^\perp$ on S-atoms. If for each clause $C_a \in N_S$, $(C_a;\pi_c)$ is an instance of $(C;\pi)$ modulo duplicate literal elimination, then $(\{D \in N^\perp \mid \text{no } S\text{-atom in } D\} \cup N_S;\pi_c)$ is a conflicting core of $N \cup \{(C;\pi)\}$.*

Proof. Let $N'^\perp = \{D \in N^\perp \mid \text{no } S\text{-atom in } D\} \cup N_S$, δ be a solution of π_c and $\mathcal{I}$ an interpretation. Then, there exists a $C^\perp \in N^\perp\delta$ such that $\mathcal{I} \not\models C^\perp$. If $C^\perp$ does not contain an S-atom, then $C^\perp \in N'^\perp\delta$. Thus, $\mathcal{I} \not\models N'^\perp\delta$. Otherwise, at least one clause with an S-atom is false under $\mathcal{I}$ in $N^\perp\delta$. By construction, any such clause has either C_l or C_r as their instantiated clause. Let $C_l\tau_1,\dots,C_l\tau_m$ and $C_r\rho_1,\dots,C_r\rho_n$ be all clauses in $N^\perp\delta$ that are false under $\mathcal{I}$. Let

$$\mathcal{I}' := \mathcal{I} \setminus \{S(x)\tau_1,\dots,S(x)\tau_m\} \cup \{S(s)\rho_1,\dots,S(s)\rho_n\},$$

i.e., change the truth value for S-atoms such that the clauses unsatisfied under $\mathcal{I}$ are satisfied under $\mathcal{I}'$. Because $\mathcal{I}$ and $\mathcal{I}'$ only differ on S-atoms, there exists a clause $D \in N^\perp\delta$ that is false under $\mathcal{I}'$ and contains an S-atom. Let $D = C_l\sigma$. Since $\mathcal{I} \models D$, $S(x)\sigma$ was added to $\mathcal{I}'$ by some clause $C_r\rho_j$, where $S(s)\rho_j = S(x)\sigma$. Let R be the resolvent of $C_r\rho_j$ and $C_1\sigma$ on $S(s)\rho_j$ and $S(x)\sigma$. Then, $\mathcal{I} \not\models R$ because $\mathcal{I} \not\models C_r\rho_j$ and $\mathcal{I} \cup \{S(s)\rho_j\} \not\models C_l\delta'$. Thus, $\mathcal{I} \not\models N'^\perp\delta$. For $D = C_r\sigma$, the proof is analogous. Therefore, $N'^\perp$ is a conflicting core of $N \cup \{(C;\pi)\}$. □

Lifting a Refinement Transformation In the case of a refinement transformation $N \cup \{(C;\pi)\} \Rightarrow_{\mathrm{Ref}} N \cup \{(C;\pi \wedge x \neq t),(C;\pi)\{x \mapsto t\}\}$, the clauses $(C;\pi \wedge x \neq t)$ and $(C;\pi)\{x \mapsto t\}$ are both instances of $(C;\pi)$. Therefore, any instance of either clause in a conflicting core $N^\perp$ of $N \cup \{(C;\pi \wedge x \neq t),(C;\pi)\{x \mapsto t\}\}$ is an instance of $(C;\pi)$. Hence, $N^\perp$ is also a conflicting core of $N \cup \{(C;\pi)\}$.

Lemma 6.4.6 (Refinement Lifting). *Let $N \Rightarrow_{\mathrm{Ref}} N'$. If $(N^\perp;\pi)$ is a conflicting core of N', then $(N^\perp;\pi)$ is a conflicting core of N.*

Lifting with Instantiation now requires that the instantiation does not contradict the conflicting core's constraint. By definition, if $(N^\perp;\pi)$ is a conflicting core of N, then $(N^\perp;\pi)\tau$ is also a conflicting core of N if $\pi\tau$ is solvable.

Static Completeness

Theorem 6.4.7 (Static Completeness)**.** *Let N_0 be an unsatisfiable clause set and $N_0 \Rightarrow^*_{\text{AP}} N_k$. Then, for every ground conflicting core $N^\perp$ of N_0 there exists a ground conflicting core $N_k^\perp$ of N_k such that $N_k^\perp$ can be lifted to $N^\perp$.*

Proof. By induction on $N_0 \Rightarrow^*_{\text{AP}} N_k$. If $N_0 = N_k$, any ground unsatisfiability core of N_0 is also a ground conflicting core of N_k. Let $N_0 \Rightarrow^*_{\text{AP}} N_{k-1} \Rightarrow_{\text{AP}} N_k$. By the inductive hypothesis, there is a ground conflicting core $N_{k-1}^\perp$ of N_{k-1} which can be lifted to N_0. The final approximation rule application is either a linear, a shallow, a monadic or a refinement transformation, considered below by case analysis.

Let $N_{k-1} = N' \cup \{(C;\pi)\} \Rightarrow_{\text{LI}} N_k = N' \cup \{(C_a;\pi_a)\}$ where a variable x is replaced by x'. Let $C\delta_1, \dots, C\delta_n$ be the instances of $(C;\pi)$ in $N_{k-1}^\perp$. Then, each $C\delta_i$ is equivalent to $C_a\{x' \mapsto x\}\delta_i$ and $\{x' \mapsto x\}\delta_i$ is a solution of $\pi_a = \pi \wedge \pi\{x' \mapsto x\}$ because δ_i is a solution of π. Thus, $N_{k-1}^\perp \setminus \{C\delta_i, \dots, C\delta_n\} \cup \{C_a\{x' \mapsto x\}\delta_i \mid 1 \leq i \leq n\}$ is a conflicting core of N_k. By Lemma 6.4.4, it can be lifted to $N_{k-1}^\perp$.

Let $N \Rightarrow^*_{\text{AP}} N_{k-1} = N' \cup \{(C;\pi)\} \Rightarrow_{\text{SH}} N_k = N' \cup \{(C_l;\pi_l), (C_r;\pi_r)\}$ where a term s is extracted from a positive literal $E[s]_p$ via introduction of fresh predicate S and variable x. Let $C\delta_1, \dots, C\delta_n$ be the instances of $(C;\pi)$ in $N_{k-1}^\perp$. Then, each $C\delta_i$ is equal modulo duplicate literal elimination to the resolvent C_i of $C_l\{x \mapsto s\}\delta_i$ and $C_r\delta_i$, and $\{x \mapsto s\}\delta_i$ and δ_i are both solutions of $\pi = \pi_l = \pi_r$. Thus, $N_k^\perp = N_{k-1}^\perp \setminus \{C\delta_i, \dots, C\delta_n\} \cup \{C_l\{x \mapsto s\}\delta_i, C_r\delta_i \mid 1 \leq i \leq n\}$ is a conflicting core of N_k. Since $\{C_1, \dots C_n\}$ is the set of resolvents of $\{C_l\{x \mapsto s\}\delta_i, C_r\delta_i \mid 1 \leq i \leq n\}$, $N_k^\perp$ can be lifted to $N_{k-1}^\perp$ by Lemma 6.4.5.

Let $N \Rightarrow^*_{\text{AP}} N_{k-1} \Rightarrow_{\text{MO}} N_k = \mu_P(N_{k-1})$ where P is a non-monadic predicate in N_{k-1}. $N_k^\perp = \mu_P(N_{k-1}^\perp)$ is a conflicting core of N_k. By Lemma 4.2.7, $N_k^\perp$ can be lifted to $N_{k-1}^\perp$.

Let $N \Rightarrow^*_{\text{AP}} N_{k-1} \Rightarrow_{\text{Ref}} N_k$ where $(C;\pi)$ is segmented into $(C;\pi \wedge x \neq t)$ and $(C;\pi)\{x \mapsto t\}$. By Lemma 6.4.6, $N_k^\perp$ also a conflicting core of N_{k-1}. □

The above lemma considers static completeness, i.e., it does not tell how the conflicting core that can eventually be lifted is found. One way is to enumerate all ground conflicting cores of N_k in a fair way. A straightforward fairness criterion is to order conflicting cores by increasing term depth of their clauses.

Assume N_0 is unsatisfiable and N_k is an MSL approximation of N_0. There is a smallest conflicting core $N^\perp$ of N_0 and by Lemma 6.4.7, a corresponding ground conflicting core $N'^\perp$ of N_k. Note that in the construction of $N'^\perp$ none of the steps increases the term depth of the clauses in $N^\perp$. Furthermore, the total number of distinct S-predicates in $N'^\perp$ is bounded by the term depths in $N^\perp$. Therefore, any conflicting core that can be lifted to $N^\perp$ is bounded by the term depths of $N^\perp$.

Since the approximation is in the MSL fragment, the decision procedure returns a resolution refutation of N_k. Let $N_k^\perp$ be the conflicting core translate from the refutation via Lemma 6.4.3. I assume that the decision procedure follows the fairness criterion such that $N_k^\perp$ contains the smallest ground conflicting core of N_k. If $N_k^\perp$ can not be lifted to N_0, I use Lemma 6.5.6 to prevent the same conflicting core from being found again. Eventually, a conflicting core that can be lifted to $N^\perp$ will be the smallest ground conflicting core of the refined approximation.

6.5 Approximation-Refinement

Thanks to the addition of straight dismatching constraints, the new refinement is now quite different from the previous versions for MSLH and MSL (see Section 4.3 and 5.3). As a result of the changed refinement, I have also adapted a new approach to the whole approximation-refinement. Whereas previously the refinement was defined separately for the linear and shallow cases, now both are generalized and treated as the same type of lift-conflict.

Consider the two cases where a lift-conflict can occur: In the linear case, there exists a clause in the conflicting core that is not an instance of the original clauses. In the shallow case, there exists a pair of clauses whose resolvent is not an instance of the original clauses. Since lifting monadic and refinement transformations always succeeds, there are no lift-conflicts in those cases.

Whenever lifting fails, there is a clause of some form, the so-called lift-conflict, that is not an instance of an approximated clause. In the following this clause is defined as a general lift-conflict which is then used to determine the refinement independently of the rule that caused it.

Definition 6.5.1 (The Lift-Conflict). *Let $N \cup \{(C,\pi)\} \Rightarrow_{\mathrm{LI}} N \cup \{(C_a,\pi_a)\}$ and $N^\perp$ be a minimal ground conflicting core of $N \cup \{(C_a,\pi_a)\}$. A conflict clause $C_c \in N^\perp$ with the instantiated clause (C_a,π_a) is called a lift-conflict if C_c is not an instance of (C,π) modulo duplicate literal elimination.*

Let $N \cup \{(C,\pi)\} \Rightarrow_{\mathrm{SH}} N \cup \{(C_l,\pi_l),(C_r,\pi_r)\}$, $(C_a;\pi_a)$ be the shallow resolvent and $N^\perp$ be a minimal ground conflicting core of $N \cup \{(C_l,\pi_l),(C_r,\pi_r)\}$. The resolvent C_c of $C_l\delta_l \in N^\perp$ and $C_r\delta_r \in N^\perp$ is called a lift-conflict if C_c is not an instance of (C,π) modulo duplicate literal elimination. Then, C_c is an instance of $(C_a;\pi_a)$, which is the instantiated *clause of C_c.*

The goal of refinement is then to change the parent clause in such a way that is both satisfiability equivalent and prevents the lift-conflict from again appearing during the lifting of the refined approximations. Solving the refined approximation will then necessarily produce a new proof because its conflicting core has to be different. For this purpose, the refinement transformation segments the original clause $(C;\pi)$ into two parts $(C;\pi \wedge x \neq t)$ and $(C;\pi)\{x \mapsto t\}$.

For example, consider N and its Linear transformation N'.

$$\begin{array}{rcl} & \rightarrow & P(x,x) \\ P(a,b) & \rightarrow & \end{array} \quad \Rightarrow_{\mathrm{LI}} \quad \begin{array}{rcl} & \rightarrow & P(x,x') \\ P(a,b) & \rightarrow & \end{array}$$

The ground conflicting core of N' is

$$\begin{array}{rcl} & \rightarrow & P(a,b) \\ P(a,b) & \rightarrow & \end{array}$$

Because $P(a, b)$ is not an instance of $P(x, x)$, lifting fails. $P(a, b)$ is the lift-conflict. Specifically $\{x \mapsto a\}$ and $\{x \mapsto b\}$ are conflicting substitutions for the parent variable x. Pick $\{x \mapsto a\}$ to segment $P(x, x)$ into $(P(x, x); x \neq a)$ and $P(x, x)\{x \mapsto a\}$. Now, any descendant of $(P(x, x); x \neq a)$ cannot have a at the position of the first x, and any descendant of $P(x, x)\{x \mapsto a\}$ must have an a at the position of the second x. Thus, $P(a, b)$ is excluded in both cases and no longer appears as a lift-conflict.

To show that the lift-conflict will not reappear in the general case, I use that the conflict clause and its ancestors have strong ties between their term structures and constraints.

Definition 6.5.2 (Constrained Term Skeleton). *The* constrained term skeleton *of a term t under constraint π,* $\text{skt}(t, \pi)$*, is defined as the normal form of the following transformation:*

$$(t[x]_{p,q}; \pi) \Rightarrow_{\text{skt}} (t[q/x']; \pi \wedge \pi\{x \mapsto x'\}), \textit{ where } p \neq q \textit{ and } x' \textit{ is fresh.}$$

The constrained term skeleton of a term t is essentially a linear version of t where the restrictions on each variable position imposed by π are preserved. For (t, π), a solution δ of π is over $\text{lvar}(\pi) \cup \text{vars}(t)$ and $t\delta$ is called a ground instance of (t, π).

Lemma 6.5.3. *Let $N_0 \Rightarrow^*_{\text{APC}} N_k$, $(C_k; \pi_k)$ in N with the ancestor clause $(C_0; \pi_0) \in N_0$ and $(N_k^\perp; \top)$ be a minimal ground conflicting core of N_k. Let δ be a solution of π_k such that $C_k\delta$ is in $N_k^\perp$. If (L', q') is a literal position in $(C_k; \pi_k)$ with the ancestor (L, q) in (C_0, π_0), then (i) $L'\delta|_{q'}$ is an instance of* $\text{skt}(L|_q, \pi_0)$*, (ii) $q = q'$ if L and L' have the same predicate, and (iii) if $L'|_{q'} = x$ and there exists an ancestor variable y of x in (C_0, π_0), then $L|_q = y$.*

Proof. By induction on the length of the approximation $N_0 \Rightarrow^*_{\text{APC}} N_k$.

The base case $N_k = N_0$, is trivial.

Let $N_0 = N \cup \{(C; \pi)\} \Rightarrow_{\text{SH}} N \cup \{(C_l; \pi_l), (C_r; \pi_r)\} = N_k$, $(C_k; \pi_k)$ be the shallow ρ-resolvent and $C_k\delta$ be the resolvent of two instances of $(C_l; \pi_l)$ and $(C_r; \pi_r)$ in $N_k^\perp$. Then, $(C_k; \pi_k)\rho^{-1}$ is equal to $(C; \pi)$ modulo duplicate literal elimination. Thus, by definition $(L, q) = (L', q')\rho^{-1}$. Therefore, (i) $L'\delta|_{q'}$ is an instance of $\text{skt}(L|_q, \pi_0)$, (ii) $q = q'$ if L and L' have the same predicate, and (iii) if $L'|_{q'} = x$ and there exists an ancestor variable y of x in (C_0, π_0), then $L|_q = y$.

Now, let $N_0 \Rightarrow_{\text{APC}} N_1 \Rightarrow^*_{\text{APC}} N_k$. Since (L', p) has an ancestor literal position in (C_0, π_0), the ancestor clause of $(C_k; \pi_k)$ in N_1, (C_1, π_1), contains the the ancestor literal position (L_1, q_1), which has (L, q) as its parent literal position. By the induction hypothesis on $N_1 \Rightarrow^*_{\text{APC}} N_k$, (i) $L'\delta|_{q'}$ is an instance of $\text{skt}(L_1|_{q_1}, \pi_1)$, (ii) $q_1 = q'$ if L_1 and L' have the same predicate, and (iii) if $L'|_{q'} = x$ and there is an ancestor variable y_1 of x in (C_1, π_1), then $L_1|_{q_1} = y_1$.

Let $N_0 = N \cup \{(C; \pi)\} \Rightarrow_{\text{Ref}} N \cup \{(C; \pi \wedge x \neq t), (C; \pi)\{x \mapsto t\}\} = N_1$. If (C_1, π_1) is neither $(C; \pi \wedge x \neq t)$ nor $(C; \pi)\{x \mapsto t\}$, then trivially $(C_0, \pi_0) = (C_1, \pi_1)$. Otherwise, $(C_1, \pi_1) = (C; \pi \wedge x \neq t)$ or $(C_1, \pi_1) = (C; \pi)\{x \mapsto t\}$. Then $(L_1, q_1) = (L, q)$ or

$(L_1, q_1) = (L, q)\{x \mapsto t\}$. In either case,(i) $L'\delta|_{q'}$ is an instance of $\text{skt}(L|_q, \pi_0)$, (ii) $q = q'$ if L and L' have the same predicate, and (iii) if $L'|_{q'} = x$ and there exists an ancestor variable y of x in (C_0, π_0), then $L|_q = y$.

Let $N_0 \Rightarrow_{\text{MO}} \mu_P(N) = N_1$. If P is not the predicate of L, then trivially $(L, q) = (L_1, q_1)$. If P is the predicate of L, then $(L, q) = (P(t_1, \ldots, t_n), q)$ and $(L_1, q_1) = (T(f_p(t_1, \ldots, t_n)), 1.q)$. Thus, (i) $L'\delta|_{q'}$ is an instance of $\text{skt}(L|_q, \pi_0) = \text{skt}(T(f_p(t_1, \ldots, t_n)|_{1.q}, \pi_0)$. (ii) The predicate of L' is not P by definition. (iii) Let $L'|_{q'} = x$ and y be the ancestor variable of x in (C_0, π_0). Then, y is also the ancestor variable of x in (C_1, π_1) and $L_1|_{q_1} = y$. Therefore, $L|_q = P(t_1, \ldots, t_n)|_q = T(f_p(t_1, \ldots, t_n)|_{1.q} = L_1|_{q_1} = y$.

Let $N_0 = N \cup \{(C; \pi)\} \Rightarrow_{\text{LI}} N \cup \{(C_a; \pi_a)\} = N_1$ where an occurrence of a variable x is replaced by a fresh x'. If $(C_1, \pi_1) \neq (C_a; \pi_a)$, then trivially $(C_0, \pi_0) = (C_1, \pi_1)$. Otherwise, $(C_1, \pi_1) = (C_a; \pi_a)$, $(C_0, \pi_0) = (C, \pi)$. By definition, $(L, q) = (L_1\{x' \mapsto x\}, q_1)$ and $\pi_0 = \pi_1\{x' \mapsto x\}$. Thus, $\text{skt}(L|_q, \pi_0) = \text{skt}(L_1|_{q_1}, \pi_1)$. Therefore, $L'\delta|_{q'}$ is an instance of $\text{skt}(L|_q, \pi_0)$. Since L and L_1 have the same predicate and $q = q_1$, $q = q'$ if L and L' have the same predicate. Let $L'|_{q'} = z$ and y be the ancestor variable of z in (C_1, π_1). If $y \neq x'$, then y is the ancestor variable of z in (C_0, π_0) and $L|_q = L_1\{x' \mapsto x\}|_{q_1} = y_1$. Otherwise, x is the ancestor variable of z in (C_0, π_0) and $L|_q = L_1\{x' \mapsto x\}|_{q_1} = x$.

Let $N_0 = N \cup \{(C; \pi)\} \Rightarrow_{\text{SH}} N \cup \{(C_l; \pi_l), (C_r; \pi_r)\} = N_1$ where a term s is extracted from a positive literal $Q(s'[s]_p)$ via introduction of the fresh predicate S and variable x. If (C_1, π_1) is neither $(C_l; \pi_l)$ nor $(C_r; \pi_r)$, then trivially $(C_0, \pi_0) = (C_1, \pi_1)$.

If $(C_1, \pi_1) = (C_l; \pi_l)$ and $L_1 = S(x)$, then $(C_0, \pi_0) = (C; \pi)$, $q_1 = 1$, $(L', q') = (S(x), 1)$ and $(Q(s'[s]_p), 1.p)$ is the parent literal position of $(S(x), 1)$. Let $L'\delta = S(t)$. Since $N_k^\perp$ is minimal and ground, there is a clause $C_k'\delta' \in N_k^\perp$ that contains $S(t)$ as a positive literal, because otherwise no resolution refutation with clauses in $N_k^\perp$ involving C_k could have derived the empty clause. The ancestor of $(C_k', \pi_k') \in N_k$ in N_1 is $(C_r; \pi_r)$ because it is the only clause in N_1 with a positive S-literal. Then, by the inductive hypothesis, $(S(s), 1)$ in $(C_r; \pi_r)$ is the ancestor literal position of $(S(x), 1)$ in (C_k', π_k'). Thus, t is an instance of $\text{skt}(S(s)|_1, \pi_r) = \text{skt}(s, \pi_r)$. Therefore, $t = L'\delta|_{q'}$ is an instance of $\text{skt}(Q(s'[s]_p)|_{1.p}, \pi) = \text{skt}(s, \pi_r)$. Further, Q and S are not the same predicate because S is fresh. Since x has no parent variable, $L'|_{q'} = x$ has no ancestor variable in (C_0, π_0).

If $(C_1, \pi_1) = (C_l; \pi_l)$ and $L_1 = Q(s'[p/x])$, $(C_0, \pi_0) = (C; \pi)$ and $(Q(s'[s]_p), q_1)$ in $(C; \pi)$ is the parent literal position of (L_1, q_1) in (C_1, π_1) and ancestor literal position of (L', q') in (C_k, π_k). If q_1 is not a position at or above p, the subterm at p is irrelevant and thus $\text{skt}(Q(s'[s]_p)|_{q_1}, \pi) = \text{skt}(Q(s'[p/x])|_{q_1}, \pi_l)$. Otherwise, let r be a position such that $q_1 r = 1.p$. Since $|p| = 2$, no following Shallow transformation step extracts a subterm of $s'[p/x]$ containing x. Thus by definition of $\Rightarrow_{\text{APC}}$, $L' = Q(t'[x]_p)$ and C_k also contains the negative literal $S(x)$. Let $S(x)\delta = S(t)$. Analogously to the previous case, t is an instance of $\text{skt}(s, \pi_r)$. Combined with $L'\delta|_{q'}$ being an instance of $\text{skt}(L_1|_{q_1}, \pi_1) = \text{skt}(Q(s'[p/x])|_{q_1}, \pi_l)$ and $L'\delta|_{1.p} = t$, $L'\delta|_{q'}$ is an instance of $\text{skt}(Q(s'[s]_p)|_q, \pi)$. Since L and L_1 have the same predicate and $q = q_1$, $q = q'$ if L and L' have the same predicate. Let $L'|_{q'} = z$ and y in (C_1, π_1) be the ancestor variable of z in (C_k, π_k). Since x has no parent, $y \neq x$ and y in (C_0, π_0) is the ancestor variable of z. Therefore, $Q(s'[s]_p)|_{q_1} = y$ because $Q(s'[p/x])|_{q_1} = y$.

If $(C_1, \pi_1) = (C_r; \pi_r)$ and $L_1 = S(s)$, let $q_1 = 1.q'_1$. Then, $(C_0, \pi_0) = (C; \pi)$ and $(L, q) = (Q(s'[s]_p), 1.pq'_1)$ in (C_0, π_0) is the parent literal position of (L_1, q_1) in (C_1, π_1). Thus, $L'\delta|_{q'}$ is an instance of $\text{skt}((Q(s'[s]_p)|_{1.pq'_1}, \pi) = \text{skt}(s|_{q'_1}, \pi) = \text{skt}(L_1|_{q_1}, \pi_r)$. Because S is fresh, Q is not the predicate of L'. Let $L'|_{q'} = z$ and y in (C_1, π_1) be the ancestor variable of z in (C_k, π_k). Then, y in (C_0, π_0) is the ancestor variable of z and $Q(s'[s]_p)|_q = s|_{q'_1} = y$ because $s|_{q'_1} = L_1|_{q_1} = y$.

Otherwise, (L_1, q_1) in (C_0, π_0) is the parent literal position of (L_1, q_1) in (C_1, π_1), by definition. Then, $\text{skt}(L_1, \pi) = \text{skt}(L_1, \pi_l)$ or $\text{skt}(L_1, \pi) = \text{skt}(L_1, \pi_r)$, respectively.

$\square$

Next, I define the notion of descendants and descendant relations to connect lift-conflicts in ground conflicting cores with their corresponding ancestor clauses. The goal, hereby, is that if a ground clause D is not a descendant of a clause in N, then it can never appear in a conflicting core of an approximation of N.

Definition 6.5.4 (Descendants)**.** *Let $N \Rightarrow^*_{\text{APC}} N'$, $[(C;\pi), N] \Rightarrow^*_{\text{A}} [(C';\pi'), N']$ and D be a ground instance of $(C';\pi')$. Then, D is called a* descendant *of $(C;\pi)$. Define the $[(C;\pi), N] \Rightarrow^*_{\text{A}} [(C';\pi'), N']$-descendant relation $\Rightarrow_D$ that maps literals in D to literal positions in $(C;\pi)$ using the following rule:*

$$L'\delta \Rightarrow_D (L, r) \text{ if } L'\delta \in D \text{ and } [r, L, (C;\pi), N] \Rightarrow^*_{\text{A}} [\varepsilon, L', (C';\pi'), N']$$

For the descendant relations it is of importance to note that while there are potentially infinite ways that a lift-conflict C_c can be a descendant of an original clause $(C;\pi)$, there are only finitely many distinct descendant relations over C_c and $(C;\pi)$. This means, if a refinement transformation can prevent one distinct descendant relation without allowing new distinct descendant relations (Lemma 6.5.5), a finite number of refinement steps can remove the lift-conflict C_c from the descendants of $(C;\pi)$ (Lemma 6.5.6). Thereby, preventing any conflicting cores containing C_c from being found again.

A clause $(C;\pi)$ can have two descendants that are the same except for the names of the S-predicates introduced by Shallow transformations. Because the used approximation $N \Rightarrow^*_{\text{APC}} N'$ is arbitrary and therefore also the choice of fresh S-predicates, if D is a descendant of $(C;\pi)$, then any clause D' equal to D up to a renaming of S-predicates is also a descendant of $(C;\pi)$. On the other hand, the actual important information about an S-predicate is which term it extracts. Two descendants of $(C;\pi)$ might be the exactly the same but their S-predicate extract different terms in $(C;\pi)$. For example, $P(a) \rightarrow S(f(a))$ is a descendant of $P(x), P(y) \rightarrow Q(f(x), g(f(x)))$ but might extract either occurrence of $f(x)$. These cases are distinguished by their respective descendant relations. In the example, they are either $S(f(a)) \Rightarrow_D (Q(f(x), g(f(x))), 1)$ or $S(f(a)) \Rightarrow_D (Q(f(x), g(f(x))), 2.1)$.

Lemma 6.5.5. *Let $N_0 = N \cup \{(C;\pi)\} \Rightarrow_{\text{Ref}} N \cup \{(C;\pi \wedge x \neq t), (C;\pi)\{x \mapsto t\}\} = N_1$ be a refinement transformation and D be a ground clause. If there exists a $[(C;\pi \wedge x \neq t), N_1] \Rightarrow^*_{\text{A}} [(C';\pi'), N_2]$- or $[(C;\pi)\{x \mapsto t\}, N_1] \Rightarrow^*_{\text{A}} [(C';\pi'), N_2]$-descendant relation $\Rightarrow^1_D$, then there is an equal $[(C;\pi), N_0] \Rightarrow^*_{\text{A}} [(C';\pi'), N_2]$-descendant relation $\Rightarrow^0_D$.*

Proof. Let L_D be a literal of D and $L' \Rightarrow^1_D (L, r)$. If D is a descendant of $(C;\pi \wedge x \neq t)$, then $[r, L, (C;\pi \wedge x \neq t), N_1] \Rightarrow^*_{\text{A}} [\varepsilon, L', (C';\pi'), N_2]$. Because $[r, L, (C;\pi), N_0] \Rightarrow_{\text{A}} [r, L, (C;\pi \wedge x \neq t), N_1]$, $L' \Rightarrow^0_D (L, r)$. If D is a descendant of $(C;\pi)\{x \mapsto t\}$, the proof is analogous. □

Lemma 6.5.6 (Refinement). *Let $N \Rightarrow_{\text{APC}} N'$ and $N^\perp$ be a minimal ground conflicting core of N'. If $C_c \in N^\perp$ is a lift-conflict, then there exists a finite refinement $N \Rightarrow^*_{\text{Ref}} N_R$ such that for any approximation $N_R \Rightarrow^*_{\text{APC}} N'_R$ and ground conflicting core $N^\perp_R$ of N'_R, C_c is not a lift-conflict in $N^\perp_R$ modulo duplicate literal elimination.*

Proof. Let (C_a, π_a) be the conflict clause of C_c and $(C;\pi) \in N$ be the parent clause of (C_a, π_a). C_c is a descendant of $(C;\pi)$ with the corresponding $[(C;\pi), N] \Rightarrow_{\text{A}} [(C_a;\pi_a), N']$-descendant relation $\Rightarrow^0_{C_c}$. Apply induction on the number of distinct $[(C;\pi), N] \Rightarrow^*_{\text{A}} [(C';\pi'), N'']$-descendant relations $\Rightarrow_{C_c}$ for arbitrary approximations $N \Rightarrow^*_{\text{APC}} N''$.

Since only the Shallow and Linear transformations can produce lift-conflicts, the clause $(C;\pi)$ is replaced by either a linearised clause $(C';\pi')$ or two shallow clauses $(C_l;\pi)$ and $(C_r;\pi)$. Then, the conflict clause $(C_a;\pi_a)$ of C_c is either the linearised $(C';\pi')$ or the resolvent of $(C_l;\pi)$ and $(C_r;\pi)$. In either case, $C_c = C_a\delta$ for some solution δ of π_a. Furthermore, there exists a substitution $\tau = \{x'_1 \mapsto x_1, \ldots, x'_n \mapsto x_n\}$ such that $(C;\pi)$ and $(C_a;\pi_a)\tau$ are equal modulo duplicate literal elimination. That is, $\tau = \{x' \mapsto x\}$ for a Linear transformation and $\tau = \rho^{-1}$ for Shallow transformation (Definition 6.3.3).

Assume $C_c = C_a\tau\sigma$ for some grounding substitution σ, where $\tau\sigma$ is a solution of π_a. Thus, σ is a solution of $\pi_a\tau$, which is equivalent to π. Then, C_c is equal to $C\sigma$ modulo duplicate literal elimination an instance of $(C;\pi)$, which contradicts with C_c being a lift-conflict. Hence, $C_c = C_a\delta$ is not an instance of $C_a\tau$ and thus, $x_i\delta \neq x'_i\delta$ for some x_i in the domain of τ.

Because $x_i\delta$ and $x'_i\delta$ are ground, there is a position p where $x_i\delta|_p$ and $x'_i\delta|_p$ have different function symbols. Construct the straight term t using the path from the root to p on $x_i\delta$ with variables that are fresh in (C, π). Then, use x_i and t to segment $(C;\pi)$ into $(C;\pi \wedge x_i \neq t)$ and $(C;\pi)\{x_i \mapsto t\}$ for the refinement $N \Rightarrow_{\text{Ref}} N_R$. Note, that $x_i\delta$ is a ground instance of t, while $x'_i\delta$ is not.

Let (L'_1, r'_1) and (L'_2, r'_2) in (C_a, π_a) be literal positions of the variables x_i and x'_i in C_a, and (L_1, r_1) and (L_2, r_2) in (C, π) be the parent literal positions of (L'_1, r'_1) and (L'_2, r'_2), respectively. Because $(C_a, \pi_a)\tau$ is equal to $(C;\pi)$ modulo duplicate literal elimination, $L_1|_{r_1} = L_2|_{r_2} = x_i$. Let $N \Rightarrow_{\text{Ref}} N_1$ be the refinement where $(C;\pi)$ is segmented into $(C;\pi \wedge x_i \neq t)$ and $(C;\pi)\{x_i \mapsto t\}$.

By Lemma 6.5.5, there exists for every $[(C;\pi \wedge x_i \neq t), N_1] \Rightarrow^*_{\mathrm{A}} [(C'_a;\pi'_a), N_2]$- or $[(C;\pi)\{x_i \mapsto t\}, N_1] \Rightarrow^*_{\mathrm{A}} [(C'_a;\pi'_a), N_2]$-descendant relation a corresponding equal $[(C;\pi), N] \Rightarrow_{\mathrm{A}} [(C'_a;\pi'_a), N_2]$-descendant relation. Now, assume there exists a $[(C;\pi \wedge x_i \neq t), N_1] \Rightarrow^*_{\mathrm{A}} [(C'_a;\pi'_a), N_2]$-descendant relation $\Rightarrow^1_{C_c}$ that is not distinct from $\Rightarrow^0_{C_c}$. Because $L'_1\delta \Rightarrow^0_{C_c} (L_1, r)$ for some literal position (L_1, r) in $(C;\pi)$, which is the parent literal position of (L_1, r) in $(C;\pi \wedge x_i \neq t)$, $L'_1\delta \Rightarrow^1_{C_c} (L_1, r)$. However, this contradicts Lemma 6.5.3 because $x_i\delta$ is not an instance of $\mathrm{skt}(L_1|_{r_1}, \pi \wedge x_i \neq t) = \mathrm{skt}(x_i, \pi \wedge x_i \neq t)$. The case that there is a $[(C;\pi)\{x_i \mapsto t\}, N_1] \Rightarrow^*_{\mathrm{A}} [(C'_a;\pi'_a), N_2]$-descendant relation that is not distinct from $\Rightarrow^0_{C_c}$ is analogous using the argument that $x'_i\delta$ is not an instance of $\mathrm{skt}(L_2\{x_i \mapsto t\}|_{r_2}, \pi) = \mathrm{skt}(t, \pi)$. Hence, there are strictly less distinct descendant relations over C_c and $(C;\pi \wedge x \neq t)$ or $(C;\pi)\{x \mapsto t\}$ than there are distinct descendant relations over C_c and (C, π).

If there are no descendant relations, then C_c can no longer appear as a lift conflict. Otherwise, by the inductive hypothesis, there exists a finite refinement $N \Rightarrow_{\mathrm{Ref}} N_1 \Rightarrow^*_{\mathrm{Ref}} N_R$ such that for any approximation $N_R \Rightarrow_{\mathrm{APC}} N'_R$ and ground conflicting core $N^\perp_R$ of N'_R, C_c is not a lift-conflict in $N^\perp_R$ modulo duplicate literal elimination. □

6.5.1 Improvements to Refinement

In Lemma 6.5.6, I segment the conflict clause's immediate parent clause. If the lifting later successfully passes this point, the refinement is lost and will be possibly repeated. Instead, I can refine any ancestor of the conflict clause as long as it contains the ancestor of the variable used in the refinement. By Lemma 6.5.3(iii), such an ancestor will contain the ancestor variable at the same positions. If I refine the ancestor in the original clause set, the refinement is permanent because lifting the refinement steps always succeeds. Only variables introduced by Shallow transformation cannot be traced to the original clause set. However, these shallow variables are already linear and the partitioning in the Shallow transformation can be chosen fixed such that they are not shared variables. Assume a shallow variable y, that is used to extract term t, is a shared variable in the Shallow transformation of $\Gamma \rightarrow E[s]_p, \Delta$ into $S(x), \Gamma_l \rightarrow E[p/x], \Delta_l$ and $\Gamma_r \rightarrow S(s), \Delta_r$. Since $\Delta_l \dot{\cup} \Delta_r = \Delta$ is a partitioning, y can only appear in either $E[p/x], \Delta_l$ or $S(s), \Delta_r$. If $y \in \mathrm{vars}(E[p/x], \Delta_l)$, instantiate Γ_r with $\{y \mapsto t\}$ and Γ_l, otherwise. Now, y is no longer a shared variable.

The refinement Lemmas only guarantee a refinement for a given ground conflicting core. In practice, however, conflicting cores contain free variables. If I only exclude a single ground case via refinement, next time the new conflicting core will likely have overlap with the previous one. But, I can often use specific instances to remove all ground instances of a given conflict clause at once.

The simplest case is when unifying the conflict clause with the original clause fails because their instantiations are structurally different at some position. For example, consider the clause set $N = \{P(x, x); P(f(x, a), f(y, b)) \rightarrow\}$. It is satisfiable but the Linear transformation is unsatisfiable with the lift-conflict $P(f(x, a), f(y, b))$

which is not unifiable with $P(x,x)$, because the two terms $f(x,a)$ and $f(y,b)$ have different functions at the second argument. A refinement of $P(x,x)$ is

$$\begin{aligned}(P(x,x);\ x \neq f(v,a))\\ (P(f(x,a),f(x,a));\ \top)\end{aligned}$$

$P(f(x,a),f(y,b))$ shares no ground instances with the refined approximations.

Next, assume that again unification fails due to structural difference, but this time the differences lie at different positions. For example, consider the clause set $N = \{P(x,x); P(f(a,b),f(x,x)) \rightarrow\}$. It is satisfiable but the Linear transformation of N is unsatisfiable with the lift-conflict $P(f(a,b),f(x,x))$ which is not unifiable with $P(x,x)$ because in $f(a,b)$ the first an second argument are different but the same in $f(x,x)$. A refinement of $P(x,x)$ is

$$\begin{aligned}(P(x,x);\ x \neq f(a,v))\\ (P(f(a,x),f(a,x)));\ x \neq a)\\ (P(f(a,a),f(a,a)));\ \top)\end{aligned}$$

$P(f(a,b),f(x,x))$ shares no ground instances with the refined approximations.

It is also possible that the conflict clause and original clause are unifiable by themselves, but the resulting constraint has no solutions. For example, consider the clause set $N = \{P(x,x); (P(x,y) \rightarrow; x \neq a \land x \neq b \land y \neq c \land y \neq d)\}$ with signature $\Sigma = \{a,b,c,d\}$. It is satisfiable but the Linear transformation of N is unsatisfiable with the lift-conflict $(\rightarrow P(x,y); x \neq a \land x \neq b \land y \neq c \land y \neq d)$. While $P(x,x)$ and $P(x,y)$ are unifiable, the resulting constraint $x \neq a \land x \neq b \land x \neq c \land x \neq d$ has no solutions. A refinement of $P(x,x)$ is

$$\begin{aligned}&(P(x,x);\ x \neq a \land x \neq b)\\ &(P(a,a);\ \top)\\ &(P(b,b);\ \top)\end{aligned}$$

$(P(x,y); x \neq a \land x \neq b \land y \neq c \land y \neq d)$ shares no ground instances with the approximations of the refined clauses.

Lastly, I should mention that there are cases where a refinement covering all ground instances of the conflict clause is impossible. For example, consider the clause set $N = \{P(x,x); P(y,g(y)) \rightarrow\}$. It is satisfiable but the Linear transformation of N is unsatisfiable with the lift-conflict $P(y,g(y))$, which is not unifiable with $P(x,x)$ because y and $g(y)$ cannot be the same term. A refinement of $P(x,x)$ based on the ground instance $P(a,g(a))$ is

$$\begin{aligned}(P(x,x);\ x \neq g(v))\\ (P(g(x),g(x));\ \top)\end{aligned}$$

While $P(y, g(y))$ is not an instances of the approximations of the refined clauses, it shares ground instances with $P(g(x), g(x'))$. The new lift-conflict is $P(g(y), g(g(y)))$ and the refinement will continue to enumerate all $P(g^i(x), g^i(x))$ instances of $P(x, x)$ without ever reaching a satisfiable approximation.

If instead of approximating $P(x, x)$ with $P(x, x')$, something on the line of $(P(x, x'); x' \neq g(x))$ could be used, satisfiability would be preserved. However, $x' \neq g(x)$ is not a dismatching constraint since x appears in the clause. Decidability is not known to me for the MSL fragment with such disequality constraints.

For the MSLH fragment with disequality constraints, decidability has been shown in [38] via transformation into bottom-up tree automata [36]. There, MSLH clauses are called $\mathcal{H}_1$-clauses. Disequality constraints $\phi = \bigwedge_{i \in \mathrm{I}} \; s_i \neq t_i$ differ from dismatching constraints in that a clause $C\sigma$ is a ground instance of (C, ϕ) if $s_i\sigma \neq t_i\sigma$ for every $i \in \mathrm{I}$.

My attempts to prove decidability by combining the resolution approach with the automata approach in [38] failed. On one hand, [36] relies on Horn clauses as input to construct automata clauses. On the other hand, the termination argument in Lemma 5.1.4 uses the fact that clauses without selected literals can be separated into variable disjoint Horn components. However, this is not apparent for MSL clauses with disequality constraints such as $(\rightarrow P(x), Q(y); x \neq y)$.

Chapter 7

Implementation

I have implemented the first-order approximation-refinement calculus (FO-AR) described in Chapter 6 as an extension of SPASS v.3.8 called SPASS-AR. As shown in Sections 5.1 and 6.2, the MSL and MSL(SDC) fragments can be decided using ordered resolution with the specific selection function sel. Therefore, by just using sel as the default selection function, SPASS already provides the decision procedure for MSL.

The addition of the straight dismatching constraints to the signature of clauses follows in a straight-forward way their description in Section 6.1. In relation to the main saturation loop of SPASS, the constraints can in most places be treated as a black-box that given a constraint either returns its normalization or NULL indicating an unsolvable constraint.

The remaining steps of FO-AR, approximation, lifting, and refinement, are build around the core routine of SPASS. A given clause set is first approximated. Each transformation step is documented in the same way resolution and reduction rule applications are documented by adding the information of rule, parent clauses and affected literals to the approximation clause. The resulting approximation is given as input to the modified solver.

Once finished, the result is analysed. If satisfiable, the result is reported and the routine finished. Otherwise, the empty clause returned by SPASS is given as input to the lifting. For the sake of the lifting, each inferred clause had its parents documented and even if it was redundant was not deleted. This information is then used to construct a DAG of the resolution refutation.

In a first version of SPASS-AR, the conflicting core was then recursively constructed and step-wise lifted as described in Section 6.4. However, the exponential size of the conflicting core compared to the refutation DAG made this approach impractical in many examples. Therefore, it was later replaced by a lifting method that, while much more complex, works directly on the smaller DAG.

If the lifting succeeds, the conflicting core or the refutation Dag lifted to the original clause set can be reported as a proof of unsatisfiability. Otherwise, all relevant information of the failed lifting step is returned as the lift-conflict and given to the refinement function.

First, the refinement uses a modified unification algorithm on the conflict clause and its parent to identify the positions of the two subterms in the conflict clause that caused the unification to fail. One of those positions is then traced upwards to the point were in the parent clause a variable is found. As the conflict clause is by construction an instance of the parent clause's term skeleton, this tracing necessarily leads to the same variable for both positions. Next, I climb the ancestry tree of the conflict clause to its corresponding original clause while keeping track of the incorrectly instantiated variable. The original clause $(C;\pi)$, its variable x and one of the instantiations t are used to replace $(C;\pi)$ with the refinements $(C;\pi)\{x \mapsto t\}$ and $(C;\pi \wedge x \neq t)$. Lastly after resetting the solvers data-structures, the refined clause set is again fed to the approximation.

7.1 Approximation Details

While Monadic and Linear transformation are implemented just as described in their definitions, the definition of the Shallow transformation left certain parts unspecified which will be discussed in the following. Furthermore, to avoid unnecessary repetition, satisfiability equivalent transformation steps are applied in a preprocessing phase before starting the approximation-refinement loop proper.

7.1.1 Shallow Transformation

In the definition of the Shallow transformation, I noted that the distribution of literals from Γ and Δ into Γ_l, Γ_r, Δ_l, and Δ_r can be almost arbitrarily chosen as long as none are dropped (see Section 3.3). Even duplicating literals and replacing s with x in Γ_l is allowed. In the following, I will introduce the way the implementation actually applies the Shallow transformation. Though I will ignore constraints here for simplicity.

The first place where the implementation deviates from the definition is that ground terms in positive literals are not extracted even if they are not shallow. For the decidability of the MSL and MSLH fragments, only complex subterms that contain variables are actually significant because they could potentially lead to terms growing indefinitely. There is no such risk with ground terms and flattening them would simply introduce an excessively large number of new predicates. Otherwise, selection of the extraction term is arbitrary.

Given a clause $\Gamma \rightarrow E[s]_p, \Delta$ where the subterm s is chosen for extraction, I first collect the variables in s, $\mathcal{V}_s := \text{vars}(s)$, and in the literal $E[s]_p$ without s, $\mathcal{V}_E := \text{vars}(E[p/c])$ where c is an arbitrary constant. Next, I pick a fresh variable x and replace each occurrence of s in Γ with x creating a Γ' where $\Gamma'\{x \mapsto s\} = \Gamma$. Now, I consider the literals in $\Gamma' \cup \Delta$ as nodes in a graph where two nodes are connected if they share a variable. Given a set of variables $\mathcal{V}$, I use reachability in this graph to find all literals that either contain a variable in $\mathcal{V}$ or are connected to

such a literal. I call these the literals connected to $\mathcal{V}$. Using this, I separate Γ' and Δ into four distinct groups:
Γ'_s and Δ_s that contain the literals connected to $\mathcal{V}_s$ but not to $\mathcal{V}_E$;
Γ'_E and Δ_E that contain the literals connected to $\mathcal{V}_E$ but not to $\mathcal{V}_s$;
Γ'_S and Δ_S that contain the literals connected to both $\mathcal{V}_s$ and $\mathcal{V}_E$;
Γ'_N and Δ_N that contain the literals connected to neither $\mathcal{V}_s$ nor $\mathcal{V}_E$.

If Γ'_S and Δ_S are empty, then there are no shared variables and I create the satisfiability equivalent approximations

$$S(x), \Gamma'_E, \Gamma'_N \to E[p/x], \Delta_E, \Delta_N$$

$$\text{and } \Gamma'_s\{x \mapsto s\} \to S(s), \Delta_s.$$

If Γ'_S and Δ_S are not empty, let Γ''_S be the subset of literals in Γ'_S that actually contain variables in $\mathcal{V}_s$. I create the approximations

$$S(x), \Gamma'_E, \Gamma'_N, \Gamma'_S \to E[p/x], \Delta_E, \Delta_N, \Delta_S$$

$$\text{and } \Gamma'_s\{x \mapsto s\}, \Gamma''_S\{x \mapsto s\} \to S(s), \Delta_s.$$

With $\{Q(y) \in \Gamma \mid y \in \text{vars}(E[p/x], \Delta_l)\} \subseteq \Gamma'_E$ and $\{Q(y) \in \Gamma \mid y \in \text{vars}(s, \Delta_r)\} \subseteq \Gamma'_s\{x \mapsto s\}$ these approximations fulfil all conditions of the Shallow transformation.

This choice of approximation stems from testing several versions. For one, I do not duplicate any positive literals because this can otherwise cause an exponential blow-up of the approximation. For example, consider

$$\Gamma \to P(g(h(x), x)),\ Q(f(f(x)))$$

where each literal in Γ contains x. If I extract $h(x)$ by replacing it with

$$S(y),\ \Gamma \to P(g(y, x)),\ Q(f(f(x)))$$

$$\text{and } \Gamma \to S(h(x)),\ Q(f(f(x))),$$

then I next need to extract $f(x)$ in both approximation clauses. In the worst case, I would create 2^n approximation clauses if the original clause requires n Shallow transformations in total.

Second, my experiments suggested that copying literals to the right approximation clause that do not share variables with s serves almost no improvement to the precision of the approximation and instead unsatisfiability proofs just require longer refutations.

Lastly, the independent literals are given to the left approximation to further increase the odds of the right approximation being Horn. How such Horn clauses are useful, I address in Section 7.2.2.

7.1.2 Refined Approximation and Preprocessing

Some of the transformations applied in the approximation are satisfiability equivalent which means for the approximation-refinement loop that lifting these steps always succeeds. If such transformations are applied exclusively, the resulting clause set is equivalent to the input set and can therefore be used as the original clause set instead. Then, all approximations applied in this preprocessing phase only need to be done once instead of being possibly repeated during every loop.

Since the Monadic transformation is always satisfiability equivalent and causes only a small increase in the problem size, it is the first step of the preprocessing. Next, I apply all Shallow transformations that have no shared variables. As shown in Section 3.3, in this special case, the Shallow transformations is also a satisfiability equivalent transformation. During the preprocessing, I even go a step further and allow the Shallow transformation to additionally extract all occurrences of s in the positive literals at the same time. Since this step is never lifted or refined, it causes no problems later.

Additionally, I use the preprocessing to prevent a possible exponential blow-up during the actual approximation phase. Specifically, I pre-process clauses where the Linear transformation would create exponentially larger clauses. For example, consider the clause

$$P_1(x), \dots, P_n(x) \to P(x, \dots, x)$$

where $P(x, \dots, x)$ contains m occurrences of x. Then, the Linear transformation creates an approximation clause with n times m negative literals

$$P_1(x_1), \dots, P_n(x_1), \dots, P_1(x_m), \dots, P_n(x_m) \to P(x_1, \dots, x_m).$$

To prevent this explosion, I use renaming [33] to replace the clause with

$$\begin{array}{rl} P_1(x), \dots, P_n(x) \to S(x) & \quad\text{and} \\ S(x) \to P(x, \dots, x) & \end{array}$$

with the fresh predicate S such that later the Linear transformation is instead

$$\begin{array}{rl} P_1(x), \dots, P_n(x) \to S(x) & \quad\text{and} \\ S(x_1), \dots, S(x_m) \to P(x_1, \dots, x_m) & \end{array}$$

In general, let $\Gamma \to \Delta$ be a clause where x occurs m times in Δ and in n negative literals. If $n \cdot m$ is significantly larger than $n + m + 2$, I introduce a fresh function symbol f and replace $\Gamma \to \Delta$ with the two clauses

$$\begin{array}{rl} T(f(x, y_1, \dots, y_n)), \Gamma_{\overline{x}} \to \Delta & \quad\text{and} \\ & \Gamma_x \to T(f(x, y_1, \dots, y_n)) \end{array}$$

where $\Gamma_x \cup \Gamma_{\overline{x}} = \Gamma$, $\text{vars}(\Gamma_x) = \{x, y_1, \dots, y_n\}$, and $x \notin \text{vars}(\Gamma_{\overline{x}})$.

For another example, consider the clause

$$P(x, y) \to Q(x, \dots, x, y, \dots, y)$$

where x and y appear n and m times respectively. Its linear approximation has again n times m negative literals

$$P(x_1, y_1), \dots, P(x_n, y_1), \dots, P(x_1, y_m), \dots, P(x_n, y_m) \to Q(x_1, \dots, x, y_1, \dots, y_m).$$

Again using the fresh function symbol f, I create

$$\begin{aligned} T(f(x_1, \dots, x_n, y)) &\to Q(x_1, \dots, x_n, y, \dots, y) \\ \text{and} \qquad P(x, y) &\to T(f(x, \dots, x, y)) \end{aligned}$$

where the approximation again has size $n + m + 2$ instead of $n \cdot m$.

In general, let $\Gamma \to \Delta$ be a clause where x occurs m times in Δ and together with other non-linear variables in at least one negative literal. I introduce a fresh function symbol f and replace $\Gamma \to \Delta[x, \dots, x]$ with the two clauses

$$\begin{aligned} T(f(x_1, \dots, x_m, y_1, \dots, y_n)), \Gamma_{\overline{x}} &\to \Delta[x_1, \dots, x_m] \qquad \text{and} \\ \Gamma_x &\to T(f(x, \dots, x, y_1, \dots, y_n)) \end{aligned}$$

where $x_1, \dots, x_m$ are fresh, $\Gamma_x \cup \Gamma_{\overline{x}} = \Gamma$, $\text{vars}(\Gamma_x) = \{x, y_1, \dots, y_n\}$, and $x \notin \text{vars}(\Gamma_{\overline{x}})$. As in the previous case, I estimate the sizes of the linear approximation before and after the transformation to decide whether to apply this preprocessing or not. Specifically, let $m_1, \dots, m_n$ be the number of occurrences of $y_1, \dots, y_n$ in Δ, I compare $m_1 \cdot m_2 \cdot \dots \cdot m_n + |\Gamma_x| \cdot m + 2$ with the sum of $m \cdot m_{j_1} \cdot \dots \cdot m_{j_l}$ for each literal $E \in \Gamma_x$ with $\{x, y_{j_1}, \dots, y_{j_l}\} = \text{vars}(E)$.

7.1.3 Preprocessing Reflexive Predicates

As previously mentioned in Sections 6.5.1, the approximation-refinement cannot show satisfiability of the simple clause set with the two clauses

$$\begin{aligned} &\to P(x, x) \\ P(y, g(y)) &\to \end{aligned}$$

The problem is that $\to P(x, x)$ cannot be refined in such a way that all instances of the conflict clause $\to P(y, g(y))$ are excluded. The refinement loop instead ends up enumerating all $(\to P(g^i(x), g^i(x)); x \neq g(v))$ but $\to P(g^{i+1}(y), g^{i+2}(y))$ will always remain as a conflict clause.

I have not found a proper solution to this problem that works in all cases, but as a partial solution to this problem, if the input clause set contains reflexivity axioms such as $\to P(x, x)$, I tag each occurrence of P as reflexive or irreflexive, i.e, whether an atom $P(s, t)$ is unifiable with $P(x, x)$ or not, by creating two new predicates P_{ref}

and P_{irr}. I essentially separate the predicate P into reflexive and irreflexive parts. For the example, this looks like

$$\begin{aligned} &\to P_{\text{ref}}(x, x) \\ P_{\text{irr}}(y, g(y)) &\to \end{aligned}$$

Since the two literals were not resolvable anyway, this replacement is satisfiability equivalent. Now, the approximation is also satisfiable.

In general, for each predicate P with a reflexivity axiom, I replace all occurrences of atoms $P(s, t)$ with $P_{\text{ref}}(s, t)$ and/or $P_{\text{irr}}(s, t)$. If s and t are not unifiable, I replace $P(s, t)$ with $P_{\text{irr}}(s, t)$. Otherwise, there is a most general unifier σ of s and t. In that case, I replace the clause $C \vee [\neg]P(s, t)$ with $C_{\text{irr}} = C \vee [\neg]P_{\text{irr}}(s, t)$ and $C_{\text{ref}} = C\sigma \vee [\neg]P_{\text{ref}}(s\sigma, t\sigma)$. If now C_{ref} contains an atom $Q_{\text{irr}}(s, s)$, I remove C_{ref} again. I repeat this as long as C_{irr} and C_{ref} still contain the predicate P.

To give the idea why this is satisfiability equivalent, let N' be the transformation of a clause set N which has P as its only predicate and contains the reflexivity axiom $\{\to P(x, x)\}$. If $\mathcal{I}$ is a Herbrand model of N then $\{P_{\text{ref}}(s, t), P_{\text{irr}}(s, t) \mid P(s, t) \in \mathcal{I}\}$ is a model of N' and if $\mathcal{I}$ is a Herbrand model of N' then I can construct the model $\{P(s, s) \mid P_{\text{ref}}(s, s) \in \mathcal{I}\} \cup \{P(s, t) \mid s \neq t \text{ and } P_{\text{irr}}(s, t) \in \mathcal{I}\}$ of N.

For symmetry and transitivity, which often accompany reflexivity,

$$\begin{aligned} &\to P(x, x) \\ P(x, y) &\to P(y, x) \\ P(x, y), P(y, z) &\to P(x, z) \end{aligned}$$

the result is

$$\begin{aligned} &\to P_{\text{ref}}(x, x) \\ P_{\text{irr}}(x, y) &\to P_{\text{irr}}(y, x) \\ P_{\text{ref}}(x, x) &\to P_{\text{ref}}(x, x) \\ P_{\text{irr}}(x, y), P_{\text{irr}}(y, z) &\to P_{\text{irr}}(x, z) \\ P_{\text{irr}}(x, y), P_{\text{irr}}(y, x) &\to P_{\text{ref}}(x, x) \\ P_{\text{irr}}(x, y), P_{\text{ref}}(y, y) &\to P_{\text{irr}}(x, y) \\ P_{\text{ref}}(x, x), P_{\text{irr}}(x, z) &\to P_{\text{irr}}(x, z) \\ P_{\text{ref}}(x, x), P_{\text{ref}}(x, x) &\to P_{\text{ref}}(x, x) \end{aligned}$$

However, after deleting redundant clauses, I am conveniently left with just

$$\begin{aligned} &\to P_{\text{ref}}(x, x) \\ P_{\text{irr}}(x, y) &\to P_{\text{irr}}(y, x) \\ P_{\text{irr}}(x, y), P_{\text{irr}}(y, z) &\to P_{\text{irr}}(x, z) \end{aligned}$$

which are no longer trivialized by the linear approximation $\rightarrow P(x, x')$ of the reflexivity axiom. Further, consider the satisfiable clause set N

$$\begin{aligned} &\rightarrow P(x, x) \\ P(f(x), f(y)) &\rightarrow P(x, y) \\ P(f(x), c) &\rightarrow \end{aligned}$$

which has only infinite models [9]. As with the first example, its approximation is unsatisfiable no matter the refinement. However, after tagging the reflexive predicate P and deleting subsumed clauses, the resulting tagged set N'

$$\begin{aligned} &\rightarrow P_{\text{ref}}(x, x) \\ P_{\text{irr}}(f(x), f(y)) &\rightarrow P_{\text{irr}}(x, y) \\ P_{\text{irr}}(f(x), c) &\rightarrow \end{aligned}$$

and its approximation are both immediately saturated. With this the approximation-refinement can show the satisfiability of N' and thereby also the satisfiability of the satisfiability equivalent N. However, note that while N has only infinite models, N' does have finite models. Hence, this does not show that the MSL fragment does not have the finite model property.

Lastly this method is not restricted to just binary predicates, but can be applied to any functions and predicates with arity larger than one.

7.2 SPASS-AR Details

SPASS, the solver at the heart of the implementation of FO-AR, required almost no changes to decide the MSL fragment aside from changing the selection function. The lion's share of the work was implementing straight dismatching constraints to turn SPASS into a decision procedure for the MSL(SDC) fragment. Nevertheless, I also tried to implemented additional improvements that take advantage of the approximation into MSL clauses.

7.2.1 Constraint Implementation

The implementation of constraints and the corresponding operations follows in a straight forward way from their descriptions in Section 6.1. Though, I did not implement most of the proposed optimizations because the impact of constraints on the total run-time during experiments was quite small already.

From the perspective of the solver, constraints mostly act like a black box that given a constraint either returns it normalized and minimized or that the constraint is unsolvable. Therefore, extending SPASS with constraints required few changes to the existing code of the solvers main loop. The largest difficulty was adapting subsumption and condensation.

For example, a standard clause C subsumes clause D if there exists a matching substitution σ such that $C\sigma \subsetneq D$. Constraint clauses $(C; \pi_1)$ and $(D; \pi_2)$ additionally require that $\mathcal{D}^{\mathcal{V}}(\pi_2) \subseteq \mathcal{D}^{\mathcal{V}}(\pi_1\sigma)$ (see Definition 6.1.26). However, since SPASS implements σ by using bindings, I first needed to adapt the existing subsumption and condensation code to produce the otherwise implicit σ. Furthermore, in the general case, there can be multiple matching substitutions σ. If the constraint check for one found σ fails, the function needs to be able to continue searching for possible other matching substitutions.

7.2.2 Unique Shallow Clauses

As a result of the Shallow transformation, an approximation often contains many clauses of the form

$$P_1(x_1), \ldots, P_n(x_n) \to S(f(y_1, \ldots, y_m))$$

where $\{x_1, \ldots, x_n\} \subseteq \{y_1, \ldots, y_m\}$ and S is the predicate introduced by a Shallow transformation step. By construction, such a clause is the only one where the predicate S appears in a positive literal. Furthermore, because $S(f(y_1, \ldots, y_m))$ is strictly maximal and none of the $P_i(x_i)$ are selected, this clause remains the only clause with a positive occurrence of S throughout the run of the solver. I therefore call such clauses *unique shallow* clauses. Consequently, a clause

$$S(t), \Gamma \to \Delta$$

where $S(t)$ is selected, is only ever resolved once with the corresponding unique shallow clause.

I make use of this fact by collecting all unique shallow clauses during approximation and flagging their predicates. Whenever in a derived clause a flagged literal can be selected, I immediately try to resolve the clause with the corresponding unique shallow clause. If successful, I replace the clause with the resolvent and repeat until no longer applicable. Otherwise, if the clauses cannot be resolved, $S(t), \Gamma \to \Delta$ is redundant and can therefore be deleted. This leads to an improvement in the search-space as there are fewer derived clauses.

7.2.3 Selection and Splitting

The selection strategy applied by the solver (Definition 5.1.1) and the conditions on MSL clauses together enforce that any clause without a selected negative literal has the form

$$\Gamma_1, \ldots, \Gamma_n \to E_1, \ldots, E_n, P_1(y_1), \ldots, P_m(y_m)$$

where the positive literals are variable disjoint, the E_i have the form $P(f(x_1, \ldots, x_l))$ and each Γ_i consists only of literals $Q(x)$ with $x \in \text{vars}(E_i)$. This has the consequence that all positive literals are maximal except for literals with a constant as its argument. While this almost always allows me to skip computing the maximal literals, it also means that these clauses are resolved on each positive literal deriving many unnecessary symmetric clauses.

For example, consider the clauses

$$S(x), R(y) \rightarrow P(f(x)), Q(g(y))$$
$$P(f(s)) \rightarrow \Delta_1 \text{ and } Q(g(t)) \rightarrow \Delta_2.$$

The two resolution inferences are

$$S(s), R(y) \rightarrow \Delta_1, Q(g(y))$$
$$S(x), R(t) \rightarrow P(f(x)), \Delta_2$$

and in a second step both produce the same clause

$$S(s), R(t) \rightarrow \Delta_1, \Delta_2.$$

Because the two parts of $S(x), R(y) \rightarrow P(f(x)), Q(g(y))$ for x and y are disjoint, the order of inferences on either side is irrelevant.

The solution is splitting (see Section 3.4). The clauses without selected literals can be split into individual Horn clauses

$$\Gamma_1 \rightarrow E_1 \;\; \dots \;\; \Gamma_n \rightarrow E_n$$
$$\rightarrow P_1(y_1) \;\; \dots \;\; \rightarrow P_m(y_m).$$

SPASS itself already has a built-in splitting feature [46]. Alternatively, splitting can be emulated by renaming a splittable non-Horn clause $\Gamma_E, \Gamma \rightarrow E, \Delta$ into $\Gamma_E \rightarrow E, P$ and $P, \Gamma \rightarrow \Delta$ where P is a fresh propositional predicate and the atom-ordering is extended such that P is minimal. Then, with P selected, $P, \Gamma \rightarrow \Delta$ is blocked until $\rightarrow P$ is derived. Lastly, AVATAR [44] presents yet another effective approach to splitting where splittable components of clauses are encoded as propositions and then solved by a SAT prover.

The prototype implementation does not support either of these features. For one, testing and determining which of these approaches is the best for MSL clause sets is outside the scope of this thesis. Furthermore, each approach to splitting changes the structure of the resulting resolution refutation and in neither case is there an obvious way of adjusting the lifting to these changes.

7.3 Lifting Details

As mentioned before, the motivation for using conflicting cores is to avoid unnecessary complexity in the theory of the framework while sacrificing the efficiency of the method. My first prototype actually implemented the lifting as described using the conflicting cores. However, the loss of efficiency cannot be afforded in practice. Therefore, I instead implemented a different more practical method that prioritizes speed over simplicity.

7.3.1 DAG Lifting

The first and most necessary difference between theory and the actual implementation is that I lift the resolution proof directly instead of creating and lifting a conflicting core. The simple reason is that a conflicting core can be exponentially larger than its corresponding resolution proof which is stored compactly as a directed acyclic graph (DAG). Note for the following that in a resolution refutation tree, the leaves are at the top while the root is at the bottom.

The general idea is to recursively traverse the proof DAG down from the approximation clauses in the leaves to the root and replace each node with a corresponding clause derived from the original problem. If this fails at some node, I can either immediately identify the conflict or check which parent node contains the conflict. In the latter case, I move back up the DAG towards the leaves looking for the conflict.

The lifting is done at each node in two steps: first the linear lifting, then the shallow lifting on top. For clarity however, I will explain each lifting separately.

Linear DAG Lifting

Assume the original clause set N is already monadic and shallow such that its approximation N_A is created using only Linear transformations.

Each leaf in the resolution proof is an approximated clause $(C;\pi)$ in N_A. I revert each Linear transformation that was applied to create $(C;\pi)$ by substituting the fresh variable with the original variable. Specifically, if x is replaced with x' at some position, then one lifting step of $(C;\pi)$ is $(C;\pi)\{x' \mapsto x\}$. Note, however, that duplicated negative literals are not removed by the lifting. The end result is a clause $(C_L;\pi_L)$ which is satisfiability equivalent to the ancestor clause of $(C;\pi)$ in N.

For the internal nodes of the proof DAG, I will here assume that there are only resolution inferences in the proof. The method is analogous for factorization inferences and clause reductions, e.g. condensation.

Let $(R;\pi)$ be the clause at the current node, $(C;\pi^C)$ and $(D;\pi^D)$ be the clauses at its left and right parent nodes, and $(C_L;\pi^C_L)$ and $(D_L;\pi^D_L)$ are their respective linear liftings. First, I try to repeat the resolution inference on $(C;\pi^C)$ and $(D;\pi^D)$ but instead using $(C_L;\pi^C_L)$ and $(D_L;\pi^D_L)$. If this succeeds, their resolvent $(R_L;\pi_L)$ is the linear lifting of $(R;\pi)$.

Otherwise, at this point lifting failed and I climb back up the DAG to find the leaf containing the linear conflict. Starting with the clause $(R;\pi)$ itself of the node where the lifting failed, the climbing algorithm receives at each parent node $(R'';\pi'')$ a clause $(R';\pi')$ which is an instance of $(R'';\pi'')$.

First, I determine the two substitutions σ_C and σ_D such that the resolvent of $(C;\pi^C)\sigma_C$ and $(D;\pi^D)\sigma_D$ is exactly $(R';\pi')$, i.e., a resolution inference with the identity as the most general unifier and without a variable renaming. In essence, I apply the resolution inference backwards to determine the exact instances of the parents $(C;\pi^C)$ and $(D;\pi^D)$ that are actually used in the resolution refutation.

Next, I check whether the liftings $(C_L;\pi^C_L)$ and $(D_L;\pi^D_L)$ have common ground instances with the respective clauses $(C\sigma_C;\pi')$ and $(D\sigma_D;\pi')$. If this is not the case for either, that means there is a lift conflict on that particular parent branch. The algorithm then continues climbing with $(C\sigma_C;\pi')$ or $(D\sigma_D;\pi')$ on the respective parent node.

However, it is possible that both have common ground instances. In that case, lifting failed because none of the left and right common instances can be resolved. There, I compute the most general unifier σ of C_L and $C\sigma_C$ and give $(D\sigma_D;\pi')\sigma$ as input to the climbing algorithm for the right branch. This restricts the right branch to the instances used by the left branch and forces a conflict.

Once a leaf is reached, I have a clause $(C';\pi')$ which is an instance of the approximation clause $(C;\pi)$ but has no common instances with its lifting $(C_L;\pi_L^C)$. I again revert step by step the Linear transformations on $(C;\pi)$ until I reach the transformation step where the instantiations t and t' for x and x' in $(C';\pi')$ are not unifiable without making the constraint π' unsolvable. The variable x, the two instantiations t and t', and the final constraint π' constitute the lift conflict and are used to refine the original ancestor clause of $(C;\pi)$.

Consider as an example the clause set

$$\begin{array}{rcl} & \rightarrow & P(x,x) \\ P(a,y) & \rightarrow & Q(y,b) \\ Q(b,x) & \rightarrow & \end{array}$$

with the linear approximation

$$\begin{array}{rcl} & \rightarrow & P(x,x') \\ P(a,y) & \rightarrow & Q(y,b) \\ Q(b,x) & \rightarrow & \end{array}$$

A possible resolution refutation tree is

$$\dfrac{\dfrac{\rightarrow P(x,x') \quad P(a,y) \rightarrow Q(y,b)}{\rightarrow Q(x',b)} \qquad Q(b,x) \rightarrow}{\square}$$

The lifting is able to lift $\rightarrow Q(x',b)$ with the lifting clause $\rightarrow Q(a,b)$.

$$\dfrac{\dfrac{\rightarrow P(x,x) \quad P(a,y) \rightarrow Q(y,b)}{\rightarrow Q(a,b)} \qquad Q(b,x) \rightarrow}{}$$

However, lifting fails because $Q(a,b)$ and $Q(b,x)$ are not unifiable. For the comparison, the relevant instantiations in the refutation of the approximation are

$$\dfrac{\dfrac{\rightarrow P(a,b) \quad P(a,b) \rightarrow Q(b,b)}{\rightarrow Q(b,b)} \qquad Q(b,b) \rightarrow}{\square}$$

Because $\rightarrow Q(b, b)$ and $\rightarrow P(a, b)$ have no common instances with $\rightarrow Q(a, b)$ and $\rightarrow P(x, x)$ respectively while $Q(b, b) \rightarrow$ and $P(a, b) \rightarrow Q(b, b)$ are instances of $Q(b, x) \rightarrow$ and $P(a, y) \rightarrow Q(y, b)$, I can trace the conflict to the leaf clause $\rightarrow P(x, x')$ with the lift-conflict clause $\rightarrow P(a, b)$.

Shallow DAG Lifting

The shallow lifting follows the same idea as the linear lifting. This time, assume the original clause set N is already monadic and linear such that its approximation N_A is created using only Shallow transformations.

Unlike the Linear transformation, lifting a Shallow transformation cannot be done locally on each leaf. Reverting a Shallow transformation would require combining $(C; \pi)$ with its left or right counterparts and thereby drastically changing the shape of the resolution proof DAG. Instead, I replace the Shallow transformation steps with general Shallow transformation (Section 3.3) by adding the shared variables as new arguments to the shallow predicate S. Specifically, when $\Gamma \rightarrow E[s]_p, \Delta$ is approximated by $S(x), \Gamma_l \rightarrow E[p/x], \Delta_l$ and $\Gamma_r \rightarrow S(s), \Delta_r$, I store the shared variables $\{y_1, \ldots, y_n\}$ defined by the intersection $\text{vars}(\Gamma_l, \Delta_l, E[p/x]) \cap \text{vars}(\Gamma_r, \Delta_r, S[s])$ such that they can be retrieved using the fresh predicate S as a key. Then during lifting, I replace any literal $S(t)$ in $(C; \pi)$ with $S(t, y_1, \ldots, y_n)$ to create the shallow lifting. Since the general Shallow transformation is satisfiability equivalent, if the lifted clauses create a valid resolution proof, then the original clause set N is also proven unsatisfiable.

For the internal nodes of the proof DAG, I again only describe the case for resolution inferences which is for the most part analogous to the linear case. The difference lies in how the climbing algorithm finds the conflict. While the linear case always climbs back to a leaf, a single approximation clause can never cause a shallow lift-conflict. As follows from the shallow lifting Lemma 6.4.5, a shallow lift-conflict is always a resolvent of the left and right approximation clauses of a Shallow transformation.

Therefore, the lifting had to have failed because at some node two shallow literals $\neg S(t, t_1, \ldots, t_n)$ and $S(s, s_1, \ldots, s_n)$ in the respective parent liftings could not be unified. This node can be found, as in the linear case, by climbing up the refutation until there is no conflict in any parent node. Since each shallow predicate S is unique to its respective Shallow transformation, this reveals the responsible ancestor clause that needs to be refined. Then, I sequentially unify each t_i and s_i until I find the first pair that does not allow unification. Those t_j and s_j are the conflict instances and the respective y_j is the conflict variable.

Consider as an example the clause set

$$\begin{array}{rcl} & \rightarrow & P(x, f(x)) \\ P(x, x) & \rightarrow & \end{array}$$

with the shallow approximation

$$\begin{array}{rcl} S(x') & \rightarrow & P(x,x') \\ & \rightarrow & S(f(x)) \\ P(x,x) & \rightarrow & \end{array}$$

A possible resolution refutation tree is

$$\dfrac{\rightarrow S(f(x)) \qquad \dfrac{S(x') \rightarrow P(x,x') \quad P(x,x) \rightarrow}{S(x') \rightarrow}}{\square}$$

For the lifting, I first replace each $S(t)$ with $S(t,x)$ and then repeat the resolution steps which stops at the last step

$$\dfrac{\rightarrow S(f(x),x) \qquad \dfrac{S(x',x) \rightarrow P(x,x') \quad P(x,x) \rightarrow}{S(x,x) \rightarrow}}{}$$

because $S(x,x)$ and $S(f(x),x)$ are not unifiable. The relevant instantiations in the approximation proof are

$$\dfrac{\rightarrow S(f(x)) \qquad \dfrac{S(f(x)) \rightarrow P(f(x),f(x)) \quad P(f(x),f(x)) \rightarrow}{S(f(x)) \rightarrow}}{\square}$$

After unifying the relevant instantiations with the respective shallow liftings, I get $\rightarrow S(f(x),x)$ and $S(f(x),f(x)) \rightarrow$. Since both are valid, it tells me that the approximation step that created the predicate S has a lift-conflict on the shared variable x with the conflicting instantiations v and $f(v)$.

Relaxed Shallow Lifting

An additional advantage of lifting the DAG instead of a conflicting core is the added precision of the shallow lifting. Conflicting cores are an abstraction of resolution refutations, where I remove the refutations DAG structure to focus instead on just its leaf set. Due to this abstraction, I lose information that could improve shallow lifting. Consider N and its Shallow transformation N':

$$\begin{array}{rcl} & & \\ P(x,z),Q(y,z) & \rightarrow & R(x,f(y)) \\ & \rightarrow & P(a,a) \\ & \rightarrow & P(a,b) \\ & \rightarrow & Q(b,a),Q(b,b) \\ R(a,f(b)) & \rightarrow & \end{array} \quad \Rightarrow_{\mathrm{SH}} \quad \begin{array}{rcl} S(y),P(x,z) & \rightarrow & R(x,y) \\ Q(y,z) & \rightarrow & S(f(y)) \\ & \rightarrow & P(a,a) \\ & \rightarrow & P(a,b) \\ & \rightarrow & Q(b,a),Q(b,b) \\ R(a,f(b)) & \rightarrow & \end{array}$$

There is a ground refutation where the resolutions

$$\dfrac{Q(b,a) \rightarrow S(f(b)) \qquad S(f(b)),P(a,a) \rightarrow R(a,f(b))}{P(a,a),Q(b,a) \rightarrow R(a,f(b))}$$

and
$$\frac{Q(b,b) \to S(f(b)) \qquad S(f(b)), P(a,b) \to R(a,f(b))}{P(a,b), Q(b,b) \to R(a,f(b))}$$
are the only resolutions on clauses with S-atoms. The corresponding conflicting core $N'^{\perp}$ of N' is then
$$\begin{array}{rcl}
 & \to & P(a,a) \\
 & \to & P(a,b) \\
 & \to & Q(b,a), Q(b,b) \\
S(f(b)), P(a,a) & \to & R(a,f(b)) \\
Q(b,a) & \to & S(f(b)) \\
S(f(b)), P(a,b) & \to & R(a,f(b)) \\
Q(b,b) & \to & S(f(b)) \\
R(a,f(b)) & \to &
\end{array}$$

Lifting the core (Lemma 6.4.5) fails because the resolvents
$$\frac{Q(b,b) \to S(f(b)) \qquad S(f(b)), P(a,a) \to R(a,f(b))}{P(a,a), Q(b,b) \to R(a,f(b))}$$
$$\frac{Q(b,a) \to S(f(b)) \qquad S(f(b)), P(a,b) \to R(a,f(b))}{P(a,b), Q(b,a) \to R(a,f(b))}$$
are not instances of $P(x,z), Q(y,z) \to R(x,f(y))$.

However, lifting the resolution proof itself succeeds because the proof only contains the two liftable resolution steps on S-atoms.

7.3.2 Dynamic Approximation Assignment

A possible further improvement, though I have not implemented it, is to allow reassigning conflict clauses to a different ancestor clause if lifting fails.

Consider the linear conflict where $(C';\pi')$ is an instance of the approximation clause $(C;\pi)$ but is not an instance of its lifting $(C_L;\pi_L)$. That $(C;\pi)$ is the instantiated clause of $(C';\pi')$ was determined by the resolution proof. However, $(C';\pi')$ can be an instance of more than one clause in the approximation. For example, a conflict clause $\to P(a), Q(b)$ can be an instance of both $\to P(a), Q(x)$ and $\to P(x), Q(b)$. For the proof, it is irrelevant which is the instantiated clause of $\to P(a), Q(b)$.

In such a case, lifting could fail with one instantiated clause, but succeed with another. Therefore, I could relax the conflict check to look whether there are alternative approximation clause that could allow $(C';\pi')$ to be lifted. The unsolved problem, preventing implementation at this point, is how to efficiently find these alternative instantiated clauses in the approximation.

7.4 Refinement Details

The refinement, which is, as the name implies, the last phase of the approximation-refinement loop, has its own technicalities and possible improvements. The main work in the refinement, however, is the extraction of the straight term used to generate the refined clauses.

7.4.1 The Lift-Conflict Selection

A lift conflict, as I have previously described it, consists of an original clause $(C;\pi)$, a variable x in C and two terms t and t' that are not unifiable under a constraint π'. However, the refinement requires an appropriate straight term s to create the refinements $(C;\pi)\{x \mapsto s\}$ and $(C;\pi \wedge x \neq s)$.

The most reliable method is to first find the minimal solution δ of π', i.e. where $x\delta \leq x\delta'$ for every variable x and solution δ' of π'. The resulting ground terms $t\delta$ and $t'\delta$ are by construction not equal and therefore, there is a position p where the top-symbol of $t|_p$ and $t'|_p$ differ while they are the same for every position above p. The function symbols on the path from the root of $t\delta$ up to and including p describe the required straight term s that *differentiates* $t\delta$ and $t'\delta$, i.e., $t\delta$ is an instance of s while s and $t'\delta$ have no common ground instances. In the worst case, however, this refinement only prevents one unliftable ground resolution proof from reoccurring in the following loops.

As shown in the examples in Section 6.5.1, there are cases where the chosen s is more or less effective at removing unliftable ground proofs. Furthermore, there can be several straight terms s that distinguish the same t and t'. For example, for $g(f(a),a)$ and $g(f(b),b)$, both $g(f(a),v)$ and $g(v,a)$ are feasible for refinement but I prefer the shorter second term. Further, for $g(f(f(a)),x)$ and $g(f(f(b)),f(x))$, I use $g(f(f(a)),v)$ rather than $g(v,f(w))$ because the latter is derived from the occurrence conflict when trying to unify x with $f(x)$ in the second argument. For this purpose, I use a modified unification algorithm that searches for an 'optimal' straight refinement term (see Figure 7.1).

The rules are applied on triples $(\mathrm{G},\mathrm{S},\pi)$ where S is the result set containing valid straight terms and G is the set of unifications $(t \doteq t', s, p)$ which track previous decompositions with the straight term s and position p. Given two terms t and t' that are not unifiable under the constraint π, the algorithm starts with the triple $(\{(t \doteq t', x, \varepsilon)\}, \{\}, \pi)$.

The generalized unification terminates for the same reason as unification, i.e., The Delete, Decompose, Conflict, Swap, and Constraint rules each lower the multiset of term depths of the left-hand sides of the equations $t \doteq t'$ in G, while the Eliminate rule permanently lowers the number of free variables in G.

Each rule preserves the invariant that if σ is the total substitution applied on G by the Eliminate rule and t and t' are the starting terms, then for any $(u \doteq u', s, p) \in G$, $t\sigma$ and $t'\sigma$ are instances of either $s[p/u]$ or $s[p/u']$, respectively, and any term $s \in S$ is a straight term that differentiates $t\sigma$ and $t'\sigma$.

While standard unification tracks the substitutions and fails upon reaching a conflict, this version instead stores for each found conflict the corresponding straight term. These straight terms are generated by extending a term with hole with each application of the decompose rule and completing them upon reaching a conflict.

Delete $(\mathrm{G} \,\dot\cup\, \{(t \doteq t, s, p)\}, \mathrm{S}, \pi) \;\Rightarrow\; (\mathrm{G}, \mathrm{S}, \pi)$

Decompose $(\mathrm{G} \,\dot\cup\, \{(f(t_1, \ldots, t_n) \doteq f(t'_1, \ldots, t'_n), s, p)\}, \mathrm{S}, \pi) \;\Rightarrow$
$(\mathrm{G} \,\dot\cup\, \{(t_i \doteq t'_i, (s[p/f(v_1, \ldots, v_n)]), p.i) \mid 1 \leq i \leq n\}, \mathrm{S}, \pi)$
where $v_1, \ldots, v_n$ are fresh variables in s.

Conflict $(\mathrm{G} \,\dot\cup\, \{(f(\bar{t}) \doteq g(\bar{s}), s, p)\}, \mathrm{S}, \pi) \;\Rightarrow$
$(\mathrm{G}, \mathrm{S} \cup \{s[p/f(\overline{v})], s[p/g(\overline{w})]\}, \pi)$
where $\overline{v}$ and $\overline{w}$ are fresh variables in s.

Swap $(\mathrm{G} \,\dot\cup\, \{(f(\bar{t}) \doteq x, s, p)\}, \mathrm{S}, \pi) \;\Rightarrow$
$(\mathrm{G} \,\dot\cup\, \{(x \doteq f(\bar{t}), s, p)\}, \mathrm{S}, \pi)$

Constraint $(\mathrm{G} \,\dot\cup\, \{(x \doteq t, s, p)\}, \mathrm{S}, \pi) \;\Rightarrow$
$(\mathrm{G}, \mathrm{S} \cup \{s[p/s'\rho] \mid x \neq s' \in \pi \text{ and } t \neq s'\!\downarrow = \bot\}, \pi)$
where s and $s'\rho$ are variable disjoint and $\pi\{x \mapsto t\}\!\downarrow = \bot$.

Eliminate $(\mathrm{G} \,\dot\cup\, \{(x \doteq t, s, p)\}, \mathrm{S}, \pi) \;\Rightarrow\; (\mathrm{G}\{x \mapsto t\}, \mathrm{S}, \pi\{x \mapsto t\})$
where $x \notin \mathrm{vars}(t)$, $\pi\{x \mapsto t\}$ is solvable, and
$(t \doteq t', s, p)\sigma = (t\sigma \doteq t'\sigma, s, p)$.

Figure 7.1: Generalized Unification with straight dismatching constraints

The added constraint rule catches a new case of conflict that can occur under the presence of constraints. For example, $f(x)$ and $f(f(a))$ are not unifiable under the constraint $x \neq f(v)$ and I generate $f(f(x))$ as the distinguishing straight term using the constraint rule. Additionally, each rule is given a priority defined by the order they are listed here, i.e., they are only applied if all previous rules are exhausted.

Once exhausted, one straight term is chosen from S depending on criteria such as depth, size, and the number of eliminate steps used. However, note that the rules are not complete. If the algorithm stops without result, I fall back on generating the straight term from a minimal ground core instead. This is because there are no cases for when x occurs in t or when $\pi\{x \mapsto t\}$ is unsolvable. The former case is impossible to refine completely as shown in Section 6.5.1 and therefore, I try to avoid whenever possible. The latter case could be refined but I consider it too complex and has proven too rare in experiments to be worth implementing.

7.4.2 Soft Reset

At the end of the approximation-refinement loop, the theoretical framework suggests a full reset of the solver and then restarting with the refined clause set. However, this is quite wasteful as only one clause of the original clause set was removed.

Therefore, I have implemented what I call a soft reset. Since I still have the

DAG structure of the resolution proof, I can use it to find and delete only those clauses that were derived from the refined original clause. All remaining clauses, would be derived again and can therefore be kept in the 'Use' set of SPASS. I can even keep clauses in the 'Worked-Off' set by putting redundant clauses back into the 'Use' set that were made redundant by a now deleted clause. This preserves the invariant of the 'Worked-Off' set that all inferences between clauses in it are either redundant, worked off or in use [46].

7.4.3 Matching Constraints

Lemma 6.1.44 in Section 6.1.3 mentioned a possible preprocessing when using matching constraints in addition to dismatching constraints. This lemma could also be applied to slightly improve the refinement. Instead of replacing a clause $(C;\pi)$ with $(C;\pi)\{x \mapsto s\}$ and $(C;\pi \wedge x \neq s)$, I can use $(C;\pi \wedge x = s)$ and $(C;\pi \wedge x \neq s)$. This has the advantage that the instantiated term s does not require an extraction through Shallow transformation.

However, this small benefit did not seem worth the additional effort implementing matching constraints would have required.

7.5 Experiments

In the following I discuss several first-order clause structures for which FO-AR implemented in SPASS-AR immediately decides satisfiability but superposition and instantiation-based methods fail. I argue both according to the respective calculi and state-of-the-art implementations, in particular SPASS 3.9 [47], Vampire 4.1 [44], for ordered-resolution/superposition, iProver 2.5 [26] an implementation of Inst-Gen [27], and Darwin v1.4.5 [4] an implementation of the model evolution calculus [5]. All experiments were run on a 64-Bit Linux computer (Xeon(R) E5-2680, 2.70GHz, 256GB main memory). For Vampire and Darwin I chose the respective CASC setting, for iProver I set the schedule to "sat" and SPASS, SPASS-AR were used in default mode. Please note that Vampire and iProver are portfolio solvers including implementations of several different calculi whereas SPASS, SPASS-AR, and Darwin only implement superposition, FO-AR, and model evolution, respectively.

For the first example

$$\begin{aligned} P(x,y) &\rightarrow P(x,z), P(z,y) \\ &\rightarrow P(a,a) \end{aligned}$$

and second example,

$$\begin{aligned} &\rightarrow Q(x,x) \\ Q(v,w), P(x,y) &\rightarrow P(x,v), P(w,y) \\ &\rightarrow P(a,a) \end{aligned}$$

the superposition calculus produces independently of the selection strategy and ordering an infinite number of clauses of form

$$\rightarrow P(a, z_1),\ P(z_1, z_2),\ \ldots,\ P(z_n, a).$$

Using the Linear transformation, however, FO-AR replaces

$$\begin{aligned} P(x,y) &\rightarrow P(x,z), P(z,y) \\ &\rightarrow Q(x,x) \end{aligned}$$

respectively with

$$\begin{aligned} P(x,y) &\rightarrow P(x,z), P(z',y) \\ &\rightarrow Q(x,x') \end{aligned}$$

Consequently, ordered resolution with selection derives $\rightarrow P(a,z_1), P(z_2,a)$ which subsumes any further inferences $\rightarrow P(a,z_1), P(z_2,z_3), P(z_4,a)$. Hence, saturation of the approximation terminates immediately. Both examples belong to the Bernays-Schönfinkel fragment, so model evolution (Darwin) and Inst-Gen (iProver) can decide them as well. Note that the concrete behaviour of superposition is not limited to the above examples but potentially occurs whenever there are variable chains in clauses.

On the third problem

$$\begin{aligned} P(x,y) &\rightarrow P(g(x),z) \\ &\rightarrow P(a,a) \end{aligned}$$

superposition derives all clauses of the form $\rightarrow P(g(\ldots g(a)\ldots), z)$. With a shallow approximation of $P(x,y) \rightarrow P(g(x),z)$ into $S(v) \rightarrow P(v,z)$ and $P(x,y) \rightarrow S(g(x))$, FO-AR (SPASS-AR) terminates after deriving $\rightarrow S(g(a))$ and $S(x) \rightarrow S(g(x))$. Again, model evolution (Darwin) and Inst-Gen (iProver) can also solve this example.

The next example

$$\begin{aligned} &\rightarrow P(a) \\ P(f(a)) &\rightarrow \\ P(f(f(x))) &\rightarrow P(x) \\ P(x) &\rightarrow P(f(f(x))) \end{aligned}$$

is already saturated under superposition. The same almost holds true for FO-AR, where $P(x) \rightarrow P(f(f(x)))$ is replaced by $S(x) \rightarrow P(f(x))$ and $P(x) \rightarrow S(f(x))$. Then ordered resolution terminates after inferring $S(a) \rightarrow$ and $S(f(x)) \rightarrow P(x)$.

The Inst-Gen and model evolution calculi, however, fail. In either, a satisfying model is represented by a finite set of literals, i.e, a model of the propositional approximation for Inst-Gen and the trail of literals in case of model evolution. Therefore, there necessarily exists a literal $P(f^n(x))$ or $\neg P(f^n(x))$ with a maximal n in these models. This contradicts the actual model where either $P(f^n(a))$

or $P(f^n(f(a)))$ is true. However, iProver can solve this problem using its built-in ordered resolution solver whereas Darwin does not terminate on this problem.

Lastly consider an example of the form

$$\begin{aligned} f(x) \approx x &\rightarrow \\ f(f(x)) \approx x &\rightarrow \\ &\vdots \\ f^n(x) \approx x &\rightarrow \end{aligned}$$

which is trivially satisfiable, e.g., saturated by superposition, but any model has at least $n+1$ domain elements. Therefore, adding these clauses to any satisfiable clause set containing f forces calculi that explicitly compute finite models to consider at least $n + 1$ elements. The performance of final model finders typically degrades in the number of different domain elements to be considered.

Combining each of these examples into one problem is then solvable by neither superposition, Inst-Gen, or model evolution and not practically solvable with increasing n via testing finite models.

Example 7.5.1.

$$\begin{aligned} P(x,y) &\rightarrow P(x,z), P(z,y) \\ &\rightarrow P(a,a) \\ P(f(a),y) &\rightarrow \\ P(f(f(x)),y) &\rightarrow P(x,y) \\ P(x,y) &\rightarrow P(f(f(x)),y) \\ f(x) \approx x &\rightarrow \\ f(f(x)) \approx x &\rightarrow \\ &\vdots \\ f^n(x) \approx x &\rightarrow \end{aligned}$$

Testing Example 7.5.1 with $n = 20$ against SPASS, Vampire, iProver, and Darwin each for more than one hour and, when applicable, using specialized satisfiability schedules without success. Only SPASS-AR solved it in less than one second (see Appendix A.1).

Another such example is Example 7.5.2.

Example 7.5.2.

$$
\begin{aligned}
&\rightarrow P(a,a,a)\\
P(x,y',z'),P(x',y,z) &\rightarrow P(f(f(x)),y,z)\\
P(x',y,z'),P(x,y',z) &\rightarrow P(x,f(f(f(y))),z)\\
P(x',y',z),P(x,y,z') &\rightarrow P(x,y,f(f(f(f(f(z))))))\\
P(f(f(x)),y,z) &\rightarrow P(x,y,z)\\
P(x,f(f(f(y))),z) &\rightarrow P(x,y,z)\\
P(x,y,f(f(f(f(f(z)))))) &\rightarrow P(x,y,z)\\
P(f(a),y,z) &\rightarrow\\
P(x,f(a),z) &\rightarrow\\
P(x,f(f(a)),z) &\rightarrow\\
P(x,y,f(a)) &\rightarrow\\
P(x,y,f(f(a))) &\rightarrow\\
P(x,y,f(f(f(a)))) &\rightarrow\\
P(x,y,f(f(f(f(a))))) &\rightarrow
\end{aligned}
$$

SPASS-AR saturates its approximation in under one second (see Appendix A.2) while SPASS, Vampire, iProver, and Darwin each cannot solve it within a test-run of over one hour. Note that while finite models exist, they require at least 30 domain elements which is too many for the finite model finders build into iProver and Vampire.

Additionally, I have tested SPASS-AR on the non-equality problems in the TPTP version 7.0.0 [40]. The problems were run for one hour each using a cluster with 64-Bit Linux servers (Intel Xeon E5620 @ 2.40GHz, 6x 8GiB DDR3 1067 MHz ECC memory). Overall, SPASS-AR solved 2277 of the 4130 problems (55%), 1803 of the 3075 unsatisfiable problems (59%), and 484 of the 927 satisfiable problems (52%). A detailed breakdown of the result for individual problem classes is presented in Table 7.1. On average SPASS-AR refines a solved problem 8.5 times, 2.9 times for satisfiable and 10.1 times for unsatisfiable problems. 1403 problems (62%) are solved without requiring a refinement, 409 (84%) are satisfiable and 994 (55%) unsatisfiable. For solved problems that are refined there are on average 21.3 refinements, 17.3 refinements for satisfiable and 21.6 refinements for unsatisfiable problems, and each with a median of 10.

For comparison, under the same conditions with default schedules, SPASS v3.8 solved 2852 problems, Vampire v4.1 solved 3152, and iProver v2.7 solved 3565. SPASS-AR solved 133 problems that SPASS did not solve, 116 problems that Vampire did not solve, and 16 problems that iProver did not solve. When restricted to using only the Inst-Gen calculus, iProver v2.7 solved 3433 problems with 19 unsolved problems solved by SPASS-AR.

After the preprocessing (see Section 7.1.2), 341 problems are already in the MSL fragment of which 226 are in the MSLH fragment. Of those, SPASS-AR solves 320 (94%) and 205 (91%) problems, respectively. Most of the unsolved MSL problems are actually large ground problems and therefore better suited for a propositional satisfiability solver.

Additionally, I have measured the MSL approximation distance of each problem, i.e., the number of non-satisfiability equivalent transformation steps required to approximate the problem into MSL. Table 7.2 shows that most problems are very close to the MSL fragment with about a third at distance below ten and half below twenty. Furthermore, for problems very close to MSL (distance 0-3) SPASS-AR has a high success-rate, but surprisingly also for problems with a large distance (200+). Note that 190 problems are missing from the total in Table 7.2 because SPASS-AR was unable to compute their MSL distance due to problem size.

As mentioned in Section 6.5.1, certain lift-conflicts cannot be fully refined away. For example, when the conflicting instantiations, t and t', cannot be unified because t occurs as a subterm in t'. Of the 2277 solved problems, only 198 (9%) contain such an occurrence-conflict, of which all are unsatisfiable problems. On the other hand, 755 of the 1824 unsolved problems (41%) reach at least one occurrence-conflict before time-out; 129 out of 442 are satisfiable (29%) and 532 out of 1265 are unsatisfiable (42%).

The preprocessing introduced in Section 7.1.2 allows SPASS-AR to solve an additional 40 problems (2% of solved problems) while the preprocessing method for reflexive predicates (see Section 7.1.3) solves an additional 66 problems (3% of solved problems). However, as only three of the latter are satisfiable, this method seems ineffective at avoiding occurrence-conflicts in satisfiable problems.

Domain	All	Solved	%	Sat	Compl.	%	Unsat	Proof	%
SYN	1128	806	71%	275	166	60%	875	640	73%
LCL	564	115	20%	139	48	35%	430	67	16%
CSR	379	142	37%	4	1	25%	372	141	38%
FLD	281	58	21%	3			183	58	32%
GEO	264	172	65%	16	1	6%	249	171	69%
SWV	260	84	32%	16	16	100%	248	68	27%
NLP	206	129	63%	210	115	55%	14	14	100%
KRS	188	94	50%	85	30	35%	103	64	62%
GRP	129	69	53%	62	11	18%	77	58	75%
PUZ	95	68	72%	23	16	70%	70	52	74%
SET	86	58	67%	5	4	80%	82	54	66%
SWB	86	53	62%	56	30	54%	29	23	79%
PLA	51	10	20%	8	3	38%	43	7	16%
MGT	45	44	98%				45	44	98%
NUM	44	15	34%	5	5	100%	28	10	36%
COL	29	21	72%				29	21	72%
MSC	28	21	75%	7	6	86%	21	15	71%
TOP	26	10	38%	19	5	26%	7	5	71%
HWV	25	13	52%	6	6	100%	19	7	37%
ANA	23	8	35%				21	8	38%
SEU	21	11	52%				21	11	52%
LAT	19	11	58%				19	11	58%
SYO	18	7	39%	6	3	50%	13	4	31%
GRA	15	3	20%	13	1	8 %	2	2	100%
COM	14	10	71%				15	10	67%
ALG	13	2	15%				13	2	15%
MED	11	5	45%	2	2	100%	10	3	30%
RNG	10	1	10%				9	1	11%
AGT	8	1	12%	1	1	100%	7	0	0%
PRD	3	1	33%	2	0	0%	1	1	100%
SEV	3	2	66%	2	1	50%	1	1	100%

Table 7.1: The non-equality problems of the TPTP v.7.0.0 broken up by domain into rows and sorted in decreasing order of total size of each. The columns show total number vs. number solved and the respective percentage, additionally separate columns for satisfiable and unsatisfiable problems.

Distance	Total	Solved	%	SumT	SumT %	SumS	SumS %
0	341	320	94%	341	9%	320	14%
1	149	146	98%	490	12%	466	21%
2	147	123	84%	637	16%	589	26%
3	81	67	83%	718	18%	656	29%
4	137	98	72%	855	22%	754	33%
5	74	51	69%	929	24%	805	35%
6	93	77	83%	1022	26%	882	39%
7	66	41	62%	1088	28%	923	41%
8	145	46	32%	1233	31%	969	43%
9	40	29	73%	1273	32%	998	44%
10	274	128	47%	1547	39%	1126	50%
11-20	459	241	53%	2006	51%	1367	60%
21-30	217	88	40%	2223	56%	1455	64%
31-40	290	84	29%	2508	64%	1539	68%
41-50	187	46	25%	2689	68%	1585	70%
51-60	202	133	66%	2891	73%	1718	76%
61-70	96	26	27%	2987	76%	1744	77%
71-80	76	27	36%	3057	78%	1771	78%
81-90	30	13	43%	3084	78%	1784	78%
91-100	113	11	10%	3196	81%	1795	79%
101-200	355	76	21%	3517	89%	1871	82%
201-300	150	117	78%	3649	93%	1988	87%
301-400	239	207	87%	3860	98%	2195	97%
401+	80	78	98%	3940	100%	2273	100%

Table 7.2: The MSL Approximation Distance Table. The first block of three columns shows the total number of problems with the given distance, as well as the respective number of solved problems and their share compared to the total. The second block lists the total number of problems with distance less or equal than the given distance, as well as the percentage compared to the whole set of problems. The last block shows the analogous result for solved problems.

Chapter 8

Conclusion

In the beginning, I set out to make a first step towards using model-based guidance in first-order reasoning. Propositional solvers based on conflict-driven clause learning [6] owe part of their surprising efficiency to the factor that a learned clause is never redundant at the time it is added. This property, which first-order superposition solvers can only imitate with expensive redundancy checks, indirectly derives from the fact that an inference on the minimal false clause under the partial model operator is guaranteed to not be redundant.

For this reason, I first proved that for the approximation of first-order clause sets into the MSLH fragment, the partial model of the original clause set is a subset of the approximation's partial model. I had hoped to use this property to the end that if I find the minimal false clause in the decidable fragment of the approximation, I could draw conclusions about the minimal false clause in the original clause set and therefore choose inferences that do not require redundancy checks. While this was ultimately unsuccessful, one approach to infer the minimal false clause from the approximation eventually became the basis of the new calculus' lifting and refinement.

Therefore, I next relaxed the requirement on the approximation to the definition of over-approximation I am using now, i.e., satisfiability of the approximation implies satisfiability of the original clause set. Based on this and the lifting, I developed an approximation-refinement approach to first-order logic without equality that forms a sound and complete calculus.

However, the theory also contained obvious issues that would have made an implementation impractical. First and foremost, lifting the Horn transformations requires, similarly to splitting, exponentially many approximations such that an unsatisfiability proof of the original clause set could be lifted by combining the proofs of each approximation. Luckily, on re-inspection of the decidability proof of the MSLH fragment, it turned out that it did not directly rely on the Horn property. Therefore, the target of the approximation switched from MSLH to the new decidable MSL fragment. This way not only was the most complex and expensive part of lifting no longer necessary but the framework could now also handle non-deterministic theories much more directly.

The second practical problem of the framework was that the initial refinement by instantiation generates up to quadratically many clauses depending on the depth of the term needed for the refinement and the signature. The method of generating specific instances given in [29] by focusing on just one relevant branch gave me the idea for using this branch as a straight term.

Bundled with the suggestion to use the dismatching constraints in [1], I developed the straight dismatching constraints which instead allow a constant size refinement. From that followed a sound and complete ordered resolution calculus for first-order clauses with straight dismatching constraints and the decidability of the corresponding MSL(SDC) fragment. Conveniently, relevant operations on straight dismatching constraints, especially the emptiness check, perform in linear or linear logarithmic time. A promising replacement for the previously quadratic refinement. This is also in stark contrast to general dismatching constraints where the emptiness check is NP-hard [14].

With the two issues solved, I finally began to implement SPASS-AR, the approximation-refinement calculus using SPASS as the underlying decision procedure for MSL(SDC). While I expected inefficiencies from the first prototype, one stood out the most. My calculus describes lifting in the form of a step by step lifting of a conflicting core generated from the resolution refutation of the approximation. However, the clause set of the conflicting core can be and often is exponentially larger than the resolution refutation when represented as a directed acyclic graph. Therefore, the implementation needed to deviate from the theory by instead lifting the resolution refutation directly.

Even with this change, my prototypical implementation could not compete with systems such as iProver or Vampire on the respective CASC categories of the TPTP [41]. This is already due to the fact that they are all meanwhile portfolio solvers. For example, iProver contains an implementation of ordered resolution and Vampire an implementation of Inst-Gen. These systems, however, may benefit from FO-AR by adding it to their portfolio.

Using a short example, I showed that FO-AR is superior to superposition, instantiation-based methods on certain classes of clause sets. Of course, there are also classes of clause sets where superposition and instantiation-based methods are superior to FO-AR, e.g., for unsatisfiable clause sets where the structure of the clause set forces FO-AR to enumerate failing ground instances due to the approximation in a bottom-up way.

The DEXPTIME-completeness result for MSLH strongly suggest that both the MSLH and also the MSL and MSL(SDC) fragments have the finite model property. If MSL(SDC) has the finite model property, the finite model finding approaches are complete on MSL(SDC). The models generated by FO-AR and superposition are typically infinite. It remains an open problem, even for fragments enjoying the finite model property, e.g., the first-order monadic fragment, to design a calculus that combines explicit finite model finding with a structural representation of infinite models.

There are not many results on calculi that operate with respect to models containing positive equations. Even for fragments that are decidable with equality, such as the Bernays-Schoenfinkel-Ramsey fragment or the monadic fragment with equality, there seem currently no convincing suggestions compared to the great amount of techniques for these fragments without equality. Adding positive equations to MSL(SDC) while keeping decidability is, to the best of my current knowledge, only possible for at most linear, shallow equations $f(x_1, \ldots, x_n) \approx h(y_1, \ldots, y_n)$ [24]. However, approximation into such equations from an equational theory with nested term occurrences typically results in an almost trivial equational theory. So this does not seem to be a very promising research direction.

In summary, the contributions of this thesis are an examination of first-order over-approximations and their properties with respect to model approximation, a novel approximation-refinement approach for first-order theorem proving based on counter-example guided abstraction refinement, a decision procedure for the MSL fragment, the introduction of straight dismatching constraints with a linear-logarithmic solvability check, and a prototype implementation which incorporates the approximation-refinement, straight dismatching constraints, and the decision procedure for the MSL fragment.

Bibliography

[1] Gábor Alagi and Christoph Weidenbach. NCRL - A model building approach to the Bernays-Schönfinkel fragment. In Carsten Lutz and Silvio Ranise, editors, *Frocos (Wroclaw)*, volume 9322 of *Lecture Notes in Artificial Intelligence*, pages 69–84. Springer, 2015.

[2] Leo Bachmair and Harald Ganzinger. Resolution theorem proving. In *Handbook of Automated Reasoning (in 2 volumes)*, pages 19–99. 2001.

[3] Jos C. M. Baeten, Jan A. Bergstra, Jan Willem Klop, and W. P. Weijland. Term-rewriting systems with rule priorities. *Theor. Comput. Sci.*, 67(2&3):283–301, 1989.

[4] Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. Implementing the model evolution calculus. *International Journal on Artificial Intelligence Tools*, 15(1):21–52, 2006.

[5] Peter Baumgartner and Cesare Tinelli. The model evolution calculus. In Franz Baader, editor, *Automated Deduction - CADE-19, 19th International Conference on Automated Deduction Miami Beach, FL, USA, July 28 - August 2, 2003, Proceedings*, volume 2741 of *Lecture Notes in Computer Science*, pages 350–364. Springer, 2003.

[6] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[7] Yohan Boichut, Roméo Courbis, Pierre-Cyrille Héam, and Olga Kouchnarenko. *Rewriting Techniques and Applications: 19th International Conference, RTA 2008 Hagenberg, Austria, July 15-17, 2008 Proceedings*, chapter Finer Is Better: Abstraction Refinement for Rewriting Approximations, pages 48–62. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[8] Ahmed Bouajjani, Peter Habermehl, Adam Rogalewicz, and Tomáš Vojnar. *Static Analysis: 13th International Symposium, SAS 2006, Seoul, Korea, August 29-31, 2006. Proceedings*, chapter Abstract Regular Tree Model Checking of Complex Dynamic Data Structures, pages 52–70. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[9] Ricardo Caferra, Alexander Leitsch, and Nicolas Peltier. *Automated Model Building*, volume 31 of *Applied Logic*. Springer, 2004. Page 55.

[10] Koen Claessen. New techniques that improve MACE-style finite model finding. In *Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications*, 2003.

[11] Koen Claessen, Ann Lillieström, and Nicholas Smallbone. Sort it out with monotonicity - translating between many-sorted and unsorted first-order logic. In *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*, pages 207–221, 2011.

[12] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. *Computer Aided Verification: 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000. Proceedings*, chapter Counterexample-Guided Abstraction Refinement, pages 154–169. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[13] Hubert Comon. Disunification: A survey. In *Computational Logic - Essays in Honor of Alan Robinson*, pages 322–359, 1991.

[14] Hubert Comon and Pierre Lescanne. Equational problems and disunification. *Journal of Symbolic Computation*, 7(3/4):371–425, 1989.

[15] Satyaki Das and David L. Dill. *Formal Methods in Computer-Aided Design: 4th International Conference, FMCAD 2002 Portland, OR, USA, November 6–8, 2002 Proceedings*, chapter Counter-Example Based Predicate Discovery in Predicate Abstraction, pages 19–32. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[16] T. Fruhwirth, E. Shapiro, M. Y. Vardi, and E. Yardeni. Logic programs as types for logic programs. In *Logic in Computer Science, 1991. LICS '91., Proceedings of Sixth Annual IEEE Symposium on*, pages 300–309, July 1991.

[17] Fausto Giunchiglia and Enrico Giunchiglia. Building complex derived inference rules: A decider for the class of prenex universal-existential formulas. In *ECAI*, pages 607–609, 1988.

[18] Fausto Giunchiglia and Toby Walsh. A theory of abstraction. *Artif. Intell.*, 57(2-3):323–389, October 1992.

[19] Jean Goubault-Larrecq. Deciding $\mathcal{H}_1$ by resolution. *Information Processing Letters*, 95(3):401 – 408, 2005.

[20] Jean Goubault-Larrecq and Fabrice Parrennes. *Verification, Model Checking, and Abstract Interpretation: 6th International Conference, VMCAI 2005,*

Paris, France, January 17-19, 2005. Proceedings, chapter Cryptographic Protocol Analysis on Real C Code, pages 363–379. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[21] Julio Cesar Lopez Hernandez and Konstantin Korovin. Towards an abstraction-refinement framework for reasoning with large theories. In Thomas Eiter, David Sands, Geoff Sutcliffe, and Andrei Voronkov, editors, *IWIL@LPAR 2017 Workshop and LPAR-21 Short Presentations, Maun, Botswana, May 7-12, 2017*, volume 1 of *Kalpa Publications in Computing*. EasyChair, 2017.

[22] Jerry R. Hobbs. Granularity. In *In Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 432–435. Morgan Kaufmann, 1985.

[23] Tomasz Imielinski. Domain abstraction and limited reasoning. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'87, pages 997–1003, San Francisco, CA, USA, 1987. Morgan Kaufmann Publishers Inc.

[24] Florent Jacquemard, Christoph Meyer, and Christoph Weidenbach. Unification in extensions of shallow equational theories. In Tobias Nipkow, editor, *Rewriting Techniques and Applications, 9th International Conference, RTA-98*, volume 1379 of *LNCS*, pages 76–90. Springer, 1998.

[25] Henry Kautz and Bart Selman. *A general framework for knowledge compilation*, pages 287–300. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991.

[26] Konstantin Korovin. iprover - an instantiation-based theorem prover for first-order logic (system description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, volume 5195 of *Lecture Notes in Computer Science*, pages 292–298. Springer, 2008.

[27] Konstantin Korovin. Inst-Gen - A modular approach to instantiation-based automated reasoning. In *Programming Logics - Essays in Memory of Harald Ganzinger*, pages 239–270, 2013.

[28] Y. Lakhnech, S. Bensalem, S. Berezin, and S. Owre. *Tools and Algorithms for the Construction and Analysis of Systems: 7th International Conference, TACAS 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2–6, 2001 Proceedings*, chapter Incremental Verification by Abstraction, pages 98–112. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[29] J.-L. Lassez and K. Marriott. Explicit representation of terms defined by counter examples. *J. Autom. Reason.*, 3(3):301–317, September 1987.

[30] Allen Newell. *Human Problem Solving*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1972.

[31] Flemming Nielson, Hanne Riis Nielson, and Helmut Seidl. Normalizable horn clauses, strongly recognizable relations, and spi. In Manuel V. Hermenegildo and Germn Puebla, editors, *SAS*, volume 2477 of *Lecture Notes in Computer Science*, pages 20–35. Springer, 2002.

[32] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving sat and sat modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll(t). *Journal of the ACM*, 53:937–977, November 2006.

[33] Andreas Nonnengart and Christoph Weidenbach. Computing small clause normal forms. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 335–367. Elsevier and MIT Press, 2001.

[34] David A. Plaisted. Theorem proving with abstraction. *Artif. Intell.*, 16(1):47–108, 1981.

[35] Andreas Reuß and Helmut Seidl. *Implementation and Application of Automata: 17th International Conference, CIAA 2012, Porto, Portugal, July 17-20, 2012. Proceedings*, chapter Crossing the Syntactic Barrier: Hom-Disequalities for $\mathcal{H}_1$-Clauses, pages 301–312. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[36] Andreas Reu and Helmut Seidl. Bottom-up tree automata with term constraints. In Christian G. Fermller and Andrei Voronkov, editors, *LPAR (Yogyakarta)*, volume 6397 of *Lecture Notes in Computer Science*, pages 581–593. Springer, 2010.

[37] Earl D. Sacerdott. Planning in a hierarchy of abstraction spaces. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, IJCAI'73, pages 412–422, San Francisco, CA, USA, 1973. Morgan Kaufmann Publishers Inc.

[38] Helmut Seidl and Andreas Reuß. Extending H1-Clauses with Disequalities. *Information Processing Letters*, 111(20):1007–1013, 2011.

[39] Helmut Seidl and Andreas Reuß. *Foundations of Software Science and Computational Structures: 15th International Conference, FOSSACS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 – April 1, 2012. Proceedings*, chapter Extending $\mathcal{H}_1$ -Clauses with Path Disequalities, pages 165–179. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[40] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[41] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[42] Josh Tenenberg. Preserving consistency across abstraction mappings. In *In Proceedings of the 10th IJCAI, pages 1011–1014. International Joint Conference on Artificial Intelligence*, 1987.

[43] Andreas Teucke and Christoph Weidenbach. First-order logic theorem proving and model building via approximation and instantiation. In Carsten Lutz and Silvio Ranise, editors, *Frontiers of Combining Systems: 10th International Symposium, FroCoS 2015, Wroclaw, Poland, September 21-24, 2015, Proceedings*, pages 85–100, Cham, 2015. Springer International Publishing.

[44] Andrei Voronkov. AVATAR: the architecture for first-order theorem provers. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pages 696–710. Springer, 2014.

[45] Christoph Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In Harald Ganzinger, editor, *16th International Conference on Automated Deduction, CADE-16*, volume 1632 of *LNAI*, pages 314–328. Springer, 1999.

[46] Christoph Weidenbach. Combining superposition, sorts and splitting. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 27, pages 1965–2012. Elsevier, 2001.

[47] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. SPASS version 3.5. In Renate A. Schmidt, editor, *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings*, volume 5663 of *Lecture Notes in Computer Science*, pages 140–145. Springer, 2009.

Appendix A

Example Saturations

In the following sections, I present the saturations and TPTP versions of the two examples from Section 7.5.

A.1 Example 7.5.1

Example 7.5.1 (see Figure A.1 for the TPTP format).

$$\begin{aligned}
P(x,y) &\rightarrow P(x,z), P(z,y)\\
&\rightarrow P(a,a)\\
P(f(a),y) &\rightarrow\\
P(f(f(x)),y) &\rightarrow P(x,y)\\
P(x,y) &\rightarrow P(f(f(x)),y)\\
f(x) \approx x &\rightarrow\\
f(f(x)) \approx x &\rightarrow\\
&\vdots\\
f^n(x) \approx x &\rightarrow\\
&\rightarrow a \approx b
\end{aligned}$$

SPASS-AR generates the following approximation and saturation. Note that I have reformatted the clauses for readability. For example, I have removed the constraints, as they were all $\top$, reverted the Monadic transformation of $P(s,t)$ into $T(\mathrm{fP}(s,t))$ and reformatted the equalities $\mathrm{equal}(s,t)$ into $s \approx t$. Furthermore, redundant clauses are not listed.

$$\begin{aligned}
&34[Inp:33] & P(u,v)^{+} &\rightarrow P(u,w), P(x,v)\\
&35[Inp:25] & &\rightarrow P(a,a)^{*}\\
&36[Inp:26] & P(f(a),u)^{+} &\rightarrow
\end{aligned}$$

$$
\begin{array}{rr@{\;\rightarrow\;}l}
37[Inp:27] & P(f(f(u)),v)^{+} & P(u,v)\\
38[Inp:32] & P(u,v)^{+} & S_2(f(u))\\
39[Inp:31] & S_2(u) & S_1(f(u))^{*}\\
40[Inp:29] & S_1(u),P(v,w)^{+} & P(u,w)\\
41[Inp:6] & f(u)\approx u & \\
42[Inp:7] & f(f(u))\approx u & \\
 & & \vdots\\
57[Inp:22] & f^{n}(u)\approx u & \\
58[Inp:23] & & a\approx b\\
59[Res:35,38] & & S_2(f(a))^{*}\\
60[Res:35,40] & S_1(u) & P(u,v)^{*}\\
62[Res:60,38] & S_1(u) & S_2(f(u))^{*}\\
64[Res:39,61] & S_2(a)^{+} & \\
65[Res:39,63] & S_2(f(u))^{+} & P(u,v)\\
76[Res:70,36] & & P(a,u)^{*}
\end{array}
$$

A.2 Example 7.5.2

Example 7.5.2 (see Figure A.2 for the TPTP format)

$$
\begin{array}{r@{\;\rightarrow\;}l}
 & P(a,a,a)\\
P(x,y',z'),P(x',y,z) & P(f(f(x)),y,z)\\
P(x',y,z'),P(x,y',z) & P(x,f(f(f(y))),z)\\
P(x',y',z),P(x,y,z') & P(x,y,f(f(f(f(f(z))))))\\
P(f(f(x)),y,z) & P(x,y,z)\\
P(x,f(f(f(y))),z) & P(x,y,z)\\
P(x,y,f(f(f(f(f(z)))))) & P(x,y,z)\\
P(f(a),y,z) & \\
P(x,f(a),z) & \\
P(x,f(f(a)),z) & \\
P(x,y,f(a)) & \\
P(x,y,f(f(a))) & \\
P(x,y,f(f(f(a)))) & \\
P(x,y,f(f(f(f(a))))) &
\end{array}
$$

```
cnf(clause1,axiom,(~sP(U,V) | sP(U,W) | sP(W,V))).

cnf(clause2,axiom,(sP(a,a))).

cnf(clause3,axiom,(~sP(f(a),U))).

cnf(clause4,axiom,(~sP(f(f(U)),V) | sP(U,V))).

cnf(clause5,axiom,(~sP(U,V) | sP(f(f(U)),V))).

cnf(clause6,axiom,(~f(U) = U)).

cnf(clause7,axiom,(~f(f(U)) = U)).

cnf(clause8,axiom,(~f(f(f(U))) = U)).

cnf(clause9,axiom,(~f(f(f(f(U)))) = U)).

cnf(clause10,axiom,(~f(f(f(f(f(U))))) = U)).

cnf(clause11,axiom,(~f(f(f(f(f(f(U)))))) = U)).

cnf(clause12,axiom,(~f(f(f(f(f(f(f(U))))))) = U)).

cnf(clause13,axiom,(~f(f(f(f(f(f(f(f(U)))))))) = U)).

cnf(clause14,axiom,(~f(f(f(f(f(f(f(f(f(U))))))))) = U)).

cnf(clause15,axiom,(~f(f(f(f(f(f(f(f(f(f(U)))))))))) = U)).

cnf(clause16,axiom,(~f(f(f(f(f(f(f(f(f(f(f(U))))))))))) = U)).

cnf(clause17,axiom,(a = b)).
```

Figure A.1: Example 7.5.1 for $n = 11$ in the TPTP format.

SPASS-AR generates the following approximation and saturation.

$65[Inp : 21]$	$\rightarrow P(a,a,a)^*$
$51[Inp : 29]$	$S_1(u), P(v,w,x)^+ \rightarrow P(u,w,x)$
$50[Inp : 31]$	$S_2(u) \rightarrow S_1(f(u))^*$
$49[Inp : 32]$	$P(u,v,w)^+ \rightarrow S_2(f(u))$
$55[Inp : 33]$	$S_3(u), P(v,w,x)^+ \rightarrow P(v,u,x)$
$54[Inp : 35]$	$S_4(u) \rightarrow S_3(f(u))^*$
$53[Inp : 37]$	$S_5(u) \rightarrow S_4(f(u))^*$
$52[Inp : 38]$	$P(u,v,w)^+ \rightarrow S_5(f(v))$
$61[Inp : 39]$	$S_6(u), P(v,w,x)^+ \rightarrow P(v,w,u)$
$60[Inp : 41]$	$S_7(u) \rightarrow S_6(f(u))^*$
$59[Inp : 43]$	$S_8(u) \rightarrow S_7(f(u))^*$
$58[Inp : 45]$	$S_9(u) \rightarrow S_8(f(u))^*$
$57[Inp : 47]$	$S_{10}(u) \rightarrow S_9(f(u))^*$
$56[Inp : 48]$	$P(u,v,w)^+ \rightarrow S_{10}(f(w))$
$62[Inp : 18]$	$P(f(f(u)),v,w)^+ \rightarrow P(u,v,w)$
$63[Inp : 19]$	$P(u,f(f(f(v))),w)^+ \rightarrow P(u,v,w)$
$64[Inp : 20]$	$P(u,v,f(f(f(f(f(w))))))^+ \rightarrow P(u,v,w)$
$66[Inp : 22]$	$P(f(a),u,v)^+ \rightarrow$
$67[Inp : 23]$	$P(u,f(a),v)^+ \rightarrow$
$68[Inp : 24]$	$P(u,f(f(a)),v)^+ \rightarrow$
$69[Inp : 25]$	$P(u,v,f(a))^+ \rightarrow$
$70[Inp : 26]$	$P(u,v,f(f(a)))^+ \rightarrow$
$71[Inp : 27]$	$P(u,v,f(f(f(a))))^+ \rightarrow$
$72[Inp : 28]$	$P(u,v,f(f(f(f(a)))))^+ \rightarrow$
$73[Res : 65,56]$	$\rightarrow S_{10}(f(a))^*$
$74[Res : 65,52]$	$\rightarrow S_5(f(a))^*$
$75[Res : 65,49]$	$\rightarrow S_2(f(a))^*$
$76[Res : 65,61]$	$S_6(u) \rightarrow P(a,a,u)^*$
$77[Res : 65,55]$	$S_3(u) \rightarrow P(a,u,a)^*$
$78[Res : 76,69]$	$S_6(f(a))^+ \rightarrow$
$79[Res : 76,70]$	$S_6(f(f(a)))^+ \rightarrow$
$80[Res : 76,71]$	$S_6(f(f(f(a))))^+ \rightarrow$
$81[Res : 76,72]$	$S_6(f(f(f(f(a)))))^+ \rightarrow$
$84[Res : 76,56]$	$S_6(u) \rightarrow S_{10}(f(u))^*$
$85[Res : 76,55]$	$S_6(u), S_3(v) \rightarrow P(a,v,u)^*$

$88[Res:60,78]$	$S_7(a)^+ \rightarrow$
$89[Res:60,79]$	$S_7(f(a))^+ \rightarrow$
$90[Res:65,51]$	$S_1(u) \rightarrow P(u,a,a)^*$
$91[Res:76,51]$	$S_6(u), S_1(v) \rightarrow P(v,a,u)^*$
$92[Res:59,89]$	$S_8(a)^+ \rightarrow$
$93[Res:60,80]$	$S_7(f(f(a)))^+ \rightarrow$
$94[Res:59,93]$	$S_8(f(a))^+ \rightarrow$
$95[Res:76,64]$	$S_6(f(f(f(f(f(u))))))^+ \rightarrow P(a,a,u)$
$96[Res:58,94]$	$S_9(a)^+ \rightarrow$
$97[Res:60,81]$	$S_7(f(f(f(a))))^+ \rightarrow$
$98[Res:77,67]$	$S_3(f(a))^+ \rightarrow$
$99[Res:77,68]$	$S_3(f(f(a)))^+ \rightarrow$
$101[Res:77,52]$	$S_3(u) \rightarrow S_5(f(u))^*$
$103[Res:77,63]$	$S_3(f(f(f(u))))^+ \rightarrow P(a,u,a)$
$104[Res:77,51]$	$S_3(u), S_1(v) \rightarrow P(v,u,a)^*$
$108[Res:54,98]$	$S_4(a)^+ \rightarrow$
$109[Res:54,99]$	$S_4(f(a))^+ \rightarrow$
$110[Res:53,109]$	$S_5(a)^+ \rightarrow$
$111[Res:90,66]$	$S_1(f(a))^+ \rightarrow$
$112[Res:90,49]$	$S_1(u) \rightarrow S_2(f(u))^*$
$115[Res:90,62]$	$S_1(f(f(u)))^+ \rightarrow P(u,a,a)$
$120[Res:50,111]$	$S_2(a)^+ \rightarrow$
$121[Res:59,97]$	$S_8(f(f(a)))^+ \rightarrow$
$122[Res:58,121]$	$S_9(f(a))^+ \rightarrow$
$123[Res:57,122]$	$S_{10}(a)^+ \rightarrow$
$124[Res:50,115]$	$S_2(f(u))^+ \rightarrow P(u,a,a)$
$127[Res:54,103]$	$S_4(f(f(u)))^+ \rightarrow P(a,u,a)$
$128[Res:53,127]$	$S_5(f(u))^+ \rightarrow P(a,u,a)$
$140[Res:85,63]$	$S_6(u), S_3(f(f(f(v))))^+ \rightarrow P(a,v,u)$
$141[Res:85,64]$	$S_6(f(f(f(f(f(u))))))^+, S_3(v) \rightarrow P(a,v,u)$
$142[Res:85,51]$	$S_6(u), S_3(v), S_1(w) \rightarrow P(w,v,u)^*$
$155[Res:91,62]$	$S_6(u), S_1(f(f(v)))^+ \rightarrow P(v,a,u)$
$156[Res:91,64]$	$S_6(f(f(f(f(f(u))))))^+, S_1(v) \rightarrow P(v,a,u)$
$168[Res:104,62]$	$S_3(u), S_1(f(f(v)))^+ \rightarrow P(v,u,a)$
$169[Res:104,63]$	$S_3(f(f(f(u))))^+, S_1(v) \rightarrow P(v,u,a)$
$175[Res:60,95]$	$S_7(f(f(f(f(u)))))^+ \rightarrow P(a,a,u)$

$176[Res: 59, 175]$	$S_8(f(f(f(u))))^+ \to P(a, a, u)$
$177[Res: 50, 155]$	$S_2(f(u))^+, S_6(v) \to P(u, a, v)$
$178[Res: 58, 176]$	$S_9(f(f(u)))^+ \to P(a, a, u)$
$179[Res: 57, 178]$	$S_{10}(f(u))^+ \to P(a, a, u)$
$184[Res: 50, 168]$	$S_2(f(u))^+, S_3(v) \to P(u, v, a)$
$187[Res: 54, 140]$	$S_4(f(f(u)))^+, S_6(v) \to P(a, u, v)$
$188[Res: 53, 187]$	$S_5(f(u))^+, S_6(v) \to P(a, u, v)$
$191[Res: 54, 169]$	$S_4(f(f(u)))^+, S_1(v) \to P(v, u, a)$
$192[Res: 53, 191]$	$S_5(f(u))^+, S_1(v) \to P(v, u, a)$
$205[Res: 142, 62]$	$S_6(u), S_3(v), S_1(f(f(w)))^+ \to P(w, v, u)$
$206[Res: 142, 63]$	$S_6(u), S_3(f(f(f(v))))^+, S_1(w) \to P(w, v, u)$
$207[Res: 142, 64]$	$S_6(f(f(f(f(f(u))))))^+, S_3(v), S_1(w) \to P(w, v, u)$
$214[Res: 60, 141]$	$S_7(f(f(f(f(u)))))^+, S_3(v) \to P(a, v, u)$
$215[Res: 60, 156]$	$S_7(f(f(f(f(u)))))^+, S_1(v) \to P(v, a, u)$
$216[Res: 59, 214]$	$S_8(f(f(f(u))))^+, S_3(v) \to P(a, v, u)$
$217[Res: 58, 216]$	$S_9(f(f(u)))^+, S_3(v) \to P(a, v, u)$
$218[Res: 57, 217]$	$S_{10}(f(u))^+, S_3(v) \to P(a, v, u)$
$221[Res: 59, 215]$	$S_8(f(f(f(u))))^+, S_1(v) \to P(v, a, u)$
$222[Res: 58, 221]$	$S_9(f(f(u)))^+, S_1(v) \to P(v, a, u)$
$223[Res: 57, 222]$	$S_{10}(f(u))^+, S_1(v) \to P(v, a, u)$
$226[Res: 50, 205]$	$S_2(f(u))^+, S_6(v), S_3(w) \to P(u, w, v)$
$227[Res: 54, 206]$	$S_4(f(f(u)))^+, S_6(v), S_1(w) \to P(w, u, v)$
$230[Res: 53, 227]$	$S_5(f(u))^+, S_6(v), S_1(w) \to P(w, u, v)$
$233[Res: 60, 207]$	$S_7(f(f(f(f(u)))))^+, S_3(v), S_1(w) \to P(w, v, u)$
$234[Res: 59, 233]$	$S_8(f(f(f(u))))^+, S_3(v), S_1(w) \to P(w, v, u)$
$235[Res: 58, 234]$	$S_9(f(f(u)))^+, S_3(v), S_1(w) \to P(w, v, u)$
$236[Res: 57, 235]$	$S_{10}(f(u))^+, S_3(v), S_1(w) \to P(w, v, u)$

```
cnf(clause1,axiom,(~sP(U,X,Y) | ~sP(Z,V,W)
| sP(f(f(U)),V,W))).

cnf(clause2,axiom,(~sP(X,V,Y) | ~sP(U,Z,W)
| sP(U,f(f(f(V))),W))).

cnf(clause3,axiom,(~sP(X,Y,W) | ~sP(U,V,Z)
| sP(U,V,f(f(f(f(f(W)))))))).

cnf(clause4,axiom,( ~sP(f(f(U)),V,W) | sP(U,V,W))).

cnf(clause5,axiom,( ~sP(U,f(f(f(V))),W) | sP(U,V,W))).

cnf(clause6,axiom,( ~sP(U,V,f(f(f(f(f(W)))))) | sP(U,V,W))).

cnf(clause7,axiom,( sP(a,a,a))).

cnf(clause8,axiom,(~sP(f(a),V,W))).

cnf(clause9,axiom,(~sP(U,f(a),W))).

cnf(clause10,axiom,(~sP(U,f(f(a)),W))).

cnf(clause11,axiom,(~sP(U,V,f(a)))).

cnf(clause12,axiom,(~sP(U,V,f(f(a))))).

cnf(clause13,axiom,(~sP(U,V,f(f(f(a)))))).

cnf(clause14,axiom,(~sP(U,V,f(f(f(f(a))))))).
```

Figure A.2: Example 7.5.2 in the TPTP format.