Incremental Decision Procedures for Modal Logics with Nominals and Eventualities

Mark Kaminski

Dissertation zur Erlangung des Grades
des Doktors der Naturwissenschaften
der Naturwissenschaftlich-Technischen Fakultäten
der Universität des Saarlandes

Saarbrücken, 2012

Dissertation zur Erlangung des Grades
des Doktors der Naturwissenschaften
der Naturwissenschaftlich-Technischen Fakultäten
der Universität des Saarlandes,

eingereicht von
Mark Kaminski, B.Sc., M.Sc.,
geboren am 23. Februar 1982 in Woronesch, Russland.

Tag des Kolloquiums:

24. Januar 2012

Textfassung vom 6. Februar 2012

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form in einem Verfahren zur Erlangung eines akademischen Grades vorgelegt.

Saarbrücken, 6. Februar 2012

Mark Kaminski

Abstract

This thesis contributes to the study of incremental decision procedures for modal logics with nominals and eventualities. Eventualities are constructs that allow to reason about the reflexive-transitive closure of relations. Eventualities are an essential feature of temporal logics and propositional dynamic logic (PDL). Nominals extend modal logics with the possibility to reason about state equality. Modal logics with nominals are often called hybrid logics. Incremental procedures are procedures that can potentially solve a problem by performing only the reasoning steps needed for the problem in the underlying calculus.

We begin by introducing a class of syntactic models called demos and showing how demos can be used for obtaining nonincremental but worst-case optimal decision procedures for extensions of PDL with nominals, converse and difference modalities. We show that in the absence of nominals, such nonincremental procedures can be refined into incremental demo search procedures, obtaining a worst-case optimal decision procedure for modal logic with eventualities. We then develop the first incremental decision procedure for basic hybrid logic with eventualities, which we eventually extend to deal with hybrid PDL.

The approach in the thesis suggests a new principled design of modular, incremental decision procedures for expressive modal logics. In particular, it yields the first incremental procedures for modal logics containing both nominals and eventualities.

Zusammenfassung

Diese Dissertation untersucht inkrementelle Entscheidungsverfahren für Modallogiken mit Nominalen und Eventualities. Eventualities sind Konstrukte, die erlauben, über den reflexiv-transitiven Abschluss von Relationen zu sprechen. Sie sind ein Schlüsselmerkmal von Temporallogiken und dynamischer Aussagenlogik (PDL). Nominale erweitern Modallogik um die Möglichkeit, über Gleichheit von Zuständen zu sprechen. Modallogik mit Nominalen nennt man Hybridlogik. Inkrementell ist ein Verfahren dann, wenn es ein Problem so lösen kann, dass für die Lösung nur solche Schritte in dem zugrundeliegenden Kalkül gemacht werden, die für das Problem relevant sind.

Wir führen zunächst eine Klasse syntaktischer Modelle ein, die wir Demos nennen. Wir nutzen Demos um nichtinkrementelle aber laufzeitoptimale Entscheidungsverfahren für Erweiterungen von PDL zu konstruieren. Wir zeigen, dass im Fall ohne Nominale solche Verfahren durch algorithmische Verfeinerung zu inkrementellen Verfahren ausgebaut werden können. Insbesondere erhalten wir so ein optimales Verfahren für Modallogik mit Eventualities. Anschließend entwickeln wir das erste inkrementelle Verfahren für Hybridlogik mit Eventualities, welches wir schließlich auf hybrides PDL erweitern.

Die Dissertation vermittelt einen neuen Ansatz zur Konstruktion modularer, inkrementeller Entscheidungsverfahren für expressive Modallogiken. Insbesondere liefert der Ansatz die ersten inkrementellen Verfahren für Modallogiken mit Nominalen und Eventualities.

Acknowledgements

This thesis reports the results of several years of research. There are many people who have contributed to this research in different ways, and I would like to take this opportunity to thank those I feel most obliged to.

First of all, I want to express my deepest gratitude to my advisor, Gert Smolka. All of the results in this thesis were obtained in close collaboration with Gert. Almost everything I know about scientific thinking and writing I owe to him. In countless discussions, he offered constant support, encouragement, and invaluable advice on all aspects of my research. I would also like to thank Renate Schmidt and Patrick Blackburn for agreeing to give their expert opinion on this thesis.

During the work on this thesis, I was employed at the Programming Systems Lab at Saarland University. As part of my work, I had the privilege to advise three excellent students: Daniel Götzmann, Sigurd Schneider, and Tobias Tebbi. Working with them was a thoroughly gratifying and a very educational experience for me. In particular, the work with Sigurd on his bachelor's thesis had a great influence on the development of my own thesis.

The Programming Systems Lab has been a great working environment thanks to the other members of the lab who were always willing to share their knowledge and offer their help. For this, I would like to thank all of my current and former colleagues: Gert, Chad E. Brown, Ralph Debusmann, Christian Doczkal, Marco Kuhlmann, Mathias Möhl, Sandra Neumann, Andreas Rossberg, Sigurd, Thomas Schneider, Jan Schwinghammer, Guido Tack, and Ann Van de Veire. Special thanks goes to Thomas, who is also a coauthor of one of the papers underlying this thesis. Thanks also to our student assistants Robert Künnemann, Hannes von Haugwitz, and Bibek Paudel, who helped with system administration at the Lab.

I would also like to mention Guillaume Hoffmann and Florian Widmann who visited me at the Lab to discuss aspects of their and my work. I greatly enjoyed our discussions and benefited from them.

Last but not least, I would like to thank my parents for their support and their patience.

Saarbrücken, October 2011 *Mark Kaminski*

Contents

1 Introduction

This thesis studies incremental decision procedures for expressive modal logics that combine features known from propositional dynamic logic [67, 104] with those of hybrid logic [28, 10]. The procedures studied in this thesis are based on the tableau method, which reduces the satisfiability problem for a formula s to the task of finding a syntactic model of s. We explore two different tableau-based approaches to incremental decision procedures and devise the first such procedure for a hybrid extension of propositional dynamic logic.

This chapter outlines the motivation behind the thesis and surveys some related work. After that, it provides an overview of the thesis and its main contributions.

1.1 Modal Logic

The term "modal logic" denotes a broad family of languages that vary in expressiveness, complexity, and application areas.

1.1.1 Basic Modal Logic

All of the logics treated in this thesis are based on *relational semantics*. Relational semantics for modal logic is usually attributed to Kripke [139, 140], and is hence also known as *Kripke semantics*. Relational models can be seen as transition systems where the states are labeled with atomic propositions. Formulas are evaluated at a state of a transition system, where an atomic proposition p holds at a state w if w is labeled with p.

The basic modal logic with a relational semantics is called K. K extends propositional logic by allowing existential and universal quantification over the direct successors of a state (with respect to a transition relation) using *diamond* and *box formulas*. The diamond formula $\Diamond s$ is true at states that have a successor satisfying s, while the box formula $\Box s$ holds at states all of whose successors satisfy s. The *satisfiability problem* is the problem of finding out whether a given formula is satisfiable, i.e., whether there is a model and a state of the model where the formula is true. Many typical reasoning tasks for modal logic can be reduced to the satisfiability problem. When speaking of the *decidability* or *complexity of a logic*, one usually means the decidability or complexity of its satisfiability problem.

The basic technique for establishing the decidability of modal logics is *filtration*, developed by Lemmon and Scott [144]. In fact, filtration establishes a stronger property, called the *small model property*. A logic has the small model property if every satisfiable formula of the logic has a finite model whose size is bounded by some function in the size of the formula. Given a model $\mathfrak{M}$ of a formula s, filtration as used by Lemmon and Scott obtains a small model of s as a quotient of $\mathfrak{M}$ by an equivalence relation that relates two states of $\mathfrak{M}$ if and only if they satisfy the same subformulas of s. Filtration yields a NExpTime upper bound on the complexity of K. As later shown by Ladner [142], the satisfiability problem for K is PSpace-complete.

1.1.2 Propositional Dynamic Logic

For the present thesis, two extensions of modal logic will be of particular importance. The first extension is known under the name *propositional dynamic logic* (PDL). PDL was introduced by Fischer and Ladner [66, 67] based on earlier work by Pratt [159] as a decidable logic for reasoning about program correctness and program equivalence. Besides seeing some use in program verification [180] (along with more expressive dynamic logics; see [21]), PDL has applications in areas like planning [162] or synthesis of web services [23]. A comprehensive introduction to PDL is given by Harel et al. [104].

In PDL, diamond formulas have the form $\langle\alpha\rangle s$ and box formulas the form $[\alpha]s$, where α is called a *program*. Intuitively, a diamond formula $\langle\alpha\rangle s$ is read as "there is a terminating execution of the program α that leads to a state satisfying s", while a box formula $[\alpha]s$ is interpreted as "every terminating execution of α ends in a state satisfying s". Formally, programs denote transition relations. They are built over a set of atomic programs, called *actions*, using three possible constructions:

- Nondeterministic choice between two programs, written $\alpha + \beta$ (where α and β are programs), denotes the union of the relations denoted by α and β.
- Program composition, written $\alpha\beta$, denotes the composition of the relations denoted by α and β.
- Iteration, written α^*, denotes the reflexive-transitive closure of the relation denoted by α.

In addition, programs can contain *tests*. A test is a formula s that is interpreted as a program that tests if s holds. The program proceeds if s is true and fails otherwise. To see how tests can be put to use, consider the program $pa+(\neg p)b$, where p, $\neg p$ are tests and a, b are actions. The program executes a whenever p is true, and b whenever p is false. In other words, the program encodes the conditional **if** p **then** a **else** b. A while-loop of the form **while** p **do** a can then be represented as the program $(pa)^*(\neg p)$. The subprogram $(pa)^*$ ensures that a can be executed repeatedly, but only as long as p is true. The test $\neg p$ then ensures that the execution does not leave the loop unless p is false.

PDL represents a decidable fragment of Hoare logic [106] since a Hoare triple of the form $\{s\}\alpha\{t\}$ (where s, t are formulas of PDL and α a program) corresponds to the formula $s \to [\alpha]t$ (see [104], § 5.1 and § 5.7). Although PDL is decidable, it is not subsumed by first-order logic. In particular, unlike first-order logic, PDL is not compact (consider the infinite set $\{\langle a^*\rangle p, [a]p, [a][a]p, \dots\}$). The non-compactness of PDL can be traced back to diamond formulas of the form $\langle a^*\rangle s$. Following an established tradition in temporal logics [158, 44, 58, 59, 57], we call such formulas *eventualities* because a formula $\langle a^*\rangle s$ expresses that after finitely many a-transitions s will eventually become true.

The satisfiability problem for PDL is ExpTime-complete. While decidability and ExpTime-hardness were already shown by Fischer and Ladner (by filtration and a reduction from linear space bounded alternating Turing machines, respectively), the first decision procedure for PDL that runs in deterministic exponential time was devised by Pratt [160].

1.1.3 Hybrid Logic

The second extension of modal logic considered in this thesis is known as *hybrid logic*. Hybrid logic originates in the work of Prior [163, 164] in the 1960s. The first rigorous treatment of hybrid logic was given by Bull [36]. In the 1980s, hybrid logic was reinvented by Passy and Tinchev [153, 155] in the context of (an expressive variant of) PDL. The name “hybrid logic” was coined by Blackburn and Seligman [28]. Hybrid logic has applications in linguistics, where it can be used to represent temporal reference [26], and in the processing of semistructured data [72]. For more background on hybrid logic, see [27, 10, 34].

Hybrid logic extends modal logic by the ability to refer to individual states of the model. The simplest hybrid logic is obtained by extending modal logic with *nominals*. Nominals are atomic propositions that are true at exactly one state of the model. Thus, nominals serve as names for states. Nominals can be used to express equality and disequality constraints between states. For instance, the formula $x \wedge \neg y$, where x and y are nominals, is satisfiable if and only if x and y denote different states. Another common feature of hybrid logic are *satisfaction operators*. They allow to form *satisfaction formulas* $x : s$ (also written as $@_x s$) where x is a nominal and s a formula. The satisfaction formula $x : s$ expresses that s holds at the state denoted by x.

The expressiveness of hybrid logic can also be obtained via *difference modalities*. Difference modalities (see, for instance, [179, 134, 54, 76, 27]), which consist of an *existential difference modality* D and its *universal* dual $\bar{\mathsf{D}}$, are expressive modal operators that combine equality reasoning as provided by nominals with global quantification over states. Intuitively, a formula $\mathsf{D}s$ means that s holds “somewhere else” in the model (i.e., at some state of the model different from the current state), while $\bar{\mathsf{D}}s$ means that s holds “everywhere else”. Difference modalities can express *global modalities* (see [89]), written E and A. Intuitively, $\mathsf{E}s$ expresses that s holds at some state of the model, while $\mathsf{A}s$ expresses that s holds everywhere in the model. This semantics is captured in terms of difference modalities by the equivalences $\mathsf{E}s \equiv s \vee \mathsf{D}s$ and $\mathsf{A}s \equiv s \wedge \bar{\mathsf{D}}s$. More interestingly, difference modalities can express both nominals and satisfaction operators. So, the formula $\mathsf{E}(p \wedge \bar{\mathsf{D}}\neg p)$ (or, equivalently, $(p \wedge \bar{\mathsf{D}}\neg p) \vee \mathsf{D}(p \wedge \bar{\mathsf{D}}\neg p)$) enforces that p is true at exactly one state. Once this is the case, we have $p : s \equiv \mathsf{E}(p \wedge s)$.

Also, every formula with difference modalities can be translated to an equisatisfiable formula with nominals and global modalities [76, 5].

Adding nominals and satisfaction operators to K or PDL does not change the complexities of the logics. While K extended with nominals and satisfaction operators remains PSpace-complete [4], PDL extended with the two hybrid features is still ExpTime-complete [155]. For other logics, however, adding these features may result in a jump in complexity [4], or even in undecidability [31]. Since difference modalities can express global modalities, adding them to K already results in an ExpTime-complete logic [185, 5]. Extending hybrid logic with more expressive features like quantifiers [163, 164, 36, 154] or the downarrow binder [87] typically results in undecidability [4, 10].

1.2 Tableau Method

1.2.1 Incremental Decision Procedures and the Tableau Method

A key factor for the applicability of an algorithm in practice is its efficiency. But how do we measure efficiency? A common practice is to call an algorithm efficient if it has polynomial worst-case complexity. This, however, does no justice to algorithms like the simplex method in linear programming (see, e.g., [176]) or Hindley-Milner type inference for ML (see [48, 157]). Although the worst-case running time of the two algorithms is exponential, both are highly successful in applications since they perform well on practical problems. Therefore, our efficiency measure for an algorithm shall be its performance on practical problems.

Efficient decision procedures for the satisfiability problem are of central importance in many application areas of modal logic (see, for instance, [20]). A property that is essential for the practical efficiency of decision procedures for expressive logics is *incrementality*. We call a decision procedure incremental if the reasoning steps it performs in order to solve a given problem are determined by the structure of the problem. This way, if the problem is relatively simple, an incremental procedure may be able to solve it fast regardless of the worst-case complexity of the logic the problem is stated in. Thus, even for logics with prohibitive worst-case complexity incremental methods often perform well on fairly large problems, and hence may be used as a basis for practical systems [113, 100, 190, 111].

One of the most successful approaches to incrementally deciding modal logics is known as the *tableau method*. The origins of the tableau method can be traced back to sequent systems introduced by Gentzen [79]. In its original form, the tableau method was developed by Beth [25] and Hintikka [105]. The first application of the method to modal logic is due to Kripke [139, 140]. In [140], Kripke devises the first decision procedures for modal logic based on the tableau method. In the following, we will ignore the proof-theoretic aspects of tableau systems and their applications to undecidable logics, and only look at tableaux as a basis for incremental decision procedures for satisfiability. Decision procedures based on the tableau method we will call *tableau procedures*.

Given a formula s, a tableau procedure searches for a syntactic model of s, i.e., a syntactic representation of a transition system satisfying s. If it succeeds in finding a model, s is satisfiable. Otherwise, s is unsatisfiable.

1.2.2 Formula Decomposition

To guarantee termination, it is essential that tableau procedures work within a finite search space. This is achieved by exploiting a key property of the tableau method known as *analyticity*. In our setting, analyticity means that one only searches for models built over a finite syntactic closure induced by the decomposition of the initial formula into simpler formulas. For K, the formula decomposition can be described by the following table (assuming all formulas are in negation normal form):

formula	type	constituents
$s \wedge t$	α	$s,\ t$
$s \vee t$	β	$s,\ t$
$\Diamond s$	μ	s
$\Box t$	μ	s

The satisfiability of the formulas on the left-hand side of the table is determined from the satisfiability of their constituents on the right-hand side, possibly in combination with other formulas. So, to determine if a state satisfies a conjunction $s \wedge t$, a tableau system will typically decompose the conjunction into its two conjuncts s and t, and check if both are satisfied. For disjunctions, one proceeds dually. To check the satisfiability of a formula $\Diamond s$ at a state that is also required to satisfy the boxes $\Box t_1, \dots, \Box t_n$, it suffices to check the satisfiability of the set $\{s, t_1, \dots, t_n\}$.

The table distinguishes between *propositional decomposition rules* (α, β) and *modal decomposition rules* (μ). Following Smullyan [184], propositional decomposition rules can be further classified into *alpha rules* for conjunctive formulas and *beta rules* for disjunctive formulas. Formulas that cannot be decomposed by propositional decomposition rules we call *literals* (they are called *elementary formulas* in [22, 57]).

The formula decomposition for K *terminates* in the sense that there is no infinite chain of decompositions of a formula into its constituents. This is easy to see since the rules always decompose formulas into proper subformulas. In fact, the syntactic closure of a formula s induced by the above decomposition rules is precisely the subformula closure of s.

Fischer and Ladner [67], extended the formula decomposition for K to PDL. Besides conjunctions and disjunctions, the Fischer-Ladner decomposition introduces alpha and beta rules for diamond and box formulas over complex programs, decomposing them into formulas over simpler programs. For instance, formulas over programs of the form γ^* are decomposed as follows.

formula	type	constituents
$[\gamma^*]s$	α	$s,\ [\gamma][\gamma^*]s$
$\langle\gamma^*\rangle s$	β	$s,\ \langle\gamma\rangle\langle\gamma^*\rangle s$

While the formula s is a subformula of $[\gamma^*]s$, $[\gamma][\gamma^*]s$ is not. Therefore, the syntactic closure of a formula s induced by the Fischer-Ladner decomposition, known as the *Fischer-Ladner closure* of s, is generally larger than the subformula closure of s. Still, like the subformula closure of s, the Fischer-Ladner closure of s is finite, its size being linear in the size of s.

As noted by Abate et al. [3, 200], the alpha and beta rules of the Fischer-Ladner decomposition are not terminating (in contrast to the decomposition rules for K). On certain

$$\begin{tikzcd}
& \langle a^{**}\rangle p \arrow[dl] \arrow[dr] & & \\
p & & \langle a^{*}\rangle\langle a^{**}\rangle p \arrow[dr] \arrow[ul, bend left=90] & \\
& & & \langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle p
\end{tikzcd}
\qquad
\begin{tikzcd}
& a^{**} \arrow[dl] \arrow[dr] & \\
\varepsilon & & a^{*}a^{**} \arrow[dr] \arrow[ul, bend left=90] \\
& & & aa^{*}a^{**}
\end{tikzcd}$$

Figure 1.1: A formula decomposition cycle
and the underlying program decomposition cycle

diamond and box formulas, they may run into cycles, as demonstrated in Figure 1.1 for the formula $\langle a^{**}\rangle p$. Abate et al. call them "at a world" cycles. Such cycles can occur for diamond and box formulas over complex programs and are caused by cycles in the underlying program decomposition (shown for the program a^{**} in Figure 1.1).

1.2.3 Candidate Models and Demos

Tableau procedures search for syntactic models of a given formula by working with intermediate representations we call *candidate models*. A candidate model is a model whose states are labeled with formulas from the syntactic closure of the initial formula. The set of all formulas labeling a state is called the *label set* of the state. Candidate models all of whose states satisfy all formulas in their respective label sets we call *demos*.

Given a formula s, a tableau procedure typically starts with a candidate model consisting of a single state labeled with s. The goal of the procedure is to extend this initial candidate model to a demo. Every demo extending the initial candidate model will satisfy s since it will contain s in the label set of one of its states.

For instance, consider the following derivation.

$$\begin{tikzcd}[row sep=small]
\text{step 0:} & \boxed{\Diamond\Diamond p \wedge \Box\neg p} & & \\
& w_1 & & \\
\text{step 1:} & \boxed{\Diamond\Diamond p \wedge \Box\neg p,\ \Diamond\Diamond p,\ \Box\neg p} & & \\
& w_1 & & \\
\text{step 2:} & \boxed{\Diamond\Diamond p \wedge \Box\neg p,\ \Diamond\Diamond p,\ \Box\neg p} \arrow[r] & \boxed{\Diamond p,\ \neg p} & \\
& w_1 & w_2 & \\
\text{step 3:} & \boxed{\Diamond\Diamond p \wedge \Box\neg p,\ \Diamond\Diamond p,\ \Box\neg p} \arrow[r] & \boxed{\Diamond p,\ \neg p} \arrow[r] & \boxed{p} \\
& w_1 & w_2 & w_3
\end{tikzcd}$$

Beginning with with a candidate model consisting of a state w_1 labeled with $\Diamond\Diamond p \wedge \Box\neg p$, we derive in three steps a demo witnessing the satisfiability of $\Diamond\Diamond p \wedge \Box\neg p$.

Based on how they represent candidate models, we classify tableau procedures into two basic approaches. The *prefixed approach* works with *prefixed formulas*, i.e., with pairs of the form $\sigma : s$ asserting that the formula s is required to hold at the state denoted by the *prefix* σ. The label set of a state denoted by a prefix σ is then the set of all formulas prefixed with σ. A major advantage of the prefixed representation is its flexibility. In particular, the prefixed representation allows a *dynamic* handling of label sets in the sense that the label set of a state is allowed to grow over time (the above example derivation is dynamic in step 1). Assuming we want to extend the label set of a state denoted by a prefix σ with a formula s, we can always do so by introducing the prefixed formula $\sigma : s$. A disadvantage of the prefixed approach is that it often requires additional techniques to ensure termination. While the label sets are contained within the finite syntactic closure induced by the analytic formula decomposition, there is no a priori bound on the number of prefixes. The techniques used in tableau procedures to limit the number of prefixes that have to be considered are known as *blocking techniques* [111] or *loop-checks* [30, 29]. For expressive logics, complex blocking techniques are necessary, which may significantly complicate termination arguments for decision procedures [29, 117, 129]. In its modern form, the prefixed approach is due to Fitting [68, 70] and Massacci [146]. Earlier tableau systems as studied by Kripke [139, 140] (see also [70, 90, 71]) can be seen as precursors of the prefixed approach. While they do not work with explicit prefixes (and hence do not achieve the full flexibility of the prefixed approach), they are conceptually similar to prefixed systems. In particular, they often rely on blocking for termination.

A different approach to representing candidate models, pioneered by Pratt [160, 161], is motivated by filtration. Filtration can be seen as a technique that, given a model $\mathfrak{M}$ and a formula s, maps every state w of $\mathfrak{M}$ to the set of formulas from the syntactic closure of s that are satisfied by w. Thus, the states of a model $\mathfrak{M}'$ obtained from $\mathfrak{M}$ by filtration correspond to sets of formulas from the syntactic closure of s. We will call such formula sets *clauses* (not to be confused with the disjunctive clauses used by resolution methods). While clauses form the states of $\mathfrak{M}'$, they can also be interpreted as label sets. Since every clause of $\mathfrak{M}'$ satisfies all of its formulas, $\mathfrak{M}'$ is in fact a demo. Procedures that use clauses for representing states of candidate models we will call *clausal*. Since clauses are formed over a finite syntactic closure and since no model obtained by filtration may contain the same clause twice, the number of states in a clausal demo is (exponentially) bounded in the size of the syntactic closure. Since they work within a bounded search space, clausal procedures terminate without additional blocking mechanisms.

In contrast to the prefixed approach, the clausal approach is inherently *static*, meaning that the label set of a state is fixed on creation and cannot be changed afterwards (indeed, the approach makes no distinction between a state of a candidate model and its label set).

1.2.4 Graph Procedures for PDL

The first tableau procedure for PDL was devised by Pratt [161]. Pratt's procedure works with an explicit representation of the search space as an AND/OR graph whose nodes are labeled with sets of formulas. We call such graphs *graph tableaux*. The label sets of graph tableau nodes form the states of a clausal candidate model. To extract a demo from such a candidate model, the procedure relies on a technique we call *pruning*. Pruning proceeds by iteratively removing states of the candidate model that do not satisfy their labels until it arrives at a demo. Pruning takes polynomial time with respect to the size of the candidate model, which results in the overall procedure running in deterministic exponential time (which is optimal). In its simplest form, pruning appears in a second decision procedure by Pratt [160], developed for showing the ExpTime upper bound on the complexity of PDL.

Procedures based on graph tableaux we call *graph procedures*. Pratt's graph procedure is not incremental since before it can apply pruning it always needs to construct a complete graph tableau, while the satisfiability of many formulas can be decided by only looking at a small fraction of the complete tableau.

Pratt's graph procedure was later reformulated by Nguyen and Szałas [151], who adapted it to reasoning problems in description logics [18, 15]. Nguyen and Szałas' presentation differs in several aspects from Pratt's original formulation. In particular, their presentation of pruning employs a so-called *graph of traces* to distinguish satisfiable diamond formulas from unsatisfiable ones. Being essentially a representational variant of Pratt's procedure, Nguyen and Szałas' approach has the same advantages and disadvantages. In particular, Nguyen and Szałas' procedure is not incremental.

The first incremental decision procedure for PDL based on graph tableaux was devised by Goré and Widmann [93, 200]. Goré and Widmann's procedure is also the first decision method for PDL that achieves incrementality while retaining worst-case optimality. Their approach combines Pratt-style pruning with incremental tableau construction techniques introduced in [91]. Their procedure is backtracking-free, exploring the search space using *graph search*, and caching intermediate results to avoid recomputation. It stops as soon as it can determine that the graph tableau contains a demo satisfying the input formula, or that no such demo exists. An implementation of the procedure by the authors demonstrates a high practical potential of the approach. In [95, 200], Goré and Widmann extend their approach to PDL extended with *converse modalities*, which make it possible to talk about the *predecessors* of a state. It should be noted that the presentation of Goré and Widmann's approach in [93, 200] is algorithmically involved, which also shows in the complexity of the correctness proofs in [200]. While the algorithmic details of the presentation may facilitate efficient implementation, they also make it difficult to understand the abstract principles behind the procedure, which is essential if one wants to extend the procedure to more expressive logics.

1.2.5 Tree Procedures for PDL

Incremental decision procedures for PDL predating Goré and Widmann's approach are all based on *tree search* (see, for instance, [149, 141, 186]). Unlike graph search, tree search organizes the search space as a tree. In the case of tableau procedures, every branch of the tree corresponds to a candidate model of the initial formula s. The search tree is explored until one either finds a branch corresponding to a demo or rejects all candidate models, in which case s is unsatisfiable. Incremental tableau procedures typically explore the search tree branch by branch, backtracking on failure. Since for ExpTime-complete logics search trees produced by the tableau method are typically at least double exponential in the worst case, decision procedures for such logics that are based on tree search are usually not optimal. Still, being incremental, they often display good performance on practical problems [113, 100, 190].

We use the term *tree procedures* as a shorthand for "tableau procedures based on tree search". Historically, tree procedures predate graph procedures as decision methods for modal logic. Indeed, Kripke's [140] first tableau procedures for modal logic employ tree search.

The first tree procedure for PDL (without tests) was devised by Baader [12]. To distinguish demos from non-demos, the procedure introduces a loop detection mechanism, distinguishing between "good" and "bad" loops. To deal with complex programs, the procedure relies on automata-theoretic methods. Since the construction of the automata needed by the procedure may require exponential time, the method cannot be considered fully incremental.

Later, De Giacomo and Massacci [52] presented a fully incremental tree procedure for PDL with converse modalities. Overall, De Giacomo and Massacci's approach resembles that of Baader, but does not involve automata-theoretic techniques. The procedure runs in NExpTime. While a way is sketched to transform the procedure into one that runs in ExpTime, the discussion leaves some open questions.

Yet another incremental tree procedure was presented by Abate et al. [3, 200]. Their procedure improves on several aspects of Baader's and De Giacomo and Massacci's procedures but retains their essential features: suboptimal complexity due to tree search and the use of a loop detection mechanism to distinguish demos from non-demos. To deal with cycles in the alpha-beta formula decomposition, Abate et al. develop a dedicated loop detection mechanism. A modification of this mechanism is later used by Goré and Widmann [93] in their graph procedure.

1.2.6 Tableau Procedures for Hybrid Logic

Existing tableau procedures for hybrid logic are all based on tree search. The first attempt at a tableau procedure for hybrid logic with satisfaction operators is due to Tzakova [191]. Bolander and colleagues [30, 29] observed and fixed a flaw in Tzakova's procedure, eventually giving a range of tableau procedures for hybrid logics of varying expressiveness. Extensions to even stronger logics as well as algorithmic refinements of the procedures are explored in [126, 127, 124, 39, 107]. The different approaches have been implemented in the provers HTab [109], Spartacus [97, 98], Herod and Pilate [42]. In a parallel line of work, tableau procedures for description logics (which are syntactic variants of hybrid logic) were developed by Horrocks and Sattler [115, 117]. Their calculi have been implemented in systems like FaCT++ [189] or Pellet [182].

All existing tableau procedures for hybrid logic are based on the prefixed approach. The prefixed approach facilitates a dynamic handling of the state equality constraints introduced by nominals. Suppose, for instance, we derive that two existing prefixes σ

and τ must denote the same state of the candidate model (this constraint can be expressed by the two prefixed formulas $\sigma : x$ and $\tau : x$ for some nominal x). This equality constraint can be realized by propagating every formula associated with σ to τ and vice versa, thus making the annotations of σ and τ equal. Once the two annotations are equal, the two prefixes can be mapped to the same state of the model.

1.3 Questions Asked

As we have seen, there are practical incremental decision procedures for both PDL and hybrid logic. However, none of the existing procedures can deal with the combination of PDL and nominals. In this thesis, we investigate the question of whether and how the treatment of complex programs as they occur in PDL can be combined with the treatment of nominals in an incremental way.

More specifically, we are interested in obtaining an incremental tableau procedure for PDL extended with nominals. On the one hand, the procedure should allow efficient implementation. On the other hand, it should be modular and have clean correctness proofs to facilitate possible adaptation or extension to other logics.

To come up with such a procedure, we will have to answer questions such as:

- How do nominals interact with programs as it comes to decision procedures?
- Should we base our procedure on graph search or on tree search? More generally, what are the trade-offs between the two approaches for the logics we consider?
- Should we base our procedure on the prefixed or the clausal approach? Should we adopt a dynamic or a static handling of constraints on candidate models?
- How do we deal with cycles produced by the alpha and beta rules of the Fischer-Ladner decomposition?

1.4 Overview of the Thesis

Rather than trying to devise a procedure for the combination of PDL and hybrid logic in one go, we approach our goal in stages. In particular, we look at different sublogics to see what techniques are necessary to deal with the constructs specific to these logics. The main logics that are of relevance in this thesis can be summarized as follows.

$$
\begin{array}{r|ccc}
 & & \text{nominals} & \\
\hline
 & \text{K} & \subseteq & \text{H} \\
 & \rotatebox{-90}{$\subseteq$} & & \rotatebox{-90}{$\subseteq$} \\
\text{eventualities} & \text{K}^* & \subseteq & \text{H}^* \\
 & \rotatebox{-90}{$\subseteq$} & & \rotatebox{-90}{$\subseteq$} \\
\text{programs} & \text{PDL} & \subseteq & \text{HPDL}
\end{array}
$$

The extension of the basic modal logic K with nominals is called the *basic hybrid logic* H. By extending K and H with eventualities we obtain K* and H*, the *basic modal* and *basic hybrid logic with eventualities*. The extension of PDL with nominals we call *hybrid PDL*

(HPDL). We will also consider some extensions and restrictions of the logics in the table, most importantly a variant of (H)PDL that is obtained by restricting the syntax of programs, disallowing tests. We call the logic *test-free (H)PDL*.

Our work can be separated in two main parts. The first part is concerned with decision procedures based on pruning and graph search. The second part is dedicated to procedures based on tree search. All of the procedures developed in this thesis are based on the clausal approach since this allows a transparent presentation of the correctness arguments and simplifies termination proofs.

Existing clausal approaches [161, 151, 93] distinguish between two types of clauses, which we call *state clauses* and *auxiliary clauses*. State clauses represent states of a candidate model, whereas auxiliary clauses represent intermediate stages in computing state clauses. We adopt this idea when looking at graph procedures. In our case, state clauses are defined to be *literal*, i.e., to contain only literals, while auxiliary clauses may also contain nonliteral formulas. When we come to tree procedures, it turns out that in the presence of eventualities, the uniqueness constraints imposed on auxiliary clauses affect the correctness of tree search. A similar problem occurring with graph search and converse modalities is addressed by Goré and Widmann [94] by relaxing the uniqueness constraints on auxiliary clauses. We solve the problem differently by separating propositional reasoning represented by auxiliary clauses from modal reasoning. This results in a procedure with two interleaved stages. The top-level modal stage searches for demos, working exclusively with literal state clauses. To compute the successors of a clause C, the modal stage relies on a propositional stage, which, for every diamond formula $s \in C$, yields a set of clauses serving as candidates for the state required by s.

The tableau procedures developed in this thesis can be related to existing approaches as shown in the following table.

	tree search	graph search
prefixed	[146, 29, 117]	
clausal (flat)		[161, 151, 94, 93, 95], this thesis
clausal (staged)	this thesis	

1.4.1 Pruning and Graph Search

We begin our investigations by looking at Pratt's [160] abstract worst-case optimal procedure for PDL. While not incremental, it introduces pruning, which is essential for the graph tableau approach to PDL. Hence, extending the method to deal with nominals may provide the insights necessary for treating nominals within incremental graph procedures. We begin by introducing the notion of a demo. Similarly to Pratt's original representation in [160], we define a demo as a set of *Hintikka sets* that satisfies certain consistency criteria. Hintikka sets are sets of formulas saturated under alpha-beta decomposition (named after Hintikka [105], who introduced them under the name "model sets"). Unlike Pratt, who for the correctness of his approach relies on the assumption that Fischer and Ladner's [67] filtration technique for PDL adapts to his representation, we establish all the necessary small model properties within our framework. We begin by formulating and proving correct a Pratt-style abstract decision procedure for test-free PDL. When we then extend this language with nominals, we observe that the deterministic pruning algorithm underlying the procedure becomes incomplete. However, if we are, like Pratt, just interested in worst-case optimality, the incompleteness of pruning can be remedied with a guessing stage, which essentially reduces the decision problem with nominals to the case without nominals. We exploit this fact to obtain a Pratt-style worst-case optimal procedure for test-free HPDL. By further extending the guessing technique, we obtain a worst-case optimal procedure for an extension of the language with difference modalities. Finally, we look into what needs to be done to extend the approach to tests and converse modalities. Put together, the extensions yield a worst-case optimal procedure for HPDL extended with converse and difference modalities.

Just like Pratt's original method, however, our abstract procedure is nonincremental. One source of nonincrementality is that the procedure performs pruning starting from a candidate model that consists of all Hintikka sets over some formula universe. This issue is addressed and resolved by Goré and Widmann's [93] graph tableau approach. However, our procedure has an additional source of nonincrementality not present in Pratt's original method, namely the guessing stage we introduce to deal with nominals.

To see whether the nonincrementality introduced by guessing can be eliminated, we first need to understand the connection between Pratt's abstract procedure and the graph procedures for PDL in the absence of nominals. To improve this understanding, we study how the abstract procedure can be stepwise refined to incremental graph procedures while preserving worst-case optimality. Since for our purposes it is not necessary to consider full PDL, we restrict ourselves to K*. We first obtain a variant of Pratt's tableau procedure from [161], which we then refine further to a variant of Goré and Widmann's [93] incremental procedure.

We achieve incrementality by refining pruning into two complementary mechanisms: *eager pruning*, used to compute the set of clauses provably satisfiable in the current graph tableau, and *cautious pruning*, which yields the clauses provably unsatisfiable in the current graph tableau. To facilitate correctness proofs, we introduce a representation of demos based on literal clauses (rather than Hintikka sets). The representation is complemented by an inductively defined notion of *support*, which allows to infer the satisfaction of nonliteral formulas from literals.

Our incremental procedure is a simpler and more abstract variant of Goré and Widmann's [93] procedure (restricted to K*) that makes explicit the connections between incremental graph procedures for logics with eventualities and Pratt's [160, 161] nonincremental procedures. However, even after making this connection explicit, we still cannot see a way of obtaining incremental graph procedures that could handle nominals without relying on a guessing stage. Hence, we turn our attention to tree search.

1.4.2 Tree Search

Our goal is now to show that tree procedures for PDL can be extended to deal with nominals. For clarity of exposition, we first restrict ourselves to H*, the simplest logic that has both nominals and eventualities. To combine tree search with a clausal representation in the presence of eventualities, we separate modal reasoning from propositional reasoning. This results in a procedure with two interleaved stages: a modal stage working with literal state clauses and a propositional stage. To introduce the approach, we start with a procedure for K, and stepwise extend it to deal with eventualities and nominals. To deal with nominals, we develop a technique that we call *nominal propagation*. Unlike previous approaches [115, 30, 29, 117, 127, 39, 107], which all rely on dynamic propagation of equational constraints, nominal propagation allows nominals to be treated completely statically, without any a posteriori modification of label sets. Thus, the approach is compatible with our clausal representation and with a loop detection mechanism in the style of Baader [11].

Finally, we extend our procedure for H* to full HPDL. For clarity, we first restrict ourselves to test-free HPDL, whereafter we present the extensions necessary to deal with tests. As with H*, the procedure decomposes into modal reasoning on literal state clauses and the computation of literal state clauses using alpha-beta decomposition rules. The main complication that we encounter with HPDL as compared to H* is caused by the fact that an alpha-beta decomposition à la Fischer and Ladner [67] does not terminate. This destroys an essential representation invariant for clausal demos and graph tableaux, namely that every Hintikka set is equivalent to the set of its literals. For instance, consider the Hintikka set $H = \{q, \langle a^{**}\rangle(p \wedge \neg p), \langle a^{*}\rangle\langle a^{**}\rangle(p \wedge \neg p)\}$. While H is clearly unsatisfiable (both of its diamond formulas require the existence of a reachable state satisfying $p \wedge \neg p$), the only literal in H is q. Clearly, $\{q\}$ is satisfiable. At the same time, the nontermination of the naive alpha-beta decomposition affects the definition of support, which is of central importance for our approach. It turns out that the notion of support induced by the naive decomposition is too weak to infer the satisfaction of all relevant formulas. For instance, we cannot inductively infer that every state satisfying p and $[a][a^{*}][a^{**}]p$ also satisfies $[a^{**}]p$. We deal with the problem by devising a terminating formula decomposition that refines the naive alpha-beta decomposition. Unlike the naive decomposition, the terminating decomposition induces a well-founded order on formulas. This order allows to reestablish the correspondence between Hintikka sets and literal clauses, at the same time forming a basis for a new inductive definition of support. To argue the correctness of the revised decomposition in a modular way, we employ a language-theoretic semantics for programs. While in the test-free case we can use regular languages, to deal with tests, we adapt the language-theoretic semantics for Kleene algebra with tests [130, 137]. Once the termination of the alpha-beta decomposition is reestablished, we obtain the decision procedure for HPDL by extending our approach for H*.

1.4.3 Chapter Breakdown

Chapter 2 gives a survey of modal and hybrid logic with the focus on languages playing a role later in the thesis.

Chapter 3 introduces the notions of demos and pruning, and uses them to develop worst-case optimal decision procedures for HPDL extended with converse and difference modalities.

Chapter 4 extends the pruning technique to graph tableaux, obtaining an incremental decision procedure for K*.

Chapter 5 develops the first incremental decision procedure for H* based on tree search.

Chapter 6 extends the procedure for H* to deal with complex programs without tests, thus obtaining the first incremental decision procedure for test-free HPDL.

Chapter 7 extends the results of Chapter 6 to programs with tests, obtaining the first incremental decision procedure for HPDL.

Chapter 8 concludes the thesis with a discussion of the results and an outlook to future work.

1.5 Contributions

The main contributions of this thesis are as follows.

1. We introduce a class of syntactic models called demos and use them to give a modular analysis of decidability results and abstract worst-case optimal procedures for extensions of PDL. In particular:
   a) We simplify and modularize the proof of Fischer and Ladner's [67] filtration theorem for PDL compared to its presentations in the literature [67, 103, 138, 104], as well as extend the result to nominals and difference modalities.
   b) We extend Pratt's [160] worst-case optimal decision procedure for PDL to a richer logic with converse, nominals, and difference modalities and prove its correctness.
2. We develop a basic theory explaining the construction and the correctness of incremental worst-case optimal decision procedures based on graph tableaux. Using this theory we obtain a decision procedure for K*, which is a simplified and more abstract variant of Goré and Widmann's [93, 200] procedure for PDL.
3. We develop the first incremental decision procedures for modal logics containing both nominals and eventualities. After devising an incremental procedure for H* based on tree search, we extend it to deal first with test-free HPDL, and then with full HPDL. Our procedures are the first to combine the clausal approach with tree search. The most important innovations of our approach are as follows.
   a) We separate modal reasoning from propositional reasoning. The modal level works with literal state clauses, interacting with the propositional level via an abstract interface. Given a clause C and a diamond formula $s \in C$, the interface returns clauses serving as candidates for the state required by s. This modularization reconciles tree search in the presence of eventualities with a clausal representation of candidate models. Moreover, it allows to replace the naive approach to propositional reasoning taken in the thesis by any algorithm implementing the abstract propositional interface while preserving the correctness of the overall procedure.

b) We introduce nominal propagation as a technique that allows to deal with equational constraints introduced by nominals without relying on fine-grained dynamic propagation rules used by existing hybrid tableau systems [115, 30, 29, 117, 127, 39, 107].

c) We devise a novel terminating formula decomposition for HPDL. The decomposition allows to obtain a clausal representation and an inductive definition of support that are essential for our approach to HPDL. We prove the adequacy of the decomposition by adapting some basic results for Kleene algebra with tests [136, 137]. Unlike techniques based on a naive Fischer-Ladner-style decomposition [161, 151, 3, 93, 200], the terminating formula decomposition combines naturally with our clausal approach and allows transparent and modular correctness proofs.

1.5.1 Published Results

Parts of this thesis revise and extend material that has already been published.

- The analysis of demos and pruning as a means of obtaining modular decidability and complexity results in Chapter 3 extends the following paper.

  Mark Kaminski, Thomas Schneider, and Gert Smolka. *Correctness and worst-case optimality of Pratt-style decision procedures for modal and hybrid logics.* In *TABLEAUX 2011*, Kai Brünnler and George Metcalfe, editors, vol. 6793 of LNCS, pp. 196–210. Springer, 2011.

- The decision procedure for K* presented in Chapter 4 extends the following paper.

  Mark Kaminski and Gert Smolka. *Correctness of an incremental and worst-case optimal decision procedure for modal logic with eventualities.* Technical report, Saarland University, 2011.

- The decision procedure for H* presented in Chapter 5 is based on the following paper.

  Mark Kaminski and Gert Smolka. *Terminating tableaux for hybrid logic with eventualities.* In *IJCAR 2010*, Jürgen Giesl and Reiner Hähnle, editors, vol. 6173 of LNCS, pp. 240–254. Springer, 2010.

- Precursors of clausal demos in Chapters 4 and 5, as well as some of the techniques used to deal with nominals and difference modalities in Chapter 3 first appear in the following paper.

  Mark Kaminski and Gert Smolka. *Clausal graph tableaux for hybrid logic with eventualities and difference.* In *LPAR-17*, Christian Fermüller and Andrei Voronkov, editors, vol. 6397 of LNCS, pp. 417–431. Springer, 2010.

- The decision procedure for hybrid PDL presented in Chapters 6 and 7 is a thorough revision and extension of the following paper.

  Mark Kaminski and Gert Smolka. *Clausal tableaux for hybrid PDL.* In *M4M-7*, Hans van Ditmarsch, David Fernández Duque, Valentin Goranko, Wojtek Jamroga, and Manuel Ojeda-Aciego, editors, vol. 278 of ENTCS, pp. 99-113. Elsevier, 2011.

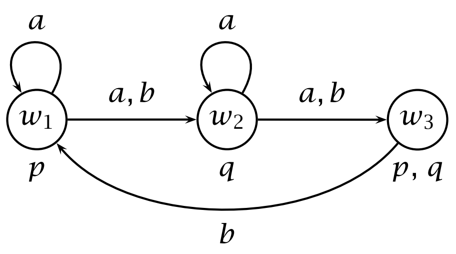
2 Modal and Hybrid Logics

Modal logic as we see it today can be traced back to the work of Lewis [145] published in 1918. However, it was not until 1959 when modal logic obtained a natural and widely accepted *relational semantics* through the work of Kripke [139, 140] (see [27, 85] for a discussion of Lewis' and Kripke's contributions). All of the logics treated in this thesis are based on relational semantics. For our purposes, a relational model will be a structure consisting of unary and binary relations over a nonempty *domain*. Depending on the particular application, these structures can be given different intuitive interpretation. For instance, elements of the domain can be seen as possible worlds, points in time, or objects of an ontology. We will view relational structures as *transition systems*, that is, directed labeled graphs. Elements of the domain are seen as *states* of the transition system (i.e., vertices of the graph). Unary relations are seen as *state labels*, and binary relations are seen as *transition labels* (i.e., edge labels), where every transition is required to have at least one label.

Consider, for instance, the following structure over the three-element domain $\{w_1, w_2, w_3\}$:

- $p = \{w_1, w_3\}$
- $q = \{w_2, w_3\}$
- $a = \{(w_1, w_1), (w_1, w_2), (w_2, w_2), (w_2, w_3)\}$
- $b = \{(w_1, w_2), (w_2, w_3), (w_3, w_1)\}$

The structure consists of two unary relations, called p and q, and two binary relations, a and b. The corresponding transition system can be represented graphically as follows:



Interpreted over transition systems, formulas denote sets of states. While the interpretation of purely propositional formulas depends solely on state labels, modal logic allows reasoning about the structure of the transition system, i.e., about how the states are connected with each other by transitions. The logics treated in this thesis will all have full propositional power but will differ in their modal expressiveness, their ability to describe structural properties of transition systems.

In this chapter, we introduce the logics covered by our decision methods. We give their syntax and semantics, discuss their expressive power and their relationship to each other. Also, we give an overview of the available decision methods for the logics. At the end, we briefly discuss some related logics not covered so far by our methods.

2.1 Basic Modal Logic

The **basic modal logic** K is the simplest modal logic discussed in this thesis. All other logics we will look at can be seen as extensions of K.

2.1.1 Syntax and Informal Semantics

We assume a nonempty set of **propositional variables**, or **predicates** for short, and a nonempty set of **relational variables**, also called **actions**. We will denote predicates by the letters p and q, and actions by the letters a and b. For every action a, the logic K introduces two **modal operators**: a **diamond operator** $\langle a \rangle$, and a **box operator** $[a]$. **Formulas** of K are defined using propositional connectives and the two modal operators with the grammar

$$s ::= p \mid \neg s \mid s \vee s \mid s \wedge s \mid \langle a \rangle s \mid [a]s$$

The formulas have the following intuitive semantics:

- p denotes all states labeled by p.
- $\neg s$ denotes the complement of (the denotation of) s.
- $s \vee t$ and $s \wedge t$ denote the union and the intersection of s and t, respectively.
- $\langle a \rangle s$ denotes all states that have an a-successor in s.
- $[a]s$ denotes all states all of whose a-successors are in s.

For instance, on the transition system on p. 17, the formula $\langle b \rangle [a]q$ denotes the states w_1 and w_2. For w_1, this is the case because it has a b-successor (w_2) all of whose a-successors (w_2 and w_3) are labeled with q. On the other hand, w_3 is not denoted by $\langle b \rangle [a]q$ since its only b-successor (w_1) has an a-successor (w_1) that is not labeled by q.

Note that the diamond operator allows existential quantification and the box operator universal quantification over the successors of a state. And indeed, the standard translation (see [27], § 2.4) of modal logic into classical first-order logic expresses every diamond by an existential quantifier and every box by a universal quantifier. Like the two classical first-order quantifiers, $\langle a \rangle$ and $[a]$ are dual to each other, obeying the following equivalence:

$$[a]s \;\equiv\; \neg \langle a \rangle \neg s$$

The formal semantics of K justifying this fact will be introduced in § 2.2.

It is common to restrict attention to only one action a and to consider only transition systems with a single transition relation labeled by a. In this case, one can omit the relational annotation of the modal operators, writing simply $\Diamond$ for $\langle a \rangle$ and $\Box$ for $[a]$. In most cases, results obtained for such **unimodal logics** easily scale to their counterparts with multiple actions.

2.1.2 Satisfiability Problem and Decision Methods

A transition system is said to *satisfy* a formula s if the denotation of s in the system is nonempty. A formula is *satisfiable* if it has at least one satisfying transition system. In the following, we will primarily be interested in answering the following question:

Given a formula s, is s satisfiable?

This question is known under the name **satisfiability problem**. The dual problem to satisfiability is determining the *validity* of a given formula. A formula s is **valid** if it is satisfied by every state in every transition system. Since a formula s is valid if and only if $\neg s$ is not satisfiable, checking validity is reducible in constant time to checking satisfiability and vice versa. In the following, whenever we speak of **deciding a logic**, we mean deciding the satisfiability of formulas of the logic, and whenever we speak of the **complexity of a logic**, we mean the complexity of its satisfiability problem.

Ladner [142] showed that the complexity of unimodal K is PSPACE-complete. The result was then extended to the multimodal case by Halpern and Moses [101].

How do we decide K? One of the most successful approaches to deciding K and modal logics in general is the tableau method. The first tableau procedures for modal logic are due to Kripke [140]. Tableau procedures for K appear in [167, 168, 70]. Also, Ladner's [142] worst-case optimal decision procedure for K can be seen as a tableau procedure. More modern presentations can be found in [146, 90]. Due to their simplicity, tableau procedures lend themselves to implementation. Currently, most implemented systems can deal with logics that are much more expressive than K. We will mention some of them when we come to these logics. While Ladner's decision procedure demonstrates that a tableau system for K can be implemented in polynomial space, efficient implementations often choose to use more space in order to cache intermediate results. As reported in [113, 83, 92], caching can considerably improve the performance of decision procedures, for modal logics, even in the case of K.

To speed up propositional reasoning in modal tableau procedures, Giunchiglia and colleagues [84, 81] propose to make use of the solver technology available for the propositional satisfiability problem (SAT). Somewhat misleadingly, the resulting approach is often called SAT-based or DPLL-based (DPLL standing for the Davis-Putnam-Logemann-Loveland [50, 49] algorithm for propositional satisfiability). More precisely, it can be seen as a hybrid approach, in which modal reasoning is done using a tableau calculus while propositional reasoning is delegated to a DPLL-based SAT solver. Provers based on this approach include KSAT [84] and its successor *SAT [82, 81].

Closely related to tableau systems are sequent calculi, also called Gentzen systems after their inventor [79]. There is a natural correspondence between tableau systems and sequent calculi, which, in particular, allows an easy transfer of results between the two approaches. For the special case of modal logic this correspondence is discussed in [90, 71]. As far as proof search is concerned, modal sequent calculi are considered largely equivalent to tableau systems and hence, in their traditional form, have not received much attention in the field of automated reasoning.

Another established decision method for K is resolution, which can be performed either directly on modal formulas [63, 148, 6] or after translating them into a decidable fragment of first-order logic [172, 53, 121, 173]. Implementations are available for both the direct [9] and the translational approach [120, 199].

Besides encodings into first-order logic, encodings into constraint satisfaction problems [32], quantified Boolean formulas [152], and even directly into SAT [178] have been proposed as decision methods for K.

Other notable approaches include the inverse method (a variant of a sequent calculus with an inverted proof search strategy) proposed by Voronkov [197, 198] and BDD-based procedures proposed by Pan et al. [152].

2.2 Propositional Dynamic Logic

Propositional dynamic logic (PDL) was introduced by Fischer and Ladner [66, 67] as a decidable fragment of dynamic logic [159, 102]. For a comprehensive introduction to PDL, see [104].

2.2.1 Syntax

As we have seen before, K allows us to quantify over the successors of a state that are reachable by a single action. PDL extends modal quantification to states reachable by complex sequences of actions. To describe such sequences, the language is extended by a new kind of terms called *programs*. Programs are defined by mutual recursion with formulas as follows:

$$\alpha ::= a \mid \alpha + \alpha \mid \alpha\alpha \mid \alpha^* \mid s \qquad \textbf{programs}$$

$$s ::= p \mid \neg s \mid s \vee s \mid s \wedge s \mid \langle\alpha\rangle s \mid [\alpha]s \qquad \textbf{formulas}$$

Note that K is a proper syntactic fragment of PDL, which is obtained by restricting the syntax of programs to actions.

Programs of the form s are called **tests**. Before we look at the full language, let us consider the sublanguage of PDL without tests, which we call **test-free PDL**. Test-free PDL is the language originally introduced by Fischer and Ladner in [66] and is studied in [27] under the name "regular PDL". Without tests, programs are nothing but regular expressions over actions. Hence, intuitively, a program denotes a regular language over transitions. In particular:

- a denotes all transitions labeled with a.
- $\alpha + \beta$ denotes the union of α and β.
- $\alpha\beta$ denotes the set of all concatenations of the words in α with those in β.
- α^* denotes the reflexive-transitive closure of α.

With this, we can already express fairly complex transition sequences. For instance, $\langle(a + b)b^*\rangle p$ denotes all states that can reach a state satisfying p by a nonempty sequence of transitions whose first transition is labeled with a or b and all of whose remaining transitions are labeled with b.

What we cannot express are sequences that depend on state labels. For instance, we cannot quantify over states reachable by transition sequences that only go through states satisfying some predicate p. Tests allow us to do exactly this. Tests denote partial identity relations on states. The partial identity relation denoted by a test s is defined on all states that satisfy s and undefined everywhere else. With this intuition, we can justify the following two equivalences:

$$\langle s\rangle t \;\equiv\; s \wedge t \qquad\qquad [s]t \;\equiv\; \neg s \vee t$$

For the first equivalence, the justification is as follows. Suppose a state w satisfies $\langle s\rangle t$. Then w has an s-successor satisfying t. Since the transition relation denoted by s is defined on w, w satisfies s. Since s denotes a partial equivalence relation, w is its only successor, and hence, by assumption, satisfies t. Therefore, w satisfies $s \wedge t$. Conversely, it is easy to see that a state satisfying $s \wedge t$ has an s-successor satisfying t.

Remark 2.2.1 In the literature, one often uses a different notation for programs, writing $\alpha \cup \beta$ for $\alpha + \beta$, $\alpha;\beta$ for $\alpha\beta$, and $s?$ for s when s is a test. ◂

By exploiting De Morgan's laws, the equivalence $\neg\neg s \equiv s$ for double negation, and the equivalences $\neg\langle\alpha\rangle s \equiv [\alpha]\neg s$ and $\neg[\alpha]s \equiv \langle\alpha\rangle\neg s$ for modal operators, every formula of PDL can be translated into **negation normal form** where the negation operator is only applied to predicates. Note that both the time needed to perform the translation and the size of the negation normal form are linear in the size of the initial formula.

2.2.2 Semantics

Let us now give the formal semantics of PDL. For this purpose, we fix some notations for binary relations. Let $\to\; \subseteq X \times X$.

$$\begin{array}{lll} \to^0 \;:=\; \{\,(x,x) \mid x \in X\,\} & \to^* \;:=\; \bigcup\limits_{n\geq 0} \to^n & \to^{-1} \;:=\; \{\,(y,x) \mid (x,y) \in \to\,\} \\ \to^{n+1} \;:=\; \to \circ \to^n & \to^+ \;:=\; \to \circ \to^* & \end{array}$$

For all relations shown above we use infix notation (i.e., $x \to^n y$ for $(x,y) \in \to^n$).

A **model** $\mathfrak{M}$ of PDL consists of the following components:

- A nonempty set $|\mathfrak{M}|$ of **states**. We call $|\mathfrak{M}|$ the **domain** of $\mathfrak{M}$.
- A **transition relation** $\xrightarrow{a}_{\mathfrak{M}} \subseteq |\mathfrak{M}| \times |\mathfrak{M}|$ for every action a.
- A set $\mathfrak{M}p \subseteq |\mathfrak{M}|$ for every predicate p.

The transition relations for complex programs and the **satisfaction relation** $\mathfrak{M}, w \vDash s$ between models $\mathfrak{M}$, states $w \in |\mathfrak{M}|$, and formulas s are defined by mutual induction on the structure of formulas and programs as shown in Figure 2.1. The satisfaction relation defines what it means for a state to satisfy a formula. Intuitively, we have $\mathfrak{M}, w \vDash s$ if and only if w is contained in the set of states denoted by s. Another useful intuition is to think of satisfaction as an evaluation function. For every model $\mathfrak{M}$ and every state $w \in |\mathfrak{M}|$, satisfaction defines a function from formulas to truth values. The function evaluates a formula s to `true` if and only if $\mathfrak{M}, w \vDash s$.

$$\begin{aligned}
\xrightarrow{\alpha+\beta}_{\mathfrak{M}} &= \xrightarrow{\alpha}_{\mathfrak{M}} \cup \xrightarrow{\beta}_{\mathfrak{M}} \\
\xrightarrow{\alpha\beta}_{\mathfrak{M}} &= \xrightarrow{\alpha}_{\mathfrak{M}} \circ \xrightarrow{\beta}_{\mathfrak{M}} \\
\xrightarrow{\alpha^*}_{\mathfrak{M}} &= \xrightarrow{\alpha}{}^{*}_{\mathfrak{M}} \\
\xrightarrow{s}_{\mathfrak{M}} &= \{\, (w,w) \mid \mathfrak{M}, w \vDash s \,\}
\end{aligned}$$

$$\begin{aligned}
\mathfrak{M}, w \vDash p &\iff w \in \mathfrak{M}p \\
\mathfrak{M}, w \vDash \neg s &\iff \text{not } \mathfrak{M}, w \vDash s \\
\mathfrak{M}, w \vDash s \vee t &\iff \mathfrak{M}, w \vDash s \text{ or } \mathfrak{M}, w \vDash t \\
\mathfrak{M}, w \vDash s \wedge t &\iff \mathfrak{M}, w \vDash s \text{ and } \mathfrak{M}, w \vDash t \\
\mathfrak{M}, w \vDash \langle\alpha\rangle s &\iff \exists v\colon\ w \xrightarrow{\alpha}_{\mathfrak{M}} v \text{ and } \mathfrak{M}, v \vDash s \\
\mathfrak{M}, w \vDash [\alpha] s &\iff \forall v\colon\ w \xrightarrow{\alpha}_{\mathfrak{M}} v \text{ implies } \mathfrak{M}, v \vDash s
\end{aligned}$$

Figure 2.1: Satisfaction relation for PDL

$$\begin{aligned}
\langle s\rangle t &\equiv s \wedge t & [s]t &\equiv \neg s \vee t \\
\langle \alpha+\beta\rangle s &\equiv \langle\alpha\rangle s \vee \langle\beta\rangle s & [\alpha+\beta]s &\equiv [\alpha]s \wedge [\beta]s \\
\langle \alpha\beta\rangle s &\equiv \langle\alpha\rangle\langle\beta\rangle s & [\alpha\beta]s &\equiv [\alpha][\beta]s \\
\langle \alpha^*\rangle s &\equiv s \vee \langle\alpha\rangle\langle\alpha^*\rangle s & [\alpha^*]s &\equiv s \wedge [\alpha][\alpha^*]s
\end{aligned}$$

Figure 2.2: Semantic equivalences for PDL

Now the notion of satisfiability from § 2.1.2 can be made formal. A model $\mathfrak{M}$ **satisfies** (or is a **model of**) a formula s if $\mathfrak{M}, w \vDash s$ for some state $w \in |\mathfrak{M}|$. A formula is **satisfiable** if it has a model.

Another notion we were so far using intuitively is semantic equivalence. Formally, two formulas s and t are **equivalent**, written $s \equiv t$, if for all models $\mathfrak{M}$ and all $w \in |\mathfrak{M}|$ we have $\mathfrak{M}, w \vDash s \iff \mathfrak{M}, w \vDash t$. Some important equivalences, including the ones we have seen for tests, are given in Figure 2.2.

2.2.3 Decision Methods

The satisfiability problem for PDL is ExpTime-complete. The lower bound was shown by Fischer and Ladner [67] by reduction from linear space bounded alternating Turing machines. Fischer and Ladner also show that PDL is decidable by proving that it has the small model property. More precisely, they show that every satisfiable formula s has a model of size exponential in the size of s. Thus, the satisfiability of a given formula s can be decided by a simple guess-and-check procedure: One guesses a model and checks if it satisfies s. Besides being unpractical, the procedure requires nondeterministic exponential time, and hence does not match Fischer and Ladner's lower complexity bound.

The first worst-case optimal decision procedures for PDL were devised by Pratt [160, 161]. Pratt's approach in [161] was later reformulated by Nguyen and Szałas [151]. Neither Pratt's procedures nor Nguyen and Szałas' procedure are incremental. Incremental but nonoptimal tableau procedures for (variants of) PDL were proposed by Baader [12], De Giacomo and Massacci [52], and Abate et al. [3]. The procedure by Abate et al. is implemented within the Tableau Workbench (TWB) [1] framework. Finally, an incremental and worst-case optimal tableau procedure for PDL was presented by Goré and Widmann [93]. Their implementation of the procedure is currently the arguably most efficient system for PDL (see [122]).

Although PDL is decidable via automata-theoretic methods (as shown by Vardi and Wolper [195]), we are not aware of any implementations explicitly based on automata. This is due to the fact that a naive implementation of automata-based methods is highly nonincremental, usually resulting in worst-case behaviour on all inputs. However, as observed by Emerson and Sistla [62, 56], Gerth et al. [80] or Baader et al. [16], there is a close correspondence between the tableau method and automata-based methods, which often allows to interpret tableau procedures as optimized automata-based procedures.

Game-theoretic decision methods can be seen as a combination of a tableau-based construction phase producing candidate models with an automata-based pruning phase eliminating candidate models that fail to satisfy the input formula. Game-theoretic methods that apply to PDL were proposed by Lange [143] and Cîrstea et al. [45]. A recent implementation of a generic game-theoretic decision procedure by Friedmann and Lange [73] can decide PDL as a special case.

2.2.4 Modal Logic with Eventualities

Note that the equivalences in Figure 2.2 allow us to convert every formula that does not contain programs of the form α^* to an equivalent formula of K. For instance, $\langle ap + b\rangle q \equiv \langle a\rangle(p \wedge q) \vee \langle b\rangle q$. On the other hand, even very simple formulas that use the reflexive-transitive closure like $\langle a^*\rangle p$ cannot be finitely expressed in K. Intuitively, this is the case since formulas of K can see only finitely far ahead.

More formally, given a model $\mathfrak{M}$, a state $w \in |\mathfrak{M}|$ and a number n, we can think of the *n-step environment of w in $\mathfrak{M}$* as a the largest submodel of $\mathfrak{M}$ that contains only states reachable from w by at most n transitions. For every formula of K there is a bound n such that the formula is satisfied by a given model $\mathfrak{M}$ at a state w if and only if it is satisfied by the n-step environment of w in $\mathfrak{M}$ (or by any model that agrees with $\mathfrak{M}$ on its n-step environment of w; see [27], § 2.3 for details and general results). It is easy to see that the formula $\langle a^*\rangle p$ does not have this property. For every number n there is a model $\mathfrak{M}$ and a state w satisfying $\langle a^*\rangle p$ such that every state satisfying p is more than n a-transitions away from w, and hence $\langle a^*\rangle p$ is not satisfied by the n-step environment of w in $\mathfrak{M}$.

In this sense, quantification over reachable states as realized by the operators $\langle a^* \rangle$ and $[a^*]$ can be seen as the distinguishing feature of PDL compared to K. What happens if we try to isolate this feature? We arrive at the language K*. For simplicity, let us restrict ourselves to the unimodal case. There, K* can be obtained by extending K with two additional operators: $\Diamond^*$ and $\Box^*$. The semantics of $\Diamond^*$ is equivalent to that of $\langle a^* \rangle$ and the semantics of $\Box^*$ to that of $[a^*]$, assuming the unique transition relation of unimodal K* is labeled with a.

Since, intuitively, a formula of the form $\Diamond^* s$ means that s needs to hold eventually, we call such formulas **eventualities** (the name can be traced back to Clarke, Emerson and Halpern [44, 58, 59, 57]; Emerson and Halpern [59] study K* under the name UB$^-$). So, K* can be characterized as the **basic modal logic with eventualities**.

It turns out that K* shares essential semantic and computational properties with PDL. Like PDL, K* is not compact (consider the set $\Diamond^* \neg p$, $\Box p$, $\Box\Box p$, $\ldots$) and hence has no semantics-preserving translation into first-order logic. Also, K* is ExpTime-complete. While the upper bound is directly inherited from PDL, the lower bound follows from Fischer and Ladner's original hardness proof for PDL (see [27], Theorem 6.52 and the following remark).

2.2.5 Converse

In the logics considered so far we can quantify over successors in a transition system. What if we want to quantify over predecessors? To add this possibility to PDL, we extend the syntax of programs by a new operator, called converse. The **converse of a program** α, written α^-, denotes the converse of the relation denoted by α. Formally:

$$\xrightarrow{\alpha^-}_{\mathfrak{M}} \;=\; \xrightarrow{\alpha}{}^{-1}_{\mathfrak{M}}$$

Propositional dynamic logic extended with converse is often called **Converse-PDL**. We abbreviate Converse-PDL as PDL$^-$.

Note that converse satisfies the following equations:

$$\begin{array}{rclcrclcrcl}
(\alpha\beta)^- & = & \beta^-\alpha^- & \quad & (\alpha^*)^- & = & (\alpha^-)^* & \quad & (\alpha^-)^- & = & \alpha \\
(\alpha+\beta)^- & = & \alpha^- + \beta^- & & s^- & = & s & & & &
\end{array}$$

The equations suggest that by pushing converse in, every program of PDL$^-$ can be brought into **converse normal form** where the converse operator is only applied to actions.

The **basic modal logic with converse** (written K$^-$) is the fragment of PDL$^-$ obtained by restricting programs to actions a and their converse a^-.

Adding converse does not increase the computational complexity of K or PDL (see, respectively, [111], § 4.5 and [193]). In fact, De Giacomo [51] showed that formulas of PDL$^-$ can be polynomially translated to PDL without converse preserving satisfiability. As it comes to practical tableau procedures, however, adding converse is not trivial. The first practical tableau procedure for PDL$^-$ is presented in recent work by Goré and Widmann [95] (earlier approaches by De Giacomo and Massacci [52] and by

Nguyen and Szałas [150] employ analytic cut rules and hence do not easily lend themselves to efficient implementation). To the best of our knowledge, Goré and Widmann's implementation of their procedure is currently the only system that supports PDL with converse.

Even for simpler logics, obtaining incremental decision procedures that can deal with converse often requires new techniques [114, 111, 29, 127, 94]. While we are not aware of any tableau systems specifically for K^-, there are various provers for stronger logics [99, 189, 109, 182, 94] that can decide K^- as a sublanguage. Also, since converse happens to pose no problems to resolution (see [111]), converse is generally supported by resolution-based provers.

2.3 Hybrid Logic

Hybrid logic is an extension of modal logic that allows to refer to individual states of the model. For background on hybrid logic, see [27, 10, 34].

2.3.1 Nominals

Hybrid logic extends the syntax of modal logic by introducing a second kind of predicates called **nominals**. We denote nominals by the letters x, y and z. In contrast to ordinary propositions, which may be true at arbitrarily many states of the model, a nominal must be true at exactly one state. Formally, we require that for every model $\mathfrak{M}$ and every nominal x, we have $|\mathfrak{M}x| = 1$. So, a nominal x can be seen as a name for the unique state in $\mathfrak{M}x$:

$$\mathfrak{M}, w \models x \iff \mathfrak{M}x = \{w\}$$

In other words, nominals allow us to express equality of states. A state satisfies a nominal x if it is equal to the state denoted by x. With this, we can state properties of transition systems that can otherwise not be expressed in modal logic. For instance, the formula

$$x \wedge \Diamond x$$

is satisfiable if and only if the state named x has a reflexive transition. Such a property is not expressible in K since it has the *tree model property*, meaning that every satisfiable formula of K is satisfied in a tree model (a model whose transition system forms a tree; see [27], § 2.1 or [88], § 2.2).

We define the **basic hybrid logic** (H for short) as K extended with nominals. The logic resulting from adding nominals to PDL we call **hybrid PDL** (**HPDL**).

2.3.2 Satisfaction Operators

Satisfaction operators provide a uniform and convenient means of evaluating formulas at named states. For each nominal x, we extend the syntax by a **satisfaction operator** $x{:}$ that allows us to construct **satisfaction formulas** of the form $x : s$ (often satisfaction operators are also written as $@_x$ and satisfaction formulas as $@_x s$). Intuitively, a formula $x : s$ states that the formula s is true at the state named x. Formally, we require:

$$\mathfrak{M}, w \vDash x : s \iff \mathfrak{M}, v \vDash s \quad \text{where} \quad \mathfrak{M}x = \{v\}$$

Note that the value of a satisfaction formula does not depend on the point of evaluation: $x : s$ is true at one state of the model if and only if it is true everywhere. Note also that satisfaction operators are self-dual: $\neg(x : s) \equiv x : \neg s$. Thus, the computation of negation normal forms naturally extends to formulas with satisfaction operators.

2.3.3 Complexity and Decision Methods

The additional expressiveness provided by nominals and satisfaction operators does not increase the computational complexity of K or PDL. Hybrid PDL extended with satisfaction operators is still ExpTime-complete [155] and H with satisfaction operators is still PSpace-complete [4]. Interestingly, adding satisfaction operators to H^- (i.e., H with converse) results in a jump in complexity to ExpTime-completeness (see § 2.4.1).

Tableau procedures based on tree search are arguably the most successful approach to deciding hybrid logic. Incremental tree procedures for hybrid logic were devised by Bolander and colleagues [30, 29]. Different variants and refinements of the procedures are studied in [126, 127, 39, 107], and implemented in the provers HTab [109], Spartacus [97, 98], Herod and Pilate [42]. Tableau procedures for hybrid extensions of description logics (see § 2.5.2) were developed by Horrocks and Sattler [115, 117] and implemented in systems like FaCT++ [189] or Pellet [182].

While tree procedures for hybrid logic have proven successful in practice, none of the existing approaches has optimal complexity. Recent work by Goré and Widmann [93, 94, 95, 200] suggests that practical and worst-case optimal procedures for expressive modal logics can be obtained based on graph search. However, as observed in [128, 200], obtaining worst-case optimal graph procedures for hybrid logic is a nontrivial task.

Besides tableau procedures, a notable decision method for hybrid logic is resolution. Resolution for hybrid logic, proposed by Areces et al. [7] and further developed by Areces and Gorin [96, 8], is implemented in the prover HyLoRes [9].

2.4 Global and Difference Modalities

2.4.1 Global Modalities

Analogously to how ordinary modal operators allow talking about the successors of a state in a transition system, global modalities allow to quantify over *all* states of the transition system. Global modalities comprise of two modal operators: the **existential modality** E and its dual, the **universal modality** A. A formula Es means that s is satisfied in at least one state of the model, while As means that s holds in every state of the model. Formally, we define:

$$\begin{aligned}\mathfrak{M}, w \vDash \mathsf{E}s &\iff \exists v \in |\mathfrak{M}|:\ \mathfrak{M}, v \vDash s\\ \mathfrak{M}, w \vDash \mathsf{A}s &\iff \forall v \in |\mathfrak{M}|:\ \mathfrak{M}, v \vDash s\end{aligned}$$

Analogously to ordinary modal operators, global modalities satisfy the equivalence $\mathsf{A}s \equiv \neg\mathsf{E}\neg s$. So, E and A are nothing but ordinary modal operators, but defined over the universal relation (i.e., the Cartesian product of the states) instead of an ordinary transition relation.

Hence, instead of enriching the syntax by additional operators, we could alternatively introduce a special relation U (for “universal relation”), and require, for every model $\mathfrak{M}$, that $\xrightarrow{U}_{\mathfrak{M}} = |\mathfrak{M}| \times |\mathfrak{M}|$. Then E would be semantically equivalent to $\langle U\rangle$ and A equivalent to $[U]$. This approach is taken, for instance, by Passy and Tinchev [155] in the context of PDL or by Horrocks et al. [112] in the context of description logics (see § 2.5.2).

Given a modal logic L, we write L_{E} for the logic obtained by extending L with global modalities.

Global modalities significantly increase the expressiveness of K by allowing to state global assumptions and constraints on models. For instance, the formula $\mathsf{A}\Diamond(p \vee \neg p)$ requires that every state of the transition system has a successor. In other words, it enforces the transition system to be serial.

As shown by Spaan [185], the additional expressiveness provided by global modalities comes at a price. Adding global modalities to K results in an ExpTime-complete logic. Adding global modalities to PDL (or HPDL), which is already ExpTime-complete, does not further increase the complexity [155].

In combination with nominals, global modalities can express satisfaction operators: $x : s \equiv \mathsf{E}(x \wedge s)$ (see [5]). More interestingly, Areces et al. [4] showed, using Blackburn and Seligmann’s [28] spy-point technique, that H^- can simulate universal quantification over all states as provided by the universal modality. Since universal quantification over all states is responsible for the ExpTime-completeness of K_{E}, satisfiability in H^- is also ExpTime-complete.

2.4.2 Difference Modalities

Difference modalities are powerful constructs that combine the expressiveness of global modalities with that of nominals. The **existential difference modality** D and its universal dual $\bar{\mathsf{D}}$ quantify over states that are distinct from the point of evaluation. Intuitively, a formula $\mathsf{D}s$ means that s holds “somewhere else”. Dually, a formula $\bar{\mathsf{D}}s$ means that s holds “everywhere else”. Formally, we have:

$$\begin{aligned}\mathfrak{M}, w \vDash \mathsf{D}s &\iff \exists v \in |\mathfrak{M}|:\ v \neq w \ \text{ and } \mathfrak{M}, v \vDash s\\ \mathfrak{M}, w \vDash \bar{\mathsf{D}}s &\iff \forall v \in |\mathfrak{M}|:\ v \neq w \ \text{ implies } \mathfrak{M}, v \vDash s\end{aligned}$$

Clearly, $\bar{\mathsf{D}}s \equiv \neg\mathsf{D}\neg s$. Given a logic L, we write L_{D} for the logic obtained by extending L with difference modalities.

Difference modalities were studied by von Wright [196] and Segerberg [179] as isolated modalities. Their expressive power and possible applications as additional modalities were later investigated in [170, 86, 134, 54, 76].

Difference modalities can express global modalities: $\mathsf{E}s \equiv s \vee \mathsf{D}s$ and $\mathsf{A}s \equiv s \wedge \bar{\mathsf{D}}s$. At the same time, they can express nominals. A hybrid formula s can be translated into modal logic with difference modalities by conjunctively adjoining a formula $\mathsf{E}(x \wedge \bar{\mathsf{D}}\neg x)$ for every nominal x that occurs in s (where we use $\mathsf{E}t$ as an abbreviation for $t \vee \mathsf{D}t$). The formula $\mathsf{E}(x \wedge \bar{\mathsf{D}}\neg x)$ ensures that the predicate x is interpreted as a nominal (i.e., is satisfied at exactly one state).

As shown by Gargov and Goranko [76] (see also [5]), there exists a polynomial satisfiability-preserving translation from modal logic with difference modalities to hybrid logic with global modalities. For this reason, all logics that include difference modalities, beginning with K_{D} and including $\mathrm{HPDL}^{-}_{\mathsf{D}}$ (i.e., hybrid Converse-PDL with difference modalities), have the same complexity as their counterparts with global modalities, namely ExpTime (the ExpTime-completeness of $\mathrm{HPDL}^{-}_{\mathsf{E}}$ was shown by Areces et al. [5]).

The first prefixed tableau procedure for hybrid logic with difference modalities is presented in [126] and extended to converse in [125, 127]. The techniques developed in [126, 125, 127] to deal with difference modalities were adapted by Hoffmann [108, 107] to tableau procedures in the style of [30, 29].

2.5 Related Logics

2.5.1 Graded Modal Logics

Graded modal logic is an expressive extension of ordinary modal logic obtained by generalizing modal operators to cardinality constraints. A formula $\Diamond s$ of ordinary modal logic states that s holds in at least one successor state. Clearly, this is equivalent to saying that s holds in more than 0 successors. A formula $\Diamond_n s$ (where $n \in \mathbb{N}$) of graded modal logic states that s holds in more than n successors. Dually, the formula $\Box_n s$ means that s holds in all but at most n successors (i.e., $\neg s$ holds in at most n successors). It is easily seen that $\Diamond s \equiv \Diamond_0 s$, $\Box s \equiv \Box_0 s$ and $\Box_n s \equiv \neg\Diamond_n\neg s$. Thus, graded modal operators correspond to counting first-order quantifiers in the same way as ordinary modal operators correspond to ordinary quantifiers.

Graded modal logic in its modern form was introduced by Fine [65] and then rediscovered by Fattorosi-Barnaba and De Caro [64]. As shown by Tobies [187] (and earlier by van Der Hoek and de Rijke [192] for unary coding of numbers in the input), the basic modal logic extended with graded modalities is still PSpace-complete. A comprehensive investigation into the complexity of graded modal logics can be found in [131].

Tableau procedures based on tree search have been developed for various expressive (syntactic variants of) graded logics in [118, 115, 117, 112, 129, 124]. While graded modalities allow a natural treatment via tree search using a prefixed representation of candidate models, as it comes to graph tableaux and a clausal representation, they pose a considerable challenge. Currently, no graph procedures for graded modal logic are known. Adapting the graph tableau approach to graded modal logic is conjectured nontrivial [200].

2.5.2 Description Logics

Description logics are a family of knowledge representation languages that were developed from earlier formalisms like semantic networks [166] or frame systems [147]. Description logics allow to reason about ontological data in terms of **concepts**, which are structured descriptions for sets of ontological objects. For a comprehensive introduction to description logics, see [15, 18].

While initially developed independently from modal logic, description logics are now known to be syntactic variants of modal logic [171]. In particular, the basic description logic $\mathcal{ALC}$ [174], which forms the basis for modern expressive logics like $\mathcal{SROIQ}$ [112], is equivalent, up to notation, to multimodal K. The syntax for concepts in $\mathcal{ALC}$ is given by the grammar

$$C ::= A \mid \neg C \mid C \sqcup C \mid C \sqcap C \mid \exists R.C \mid \forall R.C$$

where A ranges over a nonempty set of **atomic concepts** and R ranges over a nonempty set of **roles**. Now, the correspondence to the usual modal syntax is straightforward. Atomic concepts correspond to predicates and roles to actions. Concepts of the form $\neg C$, $C \sqcup C$, and $C \sqcap C$ correspond to negations, disjunctions, and conjunctions, respectively. Finally, a concept $\exists R.C$ ($\forall R.C$) corresponds to the formula $\langle a \rangle s$ ($[a]s$) where a is the action corresponding to R and s the formula corresponding to C.

Due to the immediate correspondence between modal and description logics, results obtained for modal logics often equally apply to description logics and vice versa. When this is the case, we tend not to distinguish between modal and description logics. For instance, we consider Horrocks and Sattler's [115] tableau algorithm for the description logic $\mathcal{SHOQ}$ as a decision procedure for graded hybrid logic with global modalities since formulas of this logic can be translated to $\mathcal{SHOQ}$.

The development in description logics was always primarily driven by applications. This resulted in the development of very expressive description logics like $\mathcal{SROIQ}$, which contains features like converse, nominals, global and graded modalities, while at the same time including features that have no immediate counterparts in modal logic. For $\mathcal{SROIQ}$, such a feature are **role inclusion axioms**, which are inclusion assertions of the form $R_1 \dots R_n \sqsubseteq R$ (with some additional restrictions; see [114, 116, 112]) specifying that the composition of the relations denoted by $R_1, \dots, R_n$ is contained in the relation denoted by R.

More recently, considerable interest re-emerged in *lightweight* description logics, in which many reasoning tasks are tractable. While the expressiveness of such logics is restricted to achieve tractability, it still suffices for practical applications (see [14]). Current research on lightweight description logics concentrates around two main families of logics: the $\mathcal{EL}$ family initially studied by Baader and colleagues [17, 13, 33, 14] and the DL-Lite family proposed by Calvanese et al. [37, 38].

2.5.3 Temporal Logics and Modal μ-Calculus

Temporal logics interpret relational models as consisting of points in time that are related to each other by an 'earlier-later' relation. Depending on the structure of admissible models, one can classify temporal logics into linear-time and branching-time logics.

The most common linear-time logic, **propositional linear-time logic** (LTL), was introduced in its modern form by Pnueli and colleagues [158, 75] as a framework for specifying correctness properties of programs. In LTL, formulas are interpreted over infinite time sequences $s_0, s_1, \dots$ where each state s_i describes the situation at the point i in time. In other words, models of LTL are serial transition systems where the transitive closure of the transition relation forms a linear order on states.

Unlike linear-time logics, branching-time logics allow a state to have more than one successor, which is intended to capture nondeterminism. Influenced by the work of Ben-Ari et al. [22] on the branching-time logic UB, Clarke and Emerson [44] introduced **computational tree logic** (CTL). Emerson and Halpern [60] showed that LTL and CTL are incomparable with respect to expressiveness (see also [43]) and introduced the logic CTL*, which directly subsumes both LTL and CTL. Thus, CTL* unifies linear-time and branching-time logics. The expressiveness of CTL* comes at the expense of its complexity. While LTL is PSpace-complete [183] and CTL is ExpTime-complete [59], CTL* is 2ExpTime-complete [194, 61]. In parallel to the work on temporal logics, Kozen [135] introduced the **modal μ-calculus** as an expressive program logic based on earlier work by Scott, De Bakker, Park, Pratt and others. While being ExpTime-complete [61], in its expressiveness the modal μ-calculus subsumes CTL* [47] (the translation from CTL* into the μ-calculus being doubly exponential in the worst case).

Besides automata-theoretic methods, graph tableaux have long been a method of choice for deriving decision procedures for temporal logics. Graph procedures have been developed for both for LTL [201, 132] and CTL [58, 59]. Since K* is a sublanguage of UB (see [59] for a comparison), PDL is closely related to branching-time temporal logics. In the same sense as Pratt's procedures for PDL, graph procedures for temporal logics are not incremental and hence unpractical.

More recently, incremental tree procedures were developed for LTL [177], CTL [2], and even CTL* [169] and the modal μ-calculus [123]. As is often the case with tree procedures, however, none of the systems is worst-case optimal. An even more recent approach by Friedmann and Lange [74] for the μ-calculus is worst-case optimal but lacks incrementality. Thus, obtaining incremental but at the same time optimal procedures for expressive temporal logics or the μ-calculus remains a challenging open problem.

3 Demos and Pruning

This chapter introduces the notion of a demo and a technique for constructing demos that we call pruning. Using demos and pruning, we give a modular construction of worst-case optimal decision procedures for extensions of PDL and Converse-PDL with nominals and difference modalities.

Our approach is based on Pratt's [160] worst-case optimal decision procedure for PDL. Variants of Pratt's procedure can be found with correctness proofs in [103, 138, 104, 27]. Given a formula s, Pratt's procedure constructs a model $\mathfrak{M}$ of all satisfiable formulas within the finite formula universe given by the Fischer-Ladner closure of s. It returns satisfiable if s is satisfied by $\mathfrak{M}$ and unsatisfiable otherwise.

Our view of Pratt's procedure is as follows. The procedure works with candidate models that use Hintikka sets as states. First, the procedure constructs an initial candidate model $\mathcal{H}$ consisting of all Hintikka subsets of the formula universe. It then proceeds by pruning $\mathcal{H}$, i.e., by stepwise removing from $\mathcal{H}$ Hintikka sets that contain unsatisfied diamond formulas. Pruning terminates with a demo satisfying all satisfiable formulas from the formula universe.

We extend Pratt's procedure to nominals and difference modalities, eventually obtaining a worst-case optimal decision procedure for HPDL_D^-. We observe that in the presence of nominals or difference modalities, Pratt-style pruning becomes incomplete. To deal with this problem, we introduce an additional *guessing* stage that is performed after the construction of the initial candidate model but before pruning. What we guess is a subset of all constructed Hintikka sets that is consistent with the semantics of nominals and difference modalities. The desired demo is then obtained by pruning the guessed subset. We argue that this approach preserves worst-case optimality by showing that the number of consistent subsets that can be selected by the guessing stage is exponential in the size of the input formula.

The approach is presented in a modular way. We begin with a decision procedure for test-free PDL and extend it stepwise to deal with nominals, difference modalities, tests, and converse. We show that all of these extensions are modular in the sense that their combination requires no more effort than the combined cost of adding each of the features separately. We refactor standard proofs to achieve a clean separation of syntax and semantics and to use simple induction on terms.

Structure of the chapter. We begin by introducing alpha-beta formula decomposition rules for PDL, which we then use to define Hintikka sets (§ 3.1) and to introduce the notion of a formula universe (§ 3.2). We then introduce demos as a syntactic class of models (§ 3.3). After formulating pruning and the decision procedure for test-free PDL (§ 3.4), we discuss the extensions necessary to deal with nominals (§ 3.5), difference

α	α_1	α_2
$s \wedge t$	s	t
$\langle s \rangle t$	s	t
$[\alpha + \beta]s$	$[\alpha]s$	$[\beta]s$
$[\alpha\beta]s$	$[\alpha][\beta]s$	$[\alpha][\beta]s$
$[\alpha^*]s$	s	$[\alpha][\alpha^*]s$

β	β_1	β_2
$s \vee t$	s	t
$[s]t$	$\sim s$	t
$\langle \alpha + \beta \rangle s$	$\langle \alpha \rangle s$	$\langle \beta \rangle s$
$\langle \alpha\beta \rangle s$	$\langle \alpha \rangle \langle \beta \rangle s$	$\langle \alpha \rangle \langle \beta \rangle s$
$\langle \alpha^* \rangle s$	s	$\langle \alpha \rangle \langle \alpha^* \rangle s$

Figure 3.1: Alpha and beta formulas

modalities (§ 3.6), tests (§ 3.7), and converse (§ 3.8). We conclude with a discussion of related work (§ 3.9).

3.1 Formulas and Hintikka Systems

For simplicity of presentation, we consider only formulas in negation normal form and only programs in converse normal form. In other words, we assume the syntax of formulas and programs of $\text{HPDL}_{\text{D}}^{-}$ to be given by the grammar

$$\alpha ::= a \mid a^- \mid \alpha + \alpha \mid \alpha\alpha \mid \alpha^* \mid s$$

$$\begin{aligned} s ::=\ & p \mid s \vee s \mid \langle \alpha \rangle s \mid \mathsf{D}s \\ \mid\ & \neg p \mid s \wedge s \mid [\alpha]s \mid \bar{\mathsf{D}}s \end{aligned}$$

where p ranges over both ordinary predicates and nominals. We call formulas of the form p, $\neg p$, $\langle a \rangle s$, $[a]s$, $\langle a^- \rangle s$, $[a^-]s$, $\mathsf{D}s$ and $\bar{\mathsf{D}}s$ **literals**.

Following Smullyan [184], we observe that every nonliteral formula can be viewed as a **conjunctive formula** $\alpha \equiv \alpha_1 \wedge \alpha_2$ or as a **disjunctive formula** $\beta \equiv \beta_1 \vee \beta_2$ (see also [70, 57, 71]). The partition of nonliteral formulas into **alpha** and **beta** formulas is shown in Figure 3.1. Note that this so-called **unifying notation** conflicts with our notation for programs, which we also denote by α and β. In the following, we will make the intended interpretation of the letters α and β clear whenever we want to use the unifying notation. Every line of the tables in Figure 3.1 can be seen as a **decomposition rule** mapping a formula α (β) to two **constituents** α_1 and α_2 (β_1 and β_2).

Since $\langle \alpha\beta \rangle s \equiv \langle \alpha \rangle \langle \beta \rangle s \equiv \langle \alpha \rangle \langle \beta \rangle s \wedge \langle \alpha \rangle \langle \beta \rangle s \equiv \langle \alpha \rangle \langle \beta \rangle s \vee \langle \alpha \rangle \langle \beta \rangle s$, a formula $\langle \alpha\beta \rangle s$ can be classified both as alpha and as beta. The same holds for formulas $[\alpha\beta]s$. For consistency of the presentation we classify boxes $[\alpha\beta]s$ as alpha formulas and diamonds $\langle \alpha\beta \rangle s$ as beta formulas.

One case in Figure 3.1 requires additional explanation, namely the one for box formulas of the form $[s]t$. There, we use the notation $\sim s$ for the negation normal form of $\neg s$ (this notation being used by Hintikka [105] for first-order logic). We need normalizing negation because we are restricting ourselves to negation normal formulas. Assuming $[s]t$ is in negation normal form, the equivalence $[s]t \equiv \sim s \vee t$ contains only negation normal formulas. We should remark that box formulas $[s]t$ are the only reason why we need normalizing negation. In particular, normalizing negation is not needed for test-free PDL.

We define a **Hintikka set** as a set H of formulas that satisfies the following properties.

- For every predicate p: $\{p, \neg p\} \nsubseteq H$.
- If a formula $\alpha \in H$, then $\alpha_1 \in H$ and $\alpha_2 \in H$.
- If a formula $\beta \in H$, then $\beta_1 \in H$ or $\beta_2 \in H$.

A **Hintikka system** S is a finite set of Hintikka sets. A formula s is **contained in** S if s is contained in some $H \in S$. Candidate models constructed by our approach will be represented as Hintikka systems. Demos will be defined as a class of Hintikka systems.

We extend the satisfaction relation to sets A of formulas, writing $\mathfrak{M}, w \models A$ if $\mathfrak{M}, w \models s$ for every $s \in A$. A model $\mathfrak{M}$ **satisfies a set** A **of formulas** if $\mathfrak{M}, w \models A$ for some $w \in |\mathfrak{M}|$. A model $\mathfrak{M}$ **satisfies a Hintikka system** S (or **is a model of** S) if $\mathfrak{M}$ satisfies all of its elements.

3.2 Formula Universe

All of our constructions will be carried out within a finite formula universe induced by a variant of the Fischer-Ladner closure of the input formula (see [67, 138, 104]). This provides an exponential upper bound on the size of Hintikka systems that need to be considered. This bound is later exploited to show the termination and worst-case optimality of our procedures.

A **formula universe** is a finite, nonempty set $\mathcal{F}$ of formulas satisfying the following closure properties:

- If $s \in \mathcal{F}$ and t is a subformula of s, then $t \in \mathcal{F}$.
- If $\alpha \in \mathcal{F}$, then $\{\alpha_1, \alpha_2\} \subseteq \mathcal{F}$.
- If $\beta \in \mathcal{F}$, then $\{\beta_1, \beta_2\} \subseteq \mathcal{F}$.

Note that the first closure property overlaps to some extent with the other two since formulas α_1, α_2 (β_1, β_2) are often subformulas of α (β).

Since every formula contains at least one predicate (as a subformula) and $\mathcal{F}$ is nonempty and subformula closed, $\mathcal{F}$ always contains at least one predicate. For simplicity of presentation, we further assume that $\mathcal{F}$ always contains at least one nominal whenever we look at a logic with nominals.

Remark 3.2.1 Our closure properties differ from those in [67, 138, 104] for the Fischer-Ladner closure in two minor aspects:

1. Our closure is defined on negation normal formulas that have all of the operators $\vee$, $\wedge$, $\langle\alpha\rangle$ and $[\alpha]$ as syntactic primitives while the Fischer-Ladner closure is defined on arbitrary formulas that have only one binary propositional connective and one modal operator.
2. The Fischer-Ladner closure does not necessarily contain $\sim s$ (or $\neg s$) whenever it contains $[s]t$. This is easily changed by slightly extending the closure, additionally closing under single negations (see, for instance, [27], § 4.8). This extension preserves all relevant properties of the closure. ◂

Remark 3.2.2 It is possible to further restrict the closure properties of a formula universe by replacing the first condition (closure under subformulas) with

- If $\langle a\rangle s \in \mathcal{F}$ or $[a]s \in \mathcal{F}$, then $s \in \mathcal{F}$.

To see a difference between the two definitions, consider the formula $[p]q$. The smallest formula universe containing this formula is $\{[p]q,\ \neg p,\ q,\ p\}$, where $\neg p$ and q are required by the closure under beta decomposition and p (and q) by the closure under subformulas. If we weaken the closure under subformulas as suggested above, it will no longer apply and hence $\{[p]q,\ \neg p,\ q\}$ will be the smallest formula universe containing $[p]q$.

Since all of the decision procedures in the thesis can be shown to stay within the smaller closure, the restricted definition of a formula universe would suffice to argue their termination and complexity. We use the larger closure since it saves us from having to state and prove some technical properties of formula universes that are nonobvious with the restricted closure but clearly hold with the full subformula closure. ◂

Let X be a set. We write $|X|$ for the cardinality of X. As argued in [67, 138, 104], the cardinality of the Fischer-Ladner closure of a formula s is linear in the size of s. This argument can be adapted to our case (the proof seems straightforward but tedious):

Proposition 3.2.3 For every formula s, one can compute a formula universe $\mathcal{F}$ such that $s \in \mathcal{F}$ and $|\mathcal{F}|$ is linear in the size of s.

Hence, for simplicity of our termination and complexity arguments, we reformulate the satisfiability problem as follows:

Given a formula universe $\mathcal{F}$ and a formula $s \in \mathcal{F}$, is s satisfiable?

This way, the formula universe becomes part of the input for the satisfiability problem.

In the following, we assume $\mathcal{F}$ to be a global parameter and tacitly assume $s \in \mathcal{F}$ and $A \subseteq \mathcal{F}$ for every formula s and every set of formulas A. Thus, a set of formulas is always finite, and every Hintikka system is a subset of $2^{\mathcal{F}}$, thus containing at most exponentially many sets with respect to $|\mathcal{F}|$.

3.3 Demos

We define demos as Hintikka systems satisfying an additional criterion. Demos provide a convenient syntactic representation of models that allows for purely syntactic, modular and extensible correctness proofs for our decision procedures.

We show that every demo corresponds to a model that has exactly the Hintikka sets of the demo as states, with every Hintikka set satisfying all of its formulas. Conversely, we show that every satisfiable formula s is contained in a demo.

Since the cardinality of a demo is at most exponential in that of the formula universe $\mathcal{F}$, and $\mathcal{F}$ can be chosen linear in s, the two directions taken together constitute a small model theorem.

In this section, we introduce demos for test-free PDL. The extensions to nominals, difference modalities, tests and converse will be discussed separately in §§ 3.5–3.8.

3.3.1 Definition and Example

Not every Hintikka system has a model. For instance, the system $\{\{\langle a\rangle(p \wedge \neg p)\}\}$ is clearly unsatisfiable because the formula $\langle a\rangle(p \wedge \neg p)$ requires the existence of a state satisfying $p \wedge \neg p$. The definition of demos should include only satisfiable Hintikka systems. To achieve this, we require that in a demo every diamond formula must be *realized* in the following sense. If $\mathcal{D}$ is a demo and $\langle\alpha\rangle s \in H \in \mathcal{D}$, then $\mathcal{D}$ must contain some H' such that $s \in H'$ and, moreover, one can connect H and H' by an α-transition without violating any box formulas. Let us first see how we can express the latter property.

For every program α, every Hintikka system S induces a maximal transition relation $\xrightarrow{\alpha}_S$ that contains all α-transitions between elements of S that are compatible with their box formulas.

Definition 3.3.1 Let S be a Hintikka system. The **transition relation** $\xrightarrow{\alpha}_S \subseteq S \times S$ is defined by induction on α as follows.

$$\begin{aligned}
\xrightarrow{a}_S &= \{\,(H,H') \in S \times S \mid \forall s\colon\ [a]s \in H \implies s \in H'\,\}\\
\xrightarrow{\alpha+\beta}_S &= \xrightarrow{\alpha}_S \cup \xrightarrow{\beta}_S\\
\xrightarrow{\alpha\beta}_S &= \xrightarrow{\alpha}_S \circ \xrightarrow{\beta}_S\\
\xrightarrow{\alpha^*}_S &= \xrightarrow{\alpha}{}^*_S
\end{aligned}$$

◂

Note that $\xrightarrow{\alpha}_S$ is monotone in S in the following sense.

Proposition 3.3.2 Let $S \subseteq S'$ be Hintikka systems. Then $\xrightarrow{\alpha}_S \subseteq \xrightarrow{\alpha}_{S'}$.

Proof Straightforward induction on α. ■

Once we have $\xrightarrow{\alpha}_S$, the definition of a demo is straightforward. We call a Hintikka set $H \in S$ **◇-admissible** (in S) if for every $\langle\alpha\rangle s \in H$ there is some $H' \in S$ such that $H \xrightarrow{\alpha}_S H'$ and $s \in H'$. A Hintikka system is ◇-admissible if so are all of its sets.

Definition 3.3.3 (Demo) Nonempty and ◇-admissible Hintikka systems are called **demos**. ◂

Remark 3.3.4 For our results to hold, it would suffice to work with a weaker notion of ◇-admissibility that requires the existence of an α-accessible state H' containing s only for diamonds $\langle\alpha\rangle s$ where α is an action ($\alpha = a$) or an iteration ($\alpha = \beta^*$). The remaining cases would then follow by the conditions for Hintikka sets. We do not make this restriction for simplicity of presentation. ◂

Example 3.3.5 Below, we show a graphical representation of a demo $\mathcal{D}$ consisting of three Hintikka sets: $H_1 = \{\langle a^*\rangle p,\ \langle a\rangle\langle a^*\rangle p,\ [a]\neg p\}$, $H_2 = \{\langle a^*\rangle p,\ \langle a\rangle\langle a^*\rangle p,\ \neg p\}$ and $H_3 = \{p\}$. The arrows between the sets depict the transition relation $\xrightarrow{a}_{\mathcal{D}}$.

$$\begin{tikzcd}
\boxed{\langle a^*\rangle p,\ \langle a\rangle\langle a^*\rangle p,\ [a]\neg p} \arrow[r, bend left=10] & \boxed{\langle a^*\rangle p,\ \langle a\rangle\langle a^*\rangle p,\ \neg p} \arrow[l, bend left=10] \arrow[loop above] \arrow[r, bend left=10] & \boxed{p} \arrow[l, bend left=10] \arrow[loop above] \arrow[ll, bend left=30] \\
H_1 & H_2 & H_3
\end{tikzcd}$$

Note the following facts:

- We have $H_2 \xrightarrow{a}_{\mathcal{D}} H_2$ since H_2 contains no box formulas.
- We do not have $H_1 \xrightarrow{a}_{\mathcal{D}} H_1$ since $[a]\neg p \in H_1$ but $\neg p \notin H_1$.
- We have $H_1 \xrightarrow{a^*}_{\mathcal{D}} H_3$ since $H_1 \xrightarrow{a}_{\mathcal{D}} H_2$ and $H_2 \xrightarrow{a}_{\mathcal{D}} H_3$.
- The set H_1 is $\Diamond$-admissible since
  1. $H_1 \xrightarrow{a^*}_{\mathcal{D}} H_3$ and $p \in H_3$ (condition for $\langle a^*\rangle p \in H_1$),
  2. $H_1 \xrightarrow{a}_{\mathcal{D}} H_2$ and $\langle a^*\rangle p \in H_2$ (condition for $\langle a\rangle\langle a^*\rangle p \in H_1$). ◂

3.3.2 Demo Satisfaction

We now show that every demo is satisfiable. First, observe that every nonempty Hintikka system S induces a model $\mathfrak{M}_S$ as follows:

$$\begin{aligned}
|\mathfrak{M}_S| &= S \\
\xrightarrow{a}_{\mathfrak{M}_S} &= \xrightarrow{a}_S \\
\mathfrak{M}_S p &= \{\, H \in S \mid p \in H \,\}
\end{aligned}$$

The model $\mathfrak{M}_S$ induced by some Hintikka system S does not necessarily satisfy S, even if S is satisfiable (consider, for instance, $S = \{\{\Diamond p\}\}$). This is different for demos. As we show in the following, $\mathfrak{M}_{\mathcal{D}}$ satisfies $\mathcal{D}$ whenever $\mathcal{D}$ is a demo. For this we need two lemmas.

Lemma 3.3.6 Let S be a Hintikka system and α a program. Then $\xrightarrow{\alpha}_S = \xrightarrow{\alpha}_{\mathfrak{M}_S}$.

Proof Let S and α be as required. We only show the inclusion $\xrightarrow{\alpha}_S \subseteq \xrightarrow{\alpha}_{\mathfrak{M}_S}$ (the other inclusion follows analogously). The proof proceeds by induction on α. For $\alpha = a$, the claim is immediate by the definition of $\mathfrak{M}_S$. The remaining cases are as follows.

Let $\alpha = \beta + \gamma$. Suppose $H \xrightarrow{\beta+\gamma}_S H'$. Then $H \xrightarrow{\beta}_S H'$ or $H \xrightarrow{\gamma}_S H'$. By the inductive hypothesis, we obtain $H \xrightarrow{\beta}_{\mathfrak{M}_S} H'$ or $H \xrightarrow{\gamma}_{\mathfrak{M}_S} H'$, and hence $H \xrightarrow{\beta+\gamma}_{\mathfrak{M}_S} H'$.

Let $\alpha = \beta\gamma$. Suppose $H \xrightarrow{\beta\gamma}_S H'$. Then there is some $H'' \in S$ such that $H \xrightarrow{\beta}_S H'' \xrightarrow{\gamma}_S H'$. By the inductive hypothesis, $H \xrightarrow{\beta}_{\mathfrak{M}_S} H'' \xrightarrow{\gamma}_{\mathfrak{M}_S} H'$ and hence $H \xrightarrow{\beta\gamma}_{\mathfrak{M}_S} H'$.

Let $\alpha = \beta^*$. Suppose $H \xrightarrow{\beta^*}_S H'$, i.e., $H \xrightarrow{\beta}{}^n_S H'$ for some $n \geq 0$. We proceed by induction on n. The case $n = 0$ is trivial. If $n > 0$, we have $H \xrightarrow{\beta}_S H'' \xrightarrow{\beta}{}^{n-1}_S H'$ for some $H'' \in S$. By the outer and the inner inductive hypothesis, we obtain $H \xrightarrow{\beta}_{\mathfrak{M}_S} H''$ and $H'' \xrightarrow{\beta}{}^{n-1}_{\mathfrak{M}_S} H'$, respectively. Hence, $H \xrightarrow{\beta}{}^*_{\mathfrak{M}_S} H'$. ∎

Note that in the following, we will only need the direction $\xrightarrow{\alpha}_S \subseteq \xrightarrow{\alpha}_{\mathfrak{M}_S}$ of Lemma 3.3.6.

Lemma 3.3.7 Let S be a Hintikka system, $\{H, H'\} \subseteq S$, and $[\alpha]s \in H \xrightarrow{\alpha}_{\mathfrak{M}_S} H'$. Then $s \in H'$.

Proof Let S, H, H' and $[\alpha]s$ be as required. We proceed by induction on α. For $\alpha = a$, the claim is immediate by the definition of $\mathfrak{M}_S$. The remaining cases are as follows.

Let $\alpha = \beta + \gamma$. Since H is a Hintikka set, we have $[\beta]s \in H$ and $[\gamma]s \in H$. Moreover, we have $H \xrightarrow{\beta}_{\mathfrak{M}_S} H'$ or $H \xrightarrow{\gamma}_{\mathfrak{M}_S} H'$. In either case, $s \in H'$ follows by the inductive hypothesis.

Let $\alpha = \beta\gamma$. Then there is some $H'' \in S$ such that $H \xrightarrow{\beta}_{\mathfrak{M}_S} H'' \xrightarrow{\gamma}_{\mathfrak{M}_S} H'$. Moreover, since H is a Hintikka set, we have $[\beta][\gamma]s \in H$. Applying the inductive hypothesis to β and then γ yields $[\gamma]s \in H''$ and $s \in H'$.

Let $\alpha = \beta^*$. Since H is a Hintikka set, we have (a) $s \in H$ and (b) $[\beta][\beta^*]s \in H$. Since $H \xrightarrow{\beta^*}_{\mathfrak{M}_S} H'$, we have $H \xrightarrow{\beta}{}^{n}_{\mathfrak{M}_S} H'$ for some $n \geq 0$. We proceed by induction on n. The case $n = 0$ is immediate by (a). If $n > 0$, there is some $H'' \in S$ such that $H \xrightarrow{\beta}_{\mathfrak{M}_S} H'' \xrightarrow{\beta}{}^{n-1}_{\mathfrak{M}_S} H'$. By (b) and the outer inductive hypothesis for β, we have $[\beta^*]s \in H''$. The claim, $s \in H'$, follows by the inner inductive hypothesis for $n - 1$. ∎

Lemma 3.3.8 Let $\mathcal{D}$ be a demo. Then $\forall H \in \mathcal{D}$: $\mathfrak{M}_{\mathcal{D}}, H \vDash H$.

Proof Let $\mathcal{D}$ be a demo and $s \in H \in \mathcal{D}$. We show $\mathfrak{M}_{\mathcal{D}}, H \vDash s$ by induction on s. The case $s = p$ follows by the definition of $\mathfrak{M}_{\mathcal{D}}p$. The Boolean cases are straightforward. The remaining cases are as follows.

Let $s = \langle\alpha\rangle t$. Since H is $\Diamond$-admissible, there is a set $H' \in \mathcal{D}$ with $H \xrightarrow{\alpha}_{\mathcal{D}} H'$ and $t \in H'$. By the inductive hypothesis this yields $\mathfrak{M}_{\mathcal{D}}, H' \vDash t$, while by Lemma 3.3.6 we have $H \xrightarrow{\alpha}_{\mathfrak{M}_{\mathcal{D}}} H'$. Consequently, $\mathfrak{M}_{\mathcal{D}}, H \vDash \langle\alpha\rangle t$.

Let $s = [\alpha]t$. Let $H \xrightarrow{\alpha}_{\mathfrak{M}_{\mathcal{D}}} H'$. We have to show $\mathfrak{M}_{\mathcal{D}}, H' \vDash t$. By Lemma 3.3.7, we have $t \in H'$, from which the claim follows by the inductive hypothesis. ∎

Theorem 3.3.9 (Demo Satisfaction) If $\mathcal{D}$ is a demo, then $\mathfrak{M}_{\mathcal{D}}$ satisfies $\mathcal{D}$.

Proof Immediate by Lemma 3.3.8. ∎

Remark 3.3.10 The converse direction of Theorem 3.3.9 does not hold. There are Hintikka systems S satisfied by $\mathfrak{M}_S$ that are not demos, an example being the system $S = \{H\}$ where $H = \{\langle a\rangle(p \vee q), p\}$. Clearly, $\mathfrak{M}_S, H \vDash p$, $H \xrightarrow{a}_{\mathfrak{M}_S} H$, and hence $\mathfrak{M}_S, H \vDash \langle a\rangle(p \vee q)$. Still, $p \vee q \notin H$, and hence S is not a demo. ◂

3.3.3 Demo Existence

Now we show that every satisfiable formula is contained in a demo. The claim is a variant of Fischer and Ladner's [67] filtration theorem for PDL. Let $\mathfrak{M}$ be a model and $w \in |\mathfrak{M}|$. We define $H_{\mathfrak{M},w} := \{\, s \in \mathcal{F} \mid \mathfrak{M}, w \vDash s \,\}$.

Proposition 3.3.11 $H_{\mathfrak{M},w}$ is a Hintikka set for every model $\mathfrak{M}$ and every $w \in |\mathfrak{M}|$.

Proof The claim is a straightforward consequence of the equivalences in Figure 2.2 and the definition of the satisfaction relation for conjunctions and disjunctions. ■

In the following, let us write H_w for $H_{\mathfrak{M},w}$ when the model $\mathfrak{M}$ is clear from the context. We show that $\mathcal{D}_{\mathfrak{M}} := \{ H_w \mid w \in |\mathfrak{M}| \}$ is a demo for every model $\mathfrak{M}$.

Lemma 3.3.12 Let $\mathfrak{M}$ be a model and $v \xrightarrow{\alpha}_{\mathfrak{M}} w$. Then $H_v \xrightarrow{\alpha}_{\mathcal{D}_{\mathfrak{M}}} H_w$.

Proof Let $\mathfrak{M}$, v, w and α be as required. We proceed by induction on α.

Let $\alpha = a$. We have to show that, for all s, $[a]s \in H_v$ implies $s \in H_w$. So, let $[a]s \in H_v$. Then $\mathfrak{M}, v \models [a]s$. Since, by assumption, $v \xrightarrow{a}_{\mathfrak{M}} w$, we have $\mathfrak{M}, w \models s$, and hence $s \in H_w$.

Let $\alpha = \beta + \gamma$. Then $v \xrightarrow{\beta}_{\mathfrak{M}} w$ or $v \xrightarrow{\gamma}_{\mathfrak{M}} w$. By the inductive hypothesis, $H_v \xrightarrow{\beta}_{\mathcal{D}_{\mathfrak{M}}} H_w$ or $H_v \xrightarrow{\gamma}_{\mathcal{D}_{\mathfrak{M}}} H_w$. Consequently, $H_v \xrightarrow{\beta+\gamma}_{\mathcal{D}_{\mathfrak{M}}} H_w$.

Let $\alpha = \beta\gamma$. Then there is some $u \in |\mathfrak{M}|$ such that $v \xrightarrow{\beta}_{\mathfrak{M}} u \xrightarrow{\gamma}_{\mathfrak{M}} w$. By the inductive hypothesis, $H_v \xrightarrow{\beta}_{\mathcal{D}_{\mathfrak{M}}} H_u \xrightarrow{\gamma}_{\mathcal{D}_{\mathfrak{M}}} H_w$, and hence $H_v \xrightarrow{\beta\gamma}_{\mathcal{D}_{\mathfrak{M}}} H_w$.

Let $\alpha = \beta^*$. Then there is some $n \geq 0$ such that $v \xrightarrow{\beta}{}^{n}_{\mathfrak{M}} w$. We proceed by induction on n. For $n = 0$, the claim is immediate since $H_v = H_w$. If $n > 0$, there is some $u \in |\mathfrak{M}|$ such that $v \xrightarrow{\beta}_{\mathfrak{M}} u \xrightarrow{\beta}{}^{n-1}_{\mathfrak{M}} w$. By the outer inductive hypothesis for β and the inner inductive hypothesis for $n - 1$ we obtain $H_v \xrightarrow{\beta}_{\mathcal{D}_{\mathfrak{M}}} H_u \xrightarrow{\beta}{}^{n-1}_{\mathcal{D}_{\mathfrak{M}}} H_w$. The claim follows. ■

Theorem 3.3.13 (Demo Existence) Let $\mathfrak{M}$ be a model. Then $\mathcal{D}_{\mathfrak{M}}$ is a demo.

Proof Let $\mathfrak{M}$ be a model. Since $\mathfrak{M}$ has at least one state, $\mathcal{D}_{\mathfrak{M}}$ is nonempty. Let $\langle\alpha\rangle s \in H_v \in \mathcal{D}_{\mathfrak{M}}$. It suffices to show that there is a state w such that $H_v \xrightarrow{\alpha}_{\mathcal{D}_{\mathfrak{M}}} H_w$ and $s \in H_w$. The claim follows with Lemma 3.3.12 since $\mathfrak{M}, v \models \langle\alpha\rangle s$. ■

Note that by definition, $\mathcal{D}_{\mathfrak{M}}$ contains every formula in $\mathcal{F}$ that is satisfied by $\mathfrak{M}$. We formulate two immediate consequences of Theorems 3.3.9 and 3.3.13.

Corollary 3.3.14 A formula is satisfiable if and only if it is contained in a demo.

Corollary 3.3.15 Every satisfiable formula is satisfied by a finite model obtained from a demo.

Corollary 3.3.15 states a small model property for our logic since the size of the model $\mathfrak{M}_{\mathcal{D}}$ obtained from a demo $\mathcal{D}$ is polynomial in the cardinality of $\mathcal{D}$, which, in turn, is at most exponential in that of $\mathcal{F}$. Corollary 3.3.14 gives a purely syntactic decision method. It is easy to see that demos are closed under union. Hence, there is a largest demo that contains all satisfiable Hintikka sets.

Corollary 3.3.16 A formula is satisfiable if and only if it is contained in the largest demo.

Remark 3.3.17 The model-theoretic counterpart to the closure of demos under union is the closure of models under disjoint unions (see [27], § 2.1 or [88], § 2.2). ◂

3.4 Pruning and Decision Procedure

Given a Hintikka system S, it is straightforward to compute the largest demo contained in S. One starts from S and stepwise deletes sets in S that cannot occur in a demo contained in S. We call this process **pruning**. In the absence of nominals or difference modalities, pruning always terminates with the largest demo contained in S. Applied to the set of all Hintikka sets, pruning then yields the largest demo.

Our decision procedure works by first constructing the set of all Hintikka sets and then pruning it to the largest demo.

We define a relation $\xrightarrow{\mathrm{P}}$ between Hintikka systems that represents a single pruning action: $S \xrightarrow{\mathrm{P}} S' \iff S' \subsetneq S$ and S' can be obtained from S by deleting some $H \in S$ that is not $\Diamond$-admissible in S.

Example 3.4.1 Consider the following Hintikka system S.

$$\begin{tikzcd}
\boxed{\langle a^*\rangle p,\ \langle a\rangle\langle a^*\rangle p,\ [a]\neg p} \arrow[r, shift left] & \boxed{\langle a^*\rangle p,\ \langle a\rangle\langle a^*\rangle p,\ \langle b\rangle p,\ [b]\neg p,\ \neg p} \arrow[l, shift left] \arrow[r, shift left] \arrow[loop above] & \boxed{p} \arrow[l, shift left] \arrow[loop above] \arrow[ll, bend left] \\
H_1 & H_2 & H_3
\end{tikzcd}$$

As in Example 3.3.5, the arrows between the sets depict the transition relation $\xrightarrow{a}_{\mathcal{D}}$.

Since we have $\langle b\rangle p \in H_2$, in order for H_2 to be $\Diamond$-admissible, S needs to contain some H such that $H_2 \xrightarrow{b}_S H$ and $p \in H$. However, since we also have $[b]\neg p \in H_2$, every H such that $H_2 \xrightarrow{b}_S H$ must contain $\neg p$. Clearly, S contains no set containing both p and $\neg p$, and hence H_2 is not $\Diamond$-admissible in S. Therefore, we have $S \xrightarrow{\mathrm{P}} S' = \{H_1, H_3\}$.

Without H_2, we no longer have $H_1 \xrightarrow{a^*}_S H_3$. Therefore, H_1 is not $\Diamond$-admissible in S', and we have $S' \xrightarrow{\mathrm{P}} \mathcal{D} = \{H_3\}$.

Since H_3 contains no diamond formulas, it is trivially $\Diamond$-admissible in every Hintikka system. Hence $\mathcal{D}$ is a demo and pruning terminates on $\mathcal{D}$. ◂

Note that Proposition 3.3.2 implies that if a Hintikka set H is $\Diamond$-admissible in S, it is also $\Diamond$-admissible in every superset of S. The following two lemmas capture important correctness properties of the pruning relation.

Lemma 3.4.2 Let $\mathcal{D} \subseteq S$ be a demo and $S \xrightarrow{\mathrm{P}} S'$. Then $\mathcal{D} \subseteq S'$.

Proof Straightforward since pruning removes only sets that are not $\Diamond$-admissible in S. Since $\mathcal{D}$ is $\Diamond$-admissible and $\mathcal{D} \subseteq S$, all of the sets in $\mathcal{D}$ are $\Diamond$-admissible in S ∎

Let a relation $\to$ be given. We define $\mapsto\ := \{\,(x,y) \in \to^* \mid \nexists z\colon\ y \to z\,\}$.

Lemma 3.4.3 Let $S \overset{\mathrm{P}}{\mapsto} S'$. Then S' is either empty or a demo.

Proof Immediate since when no more pruning can be done, all sets in S' are $\Diamond$-admissible. ∎

Since we have $S' \subsetneq S$ whenever $S \xrightarrow{\text{P}} S'$ (and Hintikka systems are finite), the pruning relation is terminating.

The following proposition, while not necessary for the correctness of the decision procedure, constitutes an important insight into the nature of the pruning relation. It yields that pruning is (strongly) confluent (see [19]).

Proposition 3.4.4 If $S_1 \xleftarrow{\text{P}} S \xrightarrow{\text{P}} S_2$ such that $S_1 \neq S_2$, then there is some S' such that $S_1 \xrightarrow{\text{P}} S' \xleftarrow{\text{P}} S_2$.

Proof Let $S_1 \xleftarrow{\text{P}} S \xrightarrow{\text{P}} S_2$ such that $S_1 \neq S_2$. Then there are two distinct $H_1, H_2 \in S$ such that $S_1 = S \setminus \{H_1\}$ and $S_2 = S \setminus \{H_2\}$. Let $S' = S \setminus \{H_1, H_2\}$. Since H_2 is not $\Diamond$-admissible in S, it is not $\Diamond$-admissible in S_1, and hence $S_1 \xrightarrow{\text{P}} S'$. Analogously, since H_1 is not $\Diamond$-admissible in S_2, we obtain $S_2 \xrightarrow{\text{P}} S'$. ∎

Together with termination, the confluence of $\xrightarrow{\text{P}}$ means that the relation $\overset{\text{P}}{\mapsto}$ is in fact a function. For every Hintikka system S there exists exactly one system S' such that $S \overset{\text{P}}{\mapsto} S'$. Moreover, S' is obtained from S in at most $|S|$ pruning steps.

Theorem 3.4.5 Let $S \overset{\text{P}}{\mapsto} S'$. Then S' is either empty or the largest demo contained in S. Moreover, S contains a demo if and only if S' is nonempty.

Proof By Lemma 3.4.2, S' contains all demos contained in S. Hence, S' is nonempty whenever S contains a demo.

Conversely, suppose S' is nonempty. By Lemma 3.4.3, S' is a demo contained in S. Hence S contains a largest demo $\mathcal{D}$. Since $S' \subseteq \mathcal{D}$ and, by Lemma 3.4.2, $\mathcal{D} \subseteq S'$, we have $S' = \mathcal{D}$. ∎

Corollary 3.4.6 Let $\mathcal{H}$ be the set of all Hintikka subsets of $\mathcal{F}$ and let $\mathcal{H} \overset{\text{P}}{\mapsto} S$. Then S is the largest demo over $\mathcal{F}$.

Proof The claim is immediate from Theorem 3.4.5, provided for every formula universe $\mathcal{F}$ there is a largest demo. As observed in § 3.2, $\mathcal{F}$ contains at least one predicate. Let p be such a predicate. Then $\{\{p\}\}$ is a demo. Hence the largest demo exists. ∎

The decision procedure based on pruning can be summarized as shown in Figure 3.2. Given a formula $s \in \mathcal{F}$, it constructs a set $\mathcal{H}$ of all Hintikka sets and then prunes $\mathcal{H}$ to obtain the largest demo $\mathcal{D}$. It returns SAT if and only if s is contained in some Hintikka set of $\mathcal{D}$. The correctness of the procedure is immediate by Corollaries 3.3.16 and 3.4.6.

Let us now convince ourselves that the procedure runs in exponential time with respect to $|\mathcal{F}|$. The bound easily follows from the following observations:

Input: a formula $s \in \mathcal{F}$
– Compute $\mathcal{H} := \{ H \mid H \text{ is a Hintikka subset of } \mathcal{F} \}$.
– Compute $\mathcal{D}$ with $\mathcal{H} \overset{\mathrm{P}}{\mapsto} \mathcal{D}$.
Output: SAT if $s \in H$ for some $H \in \mathcal{D}$ and UNSAT otherwise

Figure 3.2: Decision procedure for PDL

- To compute $\mathcal{H}$, one can create all the exponentially many subsets of $\mathcal{F}$ and remove those that are not Hintikka sets. Checking if a given set is Hintikka is possible in polynomial time with respect to the cardinality of the set, which in turn is linear in $|\mathcal{F}|$.
- As noted above, computing $\mathcal{D}$ from $\mathcal{H}$ takes at most $|\mathcal{H}|$ pruning steps. Each such step traverses through all Hintikka sets H in the remaining system S and all formulas $\langle\alpha\rangle s \in H$, and checks whether in each case S contains some H' such that $H \xrightarrow{\alpha}_S H'$ and $s \in H'$. Since for every Hintikka system S and every program α, computing $\xrightarrow{\alpha}_S$ can be done in polynomial time with respect to $|S|$, each pruning step takes exponential time with respect to $|\mathcal{F}|$.
- Checking if s is contained in $\mathcal{D}$ can be done in time polynomial in $|\mathcal{D}|$ and $|\mathcal{F}|$.

3.5 Nominals

3.5.1 Demos and Pruning

Let us now see what happens if we extend our language with nominals. The definition of demos needs to be extended to account for the special semantics of nominals. A Hintikka system S is **nominally admissible** if every nominal $x \in \mathcal{F}$ is contained in exactly one $H \in S$. We extend the definition of demos (Definition 3.3.3) by adding the additional requirement that demos be nominally admissible.

Likewise, we restrict the definition of the model $\mathfrak{M}_S$ from § 3.3.2 to nominally admissible Hintikka systems S. It is easy to see that $\mathfrak{M}_S$ satisfies the semantics of nominals (i.e., maps every nominal in $\mathcal{F}$ to a singleton set) if and only if S is nominally admissible.

In the presence of nominals, demos are no longer closed under union. To see this, consider the two demos $\{\{x, p\}\}$ and $\{\{x, \neg p\}\}$. Clearly, their union, $\{\{x, p\}, \{x, \neg p\}\}$, is not a demo since it is not nominally admissible. Moreover, the union contains no largest demo. The two demos it contains are mutually disjoint and hence both maximal. Hence, Corollary 3.3.16 is no longer true.

All other claims in §§ 3.3.1–3.3.3 are unaffected. The only proof that needs to be adapted is that of Theorem 3.3.13, where we need to check that $\mathcal{D}_{\mathfrak{M}}$ is nominally admissible. This is the case since every nominal x denotes a unique state $w_x \in |\mathfrak{M}|$. Therefore, $H_{\mathfrak{M},w_x}$ is the unique Hintikka set in $\mathcal{D}_{\mathfrak{M}}$ that contains x.

So far, adding nominals has not caused any notable complications. The major complication arising from the lack of closure under unions becomes apparent when we try to define pruning. Consider, for instance, the following Hintikka system S:

$$\{\{x,\ \langle a^*\rangle p,\ p\},\ \{x,\ \langle a^*\rangle p,\ \langle a\rangle\langle a^*\rangle p\}\}$$

The definition of pruning from § 3.4 does not apply to S since both of its sets are $\Diamond$-admissible. Still, S is not a demo since it is not nominally admissible.

Our first idea may be to try extending pruning such that it will establish nominal admissibility. But how should pruning behave on S? To obtain nominal admissibility, pruning would have to delete one of its sets. Let $H_1 = \{x,\ \langle a^*\rangle p,\ p\}$ and $H_2 = \{x,\ \langle a^*\rangle p,\ \langle a\rangle\langle a^*\rangle p\}$. If we delete the set H_2, we obtain the system $\{H_1\}$, which is a demo. If, however, we delete H_1, the remaining set H_2 will not be $\Diamond$-admissible in $\{H_2\}$ since $\langle a^*\rangle p \in H_2$ but $\{H_2\}$ contains no set H such that $H_2 \xrightarrow{a^*}_{\{H_2\}} H$ and $p \in H$. Hence, $\{H_2\}$ contains no demo.

More generally, since in the presence of nominals a Hintikka system that contains a demo does not necessarily contain a largest demo, we will end up with different demos (or the empty set) depending on which set we delete to achieve nominal admissibility. In particular, there is no way of extending pruning to nominals so that it will satisfy Lemma 3.4.2 (consider $\{\{x,\ p\},\ \{x,\ \neg p\}\}$), Proposition 3.4.4, or Theorem 3.4.5.

To avoid these problems, we work with the original definition of pruning from § 3.4. In the following, we show that pruning remains complete when applied to nominally admissible systems. A key prerequisite for this is that nominally admissible Hintikka systems contain largest demos.

Lemma 3.5.1 Every nominally admissible Hintikka system that contains a demo contains a largest demo.

Proof Let S be a nominally admissible Hintikka system and $\mathcal{D}_1, \mathcal{D}_2 \subseteq S$ be two demos. It suffices to show that $\mathcal{D}_1 \cup \mathcal{D}_2$ is a demo. Since $\mathcal{D}_1$ and $\mathcal{D}_2$ are nominally admissible, both of them contain all sets in S that contain nominals, and hence $\mathcal{D}_1 \cup \mathcal{D}_2$ is nominally admissible. The claim follows because $\Diamond$-admissibility is stable under union. ∎

Hence, to reestablish Lemmas 3.4.2, 3.4.3 and Theorem 3.4.5, we restrict them to nominally admissible Hintikka systems.

Lemma 3.5.2 Let S be nominally admissible, $\mathcal{D} \subseteq S$ be a demo and $S \xrightarrow{\mathrm{P}} S'$. Then S' is nominally admissible and $\mathcal{D} \subseteq S'$.

Proof The inclusion $\mathcal{D} \subseteq S'$ follows the same as for Lemma 3.4.2. The system S' is nominally admissible since so are both S and $\mathcal{D}$, and since $\mathcal{D} \subseteq S' \subseteq S$. ∎

Even when applied to a nominally admissible Hintikka system, pruning is no longer guaranteed to terminate with either an empty set or a demo. For instance, suppose x and y are the only nominals in $\mathcal{F}$. Then $S = \{\{x\},\ \{y,\ \langle a\rangle p\}\}$ is nominally admissible but, at the same time, $S \overset{\mathrm{P}}{\mapsto} \{\{x\}\}$, where $\{\{x\}\}$ is not nominally admissible (this does not contradict Lemma 3.5.2 since S contains no demo). Hence, we have to weaken Lemma 3.4.3 and Theorem 3.4.5 as follows.

Input: a formula $s \in \mathcal{F}$
- Compute $\mathcal{H} := \{ H \mid H$ is a Hintikka subset of $\mathcal{F} \}$.
- Guess a nominally admissible subset $\mathcal{H}'$ of $\mathcal{H}$.
- Compute $\mathcal{D}$ with $\mathcal{H}' \overset{\mathrm{P}}{\mapsto} \mathcal{D}$.

Output: SAT if $s \in H$ for some $H \in \mathcal{D}$ and UNSAT otherwise

Figure 3.3: Decision procedure for hybrid PDL

Lemma 3.5.3 Let $S \overset{\mathrm{P}}{\mapsto} S'$ and S' be nominally admissible. Then S' is a demo.

Proof Let S, S' be as required. Since S' is nominally admissible and $\mathcal{F}$ contains at least one nominal, S' is not empty. Since no more pruning can be done, all sets in S' are $\Diamond$-admissible. ∎

Theorem 3.5.4 If S is nominally admissible, contains a demo, and $S \overset{\mathrm{P}}{\mapsto} S'$, then S' is the largest demo contained in S.

Proof Let S, S' be as required. The system S contains a largest demo by Lemma 3.5.1. Let us call this demo $\mathcal{D}$. Since S' is nominally admissible (Lemma 3.5.2), it is a demo contained in S (Lemma 3.5.3) and hence in $\mathcal{D}$. By Lemma 3.5.2, S' contains all demos contained in S, including $\mathcal{D}$. Taken together, the two inclusions yield $S' = \mathcal{D}$. ∎

Remark 3.5.5 In the absence of satisfaction operators, global and difference modalities, one could weaken the notion of nominal admissibility of a Hintikka system S to require that every nominal in $\mathcal{F}$ is contained in *at most* one $H \in S$. The relaxed definition suffices to show Lemmas 3.5.1, 3.5.2, 3.5.3, and Theorem 3.5.4. The definition of $\mathfrak{M}_S$ would need to be adapted to provide interpretation for the nominals in $\mathcal{F}$ that do not occur in S. We prefer the strong notion of nominal admissibility because it allows to keep the definition of $\mathfrak{M}_S$ unchanged compared to the nominal-free case and, more importantly, because it scales to stronger languages. To see why weak nominal admissibility fails when we add difference modalities, consider the Hintikka system $\{\{x, \neg y, \bar{\mathsf{D}}x\}\}$. While the system satisfies weak nominal admissibility, it is clearly unsatisfiable since every model of the system would need to provide a denotation for the nominal y. This is not possible since the system restricts the domain to be a singleton set (with its only element denoted by x) while at the same time requiring y to be different from x. ◂

3.5.2 Decision Procedure

The decision procedure is adapted to nominals by inserting a nondeterministic guessing stage that is performed between the construction of the system $\mathcal{H}$ of all Hintikka sets and pruning. One guesses a nominally admissible subsystem of $\mathcal{H}$ that contains a demo containing the input formula s, provided such a demo exists. The adapted procedure is shown in Figure 3.3.

The correctness of the procedure follows from Corollary 3.3.14 and Theorem 3.5.4. A formula s is satisfiable if and only if it is contained in a demo that is contained in (and can be obtained by pruning starting from) some nominally admissible subsystem of $\mathcal{H}$.

We now show that the procedure has a determinization that runs in exponential time. Given that the construction and the pruning stage each take at most exponential time with respect to $|\mathcal{F}|$, it suffices to show that the guessing stage can be determinized by trying at most exponentially many (with respect to $|\mathcal{F}|$) subsystems of $\mathcal{H}$, each of which can be constructed in exponential time with respect to $|\mathcal{F}|$.

To see this, note that the guessing stage can be arranged as follows. Let $x_1, \dots, x_n$ be the linearly many nominals in $\mathcal{F}$. For each $i \in [1, n]$, one guesses a Hintikka set in $\mathcal{H}$ that contains x_i and removes all other sets that contain x_i. Should the system that remains after the n-th iteration contain no set for some x_i, it is rejected. Otherwise, the system is a nominally admissible subset of $\mathcal{H}$ to which we can apply pruning.

To convince ourselves that the approach is correct, it suffices to check that for every demo $\mathcal{D} \subseteq \mathcal{H}$, the algorithm can obtain a nominally admissible subset of $\mathcal{H}$ that contains $\mathcal{D}$. This is easy to see. Given a demo $\mathcal{D} \subseteq \mathcal{H}$, it suffices to choose in each step i the unique set from $\mathcal{D}$ that contains x_i. All other sets containing x_i cannot occur in $\mathcal{D}$ since $\mathcal{D}$ is nominally admissible, and hence can be safely removed.

Altogether, the approach yields at most $|\mathcal{H}|^n$ candidate subsystems, some of which are rejected as not nominally admissible. Since n is linear in $|\mathcal{F}|$ and $|\mathcal{H}|$ is exponential in $|\mathcal{F}|$, the overall number of candidate subsystems that have to be searched through by a determinization of the n-step guessing stage is exponential in $|\mathcal{F}|$. Moreover, once we fix a Hintikka set for every nominal, each such system can be built from $\mathcal{H}$ in time polynomial in $n \cdot |\mathcal{H}|$, i.e., exponential in $|\mathcal{F}|$.

To summarize, Pratt-style decision procedures can be extended to deal with nominals preserving the ExpTime worst-case complexity bound.

3.6 Difference Modalities

3.6.1 Demos

To deal with difference modalities, the definition of demos needs to be extended once again. A Hintikka set $H \in S$ is **D-admissible** (in S) if for all $\mathsf{D}s \in H$ there is some $H' \in S$ such that $H' \neq H$ and $s \in H'$.

In addition to nominal admissibility and $\Diamond$-admissibility we require a demo $\mathcal{D}$ to satisfy the following conditions.

- **D-admissibility:** Every set in $\mathcal{D}$ is D-admissible.
- **$\bar{\mathsf{D}}$-admissibility:** If $\bar{\mathsf{D}}s \in H \in \mathcal{D}$, then, for all $H' \in \mathcal{D}$ such that $H' \neq H$, we have $s \in H'$.

As with nominals, demos for PDL with difference modalities are not closed under union. This follows from $\bar{\mathsf{D}}$-admissibility alone, independently of whether or not we require nominal admissibility. For instance, consider the demos, $\{\{\bar{\mathsf{D}}p\}\}$ and $\{\{\neg p\}\}$. Clearly, their union $S = \{\{\bar{\mathsf{D}}p\}, \{\neg p\}\}$ is not a demo (and contains no largest demo) since it is not $\bar{\mathsf{D}}$-admissible. This fact is not surprising given that $\bar{\mathsf{D}}$ can express nominals (see § 2.4.2).

One may be wondering if we need nominal admissibility at all if we want to add difference modalities without nominals. The answer to this is that in order to show demo existence (Theorem 3.3.13), we will employ auxiliary nominals. These nominals will be used regardless of whether we allow nominals in the input formula. And nominal admissibility is there to ensure that these auxiliary nominals have the right semantics.

Why do we need auxiliary nominals? Suppose $\mathcal{F} = \{\mathsf{D}p, p\}$ and consider the model $\mathfrak{M}$ with $|\mathfrak{M}| = \{v, w\}$ and $\mathfrak{M}p = \{v, w\}$. Clearly, we have $\mathfrak{M}, v \vDash \mathsf{D}p$ and hence $\mathsf{D}p \in H_v$. Since $H_v = H_w$, we have $\mathcal{D}_{\mathfrak{M}} = \{H_v\}$. But then $\mathcal{D}_{\mathfrak{M}}$ is not D-admissible and hence not a demo. Generally speaking, the problem is that a model $\mathfrak{M}$ from which we obtain the demo $\mathcal{D}_{\mathfrak{M}}$ may contain several states that satisfy the same formulas from $\mathcal{F}$. In $\mathcal{D}_{\mathfrak{M}}$, all such states will be mapped to the same Hintikka set. Mapping all states in $\mathfrak{M}$ that satisfy a formula s (assuming there is more than one such state) to the same Hintikka set in $\mathcal{D}_{\mathfrak{M}}$ may invalidate $\mathsf{D}s$ in the model obtained from $\mathcal{D}_{\mathfrak{M}}$.

To deal with the problem, we assume an injective function that assigns to every literal $\mathsf{D}s$ a nominal $x_{\mathsf{D}s}$ that is isolated, i.e., occurs in no other formula in $\mathcal{F}$. Intuitively, a nominal $x_{\mathsf{D}s}$ is supposed to denote a state that satisfies s, provided such a state exists. If it does, all states that are different from the one denoted by $x_{\mathsf{D}s}$ satisfy $\mathsf{D}s$. Since $x_{\mathsf{D}s}$ is satisfied by only one state, this will suffice to ensure that whenever a state w satisfies $\mathsf{D}s$ there is a state v satisfying s such that $H_v \neq H_w$.

It remains to ensure that the auxiliary nominals $x_{\mathsf{D}s}$ denote the correct states in $\mathfrak{M}$. We call a model $\mathfrak{M}$ **nice** if for all $\mathsf{D}s \in \mathcal{F}$, $\mathfrak{M}$ satisfies $\{s, x_{\mathsf{D}s}\}$ whenever $\mathfrak{M}$ satisfies s. Theorem 3.3.13 then adapts as follows.

Theorem 3.6.1 (Demo Existence with D) Let $\mathfrak{M}$ be a nice model. Then $\mathcal{D}_{\mathfrak{M}}$ is a demo.

Proof Nominal admissibility and $\Diamond$-admissibility are shown as before. The remaining cases proceed as follows.

D*-admissibility:* Let $\mathsf{D}s \in H_w \in \mathcal{D}_{\mathfrak{M}}$. Then there is some $v \neq w$ such that $\mathfrak{M}, v \vDash s$, and hence $s \in H_v$. It remains to show that $H_v \neq H_w$. Since $\mathfrak{M}$ is nice, we either have $\mathfrak{M}, w \vDash x_{\mathsf{D}s}$ or we can choose v such that $\mathfrak{M}, v \vDash x_{\mathsf{D}s}$. In the first case, we have $x_{\mathsf{D}s} \in H_w \setminus H_v$, and in the second case, $x_{\mathsf{D}s} \in H_v \setminus H_w$. In either case, $H_v \neq H_w$.

$\bar{\mathsf{D}}$*-admissibility:* Let $\bar{\mathsf{D}}s \in H_w \in \mathcal{D}_{\mathfrak{M}}$ and let $H_v \in \mathcal{D}_{\mathfrak{M}}$ be distinct from H_w. Then $v \neq w$. Therefore, since $\mathfrak{M}, w \vDash \bar{\mathsf{D}}s$, we have $\mathfrak{M}, v \vDash s$, and hence $s \in H_v$. ∎

The proof of Lemma 3.3.8 is adapted by adding the following two cases (recall that we show $\mathfrak{M}_{\mathcal{D}}, H \vDash s$ for every demo $\mathcal{D}$ and every $s \in H \in \mathcal{D}$ by induction on s).

Let $s = \mathsf{D}t$. Then, since $\mathcal{D}$ is D-admissible, there is some $H' \in \mathcal{D}$ distinct from H such that $t \in H'$. By the inductive hypothesis $\mathfrak{M}_{\mathcal{D}}, H' \vDash t$. The claim follows.

Let $s = \bar{\mathsf{D}}t$. Let $H' \in \mathfrak{M}_{\mathcal{D}}$ be distinct from H. To show: $\mathfrak{M}_{\mathcal{D}}, H' \vDash t$. This follows by the inductive hypothesis from $t \in H'$, which holds by the $\bar{\mathsf{D}}$-admissibility of $\mathcal{D}$.

Corollaries 3.3.14 and 3.3.15 follow by observing that a formula s is satisfiable if and only if it is satisfied by a nice model. Since the auxiliary nominals are isolated by assumption, s is either an auxiliary nominal or contains no auxiliary nominals as subformulas. In the former case, it is easy to construct a satisfying nice model. In the latter case, we can always make the satisfying model nice by making the auxiliary nominals denote the right states. Since s contains no auxiliary nominals, this cannot influence the satisfaction of s.

Summing up, the main challenge so far was reestablishing the demo existence theorem (Theorem 3.6.1) in the presence of the existential difference modality D. For this purpose we added auxiliary nominals and a concomitant restriction on demos. Once the definitions are in place, adapting the proofs is straightforward.

3.6.2 Pruning

The existential modality D is treated analogously to diamond operators by suitably extending the pruning relation. Now, pruning is defined as follows: $S \xrightarrow{\text{P}} S' \iff S' \subsetneq S$ and S' can be obtained from S by deleting some $H \in S$ that is not $\Diamond$-admissible or not D-admissible in S. Note that this extension does not affect the confluence of $\xrightarrow{\text{P}}$.

Unlike with demo existence, where the complications are caused by D, the main challenge as it comes to pruning and the decision procedure is dealing with the universal difference modality $\bar{\mathsf{D}}$.

It is easily seen that if a Hintikka system S is $\bar{\mathsf{D}}$-admissible, then so are all of its subsets.

Lemma 3.6.2 Let $S \subseteq S'$ be Hintikka systems. If S' is $\bar{\mathsf{D}}$-admissible, then so is S.

Proof Immediate by the definition of $\bar{\mathsf{D}}$-admissibility and the fact that $S \subseteq S'$. ∎

Lemma 3.5.1 adapts if we add $\bar{\mathsf{D}}$-admissibility as an additional assumption.

Lemma 3.6.3 Let S be a nominally admissible and $\bar{\mathsf{D}}$-admissible Hintikka system. If S contains a demo, then S contains a largest demo.

Proof Let S be a nominally admissible, $\bar{\mathsf{D}}$-admissible Hintikka system and $\mathcal{D}_1, \mathcal{D}_2 \subseteq S$ be two demos. It suffices to show that $\mathcal{D}_1 \cup \mathcal{D}_2$ is a demo. Nominal admissibility and $\Diamond$-admissibility are established the same as for Lemma 3.5.1. Likewise, D-admissibility is stable under union. The $\bar{\mathsf{D}}$-admissibility of $\mathcal{D}_1 \cup \mathcal{D}_2$ follows with Lemma 3.6.2. ∎

Lemmas 3.5.2, 3.5.3, and Theorem 3.5.4 are adapted as follows.

Lemma 3.6.4 Let S be nominally admissible and $\bar{\mathsf{D}}$-admissible, $\mathcal{D} \subseteq S$ be a demo and $S \xrightarrow{\text{P}} S'$. Then S' is nominally admissible, $\bar{\mathsf{D}}$-admissible, and $\mathcal{D} \subseteq S'$.

Proof Proceeds analogously to the proof of Lemma 3.5.2. The preservation of $\bar{\mathsf{D}}$-admissibility follows with Lemma 3.6.2. ∎

Lemma 3.6.5 Let $S \overset{\text{P}}{\mapsto} S'$ and S' be nominally admissible and $\bar{\mathsf{D}}$-admissible. Then S' is a demo.

Proof Let S, S' be as required. Since S' is nominally admissible and $\mathcal{F}$ contains at least one nominal, S' is not empty. Since no more pruning can be done, S' is $\Diamond$-admissible and D-admissible. ∎

Theorem 3.6.6 Let S be nominally admissible and $\bar{\mathsf{D}}$-admissible. If S contains a demo and $S \overset{\mathrm{P}}{\mapsto} S'$, then S' is the largest demo contained in S.

Proof The proof proceeds analogously to that of Theorem 3.5.4 with Lemmas 3.6.3, 3.6.4 and 3.6.5 substituting, respectively, Lemmas 3.5.1, 3.5.2 and 3.5.3. ∎

3.6.3 Decision Procedure

Theorem 3.6.6 implies that we can reuse the procedure in Figure 3.3 if we reformulate the guessing stage as follows:

– Guess a subset $\mathcal{H}'$ of $\mathcal{H}$ that is nominally admissible and $\bar{\mathsf{D}}$-admissible.

Since every demo contained in $\mathcal{H}$ is contained in a nominally admissible and $\bar{\mathsf{D}}$-admissible subsystem of $\mathcal{H}$, the procedure is correct.

To see that this guessing stage can be determinized preserving the ExpTime complexity bound on the overall procedure, we will give an algorithmic refinement of the stage and observe the following two properties.

- For every demo $\mathcal{D} \subseteq \mathcal{H}$, the refined guessing stage can obtain a nominally admissible and $\bar{\mathsf{D}}$-admissible subsystem of $\mathcal{H}$ that contains $\mathcal{D}$.
- The number of subsystems of $\mathcal{H}$ from which the refined stage has to choose is at most exponential in $|\mathcal{F}|$.

The refined stage proceeds in two steps. First, it guesses a $\bar{\mathsf{D}}$-admissible subsystem $\mathcal{H}''$ of $\mathcal{H}$. It then applies the guessing algorithm from § 3.5.2 to obtain a nominally consistent subsystem $\mathcal{H}'$ of $\mathcal{H}''$.

We now describe how we obtain $\mathcal{H}''$. Observe that a Hintikka system S is $\bar{\mathsf{D}}$-admissible if and only if for every formula $\bar{\mathsf{D}}s \in \mathcal{F}$ one of the following two cases holds.

1. The formula $\bar{\mathsf{D}}s$ is not contained in S.
2. There is some $H \in S$ such that $\bar{\mathsf{D}}s \in H$ and we have $s \in H'$ for all $H' \in S \setminus \{H\}$. Moreover, if $s \notin H$, then for all $H' \in S \setminus \{H\}$ we have $\bar{\mathsf{D}}s \notin H'$.

Therefore, we can obtain a $\bar{\mathsf{D}}$-admissible subset of $\mathcal{H}$ by nondeterministically choosing, for every formula $\bar{\mathsf{D}}s$, one of the following two actions.

1. Prune $\mathcal{H}$ by removing every set that contains $\bar{\mathsf{D}}s$.
2. Choose some $H \in \mathcal{H}$ such that $\bar{\mathsf{D}}s \in H$ and prune $\mathcal{H}$ by removing every set other than H that does not contain s. If $s \notin H$, additionally remove every set other than H that contains $\bar{\mathsf{D}}s$.

By the above observation, it is immediate that performing one of the two pruning actions for every formula $\bar{\mathsf{D}}s$ indeed results in a $\bar{\mathsf{D}}$-admissible set. Moreover, since demos are $\bar{\mathsf{D}}$-admissible, for every demo $\mathcal{D} \subseteq \mathcal{H}$ there is a sequence of pruning steps starting from $\mathcal{H}$ that yields some $\mathcal{H}''$ that contains $\mathcal{D}$. To obtain $\mathcal{H}''$, for every formula $\bar{\mathsf{D}}s$, one performs action (1) if $\mathcal{D}$ does not contain $\bar{\mathsf{D}}s$. Otherwise, one performs action (2) for some set in $\mathcal{D}$ that contains $\bar{\mathsf{D}}s$. In both cases, it is easily verified that one removes no set that occurs in $\mathcal{D}$, and hence the resulting system is a superset of $\mathcal{D}$. We conclude that the algorithm is correct.

Example 3.6.7 Suppose $\mathcal{F} = \{\bar{\mathsf{D}}\neg p, \bar{\mathsf{D}}p, \neg p, p\}$. Then the system $\mathcal{H}$ of all Hintikka sets contains 11 elements:

$$\boxed{\bar{\mathsf{D}}p} \quad \boxed{p} \quad \boxed{\neg p} \quad \boxed{\bar{\mathsf{D}}\neg p}$$

$$\boxed{\bar{\mathsf{D}}p,\ p} \quad \boxed{\bar{\mathsf{D}}p,\ \neg p} \quad \boxed{\bar{\mathsf{D}}\neg p,\ p} \quad \boxed{\bar{\mathsf{D}}\neg p,\ \neg p}$$

$$\boxed{\bar{\mathsf{D}}p,\ \bar{\mathsf{D}}\neg p,\ p} \quad \boxed{\bar{\mathsf{D}}p,\ \bar{\mathsf{D}}\neg p} \quad \boxed{\bar{\mathsf{D}}p,\ \bar{\mathsf{D}}\neg p,\ \neg p}$$

Clearly, $\mathcal{H}$ is not $\bar{\mathsf{D}}$-admissible. Consider the demo $\mathcal{D} = \{\{\bar{\mathsf{D}}p, \neg p\}, \{\bar{\mathsf{D}}\neg p, p\}\}$. Let us extract a $\bar{\mathsf{D}}$-admissible subsystem of $\mathcal{H}$ that contains $\mathcal{D}$ using the above algorithm (in fact, in this particular case the algorithm will yield $\mathcal{D}$). Since $\mathcal{H}$ contains two distinct formulas of the form $\bar{\mathsf{D}}s$, we proceed in two steps. In the first step we consider the formula $\bar{\mathsf{D}}p$. We choose the set $\{\bar{\mathsf{D}}p, \neg p\}$ (since it occurs in $\mathcal{D}$) and remove every other set that does not contain p or contains $\bar{\mathsf{D}}p$ (action (2)). This yields the system $\mathcal{H}_1 = \{\{p\}, \{\bar{\mathsf{D}}p, \neg p\}, \{\bar{\mathsf{D}}\neg p, p\}\}$. As expected, $\mathcal{D} \subseteq \mathcal{H}_1$. The system $\mathcal{H}_1$ is still not $\bar{\mathsf{D}}$-admissible. In the second step we consider $\bar{\mathsf{D}}\neg p$. We again perform action (2), now for the set $\{\bar{\mathsf{D}}\neg p, p\}$ (since $\{\bar{\mathsf{D}}\neg p, p\}$ occurs in $\mathcal{D}$), and obtain $\mathcal{D}$. ◂

Since we have to decide between no more than $|\mathcal{H}| + 1$ alternatives for every formula $\bar{\mathsf{D}}s \in \mathcal{F}$, and $|\mathcal{H}| + 1 \leq 2^{|\mathcal{F}|}$, the number of candidates that need to be tried by a determinization of the first step of the guessing stage is bounded by $2^{|\mathcal{F}|^2}$. Each of the candidates can be computed in time polynomial in $|\mathcal{H}|$ and $|\mathcal{F}|$.

Once we have a $\bar{\mathsf{D}}$-admissible Hintikka system S, a nominally admissible subsystem of S (if one exists) can be obtained as in § 3.5.2 by guessing, for each nominal in $x \in \mathcal{F}$, a Hintikka set in S that contains x and removing all other sets containing x. The resulting system is still $\bar{\mathsf{D}}$-admissible by Lemma 3.6.2.

If enumerating $\bar{\mathsf{D}}$-admissible subsystems produces at most m candidates, each of which contains at most n maximal nominally admissible subsystems, then the overall number of $\bar{\mathsf{D}}$-admissible and nominally admissible candidates that have to be tried by the deterministic procedure is bounded by $m \cdot n$. Since both numbers are exponentially bounded in $|\mathcal{F}|$, so is their product. Hence, the overall procedure runs in deterministic single exponential time.

3.7 Tests

Now, let us see what needs to be done in order to add tests. Adding tests primarily affects the proofs of demo satisfaction and demo existence in § 3.3.

First, we need to extend the definition of $\xrightarrow{\alpha}_S$ (Definition 3.3.1) by adding a case for tests: $\xrightarrow{s}_S = \{\,(H,H) \mid s \in H \in S\,\}$.

To obtain demo satisfaction (Theorem 3.3.9), it suffices to reprove Lemmas 3.3.6 and 3.3.7. With tests, however, the inclusion $\xrightarrow{\alpha}_{\mathfrak{M}_S} \subseteq \xrightarrow{\alpha}_S$ for Lemma 3.3.6 does no longer hold. Consider $S = \{\{p\}\}$ and suppose $p \vee q \in \mathcal{F}$. Then $\{p\} \xrightarrow{p \vee q}_{\mathfrak{M}_S} \{p\}$ but not $\{p\} \xrightarrow{p \vee q}_S \{p\}$ since $p \vee q \notin \{p\}$. Fortunately, the only inclusion needed for demo satisfaction is $\xrightarrow{\alpha}_S \subseteq \xrightarrow{\alpha}_{\mathfrak{M}_S}$, which still holds.

Moreover, both Lemma 3.3.6 (restricted to the inclusion $\xrightarrow{\alpha}_S \subseteq \xrightarrow{\alpha}_{\mathfrak{M}_S}$) and Lemma 3.3.7 require additional assumptions. We restate the two lemmas as follows.

Lemma 3.7.1 Let S be a Hintikka system and α a program such that, for all tests t in α and all $H \in S$, $t \in H$ implies $\mathfrak{M}_S, H \vDash t$. Then $\xrightarrow{\alpha}_S \subseteq \xrightarrow{\alpha}_{\mathfrak{M}_S}$.

Proof Let S and α be as required. We proceed by induction on α. All cases but $\alpha = t$ proceed the same as in the proof of Lemma 3.3.6. So, let $\alpha = t$. Suppose $H \xrightarrow{t}_S H'$. Then $H = H'$ and $t \in H$. By the additional assumption, we have $\mathfrak{M}_S, H \vDash t$, and hence $H \xrightarrow{t}_{\mathfrak{M}_S} H$. ∎

Lemma 3.7.2 Let S be a Hintikka system, $\{H, H'\} \subseteq S$, and $[\alpha]s \in H$ such that
- $H \xrightarrow{\alpha}_{\mathfrak{M}_S} H'$ and
- for all tests t in α and all $H \in S$, $\sim t \in H$ implies $\mathfrak{M}_S, H \vDash \sim t$.

Then $s \in H'$.

Proof Let S, H, H' and $[\alpha]s$ be as required. We proceed by induction on α. All cases but $\alpha = t$ proceed the same as in the proof of Lemma 3.3.7. So, let $\alpha = t$. Then $H = H'$ and $\mathfrak{M}_S, H \vDash t$. Since H is a Hintikka set and $[t]s \in H$, we have $\sim t \in H$ or $s \in H$. If $\sim t \in H$, then, by the additional assumption, $\mathfrak{M}_S, H \vDash \sim t$, which contradicts $\mathfrak{M}_S, H \vDash t$. Hence, we have $s \in H$, which yields the claim since $H = H'$. ∎

Lemma 3.3.8 is then proven similarly to before with Lemma 3.7.1 in place of Lemma 3.3.6 and Lemma 3.7.2 in place of Lemma 3.3.7. Moreover, instead of proving $\mathfrak{M}_{\mathcal{D}}, H \vDash s$ by induction on the structure of s, we now proceed by induction on a modified notion of the **size of** s (notation $|s|$), which is defined as usual with the only exception being $|\neg p| = |p|$. This ensures that $|\sim s| = |s|$. Induction on $|s|$ works since we are working with negation normal terms, and the case $s = \neg p$ can be treated as a base case. We need to modify the induction order since then, the additional assumption in Lemma 3.7.2 follows by the inductive hypothesis of the proof of Lemma 3.3.8 (the additional assumption in Lemma 3.7.1 also follows by the inductive hypothesis of the proof of Lemma 3.3.8 with either induction order).

For demo existence (Theorem 3.3.13), we need to reprove Lemma 3.3.12, for which it suffices to extend the proof by one additional case (recall that we show $H_v \xrightarrow{\alpha}_{\mathcal{D}_{\mathfrak{M}}} H_w$ assuming $v \xrightarrow{\alpha}_{\mathfrak{M}} w$ by induction on α):

Let $\alpha = t$. Then $v = w$ and $\mathfrak{M}, v \vDash t$. Clearly, this implies $H_v = H_w$ and $t \in H_v$. Consequently, $H_v \xrightarrow{t}_{\mathcal{D}_{\mathfrak{M}}} H_w$.

The formulation of the decision procedure is unaffected by the addition of tests. The only difference is that now computing the relation $\xrightarrow{\alpha}_S$ for a given Hintikka system S (which is necessary for pruning) may involve computing $\xrightarrow{s}_S$ for some tests s. Clearly, this can be done in polynomial time with respect to $|S|$ and $|\mathcal{F}|$.

3.8 Converse

The extension with converse is straightforward. The definition of $\xrightarrow{\alpha}_S$ (Definition 3.3.1) is adapted by replacing the case for $\xrightarrow{a}_S$ with

$$\xrightarrow{a}_S \;=\; \{\,(H,H') \in S \times S \mid \forall s\colon\; ([a]s \in H \;\Rightarrow\; s \in H') \;\text{ and }\; ([a^-]s \in H' \;\Rightarrow\; s \in H)\,\}$$

and adding the following new case: $\xrightarrow{a^-}_S \;=\; \xrightarrow{a}_S^{-1}$. All proofs in § 3.3 and § 3.4 go through after adding a case for converse in three places:

- Proof of Lemma 3.3.6:

  The claim for $\alpha = a^-$ follows by observing that $\xrightarrow{a^-}_S = \xrightarrow{a}_S^{-1} \subseteq \xrightarrow{a}_{\mathfrak{M}_S}^{-1} = \xrightarrow{a^-}_{\mathfrak{M}_S}$ since $\xrightarrow{a}_S \subseteq \xrightarrow{a}_{\mathfrak{M}_S}$.
- Proof of Lemma 3.3.7:

  Let $\alpha = a^-$. By assumption, we have $[a^-]s \in H \xrightarrow{a^-}_{\mathfrak{M}_S} H'$. Then $H' \xrightarrow{a}_{\mathfrak{M}_S} H$ and hence $H' \xrightarrow{a}_S H$. The claim, $s \in H'$, follows by the definition of $\xrightarrow{a}_S$.
- Proof of Lemma 3.3.12:

  The claim for $\alpha = a^-$ follows analogously to the argument for a after observing that $v \xrightarrow{a^-}_{\mathfrak{M}} w$ if and only if $w \xrightarrow{a}_{\mathfrak{M}} v$, and $H_v \xrightarrow{a^-}_{\mathcal{D}_{\mathfrak{M}}} H_w$ if and only if $H_w \xrightarrow{a}_{\mathcal{D}_{\mathfrak{M}}} H_v$.

The decision procedure requires no modifications except that now the computation of $\xrightarrow{\alpha}_S$, which is necessary for pruning, needs to take converse into account. Summing up, the changes needed to incorporate converse are very minor.

3.9 Related Work

► The methods related to demos used in this chapter are strongly inspired by Fischer and Ladner's [67] filtration argument for PDL. Demos constitute a syntactic representation of the quotient structures used by Fischer and Ladner. Also, the proofs of demo satisfaction and demo existence (Theorems 3.3.9 and 3.3.13) follow the general ideas in Fischer and Ladner's work. Unlike in their construction but similar to [103, 27], we do not derive the transition relations for demos by projection of the corresponding relations in the original model, but instead define them as the largest relations compatible with the box formulas in the Hintikka sets. Using the terminology of Blackburn et al. [27], one can say that Fischer and Ladner work with *smallest filtrations* while demos correspond to *largest filtrations*.
Our proofs of demo satisfaction and demo existence differ notably from proofs of Fischer and Ladner's filtration theorem in the literature [67, 103, 138, 104]. There, tests cause considerable technical complications as compared to the test-free case, forcing the authors to use mutual recursion and special-purpose induction orders (two such orders being made explicit in [67, 138]). In our case, adapting the proofs to deal with tests turns out to require only minor changes to the formulation and the proofs of Lemmas 3.3.6 and 3.3.7.

► Syntactic constructions related to demos are used by Pratt [160] (see also [103, 138, 104, 27]) or, in the context of temporal logics, by Emerson and Halpern [59, 57] under the name “Hintikka structures”. The constructions in [59, 57] are more flexible but also more involved than demos since they represent the transition relations explicitly and allow, in principle, to have several distinct copies of the same Hintikka set.
Further syntactic constructions related to demos include canonical models (see [144], § 2, [41], § 5, or [27], § 4) or model graphs [168]. While closely related to Emerson and Halpern’s Hintikka structures, these constructions are conceptually more involved since they build on the notion of consistency, which requires additional proof-theoretic machinery.
► Alternative correctness proofs for Pratt’s decision procedure in [160] can be found in [138, 104, 27]. Unlike our proof in § 3.4, they involve complex semantic arguments, essentially reproving the small model property of PDL. In contrast, our correctness proof delegates all semantic concerns to Corollary 3.3.16. By using demos as a syntactic interface, we are able to reuse the small model results established by filtration for the decision procedure.

4 Graph Search for Modal Logic with Eventualities

This chapter presents a simple theory explaining the construction and the correctness of an incremental and worst-case optimal decision procedure for modal logic with eventualities.

Although worst-case optimal, procedures following Pratt's approach in [160] as presented in Chapter 3 are not incremental since they always construct all Hintikka subsets of the formula universe. A more elaborate decision procedure for PDL was presented by Pratt in [161]. The procedure employs graph-like structures we call *graph tableaux*. Given a formula, this procedure first constructs a complete graph tableau for the formula, after which it computes the largest demo contained in the tableau by pruning (returning satisfiable if the formula is contained in the demo and unsatisfiable otherwise). While an improvement over the procedures in Chapter 3, Pratt's tableau procedure is still nonincremental since it needs to construct a complete graph tableau before it can apply pruning. As shown by Goré, Nguyen and Widmann [91, 93, 200], graph tableaux can serve as a basis for incremental and worst-case optimal procedures that are obtained by interleaving the tableau construction with pruning. Such procedures can often decide the satisfiability of a formula without building a complete tableau.

We present a theory explaining the construction and the correctness of such an incremental procedure for K*. The procedure covered by the theory is a simplified and more abstract variant of Goré and Widmann's [93, 200] decision procedure for PDL. The key idea behind our theory is to refine pruning as presented in Chapter 3 to two complementary algorithms, called *eager* and *cautious pruning*, which check whether a graph tableau certifies the satisfiability or the unsatisfiability of a formula. The decision procedure then incrementally constructs a tableau certificate for the input formula, where eager and cautious pruning guide the expansion of the tableau.

In the development, we first introduce eager pruning, which already suffices to formulate a variant of Pratt's tableau procedure from [161]. We then introduce cautious pruning, which allows us to further refine the procedure to an incremental procedure in the style of Goré and Widmann [93, 200]. We define graph tableaux based on a notion of *clauses* that is more restricted than that of Hintikka sets used in Chapter 3. To simplify correctness arguments, we also introduce a representation of demos based on clauses and a concomitant notion of *support* relating clausal demos to Hintikka demos.

We do not treat full PDL, restricting ourselves to K*, to simplify the presentation and to separate the fundamental algorithmic problem of dealing with eventualities from technical complications caused by the complex syntax of PDL. We expect that our theory extends to full PDL.

Structure of the chapter. We begin by introducing the notions of clauses, support, and clausal demos (§ 4.1). We then reformulate pruning from Chapter 3 as a marking procedure (§ 4.2). In this form, it will serve as a blueprint for the pruning operations on tableaux. Then, we define graph tableaux (§ 4.3) and establish the subclass of evident tableaux (§ 4.4), which certify the satisfiability of formulas. We introduce eager (§ 4.5) and cautious pruning (§ 4.6), two complementary algorithms identifying, respectively, satisfiable and unsatisfiable tableau nodes. Once we have eager and cautious pruning, we formulate the incremental decision procedure (§ 4.7). We conclude with a discussion of related work (§ 4.8).

4.1 Formulas and Demos

We consider the unimodal basic modal logic with eventualities K* and restrict ourselves to formulas in negation normal form. The resulting syntax can be defined with the following grammar.

$$\begin{aligned} s ::= \quad & p \mid s \vee s \mid \Diamond s \mid \Diamond^* s \\ \mid\ & \neg p \mid s \wedge s \mid \Box s \mid \Box^* s \end{aligned}$$

We write $\Diamond^+ s$ for $\Diamond\Diamond^* s$ and $\Box^+ s$ for $\Box\Box^* s$. Recall that in § 2.2.4, we defined **eventualities** as formulas of the form $\Diamond^* s$. For convenience, from now on, formulas of the form $\Diamond^+ s$ will also be called eventualities. Diamond formulas that are not of the form $\Diamond^* s$ or $\Diamond^+ s$ are called **simple**. In analogy to the definitions in § 3.1, **literals** are now formulas of the form p, $\neg p$, $\Diamond s$ and $\Box s$ (note that the definition of literals includes formulas of the form $\Diamond^+ s$ and $\Box^+ s$). Consequently, the only alpha formulas are those of the form $s \wedge t$ and $\Box^* s$, while the only beta formulas are $s \vee t$ and $\Diamond^* s$, where we have:

$$\Box^* s \ \equiv\ s \wedge \Box^+ s \qquad\qquad \Diamond^* s \ \equiv\ s \vee \Diamond^+ s$$

Note that every constituent of a nonliteral formula s is either a literal or a proper subformula of s (the only case when α_i is not a subformula of α is when $\alpha = \Box^* s$ and $\alpha_i = \Box^+ s$, where $\Box^+ s$ is clearly a literal; the same is true for beta formulas $\Diamond^* s$).

As before, we assume the formula universe $\mathcal{F}$ to be a global parameter of the decision procedure and tacitly assume $s \in \mathcal{F}$ and $A \subseteq \mathcal{F}$ for every formula s and every set of formulas A.

4.1.1 Clauses, Support and Demos

In § 3.3, we defined demos as Hintikka systems. While convenient for abstract procedures following Pratt's approach in [160], representing states by Hintikka sets introduces redundancy. Consider two Hintikka sets $H_1 = \{p, p \vee q_1\}$ and $H_2 = \{p, p \vee q_2\}$. Since the two sets agree on literals, it can be shown that for every Hintikka system S containing H_1 and H_2 and every formula s, we have $\mathfrak{M}_S, H_1 \models s$ if and only if $\mathfrak{M}_S, H_2 \models s$. The only difference between H_1 and H_2 is that H_1 explicitly guarantees the satisfaction of $p \vee q_1$ while H_2 does so for $p \vee q_2$. Suppose now we have an external mechanism that allows us to determine if a set satisfies a given nonliteral formula by looking only at its literal formulas. Then there is no need to distinguish between Hintikka sets like H_1 and H_2. Instead, we can represent the two sets, and possibly more, by a single set of literals that all of the represented sets agree on. Since in this section we are interested in minimizing redundancy, this is precisely what we are going to do. We will call sets of literals *normal clauses* and introduce the *support relation* as the external mechanism for determining if a normal clause satisfies a given formula.

We proceed as follows. A **clause** is a set $C \subseteq \mathcal{F}$ of literals and beta formulas. Our semantic definitions apply to clauses since they are sets of formulas. In particular, $\mathfrak{M}, w \models C$ if and only if $\mathfrak{M}, w \models s$ for every formula $s \in C$. A clause containing only literals is called a **normal clause** (**N-clause** for short) provided it contains no complementary pair p and $\neg p$.

The table in Figure 3.1 induces a corresponding **decomposition relation** as the least relation $\to$ on formulas such that, for every formula α (and β) and every $i \in \{1, 2\}$, we have $\alpha \to \alpha_i$ ($\beta \to \beta_i$). When we restrict the language to K*, the decomposition relation becomes terminating since the constituents of a nonliteral formula are literals (and $\to$ terminates on literals) or proper subformulas of the formula. Hence, the transitive closure of the decomposition relation for K* is a well-founded partial order on formulas. We call it the **decomposition order** for K*.

Definition 4.1.1 We define the **upward closure** of a formula set A as the least set B of formulas containing A such that:

$$\alpha \in B \iff \alpha_1 \in B \text{ and } \alpha_2 \in B$$
$$\beta \in B \iff \beta_1 \in B \text{ or } \beta_2 \in B$$

Let C be an N-clause and s a formula. We say C **supports** s and write $C \rhd s$ if s is contained in the upward closure of C. We see support as a relation between N-clauses and formulas. ◂

Note that if s is a literal, then $C \rhd s \iff s \in C$. Given an N-clause C, the upward closure of C satisfies the closure properties of a Hintikka set (see Proposition 4.1.4). Thus, we can see an N-clause C as a representation of all Hintikka subsets of the upward closure of C, and the support relation as the concomitant membership relation. We write $C \rhd A$ and say C **supports** A if $C \rhd s$ for every $s \in A$. A set S of clauses **supports** s (A) if S contains a clause that supports s (A).

Note that given an N-clause C and a formula s, the relation $C \rhd s$ is decidable in time linear in the size of s and the cardinality of C (where the decision procedure simply follows the recursive definition of the upward closure).

Lemma 4.1.2 Let $\mathfrak{M}$ be a model such that $\mathfrak{M}, w \models C$ and $C \rhd s$. Then $\mathfrak{M}, w \models s$.

Proof By induction on the decomposition order of s. We proceed by case analysis on s.

Let s be a literal. Then $C \rhd s$ implies $s \in C$, and the claim follows by assumption.

Let $s = \alpha$. Then $C \rhd \alpha_1$ and $C \rhd \alpha_2$. By the inductive hypothesis, we have $\mathfrak{M}, w \models \alpha_1$ and $\mathfrak{M}, w \models \alpha_2$. The claim follows. The case $s = \beta$ proceeds similarly. ∎

We now adapt the relation $\xrightarrow{\alpha}_S$ introduced in § 3.3.1 to sets of clauses. Since K* does not allow complex programs except for the reflexive-transitive closure, it suffices to define the relation for atomic transitions. The **request** of a clause C is $\mathfrak{R}C := \{\, s \mid \Box s \in C \,\}$. Given a set S of N-clauses, we define the **transition relation** $\to_S \subseteq S \times S$ as follows:

$$C \to_S D \iff D \rhd \mathfrak{R}C$$

We call an N-clause $C \in S$ **$\Diamond$-admissible** (in S) if it satisfies the following conditions:

1. If $\Diamond s \in C$, then $C \to_S D \rhd s$ for some $D \in S$.
2. If $\Diamond^+ s \in C$, then $C \to_S^+ D \rhd s$ for some $D \in S$.

With this, demos are now defined as follows.

Definition 4.1.3 A nonempty set of N-clauses is called a (**clausal**) **demo** if all of its clauses are $\Diamond$-admissible. ◂

We call demos as defined above clausal to distinguish them from demos as defined in § 3.3, which we also call **Hintikka demos**.

Adapting Example 3.3.5 to the current setup (K* and clausal demos) yields the following demo $\mathcal{D}$:

$$\begin{tikzcd}
\boxed{\Diamond^+ p,\ \Box\neg p} \arrow[r, bend left] & \boxed{\Diamond^+ p,\ \neg p} \arrow[l, bend left] \arrow[r, bend left] \arrow[loop above] & \boxed{p} \arrow[l, bend left] \arrow[loop above] \arrow[ll, bend left=40]
\end{tikzcd}$$

The demo consists of three clauses: $\{\Diamond^+ p,\ \Box\neg p\}$, $\{\Diamond^+ p,\ \neg p\}$ and $\{p\}$. The edges between the clauses depict the relation $\to_{\mathcal{D}}$.

4.1.2 Demo Satisfaction and Demo Existence

We now prove demo satisfaction and demo existence for clausal demos by establishing a correspondence between clausal demos and Hintikka demos.

First, we show that for every clausal demo S there is a Hintikka demo that contains every formula in $\mathcal{F}$ that is supported by S. We define $H_C := \{\, s \in \mathcal{F} \mid C \rhd s \,\}$ and $\mathcal{H}_S := \{\, H_C \mid C \in S \,\}$.

Proposition 4.1.4 Let C be an N-clause. Then H_C is a Hintikka set.

Proof $\{p, \neg p\} \nsubseteq H_C$ since H_C and C agree on literals and C is not clashing. The Hintikka conditions for alpha and beta formulas follow by the definition of support. ∎

Lemma 4.1.5 Let S be a demo and $C \to_S D$. Then $H_C \to_{\mathcal{H}_D} H_D$.

Proof The claim follows from $C \to_S D$ since C and H_C agree on literals and since, by definition, H_D contains every formula supported by D. ∎

Proposition 4.1.6 Let S be a clausal demo. Then $\mathcal{H}_S$ is a Hintikka demo.

Proof By Proposition 4.1.4, $\mathcal{H}_S$ is a Hintikka system. To show that $\mathcal{H}_S$ is $\Diamond$-admissible, it suffices (in the case of K*) to show that:
1. $\forall C \in S\colon\ \Diamond s \in H_C \implies \exists D \in S\colon\ H_C \to_{\mathcal{H}_S} H_D$ and $s \in H_D$.
2. $\forall C \in S\colon\ \Diamond^* s \in H_C \implies \exists D \in S\colon\ H_C \to^*_{\mathcal{H}_S} H_D$ and $s \in H_D$.

For claim (2), suppose $\Diamond^* s \in H_C \in \mathcal{H}_S$ for some $C \in S$. Then $s \in H_C$ or $\Diamond^+ s \in H_C$ since H is a Hintikka set. In the first case, the claim follows since $H \to^*_S H$. In the second case, the claim follows by Lemma 4.1.5 and the $\Diamond$-admissibility of S. Claim (1) is similar but simpler. ∎

Next, we show that every Hintikka demo S corresponds to a clausal demo that supports every formula contained in S. We define $\Lambda A := \{s \in A \mid s \text{ literal}\}$. Given a Hintikka system S, we write $\Lambda S := \{\Lambda H \mid H \in S\}$. Note that restricted to unimodal K*, the syntactic transition relations $\xrightarrow{\alpha}_S$ as defined in § 3.3.1 reduce to only two forms: an atomic transition relation $\to_S$ and its transitive closure.

Proposition 4.1.7 $\Lambda H \rhd H$

Proof Let H be a Hintikka set and $s \in H$. We show $\Lambda H \rhd s$ by induction on the decomposition order of s. If s is a literal, the claim is immediate since then $\Lambda H \rhd s \iff s \in \Lambda H$. If $s = \alpha$, then $\alpha_1 \in H$ or $\alpha_2 \in H$. By the inductive hypothesis, $\Lambda H \rhd \alpha_1$ or $\Lambda H \rhd \alpha_2$, and hence $\Lambda H \rhd \alpha$. The case for beta formulas proceeds similarly. ∎

Lemma 4.1.8 Let S be a Hintikka demo and $H \to_S H'$. Then $\Lambda H \to_{\Lambda S} \Lambda H'$.

Proof It suffices to show that $\Lambda H' \rhd \mathfrak{R}(\Lambda H)$, which follows by the definition of $\to_S$ (Definition 3.3.1 restricted to K*) and Proposition 4.1.7. ∎

Proposition 4.1.9 Let S be a Hintikka demo. Then ΛS is a clausal demo supporting every formula contained in S.

Proof Let S be a Hintikka demo. We have to show that ΛS is $\Diamond$-admissible, which follows from the $\Diamond$-admissibility of S with Lemma 4.1.8. Hence, ΛS is a demo. Now, let $s \in H \in S$. It suffices to show that $\Lambda H \rhd s$, which follows by Proposition 4.1.7. ∎

Together with Theorems 3.3.9 and 3.3.13, Propositions 4.1.6 and 4.1.9 yield demo satisfaction and demo existence for clausal demos.

Theorem 4.1.10 (Demo Satisfaction) If S is a clausal demo, then $\mathfrak{M}_{\mathcal{H}_S}$ satisfies every clause that S supports.

Proof Follows by Theorem 3.3.9 and Proposition 4.1.6. ∎

Theorem 4.1.11 (Demo Existence) Let $\mathfrak{M}$ be a model. Then $\Lambda\mathcal{D}_{\mathfrak{M}}$ is a clausal demo. Moreover, $\Lambda\mathcal{D}_{\mathfrak{M}}$ supports a formula s if and only if $\mathfrak{M}$ satisfies s.

Proof Follows by Theorem 3.3.13, Proposition 4.1.9. To show that $\mathfrak{M}$ satisfies every formula supported by $\Lambda\mathcal{D}_{\mathfrak{M}}$ we additionally need Lemma 4.1.2. ∎

Corollary 4.1.12 A formula is satisfiable if and only if it is supported by a clausal demo.

Remark 4.1.13 While certainly an important insight, generic demo existence results like the one in Theorem 4.1.11 will play no role for the correctness of the decision procedures in the present or the following chapters. The reason for this is that such generic results are too weak. To prove the correctness of a particular decision procedure one needs to show demo existence with respect to the restricted class of demos that can be produced by the procedure, which will always be smaller than the class obtained in Theorem 4.1.11. Still, we will continue giving generic demo existence results where appropriate since they are more elegant and much easier to establish than the results needed for specific decision procedures. ◂

As with Hintikka demos, clausal demos are closed under union. Hence, there is a largest demo that contains all satisfiable N-clauses. This yields the following corollary.

Corollary 4.1.14 A formula is satisfiable if and only if it is supported by the largest clausal demo.

Since for the rest of the chapter we will not have to consider Hintikka demos, we will refer to clausal demos simply as demos.

4.2 Pruning Revisited

In § 3.4 we detailed how to compute the largest demo contained in a given Hintikka system S. This approach can be easily adapted to the clausal setup. We can define a pruning relation on clause sets that stepwise removes clauses that are not $\Diamond$-admissible, eventually terminating with the largest demo contained in S.

In the following, it will be more convenient to assume a slightly different view of pruning. Instead of a procedure that deletes offending clauses, we will view pruning as a marking procedure that marks offending clauses in the initial clause set.

To distinguish it from the pruning relations to be defined later, we call the relation corresponding to this marking procedure *abstract pruning*. While abstract pruning will not be used by our final decision procedure, it provides a helpful blueprint for the more elaborate methods to be used.

Given a set X and an object x, we write $X;x$ for $X \cup \{x\}$. Let $\mathcal{U}$ be a set of N-clauses. We define the **abstract pruning relation** $\overset{\text{AP}}{\to}_{\mathcal{U}}$ as a binary relation on clause sets: $S \overset{\text{AP}}{\to}_{\mathcal{U}} S' \iff S \subsetneq S' \subseteq \mathcal{U}$ and there is a clause C such that $S;C = S'$ and C is not $\Diamond$-admissible in $\mathcal{U} \setminus S$.

Intuitively, we think of the clauses in S and of C in the definition of $\overset{\text{AP}}{\to}_{\mathcal{U}}$ as clauses that are marked in $\mathcal{U}$ as bad. This is equivalent to thinking of them as being pruned from $\mathcal{U}$, which is the view assumed by the pruning relation $\overset{\text{P}}{\to}$ in § 3.4. In fact, if we adapt the relation $\overset{\mathrm{P}}{\to}$ to the current setup in the intuitive way, the two relations will satisfy the following equivalence: $S \overset{\mathrm{AP}}{\to}_{\mathcal{U}} S' \iff \mathcal{U} \setminus S \overset{\mathrm{P}}{\to} \mathcal{U} \setminus S'$. Clearly, the relation $\overset{\mathrm{AP}}{\to}_{\mathcal{U}}$ is terminating (since $\mathcal{U} \subseteq 2^{\mathcal{F}}$ and $\mathcal{F}$ is finite).

The following two lemmas restate Lemmas 3.4.2 and 3.4.3 for the current setup and the relation $\overset{\mathrm{AP}}{\to}_{\mathcal{U}}$. Their proofs proceed analogously to those of Lemmas 3.4.2 and 3.4.3.

Lemma 4.2.1 Let $\mathcal{D} \subseteq \mathcal{U}$ be a demo, $S \subseteq \mathcal{U} \setminus \mathcal{D}$, and $S \overset{\mathrm{AP}}{\to}_{\mathcal{U}} S'$. Then $S' \subseteq \mathcal{U} \setminus \mathcal{D}$.

Lemma 4.2.2 Let $S \overset{\mathrm{AP}}{\mapsto}_{\mathcal{U}} S'$. Then $\mathcal{U} \setminus S'$ is either empty or a demo.

Also, Proposition 3.4.4 adapts in the intuitive way (the proof remaining essentially unchanged), meaning that $\overset{\mathrm{AP}}{\to}_{\mathcal{U}}$ is strongly confluent for every $\mathcal{U}$. Theorem 3.4.5 is restated as follows.

Theorem 4.2.3 Let $\emptyset \overset{\mathrm{AP}}{\mapsto}_{\mathcal{U}} S$. Then $\mathcal{U} \setminus S$ is either empty or the largest demo contained in $\mathcal{U}$. Moreover, $\mathcal{U}$ contains a demo if and only if $\mathcal{U} \setminus S$ is nonempty.

Proof Follows with Lemmas 4.2.1 and 4.2.2. ∎

4.3 Graph Tableaux

A decision procedure that starts from the set of all N-clauses cannot be practical. Instead, we aim at a procedure that starts from a clause representing the input formula and that incrementally builds a graph tableau by adding clauses and *links*. A link is a triple $C\xi D$ where the *annotation* ξ documents how the clause D was obtained from the clause C. The procedure stops if the tableau determines the satisfiability or unsatisfiability of the input formula. Since the satisfiability or unsatisfiability of a formula can often be determined with a small tableau, the procedure provides for a practical prover. Figure 4.1 gives a first impression of what a graph tableau looks like.

Given a formula s, we define the **induced clause** $\lfloor s \rfloor$ by induction on s:

$$\begin{aligned} \lfloor \alpha \rfloor &:= \lfloor \alpha_1 \rfloor \cup \lfloor \alpha_2 \rfloor \\ \lfloor s \rfloor &:= \{s\} \qquad\qquad \text{if } s \text{ not an alpha formula} \end{aligned}$$

Intuitively, the induced clause is obtained by decomposing all top-level alpha formulas. The **induced clause** $\lfloor A \rfloor$ for a set A of formulas is the union of the induced clauses for the formulas in A. We have $\mathfrak{M}, w \vDash A$ if and only if $\mathfrak{M}, w \vDash \lfloor A \rfloor$. The same holds for support:

Proposition 4.3.1 $C \rhd A \iff C \rhd \lfloor A \rfloor$

Proof Let $s \in A$. It suffices to show that $C \rhd s$ if and only if $C \rhd \lfloor s \rfloor$, which follows by induction on the decomposition order of s. ∎

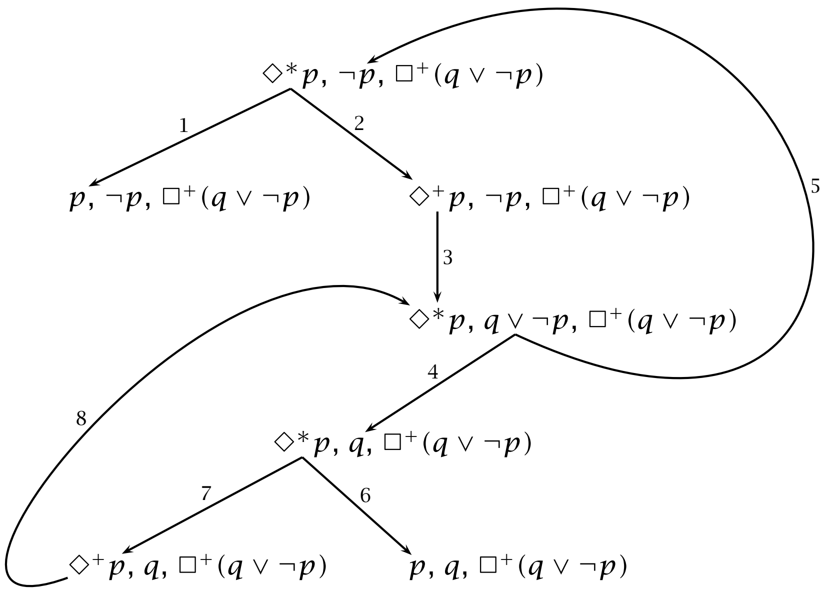

$\Diamond^* p,\ \neg p,\ \Box^+(q \vee \neg p)$
1
2
$p,\ \neg p,\ \Box^+(q \vee \neg p)$
$\Diamond^+ p,\ \neg p,\ \Box^+(q \vee \neg p)$
5
3
$\Diamond^* p,\ q \vee \neg p,\ \Box^+(q \vee \neg p)$
4
8
$\Diamond^* p,\ q,\ \Box^+(q \vee \neg p)$
7
6
$\Diamond^+ p,\ q,\ \Box^+(q \vee \neg p)$
$p,\ q,\ \Box^+(q \vee \neg p)$

Figure 4.1: A complete tableau

We partition the set of clauses into clashing clauses, normal clauses and branching clauses. A **clashing clause** (**C-clause** for short) is a clause that contains a pair p and $\neg p$. A **branching clause** (**B-clause**) is a clause that is not clashing and that contains a beta formula.

We distinguish between normal links and branching links. A **normal link** (**N-link**) is a triple $C\binom{\Diamond s}{s}D$ such that C is normal, $\Diamond s \in C$ and $D = \lfloor \mathfrak{R} C\, ; s \rfloor$. A **branching link** (**B-link**) is a triple $C\binom{\beta}{\beta_i}D$ ($i \in \{1, 2\}$) such that C is branching, $\beta \in C$, and $D = (C \setminus \{\beta\}) \cup \lfloor \beta_i \rfloor$.

A **tableau** $\mathcal{T}$ is a set of clauses and links such that the following conditions are satisfied:

1. If $C \xi D \in \mathcal{T}$, then C and D are in $\mathcal{T}$.
2. If $C\binom{s}{t}D$ and $C\binom{u}{v}E$ are B-links in $\mathcal{T}$, then $s = u$.

A graphical representation of a tableau is given in Figure 4.1. It contains one clashed clause, $\{p,\ \neg p,\ \Box^+(q \vee \neg p)\}$, three branching clauses and three normal clauses. Links are represented as arrows. More concretely, a link $C\binom{s}{t}D$ is represented as an arrow that departs from the formula s in C and points to the formula t in D. We see a tableau as a directed graph where the nodes are clauses and the edges are labeled with one or several binomial annotations. Given a tableau, we call a clause D a ξ**-successor** of a clause C if the tableau contains the link $C \xi D$. Note that a B-clause can have at most two successors in a tableau.

A **path** is a possibly empty sequence $(C_0 \xi_0 C_1)(C_1 \xi_1 C_2) \ldots (C_{n-1} \xi_{n-1} C_n)$ of links. We say that a tableau **contains a path** or that a **path is in a tableau** if the tableau contains every link of the path. Given a tableau $\mathcal{T}$, a clause D is **reachable** from a clause C if

$\mathcal{T}$ contains a path whose first clause is C and whose last clause is D. In the tableau in Figure 4.1, every clause is reachable from the topmost clause, which is a B-clause with two successors.

Let $\mathcal{T}$ and $\mathcal{T}'$ be two tableaux such that $\mathcal{T} \subseteq \mathcal{T}'$. Then we call $\mathcal{T}$ a **subtableau** of $\mathcal{T}'$ and $\mathcal{T}'$ a **supertableau** of $\mathcal{T}$. Moreover, we call a clause $C \in \mathcal{T}$ **expanded** if every link $C\xi D$ that appears in some supertableau of $\mathcal{T}$ already appears in $\mathcal{T}$. We call a tableau **complete** if each of its clauses is expanded. Note that the tableau in Figure 4.1 is complete. Given a tableau $\mathcal{T}$, one can compute a complete tableau that contains $\mathcal{T}$. Since only one disjunction in a B-clause can be expanded (condition (2) in the definition of tableaux), there may be different completions of a tableau.

4.4 Evidence

The idea of a demo carries over to tableaux. We define a class of *evident tableaux* such that we obtain two properties:

1. The N-clauses of an evident tableau comprise a demo supporting all clauses of the tableau.
2. Evidence of a tableau can be checked without checking support by just looking at the clauses and links of the tableau.

An evident tableau $\mathcal{T}$ must contain for every eventuality $\Diamond^* s \in C \in \mathcal{T}$ or $\Diamond^+ s \in C \in \mathcal{T}$ a fulfilling path that starts at C and ends with a B-link annotated with $\binom{\Diamond^* s}{s}$. We call such paths runs. In the tableau in Figure 4.1, the eventuality $\Diamond^* p$ in the topmost clause has a run consisting of the links 2, 3, 4, and 6. The eventuality $\Diamond^+ p$ in the clause on the bottom left has a run consisting of the links 8, 4, and 6. Precise definitions follow.

We use the notation $\Diamond^\sigma s$ to denote eventualities of the form $\Diamond^* s$ or $\Diamond^+ s$. A **claim** $\Diamond^\sigma s \mid C$ is a pair of a clause C and an eventuality $\Diamond^\sigma s \in C$. We define inductively what it means for a path π to be a **run for a claim** γ:

1. If $\pi = C\binom{\Diamond^* s}{s}D$, then π is a run for $\Diamond^* s \mid C$.
2. If $\pi = (C\binom{\Diamond^+ s}{\Diamond^* s}D)\pi'$ and π' is a run for $\Diamond^* s \mid D$, then π is a run for $\Diamond^+ s \mid C$.
3. If $\pi = (C\binom{\Diamond^* s}{\Diamond^+ s}D)\pi'$ and π' is a run for $\Diamond^+ s \mid D$, then π is a run for $\Diamond^* s \mid C$.
4. If $\pi = (C\binom{\beta}{\beta_i}D)\pi'$ ($i \in \{1, 2\}$), π' is a run for $\Diamond^\sigma s \mid D$, and $\Diamond^\sigma s \in C$, then π is a run for $\Diamond^\sigma s \mid C$.

A tableau $\mathcal{T}$ is **evident** if it satisfies the following conditions:

1. $\mathcal{T}$ contains no C-clause.
2. If $\Diamond s \in C \in \mathcal{T}$ and C is normal, then $\mathcal{T}$ contains a link $C\binom{\Diamond s}{s}D$.
3. If $C \in \mathcal{T}$ is branching, then C has at least one successor in $\mathcal{T}$.
4. If $\Diamond^\sigma s \in C \in \mathcal{T}$, then $\mathcal{T}$ contains a run for $\Diamond^\sigma s \mid C$.

Note that the empty tableau is evident. Also, condition (2) requires nothing other than that every N-clause in $\mathcal{T}$ is expanded.

A clause C is **evident in a tableau** $\mathcal{T}$ if C is contained in an evident subtableau of $\mathcal{T}$. We use $\mathcal{E}^{\mathcal{T}}$ to denote the set of all clauses that are evident in $\mathcal{T}$. Note that evident subtableaux of a given tableau are closed under union:

Proposition 4.4.1 Let $\mathcal{T}$ be a tableau and $\mathcal{T}_1$, $\mathcal{T}_2$ be evident subtableaux of $\mathcal{T}$. Then $\mathcal{T}_1 \cup \mathcal{T}_2$ is an evident subtableau of $\mathcal{T}$.

Proof Let $\mathcal{T}$, $\mathcal{T}_1$, $\mathcal{T}_2$ be as required. Clearly, $\mathcal{T}_1 \cup \mathcal{T}_2 \subseteq \mathcal{T}$. The tableau $\mathcal{T}_1 \cup \mathcal{T}_2$ satisfies condition (1) in the definition of tableaux since so do both $\mathcal{T}_1$ and $\mathcal{T}_2$. Condition (2) is satisfied since $\mathcal{T}_1 \cup \mathcal{T}_2 \subseteq \mathcal{T}$ and $\mathcal{T}$ is a tableau. The tableau $\mathcal{T}_1 \cup \mathcal{T}_2$ is evident since so are $\mathcal{T}_1$ and $\mathcal{T}_2$. ∎

Hence, the largest subtableau of $\mathcal{T}$ that contains only clauses from $\mathcal{E}^{\mathcal{T}}$ is precisely the largest evident subtableau of $\mathcal{T}$.

Example 4.4.2 Consider the tableau in Figure 4.1. The largest evident subtableau of this tableau is obtained by deleting link 1 and the C-clause it points to. The situation changes if we delete link 6. Then the largest evident subtableau consists of only one clause, namely $\{p, q, \Box^+(q \vee \neg p)\}$. This example tells us that non-evident clauses may still be satisfiable. ◂

Proposition 4.4.3 Let $\mathcal{T}$ and $\mathcal{U}$ be tableaux such that $\mathcal{T} \subseteq \mathcal{U}$. Then $\mathcal{E}^{\mathcal{T}} \subseteq \mathcal{E}^{\mathcal{U}}$.

Proof Follows since every evident subtableau of $\mathcal{T}$ is also a subtableau of $\mathcal{U}$. ∎

Proposition 4.4.4 If $C\xi D$ is a B-link and $E \rhd D$, then $E \rhd C$.

Proof Let $C\binom{\beta}{\beta_i}D$ ($i \in \{1, 2\}$) be a B-link and $E \rhd D$. Then $D = (C \setminus \{\beta\}) \cup \lfloor \beta_i \rfloor$. Since $E \rhd D$, we immediately have $E \rhd C \setminus \{\beta\}$. Moreover, since $E \rhd \lfloor \beta_i \rfloor$, we have $E \rhd \beta_i$ (Proposition 4.3.1), and hence $E \rhd \beta$. ∎

Lemma 4.4.5 Let $\mathcal{T}$ be an evident tableau and $C \in \mathcal{T}$ be a clause. Then there exists an N-clause $D \in \mathcal{T}$ that supports C.

Proof By induction on the sum of the sizes of the beta formulas in C using Proposition 4.4.4 and the following two observations:

1. Every clause in an evident tableau that is not normal is branching and has an outgoing B-link.
2. For every B-link $C\xi D$, the sum of the sizes of the beta formulas in C is strictly greater than that in D. ∎

Theorem 4.4.6 The N-clauses of a nonempty evident tableau comprise a demo that supports every clause of the tableau.

Proof Let $\mathcal{T}$ be a nonempty evident tableau, and let $\mathcal{D}$ be the set of all N-clauses in $\mathcal{T}$. By Lemma 4.4.5, $\mathcal{D}$ supports every clause of $\mathcal{T}$. It remains to show that $\mathcal{D}$ is a demo. Since $\mathcal{T}$ is nonempty, $\mathcal{D}$ is nonempty (again by Lemma 4.4.5). It now remains to show that all clauses in $\mathcal{D}$ are $\Diamond$-admissible.

Let $\Diamond s \in C \in \mathcal{D}$. Since $\mathcal{T}$ is evident, it contains a link $C\binom{\Diamond s}{s}D$ where $D = \lfloor \mathcal{R}C\,;s\rfloor$. Hence, by Lemmas 4.4.5 and 4.3.1, there is some $E \in \mathcal{D}$ such that $E \rhd \mathcal{R}C\,;s$. Therefore, we have $C \to_{\mathcal{D}} E \rhd s$.

Let $\Diamond^+ s \in C \in \mathcal{D}$. Since $\mathcal{T}$ is evident, it contains a run π for $\Diamond^+ s \mid C$. By definition, a run for $\Diamond^+ s \mid C$ must contain at least one N-link. We proceed by induction on the number n of N-links in π.

Let $n = 1$. Then $\pi = \lambda_1 \dots \lambda_k$ where $\lambda_1 = C\binom{\Diamond^+ s}{\Diamond^* s}C_1$, $\lambda_k = C_{k-1}\binom{\Diamond^* s}{s}C_k$, and λ_1 is the only N-link in π. Let $D \in \mathcal{D}$ be an N-clause that supports C_k (D exists by Lemma 4.4.5). Then $D \rhd s$ and, by Proposition 4.4.4 (applied k times), $D \rhd C_1$. Hence, we have $C \to_{\mathcal{D}} C_1 \rhd s$.

Let $n > 1$. Then $\pi = \lambda_1 \dots \lambda_k \pi'$ where $k \geq 2$, $\lambda_1 = C\binom{\Diamond^+ s}{\Diamond^* s}C_1$, $\lambda_k = C_{k-1}\binom{\Diamond^+ s}{\Diamond^* s}C_k$, and all of $\lambda_2, \dots, \lambda_{k-1}$ are branching. Since λ_k is normal, we have $C_{k-1} \in \mathcal{D}$. Moreover, $\Diamond^+ s \in C_{k-1}$ and $\lambda_k \pi'$ is a run for $\Diamond^+ s \mid C_{k-1}$. By the inductive hypothesis, we have $C_{k-1} \to_{\mathcal{D}}^{+} D \rhd s$ for some $D \in \mathcal{D}$. By Proposition 4.4.4 (applied $k-1$ times), $C_{k-1} \rhd C_1$, and hence $C \to_{\mathcal{D}} C_{k-1}$. The claim follows. ∎

Combining Theorems 4.1.10 and 4.4.6, we obtain that evident tableaux certify the satisfiability of all of their clauses:

Corollary 4.4.7 If a clause is evident in a tableau, then it is satisfiable.

As we saw earlier (Example 4.4.2), as long as a tableau is not complete, it may contain clauses that are non-evident but satisfiable. Now, we show that in complete tableaux, evidence and satisfiability coincide. For this, we first lift Theorem 4.1.11 from demos to evident tableaux. Let $\mathcal{T}$ be a tableau and $\mathfrak{M}$ be a model. We define $\mathcal{T} \mid \mathfrak{M}$ as the largest subtableau of $\mathcal{T}$ that contains only clauses satisfied by $\mathfrak{M}$.

Lemma 4.4.8 Let $\mathcal{T}$ be a complete tableau, $\mathfrak{M}$ be a model, and $\Diamond^\sigma s \in C \in \mathcal{T} \mid \mathfrak{M}$. Then $\Diamond^\sigma s \mid C$ has a run in $\mathcal{T} \mid \mathfrak{M}$.

Proof Given a clause D, we define $\sharp D$ as the sum of the sizes of the beta formulas in D. Recall that $\sharp C > \sharp D$ whenever $C \xi D$ is a B-link (first noted in the proof of Lemma 4.4.5).

For the proof, let us restate the claim more verbosely as follows: Let $\mathcal{T}$ be a complete tableau, $\mathfrak{M}$ be a model, $\Diamond^\sigma s \in C \in \mathcal{T} \mid \mathfrak{M}$, $\mathfrak{M}, v \vDash C$, $v \to_{\mathfrak{M}}^{n} w$, $\mathfrak{M}, w \vDash s$, and $n > 0$ if $\sigma = +$. Then $\Diamond^\sigma s \mid C$ has a run in $\mathcal{T} \mid \mathfrak{M}$.

We proceed by lexicographic induction on n and $\sharp C$. Let $\mathcal{T}$, $\mathfrak{M}$, $\Diamond^\sigma s \in C \in \mathcal{T} \mid \mathfrak{M}$, $v, w \in |\mathfrak{M}|$, and n be as required. We show that $\Diamond^\sigma s \mid C$ has a run in $\mathcal{T} \mid \mathfrak{M}$ by case analysis on n.

Let $n = 0$. Then $\sigma = *$. Therefore, C is branching and hence $\sharp C > 0$. Since $\mathcal{T}$ is complete, we have $C\binom{\Diamond^* s}{s}D \in \mathcal{T}$ where $D = (C \setminus \{\Diamond^* s\}) \cup \lfloor s \rfloor$. Since, by assumption, $\mathfrak{M}, v \vDash C$ and $\mathfrak{M}, v \vDash s$, we have $\mathfrak{M}, v \vDash D$, and hence $C\binom{\Diamond^* s}{s}D \in \mathcal{T} \mid \mathfrak{M}$. The claim follows.

Let $n > 0$. We distinguish two subcases.

Let $\sharp C = 0$. Then C is normal and $\sigma = +$. Since $\mathcal{T}$ is complete, we have $C\binom{\Diamond^+ s}{\Diamond^* s}D \in \mathcal{T}$ where $D = \lfloor \mathcal{R}C\,; \Diamond^* s\rfloor$. Since $\mathfrak{M}, v \vDash C$ and $\Diamond^+ s \in C$, there is some $v' \in |\mathfrak{M}|$ such that

$v \to_{\mathfrak{M}} v' \to_{\mathfrak{M}}^{n-1} w$. Clearly, $\mathfrak{M}, v' \vDash D$. Therefore, $C\binom{\Diamond^+ s}{\Diamond^* s}D \in \mathcal{T} \mid \mathfrak{M}$. By the inductive hypothesis for $n-1$ and $\sharp D$, $\Diamond^* s \mid D$ has a run π in $\mathcal{T} \mid \mathfrak{M}$. Then $(C\binom{\Diamond^+ s}{\Diamond^* s}D)\pi$ is a run for $\Diamond^+ s \mid C$ in $\mathcal{T} \mid \mathfrak{M}$.

Let $\sharp C > 0$. Then C is branching and, since $\mathcal{T}$ is complete, $\{C\binom{\beta}{\beta_1}D_1,\ C\binom{\beta}{\beta_2}D_2\} \subseteq \mathcal{T}$ for some β, D_1 and D_2. Since $\mathfrak{M}, v \vDash C$, we have $\mathfrak{M}, v \vDash D_1$ or $\mathfrak{M}, v \vDash D_2$. Hence, for some $i \in \{1, 2\}$, $C\binom{\beta}{\beta_i}D_i \in \mathcal{T} \mid \mathfrak{M}$. We distinguish two subcases.

If $\beta \neq \Diamond^\sigma s$, then $\Diamond^\sigma s \in D_i$. Since $\sharp D_i < \sharp C$, the inductive hypothesis applies to n and $\sharp D_i$, yielding that $\Diamond^\sigma s \mid D_i$ has a run π in $\mathcal{T} \mid \mathfrak{M}$. Then $(C\binom{\beta}{\beta_i}D_i)\pi$ is a run for $\Diamond^\sigma s \mid C$ in $\mathcal{T} \mid \mathfrak{M}$.

If $\beta = \Diamond^\sigma s$, then $\sigma = *$. By the inductive hypothesis for n and $\sharp D_i$, $\Diamond^+ s \mid D_i$ has a run π in $\mathcal{T} \mid \mathfrak{M}$. Then $(C\binom{\Diamond^* s}{\Diamond^+ s}D_i)\pi$ is a run for $\Diamond^* s \mid C$ in $\mathcal{T} \mid \mathfrak{M}$. ∎

Lemma 4.4.9 Let $\mathcal{T}$ be a complete tableau and $\mathfrak{M}$ be a model. Then $\mathcal{T} \mid \mathfrak{M}$ is an evident subtableau of $\mathcal{T}$.

Proof Let $\mathcal{T}$ and $\mathfrak{M}$ be as required. We show that $\mathcal{T} \mid \mathfrak{M}$ satisfies the four evidence conditions for tableaux. Clearly, $\mathcal{T} \mid \mathfrak{M}$ contains no C-clause.

Let $\Diamond s \in C \in \mathcal{T} \mid \mathfrak{M}$. Since $\mathcal{T}$ is complete, $C\binom{\Diamond s}{s}D \in \mathcal{T}$ where $D = \lfloor \mathfrak{R}C ; s \rfloor$. Since $\mathfrak{M}$ satisfies C, $\mathfrak{M}$ satisfies D, and therefore $C\binom{\Diamond s}{s}D \in \mathcal{T} \mid \mathfrak{M}$.

Let $C \in \mathcal{T} \mid \mathfrak{M}$ be branching. Since $\mathcal{T}$ is complete, we have $\{C\binom{\beta}{\beta_1}D_1,\ C\binom{\beta}{\beta_2}D_2\} \subseteq \mathcal{T}$ for some $\beta \in C$. Since $\mathfrak{M}$ satisfies C, $\mathfrak{M}$ satisfies D_1 or D_2. Therefore, for some $i \in \{1, 2\}$, $C\binom{\beta}{\beta_i}D_i \in \mathcal{T} \mid \mathfrak{M}$.

Finally, the condition for $\Diamond^\sigma s \in C \in \mathcal{T} \mid \mathfrak{M}$ follows by Lemma 4.4.8. ∎

Theorem 4.4.10 Let $\mathcal{T}$ be a complete tableau. Then a clause $C \in \mathcal{T}$ is satisfiable if and only if C is evident in $\mathcal{T}$.

Proof Follows with Lemma 4.4.9 and Corollary 4.4.7. ∎

Corollary 4.4.11 Let $\mathcal{T}$ be a complete tableau and $\lfloor s \rfloor \in \mathcal{T}$, Then s is satisfiable if and only if $\lfloor s \rfloor$ is evident in $\mathcal{T}$.

4.5 Eager Pruning

The idea of pruning carries over from demos to tableaux. Given a tableau, one stepwise marks clauses that cannot appear in an evident subtableau. We call this process eager pruning. When eager pruning terminates, exactly the non-evident clauses of the tableau are marked. Eager pruning gives us a worst-case optimal decision method: Given a formula s, construct a complete tableau containing $\lfloor s \rfloor$ and run eager pruning. Then s is satisfiable if and only if $\lfloor s \rfloor$ has not been marked. This decision method is a variant of Pratt's procedure in [161].

Let $\mathcal{T}$ be a tableau. We define the **eager pruning relation** $\overset{\text{EP}}{\to}_{\mathcal{T}}$ as a binary relation on clause sets: $S \overset{\text{EP}}{\to}_{\mathcal{T}} S' \iff S' \subsetneq S \subseteq \mathcal{T}$ and there is a clause C such that $S ; C = S'$ and one of the following conditions holds:

1. C is clashing.
2. C is an N-clause that is not expanded in $\mathcal{T}$.
3. C is an N-clause that has a successor in $\mathcal{T}$ that is in S.
4. C is a B-clause all of whose successors in $\mathcal{T}$ are in S.
5. C contains an eventuality s such that every run for $s \mid C$ in $\mathcal{T}$ contains a clause in S.

Note that $\overset{\text{EP}}{\to}_{\mathcal{T}}$ is a terminating relation. The following two lemmas capture the correctness of eager pruning analogously to how the correctness of abstract pruning follows from Lemmas 4.2.1 and 4.2.2.

Lemma 4.5.1 (Soundness) Let $S \subseteq \mathcal{T} \setminus \mathcal{E}^{\mathcal{T}}$ and $S \overset{\text{EP}}{\to}_{\mathcal{T}} S'$. Then $S' \subseteq \mathcal{T} \setminus \mathcal{E}^{\mathcal{T}}$.

Proof Let $S \subseteq \mathcal{T} \setminus \mathcal{E}^{\mathcal{T}}$ and $S \overset{\text{EP}}{\to}_{\mathcal{T}} S'$. Let $S' = S;C$. We proceed by case analysis on the properties of C in the definition of $\overset{\text{EP}}{\to}_{\mathcal{T}}$.

Clearly, if C is a C-clause or an N-clause that is not expanded in $\mathcal{T}$, it cannot possibly occur in an evident tableau, and hence $C \notin \mathcal{E}^{\mathcal{T}}$.

Now suppose C is an N-clause that has a successor in $\mathcal{T}$ that is in S. Since S and $\mathcal{E}^{\mathcal{T}}$ are disjoint, C cannot be expanded in any evident tableau, and hence $C \notin \mathcal{E}^{\mathcal{T}}$. The remaining two cases proceed similarly. ∎

Let $\mathcal{T}$ be a tableau and S be a set of clauses. We write $\mathcal{T}{-}S$ for the largest subtableau of $\mathcal{T}$ not containing a clause from S.

Lemma 4.5.2 Let $S \overset{\text{EP}}{\mapsto}_{\mathcal{T}} S'$. Then $\mathcal{T}{-}S'$ is an evident tableau.

Proof The claim is shown by case analysis on the definition of evident tableaux. Each of the cases easily follows by the definition of eager pruning. ∎

As with previous pruning relations, eager pruning is confluent, which easily follows with the following lemma:

Lemma 4.5.3 If $S \subseteq S'$ and $S \overset{\text{EP}}{\to}_{\mathcal{T}} S;C$, then $C \in S'$ or $S' \overset{\text{EP}}{\to}_{\mathcal{T}} S';C$.

Proof Straightforward case analysis on the definition of $\overset{\text{EP}}{\to}_{\mathcal{T}}$. ∎

As before, the confluence of $\overset{\text{EP}}{\to}_{\mathcal{T}}$ is not important for the following arguments, but allows us to think of eager pruning as a function.

Theorem 4.5.4 Let $\mathcal{T}$ be a tableau, $\mathcal{R} \subseteq \mathcal{T}$ be a set of unsatisfiable clauses, and $\mathcal{R} \overset{\text{EP}}{\mapsto}_{\mathcal{T}} S$. Then $\mathcal{E}^{\mathcal{T}} = \{\, C \in \mathcal{T} \mid C \notin S \,\}$.

Proof By Lemma 4.5.1, $S \cap \mathcal{E}^{\mathcal{T}} = \emptyset$ since $\mathcal{R} \cap \mathcal{E}^{\mathcal{T}} = \emptyset$. Thus, $\mathcal{E}^{\mathcal{T}} \subseteq \{\, C \in \mathcal{T} \mid C \notin S \,\}$. By Lemma 4.5.2, $\mathcal{T}{-}S$ is evident and hence $\{\, C \in \mathcal{T} \mid C \notin S \,\} \subseteq \mathcal{E}^{\mathcal{T}}$. ∎

Example 4.5.5 Consider the tableau in Figure 4.1. Eager pruning starting with the empty set of clauses marks the C-clause and no other clause. The situation changes if we apply eager pruning to the tableau with link 6 removed. Then all clauses of the tableau but $\{p,\ q,\ \Box^{+}(q \vee \neg p)\}$ are marked as non-evident. This is the case since all eventualities for p lose their runs. Compare this to Example 4.4.2 to verify that eager pruning computes (the complement of) the set of clauses in the largest evident subtableau. ◂

Remark 4.5.6 In the absence of eventualities, $\mathcal{E}^{\mathcal{T}}$ can be computed in a more direct, "inductive" way as the unique set $\mathcal{E}$ of clauses such that $\emptyset \overset{\text{EM}}{\mapsto}_{\mathcal{T}} \mathcal{E}$, where $S \overset{\text{EM}}{\to}_{\mathcal{T}} S' \iff S \subsetneq S' \subseteq \mathcal{T}$ and there is a clause C such that $S;C = S'$ and one of the following conditions holds:
1. C is an N-clause that is expanded and all of whose successors in $\mathcal{T}$ are in S.
2. C is a B-clause that has a successor in $\mathcal{T}$ that is in S.

Note that N-clauses that contain no diamond formulas satisfy condition (1).

Since rather than *pruning* clauses that *are not* evident in $\mathcal{T}$, the relation $\overset{\text{EM}}{\to}_{\mathcal{T}}$ *marks* clauses that *are* evident in $\mathcal{T}$, we call the relation $\overset{\text{EM}}{\to}_{\mathcal{T}}$ **eager marking.** ◂

4.6 Cautious Pruning

We define cautious pruning as a constrained form of eager pruning that marks unsatisfiable clauses. We base cautious pruning on the following consequence of Theorem 4.4.10.

Corollary 4.6.1 Let $\mathcal{T}$ be a tableau. Then a clause $C \in \mathcal{T}$ is unsatisfiable if and only if there is no complete supertableau of $\mathcal{T}$ in which C is evident.

Cautious pruning will mark a clause in a tableau if it is clear that the clause cannot become evident by adding further links and clauses. To account for eventualities, we need the notion of a *plan*. A plan is a path in a tableau that is either a run or a partial run that ends with a non-expanded clause. If a claim has a plan, it may have a run in a completion of the tableau. If, however, a claim has no plan, it cannot have a run in any completion of the tableau.

We define inductively what it means that a path π in a tableau $\mathcal{T}$ is a **plan for a claim γ in $\mathcal{T}$**:
1. If $\Diamond^+ s \in C \in \mathcal{T}$, C is normal and has no $\binom{\Diamond^+ s}{\Diamond^* s}$-successor in $\mathcal{T}$, then the empty path is a plan for $\Diamond^+ s \mid C$ in $\mathcal{T}$.
2. If $\Diamond^\sigma s \in C \in \mathcal{T}$, C is branching and not expanded in $\mathcal{T}$, then the empty path is a plan for $\Diamond^\sigma s \mid C$ in $\mathcal{T}$.
3. If $\pi = C\binom{\Diamond^* s}{s}D$, then π is a plan for $\Diamond^* s \mid C$ in $\mathcal{T}$.
4. If $\pi = (C\binom{\Diamond^+ s}{\Diamond^* s}D)\pi'$ and π' is a plan for $\Diamond^* s \mid D$ in $\mathcal{T}$, then π is a plan for $\Diamond^+ s \mid C$ in $\mathcal{T}$.
5. If $\pi = (C\binom{\Diamond^* s}{\Diamond^+ s}D)\pi'$ and π' is a plan for $\Diamond^+ s \mid D$ in $\mathcal{T}$, then π is a plan for $\Diamond^* s \mid C$ in $\mathcal{T}$.
6. If $\pi = (C\binom{\beta}{\beta_i}D)\pi'$ $(i \in \{1, 2\})$, π' is a plan for $\Diamond^\sigma s \mid D$ in $\mathcal{T}$, and $\Diamond^\sigma s \in C$, then π is a plan for $\Diamond^\sigma s \mid C$ in $\mathcal{T}$.

Lemma 4.6.2 Let $\mathcal{T}$ be a tableau and γ be a claim whose clause is in $\mathcal{T}$. Then:
1. Every run for γ in $\mathcal{T}$ is a plan for γ in $\mathcal{T}$.
2. Every plan π for γ in $\mathcal{T}$ such that the clause in γ and all clauses in π are expanded in $\mathcal{T}$ is a run for γ.

3. If γ has a plan π in a supertableau of $\mathcal{T}$, then it has a plan in $\mathcal{T}$ that is a prefix of π.

Proof
1. Immediate from the definitions of runs and plans.
2. The only plans in $\mathcal{T}$ that are not runs are those whose derivation involves conditions (1) and (2) in the definition of plans. Let π be a plan for $s \mid C$ in $\mathcal{T}$. If C and all clauses contained in π are expanded in $\mathcal{T}$, conditions (1) and (2) are not applicable. Hence, π is a run. Formally, the argument proceeds by induction on the derivation of a plan.
3. Let $\gamma = \Diamond^\sigma s \mid C$ be a claim where $C \in \mathcal{T}$ and π a plan for γ in a supertableau of $\mathcal{T}$. Let π' be the largest prefix of π that is contained in $\mathcal{T}$. One shows that π' is a plan for γ in $\mathcal{T}$ by induction on the derivation of π. The argument is straightforward if $\pi' = \pi$. Otherwise, let $\pi = \pi'\pi''$ and D be the last clause in π' if π' is nonempty, and $D = C$ otherwise.
If D is normal, then (since $\pi' \neq \pi$) it has no $\binom{\Diamond^+ s}{\Diamond^* s}$-successor in $\mathcal{T}$, and so the empty path is a plan for $\Diamond^+ s \mid D$ in $\mathcal{T}$.
If D is branching, then D is not expanded in $\mathcal{T}$ (since $\pi' \neq \pi$), and so the empty path is a plan for $\Diamond^{\sigma'} s \mid D$ in $\mathcal{T}$ for every $\Diamond^{\sigma'} s \in D$.
In both cases, the argument proceeds by induction on the derivation of π from π'', with the empty plan for, respectively, $\Diamond^+ s \mid D$ or $\Diamond^{\sigma'} s \mid D$ being used in place of π''. ∎

Let $\mathcal{T}$ be a tableau. We define the **cautious pruning relation** $\overset{\text{CP}}{\to}_{\mathcal{T}}$ as a binary relation on clause sets: $S \overset{\text{CP}}{\to}_{\mathcal{T}} S' \iff S \subsetneq S' \subseteq \mathcal{T}$ and there is a clause C such that $S ; C = S'$ and one of the following conditions holds:
1. C is clashing.
2. C is an N-clause that has a successor in $\mathcal{T}$ that is in S.
3. C is a B-clause that is expanded in $\mathcal{T}$ and all successors of C in $\mathcal{T}$ are in S.
4. C contains an eventuality s such that every plan for $s \mid C$ in $\mathcal{T}$ contains a clause in S.

Consider the tableau in Figure 4.2. Cautious pruning will mark all clauses of the tableau since they are either clashing or contain an eventuality that has no plan. Note that the tableau is incomplete since the topmost clause has no successor for the formula $\Diamond q$. If we replace q with a complex formula, completing the tableau may be expensive.

Lemma 4.5.3 adapts to cautious pruning as follows.

Lemma 4.6.3 If $S \subseteq S'$ and $S \overset{\text{CP}}{\to}_{\mathcal{T}} S;C$, then $C \in S'$ or $S' \overset{\text{CP}}{\to}_{\mathcal{T}} S';C$.

Proof Straightforward case analysis on the definition of $\overset{\text{CP}}{\to}_{\mathcal{T}}$. ∎

Proposition 4.6.4 Let $\mathcal{T}$ be a tableau. Then $\overset{\text{CP}}{\to}_{\mathcal{T}} \subseteq \overset{\text{EP}}{\to}_{\mathcal{T}}$. Moreover, $\overset{\text{CP}}{\to}_{\mathcal{T}}$ is terminating and confluent. Hence there exists a unique set $\mathcal{R}$ such that $\emptyset \overset{\text{CP}}{\mapsto}_{\mathcal{T}} \mathcal{R}$.

Proof We have $\overset{\text{CP}}{\to}_{\mathcal{T}} \subseteq \overset{\text{EP}}{\to}_{\mathcal{T}}$ since every run is a plan (Lemma 4.6.2 (1)). Cautious pruning is terminating since there are only finitely many clauses (or since $\overset{\text{EP}}{\to}_{\mathcal{T}}$ is terminating and $\overset{\text{CP}}{\to}_{\mathcal{T}} \subseteq \overset{\text{EP}}{\to}_{\mathcal{T}}$). The confluence of $\overset{\text{CP}}{\to}_{\mathcal{T}}$ follows with Lemma 4.6.3. ∎

$$\begin{tikzcd}
& \Diamond^+ p,\ \Box^+\neg p,\ \Diamond q \arrow[d, "1"] & \\
& \Diamond^* p,\ \neg p,\ \Box^+\neg p \arrow[dl, "3"'] \arrow[dr, "2"] & \\
\Diamond^+ p,\ \neg p,\ \Box^+\neg p \arrow[ur, bend left=60, "4"] & & p,\ \neg p,\ \Box^+\neg p
\end{tikzcd}$$

Figure 4.2: An incomplete tableau

Given a tableau $\mathcal{T}$, we use $\mathcal{R}^{\mathcal{T}}$ to denote the unique set $\mathcal{R}$ of clauses such that $\emptyset \overset{\text{CP}}{\mapsto}_{\mathcal{T}} \mathcal{R}$. We say that a clause C is **refuted in** $\mathcal{T}$ if $C \in \mathcal{R}^{\mathcal{T}}$.

Proposition 4.6.5 Let $S \overset{\text{CP}}{\rightarrow}_{\mathcal{T}} S'$ and $S \subseteq \mathcal{R}^{\mathcal{T}}$. Then $S' \subseteq \mathcal{R}^{\mathcal{T}}$.

Proof Let $S \overset{\text{CP}}{\rightarrow}_{\mathcal{T}} S'$ and $S \subseteq \mathcal{R}^{\mathcal{T}}$. Since $S \subseteq \mathcal{R}^{\mathcal{T}}$, there is some S'' such that $S \subseteq S'' \overset{\text{CP}}{\mapsto}_{\mathcal{T}} \mathcal{R}^{\mathcal{T}}$. By Lemma 4.6.3, we have $S' \subseteq S'' \subseteq \mathcal{R}^{\mathcal{T}}$ or $S'' \overset{\text{CP}}{\rightarrow}_{\mathcal{T}} S'' \cup S'$. The claim follows since, by the confluence of cautious pruning, $S'' \cup S' \overset{\text{CP}}{\mapsto}_{\mathcal{T}} \mathcal{R}^{\mathcal{T}}$. ∎

Since the idea behind cautious pruning is to mark clauses that cannot become evident in a completion of a given tableau, on complete tableaux cautious pruning coincides with eager pruning:

Proposition 4.6.6 If $\mathcal{T}$ is a complete tableau, then $\overset{\text{CP}}{\rightarrow}_{\mathcal{T}} = \overset{\text{EP}}{\rightarrow}_{\mathcal{T}}$.

Proof Let $\mathcal{T}$ be a complete tableau. Then condition (2) in the definition of eager pruning is not applicable, and condition (4) for eager pruning coincides with condition (3) for cautious pruning. Condition (5) for eager pruning and condition (4) for cautious pruning coincide by Lemma 4.6.2 (1, 2). ∎

Lemma 4.6.7 (Progress) Let $S = \{\, C \in \mathcal{T} \mid C \notin \mathcal{R}^{\mathcal{T}} \cup \mathcal{E}^{\mathcal{T}} \,\}$. If $S \neq \emptyset$, then S contains a clause that is not expanded in $\mathcal{T}$.

Proof The claim is immediate by Proposition 4.6.6. ∎

Lemma 4.6.8 Let $\mathcal{U}$ and $\mathcal{T}$ be tableaux such that $\mathcal{U} \subseteq \mathcal{T}$. Then $\overset{\text{CP}}{\rightarrow}_{\mathcal{U}} \subseteq \overset{\text{CP}}{\rightarrow}_{\mathcal{T}}$ and $\mathcal{R}^{\mathcal{U}} \subseteq \mathcal{R}^{\mathcal{T}}$.

Proof Straightforward case analysis on the definition of $\overset{\text{CP}}{\rightarrow}_{\mathcal{U}}$. The case for condition (4) follows with Lemma 4.6.2 (3). ∎

Lemma 4.6.8 states that cautious pruning is monotone with respect to the underlying tableau. This is not the case for eager pruning. For instance, let $\mathcal{U} = \{\{\Diamond p\}\}$ and $\mathcal{T} = \{\{\Diamond p\}, \{\Diamond p\}\binom{\Diamond p}{p}\{p\}, \{p\}\}$. Then $\emptyset \overset{\text{EP}}{\rightarrow}_{\mathcal{U}} \{\{\Diamond p\}\}$ but $\emptyset \overset{\text{EP}}{\mapsto}_{\mathcal{T}} \emptyset$.

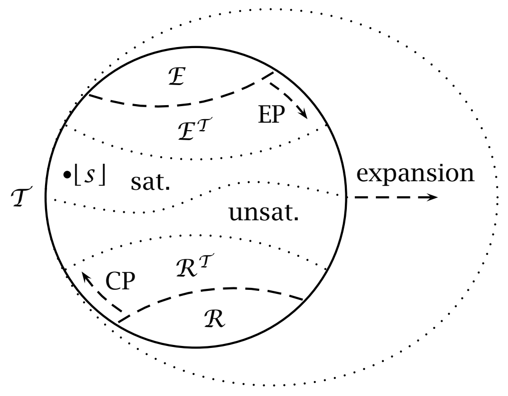

$\mathcal{E}$
$\mathcal{E}^{\mathcal{T}}$
EP
$\bullet\lfloor s \rfloor$
sat.
unsat.
expansion
$\mathcal{T}$
$\mathcal{R}^{\mathcal{T}}$
CP
$\mathcal{R}$

Figure 4.3: Schematic overview of IDP

Lemma 4.6.9 Let every clause in S be unsatisfiable and $S \overset{\text{CP}}{\to}_{\mathcal{T}} S'$. Then every clause in S' is unsatisfiable.

Proof Consider the four cases in the definition of $\overset{\text{CP}}{\to}_{\mathcal{T}}$. Except for the last case, the claim is obvious. Now let $C \in \mathcal{T}$ be a clause that contains an eventuality s such that every plan for $s \mid C$ in $\mathcal{T}$ contains a clause in S. Furthermore, let $\mathcal{T}'$ be a complete supertableau of $\mathcal{T}$. Suppose C is satisfiable. By Theorem 4.4.10, C is evident in $\mathcal{T}'$. Hence, there is an evident subtableau of $\mathcal{T}'$ that contains a run π for $s \mid C$. Since the clauses in S are unsatisfiable, it follows with Corollary 4.4.7 that π contains no clause in S. By Lemma 4.6.2 (1, 3), some prefix of π is a plan for $s \mid C$ in $\mathcal{T}$. Contradiction. ∎

Theorem 4.6.10 (Soundness) Every clause in $\mathcal{R}^{\mathcal{T}}$ is unsatisfiable.

Proof Follows from Lemma 4.6.9. ∎

4.7 Decision Procedure

We now have everything in place to formulate an incremental decision procedure. Starting from an input formula s, the procedure grows a tableau $\mathcal{T}$ and two clause sets $\mathcal{R} \subseteq \mathcal{T}$ and $\mathcal{E} \subseteq \mathcal{T}$ such that all clauses in $\mathcal{R}$ are refuted and all clauses in $\mathcal{E}$ are evident in $\mathcal{T}$. The set $\mathcal{R}$ is grown by cautious pruning, and the set $\mathcal{E}$ is grown by eager pruning (taking complements). The procedure stops once the initial clause $\lfloor s \rfloor$ appears in $\mathcal{R}$ or $\mathcal{E}$. Figure 4.3 gives an schematic overview and Figure 4.4 a precise formulation of the procedure, which we name IDP.

We argue the correctness of IDP as follows:

1. Preservation of the invariant for $\mathcal{R}$ (i.e., $\mathcal{R} \subseteq \mathcal{R}^{\mathcal{T}}$) follows from Lemma 4.6.8 and Proposition 4.6.5. Preservation of the invariant for $\mathcal{E}$ follows from Theorems 4.6.10 and 4.5.4, and Proposition 4.4.3.

```
Input:   a formula $s \in \mathcal{F}$
Variables:   $\mathcal{T} := \{\lfloor s \rfloor\}$,   $\mathcal{R} := \emptyset$,   $\mathcal{E} := \emptyset$
Invariants:
– $\mathcal{T}$ is a tableau such that every clause is reachable from $\lfloor s \rfloor$
– $\mathcal{R} \subseteq \mathcal{R}^{\mathcal{T}}$ and $\mathcal{E} \subseteq \mathcal{E}^{\mathcal{T}}$
while $\lfloor s \rfloor \notin \mathcal{R} \cup \mathcal{E}$ do one of the following:
– Grow $\mathcal{T}$ by adding a link to a clause in $\mathcal{T} \setminus (\mathcal{R} \cup \mathcal{E})$
– Grow $\mathcal{R}$ with cautious pruning, that is, $\mathcal{R} := \mathcal{R}'$ where $\mathcal{R} \overset{\text{CP}}{\to}_{\mathcal{T}} \mathcal{R}'$
– Grow $\mathcal{E}$ with eager pruning, that is, $\mathcal{E} := \{\, C \in \mathcal{T} \mid C \notin S \,\}$ where $\mathcal{R} \overset{\text{EP}}{\mapsto}_{\mathcal{T}} S$
Output:   SAT if $\lfloor s \rfloor \in \mathcal{E}$ and UNSAT otherwise
```

Figure 4.4: Incremental decision procedure IDP

2. That the loop can perform at least one of the three possible steps follows from the loop condition, the invariants, and Lemma 4.6.7: If $\lfloor s \rfloor \notin \mathcal{R} \cup \mathcal{E}$, at least one of the following three cases must hold:
   - $\mathcal{E} \subsetneq \mathcal{E}^{\mathcal{T}}$. Then we can grow $\mathcal{E}$ with eager pruning.
   - $\mathcal{R} \subsetneq \mathcal{R}^{\mathcal{T}}$. Then we can grow $\mathcal{R}$ with cautious pruning.
   - $\lfloor s \rfloor \notin \mathcal{R}^{\mathcal{T}} \cup \mathcal{E}^{\mathcal{T}}$. Then we can grow $\mathcal{T}$ by Lemma 4.6.7.
3. The loop terminates since each iteration grows one of the sets $\mathcal{T}$, $\mathcal{R}$ and $\mathcal{E}$, and each of these sets is bounded in the finite formula universe $\mathcal{F}$.
4. Correctness of the output follows from the negated loop condition, the invariants, Corollary 4.4.7 and Theorem 4.6.10.

IDP runs in deterministic exponential time with respect to the size of s. To see this, note that every traversal of the loop adds at least one element to one of the sets $\mathcal{T}$, $\mathcal{R}$, and $\mathcal{E}$, and that the cardinality of each of these sets is exponentially bounded in $|\mathcal{F}|$ (which can be assumed linear in the size of s). Moreover, each of the three possible actions of the loop can be realized in polynomial time with respect to the size of $\mathcal{T}$, $\mathcal{R}$, and $\mathcal{E}$.

We call a tableau $\mathcal{T}$ a **certificate** for a formula s and say that $\mathcal{T}$ **determines** s if $\lfloor s \rfloor \in \mathcal{R}^{\mathcal{T}} \cup \mathcal{E}^{\mathcal{T}}$ and every clause in $\mathcal{T}$ is reachable from $\lfloor s \rfloor$. We speak of a **negative certificate** if $\lfloor s \rfloor \in \mathcal{R}^{\mathcal{T}}$ and of a **positive certificate** if $\lfloor s \rfloor \in \mathcal{E}^{\mathcal{T}}$. For example, the tableau in Figure 4.2 is a negative certificate for $\Diamond^{+}p \wedge \Box^{+}\neg p \wedge \Diamond q$, and the tableau in Figure 4.1 is a positive certificate for $\Diamond^{*}p \wedge \neg p \wedge \Box^{+}(q \vee \neg p)$. Note that it can be decided in polynomial time whether a tableau is a positive or a negative certificate for a formula (with respect to the size of the tableau).

Proposition 4.7.1 IDP always terminates with a tableau that determines the input formula. Thus, every formula has a certificate.

Proof Follows from the invariants and the negated loop condition. ∎

Vice versa, let $\mathcal{T}$ determine s. Then IDP can first construct the tableau $\mathcal{T}$ and then do the necessary pruning to establish $\mathcal{T}$ as a certificate of s. Thus, IDP can be efficient for formulas with small certificates. Of course, the question remains how IDP selects the right tableau expansion steps to construct $\mathcal{T}$. Given the complexity of the problem, there will be no perfect answer. Nevertheless, it is a good idea to grow $\mathcal{R}$ and $\mathcal{E}$ eagerly so that unnecessary expansions are avoided. Moreover, one can show that it is unnecessary to expand clauses that can only be reached from the initial clause by a path that passes through a clause in $\mathcal{R}$ or $\mathcal{E}$:

Lemma 4.7.2 Let $C \in \mathcal{T}$ and $S = \{D \in \mathcal{T} \mid D \text{ reachable in } \mathcal{T}-(\mathcal{R}^{\mathcal{T}} \cup \mathcal{E}^{\mathcal{T}}) \text{ from } C\}$. If $S \neq \emptyset$, then S contains a clause that is not expanded in $\mathcal{T}$.

Proof We obtain the claim by contradiction. Let $D \in S$ and suppose every clause in S is expanded in $\mathcal{T}$. Then D is expanded in $\mathcal{T}$. Let $\mathcal{U}$ be the largest subtableau of $\mathcal{T}$ that contains only the clauses in $S \cup \mathcal{E}^{\mathcal{T}}$. Since D violates an evidence condition for $\mathcal{U}$, there must be an eventuality $s \in D$ such that $s \mid D$ has no run in $\mathcal{U}$. Since $D \notin \mathcal{R}^{\mathcal{T}}$, the claim $s \mid D$ has a plan in $\mathcal{T}$ that does not contain a clause in $\mathcal{R}^{\mathcal{T}}$. By Lemma 4.6.2 (3), $s \mid D$ then has a plan in $\mathcal{U}$. Since every clause in S is expanded and every eventuality in the largest evident subtableau of $\mathcal{T}$ (which contains precisely the clauses in $\mathcal{E}^{\mathcal{T}}$ and is hence a subset of $\mathcal{U}$) has a run in $\mathcal{U}$, the claim $s \mid D$ has a run in $\mathcal{U}$ (follows with Lemma 4.6.2 (2)). Contradiction. ∎

Let $S[\mathcal{T}, s, \mathcal{R}, \mathcal{E}] := \{D \in \mathcal{T} \mid D \text{ reachable in } \mathcal{T}-(\mathcal{R} \cup \mathcal{E}) \text{ from } \lfloor s \rfloor\}$. If we instantiate the clause C in Lemma 4.7.2 with $\lfloor s \rfloor$, we obtain that we can always grow $\mathcal{T}$ by adding a link to a clause in $S[\mathcal{T}, s, \mathcal{R}, \mathcal{E}]$ (rather than $\mathcal{T} \setminus (\mathcal{R} \cup \mathcal{E})$) whenever $\lfloor s \rfloor \notin \mathcal{R} \cup \mathcal{E}$, $\mathcal{R} = \mathcal{R}^{\mathcal{T}}$ and $\mathcal{E} = \mathcal{E}^{\mathcal{T}}$. Since $S[\mathcal{T}, s, \mathcal{R}, \mathcal{E}]$ is clearly a subset, and often a proper subset, of the clauses in $\mathcal{T} \setminus (\mathcal{R} \cup \mathcal{E})$, this allows us to further narrow down the search space for IDP without sacrificing correctness.

4.8 Related Work

► As already mentioned at the beginning of the chapter, the first decision method for PDL based on graph tableaux is due to Pratt [161]. The development in this chapter up to § 4.5 can be seen as a reformulation of Pratt's approach and the corresponding correctness arguments in our framework.
► Incremental graph procedures for description logics without eventualities are presented by Goré, Nguyen and Widmann [91, 94, 200]. These procedures do not distinguish between eager and cautious pruning. Instead, they have a single marking procedure that propagates both evidence and refutability. This is possible since in the absence of eventualities, $\mathcal{E}^{\mathcal{T}}$ can be computed in much the same way as $\mathcal{R}^{\mathcal{T}}$ (see Remark 4.5.6).
► Building on their work on logics without eventualities, Goré and Widmann [93, 200] devise the first incremental graph tableau procedure for PDL, on which our procedure is based. In contrast to our design, Goré and Widmann still make no explicit distinction between eager and cautious pruning. Instead, they approach eventualities via the notion of *potential rescuers*. A potential rescuer for a claim γ indicates the existence of a

plan for γ that does not involve a clause in $\mathcal{R}^{\mathcal{T}}$. The existence of a run for a claim is represented by a special potential rescuer $\perp$. Consequently, a clause can be marked as refuted (or *closed* [200]) if and only if it has no potential rescuer. While the notion of potential rescuers can be used to define eager pruning as well as cautious pruning, the presentation of the procedure in [93, 200] is primarily concerned with cautious pruning, elaborating it in great detail. Eager pruning is only briefly sketched as an optimization. Without eager pruning (i.e., without an explicit set $\mathcal{E}^{\mathcal{T}}$), the presented procedure is only incremental on unsatisfiable inputs. On satisfiable formulas, it can only terminate after building the complete tableau. This is due to the fact that in the absence of $\mathcal{E}^{\mathcal{T}}$, we can only give complete tableaux as positive certificates since there $\mathcal{E}^{\mathcal{T}}$ is implicitly given as the complement of $\mathcal{R}^{\mathcal{T}}$. Goré and Widmann's implementation of their procedure, however, performs both cautious and eager pruning.
► Goré and Widmann [95, 200] have extended their method to PDL with converse. The extension to converse is not straightforward and will require a significant update of our theory, which is future work.

5 Tree Search for Hybrid Logic with Eventualities

Our next goal is to develop incremental decision procedures for modal logics with both eventualities and nominals. Our main focus for the chapter will be on H^* as the simplest such logic.

Existing incremental procedures for hybrid logic [115, 30, 29, 117, 127, 40, 107] are all based on tree search and employ a prefixed representation of candidate models. While none of these procedures has optimal worst-case complexity, due to their incrementality, they often display good performance on practical problems.

As we saw in § 3.5.1, an essential complication with the graph tableau approach as it comes to nominals is giving a pruning relation that, on the one hand, removes enough clauses to achieve nominal admissibility but, on the other hand, does not exclude any potential demos. In fact, we saw that in the presence of nominals, no pruning relation can satisfy both of these correctness properties while remaining confluent. The solution taken in § 3.5, i.e., to introduce an additional guessing stage, is not compatible with the idea of having an incremental procedure with good practical performance.

For this reason, we turn our attention to tree procedures. Unlike existing procedures for hybrid logic, we want to retain a clausal representation of candidate models similar to that in Chapter 4 since this allows for considerably simpler termination and complexity arguments than those for prefixed approaches (compare, for instance, our termination arguments in Chapters 3 and 4 to those in [29, 117, 127]).

But how do we combine tree search with a clausal representation? As mentioned in § 1.2.5, tree procedures organize the search space for demos as a tree where every branch corresponds to a candidate model. Because of the close correspondence between branches of the search tree and candidate models, in the following we often say "branch" when referring to the candidate model represented by a branch. Tree procedures typically search for a demo by exploring the search tree branch by branch, backtracking as soon as they can determine that the current branch cannot be extended to a demo. Given a branch containing a disjunction $\sigma : s \vee t$, a prefixed tree procedure will typically explore two alternative ways of extending the branch, namely by the formula $\sigma : s$ or by $\sigma : t$. The two extensions form two different branches that are further explored in isolation from each other.

A first idea for obtaining clausal tree procedures may be to combine branching on disjunctive formulas as done by prefixed tree procedures with the clausal representation of candidate models used in Chapter 4. The search would then branch on B-clauses (admitting only one departing link per B-clause and branch) and backtrack as soon as a branch becomes *closed*, i.e., as soon as it becomes obvious that the branch cannot be

$$s = \Box^+(\Diamond^* p \wedge \Diamond^* \neg p)$$

$$\begin{tikzcd}
 & & & \Diamond^* p,\ \Diamond^* \neg p,\ s \arrow[dll, "1"'] \arrow[drr, "5"] & & & \\
 & p,\ \Diamond^* \neg p,\ s \arrow[dl, "3"'] \arrow[dr, "2"] & & & & \Diamond^+ p,\ \Diamond^* \neg p,\ s \arrow[dl, "6"'] \arrow[dr, "8"] & \\
p,\ \Diamond^+ \neg p,\ s \arrow[uurrr, bend left=50, "4"] & & p,\ \neg p,\ s & & \Diamond^+ p,\ \neg p,\ s \arrow[uul, "7"'] & & \Diamond^+ p,\ \Diamond^+ \neg p,\ s \arrow[uulll, bend right=50, "10"'] \arrow[uulll, bend left=80, "9"]
\end{tikzcd}$$

Figure 5.1: A complete tableau

extended to a demo. This may be the case either because the branch contains a C-clause or a *loop*, which is a cyclic path resulting from a failed attempt to construct a run. Since a branch has at most one departing link per B-clause, and hence there is at most one plan per claim, a branch that has a loop cannot have a run for any claim that is part of the loop (and hence cannot be evident in the sense of Chapter 4).

For instance, applied to the formula $\Diamond^* p \wedge \Box^* \neg p$, tree search starts with the branch consisting of the single clause $\{\Diamond^* p, \neg p, \Box^* \neg p\}$, which it then extends to two branches:

$$\begin{tikzcd}
\Diamond^* p,\ \neg p,\ \Box^+ \neg p \arrow[d] & & \Diamond^* p,\ \neg p,\ \Box^+ \neg p \arrow[d] \\
p,\ \neg p,\ \Box^+ \neg p & \text{and} & \Diamond^+ p,\ \neg p,\ \Box^+ \neg p \arrow[u, bend left=60]
\end{tikzcd}$$

Both branches are closed. The first contains a C-clause and the second contains a loop. Since there are no more branches to explore, tree search returns unsatisfiable. Note that it is essential that the second branch is closed since it does not represent a model of $\Diamond^* p \wedge \Box^* \neg p$.

It can be shown that without eventualities, the combination of tree search and the clausal representation from Chapter 4 can be used as a basis for incremental decision procedures. With eventualities, however, tree search on this clausal representation is incomplete in that it may fail to find demos for satisfiable formulas. Consider the complete tableau in Figure 5.1 for illustration. Tree search unrolls this tableau into four branches containing the links {1, 2}, {1, 3, 4}, {5, 6, 7}, and {5, 8, 9, 10}, respectively. The branch {1, 2} is closed since it contains a C-clause, and the other branches are closed since they contain loops, leaving eventualities unfulfilled. Thus, tree search is not able to find a demo for the formula $\Box^*(\Diamond^* p \wedge \Diamond^* \neg p)$. However, the formula is satisfiable (in fact, the tableau becomes evident if we delete the clause $\{p,\ \neg p,\ s\}$).

To solve this problem, we switch to a more modular representation. We replace B-clauses and C-clauses, used up to now to represent propositional reasoning steps, by an abstract DNF computation. Abstracting away from propositional reasoning makes it possible to disallow loops while preserving the correctness of the procedure.

To extend our approach to nominals, we use a technique that we call *nominal propagation*. When we add a new clause to the branch, we add to the new clause all formulas that occur in clauses of the branch that have a nominal in common with the new clause. Unlike previous approaches to nominals, nominal propagation is static since clauses and links that are already part of the branch remain unchanged. This property of nominal propagation will be important for the crucial part of the correctness proof for our approach, which shows that loops can be safely ignored.

Structure of the chapter. First, we present a general view of tree procedures (§ 5.1) and an abstract notion of a propositional DNF (§ 5.2) that underlie our method. To introduce the basic ideas behind our approach to tree procedures for modal logic, we begin with a procedure for K (§ 5.3). In the next step, we extend the procedure to deal with eventualities (§ 5.4), after which we show how to handle nominals (§ 5.5). Afterwards, we discuss an extension of the procedure to satisfaction operators, present a simple optimization, and relate the approach to graph tableaux (§ 5.6). The chapter concludes with a discussion of related work (§ 5.7).

5.1 Tree Search

Before we come to the details of our approach, we present our view of tree procedures and introduce some vocabulary for talking about tree procedures.

Central for our presentation of tree procedures is the notion of a *branch*. As already mentioned at the beginning of this chapter, every branch of a search tree explored by a tree procedure corresponds to a candidate model. Our decision procedures will work on branches defined similarly to graph tableaux in Chapter 4 as sets of clauses and links. We will extend the notion of satisfaction to branches, distinguishing between satisfiable and unsatisfiable branches.

A tree procedure is commonly presented by giving **expansion rules** that describe the search space that needs to be explored by the procedure. In our notation, an expansion rule is written as

$$\frac{\Gamma}{\Gamma_1 \mid \cdots \mid \Gamma_n}$$

and specifies how a branch Γ can be extended to zero or more alternative branches $\Gamma_1, \ldots, \Gamma_n$. We require that $\Gamma \subsetneq \Gamma_i$ for all $i \in [1, n]$. Rules that produce no extension (i.e., rules where $n = 0$) are also written as

$$\frac{\Gamma}{\otimes}$$

A branch to which we can apply a rule that produces no extension is called **closed.** Branches that are not closed are called **open.** Branches to which no rule is applicable are called **maximal.**

Consider, for instance, the following rules for propositional logic.

$$\frac{A}{A;s \mid A;t}\ s\vee t\in A \qquad\qquad \frac{A}{A;s;t}\ s\wedge t\in A \qquad\qquad \frac{A}{\otimes}\ \{p,\neg p\}\subseteq A$$

The rules operate on branches that are sets of propositional formulas in negation normal form (which is reflected in the formulation of the rules by the use of A in place of Γ). The first rule states that a branch A containing a disjunction $s \vee t$ can be extended either by adding s or by adding t. Because of the requirement that every extension of a branch has to be a proper superset of the branch, the rule does not apply to branches A that already contain either s or t. The second rule states that a branch containing a conjunction $s \wedge t$ can be extended by adding s and t. The rule does not apply to branches that already contain both s and t. The third rule states that every branch containing a pair of complementary literals p and $\neg p$ is closed.

An incremental tree procedure searches for a maximal branch containing the initial formula. Given an initial branch, the procedure expands the branch using expansion rules. The rule and the formula used to expand a given branch are chosen among all applicable rules and all formulas of the branch in a "don't care" fashion (see [133], § 5), meaning that the choice does not matter for the correctness of the procedure. Whenever the branch is expanded by a rule that produces more than one extension, the procedure has to choose one of the extended branches to work on. This decision is "don't know" (see [133], § 5), meaning that if the procedure fails to complete the chosen branch to a maximal branch, it has to backtrack and try another extension. Whenever the current branch becomes closed, the procedure backtracks to the last choice point, or returns unsatisfiable if there are no more alternatives to explore (of course, instead of always backtracking to the last choice point, the procedure may employ a more intelligent backtracking strategy like backjumping; see [77, 165, 55, 119, 190]). Once the procedure arrives at a maximal branch, it halts and returns satisfiable.

For instance, consider the formula $(p \vee q) \wedge ((p \vee r) \wedge (\neg p \wedge r))$. A possible run of the procedure on the formula (using the above rules) can be visualized as follows.

$$\begin{array}{ll|ll|ll}
\multicolumn{6}{l}{0.\quad (p \vee q) \wedge ((p \vee r) \wedge (\neg p \wedge r))}\\
\multicolumn{6}{l}{1.\quad p \vee q,\ (p \vee r) \wedge (\neg p \wedge r)}\\
\multicolumn{6}{l}{2.\quad p \vee r,\ \neg p \wedge r}\\ \hline
3. & p & 3. & \multicolumn{3}{l}{q}\\ \cline{3-6}
4. & \neg p,\ r & 5. & p & 5. & r\\
 & \otimes & & & 6. & \neg p
\end{array}$$

The numbers indicate steps of the procedure, 0 being the initial state. In step 1, the procedure applies the rule for conjunctions to the initial formula, extending the branch by $p \vee q$ and $(p \vee r) \wedge (\neg p \wedge r)$. Now, the procedure has two possibilities to extend the branch further, namely either by applying the rule for conjunctions to $p \vee q$ or the rule for disjunctions to $(p \vee r) \wedge (\neg p \wedge r)$. It chooses "don't care" the second option, which yields $p \vee r$ and $\neg p \wedge r$. For step 3, the procedure chooses to apply the rule for disjunctions to $p \vee q$, which yields to two possible extensions, by p and q, respectively. The procedure chooses "don't know" to explore the extension by p. On this extension, the rule for disjunctions does not apply to $p \vee r$ since the branch already contains p. Therefore, the only possible extension of the branch is obtained by applying the rule for conjunctions to $\neg p \wedge r$. The resulting branch is closed since it contains p and $\neg p$. Therefore, the procedure backtracks to the last "don't know" choice point and explores the second possible extension of the branch constructed in step 3. The procedure chooses "don't care" to expand this extension using the rule for disjunctions applied to $p \vee r$ (step 5). Then it chooses "don't know" to explore the resulting extension by r, which it expands using the rule for conjunctions applied to $\neg p \wedge r$ (step 6). Since the chosen branch already contains r, the only formula added to the branch in step 6 is $\neg p$. Since the resulting branch is maximal, the procedure returns satisfiable.

Note that one of the two extensions produced in step 5 remains unexplored. Depending on the "don't know" choices made by the procedure, it may be able to find a maximal branch without going through any closed branches. This is essential for the practical efficiency of the procedure. Only in the worst case will it have to explore the entire search space. With intelligent heuristics guiding the "don't know" choices, tree search can be efficient even when the search space is too large to be traversed completely.

Remark 5.1.1 While our presentation only uses the tree metaphor informally to visualize the search space, traditional presentations of the tableau method (see, for instance, [25, 184, 71]) often introduce trees as depicted above formally, calling them *tableaux*. This view is helpful in the study of proof systems based on the tableau method, but not necessary in our case. ◂

Since every branch contains the initial formula, one can show that a tree procedure as described above is correct if it terminates and the expansion rules underlying the procedure satisfy the following two correctness properties:

- **Demonstration-soundness**: Every maximal branch is satisfiable.
- **Refutation-soundness**: For every rule $\frac{\Gamma}{\Gamma_1 | \cdots | \Gamma_n}$ we have that Γ is satisfiable if and only if at least one of the branches $\Gamma_1, \ldots, \Gamma_n$ is satisfiable.

Note that, in particular, refutation-soundness implies that every closed branch is unsatisfiable.

If the procedure halts because it has found a maximal branch, the satisfiability of the initial formula (which is part of the branch) is immediate by demonstration-soundness. If the procedure halts because all branches of the search tree are closed, the unsatisfiability of the initial formula follows with refutation-soundness by induction on the structure of the search tree explored by the procedure. Notably, the proof does not depend on the "don't care" choices made by the procedure when constructing the search tree.

Remark 5.1.2 In the literature, in place of demonstration-soundness one often uses a related property called *refutational completeness* or simply *completeness* (refutation-soundness is then simply called *soundness*). A refutationally complete calculus can refute every unsatisfiable formula. This notion originates in a view of the tableau method as a refutation proof system, which we do not elaborate on in this thesis (see also Remark 5.1.1). For terminating tableau systems, refutational completeness coincides with demonstration-soundness.

Also, demonstration-soundness is commonly called *model existence* (for instance, see [69]). ◂

To prove that a tree procedure terminates, it suffices to show that the search space it explores is finite. Since expansion rules are finitely branching, with König's lemma it follows that a tree procedure terminates if no branch can be extended infinitely often. In our case, this will follow from the requirement that all extensions of a branch Γ are proper supersets of Γ since the cardinality of branches will be bounded from above (branches will only contain syntactic material from a finite formula universe).

Remark 5.1.3 In general, detecting whether a given branch Γ is closed may be expensive since it involves checking whether there is a way to apply some expansion rule to Γ such that the rule produces no extension. Therefore, rather than trying to detect closed branches as early as possible, a practical implementation may continue extending a closed branch until it encounters a closing rule application.

Also, note that while the "don't care" decisions do not influence the correctness of a tree procedure, they may have a big influence on its performance. This is the case since the decisions may change the search space explored by the procedure. For instance, consider again the run depicted on p. 76. If in step 3 the procedure chooses to apply the rule for conjunctions to $\neg p \wedge r$ instead of the rule for disjunctions, the run simplifies as follows.

$$\begin{array}{ll|ll}
0. & \multicolumn{3}{l}{(p \vee q) \wedge (p \vee r) \wedge (\neg p \wedge r)} \\
1. & \multicolumn{3}{l}{\quad p \vee q,\ (p \vee r) \wedge (\neg p \wedge r)} \\
2. & \multicolumn{3}{c}{p \vee r,\ \neg p \wedge r} \\
3. & \multicolumn{3}{c}{\neg p,\ r} \\
\hline
4. & \quad p & 4. & \quad q \\
 & \quad \otimes & &
\end{array}$$

In general, devising efficient rule and formula selection strategies is a nontrivial task (see [188, 190, 97, 98]). ◂

5.2 Disjunctive Normal Forms

By the semantic equivalences for alpha and beta formulas, every nonliteral formula of H* is equivalent to a formula in *disjunctive normal form (DNF)*, i.e., to a disjunction of conjunctions of literals. For instance, the formula $s = \Diamond^*(p \vee q) \wedge \neg r$ is equivalent to the disjunction $t = (p \wedge \neg r) \vee (q \wedge \neg r) \vee (\Diamond^+(p \vee q) \wedge \neg r)$. Hence, the formula t is commonly called a *DNF of* s. A formula may have more than one DNF. So, besides t, s has the DNF $(p \wedge \neg r) \vee (\neg p \wedge q \wedge \neg r) \vee (\Diamond^+(p \vee q) \wedge \neg r)$. Since in our framework conjunctions of literals are naturally represented by N-clauses, a DNF of a formula can be represented as a set $\mathcal{D}$ of N-clauses, where every clause in $\mathcal{D}$ corresponds to a disjunct of the DNF. For instance, the formula t can be represented as the set $\{\{p, \neg r\}, \{q, \neg r\}, \{\Diamond^+(p \vee q), \neg r\}\}$.

Formally, we proceed as follows. As before, we consider only formulas in negation normal form. We fix a formula universe $\mathcal{F}$ and consider only formulas from $\mathcal{F}$. Let A be a nonempty set of formulas. A set $\mathcal{D}$ of N-clauses is a **DNF of** A if $\mathcal{D}$ satisfies the following conditions:

1. $\mathfrak{M}, w \vDash A \iff \exists D \in \mathcal{D}\colon\ \mathfrak{M}, w \vDash D$.
2. $C \rhd A \iff \exists D \in \mathcal{D}\colon\ D \subseteq C$.

Note that condition (1) implies that every set A that has an empty DNF is unsatisfiable.

Example 5.2.1

- $\{\{\Diamond p\}\}$ is a DNF of $\{\Diamond p\}$.
- $\emptyset$ is a DNF of $\{p, \neg p\}$.
- $\{\{p\},\ \{q\}\}$ is a DNF of $\{p \vee q\}$.
- $\{\{\Diamond p,\ p\},\ \{\Diamond p,\ q\}\}$ is a DNF of $\{\Diamond p,\ p \vee q\}$. ◂

There are many ways to compute a DNF. We now present a simple procedure for computing DNFs of sets of formulas of H*, which is a variant of the tree procedure for propositional logic discussed in § 5.1. The procedure is based on the following generalization of the expansion rules in § 5.1.

$$\frac{A}{A\,;\beta_1 \mid A\,;\beta_2}\ \beta \in A \qquad\qquad \frac{A}{A\,;\alpha_1\,;\alpha_2}\ \alpha \in A \qquad\qquad \frac{A}{\otimes}\ \{p,\ \neg p\} \subseteq A$$

Note that each rule directly corresponds to a line in the definition of Hintikka sets in § 3.1. Therefore, every branch that is maximal with respect to the expansion rules is a Hintikka set.

As before, the procedure expands the initial branch using the rules in a depth-first manner. Unlike the procedure in § 5.1, instead of searching for *some* maximal branch, the present procedure always explores the entire search space, returning *all* maximal branches it encounters.

The procedure is terminating. To see this, it suffices to convince ourselves that no branch can be extended infinitely often. This is the case since, as always, the expansion rules are restricted to produce only extensions of a branch A that are proper supersets of A, while the cardinality of all branches is bounded from above by $|\mathcal{F}|$.

We call the set of maximal branches returned by a run of the procedure on a set A a **Hintikka decomposition of** A. The literals of each Hintikka set in a Hintikka decomposition of A form an N-clause. All N-clauses obtained this way from a Hintikka decomposition of A constitute a DNF of A.

Take, for instance, the set $A = \{\Diamond^* p,\ \Box^*\Diamond^+ p\}$. Depending on whether the rule for alpha formulas or the rule for beta formulas is applied first, we obtain two possible runs of the procedure.

0.	$\Diamond^* p,\ \Box^*\Diamond^+ p$
1.	$\Diamond^+ p,\ \Box^+\Diamond^+ p$

0.	$\Diamond^* p,\ \Box^*\Diamond^+ p$		
1.	p	1.	$\Diamond^+ p$
2.	$\Diamond^+ p,\ \Box^+\Diamond^+ p$	3.	$\Box^+\Diamond^+ p$

Note that the first run ends immediately after applying the rule for alpha formulas to $\Box^*\Diamond^+p$ since the resulting set contains $\Diamond^+p$, and so the rule for beta formulas does not apply to $\Diamond^*p$. The two runs yield the Hintikka decompositions $\{\{\Diamond^*p,\ \Box^*\Diamond^+p,\ \Diamond^+p,\ \Diamond^+\Box^+p\}\}$ and $\{\{\Diamond^*p,\ \Box^*\Diamond^+p,\ p,\ \Diamond^+p,\ \Box^+\Diamond^+p\},\ \{\Diamond^*p,\ \Box^*\Diamond^+p,\ \Diamond^+p,\ \Diamond^+\Box^+p\}\}$, respectively. From the two decompositions, we obtain the DNFs $\{\{\Diamond^+p,\ \Box^+\Diamond^+p\}\}$ and $\{\{p,\ \Diamond^+p,\ \Box^+\Diamond^+p\},\ \{\Diamond^+p,\ \Box^+\Diamond^+p\}\}$.

We now show that the set of N-clauses obtained from a Hintikka decomposition of A satisfies the two conditions for a DNF of A. First, observe that a Hintikka decomposition $\mathcal{D}$ of a formula set A satisfies the following equivalence:

$$\mathfrak{M}, w \models A \iff \exists H \in \mathcal{D}\colon\ \mathfrak{M}, w \models H$$

The direction from right to left follows trivially by inclusion. The other direction follows from the refutation-soundness of the expansion rules, which in turn follows from the semantics of alpha and beta formulas. Therefore, for a set of N-clauses obtained from a Hintikka decomposition to satisfy condition (1) for DNFs it suffices if we have:

$$\mathfrak{M}, w \models H \iff \mathfrak{M}, w \models \Lambda H$$

The direction from left to right is immediate since $\Lambda H \subseteq H$. The other direction follows with the claim

$$H \text{ Hintikka set and } s \in H \text{ and } \mathfrak{M}, w \models \Lambda H \implies \mathfrak{M}, w \models s$$

which follows by induction on the decomposition order of s with the semantic equivalences for alpha and beta formulas.

Condition (2) for DNFs is shown as follows. The direction from right to left follows since $\Lambda H \rhd H$ (Proposition 4.1.7) and since $A \subseteq H$ for every set H in a Hintikka decomposition of A.

The direction from left to right follows from the implication

$$C \rhd B \implies C \rhd \Lambda B \iff \Lambda B \subseteq C$$

which holds for arbitrary formula sets B, and the claim

$$\mathcal{D} \text{ Hintikka decomposition of } A \text{ and } C \rhd A \implies \exists H \in \mathcal{D}\colon\ C \rhd H$$

This last claim follows from the following observations (we leave out the details of the proof):

1. The computation of Hintikka decompositions terminates for every A.
2. If $C \rhd B$ and $\alpha \in B$ (or $\beta \in B$), then $C \rhd B\,;\alpha_1\,;\alpha_2$ (or $C \rhd B\,;\beta_i$ for some $i \in \{1,2\}$).
3. If $C \rhd B$, then B contains no complementary literals.

Note that since the expansion rules for computing Hintikka decompositions are at most binary (i.e., produce at most two alternative extensions) and properly increase the cardinality of branches (which is bounded by $|\mathcal{F}|$), the size of a Hintikka decomposition is at most exponential in $|\mathcal{F}|$. Hence, the size of a DNF is at most exponential in $|\mathcal{F}|$ and can be computed in exponential time with respect to $|\mathcal{F}|$.

Remark 5.2.2 In practice, one is typically interested in obtaining small DNFs to reduce the search space for the decision procedure. To reduce the size of a DNF computed from a Hintikka decomposition, one could prune it by removing clauses that are nonminimal with respect to inclusion. For instance, consider the set $A = \{\Diamond^* p, \Diamond^+ p \vee q\}$. Depending on the order in which we apply the expansion rules, we obtain two possible DNFs: $\mathcal{D}_1 = \{\{\Diamond^+ p\}, \{p, q\}, \{\Diamond^+ p, p\}\}$ and $\mathcal{D}_2 = \{\{\Diamond^+ p\}, \{p, q\}, \{\Diamond^+ p, q\}\}$. If we prune nonminimal clauses, both DNFs converge to $\mathcal{D}_3 = \{\{\Diamond^+ p\}, \{p, q\}\}$. It is easily seen that $\mathcal{D}_3$ is still a DNF of A. However, to compute $\mathcal{D}_3$, it seems necessary to compute all of the clauses in $\mathcal{D}_1$ or $\mathcal{D}_2$ first. This does not seem practical since DNFs may become very large and the computation of a complete DNF very expensive. Practical implementations are more likely to compute individual clauses of a DNF incrementally, in a demand-driven fashion.

Also, a practical procedure for computing DNFs may choose to replace the rule for beta formulas by an asymmetric version:

$$\frac{A}{A\,;\beta_1 \mid A\,;\sim\!\beta_1\,;\beta_2}\ \beta \in A$$

This so-called *semantic branching* can considerably speed up propositional reasoning, thus improving the overall performance of tableau procedures [46, 113]. With semantic branching, $\mathcal{F}$ needs to be additionally closed under normalizing negation to make sure that it still contains all conclusions of the modified rule. Other than for this adaptation, semantic branching is fully compatible with our abstract definition of DNFs. ◂

Finally, let us state two important properties of DNFs.

Proposition 5.2.3 Let $\mathcal{D}$ be a DNF of A and $C \in \mathcal{D}$. Then $C \rhd A$.

Proof If $C \in \mathcal{D}$, there trivially exists some $D \in \mathcal{D}$ such that $D \subseteq C$. The claim follows by condition (2) for DNFs. ∎

Proposition 5.2.4 If $\emptyset$ is a DNF of A, then A has no other DNF.

Proof Let A be a nonempty set of formulas and $\emptyset$, $\mathcal{D}$ be two DNFs of A. It suffices to show that $\mathcal{D} = \emptyset$. Since $\emptyset$ is a DNF of A, by condition (2) for DNFs, A is not supported by any N-clause. The claim follows by Proposition 5.2.3. ∎

Remark 5.2.5 While we are currently only interested in DNFs for H*, the definition of DNFs and their computation via Hintikka decompositions adapts to every logic whose formulas can be classified into literals, alpha and beta formulas (also in cases where nonliteral formulas decompose into more than two constituents), provided the induced decomposition relation is terminating. We need this since we exploit the decomposition order to prove the claims

$$H \text{ Hintikka set and } s \in H \text{ and } \mathfrak{M}, w \models \bigwedge H \implies \mathfrak{M}, w \models s$$

and

$$H \text{ Hintikka set } \implies \Lambda H \rhd H$$

which are essential for showing that the N-clauses of Hintikka decompositions satisfy conditions (1) and (2) for DNFs, respectively. A more detailed discussion of why termination of the decomposition relation is necessary will be given in § 6.1. ◂

5.3 Procedure for K

We start with a tree procedure for K to demonstrate the basic ideas of our approach. Recall that we assume the following grammar for negation normal formulas of K.

$$\begin{aligned} s ::= {} & p \mid s \vee s \mid \Diamond s \\ & \mid \neg p \mid s \wedge s \mid \Box s \end{aligned}$$

5.3.1 Branches and Evidence

We begin by defining branches as used by the procedure. While branches resemble graph tableaux as defined in Chapter 4, they only employ N-clauses and satisfy an additional functionality condition. Similarly to graph tableaux, branches represent candidate models. To distinguish branches that describe demos, we introduce the notion of evidence.

As before, links are triples of the form $C\xi D$. For K, we allow only one type of links, namely links of the form $C\binom{\Diamond s}{s}D$ where C, D are N-clauses, $\Diamond s \in C$, and $D \in \mathcal{D}$ for some DNF $\mathcal{D}$ of $\mathfrak{R}C\,;s$. We call such links **simple**.

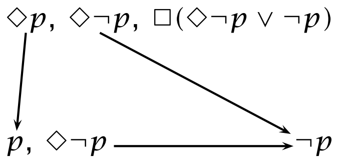
A **branch** is a nonempty set Γ of N-clauses and simple links that satisfies the following conditions:

- **Link coherence**: If $C\xi D \in \Gamma$, then $\{C, D\} \subseteq \Gamma$.
- **Functionality**: If $C\binom{\Diamond s}{s}D \in \Gamma$ and $C\binom{\Diamond s}{s}E \in \Gamma$, then $D = E$.

Since in the following we will never have to look at B-clauses or C-clauses, for the rest of the chapter we will use the word “clause” as a shorthand for “N-clause”.

For convenience, we extend the notation $s \mid C$ from claims to arbitrary formula-clause pairs (s, C) such that $C \rhd s$. A branch Γ **realizes** $\Diamond s \mid C$ (or **realizes** $\Diamond s$ **in** C) if $C\binom{\Diamond s}{s}D \in \Gamma$ for some $D \in \Gamma$. A branch Γ is **evident** if Γ realizes $\Diamond s \mid C$ for every $\Diamond s \in C \in \Gamma$. A model **satisfies a branch** Γ (or **is a model of** Γ) if it satisfies all clauses of Γ.

Example 5.3.1 Below we show a graphical representation of a branch consisting of the clauses $C_1 = \{\Diamond p, \Diamond\neg p, \Box(\Diamond\neg p \vee \neg p)\}$, $C_2 = \{p, \Diamond\neg p\}$ and $C_3 = \{\neg p\}$, as well as the links $C_1\binom{\Diamond p}{p}C_2$, $C_1\binom{\Diamond\neg p}{\neg p}C_3$ and $C_2\binom{\Diamond\neg p}{\neg p}C_3$.


$$\begin{tikzcd} \Diamond p,\ \Diamond\neg p,\ \Box(\Diamond\neg p \vee \neg p) \arrow[d] \arrow[dr] & \\ p,\ \Diamond\neg p \arrow[r] & \neg p \end{tikzcd}$$

As usual, links are represented with arrows. The branch is evident since it contains an outgoing link for every diamond formula. ◂

Let us adopt the notion of demos from Chapter 4 (restricted to K). Similarly to evident graph tableaux in Chapter 4, every evident branch describes a demo that supports all of the formulas in the branch. Hence, every evident branch is satisfiable. In the present case, the correspondence of evident branches to demos is particularly close since branches contain only N-clauses.

Theorem 5.3.2 The clauses of an evident branch form a demo.

Proof Let Γ be an evident branch. In the case of K, to show $\Diamond$-admissibility, it suffices to show that for every $\Diamond s \in C \in \Gamma$, Γ contains a clause D such that $D \rhd \mathcal{R}C\,;s$. This follows by Proposition 5.2.3 from the fact that Γ realizes $\Diamond s \mid C$, i.e., contains a link $C\binom{\Diamond s}{s}D \in \Gamma$ since $D \in \mathcal{D}$ for some DNF $\mathcal{D}$ of $\mathcal{R}C\,;s$. ∎

5.3.2 Expansion Rules

The tree procedure is obtained as discussed in § 5.1 from a single expansion rule:

$$\frac{\Gamma}{\Gamma\,;D_1\,;C\binom{\Diamond s}{s}D_1 \mid \cdots \mid \Gamma\,;D_n\,;C\binom{\Diamond s}{s}D_n} \quad \begin{array}{l}\Diamond s \in C \in \Gamma,\\ \Gamma \text{ does not realize } \Diamond s \mid C,\\ \{D_1,\dots,D_n\} \text{ DNF of } \mathcal{R}C\,;s\end{array}$$

We call it the **diamond rule**. Note that the the selection of $\Diamond s \in C \in \Gamma$ as well as the choice of a DNF of $\mathcal{R}C\,;s$ to use with the rule is "don't care". The restriction that Γ does not realize $\Diamond s \mid C$ is needed to ensure that the rule is only applicable once per clause and diamond formula, which is necessary to preserve the functionality of branches. Without the restriction, the rule may apply to the same formula $\Diamond s \in C \in \Gamma$ more than once since $\mathcal{R}C\,;s$ may have more than one pairwise disjoint DNF.

Note that according to the diamond rule, a branch is closed if and only if it contains a clause C and a diamond $\Diamond s \in C$ such that $\mathcal{R}C\,;s$ has an empty DNF. By condition (1) for DNFs, this is the case if and only if $\mathcal{R}C\,;s$ is unsatisfiable. Hence, closed branches are unsatisfiable.

Example 5.3.3 Consider the clause $C = \{\Diamond p,\ \Box\neg p\}$. Since $\Diamond p \mid C$ is not realized in $\Gamma = \{C\}$, the diamond rule applies to Γ. Since there is no clause supporting $\mathcal{R}C\,;p = \{p,\ \neg p\}$, every, DNF of $\mathcal{R}C\,;p$ is empty (Propositions 5.2.3, 5.2.4). Consequently, the diamond rule applied to $\Diamond p \in C \in \Gamma$ produces no extension, meaning that Γ is closed. ◂

Moreover, note that the diamond rule applies to a branch if and only if the branch is not evident. Together with Theorem 5.3.2, this implies that the rule is demonstration-sound.

Example 5.3.4 Consider the following run of the procedure.

0.	$C_1 = \{\Diamond\Diamond p,\ \Box(q \vee \Box\neg p)\}$		
1.	$C_2 = \{\Diamond p, q\},\ C_1\binom{\Diamond\Diamond p}{\Diamond p}C_2$	1.	$C_3 = \{\Diamond p, \Box\neg p\},\ C_1\binom{\Diamond\Diamond p}{\Diamond p}C_3$
2.	$C_4 = \{p\},\ C_2\binom{\Diamond p}{p}C_4$		$\otimes$

The initial branch is $\{C_1\}$. Note that $\{C_2, C_3\}$ is a DNF of $\mathfrak{R}C_1\,;\Diamond p$. Application of the diamond rule to $\Diamond\Diamond p \mid C_1$ with this DNF yields the branches $\Gamma_1 = \{C_1, C_1\binom{\Diamond\Diamond p}{\Diamond p}C_2, C_2\}$ and $\Gamma_2 = \{C_1, C_1\binom{\Diamond\Diamond p}{\Diamond p}C_3, C_3\}$. The branch Γ_2 is closed because of clause C_3 (see Example 5.3.3). Expansion of $\Diamond p \in C_2 \in \Gamma_1$ with the set $\{\{p\}\}$ as a DNF of $\mathfrak{R}C_2\,;p$ yields the evident branch $\{C_1, C_1\binom{\Diamond\Diamond p}{\Diamond p}C_2, C_2, C_2\binom{\Diamond p}{p}C_4, C_4\}$. ◂

Theorem 5.3.5 (Termination) The tree procedure for K induced by the diamond rule is terminating.

Proof The claim follows by König's lemma from the fact that a branch cannot be extended infinitely often. This, in turn, follows since every application of the diamond rule strictly enlarges the branch, while the cardinality of branches (i.e., the number of clauses and links that may occur in a branch) is exponentially bounded in $|\mathcal{F}|$. ∎

For refutation-soundness, we show that every non-evident, satisfiable branch has a satisfiable extension. The other direction follows trivially by inclusion.

Theorem 5.3.6 (Refutation-Soundness) Let Γ be a branch that does not realize $\Diamond s \mid C$ for some $\Diamond s \in C \in \Gamma$. Let $\mathfrak{M}$ be a model of Γ and let $\mathcal{D}$ be a DNF of $\mathfrak{R}C\,;s$. Then there is a clause $D \in \mathcal{D}$ such that $\Gamma\,;D\,;C\binom{\Diamond s}{s}D$ is a branch satisfied by $\mathfrak{M}$.

Proof Let Γ, $\Diamond s \in C \in \Gamma$, $\mathfrak{M}$ and $\mathcal{D}$ be as required. Since $\mathfrak{M}$ satisfies C, it also satisfies $\mathfrak{R}C\,;s$. By condition (1) for DNFs, there is some $D \in \mathcal{D}$ such that $\mathfrak{M}$ satisfies D. Let $\Delta = \Gamma\,;D\,;C\binom{\Diamond s}{s}D$. Since Γ does not realize $\Diamond s \mid C$, Δ is functional and hence a branch. The model $\mathfrak{M}$ satisfies Δ since $\mathfrak{M}$ satisfies Γ and D. ∎

Since the diamond rule is demonstration-sound and refutation-sound, the procedure decides the satisfiability of branches. The procedure can be used to decide the satisfiability of (not necessarily literal) formulas by exploiting that a given formula s is satisfiable if and only if so is $\Diamond s$. Hence, to decide s, it suffices to decide the branch $\{\{\Diamond s\}\}$.

Alternatively, one can exploit that, given a formula s and a DNF $\mathcal{D}$ of s, s is satisfiable if and only if there is some $C \in \mathcal{D}$ such that the branch $\{C\}$ is satisfiable.

5.4 Eventualities

Next, we consider the logic K*, which extends K with eventualities. To account for eventualities, we extend our syntactic machinery by two additional types of links. This will allow us to verify the satisfaction of eventualities in a branch via a simple cycle check. To prove the refutation-soundness of the expansion rules, in particular of the rule performing the cycle checks, we will give the new links a semantic interpretation.

$$\begin{tikzcd}
\Diamond^+ p,\ \Box\neg p \arrow[d] \\
\Diamond^+ p,\ \neg p \arrow[d] \\
p
\end{tikzcd}
\qquad
\begin{tikzcd}
\Diamond^+ p,\ \Box\neg p \arrow[d] \\
\Diamond^+ p,\ \neg p \arrow[d] \\
\Diamond^+ p \arrow[d] \\
p
\end{tikzcd}
\qquad
\begin{tikzcd}
\Diamond^+ p,\ \Box^+\neg p \arrow[d] \\
\Diamond^+ p,\ \Box^+\neg p,\ \neg p \arrow[loop below]
\end{tikzcd}$$

Figure 5.2: Three branches

5.4.1 Branches and Evidence

As one would expect, realization of eventualities $\Diamond^+ s$ is more involved that realization of simple diamond formulas.

Example 5.4.1 Suppose we extend our definitions of branches and evidence for K to K* in the intuitive way. Then $\Gamma = \{C, C\binom{\Diamond^+ p}{\Diamond^* p}C\}$ where $C = \{\Diamond^+ p, \Box^+\neg p, \neg p\}$ is an evident branch (Γ is a branch since $\{C\}$ is a DNF of $\mathcal{R}C\,;\Diamond^* p$). However, Γ is clearly not satisfiable.

Note also that it is impossible to distinguish if the eventuality $\Diamond^+ p$ is realized in Γ or not just by looking at the annotation of the link $C\binom{\Diamond^+ p}{\Diamond^* p}C$. We additionally need to determine whether or not $C \rhd p$. If we had, for instance, $C = \{\Diamond^+ p, p, \Box^+\Diamond^+ p\}$, then Γ would be satisfiable. ◂

To obtain an adequate notion of realization for eventualities, we adapt the notions of paths and runs from Chapter 4 to the current setting. To be able to distinguish when an eventuality is realized without having to compute support, we introduce two special types of links:

- **Fulfilling links** $C\binom{\Diamond^+ s}{s}D$ where $\Diamond^+ s \in C$, $D \in \mathcal{D}$ for some DNF $\mathcal{D}$ of $\mathcal{R}C\,; s$.
- **Delegating links** $C\binom{\Diamond^+ s}{\Diamond^+ s}D$ where $\Diamond^+ s \in C$, $D \in \mathcal{D}$ for some DNF $\mathcal{D}$ of $\mathcal{R}C\,;\Diamond^+ s$.

For simple diamond formulas $\Diamond s$, we still use simple links $C\binom{\Diamond s}{s}D$.

Branches are essentially defined as before but may now additionally contain fulfilling and delegating links. To account for the different types of links, we generalize the **functionality condition** for branches as follows: If Γ is a branch, $C\binom{s}{t}D \in \Gamma$ and $C\binom{s}{u}E \in \Gamma$, then $t = u$ and $D = E$. A branch Γ **realizes** $\Diamond s \mid C$ if $C\binom{\Diamond s}{t}D \in \Gamma$ for some t and D.

Figure 5.2 shows three branches. In all three cases, all diamond formulas are realized with links. The first two branches describe models of the clauses. This is not the case for the rightmost branch, where both clauses are unsatisfiable. Unlike the first two branches, the rightmost branch contains no fulfilling link to a clause that contains p. Instead, it has a loop formed by the delegating link from the lower clause to itself.

A **path for $\Diamond^+s$ in** Γ is a nonempty sequence

$$(C_1\binom{\Diamond^+s}{\Diamond^+s}C_2)(C_2\binom{\Diamond^+s}{\Diamond^+s}C_3)\dots(C_{n-2}\binom{\Diamond^+s}{\Diamond^+s}C_{n-1})(C_{n-1}\binom{\Diamond^+s}{t}C_n)$$

of links in Γ. A path of the above form is called a **run for $\Diamond^+s \mid C$** if $C_1 = C$ and $t = s$, and a **loop for $\Diamond^+s$** if $C_1 = C_n$ and $t = \Diamond^+s$. A branch Γ is **evident** if Γ contains no loops and realizes $\Diamond s \mid C$ for all $\Diamond s \in C \in \Gamma$. Because of the functionality restriction on branches, a branch that has a loop $(C_1\binom{\Diamond^+s}{\Diamond^+s}C_2)\dots(C_{n-1}\binom{\Diamond^+s}{\Diamond^+s}C_n)$ cannot have a run for any of the claims $\Diamond^+s \mid C_i$ ($i \in [1,n]$). In the first two branches in Figure 5.2, for every clause C that contains $\Diamond^+p$ we have a run for $\Diamond^+p \mid C$. The third branch contains a loop.

Proposition 5.4.2 Let Γ be an evident branch and $\Diamond^+s \in C \in \Gamma$. Then there is a unique run for $\Diamond^+s \mid C$ in Γ.

Proof Let Γ be evident and $\Diamond^+s \in C \in \Gamma$. We begin by showing that there is at most one run for $\Diamond^+s \mid C$ in Γ. Assume for contradiction there are two distinct runs π, π' for $\Diamond^+s \mid C$ in Γ. Then $\pi = \lambda_1 \dots \lambda_m$ and $\pi' = \lambda_1 \dots \lambda_i \lambda'_{i+1} \dots \lambda'_n$ where $i \in [0, m-1]$ and $\lambda_{i+1} \neq \lambda'_{i+1}$. Clearly, $\lambda_{i+1} = D\binom{\Diamond^+s}{t}E$ while $\lambda'_{i+1} = D\binom{\Diamond^+s}{t'}E'$ for some formulas t, t' and clauses D, E, E' such that $t \neq t'$ or $E \neq E'$. Contradiction to Γ being functional.

Now assume that $\Diamond^+s \mid C$ has no run in Γ. Then, since every eventuality in every clause is realized, Γ contains an infinite sequence $\pi = (C_1\binom{\Diamond^+s}{\Diamond^+s}C_2)(C_2\binom{\Diamond^+s}{\Diamond^+s}C_3)\dots$ where $C_1 = C$. Since Γ is finite, there are some $j > i \geq 1$ such that $C_i = C_j$, i.e., π contains a loop. Contradiction to Γ being evident. ■

Remark 5.4.3 By Proposition 5.4.2, a branch Γ that realizes every diamond formula is evident if and only if Γ contains a run for every $\Diamond^+s \mid C$ such that $\Diamond^+s \in C \in \Gamma$. So, instead of requiring the absence of loops we could equivalently define evidence by requiring that every claim has a run (which is also what we did in Chapter 4). The reason for the present definition is that it better matches the algorithmic ideas behind our present procedures. Unlike in Chapter 4, the procedures will never explicitly check the existence of runs. Instead, they will maintain the absence of loops as an invariant, whose violation indicates a failure in the search for an evident branch. ◂

Remark 5.4.4 We can now give an evident branch that contains all the N-clauses of the tableau in Figure 5.1:

$$s = \Box^+(\Diamond^* p \wedge \Diamond^* \neg p)$$

$$\begin{tikzcd}
 & \Diamond^+p,\ \Diamond^+\neg p,\ s \arrow[dl] \arrow[dr] & \\
p,\ \Diamond^+\neg p,\ s \arrow[rr, bend right] & & \Diamond^+p,\ \neg p,\ s \arrow[ll, bend right]
\end{tikzcd}$$

In Figure 5.1, every path from $\Diamond^+p$ in the clause $C_1 = \{\Diamond^+p,\ \Diamond^+\neg p,\ s\}$ to the clause $C_2 = \{p,\ \Diamond^+\neg p,\ s\}$ contains the link 1, while every path from C_1 to $C_3 = \{\Diamond^+p,\ \neg p,\ s\}$

contains the link 5. Hence, once link 1 or link 5 is removed, either C_2 or C_3 becomes inaccessible from C_1.

In the present case, the situation is different since the computation of a DNF is performed for every claim independently. In a sense, we can now have both link 1 and link 5 in the same branch, only that now the two links belong to two different DNF computations, which are not explicitly represented at the level of branches. ◂

We now show that the clauses of every evident branch are satisfiable. As before, it suffices to show the existence of a demo, with demos being defined as in § 4.1.

Theorem 5.4.5 The clauses of an evident branch form a demo.

Proof Let Γ be an evident branch and S the set of its clauses. We have to show that all clauses in S are $\Diamond$-admissible. The condition for $\Diamond s \in C$ follows the same as in the proof of Theorem 5.3.2. The condition for $\Diamond^+ s \in C$ proceeds as follows. Let $\Diamond^+ s \in C \in S$. By Proposition 5.4.2, the claim $\Diamond^+ s \mid C$ has a run $(C_1 \binom{\Diamond^+ s}{\Diamond^+ s} C_2) \dots (C_{n-1} \binom{\Diamond^+ s}{s} C_n)$ in Γ where $C_1 = C$. By Proposition 5.2.3, we obtain $C_{i+1} \rhd \mathfrak{R}C_i\,; \Diamond^+ s$ for every $i \in [1, n-2]$ and $C_n \rhd \mathfrak{R}C_{n-1}\,; s$. Hence, $C_1 \to_S^+ C_n \rhd s$. ■

5.4.2 Expansion Rules

To treat simple diamonds and eventualities in a uniform way, we introduce the notion of an expansion. Let C be a clause and $\Diamond s \in C$. An **expansion of** $\Diamond s \mid C$ is defined by case analysis on the shape of s:

- If $\Diamond s$ is simple and $\mathcal{D}$ is a DNF of $\mathfrak{R}C\,; s$,
  then $\{\, (s \mid D) \mid D \in \mathcal{D} \,\}$ is an expansion of $\Diamond s \mid C$.
- If $\Diamond s = \Diamond^+ t$, $\mathcal{D}$ is a DNF of $\mathfrak{R}C\,; t$, and $\mathcal{E}$ is a DNF of $\mathfrak{R}C\,; \Diamond^+ t$,
  then $\{\, (t \mid D) \mid D \in \mathcal{D} \,\} \cup \{\, (\Diamond^+ t \mid D) \mid D \in \mathcal{E} \,\}$ is an expansion of $\Diamond s \mid C$.

Example 5.4.6

- $\emptyset$ is an expansion of $\Diamond p \mid \{\Diamond p,\, \Box\neg p\}$.
- $\{p \vee \neg p \mid \{p, q\},\; p \vee \neg p \mid \{\neg p, q\}\}$ is an expansion of $\Diamond(p \vee \neg p) \mid \{\Diamond(p \vee \neg p),\, \Box q\}$.
- $\{\Diamond^+ p \mid \{\Diamond^+ p,\, \neg p,\, \Box^+\neg p\}\}$ is an expansion of $\Diamond^+ p \mid \{\Diamond^+ p,\, \Box^+\neg p\}$ since the set $\{p,\, \Box^*\neg p\}(= \mathfrak{R}\{\Diamond^+ p,\, \Box^+\neg p\}\,; p)$ has an empty DNF and $\{\{\Diamond^+ p,\, \neg p,\, \Box^+\neg p\}\}$ is a DNF of $\{\Diamond^+ p,\, \Box^*\neg p\}(= \mathfrak{R}\{\Diamond^+ p,\, \Box^+\neg p\}\,; \Diamond^+ p)$.
- $\{p \mid \{p\},\; \Diamond^+ p \mid \{p, \Diamond^+ p\}\}$ is an expansion of $\Diamond^+ p \mid \{\Diamond^+ p,\, \Box p\}$ since $\{\{p\}\}$ is a DNF of $\{p\}(= \mathfrak{R}\{\Diamond^+ p,\, \Box p\}\,; p)$ and $\{\{\Diamond^+ p,\, p\}\}$ is a DNF of the set $\{p,\, \Diamond^+ p\}(= \mathfrak{R}\{\Diamond^+ p,\, \Box p\}\,; \Diamond^+ p)$. ◂

The tree procedure for K* is obtained from the expansion rules in Figure 5.3. In addition to modifying the diamond rule to use expansions, we introduce a new rule, (LOOP), which closes branches that contain loops. We call (LOOP) the **loop rule**. Note that restricted to simple diamonds, the diamond rule behaves exactly the same as for K.

Remark 5.4.7 If one is willing to check support in order to recognize loops, one may be tempted to simplify the overall setup as follows.

$$(\Diamond)\ \frac{\Gamma}{\Gamma\,;D_1\,;C\binom{\Diamond s}{t_1}D_1\ \mid \cdots \mid \Gamma\,;D_n\,;C\binom{\Diamond s}{t_n}D_n}\quad \begin{array}{l}\Diamond s \in C \in \Gamma,\\ \Gamma \text{ does not realize } \Diamond s \mid C,\\ \{t_1 \mid D_1\,,\ldots,\,t_n \mid D_n\} \text{ expansion of } \Diamond s \mid C\end{array}$$

$$\textbf{(LOOP)}\ \frac{\Gamma}{\otimes}\quad \Gamma \text{ contains a loop}$$

Figure 5.3: Expansion rules for K*

- We do away with fulfilling and delegating links and employ simple links for eventualities as well as for simple diamonds (a simple link for an eventuality $\Diamond^+ s \in C$ would have the form $C\binom{\Diamond^+ s}{\Diamond^* s}D$). Since then every link has the form $C\binom{\Diamond s}{s}D$, we can simplify this notation to $C(\Diamond s)D$.
- We define a path for $\Diamond^+ s$ in Γ as a nonempty sequence

$$(C_1(\Diamond^+ s)C_2)(C_2(\Diamond^+ s)C_3)\ldots(C_{n-1}(\Diamond^+ s)C_n)$$

  of links in Γ. We call a path of the above form a run for $\Diamond^+ s \mid C$ if $C_1 = C$ and $C_n \rhd s$, and a loop for $\Diamond^+ s$ if $C_1 = C_n$ and $C_i \not\rhd s$ for every $i \in [1, n]$.
- The expansion rules for K* then consist of the diamond rule as given in § 5.3.2 for K and the loop rule using the new definition of loops.

The problem with this system (besides the fact that it relies on support) is that it results in an incorrect procedure. For instance, consider $\Gamma = \{C\}$ where $C = \{\Diamond^+ p,\ \Box^+\Diamond^+ p\}$. Clearly, C is satisfiable. Note that $\{C\}$ is a DNF of $\mathfrak{R}C\,;\Diamond^* p$. Since $C \not\rhd p$, the branch $\Gamma;C(\Diamond^+ p)C$, which is the only possible extension of Γ for the chosen DNF, contains a loop and is hence closed. Therefore, despite starting with a satisfiable clause and choosing a valid DNF, the procedure may not be able to find an evident branch.

To restore the correctness of the procedure, we need to impose the following additional restriction on DNFs:

- For every DNF $\mathcal{D}$ of $A\,;\Diamond^* s$ there is a DNF $\mathcal{D}_1$ of $A\,;s$ and a DNF $\mathcal{D}_2$ of $A\,;\Diamond^+ s$ such that $\mathcal{D}_1 \cup \mathcal{D}_2 \subseteq \mathcal{D}$.

In the above example, this condition would ensure that every DNF of $\mathfrak{R}C\,;\Diamond^* p$ contains at least one clause containing p (since $\mathfrak{R}C\,;p$ has a nonempty DNF).

However, this restriction is not natural for DNFs. In particular, it is not satisfied by DNFs computed via Hintikka decompositions (in § 5.2, we show that $\{C\}$ is a DNF of $\mathfrak{R}C\,;\Diamond^* p$ that can be computed via Hintikka decompositions).

Moreover, the restriction hides the fact that there are conceptually two degrees of freedom that are needed for a clausal tree procedure in the presence of eventualities. First, given a satisfiable formula set A, we need to be able to find a clause supporting A. This is achieved by DNFs. Second, given an eventuality $\Diamond^+ s \in C$ such that $\mathfrak{M}, v \models C$ and $v \to_{\mathfrak{M}} w$, we need to be able to find a clause supporting $\mathfrak{R}C\,;s$ if $\mathfrak{M}, w \models s$ and a clause supporting $\mathfrak{R}C\,;\Diamond^+ s$ if $\mathfrak{M}, w \models \Diamond^+ s$. This second requirement is achieved by our definition of expansions. It is fundamentally different from the first requirement, which will become clearer when we look at tree procedures for HPDL in Chapter 6 (see Remark 6.4.3). ◂

Clearly, a branch is evident if and only if it is maximal with respect to the rules in Figure 5.3. Hence, the rules are demonstration-sound (Theorem 5.4.5). The tree procedure induced by the rules is terminating by the same argument as for K (Theorem 5.3.5).

As it comes to refutation-soundness, the loop rule causes considerable complications. We must make sure that every branch to which the loop rule applies is unsatisfiable. The problem is that the rule may apply to branches all of whose clauses are satisfiable. This is shown by the next example.

Example 5.4.8 Consider the literal $s = \Box^+(q \vee \Box\neg p)$ and the following branch:

$$\begin{tikzcd}
\Diamond^+p,\ \neg p,\ q,\ s \arrow[d] \\
\Diamond^+p,\ \Box\neg p,\ s \arrow[u, bend left]
\end{tikzcd}$$

All clauses of the branch are satisfiable. Nevertheless, the branch contains a loop and is hence closed. Note that the branch can be obtained by starting with the upper clause. Essentially, the situation occurs because we choose from the possible extensions of the upper clause for $\Diamond^+p$ one that delegates the satisfaction of $\Diamond^+p$. If instead we added a fulfilling link to the clause $\{p, q, s\}$ or $\{p, \Box\neg p, s\}$ (which is possible since $\{\{p, q, s\}, \{p, \Box\neg p, s\}\}$ is a DNF of $\mathfrak{R}\{\Diamond^+p, \neg p, q, s\}\,;p$), the resulting branch would be evident. ◂

We solve the problem by extending the notion of satisfaction to delegating links. The idea is that a satisfiable delegating link for an eventuality $\Diamond^+s$ has to reduce the distance to a clause satisfying s. This way, branches with loops become unsatisfiable, which guarantees the refutation-soundness of the loop rule.

Let $\mathfrak{M}$ be a model, A be a set of formulas, and s a formula. The **distance from A to s in $\mathfrak{M}$** is defined as follows:

$$\delta_{\mathfrak{M}}As \;:=\; \min\{\, n \in \mathbb{N} \mid \exists v, w\colon\ v \to^{n}_{\mathfrak{M}} w \ \text{ and } \ \mathfrak{M}, v \vDash A \ \text{ and } \ \mathfrak{M}, w \vDash s \,\}$$

where $\min\emptyset = \infty$ and $n < \infty$ for all $n \in \mathbb{N}$.

Proposition 5.4.9 $\delta_{\mathfrak{M}}As < \infty$ if and only if $\mathfrak{M}$ satisfies $A\,;\Diamond^*s$.

Proof The claim is immediate by the definition of δ and the semantics of $\Diamond^*s$. ∎

A model $\mathfrak{M}$ **satisfies a link** $C\binom{\Diamond^+s}{\Diamond^+s}D$ if the following conditions are satisfied:

1. $\delta_{\mathfrak{M}}Cs > 0 \implies \delta_{\mathfrak{M}}Cs > \delta_{\mathfrak{M}}Ds$
2. $\delta_{\mathfrak{M}}Ds > 0$

A model **satisfies a branch** Γ if it satisfies all clauses and all delegating links of Γ.

Remark 5.4.10 If we adapt the new notion of satisfiability to the setup proposed in Remark 5.4.7, it turns out that the reason for the incorrect behaviour of the tree procedure is that the diamond rule is not refutation-sound. ◂

Remark 5.4.11 Theorem 5.4.5 actually makes a weaker statement than what is needed for demonstration-soundness if we take the new notion of branch satisfiability. Rather than showing that every evident branch is satisfiable, it shows that the *clauses* of every evident branch are satisfiable. This weaker claim suffices for the correctness of the decision procedure since in the end, we are just interested in the satisfiability of clauses. In fact, the stronger claim is not even true, which can be seen by looking at the branch

$$\begin{tikzcd}
C_1 = \Diamond^+ p \arrow[d] \\
C_2 = \Diamond^+ p,\ p \arrow[d] \\
C_3 = \quad p
\end{tikzcd}$$

The branch is evident but the link $C_1\binom{\Diamond^+ p}{\Diamond^+ p}C_2$ is not satisfiable since for every model $\mathfrak{M}$ satisfying the clause C_2 we have $\delta_{\mathfrak{M}}C_2 p = 0$. The example shows that link satisfiability differs considerably from our intuitive understanding of satisfiability and should primarily be seen as a technical device for showing refutation-soundness. ◂

We now argue that the loop rule is refutation-sound.

Proposition 5.4.12 Satisfiable branches contain no loops.

Proof Let Γ be a branch and $\mathfrak{M}$ a model of Γ. Assume, for contradiction, Γ has a loop $(C_1\binom{\Diamond^+ s}{\Diamond^+ s}C_2)\dots(C_{n-1}\binom{\Diamond^+ s}{\Diamond^+ s}C_n)$. Then $C_n = C_1$ and $n \geq 2$. Since for every $i \in [2,n]$, $\mathfrak{M}$ satisfies the link $C_{i-1}\binom{\Diamond^+ s}{\Diamond^+ s}C_i$ and $C_n = C_1$, we have $\delta_{\mathfrak{M}}C_i s > 0$ for every $i \in [1,n]$. Therefore, $\delta_{\mathfrak{M}}C_1 s > \delta_{\mathfrak{M}}C_2 s > \cdots > \delta_{\mathfrak{M}}C_n s = \delta_{\mathfrak{M}}C_1 s$. Contradiction. ∎

The refutation-soundness of the diamond rule is obtained with the following theorem.

Theorem 5.4.13 (Refutation-Soundness of ($\Diamond$)) Let Γ be a branch that does not realize $\Diamond s \mid C$ for some $\Diamond s \in C \in \Gamma$. Let $\mathfrak{M}$ be a model of Γ, and $\mathcal{E}$ be an expansion of $\Diamond s \mid C$. Then there is some $t \mid D \in \mathcal{E}$ such that $\Gamma; D; C\binom{\Diamond s}{t}D$ is a branch satisfied by $\mathfrak{M}$.

Proof Let Γ, $\Diamond s \in C \in \Gamma$, $\mathfrak{M}$ and $\mathcal{E}$ be as required. If $\Diamond s$ is simple, the claim reduces to Theorem 5.3.6. So let $\Diamond s = \Diamond^+ s'$. Let $\mathcal{D}$ be a DNF of $\mathfrak{R}C; s'$ and $\mathcal{D}'$ be a DNF of $\mathfrak{R}C; \Diamond^+ s'$ such that $\mathcal{E} = \{\, (s' \mid D) \mid D \in \mathcal{D} \,\} \cup \{\, (\Diamond^+ s' \mid D) \mid D \in \mathcal{D}' \,\}$. Since $\mathfrak{M}$ satisfies C, it also satisfies $\mathfrak{R}C; \Diamond^* s'$, i.e., $\mathfrak{M}$ satisfies $\mathfrak{R}C; s'$ or $\mathfrak{M}$ satisfies $\mathfrak{R}C; \Diamond^+ s'$.

Suppose $\mathfrak{M}$ satisfies $\mathfrak{R}C\,;s'$. Then there is some $D \in \mathcal{D}$ such that $\mathfrak{M}$ satisfies D. Let $\Delta = \Gamma\,;D\,;C\binom{\Diamond^+ s'}{s'}D$. Since Γ does not realize $\Diamond^+ s' \mid C$, Δ is functional and hence a branch. The model $\mathfrak{M}$ satisfies Δ since $\mathfrak{M}$ satisfies Γ and D.

Suppose $\mathfrak{M}$ does not satisfy $\mathfrak{R}C\,;s'$. Then $\mathfrak{M}$ satisfies $\mathfrak{R}C\,;\Diamond^+ s'$. Hence $\mathcal{D}'$ is nonempty and $\mathfrak{M}$ satisfies a clause in $\mathcal{D}'$. Let D be a clause in $\mathcal{D}'$ such that $\delta_{\mathfrak{M}}Ds'$ is minimal. Let $\Delta = \Gamma\,;D\,;C\binom{\Diamond^+ s'}{\Diamond^+ s'}D$. Since Γ does not realize $\Diamond^+ s' \mid C$, Δ is functional and hence a branch. Since $\mathfrak{M}$ satisfies some clause in $\mathcal{D}'$, $\mathfrak{M}$ satisfies D (Proposition 5.4.9). To show that $\mathfrak{M}$ satisfies Δ, it remains to verify that $\mathfrak{M}$ satisfies $C\binom{\Diamond^+ s'}{\Diamond^+ s'}D$.

First, we show that $\delta_{\mathfrak{M}}Ds' > 0$. Assume, for contradiction, $\delta_{\mathfrak{M}}Ds' = 0$. Then $\mathfrak{M}$ satisfies $D\,;s'$. Since $D \in \mathcal{D}'$, $\mathfrak{M}$ then satisfies $\mathfrak{R}C\,;\Diamond^+ s'\,;s'$, contradicting the assumption that $\mathfrak{M}$ does not satisfy $\mathfrak{R}C\,;s'$.

Finally, suppose $\delta_{\mathfrak{M}}Cs' > 0$. We show that $\delta_{\mathfrak{M}}Cs' > \delta_{\mathfrak{M}}Ds'$. Note that $\delta_{\mathfrak{M}}Cs' > 1$ since otherwise $\mathfrak{M}$ would satisfy $\mathfrak{R}C\,;s'$, contradicting our assumption. Also note that $\delta_{\mathfrak{M}}Cs' < \infty$ by Proposition 5.4.9 (since $\mathfrak{M}$ satisfies C and $\Diamond^+ s' \in C$). Hence, there are some v, w, u such that $v \rightarrow_{\mathfrak{M}} w \rightarrow_{\mathfrak{M}}^{\delta_{\mathfrak{M}}Cs'-1} u, \mathfrak{M}, v \vDash C$, and $\mathfrak{M}, u \vDash s'$. Then $\mathfrak{M}, w \vDash \mathfrak{R}C$ and, since $\delta_{\mathfrak{M}}Cs' - 1 > 0$, $\mathfrak{M}, w \vDash \Diamond^+ s'$. Consequently, there is a clause $E \in \mathcal{D}'$ such that $\mathfrak{M}, w \vDash E$. Clearly, $\delta_{\mathfrak{M}}Es' \le \delta_{\mathfrak{M}}Cs' - 1$. Since D is chosen from $\mathcal{D}'$ such that $\delta_{\mathfrak{M}}Ds'$ is minimal, we have $\delta_{\mathfrak{M}}Ds' \le \delta_{\mathfrak{M}}Es' < \delta_{\mathfrak{M}}Cs'$. The claim follows. ■

Since the expansion rules are demonstration-sound and refutation-sound, the induced tree procedure decides the satisfiability of clauses for K*. Formula satisfiability can be reduced to clause satisfiability as discussed in § 5.3.2.

5.5 Nominals

We now develop a tree procedure for the logic H*, which extends K* with nominals. The key observation underlying the procedure is that two clauses that contain the same nominal must be satisfied by the same state in every model of the branch. Thus, whenever two clauses of a branch have a nominal in common, we can add the union of the two clauses to the branch.

5.5.1 Branches and Evidence

Let Γ be a set of clauses and A be a set of formulas. We define the following notation to realize what we call **nominal propagation**:

$$A^\Gamma := A \cup \{\, s \mid \exists x \in A\, \exists C \in \Gamma\colon\ x \in C \text{ and } s \in C \,\}$$

Note that A^Γ is the least set of formulas that includes A and all clauses $C \in \Gamma$ that have a nominal in common with A, i.e., $A^\Gamma = A \cup \bigcup\{\, C \in \Gamma \mid A \cap C \text{ contains a nominal} \,\}$. In particular, we may only have $A^\Gamma \neq A$ if A contains a nominal.

Proposition 5.5.1 Let A be a set of formulas, $\mathfrak{M}$ be a model of a clause set Γ, and $w \in |\mathfrak{M}|$. Then $\mathfrak{M}, w \vDash A \iff \mathfrak{M}, w \vDash A^\Gamma$.

Proof The direction from right to left is immediate. As for the other direction, let $\mathfrak{M}$ be a model of a set Γ, and let $\mathfrak{M}, w \models A$. It suffices to show $\mathfrak{M}, w \models A \cup C$ for every clause $C \in \Gamma$ that has a nominal in common with A (if Γ contains no such clause, we have $A^\Gamma = A$ and the claim is trivial). So, let $C \in \Gamma$ have a nominal in common with A. Then, by the semantics of nominals, we have $\mathfrak{M}, w \models C$, and hence $\mathfrak{M}, w \models A \cup C$. ∎

A **branch** is now a nonempty set Γ of clauses as well as simple, fulfilling and delegating links that satisfies the following conditions:

- **Link coherence:** If $C\xi D \in \Gamma$, then $\{C, D^\Gamma\} \subseteq \Gamma$.
- **Functionality:** If $C\binom{s}{t}D \in \Gamma$ and $C\binom{s}{u}E \in \Gamma$, then $t = u$ and $D = E$.
- **Nominal coherence:** If $C \in \Gamma$, then $C^\Gamma \in \Gamma$.

A branch Γ may have several clauses that contain the same nominal. Nominal coherence guarantees that for every nominal in the branch there is always a unique largest clause that contains the nominal. This clause can be seen as the canonical representative of the state corresponding to the nominal.

Note that link coherence no longer requires a branch Γ to contain the target clause D of every link $C\xi D \in \Gamma$. It suffices if Γ contains D^Γ. This is because now, we think of a link $C\xi D$ as pointing to the clause D^Γ rather than D.

Proposition 5.5.2 If Γ is nominally coherent, then $(A^\Gamma)^\Gamma = A^\Gamma$.

Proof The inclusion $A^\Gamma \subseteq (A^\Gamma)^\Gamma$ is immediate. We now show $(A^\Gamma)^\Gamma \subseteq A^\Gamma$. Let $s \in (A^\Gamma)^\Gamma$. Then $s \in A^\Gamma$ or $s \in C$ for some $C \in \Gamma$ such that $C \cap A^\Gamma$ contains a nominal. In the former case we are done. In the latter case, we distinguish two subcases.

Let $C \cap A$ contain a nominal. Then $s \in C \subseteq A^\Gamma$, and we are done.

Let $C \cap D$ contain a nominal for some D such that $D \cap A$ contains a nominal. Since Γ is nominally coherent, $C \cup D \subseteq C^\Gamma \in \Gamma$. Since $D \subseteq C^\Gamma$, the set $C^\Gamma \cap A$ contains a nominal, and hence $s \in C \subseteq C^\Gamma \subseteq A^\Gamma$. ∎

Our next goal is to define what it means for a branch to be evident in the presence of nominals. First, we need to adapt our notion of paths. A **path for $\Diamond^+ s$ in Γ** is a nonempty sequence

$$(C_1\binom{\Diamond^+ s}{\Diamond^+ s}C_2)(C_2^\Gamma\binom{\Diamond^+ s}{\Diamond^+ s}C_3)\dots(C_{n-2}^\Gamma\binom{\Diamond^+ s}{\Diamond^+ s}C_{n-1})(C_{n-1}^\Gamma\binom{\Diamond^+ s}{t}C_n)$$

of links in Γ. A path of the above form is called a **run for $\Diamond^+ s \mid C$** if $C_1 = C$ and $t = s$, and a **loop for $\Diamond^+ s$** if $C_1 = C_n^\Gamma$ and $t = \Diamond^+ s$. Realization of diamond formulas is defined the same as for K*. The **core** $\mathcal{C}\Gamma := \{\, C \in \Gamma \mid C^\Gamma = C \,\}$ of a branch Γ is the set of all clauses of Γ that are either maximal or contain no nominal. The clauses in the core of a branch are the clauses that may potentially be used for the demo construction. Clauses that are not in the core are subsumed by larger clauses containing the same nominals. Note that no two clauses in the core of a branch contain the same nominal:

Lemma 5.5.3 Let Γ be a branch. If $x \in C \in \mathcal{C}\Gamma$ and $x \in D \in \mathcal{C}\Gamma$, then $C = D$.

Proof Since $x \in C \cap D$, we have $D \subseteq C^\Gamma = C$ and $C \subseteq D^\Gamma = D$. The claim follows. ∎

Also note that, by Proposition 5.5.2, we have $C^\Gamma \in \mathcal{C}\Gamma$ for every $C \in \Gamma$.

Given the new definition of loops, the formulation of evidence is almost unchanged compared to K*: A branch Γ is **evident** if Γ contains no loops and realizes $\Diamond s \mid C$ for every $\Diamond s \in C \in \mathcal{C}\Gamma$.

Remark 5.5.4 The fact that we only require realization of diamonds in the core of a branch can be seen as an optimization. It is equally possible to require an evident branch to realize all of its diamond formulas, like we do for K*. ◂

We can reprove Proposition 5.4.2 for the new definitions of loops and runs:

Proposition 5.5.5 Let Γ be an evident branch and $\Diamond^+ s \in C \in \mathcal{C}\Gamma$. Then there is a unique run for $\Diamond^+ s \mid C$ in Γ.

Proof Let Γ be evident and $\Diamond^+ s \in C \in \mathcal{C}\Gamma$. The proof that there is at most one run for $\Diamond^+ s \mid C$ in Γ proceeds exactly the same as for Proposition 5.4.2.

Now assume that $\Diamond^+ s \mid C$ has no run in Γ. Then, since every eventuality in every clause in $\mathcal{C}\Gamma$ is realized, Γ contains an infinite sequence $\pi = (C_1 \binom{\Diamond^+ s}{\Diamond^+ s} C_2)(C_2^\Gamma \binom{\Diamond^+ s}{\Diamond^+ s} C_3)\dots$ where $C_1 = C$. Since Γ is finite, there are some $j > i \geq 1$ such that $C_i = C_j^\Gamma$, i.e., π contains a loop. Contradiction to Γ being evident. ∎

Analogously to § 3.5, we call a clause set S **nominally admissible** if every nominal $x \in \mathcal{F}$ is contained in exactly one $C \in S$. We adapt the notion of **demos** from § 4.1 to H* by requiring that demos be nominally admissible. The proofs of demo satisfaction and demo existence remain essentially unchanged compared to § 4.1.2:

- Demo satisfaction and demo existence now follow by establishing a correspondence to nominally admissible Hintikka demos as defined in § 3.5.
- For Proposition 4.1.6, we additionally have to show that $\mathcal{H}_S$ is nominally admissible, which is immediate since S is nominally admissible and its clauses agree on literals with the Hintikka sets in $\mathcal{H}_S$.
- For Proposition 4.1.9, we have to show that ΛS is nominally admissible, which again follows since S is nominally admissible.

We now show that the clauses of an evident branch are satisfiable by proving that the core of the branch is contained in a demo. This suffices since every clause of the branch is a subset of a clause in its core.

Proposition 5.5.6 If $C \rhd s$ and $C \subseteq D$, then $D \rhd s$.

Proof Straightforward induction on the decomposition order of s. The base case of s being a literal is immediate since $C \rhd s \iff s \in C$. ∎

Theorem 5.5.7 The core of an evident branch is contained in a demo.

Proof Let Γ be an evident branch and let $\mathcal{D} = \mathcal{C}\Gamma \cup \{\, \{x\} \mid x \in \mathcal{F},\ \nexists C \in \mathcal{C}\Gamma\colon\ x \in C \,\}$. We show that $\mathcal{D}$ is a demo. By construction, $\mathcal{D}$ is nominally admissible (provided no two clauses in $C\Gamma$ contain the same nominal, which holds by Lemma 5.5.3). It remains to show that all clauses in $\mathcal{D}$ are $\Diamond$-admissible.

Let $\Diamond s \in C \in \mathcal{D}$. Then $C \in C\Gamma$. Since Γ realizes $\Diamond s \mid C$, by Proposition 5.2.3, there is some $D \in \Gamma$ that supports $\mathfrak{R}C\,;s$. Then $C^\Gamma \in C\Gamma \subseteq \mathcal{D}$. By Proposition 5.5.6, $D^\Gamma \rhd \mathfrak{R}C\,;s$. The claim follows.

Let $\Diamond^+ s \in C \in \mathcal{D}$. Then $C \in C\Gamma$. By Proposition 5.5.5, the claim $\Diamond^+ s \mid C$ has a run $(C_1\binom{\Diamond^+ s}{\Diamond^+ s}C_2)\dots(C^\Gamma_{n-1}\binom{\Diamond^+ s}{s}C_n)$ in Γ where $C_1 = C$. By Propositions 5.2.3 and 5.5.6, we obtain $C^\Gamma_{i+1} \rhd \mathfrak{R}C^\Gamma_i\,;\Diamond^+ s$ for every $i \in [1, n-2]$ and $C^\Gamma_n \rhd \mathfrak{R}C^\Gamma_{n-1}\,;s$. Clearly, all of the clauses C^Γ_i ($i \in [2, n]$) are in $C\Gamma$ and hence in $\mathcal{D}$. So, $C_1 = C^\Gamma_1 \rightarrow^+_{\mathcal{D}} C^\Gamma_n \rhd s$. ∎

Corollary 5.5.8 The clauses of an evident branch are satisfiable.

5.5.2 Expansion Rules

Let Γ be a branch and A a nonempty set of formulas. We call a clause set $\mathcal{D}$ a **nominally consistent DNF (nc-DNF) of** A **in** Γ if there is a DNF $\mathcal{D}'$ of A such that $\mathcal{D} = \{\, C \in \mathcal{D}' \mid C^\Gamma \text{ N-clause} \,\}$. Condition (1) in the definition of DNFs still holds for nc-DNFs in the following form.

Proposition 5.5.9 Let $\mathfrak{M}$ be a model of the branch Γ, A a nonempty set of formulas, and $\mathcal{D}$ an nc-DNF of A in Γ. Then, for all $w \in |\mathfrak{M}|$: $\mathfrak{M}, w \vDash A \iff \exists D \in \mathcal{D}\colon\ \mathfrak{M}, w \vDash D$.

Proof The direction from right to left follows from condition (1) for DNFs. As for the other direction, suppose $\mathfrak{M}, w \vDash A$. By the definition of nc-DNFs, there is a DNF $\mathcal{E}$ of A such that $\mathcal{D} = \{\, C \in \mathcal{E} \mid C^\Gamma \text{ N-clause} \,\}$. By condition (1) for DNFs, there is some $D \in \mathcal{E}$ such that $\mathfrak{M}, w \vDash D$. By Proposition 5.5.1, $\mathfrak{M}, w \vDash D^\Gamma$, and hence D^Γ is a clause (i.e., is not clashing). Consequently, $D \in \mathcal{D}$. ∎

Let $\Diamond s \in C \in \Gamma$. We define **nominally consistent expansions (nc-expansions) of** $\Diamond s \mid C$ **in** Γ by case analysis on the shape of s:

- If $\Diamond s$ is simple and $\mathcal{D}$ is an nc-DNF of $\mathfrak{R}C\,;s$ in Γ,
  then $\{\, (s \mid D) \mid D \in \mathcal{D} \,\}$ is an nc-expansion of $\Diamond s \mid C$ in Γ.
- If $\Diamond s = \Diamond^+ t$, $\mathcal{D}$ is an nc-DNF of $\mathfrak{R}C\,;t$ in Γ, and $\mathcal{E}$ is an nc-DNF of $\mathfrak{R}C\,;\Diamond^+ t$ in Γ, then $\{\, (t \mid D) \mid D \in \mathcal{D} \,\} \cup \{\, (\Diamond^+ t \mid D) \mid D \in \mathcal{E} \,\}$ is an nc-expansion of $\Diamond s \mid C$ in Γ.

Note that nc-expansions are defined from nc-DNFs in exactly the same way as expansions are defined from DNFs. Since, by definition, for every nc-DNF $\mathcal{D}$ of a formula set A there is a DNF $\mathcal{D}'$ of A such that $\mathcal{D} \subseteq \mathcal{D}'$, for every nc-expansion $\mathcal{E}$ of $\Diamond s \mid C$ there is an expansion $\mathcal{E}'$ of $\Diamond s \mid C$ such that $\mathcal{E} \subseteq \mathcal{E}'$.

The expansion rules for H* are adapted from the rules for K* as shown in Figure 5.4. The only change to the rules for K* (Figure 5.3) is that now the diamond rule introduces only nominally propagated clauses. This is done to maintain the nominal coherence of branches. To make sure that nominal propagation introduces no clashing clauses, we work with nc-expansions rather than ordinary expansions. Also, our definition of evidence for H* allows us to restrict the diamond rule to clauses in the core of the branch.

$$(\Diamond)\quad \frac{\Gamma}{\Gamma\,;D_1^\Gamma\,;C\binom{\Diamond s}{t_1}D_1 \mid \cdots \mid \Gamma\,;D_n^\Gamma\,;C\binom{\Diamond s}{t_n}D_n}\quad \begin{array}{l}\Diamond s \in C \in \mathcal{C}\Gamma,\\ \Gamma \text{ does not realize } \Diamond s \mid C,\\ \{t_1 \mid D_1, \dots, t_n \mid D_n\} \text{ nc-expansion}\\ \qquad\qquad\qquad\quad \text{of } \Diamond s \mid C \text{ in } \Gamma\end{array}$$

$$(\textsc{loop})\quad \frac{\Gamma}{\otimes}\quad \Gamma \text{ contains a loop}$$

Figure 5.4: Expansion rules for H*

Example 5.5.10 Here is a derivation of an evident branch from an initial clause with eventualities. The numbers indicate the order in which the links and clauses are introduced (clause 0 being the initial clause).

$$\begin{tikzcd}
\Diamond^+p,\ \neg p,\ \Box(x \wedge \Diamond^+p \wedge \neg p),\ \Diamond\Box\neg p\ {}^0 \arrow[d, "1"'] \arrow[dr, dashed] \arrow[dr, "3"] & \\
x,\ \Diamond^+p,\ \neg p\ {}^1 \arrow[d, "2"'] & x,\ \Diamond^+p,\ \neg p,\ \Box\neg p\ {}^3 \arrow[d, "4"] \\
p\ {}^2 & \Diamond^+p,\ \neg p\ {}^4 \arrow[l, "5"]
\end{tikzcd}$$

In the final branch, the links 1, 4, 5 constitute a run for $\Diamond^+p$ in clause 0.

The dashed link is used to indicate the implicit redirection of link 1 (due to nominal propagation) that occurs when clause 3 is added. Note that before clause 3 is added, the links 1, 2 constitute a run for $\Diamond^+p$ in clause 0. When clause 3 is added, this run disappears since clause 1 is no longer in the core. ◂

Once again, we have that a branch is evident if and only if it is maximal. Hence, by Corollary 5.5.8, the rules are demonstration-sound (with respect to the weak notion of branch satisfaction that does not require the satisfaction of links). The termination of the tree procedure induced by the rules is once again immediate since the rules only introduce clauses and links over $\mathcal{F}$ (see Theorem 5.3.5).

The proof of refutation-soundness needs some adaptations. Note that adding nominals to the language does not affect the validity of Proposition 5.4.9. The most important additional observation that we need for nominals is that distance is invariant under nominal propagation:

Proposition 5.5.11 Let $\mathfrak{M}$ be a model of a branch Γ. Then $\delta_{\mathfrak{M}}As = \delta_{\mathfrak{M}}A^\Gamma s$.

Proof Immediately follows from Proposition 5.5.1 by the definition of δ. ∎

With this, we can reprove Proposition 5.4.12 as follows.

Proposition 5.5.12 Satisfiable branches contain no loops.

Proof Let Γ be a branch and $\mathfrak{M}$ a model of Γ. Assume, for contradiction, Γ has a loop $(C_1\binom{\Diamond^+ s}{\Diamond^+ s}C_2)\dots(C_{n-1}^\Gamma\binom{\Diamond^+ s}{\Diamond^+ s}C_n)$. Then $C_1 = C_1^\Gamma = C_n^\Gamma$ and $n \geq 2$. Since for every $i \in [2, n]$, $\mathfrak{M}$ satisfies the link $C_{i-1}^\Gamma\binom{\Diamond^+ s}{\Diamond^+ s}C_i$, by Proposition 5.5.11, we have $\delta_{\mathfrak{M}}C_i^\Gamma s > 0$ for every $i \in [1, n]$ (the claim for $i = 1$ follows with $C_1^\Gamma = C_n^\Gamma$). Therefore, $\delta_{\mathfrak{M}}C_i^\Gamma s > \delta_{\mathfrak{M}}C_{i+1}s = \delta_{\mathfrak{M}}C_{i+1}^\Gamma s$ for all $i \in [1, n]$, which contradicts $C_1^\Gamma = C_n^\Gamma$. ∎

Proposition 5.5.12 justifies the refutation-soundness of the loop rule. Since the diamond rule now employs nominally consistent expansions, its proof of refutation-soundness also needs to be adapted.

Theorem 5.5.13 (Refutation-Soundness of ($\Diamond$)) Let Γ be a branch that does not realize $\Diamond s \mid C$ for some $\Diamond s \in C \in \mathcal{C}\Gamma$. Let $\mathfrak{M}$ be a model of Γ, and $\mathcal{E}$ be an nc-expansion of $\Diamond s \mid C$ in Γ. Then there is some $t \mid D \in \mathcal{E}$ such that $\Gamma ; D^\Gamma ; C\binom{\Diamond s}{t}D$ is a branch satisfied by $\mathfrak{M}$.

Proof To obtain the claim, we need to adapt the proofs of Theorems 5.3.6 and 5.4.13 in the following places:

1. Assuming that $\mathfrak{M}$ satisfies $\mathfrak{R}C ; s$ (or $\mathfrak{R}C ; s'$ where $\Diamond s = \Diamond^+ s'$), we select a clause $D \in \mathcal{D}$ (where $\mathcal{D}$ is now an nc-DNF of $\mathfrak{R}C ; s$ or $\mathfrak{R}C ; s'$, respectively) such that $\mathfrak{M}$ satisfies D. The existence of D now follows by Proposition 5.5.9.
2. Assuming that $\mathfrak{M}$ satisfies $\mathfrak{R}C ; \Diamond^+ s'$, we select a clause $D \in \mathcal{D}'$ (where $\mathcal{D}'$ is now an nc-DNF of $\mathfrak{R}C ; \Diamond^+ s'$) such that $\delta_{\mathfrak{M}}Ds'$ is minimal. Since $\mathfrak{M}$ satisfies $\mathfrak{R}C ; \Diamond^+ s'$, every DNF of $\mathfrak{R}C ; \Diamond^+ s'$ contains a clause D such that $\delta_{\mathfrak{M}}Ds' < \infty$ (Proposition 5.4.9). Therefore, by Proposition 5.5.9, our choice of D is not restricted by going from ordinary DNFs to nc-DNFs in Γ.
3. Given a clause D satisfied by $\mathfrak{M}$, we have to show that D^Γ is satisfied by $\mathfrak{M}$, which is immediate by Proposition 5.5.1.
4. We have to show that $\Delta = \Gamma ; D^\Gamma ; C\xi D$ (where ξ is one of $\binom{\Diamond s}{s}$, $\binom{\Diamond^+ s'}{s'}$ and $\binom{\Diamond^+ s'}{\Diamond^+ s'}$) is nominally coherent. This follows from the nominal coherence of Γ since $(D^\Gamma)^\Delta = (D^\Gamma)^\Gamma = D^\Gamma \in \Delta$ (Proposition 5.5.2). ∎

We conclude that the expansion rules induce a decision procedure for H*.

5.6 Discussion

5.6.1 Satisfaction Operators

It is straightforward to extend our system for H* to the logic $\text{H}^*_@$, which extends H* with satisfaction operators. We introduce **satisfaction links** of the form $C\binom{x:s}{s}D$ where $x : s \in C$ and $D \in \mathcal{D}$ for some DNF $\mathcal{D}$ of $\{x, s\}$. A branch Γ **realizes** $(x : s)|\mathbf{C}$ if it contains a link $C\binom{x:s}{s}D$. **Evident** branches are now those that realize every diamond and every satisfaction formula in their core.

Demos for $\mathrm{H}^*_@$ are defined as for H^* with the additional condition that a demo $\mathcal{D}$ has to contain a clause D with $D \rhd \{x, s\}$ for every $x : s \in C \in \mathcal{D}$. Together with nominal admissibility, this suffices to show demo satisfaction and demo existence.

The adaptation of Theorem 5.5.7 is also straightforward: Since an evident branch Γ contains a link $C\binom{x:s}{s}D$ for every $x : s \in C \in C\Gamma$, we also have $D^\Gamma \in \Gamma$. This entails the additional demo condition for $\mathrm{H}^*_@$ since $D^\Gamma \rhd \{x, s\}$ (Propositions 5.2.3 and 5.5.6).

The expansion rule for satisfaction formulas looks as follows.

$$\frac{\Gamma}{\Gamma\,;D_1^\Gamma\,;C\binom{x:s}{s}D_1 \mid \ldots \mid \Gamma\,;D_n^\Gamma\,;C\binom{x:s}{s}D_n} \quad \begin{array}{l} x : s \in C \in C\Gamma, \\ \Gamma \text{ does not realize } (x : s) \mid C, \\ \{D_1, \ldots, D_n\} \text{ nc-DNF of } \{x, s\} \text{ in } \Gamma \end{array}$$

When the rule is added to the rules for H^* (Figure 5.4), we again have that a branch is evident if and only if it is maximal. Hence, the rules are demonstration-sound. Since the new rule respects $\mathcal{F}$, the termination of the procedure is unaffected. For refutation-soundness, we have to show the following proposition:

Proposition 5.6.1 Let Γ be a branch that does not realize $(x : s) \mid C$ for some $x : s \in C \in C\Gamma$. Let $\mathfrak{M}$ be a model of Γ and $\mathcal{D}$ be an nc-DNF of $\{x, s\}$. Then there is a clause $D \in \mathcal{D}$ such that $\Gamma\,;D^\Gamma\,;C\binom{x:s}{s}D$ is a branch satisfied by $\mathfrak{M}$.

Proof Analogous to the case for simple diamonds in the proof of Theorem 5.5.13. ∎

Since the treatment of satisfaction operators is largely orthogonal to that of other constructs, the present approach can also be used to extend the procedures in Chapters 6 and 7 to satisfaction operators. Therefore, in the following, we restrict our considerations to logics without satisfaction operators.

5.6.2 Pattern-Based Blocking

For simplicity and uniformity of presentation, we omitted several optimizations that may considerably reduce the number of clauses and branches that need to be explored by the decision procedure. Let us now discuss one such optimization that is known as **pattern-based blocking** (see [126, 127]).

Consider the following branch:

$$\begin{tikzcd}
\Diamond p,\ \Box(\neg p \vee q),\ p \arrow[d] & \Diamond p,\ \Box(\neg p \vee q),\ q \\
p,\ q &
\end{tikzcd}$$

Clearly, the diamond $\Diamond p$ in the upper right clause (C_2) can be realized by a link to the bottom clause (C_3). How can we determine this without computing a DNF of $\mathfrak{R}C_2\,;p$? We can look at the upper left clause (C_1) and observe that $\mathfrak{R}C_2\,;p = \mathfrak{R}C_1\,;p$. Since $\Diamond p \mid C_1$ is realized with a link to C_3, we can realize $\Diamond p \mid C_2$ in the same way.

In general, we have that a simple diamond $\Diamond s$ can be realized in a clause in C by a link to an existing clause if there is some $\Diamond t \in D \in \Gamma$ such that $\Diamond t$ is realized in D and $\Re C ; s = \Re D ; t$.

Moreover, all of our results still hold if we allow links of the form $C\binom{\Diamond s}{s}D$ where D is a *superset* of some clause in a DNF of $\Re C ; s$. If we do this, we can realize a simple diamond $\Diamond s$ in C if there is some $\Diamond t \in D \in \Gamma$ such that $\Diamond t$ is realized in D and $\Re C ; s \subseteq \Re D ; t$. Extending the idea to eventualities and fulfilling links, we arrive at the following two expansion rules:

$$\frac{\Gamma}{\Gamma ; C\binom{\Diamond s}{s}E} \quad \begin{array}{l} \Diamond s \in C \in C\Gamma \text{ simple,} \\ \Gamma \text{ does not realize } \Diamond s \mid C, \\ D\binom{t}{u}E \in \Gamma,\ \Re C ; s \subseteq \Re D ; u \end{array} \qquad \frac{\Gamma}{\Gamma ; C\binom{\Diamond^+ s}{s}E} \quad \begin{array}{l} \Diamond^+ s \in C \in C\Gamma, \\ \Gamma \text{ does not realize } \Diamond^+ s \mid C, \\ D\binom{t}{u}E \in \Gamma,\ \Re C ; s \subseteq \Re D ; u \end{array}$$

We call these rules **blocking rules**. In practice [98], pattern-based blocking is realized by maintaining a data structure containing a set $\Re C ; s$ for every $\Diamond s \in C \in \Gamma$ such that $\Diamond s \mid C$ has been previously realized with the diamond rule. Before applying the diamond rule to some unrealized $\Diamond t \mid D$, we check if $\Re D ; t$ is contained (possibly as a subset) in the data structure. If so, we know that $\Diamond t \mid D$ can be realized. Actually realizing $\Diamond t \mid D$ by inserting a link is not necessary as long as one is just interested in deciding satisfiability and not in obtaining a satisfying model.

Remark 5.6.2 The name "pattern-based blocking" originates in the notion of a *pattern*, which is used in [126, 127] for sets of the form $\Re C ; s$ (where $\Diamond s \in C$). ◂

What about delegating links? A blocking rule in the spirit of the above two rules for delegating links would look as follows.

$$\frac{\Gamma}{\Gamma ; C\binom{\Diamond^+ s}{\Diamond^+ s}E} \quad \begin{array}{l} \Diamond^+ s \in C \in C\Gamma, \\ \Gamma \text{ does not realize } \Diamond^+ s \mid C, \\ D\binom{t}{u}E \in \Gamma,\ \Re C ; \Diamond^+ s \subseteq \Re D ; u \end{array}$$

However, this rule is not refutation-sound. Consider, for instance, the first branch in Figure 5.2 up to the second clause, i.e., $\Gamma = \{C_1,\ C_1\binom{\Diamond^+ p}{\Diamond^+ p}C_2,\ C_2\}$. Clearly, Γ is satisfiable. Since $C_1\binom{\Diamond^+ p}{\Diamond^+ p}C_2 \in \Gamma$ and $\Re C_2 = \emptyset \subseteq \{\neg p\} = \Re C_1$, the rule introduces the unsatisfiable link $C_2\binom{\Diamond^+ p}{\Diamond^+ p}C_2$, after which the branch becomes closed.

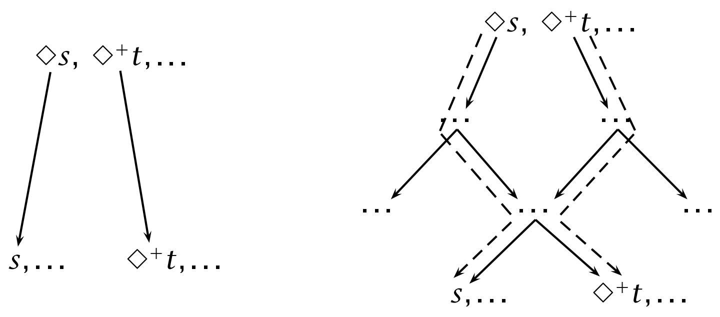
5.6.3 Tree Search on Graph Tableaux

Finally, let us relate tree search and graph tableaux by describing how tree search can be realized as a tableau construction and search strategy on top of graph tableaux. We restrict our attention to the procedure for K*.

Although we did not make it explicit in § 5.2, graph tableaux as defined in Chapter 4 also provide an intuitive way to compute DNFs of formula sets. Let $\mathcal{T}$ be a graph tableau and $C \in \mathcal{T}$. We call an N-clause $D \in \mathcal{T}$ an **N-successor of C in $\mathcal{T}$** if D is reachable from C by a path in $\mathcal{T}$ that contains no N-clauses besides D and possibly C. Let $\mathcal{T}$ be a complete graph tableau containing $\lfloor A \rfloor$ for some formula set A. It is easily seen that the set of all N-successors of $\lfloor A \rfloor$ in $\mathcal{T}$ is a DNF of A.

With this in mind, a tableau branch Γ can be seen as a graph tableau $\mathcal{T}$ together with a partial function φ, which represents the links of Γ. We have $\varphi(\Diamond s, C) = (t, D)$ if and only if Γ contains the link $C\binom{\Diamond s}{t}D$. For compatibility with the underlying graph tableau, we require D to be an N-successor of $\lfloor \mathfrak{R}C; s \rfloor$ if $\Diamond s$ is simple, and an N-successor of $\lfloor \mathfrak{R}C; t \rfloor$ or $\lfloor \mathfrak{R}C; \Diamond^+ t \rfloor$ if $\Diamond s = \Diamond^+ t$. Note that the functionality of Γ corresponds to φ being a function.

Schematically, the correspondence can be depicted as follows:



The branch on the left-hand side is represented as the graph tableau on the right-hand side (links being depicted by solid arrows) together with the function φ, represented by the two dashed arrows (one for every link of the branch). As suggested by the picture, φ can be thought of as a set of paths between N-clauses of the graph tableau. For every N-clause C and every formula $\Diamond s \in C$, φ is allowed to contain at most one path.

A tree procedure can be seen as a backtracking procedure that combines an incremental construction of a graph tableau with the construction of φ. The goal of the procedure is to make φ total on diamonds in N-clauses that are φ-reachable from some initial N-clause (where a clause D is a φ-successor of C if $\varphi(\Diamond s, C) = (t, D)$ for some $\Diamond s$ and t, and φ-reachability is the reflexive-transitive closure of the φ-successor relation) without introducing any cycles of the form $\varphi(\ldots\varphi(\Diamond^+ s, C)) = (\Diamond^+ s, C)$. The underlying graph tableau only needs to be extended incrementally and on demand, namely when the construction of φ rejects all N-successors of a clause in the tableau but there are potentially more N-successors in a completion of the tableau. When φ is loop-free and total on the reachable clauses, the graph tableau is a positive certificate for the satisfiability of the initial clause. When the search for φ fails, the tableau is a negative certificate. In both cases, we can obtain small certificates (provided they exist) by suitably guiding the search for φ.

5.7 Related Work

► Existing tableau procedures for hybrid logic [115, 30, 29, 117, 125, 127, 39, 107] all employ prefixes. This is because they rely on fine-grained propagation of equational information introduced by nominals. To make such propagation possible, one needs prefixes as a syntactic representation of states that is independent from the formulas that are assumed to hold at the states. Two prefixes are usually seen as equivalent (denoting the same state) if they carry the same formulas. One consequence of this representation is that prefixes may grow over time by accumulating new formulas, which also means that two equivalent prefixes may at some time lose their equivalence. In the presence of eventualities, this poses a problem since it conflicts with the refutation-soundness of the loop rule. We cannot reject a loop unless we know that the loop is persistent, i.e., cannot disappear if we propagate new formulas to one of the prefixes on the loop.

Our procedure differs from existing procedures for hybrid logic in that it does not employ prefixes. It is static in that, once part of the branch, clauses and links remain unchanged. Nominal propagation is realized by creating new clauses and hence does not interfere with the refutation-soundness of the loop rule.

Also, the clausal representation considerably simplifies our termination argument compared to the arguments for prefixed calculi. The reason is that, unlike with prefixes, there can be no two distinct clauses that contain the same formulas. Hence, the termination of the procedure is immediately implied by the finiteness of the formula universe.

► The fundamental idea behind our refutation-soundness arguments for the tree procedures for K* and H* evolved in work with Sigurd Schneider [175]. It builds on an idea in Baader's [11] correctness proof (Proposition 4.7) for a tree procedure for test-free PDL. Besides the extension to hybrid logic, the present approach differs from the arguments in [11] and [175] in that, by assigning semantics to delegating links, we are able to argue the refutation-soundness of our system in a largely standard way, without introducing special-purpose invariants on branches.

► Besides the connections to Baader's and Schneider's work, our loop rule resembles the criterion for the *ignorability* of branches in De Giacomo and Massacci's [52] tableau system for CPDL. Technically, however, our system is quite different from De Giacomo and Massacci's. In particular, De Giacomo and Massacci's correctness proof relies on complex model transformations that are unnecessary in our case.

► Yet another approach to tree procedures for PDL is taken by Abate et al. [3]. Their procedure is rather different from our system in that it decides the satisfiability of a branch not only based on the structure of the branch itself but also based on information from previously explored branches. This allows Abate et al. to avoid proving the refutation-soundness of the loop rule. Rather than seeing branches with loops as unsatisfiable, they view their satisfiability as not yet determined. The obligation to determine the satisfiability of eventualities causing loops can then be passed on to later branches. By reasoning over multiple branches, the system incorporates certain aspects of graph procedures. This is also the reason why, similarly to graph procedures, it does not scale easily to hybrid logic. Since satisfying models produced by the procedure may include material from multiple branches, one would have to enforce that these branches taken together satisfy some form of nominal coherence, which is a nontrivial task.

6 Tree Search for Test-Free Hybrid PDL

In this chapter, we extend our incremental procedure for H* to HPDL. While the overall design of the procedure and its correctness proofs at the modal level remain the same, the computation of DNFs needs to be refined considerably.

The reason for this is that, while terminating when the language is restricted to K* or H*, the decomposition relation induced by the table in Figure 3.1 (see § 4.1.1) is not terminating for full (H)PDL. If the decomposition relation does not terminate, the definitions of upward closure, support, and Hintikka decompositions no longer ensure the desired correspondence between clauses and Hintikka sets. In particular, a Hintikka set H no longer satisfies $\bigwedge H \rhd H$ (Proposition 4.1.7), which is crucial for establishing a connection between clausal demos and Hintikka demos.

To reestablish the required properties, we refine the decomposition table (i.e., the set of decomposition rules) in Figure 3.1 so as to make the induced decomposition relation terminating. Since the refined table yields different constituents for diamond and box formulas than the original table, we need to argue the correctness of the new table. We base the correctness argument on a straightforward language-theoretic semantics of programs. Once the new table is in place, the construction of the procedure and the proof of its correctness proceed following the ideas in Chapter 5.

Since tests cause additional complications, for clarity of presentation we restrict ourselves to the test-free case. The extension of the procedure to HPDL with tests will be discussed separately in Chapter 7.

Structure of the chapter. We begin by detailing the problems caused by the lack of termination of the decomposition relation and formulate requirements on a refined decomposition table (§ 6.1). Next, we make a brief excursion into the theory of regular languages (§ 6.2). The theory is used to state and prove the correctness properties of the refined decomposition table that we devise in § 6.3. We then introduce the notions of expansion, links, branches, and evidence (§ 6.4). To show that evident branches are satisfiable, we adapt the definition of demos to work with the refined decomposition table (§ 6.5). With all the prerequisites in place, we state the expansion rules underlying the decision procedure and argue their correctness (§ 6.6). We conclude with a discussion of related work (§ 6.7).

6.1 Decomposition Tables

Like in Chapters 4 and 5, and unlike in the more theoretical approach in Chapter 3, we want to determine the satisfaction of nonliteral formulas from the satisfaction of literals. In Chapters 4 and 5, this is achieved via the notion of support. As noted in § 4.1, in the case of K* and H*, we have $\Lambda H \rhd H$ for every Hintikka set H (Proposition 4.1.7). Hence, the following three claims are equivalent:

1. $\mathfrak{M}, w \models s$
2. $\exists C\colon\ \mathfrak{M}, w \models C$ and $C \rhd s$
3. $\exists H\colon\ H$ Hintikka set and $s \in H$ and $\mathfrak{M}, w \models \Lambda H$

It is precisely these equivalences that allow us to reduce the satisfaction of nonliteral formulas to the satisfaction of literals. Equivalence (1) $\iff$ (2) is a key property of support and follows for K* and H* by Lemma 4.1.2, Proposition 3.3.11, and Proposition 4.1.7. Equivalence (1) $\iff$ (3) is an essential prerequisite for the correctness of the DNF computation via Hintikka decompositions in § 5.2 (see Remark 5.2.5). The implication (2) $\implies$ (3) follows since H_C is a Hintikka set for every N-clause C (Proposition 4.1.4), while Proposition 4.1.7 is used to show the converse implication.

When we go to HPDL, Proposition 4.1.7 is no longer true and hence (2) and (3) are no longer equivalent. More importantly, neither of the equivalences (1) $\iff$ (2) and (1) $\iff$ (3) is true. The reason for this is that the alpha-beta decomposition rules in Figure 3.1 may not terminate on formulas $\langle\alpha^*\rangle s$ or $[\alpha^*]s$ where α is a complex program. For instance, $\langle a^{**}\rangle p$ decomposes into p and $\langle a^*\rangle\langle a^{**}\rangle p$, while $\langle a^*\rangle\langle a^{**}\rangle p$ decomposes into $\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p$ and $\langle a^{**}\rangle p$, the original formula (see Figure 1.1).

For the equivalence (1) $\iff$ (2), decomposition cycles affect the implication from left to right. For instance, consider the formula $[a^{**}]p$. Although the formula is clearly satisfiable, it is not supported by any N-clause. For contradiction, suppose $C \rhd [a^{**}]p$. Consider the shortest derivation of $[a^{**}]p \in A$ where A is the upward closure of C. Clearly, the derivation contains as a subderivation a derivation of $[a^*][a^{**}]p \in A$. However, every derivation of $[a^*][a^{**}]p \in A$ must contain a strictly shorter derivation of $[a^{**}]p \in A$. Contradiction.

For the equivalence (1) $\iff$ (3), the implication from left to right holds. What fails is the converse implication. Consider the formula $\langle a^{**}\rangle(p \wedge \neg p)$. Clearly, the formula is unsatisfiable. At the same time, $H = \{q,\ \langle a^{**}\rangle(p \wedge \neg p),\ \langle a^*\rangle\langle a^{**}\rangle(p \wedge \neg p)\}$ is a Hintikka set containing $\langle a^{**}\rangle(p \wedge \neg p)$. We have $\Lambda H = \{q\}$, and $\{q\}$ is clearly satisfiable. Note also that $\{q\} = \Lambda H \not\rhd H$.

We call a set of alpha-beta decomposition rules as given in Figure 3.1 a **decomposition table**. Both the definition of Hintikka sets and that of support build on the decomposition table in Figure 3.1. To deal with the above anomalies, we will adopt a second, more general decomposition table that will allow to decompose an alpha formula α into constituents $\alpha_1, \dots, \alpha_n$ and a beta formula β into $\beta_1, \dots, \beta_n$ (where $n \geq 1$ may vary for different formulas) such that:

- If $\mathcal{F}$ is a formula universe and $s \in \mathcal{F}$ decomposes into $t_1, \dots, t_n$, then $\{t_1, \dots, t_n\} \subseteq \mathcal{F}$.
- If α is an alpha formula, then $\mathfrak{M}, w \models \alpha \iff \mathfrak{M}, w \models \alpha_i$ for every $i \in [1, n]$.
- If β is a beta formula, then $\mathfrak{M}, w \models \beta \iff \mathfrak{M}, w \models \beta_i$ for some $i \in [1, n]$.

We view these three conditions as fundamental compatibility properties that should be satisfied by every decomposition table.

Analogously to the table in Figure 3.1, every decomposition table induces a notion of literals, decomposition relation, Hintikka sets, Hintikka decompositions, upward closure and support in the intuitive way. For instance, given a decomposition table, a **Hintikka set** defined with respect to the table is a nonempty set H of formulas that satisfies the following properties.

- For every predicate p: $\{p, \neg p\} \nsubseteq H$.
- If $\alpha \in H$, then $\{\alpha_1, \dots, \alpha_n\} \subseteq H$.
- If $\beta \in H$, then $\{\beta_1, \dots, \beta_n\} \cap H \neq \emptyset$.

We call a decomposition table **admissible** if for every Hintikka set H it holds $\Lambda H \rhd H$, where Hintikka sets and support are defined with respect to the table. As we mentioned above, this entails the equivalence of the claims (2) and (3). From this, it follows that the equivalences (1) $\iff$ (2) and (1) $\iff$ (3) are both valid for every admissible table with the following two observations.

- The proof of Proposition 4.1.2 (i.e., the implication (2) $\implies$ (1)) generalizes to arbitrary decomposition tables (not necessarily admissible). The generalized proof is mostly analogous to the proof of Proposition 4.1.2 but proceeds by induction on the derivation of $s \in A$ where A is the upward closure of C (this induction order does not require the decomposition relation to be terminating).
- The implication (1) $\implies$ (3) immediately follows from the property $\mathfrak{M}, w \models s \implies \exists H \in \mathcal{D}\colon \mathfrak{M}, w \models H$, satisfied by every Hintikka decomposition $\mathcal{D}$ of $\{s\}$. While in § 5.2, the property is argued for H* and Hintikka decompositions with respect to the table in Figure 3.1, the argument extends to HPDL and arbitrary decomposition tables as long as they satisfy the fundamental compatibility properties from above (containment in the formula universe and compatibility with the semantics of alpha and beta formulas).

A decomposition table is called **terminating** if so is the induced decomposition relation. As we already noted in § 4.1.1, the transitive closure of a terminating decomposition relation forms a well-founded decomposition order. So, a decomposition table is terminating if and only if it induces a well-founded decomposition order.

Let us now show that the termination of a decomposition table is a sufficient criterion for its admissibility.

Proposition 6.1.1 If a decomposition table is terminating, then it is admissible.

Proof Let the notions of Hintikka sets, upward closure and support be defined with respect to some terminating decomposition table. Let H be a Hintikka set, $s \in H$, and C be the set of all literals in H. We show $C \rhd s$ by induction on the decomposition order of s as induced by the table. We distinguish three cases. If s is a literal, the claim follows since $s \in C$. So, let $s = \alpha$ decompose into n constituents $\alpha_1, \dots, \alpha_n$. By the inductive hypothesis, we have $C \rhd \alpha_i$ for all $i \in [1, n]$. Hence, by the definition of the upward closure, $C \rhd \alpha$. The case for $s = \beta$ proceeds similarly. ∎

Our goal for the next two sections will be to define a terminating decomposition table that we can use for the decision procedure. It will coincide with the table in Figure 3.1 on conjunctions and disjunctions but will differ on nonliteral diamond and box formulas.

Remark 6.1.2 Instead of modifying the decomposition table to enforce termination, it is possible to work with the original decomposition table in Figure 3.1 (complemented by the notion of demos from Chapter 3.3). However, instead of clauses, we would have to represent states by Hintikka sets. Moreover, to check that a Hintikka set H is "admissible" (in the sense that the satisfaction of all of its literals entails that of its nonliteral formulas), the decision procedure would have to explicitly compute the relation $\xrightarrow{\alpha}_S$ for every $\langle\alpha\rangle s \in H$. Previous tableau-based approaches to PDL [161, 52, 3, 93, 200] all rely on explicit admissibility checks. We choose modifying the decomposition table over explicit admissibility checks since it is conceptually simpler and more modular in that the correctness of the modified decomposition table can be proven in isolation from the correctness of the overall procedure. ◂

6.2 Language-Theoretic Semantics

In this section, we give a brief summary of regular languages (see [110, 156, 181]), which we use as a semantics for programs of test-free HPDL. Based on this semantics, we state some properties that we will need to justify the correctness of our terminating decomposition rules for diamond and box formulas. We do not give any proofs. Proofs will be given in § 7.1 for a more general semantics that also accounts for tests.

A **word** is a finite sequence of actions $a_1 \dots a_n$ where $n \geq 0$. We denote the empty word by ε. The letters σ and τ range over words. A **language** is a set of words. We write $L \cdot L'$ for the concatenation of the languages L and L' (i.e., for $\{\, \sigma\tau \mid \sigma \in L,\ \tau \in L' \,\}$). Given a language L, we define:

$$L^0 := \{\varepsilon\} \qquad L^{n+1} := L \cdot L^n \qquad L^* := \bigcup_{n \in \mathbb{N}} L^n$$

To every test-free program α, we assign a **language** $\mathcal{L}\alpha$:

$$\mathcal{L}a := \{a\} \qquad \mathcal{L}(\alpha + \beta) := \mathcal{L}\alpha \cup \mathcal{L}\beta \qquad \mathcal{L}(\alpha\beta) := \mathcal{L}\alpha \cdot \mathcal{L}\beta \qquad \mathcal{L}(\alpha^*) := (\mathcal{L}\alpha)^*$$

Proposition 6.2.1 Let L_1, L_2, L_3 be languages. Then:

1. $(L_1 \cup L_2) \cdot L_3 = (L_1 \cdot L_3) \cup (L_2 \cdot L_3)$
2. $(L_1 \cdot L_2) \cdot L_3 = L_1 \cdot (L_2 \cdot L_3)$
3. $L_1 \cdot \{\varepsilon\} = \{\varepsilon\} \cdot L_1 = L_1$
4. $L_1^* = \{\varepsilon\} \cup (L_1 \cdot L_1^*)$

Given a model $\mathfrak{M}$, we define the relations $\xrightarrow{\sigma}_{\mathfrak{M}} \subseteq |\mathfrak{M}| \times |\mathfrak{M}|$ by induction on σ:

$$v \xrightarrow{\varepsilon}_{\mathfrak{M}} w \iff v = w$$
$$v \xrightarrow{a\sigma}_{\mathfrak{M}} w \iff \exists u\colon\ v \xrightarrow{a}_{\mathfrak{M}} u \text{ and } u \xrightarrow{\sigma}_{\mathfrak{M}} w$$

Lemma 6.2.2 $v \xrightarrow{\sigma\tau}_{\mathfrak{M}} w \iff \exists u\colon\ v \xrightarrow{\sigma}_{\mathfrak{M}} u$ and $u \xrightarrow{\tau}_{\mathfrak{M}} w$

Proposition 6.2.3

1. $v \xrightarrow{\alpha}_{\mathfrak{M}} w \iff \exists \sigma \in \mathcal{L}\alpha\colon\ v \xrightarrow{\sigma}_{\mathfrak{M}} w$
2. $\mathfrak{M}, v \models \langle\alpha\rangle s \iff \exists \sigma \in \mathcal{L}\alpha\, \exists w\colon\ v \xrightarrow{\sigma}_{\mathfrak{M}} w$ and $\mathfrak{M}, w \models s$
3. $\mathfrak{M}, v \models [\alpha] s \iff \forall \sigma \in \mathcal{L}\alpha\, \forall w\colon\ v \xrightarrow{\sigma}_{\mathfrak{M}} w$ implies $\mathfrak{M}, w \models s$

6.3 Diamond and Box Decompositions

We now introduce the terminating decomposition rules for diamond and box formulas and argue their correctness with respect to the criteria formulated in § 6.1. We begin by formulating the central properties that we require of our construction.

A set $\mathcal{D}$ of formulas is a **decomposition of** $\langle\alpha\rangle s$ if it satisfies the following conditions:

1. If $t \in \mathcal{D}$, then:
   a) $t \in \mathcal{F}$ for every formula universe $\mathcal{F}$ containing $\langle\alpha\rangle s$.
   b) $t = s$ or $t = \langle a\rangle\langle\beta_1\rangle \dots \langle\beta_n\rangle s$ for some action a and programs $\beta_1, \dots, \beta_n$ ($n \geq 0$).
2. $\mathcal{L}\alpha = \bigcup_{\langle\beta_1\rangle\dots\langle\beta_n\rangle s \in \mathcal{D}} \mathcal{L}\beta_1 \cdot \ldots \cdot \mathcal{L}\beta_n$

Note that the inclusion $\langle\beta_1\rangle \dots \langle\beta_n\rangle s \in \mathcal{D}$ in condition (2) also applies to formulas of the form s (i.e., formulas where $n = 0$). Note also that condition (1.b) implies that every formula in $\mathcal{D}$ is either a literal (as defined with respect to the table in Figure 3.1) or a proper subformula of $\langle\alpha\rangle s$.

The definition of **box decompositions** (i.e., decompositions of box formulas) is obtained from the above definition for diamonds by replacing every occurrence of the formulas $\langle\alpha\rangle s$, $\langle a\rangle\langle\beta_1\rangle \dots \langle\beta_n\rangle s$, and $\langle\beta_1\rangle \dots \langle\beta_n\rangle s$ by the formulas $[\alpha]s$, $[a][\beta_1] \dots [\beta_n]s$, and $[\beta_1] \dots [\beta_n]s$, respectively.

Conditions (1.b) and (2) ensure that the constituents of s contained in a diamond or a box decomposition of s satisfy the semantic equivalences for beta and alpha formulas, respectively:

Proposition 6.3.1 Let $\mathcal{D}$ be a decomposition of $\langle\alpha\rangle s$ and $\mathcal{E}$ be a decomposition of $[\alpha]s$.
1. $\mathfrak{M}, w \models \langle\alpha\rangle s \iff \exists t \in \mathcal{D}\colon \mathfrak{M}, w \models t$
2. $\mathfrak{M}, w \models [\alpha]s \iff \forall t \in \mathcal{E}\colon \mathfrak{M}, w \models t$

Proof We show claim (1) (claim (2) follows analogously). For the direction from left to right, let $\mathfrak{M}, w \models \langle\alpha\rangle s$. By Proposition 6.2.3 (2), there is some $\sigma \in \mathcal{L}\alpha$ and some v such that $w \xrightarrow{\sigma}_{\mathfrak{M}} v$ and $\mathfrak{M}, v \models s$. By condition (2) for diamond decompositions, $\sigma \in \mathcal{L}\beta_1 \cdot \ldots \cdot \mathcal{L}\beta_n$ for some $\langle\beta_1\rangle \dots \langle\beta_n\rangle s \in \mathcal{D}$. By n applications of Proposition 6.2.3 (2) and Lemma 6.2.2, we obtain $\mathfrak{M}, w \models \langle\beta_1\rangle \dots \langle\beta_n\rangle s$.

For the other direction, let $t \in \mathcal{D}$ and $\mathfrak{M}, w \models t$. By condition (1.b) for diamond decompositions, $t = \langle\beta_1\rangle \dots \langle\beta_n\rangle s$ for some $n \geq 0$ and $\beta_1, \dots, \beta_n$. By n applications of Proposition 6.2.3 (2) and Lemma 6.2.2, there is some $\sigma \in \mathcal{L}\beta_1 \cdot \ldots \cdot \mathcal{L}\beta_n$ and v such that $w \xrightarrow{\sigma}_{\mathfrak{M}} v$ and $\mathfrak{M}, v \models s$. By condition (2) for diamond decompositions, $\sigma \in \mathcal{L}\alpha$. Hence, by Proposition 6.2.3 (2), $\mathfrak{M}, w \models \langle\alpha\rangle s$. ∎

Once again, let us fix a formula universe $\mathcal{F}$ and consider only formulas from $\mathcal{F}$.

We now give a construction that, given a diamond or a box formula s, yields a decomposition of s. We will use this construction to define the decomposition rules for diamonds and boxes. Taken together with the decomposition rules for conjunctions and disjunctions, they will yield a terminating decomposition table for HPDL. Intuitively, we define the diamond and box decomposition rules as "macro rules" based on multiple applications of the rules in Figure 3.1.

We define the **alpha-beta closure** of a formula set A (with respect to the table in Figure 3.1) as the least set B of formulas containing A such that

- $\{\alpha_1, \alpha_2\} \subseteq B$ whenever $\alpha \in B$, and
- $\{\beta_1, \beta_2\} \subseteq B$ whenever $\beta \in B$.

We write $\mathcal{A}A$ for the alpha-beta closure of A. Clearly, $\mathcal{A}A \subseteq \mathcal{F}$ whenever $A \subseteq \mathcal{F}$. Consequently (Proposition 3.2.3), $\mathcal{A}A$ is finite whenever so is A, its cardinality being at most linear in the sum of the sizes of the formulas in A.

The algorithm for computing diamond and box decompositions employs a restriction of the alpha-beta closure that we define as follows. Given a formula s and a set A, we define the **alpha-beta closure of** A **halting at** s (written $\mathcal{A}_sA$) as the least set B of formulas containing A such that

- $\{\alpha_1, \alpha_2\} \subseteq B$ whenever $\alpha \in B$ and $\alpha \neq s$, and
- $\{\beta_1, \beta_2\} \subseteq B$ whenever $\beta \in B$ and $\beta \neq s$.

Clearly, for every s and A, we have $\mathcal{A}_sA \subseteq \mathcal{A}A$. Let $s = \langle\alpha\rangle t$ or $s = [\alpha]t$. We define:

$$\mathfrak{D}s := \{\, u \in \mathcal{A}_t\{s\} \mid u = t \ \text{ or } \ u \text{ literal}\,\}$$

Example 6.3.2 Consider the formula $\langle(a+b)^*\rangle p$. We have:

$$\begin{aligned} \mathcal{A}_p\{\langle(a+b)^*\rangle p\} &= \{\langle(a+b)^*\rangle p,\ p,\ \langle a+b\rangle\langle(a+b)^*\rangle p, \\ &\qquad \langle a\rangle\langle(a+b)^*\rangle p,\ \langle b\rangle\langle(a+b)^*\rangle p\} \\ \mathfrak{D}(\langle(a+b)^*\rangle p) &= \{p,\ \langle a\rangle\langle(a+b)^*\rangle p,\ \langle b\rangle\langle(a+b)^*\rangle p\} \end{aligned}$$ ◂

As we will show in the following, $\mathfrak{D}(\langle\alpha\rangle s)$ is a decomposition of $\langle\alpha\rangle s$ and $\mathfrak{D}([\alpha]s)$ is a decomposition of $[\alpha]s$. Since the argument proceeds analogously for diamond and box decompositions, we will only detail the claim for $\mathfrak{D}(\langle\alpha\rangle s)$. The argument for $\mathfrak{D}([\alpha]s)$ is then obtained simply by replacing diamonds by boxes everywhere in the argument for $\mathfrak{D}(\langle\alpha\rangle s)$.

We begin by observing that $\mathfrak{D}s$ satisfies condition (1.a) for diamond (and box) decompositions. This holds since, as noted above, $\mathcal{A}\{s\} \subseteq \mathcal{F}$ whenever $s \in \mathcal{F}$, and $\mathfrak{D}s \subseteq \mathcal{A}\{s\}$.

For conditions (1.b) and (2), it is essential that $\mathfrak{D}(\langle\alpha\rangle s)$ is defined based on $\mathcal{A}_s\{\langle\alpha\rangle s\}$ rather than $\mathcal{A}\{\langle\alpha\rangle s\}$, which is demonstrated by the following example.

Example 6.3.3 Consider the formula $\langle a^*\rangle\langle b^*\rangle p$. We have:

$$\begin{aligned} \mathcal{A}_{\langle b^*\rangle p}\{\langle a^*\rangle\langle b^*\rangle p\} &= \{\langle a^*\rangle\langle b^*\rangle p,\ \langle b^*\rangle p,\ \langle a\rangle\langle a^*\rangle\langle b^*\rangle p\} \\ \mathfrak{D}(\langle a^*\rangle\langle b^*\rangle p) &= \{\langle b^*\rangle p,\ \langle a\rangle\langle a^*\rangle\langle b^*\rangle p\} \end{aligned}$$

However, $\mathcal{A}\{\langle a^*\rangle\langle b^*\rangle p\} = \mathcal{A}_{\langle b^*\rangle p}\{\langle a^*\rangle\langle b^*\rangle p\} \cup \{p,\ \langle b\rangle\langle b^*\rangle p\}$. Hence, if we replace $\mathcal{A}_s$ by $\mathcal{A}$ in the definition of $\mathfrak{D}$, we obtain the set $\mathcal{D} = \{\langle b^*\rangle p,\ \langle a\rangle\langle a^*\rangle\langle b^*\rangle p,\ p,\ \langle b\rangle\langle b^*\rangle p\}$ as a candidate decomposition of $\langle a^*\rangle\langle b^*\rangle p$. Clearly, the set violates condition (1.b) for diamond decompositions because it contains p. Moreover, it violates condition (2) because it contains $\langle b\rangle\langle b^*\rangle p$ but $\mathcal{L}b \nsubseteq \mathcal{L}(a^*)$. ◂

Condition (1.b) for diamond decompositions follows for $\mathfrak{D}(\langle\alpha\rangle s)$ by the following lemma.

Lemma 6.3.4 If $t \in \mathcal{A}_s\{\langle\alpha\rangle s\}$, then $t = \langle\beta_1\rangle \dots \langle\beta_n\rangle s$ for some $n \geq 0$ and $\beta_1, \dots, \beta_n$.

Proof Straightforward induction on the derivation of the formulas in $\mathcal{A}_s\{\langle\alpha\rangle s\}$. ∎

Finally, we show that $\mathfrak{D}(\langle\alpha\rangle s)$ satisfies condition (2) for diamond decompositions.

Lemma 6.3.5 If $\langle\beta_1\rangle \dots \langle\beta_n\rangle s \in \mathcal{A}_s\{\langle\alpha\rangle s\}$, then $\mathcal{L}\beta_1 \cdot \dots \cdot \mathcal{L}\beta_n \subseteq \mathcal{L}\alpha$.

Proof Let $\langle\beta_1\rangle \dots \langle\beta_n\rangle s \in \mathcal{A}_s\{\langle\alpha\rangle s\}$. We proceed by induction on the derivation of $\langle\beta_1\rangle \dots \langle\beta_n\rangle s \in \mathcal{A}_s\{\langle\alpha\rangle s\}$. The base case is trivial, so let $\langle\beta_1\rangle \dots \langle\beta_n\rangle s \in \mathcal{A}_s\{\langle\alpha\rangle s\} \setminus \{\langle\alpha\rangle s\}$. Then there is some formula $t \in \mathcal{A}_s\{\langle\alpha\rangle s\}$ such that $\langle\beta_1\rangle \dots \langle\beta_n\rangle s$ is obtained from t by the application of one of the decomposition rules in Figure 3.1. By Lemma 6.3.4, t has the form $\langle\gamma_1\rangle \dots \langle\gamma_m\rangle s$. By the inductive hypothesis, we have $\mathcal{L}\gamma_1 \cdot \dots \cdot \mathcal{L}\gamma_m \subseteq \mathcal{L}\alpha$. Moreover, since we may not apply any rules to s, we have $m > 0$. The claim follows by a case analysis on γ_1 with Proposition 6.2.1. ∎

Lemma 6.3.6 $\mathcal{A}_{\langle\beta\rangle s}\{\langle\alpha\rangle\langle\beta\rangle s\} \subseteq \mathcal{A}_s\{\langle\alpha\rangle\langle\beta\rangle s\}$

Proof Let $t \in \mathcal{A}_{\langle\beta\rangle s}\{\langle\alpha\rangle\langle\beta\rangle s\}$. We show that $t \in \mathcal{A}_s\{\langle\alpha\rangle\langle\beta\rangle s\}$ by induction on the derivation of $t \in \mathcal{A}_{\langle\beta\rangle s}\{\langle\alpha\rangle\langle\beta\rangle s\}$. The base case follows since, clearly, $\langle\alpha\rangle\langle\beta\rangle s \in \mathcal{A}_s\{\langle\alpha\rangle\langle\beta\rangle s\}$. For the inductive step, suppose $t \in \mathcal{A}_{\langle\beta\rangle s}\{\langle\alpha\rangle\langle\beta\rangle s\} \setminus \{\langle\alpha\rangle\langle\beta\rangle s\}$. Then there is some formula $u \in \mathcal{A}_{\langle\beta\rangle s}\{\langle\alpha\rangle\langle\beta\rangle s\}$ such that t is obtained from u by the application of one of the decomposition rules in Figure 3.1. By the inductive hypothesis, $u \in \mathcal{A}_s\{\langle\alpha\rangle\langle\beta\rangle s\}$. By Lemma 6.3.4, u has the form $\langle\gamma_1\rangle \dots \langle\gamma_n\rangle\langle\beta\rangle s$. Moreover, since we may not apply any rules to $\langle\beta\rangle s$, we have $n > 0$. The claim follows by a straightforward case analysis on γ_1. ∎

Proposition 6.3.7 $\mathcal{L}\alpha = \bigcup\limits_{\langle\beta_1\rangle \dots \langle\beta_n\rangle s \in \mathfrak{D}(\langle\alpha\rangle s)} \mathcal{L}\beta_1 \cdot \dots \cdot \mathcal{L}\beta_n$

Proof The inclusion $\bigcup\limits_{\langle\beta_1\rangle \dots \langle\beta_n\rangle s \in \mathfrak{D}(\langle\alpha\rangle s)} \mathcal{L}\beta_1 \cdot \dots \cdot \mathcal{L}\beta_n \subseteq \mathcal{L}\alpha$ follows with Lemma 6.3.5. As for the other inclusion, let α be a program. We show $\forall s\, \forall\sigma \in \mathcal{L}\alpha$: $\sigma \in \mathcal{L}\beta_1 \cdot \dots \cdot \mathcal{L}\beta_n$ for some $\langle\beta_1\rangle \dots \langle\beta_n\rangle s \in \mathfrak{D}(\langle\alpha\rangle s)$ by induction on α. We distinguish four cases depending on the shape of α. Each of these cases except for the first one has two subcases depending on the shape of σ.

Let $\alpha = a$. Then the claim is trivial since $\mathfrak{D}(\langle a\rangle s) = \{\langle a\rangle s\}$ (which is the case since $\langle a\rangle s$ is a literal).

Let $\alpha = \beta + \gamma$. Let $\sigma = \varepsilon \in \mathcal{L}(\beta + \gamma)$. Then $\varepsilon \in \mathcal{L}\beta$ or $\varepsilon \in \mathcal{L}\gamma$. Without loss of generality, let $\varepsilon \in \mathcal{L}\beta$. By the inductive hypothesis for β, we have $\varepsilon \in \mathcal{L}\beta_1 \cdot \dots \cdot \mathcal{L}\beta_n$ for some $\langle\beta_1\rangle \dots \langle\beta_n\rangle s \in \mathfrak{D}(\langle\beta\rangle s)$. Since $\varepsilon \notin \mathcal{L}a \cdot L$ for any action a and language L, β_1 is not an action. Hence, we have $n = 0$. It now suffices to show that $s \in \mathfrak{D}(\langle\alpha\rangle s)$, which follows since $s \in \mathcal{A}_s\{\langle\beta\rangle s\}$ and $\langle\beta\rangle s \in \mathcal{A}_s\{\langle\alpha\rangle s\}$.

Let $\sigma = a\tau \in \mathcal{L}(\beta + \gamma)$. Then $\sigma \in \mathcal{L}\beta$ or $\sigma \in \mathcal{L}\gamma$. Without loss of generality, let $\sigma \in \mathcal{L}\beta$. By the inductive hypothesis for β, we have $\sigma \in \mathcal{L}\beta_1 \cdot \ldots \cdot \mathcal{L}\beta_n$ for some $t = \langle\beta_1\rangle \ldots \langle\beta_n\rangle s \in \mathfrak{D}(\langle\beta\rangle s)$. Since $a\tau \notin \{\varepsilon\}$, t is a literal (such that $\beta_1 = a$). It remains to show that $t \in \mathfrak{D}(\langle\alpha\rangle s)$, which follows since t is a literal, $t \in \mathcal{A}_s\{\langle\beta\rangle s\}$, and $\langle\beta\rangle s \in \mathcal{A}_s\{\langle\alpha\rangle s\}$.

Let $\alpha = \beta\gamma$. Let $\sigma = \varepsilon \in \mathcal{L}(\beta\gamma)$. Then $\varepsilon \in \mathcal{L}\beta$ and $\varepsilon \in \mathcal{L}\gamma$. By the inductive hypothesis for β, we have $\varepsilon \in \mathcal{L}\beta_1 \cdot \ldots \cdot \beta_m$ for some $\langle\beta_1\rangle \ldots \langle\beta_m\rangle\langle\gamma\rangle s \in \mathfrak{D}(\langle\beta\rangle\langle\gamma\rangle s)$. As in the case for $\varepsilon \in \mathcal{L}(\beta+\gamma)$, β_1 is not an action, and hence $m = 0$. So, $\langle\gamma\rangle s \in \mathcal{A}_{\langle\gamma\rangle s}\{\langle\beta\rangle\langle\gamma\rangle s\}$. By the inductive hypothesis for γ, we have $\varepsilon \in \mathcal{L}\gamma_1 \cdot \ldots \cdot \gamma_n$ for some $\langle\gamma_1\rangle \ldots \langle\gamma_n\rangle s \in \mathfrak{D}(\langle\gamma\rangle s)$. Again, since γ_1 is not an action, we have $n = 0$. It remains to show that $s \in \mathfrak{D}(\langle\alpha\rangle s)$. By the above, $\langle\gamma\rangle s \in \mathcal{A}_{\langle\gamma\rangle s}\{\langle\beta\rangle\langle\gamma\rangle s\} \subseteq \mathcal{A}_s\{\langle\beta\rangle\langle\gamma\rangle s\}$ (Lemma 6.3.6) and $s \in \mathcal{A}_s\{\langle\gamma\rangle s\}$. Hence, $s \in \mathcal{A}_s\{\langle\beta\rangle\langle\gamma\rangle s\}$. Since $\langle\beta\rangle\langle\gamma\rangle s \in \mathcal{A}_s\{\langle\alpha\rangle s\}$, we have $s \in \mathcal{A}_s\{\langle\alpha\rangle s\}$. The claim follows.

Let $\sigma = a\tau \in \mathcal{L}(\beta\gamma)$. We distinguish two possible subcases.

- Let $\tau = \tau_1\tau_2$ such that $a\tau_1 \in \mathcal{L}\beta$ and $\tau_2 \in \mathcal{L}\gamma$. By the inductive hypothesis for β, we have $a\tau_1 \in \mathcal{L}\beta_1 \cdot \ldots \cdot \mathcal{L}\beta_n$ for some $t = \langle\beta_1\rangle \ldots \langle\beta_n\rangle\langle\gamma\rangle s \in \mathfrak{D}(\langle\beta\rangle\langle\gamma\rangle s)$. Since $a\tau_1 \notin \{\varepsilon\}$, t is a literal. Then $a\tau \in \mathcal{L}\beta_1 \cdot \ldots \cdot \mathcal{L}\beta_n \cdot \mathcal{L}\gamma$. It remains to show that $t \in \mathfrak{D}(\langle\alpha\rangle s)$, which follows since t is a literal, $t \in \mathcal{A}_{\langle\gamma\rangle s}\{\langle\beta\rangle\langle\gamma\rangle s\} \subseteq \mathcal{A}_s\{\langle\beta\rangle\langle\gamma\rangle s\}$ (Lemma 6.3.6), and $\langle\beta\rangle\langle\gamma\rangle s \in \mathcal{A}_s\{\langle\alpha\rangle s\}$.
- Let $\varepsilon \in \mathcal{L}\beta$ and $a\tau \in \mathcal{L}\gamma$. By the inductive hypothesis for β, we have $\varepsilon \in \mathcal{L}\beta_1 \cdot \ldots \cdot \mathcal{L}\beta_m$ for some $t = \langle\beta_1\rangle \ldots \langle\beta_m\rangle\langle\gamma\rangle s \in \mathfrak{D}(\langle\beta\rangle\langle\gamma\rangle s)$. Since β_1 may not be an action, we have $m = 0$. Hence, $\langle\gamma\rangle s \in \mathcal{A}_{\langle\gamma\rangle s}\{\langle\beta\rangle\langle\gamma\rangle s\}$. By the inductive hypothesis for γ, we have $a\tau \in \mathcal{L}\gamma_1 \cdot \ldots \cdot \mathcal{L}\gamma_n$ for some $t = \langle\gamma_1\rangle \ldots \langle\gamma_n\rangle s \in \mathfrak{D}(\langle\gamma\rangle s)$. Since $a\tau \notin \{\varepsilon\}$, t is a literal. It remains to show that $t \in \mathfrak{D}(\langle\alpha\rangle s)$, which follows since t is a literal, $t \in \mathcal{A}_s\{\langle\gamma\rangle s\}$, $\langle\gamma\rangle s \in \mathcal{A}_{\langle\gamma\rangle s}\{\langle\beta\rangle\langle\gamma\rangle s\} \subseteq \mathcal{A}_s\{\langle\beta\rangle\langle\gamma\rangle s\}$ (Lemma 6.3.6), and $\langle\beta\rangle\langle\gamma\rangle s \in \mathcal{A}_s\{\langle\alpha\rangle s\}$.

Let $\alpha = \beta^*$. Let $\sigma = \varepsilon \in \mathcal{L}(\beta^*)$. Then $s \in \mathfrak{D}(\langle\alpha\rangle s)$. The claim follows since $\varepsilon \in \{\varepsilon\} = (\mathcal{L}\beta)^0 \subseteq (\mathcal{L}\beta)^* = \mathcal{L}(\beta^*)$.

Finally, let $\sigma = a\tau \in \mathcal{L}(\beta^*)$. Since $a\tau \notin \{\varepsilon\}$, we have $\tau = \tau_1\tau_2$ such that $a\tau_1 \in \mathcal{L}\beta$ and $\tau_2 \in (\mathcal{L}\beta)^*$. By the inductive hypothesis for β, we have $a\tau \in \mathcal{L}\beta_1 \cdot \ldots \cdot \mathcal{L}\beta_n$ for some $t = \langle\beta_1\rangle \ldots \langle\beta_n\rangle\langle\beta^*\rangle s \in \mathfrak{D}(\langle\beta\rangle\langle\beta^*\rangle s)$. Again since $a\tau \notin \{\varepsilon\}$, t is a literal. It remains to show that $t \in \mathfrak{D}(\langle\alpha\rangle s)$, which follows since t is a literal, $t \in \mathcal{A}_{\langle\beta^*\rangle s}\{\langle\beta\rangle\langle\beta^*\rangle s\} \subseteq \mathcal{A}_s\{\langle\beta\rangle\langle\beta^*\rangle s\}$ (Lemma 6.3.6), and $\langle\beta\rangle\langle\beta^*\rangle s \in \mathcal{A}_s\{\langle\alpha\rangle s\}$. ∎

We have shown that $\mathfrak{D}(\langle\alpha\rangle s)$ is a decomposition of $\langle\alpha\rangle s$. By an analogous argument for boxes, we conclude that $\mathfrak{D}([\alpha]s)$ is a decomposition of $[\alpha]s$. Finally, note that since $\mathcal{A}\{s\}$ is linear in the size of s, $\mathfrak{D}s$ can be computed in polynomial time.

By Proposition 6.3.1, we have:

Corollary 6.3.8

1. $\mathfrak{M}, w \vDash \langle\alpha\rangle s \iff \exists t \in \mathfrak{D}(\langle\alpha\rangle s)\colon \mathfrak{M}, w \vDash t$
2. $\mathfrak{M}, w \vDash [\alpha]s \iff \forall t \in \mathfrak{D}([\alpha]s)\colon \mathfrak{M}, w \vDash t$

The terminating decomposition table for test-free HPDL then looks as shown in Figure 6.1. Note that the table yields the same notions of alpha formulas, beta formulas

α	$\{\alpha_1,\dots,\alpha_n\}$	
$s \wedge t$	$\{s,\, t\}$	
$[\gamma]s$	$\mathfrak{D}([\gamma]s)$	γ not an action

β	$\{\beta_1,\dots,\beta_n\}$	
$s \vee t$	$\{s,\, t\}$	
$\langle\gamma\rangle s$	$\mathfrak{D}(\langle\gamma\rangle s)$	γ not an action

Figure 6.1: Terminating decomposition table for test-free HPDL

and literals as the table in Figure 3.1. By Corollary 6.3.8, the rules for diamonds and boxes satisfy the respective semantic equivalences. The table is terminating since every constituent of a nonliteral formula s is either a literal or a proper subformula of s. Since the table is terminating, it induces a well-founded decomposition order on formulas.

6.4 Branches and Evidence

We now introduce the notions of branches and evidence. The definitions follow the general pattern developed for K* and H* in Chapter 5. Given the terminating decomposition table in Figure 6.1, the **upward closure** of a formula set A induced by the table is the least set B of formulas containing A such that:

$$\alpha \in B \iff \{\alpha_1,\dots,\alpha_n\} \subseteq B$$
$$\beta \in B \iff \{\beta_1,\dots,\beta_n\} \cap B \neq \emptyset$$

The notion of support adapts accordingly. From now on, we will consider only support based on the terminating decomposition table. A DNF of a set A is defined in the same way as in for H* (§ 5.2), now with the new notion of support. Thus, the validity of Propositions 5.2.3 and 5.2.4 remains unaffected. Also, the computation of DNFs via Hintikka decompositions adapts in the intuitive way (see Remark 5.2.5).

Example 6.4.1
- $\{\{p\},\ \{\langle a\rangle\langle(a+b)^*\rangle p\},\ \{\langle b\rangle\langle(a+b)^*\rangle p\}\}$ is a DNF of $\{\langle(a+b)^*\rangle p\}$ (see Example 6.3.2).
- $\{\{[a]\neg p,\ \neg p,\ [b][b^*]\neg p\}\}$ is a DNF of $\{[a+b^*]\neg p\}$.
- $\{\{\langle a\rangle\langle(a+b)^*\rangle p,\ [a]\neg p,\ \neg p,\ [b][b^*]\neg p\},$
  $\{\langle b\rangle\langle(a+b)^*\rangle p,\ [a]\neg p,\ \neg p,\ [b][b^*]\neg p\}\}$
  is a DNF of $\{\langle(a+b)^*\rangle p,\ [a+b^*]\neg p\}$. ◂

As in Chapter 5, we consider only N-clauses, and hence often abbreviate "N-clauses" to "clauses". The **request** of a clause C with respect to an action a is defined as $\mathfrak{R}_a C := \{\, s \mid [a]s \in C \,\}$.

Let C be a clause and $\langle a\rangle s \in C$. An **expansion of** $\langle a\rangle s \mid C$ is defined by case analysis on the shape of s:
- If s is not a diamond formula and $\mathcal{D}$ is a DNF of $\mathfrak{R}_a C\,;s$, then $\{\,(s \mid D) \mid D \in \mathcal{D}\,\}$ is an expansion of $\langle a\rangle s \mid C$.

- If s is a diamond formula and, for every $t \in \mathfrak{D}s$, $\mathcal{D}_t$ is a DNF of $\mathfrak{R}_a C\,;t$, then $\bigcup_{t \in \mathfrak{D}s} \{\, (t \mid D) \mid D \in \mathcal{D}_t \,\}$ is an expansion of $\langle a\rangle s \mid C$.

Example 6.4.2

- $\{\langle a\rangle\langle b\rangle p \mid \{\langle a\rangle\langle b\rangle p,\ q\},\ \langle a\rangle\langle b\rangle p \mid \{\langle a\rangle\langle b\rangle p,\ \neg q\}\}$ is an expansion of $\langle a\rangle\langle ab\rangle p \mid \{\langle a\rangle\langle ab\rangle p,\ [a](q \vee \neg q)\}$.
- $\{\langle a\rangle p \mid \{\langle a\rangle p\},\ p \mid \{p\},\ \langle b\rangle\langle b^*\rangle p \mid \{\langle b\rangle\langle b^*\rangle p\}\}$ is an expansion of $\langle a\rangle\langle a + b^*\rangle p \mid \{\langle a\rangle\langle a + b^*\rangle p\}$.
- $\{\langle a\rangle p \mid \{\langle a\rangle p,\ \neg p\},\ \langle b\rangle\langle b^*\rangle p \mid \{\langle b\rangle\langle b^*\rangle p,\ \neg p\}\}$ is an expansion of $\langle a\rangle\langle a + b^*\rangle p \mid \{\langle a\rangle\langle a + b^*\rangle p,\ [a]\neg p\}$. ◂

Remark 6.4.3 Note that we define an expansion $\mathcal{E}$ of $\langle a\rangle\langle\alpha\rangle s \mid C$ such that it explicitly captures all ways of reaching s via $\xrightarrow{a\alpha}_s$ (since $\mathcal{L}\alpha = \bigcup_{(\langle\beta_1\rangle\ldots\langle\beta_n\rangle s \mid D) \in \mathcal{E}} \mathcal{L}\beta_1 \cdot \ldots \cdot \mathcal{L}\beta_n$). In this sense, expansions are defined to respect the regular semantics of programs, which is not necessarily respected by the purely propositional computation of DNFs (see Remark 5.4.7). ◂

Links are now triples $C\binom{\langle a\rangle s}{t}D$ where C is an N-clause, $\langle a\rangle s \in C$, and $t \mid D$ is contained in some expansion of $\langle a\rangle s \mid C$. We call links of the form $C\binom{\langle a\rangle s}{\langle b\rangle t}D$ **delegating**, and links $C\binom{\langle a\rangle s}{t}D$ where t is not a diamond formula **fulfilling**.

Note that we no longer have a notion of a simple link. This is because the complex syntax of PDL blurs the difference between simple diamonds and eventualities. For K* and H*, we consider diamond literals of the form $\Diamond^+ s$ as eventualities since they can be obtained by beta decomposition from eventualities $\Diamond^* s$. Now, consider the diamond literal $\langle a\rangle\langle b\rangle\langle(ab)^*\rangle p$. The literal cannot be seen as simple according to the intuition for H* since it is contained in $\mathfrak{D}(\langle(ab)^*\rangle p)$. On the other hand, the literal $\langle a\rangle\langle b^*\rangle p$ is not an eventuality since there is no program α such that $\langle a\rangle\langle b^*\rangle p \in \mathfrak{D}(\langle\alpha^*\rangle p)$.

Note also that many links that would be considered fulfilling in the context of H* are now delegating (for instance, the link $C\binom{\langle a\rangle\langle a^*\rangle\langle b\rangle p}{\langle b\rangle p}D$). For H*, the distinction between the two types of links is done such that a link is delegating if it can possibly occur in a loop, while a fulfilling link marks the end of a run, and hence cannot occur in a loop. For HPDL, we need to be more cautious since now links of the form $C\binom{\langle a\rangle\langle a^*\rangle s}{s}D$ can occur in loops. For instance, consider the link $C\binom{\langle a\rangle\langle a^*\rangle\langle b\rangle\langle(a^*b)^*\rangle p}{\langle b\rangle\langle(a^*b)^*\rangle p}D$ where $C = \{\langle a\rangle\langle a^*\rangle\langle b\rangle\langle(a^*b)^*\rangle p\}$ and $D = \{\langle b\rangle\langle(a^*b)^*\rangle p\}$. While the overall shape of the link corresponds to that of a fulfilling link for H* (i.e., $C'\binom{\Diamond^+ s}{s}D'$), the triple $D\binom{\langle b\rangle\langle(a^*b)^*\rangle p}{\langle a\rangle\langle a^*\rangle\langle b\rangle\langle(a^*b)^*\rangle p}C$ is also a link. Taken together, the two links clearly form a loop (in the intuitive sense; the formal definition of loops will be given later). Hence, neither of the two links can be seen as fulfilling.

Given the new definition of links, **branches** are defined the same as for H*, namely as nonempty sets Γ of clauses and links that satisfy the link coherence, functionality and nominal coherence conditions, all defined exactly the same as in § 5.5.1. The mechanism of nominal propagation and the notion of a core are also unchanged compared to § 5.5.1. A branch Γ **realizes** $\langle a\rangle s \mid C$ if $C\binom{\langle a\rangle s}{t}D \in \Gamma$ for some t and D.

A **path for $\langle a\rangle s$ in** Γ is now a nonempty sequence

$$(C_1\binom{\langle a_1\rangle s_1}{\langle a_2\rangle s_2}C_2)(C_2^\Gamma\binom{\langle a_2\rangle s_2}{\langle a_3\rangle s_3}C_3)\dots(C_{n-2}^\Gamma\binom{\langle a_{n-2}\rangle s_{n-2}}{\langle a_{n-1}\rangle s_{n-1}}C_{n-1})(C_{n-1}^\Gamma\binom{\langle a_{n-1}\rangle s_{n-1}}{t}C_n)$$

of links in Γ such that $\langle a_1\rangle s_1 = \langle a\rangle s$. A path of the above form is called a **run for** $\langle a\rangle s \mid C$ if $C_1 = C$ and t is not a diamond formula (i.e., if the last link of the path is fulfilling). The path is called a **loop for** $\langle a\rangle s$ if $C_1 = C_n^\Gamma$ and $t = \langle a_1\rangle s_1 = \langle a\rangle s$. A branch Γ is called **evident** if Γ contains no loops and realizes $\langle a\rangle s \mid C$ for all $\langle s\rangle \in C \in \mathcal{C}\Gamma$.

Example 6.4.4 Consider the following branch:

$$\begin{tikzcd}
\langle a\rangle\langle(a+b)^*\rangle p, \arrow[rr, bend right] & \neg p, & \langle b\rangle\langle(a+b)^*\rangle p, \arrow[d] & {[a][a^*](\neg p \wedge \langle a\rangle\langle(a+b)^*\rangle p)} \\
& & p &
\end{tikzcd}$$

Note that the link departing from the formula $\langle a\rangle\langle(a+b)^*\rangle p$ in the upper clause does not form a loop despite pointing to the same clause from which it departs since it points to a different formula. Therefore, the branch is evident since every claim is realized with a link. On the other hand, the branch

$$\begin{tikzcd}
\langle a\rangle\langle(a+b)^*\rangle p, \arrow[d] & \neg p, & {[a][(a+b)^*]\neg p,} & {[b][(a+b)^*]\neg p} \\
\langle b\rangle\langle(a+b)^*\rangle p, \arrow[u, bend left=90] & \neg p, & {[a][(a+b)^*]\neg p,} & {[b][(a+b)^*]\neg p}
\end{tikzcd}$$

contains a loop. ◂

As for K* and H* (see Propositions 5.4.2 and 5.5.5, respectively), an evident branch has a unique run for every claim:

Proposition 6.4.5 Let Γ be an evident branch and $\langle a\rangle s \in C \in \mathcal{C}\Gamma$. Then there is a unique run for $\langle a\rangle s \mid C$ in Γ.

Proof Let Γ be evident and $\langle a\rangle s \in C \in \mathcal{C}\Gamma$. Both the uniqueness and the existence of runs are shown analogously to the corresponding claims in Proposition 5.5.5. For the uniqueness proof, we observe that every two distinct runs for the same claim must contain, respectively, two distinct links $D\binom{\langle b\rangle t}{u}E$ and $D\binom{\langle b\rangle t}{u'}E'$, which is impossible since Γ is functional (see the proof of Proposition 5.4.2).

For the existence proof, assume $\langle a\rangle s \mid C$ has no run in Γ. Then, since every diamond in every clause in $\mathcal{C}\Gamma$ is realized, Γ contains an infinite sequence $\pi = (C_1\binom{\langle a_1\rangle s_1}{\langle a_2\rangle s_2}C_2)$ $(C_2\binom{\langle a_2\rangle s_2}{\langle a_3\rangle s_3}C_3)\dots$ where $C_1 = C$ and $\langle a_1\rangle s_1 = \langle a\rangle s$. Since Γ is finite, there are some $j > i \geq 1$ such that $C_i = C_j^\Gamma$, i.e., π contains a loop. Contradiction to Γ being evident. ∎

6.5 Demos

We now show that every evident branch is satisfiable. As usual, we proceed by showing that every evident branch describes a demo.

Unlike in Chapters 4 and 5, we now bypass clausal demos and build directly on Hintikka demos. We do so because, unlike for K* and H*, the definition of clausal demos for HPDL becomes very similar to that of Hintikka demos. One introduces a clausal version of the relation $\xrightarrow{\alpha}_S$ for complex programs α and defines a clause C to be $\Diamond$-admissible (in a clause set S) if for every $\langle\alpha\rangle s$ supported by C there is some $D \in S$ such that $C \xrightarrow{\alpha}_S D$ and $D \rhd s$. Because there is only little difference between clausal demos and Hintikka demos, the overhead of using clausal demos as an intermediate representation outweighs their usefulness in modularizing the proofs.

Unlike the construction in Chapter 3, now our notion of demos will be based on the terminating decomposition table. We do this to facilitate the construction of demos from the clausal representation given by evident branches, which was not necessary in Chapter 3.

The definition of the syntactic transition relations $\xrightarrow{\alpha}_S$ (Definition 3.3.1) remains unchanged. **Demos** are now defined as nonempty Hintikka systems that are $\Diamond$-admissible (where $\Diamond$-admissibility is defined as in § 3.3.1) and nominally admissible (nominal admissibility defined as in § 3.5.1).

The proof of demo existence adapts in a straightforward way. First, observe that the set $H_{\mathfrak{M},w} = \{ s \in \mathcal{F} \mid \mathfrak{M}, w \models s \}$ is still a Hintikka set for every model $\mathfrak{M}$ and every $w \in |\mathfrak{M}|$, which follows by the semantic equivalences for diamond and box decompositions (which, in turn, follow by Corollary 6.3.8). The proofs of Lemma 3.3.12 and Theorem 3.3.13 remain unchanged.

Showing demo satisfaction requires some adaptations, which we detail in the following section.

6.5.1 Demo Satisfaction

Recall that every nonempty nominally admissible Hintikka system S induces a model $\mathfrak{M}_S$ as defined in § 3.3.2. Since the definition of $\mathfrak{M}_S$ does not depend on the decomposition table underlying the Hintikka system and the definition of $\xrightarrow{\alpha}_S$ is unchanged, the validity of Lemma 3.3.6 is unaffected.

The proof of Lemma 3.3.7, however, now requires more work. The reason for this is that now, decomposition of boxes no longer corresponds to the equations in the definitions of $\xrightarrow{\alpha}_S$ and $\xrightarrow{\alpha}_{\mathfrak{M}}$. To relate the modified Hintikka conditions with the two relations, we resort once again to the language-theoretic view of programs from § 6.2.

First, we restate Lemma 3.3.7 as follows.

Lemma 6.5.1 Let S be a Hintikka system, $\{H, H'\} \subseteq S$, $[\alpha_1]\dots[\alpha_n]s \in H$ $(n \geq 0)$, and $H \xrightarrow{\sigma}_{\mathfrak{M}_S} H'$ for some $\sigma \in \mathcal{L}\alpha_1 \cdot \ldots \cdot \mathcal{L}\alpha_n$. Then $s \in H'$.

Proof We proceed by lexicographic induction on the length of σ and the decomposition order of $[\alpha_1]\dots[\alpha_n]s$. If $n = 0$, the claim is trivial. Let $n \geq 1$. Depending on the shape of α_1, we distinguish two cases.

If $\alpha_1 = a$, we have $\sigma = a\tau$. Hence, there is some set $H'' \in S$ such that $H \xrightarrow{a}_{\mathfrak{W}_S} H''$ and $H'' \xrightarrow{\tau}_{\mathfrak{W}_S} H'$. By the definition of $\xrightarrow{a}_{\mathfrak{W}_S}$, we have $[\alpha_2]\dots[\alpha_n]s \in H''$. Moreover, $\tau \in \mathcal{L}\alpha_2 \cdot \dots \cdot \mathcal{L}\alpha_n$. The claim follows by the inductive hypothesis for τ and $[\alpha_2]\dots[\alpha_n]s$.

Otherwise, we have $t \in H$ for all $t \in \mathfrak{D}([\alpha_1]\dots[\alpha_n]s)$. By condition (2) for box decompositions, there is some $[\beta_1]\dots[\beta_m]s \in \mathfrak{D}([\alpha_1]\dots[\alpha_n]s)$ such that $\sigma \in \mathcal{L}\beta_1 \cdot \dots \cdot \mathcal{L}\beta_m$. The claim follows by the inductive hypothesis for σ and $[\beta_1]\dots[\beta_m]s$ ($[\beta_1]\dots[\beta_m]s$ is a constituent of $[\alpha_1]\dots[\alpha_n]s$ and is hence smaller according to the decomposition order). ∎

From this, we conclude the original formulation of Lemma 3.3.7:

Lemma 6.5.2 Let S be a Hintikka system, $\{H, H'\} \subseteq S$, and $[\alpha]s \in H \xrightarrow{\alpha}_{\mathfrak{W}_S} H'$. Then $s \in H'$.

Proof By Proposition 6.2.3 (1), we have $H \xrightarrow{\sigma}_{\mathfrak{W}_S} H'$ for some $\sigma \in \mathcal{L}\alpha$. The claim follows with Lemma 6.5.1. ∎

With Lemma 6.5.2 in place, the proof of Lemma 3.3.8 proceeds the same as before. Theorem 3.3.9 follows.

6.5.2 Satisfaction of Evident Branches

Let C be an N-clause. Since the terminating decomposition table is admissible, the set $H = \{\, s \in \mathcal{F} \mid C \rhd s \,\}$ is the largest Hintikka set over $\mathcal{F}$ such that $\Lambda H = C$. We prove that every evident branch Γ is satisfiable by showing that the Hintikka system obtained from the clauses in the core of an evident branch is a demo.

For this, we need an equivalent of Proposition 6.2.3 (1) for the syntactic transition relations $\xrightarrow{\alpha}_S$. Given a Hintikka system S, we define the relations $\xrightarrow{\sigma}_S \subseteq S \times S$ by induction on the structure of σ:

$$H \xrightarrow{\varepsilon}_S H' \iff H = H'$$
$$H \xrightarrow{a\sigma}_S H' \iff \exists H''\colon\ H \xrightarrow{a}_S H'' \text{ and } H'' \xrightarrow{\sigma}_S H'$$

Proposition 6.2.3 (1) can then be adapted as follows.

Proposition 6.5.3 $H \xrightarrow{\alpha}_S H' \iff \exists \sigma \in \mathcal{L}\alpha\colon\ H \xrightarrow{\sigma}_S H'$

The adaptation of the proof is straightforward (see § 7.1 for a related proof).

Recall that by Proposition 5.5.2, whose validity does not depend on the definition of the decomposition table, implies that, for every clause $C \in \Gamma$, we have $C^\Gamma \in C\Gamma$. Also, Proposition 5.5.6 still holds for the new decomposition table.

We call a Hintikka system S $\Diamond$-**preadmissible** if for every $H \in S$, every $n \geq 0$ and every $\langle a\rangle\langle\alpha_1\rangle\ldots\langle\alpha_n\rangle s \in H$ there is some $H' \in S$ such that $H \xrightarrow{a\alpha_1\ldots\alpha_n}_S H'$ and $s \in H'$. We view $\Diamond$-preadmissibility as auxiliary in proving $\Diamond$-admissibility.

Recall the notations $H_C := \{\, s \in \mathcal{F} \mid C \rhd s \,\}$ and $\mathcal{H}_S := \{\, H_C \mid C \in S \,\}$ introduced in § 4.1.2.

Lemma 6.5.4 If Γ is evident, then $\mathcal{H}_{C\Gamma}$ is $\Diamond$-preadmissible.

Proof Let Γ be evident and $\langle a\rangle\langle\alpha_1\rangle\ldots\langle\alpha_n\rangle s \in C \in \mathcal{H}_{C\Gamma}$. By Proposition 6.4.5, the claim $\langle a\rangle\langle\alpha_1\rangle\ldots\langle\alpha_n\rangle s \,|\, C$ has a run $\pi = (C\Big(\begin{smallmatrix}\langle a\rangle\langle\alpha_1\rangle\ldots\langle\alpha_n\rangle s\\ t\end{smallmatrix}\Big)D)\pi'$ in Γ. We proceed by induction on the length of π. By Propositions 5.2.3 and 5.5.6, $D^\Gamma \rhd \mathfrak{R}_a C\,; t$, and hence $H_C \xrightarrow{a}_{\mathcal{H}_{C\Gamma}} H_{D^\Gamma}$. We distinguish two cases.

If $n = 0$, it suffices to show that $D^\Gamma \rhd s$. This is trivial if $t = s$ (since $D^\Gamma \rhd t$), and otherwise follows from $D^\Gamma \rhd t$ and $t \in \mathfrak{D}s$ ($t \neq s$ means s is a diamond formula) by the definition of the upward closure.

If $n \geq 1$, we have $t \in \mathfrak{D}(\langle\alpha_1\rangle\ldots\langle\alpha_n\rangle s)$, and hence $t = \langle\beta_1\rangle\ldots\langle\beta_m\rangle s$ (Lemma 6.3.4). If $m = 0$, the claim follows since $D^\Gamma \xrightarrow{\alpha_1\ldots\alpha_n}_{\mathcal{H}_{C\Gamma}} D^\Gamma$ (follows with condition (2) for diamond decompositions and Proposition 6.5.3), $t = s$ and $D^\Gamma \rhd t$. So, let $m \geq 1$. Then β_1 is a literal and π' is a run for $\langle\beta_1\rangle\ldots\langle\beta_m\rangle s \,|\, D^\Gamma$. Hence, by the inductive hypothesis for π', there is some $H \in \mathcal{H}_{C\Gamma}$ such that $H_{D^\Gamma} \xrightarrow{\beta_1\ldots\beta_m}_{\mathcal{H}_{C\Gamma}} H$ and $s \in H$. By condition (2) for diamond decompositions and Proposition 6.5.3, $H_{D^\Gamma} \xrightarrow{\alpha_1\ldots\alpha_n}_{\mathcal{H}_{C\Gamma}} H$. Hence $H_C \xrightarrow{a\alpha_1\ldots\alpha_n}_{\mathcal{H}_{C\Gamma}} H$, and we are done. ∎

By the following lemma, a Hintikka system is $\Diamond$-preadmissible if and only if it is $\Diamond$-admissible. Note that the implication from left to right (which is also the important direction for us) holds only because our present decomposition table is terminating.

Lemma 6.5.5 A Hintikka system is $\Diamond$-preadmissible if and only if it is $\Diamond$-admissible.

Proof The direction from right to left is straightforward. Given a diamond formula $\langle a\rangle\langle\alpha_1\rangle\ldots\langle\alpha_n\rangle s \in H \in S$, the existence of some $H' \in S$ such that $H \xrightarrow{a\alpha_1\ldots\alpha_n}_S H'$ and $s \in H'$ follows by $n+1$ applications of $\Diamond$-admissibility and the definition of $\xrightarrow{\alpha}_S$.

As for the other direction, let S be a $\Diamond$-preadmissible Hintikka system and $\langle\alpha\rangle s \in H \in S$. We do a case analysis on the shape of α. If α is an action, the claim follows since S is $\Diamond$-preadmissible. Otherwise, we have $t \in H$ for some $t \in \mathfrak{D}(\langle\alpha\rangle s)$, where $t = \langle\beta_1\rangle\ldots\langle\beta_n\rangle s$ for some $n \geq 0$ and $\beta_1, \ldots, \beta_n$ (Lemma 6.3.4).

If $n = 0$, then $\varepsilon \in \mathcal{L}\alpha$ (condition (2) for diamond decompositions), and hence the claim follows by Proposition 6.5.3.

If $n \geq 1$, then $\beta_1 = b$ (for some action b). Since S is $\Diamond$-preadmissible, there is some $H' \in S$ such that $H \xrightarrow{b\beta_2\ldots\beta_n}_S H'$ and $s \in H'$. The claim follows by condition (2) for diamond decompositions and Proposition 6.5.3. ∎

Theorem 6.5.6 The core of an evident branch can be extended to a demo.

$$(\Diamond)\ \frac{\Gamma}{\Gamma; D_1^\Gamma; C\binom{\langle a\rangle s}{t_1} D_1 \mid \ldots \mid \Gamma; D_n^\Gamma; C\binom{\langle a\rangle s}{t_n} D_n} \quad \begin{array}{l}\langle a\rangle s \in C \in \mathcal{C}\Gamma,\\ \Gamma \text{ does not realize } \langle a\rangle s \mid C,\\ \{t_1 \mid D_1, \ldots, t_n \mid D_n\} \text{ nc-expansion}\\ \qquad\qquad \text{of } \langle a\rangle s \mid C \text{ in } \Gamma\end{array}$$

$$(\textsc{loop})\ \frac{\Gamma}{\otimes}\ \ \Gamma \text{ contains a loop}$$

Figure 6.2: Expansion rules for (test-free) HPDL

Proof Let Γ be an evident branch. by Lemmas 6.5.4 and 6.5.5, the Hintikka system $\mathcal{H}_{C\Gamma}$ is $\Diamond$-admissible. Moreover, since $\mathcal{H}_{C\Gamma}$ contains only clauses from $C\Gamma$, every nominal $x \in \mathcal{F}$ is contained in at most one $C \in \mathcal{H}_{C\Gamma}$ (Lemma 5.5.3). To obtain nominal admissibility, it suffices to add to $\mathcal{H}_{C\Gamma}$ all nominals in $\mathcal{F}$ that do not occur in any clause of $\mathcal{H}_{C\Gamma}$. Let $\mathcal{D} = \mathcal{H}_{C\Gamma} \cup \{\{x\} \mid x \in \mathcal{F},\ \nexists C \in \mathcal{H}_{C\Gamma}\colon\ x \in C\}$. Clearly, $\mathcal{D}$ is nominally admissible. Since the clauses in $\mathcal{D} \setminus \mathcal{H}_{C\Gamma}$ contain no diamond formulas, $\mathcal{D}$ is still $\Diamond$-admissible. Hence, $\mathcal{D}$ is a demo. ∎

Corollary 6.5.7 The clauses of an evident branch are satisfiable.

6.6 Expansion Rules

Let nc-DNFs be defined from ordinary DNFs as described in § 5.5.2, and nc-expansions be defined from nc-DNFs analogously to how expansions are defined from DNFs (compare the definition of expansions in § 5.4.2 to that of nc-expansions in § 5.5.2). Proposition 5.5.9 still holds for HPDL and our modified decomposition table (the proof remains unchanged).

Figure 6.2 shows the expansion rules for test-free HPDL. Clearly, a branch is evident if and only if it is maximal. Hence, the rules are demonstration-sound by Corollary 6.5.7. Just as for K (Theorem 5.3.5) and H*, the termination of the procedure induced by the rules is immediate since the number of clauses and links is still exponentially bounded in $|\mathcal{F}|$.

Example 6.6.1 Figure 6.3 shows a derivation of an evident branch. As in Example 5.5.10, the numbers indicate the order in which the links and clauses are introduced, clause 0 being the initial clause. Note that the construction of clauses 1 and 2 involves nominal propagation. Once again, the dashed link indicates the implicit redirection of a link (the redirection of link 1 that occurs when clause 2 is added).

Since clauses 0 and 1 are not in the core of the final branch, the only relevant runs are the one formed by link 4 for $\langle b\rangle\langle(a+b)^*\rangle p$ in clause 2 and the one for $\langle a\rangle\langle(a+b)^*\rangle p$ in clause 2 (consisting of the links 3, 4). ◂

To prove that the rules induce a decision procedure for test-free HPDL, it remains to show that they are refutation-sound. For this, we adapt our approach from Chapter 5.

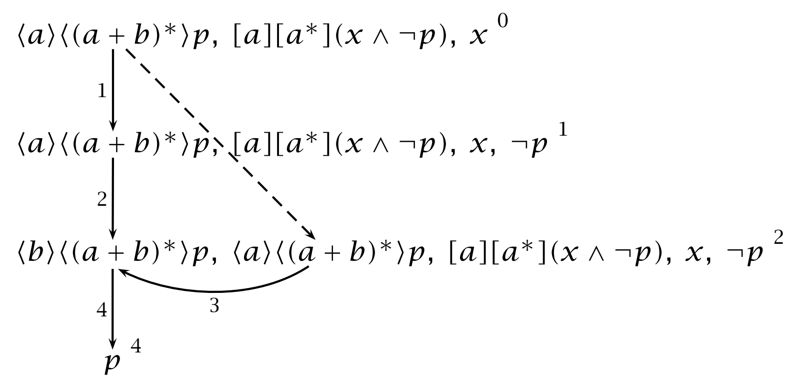

$\langle a\rangle\langle(a+b)^*\rangle p,\ [a][a^*](x\wedge\neg p),\ x$ 0
1
$\langle a\rangle\langle(a+b)^*\rangle p,\ [a][a^*](x\wedge\neg p),\ x,\ \neg p$ 1
2
$\langle b\rangle\langle(a+b)^*\rangle p,\ \langle a\rangle\langle(a+b)^*\rangle p,\ [a][a^*](x\wedge\neg p),\ x,\ \neg p$ 2
4
3
p 4

Figure 6.3: Derivation of an evident branch

We begin with the definition of δ, which we adapt to match our new definition of fulfilling and delegating links. Let $\langle\alpha_1\rangle\dots\langle\alpha_n\rangle s$ be a formula where $n \geq 0$ and s is not a diamond formula. We define the **depth of** A **with respect to** $\langle\alpha_1\rangle\dots\langle\alpha_n\rangle s$ **in** $\mathfrak{M}$ as follows:

$$\begin{aligned}\delta_{\mathfrak{M}}A(\langle\alpha_1\rangle\dots\langle\alpha_n\rangle s) \;:=\; \min\{\,|\sigma| \mid \sigma \in \mathcal{L}\alpha_1\cdot\ldots\cdot\mathcal{L}\alpha_n \text{ and}\\ \exists v,w\colon\ v \xrightarrow{\sigma}_{\mathfrak{M}} w \text{ and } \mathfrak{M},v\vDash A \text{ and } \mathfrak{M},w\vDash s\,\}\end{aligned}$$

where we write $|\sigma|$ for the length of σ and assume $\min\emptyset = \infty$ and $n < \infty$ for all $n \in \mathbb{N}$.

Proposition 6.6.2 $\delta_{\mathfrak{M}}As < \infty$ if and only if $\mathfrak{M}$ satisfies $A\,;s$.

Proof If s is not a diamond formula, the claim is immediate by the definition of δ. Otherwise, we have $s = \langle\alpha_1\rangle\dots\langle\alpha_n\rangle t$ where $n \geq 1$, and the claim follows by n applications of Proposition 6.2.3 (2) (and $n-1$ applications of Lemma 6.2.2). ■

Using δ, we extend the notion of satisfaction to links (now including fulfilling links) as follows. A model $\mathfrak{M}$ **satisfies a link** $C\binom{s}{t}D$ if $\delta_{\mathfrak{M}}Cs > \delta_{\mathfrak{M}}Dt$. A model **satisfies a branch** if it satisfies all of its clauses and links.

Remark 6.6.3 Note that the satisfaction condition for links is now simpler than the corresponding condition in Chapter 5. For every link $C\binom{s}{t}D$, we have $\delta_{\mathfrak{M}}Cs > 0$ since s is always a diamond literal of the form $\langle a\rangle\langle\alpha_1\rangle\dots\langle\alpha_n\rangle s'$ and $\mathcal{L}a\cdot\mathcal{L}\alpha_1\cdot\ldots\cdot\mathcal{L}\alpha_n$ does not contain ε (it contains only words beginning with a). For the same reason, we have $\delta_{\mathfrak{M}}Dt > 0$ whenever $C\binom{s}{t}D$ is delegating. If $C\binom{s}{t}D$ is fulfilling, we have $\delta_{\mathfrak{M}}Cs > 0 = \delta_{\mathfrak{M}}Dt$ whenever $\mathfrak{M}$ satisfies $D\,;t$. Hence, a fulfilling link is satisfied by $\mathfrak{M}$ whenever so are both of its clauses. Although technically not necessary, the extension of satisfaction to fulfilling links makes the definition more uniform.

To transfer these simplifications back to H*, however, we would have to complicate the definition of δ in Chapter 5, which would make the overall setup less intuitive. ◂

Note that the validity of Proposition 5.5.1 is unaffected by going from H* to HPDL. Hence, Proposition 5.5.11 holds in its original formulation for the new definition of δ. With this, we reprove Proposition 5.5.12 as follows.

Proposition 6.6.4 Satisfiable branches contain no loops.

Proof Let Γ be a branch and $\mathfrak{M}$ a model of Γ. Assume, for contradiction, Γ has a loop $(C_1\binom{s_1}{s_2}C_2)\dots(C^\Gamma_{n-1}\binom{s_{n-1}}{s_n}C_n)$. Then $s_1 = s_n$, $C_1 = C^\Gamma_1 = C^\Gamma_n$ and $n \geq 2$. With Proposition 5.5.11, we obtain $\delta_{\mathfrak{M}}C_1s_1 > \delta_{\mathfrak{M}}C_2s_2 = \delta_{\mathfrak{M}}C^\Gamma_2s_2 > \cdots > \delta_{\mathfrak{M}}C_ns_n = \delta_{\mathfrak{M}}C^\Gamma_ns_n = \delta_{\mathfrak{M}}C_1s_1$. Contradiction. ∎

Proposition 6.6.4 justifies the refutation-soundness of the loop rule. It remains to argue the refutation-soundness of the diamond rule.

Lemma 6.6.5 Let Γ be a branch, $\langle a\rangle s \in C \in \Gamma$, and $\mathcal{E}$ be an nc-expansion of $\langle a\rangle s \mid C$ in Γ. Let $\mathfrak{M}$ be a model of all clauses in Γ. Then there is some $t \mid D \in \mathcal{E}$ such that $\delta_{\mathfrak{M}}Dt < \delta_{\mathfrak{M}}C(\langle a\rangle s)$.

Proof Let Γ, $\langle a\rangle s \in C \in \Gamma$, $\mathcal{E}$, and $\mathfrak{M}$ be as required. If s is not a diamond formula, let $\mathcal{D}$ be an nc-DNF of $\mathfrak{R}_aC;s$ such that $\mathcal{E} = \{\,(s \mid D) \mid D \in \mathcal{D}\,\}$. The claim follows since, by Proposition 5.5.9, $\mathfrak{M}$ satisfies some clause $D \in \mathcal{D}$, and hence $\delta_{\mathfrak{M}}Ds = 0 < 1 = \delta_{\mathfrak{M}}C(\langle a\rangle s)$.

If s is a diamond formula, we have $s = \langle\alpha_1\rangle\dots\langle\alpha_n\rangle t$ for some $n \geq 1$. By Proposition 6.6.2, $\delta_{\mathfrak{M}}C(\langle a\rangle s) < \infty$. Let $v, w \in |\mathfrak{M}|$ and $\sigma \in \mathcal{L}\alpha_1 \cdot \dots \mathcal{L}\alpha_n$ be such that $v \xrightarrow{a\sigma}_{\mathfrak{M}} w$, $\mathfrak{M}, v \models C$, $\mathfrak{M}, w \models t$, and $|\sigma| = \delta_{\mathfrak{M}}C(\langle a\rangle s) - 1$. Let $u \in |\mathfrak{M}|$ be such that $v \xrightarrow{a}_{\mathfrak{M}} u$ and $u \xrightarrow{\sigma}_{\mathfrak{M}} w$. By condition (2) for diamond decompositions, there is some $\langle\beta_1\rangle\dots\langle\beta_m\rangle t \in \mathbb{D}s$ such that $\sigma \in \mathcal{L}\beta_1 \cdot \dots \cdot \mathcal{L}\beta_m$. Since $\mathfrak{M}, u \models \mathfrak{R}_aC$, we have $\delta_{\mathfrak{M}}(\mathfrak{R}_aC)(\langle\beta_1\rangle\dots\langle\beta_m\rangle t) \leq |\sigma|$, and hence $\mathfrak{M}, u \models \mathfrak{R}_aC;\langle\beta_1\rangle\dots\langle\beta_m\rangle t$ (Proposition 6.6.2). Let $\mathcal{D}$ be an nc-DNF of $\mathfrak{R}_aC;\langle\beta_1\rangle\dots\langle\beta_m\rangle t$ such that $\{\,(\langle\beta_1\rangle\dots\langle\beta_m\rangle t \mid D) \mid D \in \mathcal{D}\,\} \subseteq \mathcal{E}$. By Proposition 5.5.9, there is some $D \in \mathcal{D}$ such that $\mathfrak{M}, u \models D$. Hence, $\langle\beta_1\rangle\dots\langle\beta_m\rangle t \mid D \in \mathcal{E}$ and $\delta_{\mathfrak{M}}D(\langle\beta_1\rangle\dots\langle\beta_m\rangle t) \leq |\sigma| = \delta_{\mathfrak{M}}C(\langle a\rangle s) - 1$. ∎

Theorem 6.6.6 (Refutation-Soundness of ($\Diamond$)) Let Γ be a branch that does not realize $\langle a\rangle s \mid C$ for some $\langle a\rangle s \in C \in \Gamma$. Let $\mathfrak{M}$ be a model of Γ, and $\mathcal{E}$ be an nc-expansion of $\langle a\rangle s \mid C$ in Γ. Then there is some $t \mid D \in \mathcal{E}$ such that $\Gamma;D^\Gamma;C\binom{\langle a\rangle s}{t}D$ is a branch satisfied by $\mathfrak{M}$.

Proof Let Γ, $\langle a\rangle s \in C \in \Gamma$, $\mathfrak{M}$ and $\mathcal{E}$ be as required. By Lemma 6.6.5, there is some $t \mid D \in \mathcal{E}$ such that $\mathfrak{M}$ satisfies $C\binom{\langle a\rangle s}{t}D$. Let $\Delta = \Gamma;D^\Gamma;C\binom{\langle a\rangle s}{s}D$.

We first show that Δ is a branch. Since Γ does not realize $\langle a\rangle s \mid C$, Δ is functional. Since Γ is nominally coherent, we have $(D^\Gamma)^\Delta = (D^\Gamma)^\Gamma = D^\Gamma$ (Proposition 5.5.2), and hence Δ is nominally coherent.

Since $\mathfrak{M}$ satisfies Γ (by assumption) and $C\binom{\langle a\rangle s}{t}D$ (by construction), it remains to show that $\mathfrak{M}$ satisfies D^Γ. This follows from the satisfaction of $C\binom{\langle a\rangle s}{t}D$ with Propositions 6.6.2 and 5.5.1. ∎

6.7 Related Work

► Unlike in previous work on PDL by Pratt [161] and Widmann [200], we introduce a language-theoretic semantics of programs and exploit it in our correctness proofs. The main motivation for introducing a language-theoretic semantics is the need to capture the inductive nature of diamond formulas, which becomes nontrivial when the naive decomposition table is not terminating. Both Pratt and Widmann are faced with the same problem and solve it in different ways.

Pratt introduces a nonstandard semantics for formulas that explicitly describes the inductive nature of diamond satisfaction. He then shows that his nonstandard semantics is equivalent to the standard semantics of PDL with respect to satisfiability. Thus, he only needs to show the correctness of his procedure with respect to the nonstandard semantics.

Widmann faces the problem by strengthening the notion of satisfiability to what he calls *annotated-satisfiability* (Definition 4.26 in [200]). Annotated-satisfiability requires, besides satisfiability in the usual sense, the existence of specific sequences of state-formula pairs satisfying, for every eventuality s, a given chain of alpha-beta decompositions starting at s. While the technical realization differs from Pratt's approach, the idea is still the same: to introduce a well-founded characterization of diamond satisfaction.

Having a language-theoretic semantics allows us to prove the correctness of the terminating decomposition table without modifying the usual model-theoretic semantics of formulas.

► Diamond and box decompositions developed in this chapter are closely related to the notion of *derivatives* introduced by Brzozowski [35]. In particular, it is easily seen that conditions (1.b) and (2) of diamond and box decompositions directly correspond to Theorem 4.4 in [35]. At the same time, our algorithm for computing diamond and box decompositions is not based on Brzozowski's construction, instead building directly on the language-theoretic semantics of programs. Work by Berry and Sethi [24] suggests that computing diamond and box decompositions for test-free (H)PDL can be efficiently realized via translation into deterministic finite automata. We expect that similar constructions are also possible for the case with tests.

7 Tree Search for Hybrid PDL

What happens if we add tests? While extending our approach to tests does not require new fundamental insights, it causes considerable technical complications, which is what we deal with in this chapter.

In the presence of tests, regular languages no longer suffice as a semantics for programs. Therefore, we introduce a more general language-theoretic semantics, which is a variant of the language-theoretic semantics of Kleene algebras with tests [137].

As it comes to diamond and box decompositions, the main complication that arises with tests is that a diamond or a box formula can no longer be understood as a pure beta or alpha formula with respect to the corresponding decomposition. Instead, a diamond formula decomposes into a disjunctively interpreted set of conjunctions, while a box formula decomposes into a conjunctively interpreted set of disjunctions. To represent such complex decompositions, we introduce *guarded formulas*, which can be seen as sets of formulas with a distinguished element. We redefine the alpha-beta closure used in Chapter 6 to compute diamond and box decompositions as a closure on guarded formulas. Expansions are then redefined to use the modified diamond and box decompositions.

Once the modified definition of expansions is in place, the formulation of the expansion rules is unchanged from Chapter 6. The corresponding correctness proofs require only minor adaptations.

Structure of the chapter. We will follow the structure of Chapter 6, detailing the changes necessary to deal with tests. We begin by presenting a language-theoretic semantics for programs with tests (§ 7.1). After this, we devise a decomposition table for full HPDL and use the new language-theoretic semantics to argue its correctness (§ 7.2). We then outline the changes to the definitions of expansions, branches and evidence (§ 7.3), as well as extensions to the notion of a demo (§ 7.4) necessary to deal with tests. Finally, we present the expansion rules for full HPDL and discuss the changes in the correctness arguments for the decision procedure (§ 7.5).

7.1 Language-Theoretic Semantics

The language-theoretic semantics of regular expressions, which we crucially exploit in our proofs for test-free HPDL, does not account for tests. To deal with tests, we define a more general semantics, which is an adaptation of the language-theoretic model of Kleene algebras with tests [137] (initially introduced independently of Kleene algebras with tests by Kaplan [130]).

A **guarded string** is a finite sequence $Aa_1A_1 \dots a_nA_n$ where $n \geq 0$ and $A, A_1, \dots, A_n$ are sets of formulas. The letters σ and τ range over guarded strings. The **length** $|\sigma|$ **of a guarded string** $\sigma = Aa_1A_1 \dots a_nA_n$ is n. A **language** is a set of guarded strings. Let L and L' be languages and A a set of formulas. We define:

$$\begin{aligned} \mathcal{L}A &:= \{\, B \mid A \subseteq B \subseteq \mathcal{F} \,\} \\ L \cdot L' &:= \{\, \omega A \omega' \mid \omega A \in L,\ A\omega' \in L' \,\} \\ L^0 &:= \mathcal{L}\emptyset \end{aligned}$$

where ω and ω' range over partial, possibly empty guarded strings. The notations L^{n+1} and L^* are then defined from L^0 and $L \cdot L'$ as in § 6.2. To every program α, we assign a **language** $\mathcal{L}\alpha$:

$$\begin{aligned} \mathcal{L}a &:= \{\, AaB \mid A, B \subseteq \mathcal{F} \,\} & \mathcal{L}(\alpha + \beta) &:= \mathcal{L}\alpha \cup \mathcal{L}\beta \\ \mathcal{L}s &:= \mathcal{L}\{s\} & \mathcal{L}(\alpha\beta) &:= \mathcal{L}\alpha \cdot \mathcal{L}\beta \\ & & \mathcal{L}(\alpha^*) &:= (\mathcal{L}\alpha)^* \end{aligned}$$

Example 7.1.1
- $\mathcal{L}(pa + \neg pb) = \{\, AaA',\ BbA' \mid p \in A,\ \neg p \in B \,\}$
- $\mathcal{L}((pa)^*\neg p) = \{\, A_0aA_1 \dots aA_n \mid n \geq 0,\ p \in A_0 \cap \dots \cap A_{n-1},\ \neg p \in A_n \,\}$ ◂

Proposition 7.1.2 Let L_1, L_2, L_3 be languages. Then:
1. $(L_1 \cup L_2) \cdot L_3 = (L_1 \cdot L_3) \cup (L_2 \cdot L_3)$
2. $(L_1 \cdot L_2) \cdot L_3 = L_1 \cdot (L_2 \cdot L_3)$
3. $L_1 \cdot \mathcal{L}\emptyset = \mathcal{L}\emptyset \cdot L_1 = L_1$
4. $L_1^* = \mathcal{L}\emptyset \cup (L_1 \cdot L_1^*)$

Proof The proofs are straightforward. For instance, consider the inclusion from left to right of (1). Let $\sigma \in (L_1 \cup L_2) \cdot L_3$. Then $\sigma = \omega A \omega'$ such that $\omega A \in L_1 \cup L_2$ and $A\omega' \in L_3$. Then $\omega A \in L_1$ or $\omega A \in L_2$. Consequently, $\sigma \in L_1 \cdot L_3$ or $\sigma \in L_2 \cdot L_3$. The claim follows. ∎

Lemma 7.1.3 If $\omega A \omega' \in \mathcal{L}\alpha$ and $A \subseteq B$, then $\omega B \omega' \in \mathcal{L}\alpha$.

Proof Let α be a program, $\omega A \omega' \in \mathcal{L}\alpha$, and $A \subseteq B$. We show $\omega B \omega' \in \mathcal{L}\alpha$ by induction on α.

Let $\alpha = a$. Then $\omega A \omega' = A_1aA_2$ where $A = A_1$ or $A = A_2$. The claim follows since $\{BaA_2,\ A_1aB\} \subseteq \mathcal{L}a$.

Let $\alpha = s$. Then ω and ω' are empty and $s \in A$. The claim follows since $A \subseteq B$.

Let $\alpha = \beta + \gamma$. Then $\omega A \omega' \in \mathcal{L}\beta$ or $\omega A \omega' \in \mathcal{L}\gamma$. By the inductive hypothesis for β or γ, we have $\omega B \omega' \in \mathcal{L}\beta$ or $\omega B \omega' \in \mathcal{L}\gamma$. The claim follows.

Let $\alpha = \beta\gamma$. Then either $\omega = \omega_1 A_1 \omega_2$ such that $\omega_1 A_1 \in \mathcal{L}\beta$ and $A_1\omega_2 A \omega' \in \mathcal{L}\gamma$ or $\omega' = \omega_1' A_2 \omega_2'$ such that $\omega A \omega_1' A_2 \in \mathcal{L}\beta$ and $A_2\omega_2' \in \mathcal{L}\gamma$. Let us consider the first case (the second case is shown analogously). There, by the inductive hypothesis for γ, we obtain $A_1\omega_2 B \omega' \in \mathcal{L}\gamma$. Hence, $\omega_1 A_1 \omega_2 B \omega' \in \mathcal{L}(\beta\gamma)$. The claim follows.

Let $\alpha = \beta^*$. Then, for some $n \geq 0$, $\omega A \omega' \in (\mathcal{L}\beta)^n$. If $n = 0$, then ω and ω' are empty and the claim holds since $B \in 2^{\mathcal{F}} = \mathcal{L}\emptyset = (\mathcal{L}\beta)^0$. If $n > 0$, then $\omega = \omega_1 A_1 \omega_2$ and $\omega' = \omega_1' A_2 \omega_2'$ such that $A_1 \omega_2 A \omega_1' A_2 \in \mathcal{L}\beta$ and, for some $m \in [0, n-1]$, $\omega_1 A_1 \in (\mathcal{L}\beta)^m$ and $A_2 \omega_2' \in (\mathcal{L}\beta)^{n-1-m}$. By the inductive hypothesis for β, we have $A_1 \omega_2 B \omega_1' A_2 \in \mathcal{L}\beta$. Hence, $\omega_1 A_1 \omega_2 B \omega_1' A_2 \omega_2' \in (\mathcal{L}\beta)^n$. The claim follows. ∎

Given a model $\mathfrak{M}$, we define the relations $\xrightarrow{\sigma}_{\mathfrak{M}} \subseteq |\mathfrak{M}| \times |\mathfrak{M}|$ by induction of σ:

$$v \xrightarrow{A}_{\mathfrak{M}} w \iff v = w \text{ and } \mathfrak{M}, v \vDash A$$
$$v \xrightarrow{Aa\sigma}_{\mathfrak{M}} w \iff \mathfrak{M}, v \vDash A \text{ and } \exists u\colon\ v \xrightarrow{a}_{\mathfrak{M}} u \text{ and } u \xrightarrow{\sigma}_{\mathfrak{M}} w$$

Lemma 7.1.4
1. $v \xrightarrow{A\omega B}_{\mathfrak{M}} w \iff v \xrightarrow{\emptyset\omega A}_{\mathfrak{M}} w$ and $\mathfrak{M}, v \vDash A$
2. $v \xrightarrow{A\omega B}_{\mathfrak{M}} w \iff v \xrightarrow{A\omega\emptyset}_{\mathfrak{M}} w$ and $\mathfrak{M}, w \vDash B$

Proof Claim (1) is shown by case analysis on the length n of $A\omega B$. Let $n = 0$. Then ω is empty and $A = B$. We have $v \xrightarrow{A}_{\mathfrak{M}} w \iff v = w$ and $\mathfrak{M}, v \vDash A \iff v \xrightarrow{\emptyset}_{\mathfrak{M}} w$ and $\mathfrak{M}, v \vDash A$ since $\mathfrak{M}, v \vDash \emptyset$ is vacuously true.

Let $n > 0$. Then $\omega = aA'\omega'$ for some a, A' and ω'. Hence, we have

$$\begin{aligned} v \xrightarrow{AaA'\omega'B}_{\mathfrak{M}} w &\iff \mathfrak{M}, v \vDash A \text{ and } \exists u\colon\ v \xrightarrow{a}_{\mathfrak{M}} u \text{ and } u \xrightarrow{A'\omega'B}_{\mathfrak{M}} w \\ &\iff \mathfrak{M}, v \vDash A \text{ and } v \xrightarrow{\emptyset aA'\omega'B}_{\mathfrak{M}} w \end{aligned}$$

where the second equivalence follows since $\mathfrak{M}, v \vDash \emptyset$ is vacuously true.

For claim (2), we proceed by induction on the length n of $A\omega B$. If $n = 0$, the claim coincides with claim (1), which we have already established. So, let $n > 0$. Then $\omega = aA'\omega'$ for some a, A' and ω'. Hence, we have

$$\begin{aligned} v \xrightarrow{AaA'\omega'B}_{\mathfrak{M}} w &\iff \mathfrak{M}, v \vDash A \text{ and } \exists u\colon\ v \xrightarrow{a}_{\mathfrak{M}} u \text{ and } u \xrightarrow{A'\omega'B}_{\mathfrak{M}} w \\ &\iff \mathfrak{M}, v \vDash A \text{ and } \exists u\colon\ v \xrightarrow{a}_{\mathfrak{M}} u \text{ and } u \xrightarrow{A'\omega'\emptyset}_{\mathfrak{M}} w \text{ and } \mathfrak{M}, w \vDash B \\ &\iff v \xrightarrow{AaA'\omega'\emptyset}_{\mathfrak{M}} w \text{ and } \mathfrak{M}, w \vDash B \end{aligned}$$

where the second equivalence holds by the inductive hypothesis for $A'\omega'B$. ∎

Lemma 7.1.5 $v \xrightarrow{\omega A\omega'}_{\mathfrak{M}} w \iff \exists u\colon\ v \xrightarrow{\omega A}_{\mathfrak{M}} u$ and $u \xrightarrow{A\omega'}_{\mathfrak{M}} w$

Proof Straightforward induction on the length of ωA. Proceeds similarly to the proof of Lemma 7.1.4. ∎

Proposition 7.1.6
1. $v \xrightarrow{\alpha}_{\mathfrak{M}} w \iff \exists \sigma \in \mathcal{L}\alpha\colon\ v \xrightarrow{\sigma}_{\mathfrak{M}} w$
2. $\mathfrak{M}, v \vDash \langle\alpha\rangle s \iff \exists \sigma \in \mathcal{L}\alpha\, \exists w\colon\ v \xrightarrow{\sigma}_{\mathfrak{M}} w$ and $\mathfrak{M}, w \vDash s$
3. $\mathfrak{M}, v \vDash [\alpha] s \iff \forall \sigma \in \mathcal{L}\alpha\, \forall w\colon\ v \xrightarrow{\sigma}_{\mathfrak{M}} w$ implies $\mathfrak{M}, w \vDash s$

Proof Claims (2) and (3) follow from (1) and the definition of the satisfaction relation for diamonds and boxes. Hence, it suffices to prove (1).

The direction from left to right is shown as follows. Let $v \xrightarrow{\alpha}_{\mathfrak{M}} w$. We show the existence of $\sigma \in \mathcal{L}\alpha$ such that $v \xrightarrow{\sigma}_{\mathfrak{M}} w$ by induction on α.

Let $\alpha = a$. Then $\sigma = \emptyset a \emptyset \in \mathcal{L}a$. The claim follows since $v \xrightarrow{\sigma}_{\mathfrak{M}} w \iff v \xrightarrow{a}_{\mathfrak{M}} w$.

Let $\alpha = s$. Then $\sigma = \{s\} \in \mathcal{L}s$. The claim follows since $v \xrightarrow{\sigma}_{\mathfrak{M}} w \iff v \xrightarrow{s}_{\mathfrak{M}} w \iff v = w$ and $\mathfrak{M}, v \vDash s$.

Let $\alpha = \beta + \gamma$. Then $v \xrightarrow{\beta}_{\mathfrak{M}} w$ or $v \xrightarrow{\gamma}_{\mathfrak{M}} w$. By the inductive hypothesis for β or γ, there is some $\sigma \in \mathcal{L}\beta \cup \mathcal{L}\gamma$ such that $v \xrightarrow{\sigma}_{\mathfrak{M}} w$. The claim follows since $\mathcal{L}(\beta + \gamma) = \mathcal{L}\beta \cup \mathcal{L}\gamma$.

Let $\alpha = \beta\gamma$. Then there is some u such that $v \xrightarrow{\beta}_{\mathfrak{M}} u$ and $u \xrightarrow{\gamma}_{\mathfrak{M}} w$. By the inductive hypothesis for β and γ, there are $\omega A \in \mathcal{L}\beta$ and $A'\omega' \in \mathcal{L}\gamma$ such that $v \xrightarrow{\omega A}_{\mathfrak{M}} u$ and $u \xrightarrow{A'\omega'}_{\mathfrak{M}} w$. Let $B = A \cup A'$. By Lemma 7.1.3, $\omega B \in \mathcal{L}\beta$ and $B\omega' \in \mathcal{L}\gamma$. Hence, $\omega B \omega' \in \mathcal{L}(\beta\gamma)$. It remains to show that $v \xrightarrow{\omega B \omega'}_{\mathfrak{M}} w$. Since $\mathfrak{M}, u \vDash A$ and $\mathfrak{M}, u \vDash A'$, we also have $\mathfrak{M}, u \vDash B$, and hence $v \xrightarrow{\omega B}_{\mathfrak{M}} u$ and $u \xrightarrow{B\omega'}_{\mathfrak{M}} w$ (Lemma 7.1.4). The claim follows by Lemma 7.1.5.

Let $\alpha = \beta^*$. Then, for some $n \geq 0$, $v \xrightarrow{\beta}{}^{n}_{\mathfrak{M}} w$. We proceed by induction on n. If $n = 0$, then $v = w$, and hence $v \xrightarrow{\emptyset}_{\mathfrak{M}} w$. The claim follows since $\emptyset \in \mathcal{L}\emptyset \subseteq \mathcal{L}(\beta^*)$. If $n > 0$, then there is some u such that $v \xrightarrow{\beta}_{\mathfrak{M}} u$ and $u \xrightarrow{\beta}{}^{n-1}_{\mathfrak{M}} w$. By the outer and the inner inductive hypothesis, there are, respectively, $\omega A \in \mathcal{L}\beta$ and $A'\omega' \in \mathcal{L}(\beta^*)$ such that $v \xrightarrow{\omega A}_{\mathfrak{M}} u$ and $u \xrightarrow{A'\omega'}_{\mathfrak{M}} w$. Let $B = A \cup A'$. As in the previous case, we have $\omega B \in \mathcal{L}\beta$ and $B\omega' \in \mathcal{L}(\beta^*)$ (Lemma 7.1.3), and hence $\omega B \omega' \in \mathcal{L}(\beta^*)$. Also, we have $\mathfrak{M}, u \vDash B$, and hence $v \xrightarrow{\omega B}_{\mathfrak{M}} u$ and $u \xrightarrow{B\omega'}_{\mathfrak{M}} w$ (Lemma 7.1.4). Again, the claim follows by Lemma 7.1.5.

As for the direction from right to left, let $\sigma \in \mathcal{L}\alpha$ and $v \xrightarrow{\sigma}_{\mathfrak{M}} w$. We show $v \xrightarrow{\alpha}_{\mathfrak{M}} w$ by induction on α.

Let $\alpha = a$. Then $\sigma = AaB$ for some $A, B \subseteq \mathcal{F}$. Hence, in particular $v \xrightarrow{a}_{\mathfrak{M}} w$.

Let $\alpha = s$. Then $\sigma = A$ where $s \in A$. Then $v = w$, $\mathfrak{M}, v \vDash A$, and hence $\mathfrak{M}, w \vDash s$. The claim follows.

Let $\alpha = \beta + \gamma$. Then $\sigma \in \mathcal{L}\beta$ or $\sigma \in \mathcal{L}\gamma$. By the inductive hypothesis for β or γ, we have $v \xrightarrow{\beta}_{\mathfrak{M}} w$ or $v \xrightarrow{\gamma}_{\mathfrak{M}} w$. Hence, $v \xrightarrow{\beta+\gamma}_{\mathfrak{M}} w$.

Let $\alpha = \beta\gamma$. Then $\sigma = \omega A \omega'$ where $\omega A \in \mathcal{L}\beta$ and $A\omega' \in \mathcal{L}\gamma$. By Lemma 7.1.5, there is some u such that $v \xrightarrow{\omega A}_{\mathfrak{M}} u$ and $u \xrightarrow{A\omega'}_{\mathfrak{M}} w$. By the inductive hypothesis for β and γ, we have $v \xrightarrow{\beta}_{\mathfrak{M}} u$ and $u \xrightarrow{\gamma}_{\mathfrak{M}} w$. Hence, $v \xrightarrow{\gamma\beta}_{\mathfrak{M}} w$.

Let $\alpha = \beta^*$. Then, for some $n \geq 0$, we have $\sigma = A_0\omega_1A_1\omega_2 \ldots A_{n-1}\omega_nA_n$ where $A_n \in \mathcal{L}\emptyset$ and, for all $i \in [1, n]$, $A_{i-1}\omega_iA_i \in \mathcal{L}\beta$. We proceed by induction on n. If $n = 0$, then $\sigma = A_0$. Thus $v = w$, and hence $v \xrightarrow{\beta^*}_{\mathfrak{M}} w$. If $n > 0$, we have $\sigma = A_0\omega_1A_1\omega'$ where $A_1\omega' \in (\mathcal{L}\beta)^{n-1} \subseteq \mathcal{L}(\beta^*)$. By Lemma 7.1.5, there is some u such that $v \xrightarrow{A_0\omega_1A_1}_{\mathfrak{M}} u$ and $u \xrightarrow{A_1\omega'}_{\mathfrak{M}} w$. By the outer and the inner inductive hypothesis we obtain, respectively, $v \xrightarrow{\beta}_{\mathfrak{M}} u$ and $u \xrightarrow{\beta^*}_{\mathfrak{M}} w$. Hence, $v \xrightarrow{\beta^*}_{\mathfrak{M}} w$. ∎

7.2 Diamond and Box Decompositions

Without tests, the decomposition $\mathfrak{D}(\langle\alpha\rangle s)$ of a diamond formula $\langle\alpha\rangle s$ is a set of formulas that is equivalent to $\langle\alpha\rangle s$ when interpreted disjunctively, while the decomposition $\mathfrak{D}([\alpha]s)$ of a box formula $[\alpha]s$ is equivalent to $[\alpha]s$ when seen as a conjunction. Hence, every diamond formula is treated as a beta formula while every box formula is seen as an alpha formula. If we add tests, this simple view is no longer adequate. So, a diamond formula $\langle s\rangle t$ is seen as the *conjunction* of s and t, while a formula $[s]t$ is equivalent to the *disjunction* of $\sim s$ and t. Hence, for instance, $\langle(p+q)a\rangle p \equiv (p\wedge\langle a\rangle p)\vee(q\wedge\langle a\rangle p)$. Intuitively, the solution to this problem is to let diamond and box decompositions return sets of formulas where, in the case for diamonds, the overall set is interpreted disjunctively, while the element sets are interpreted conjunctively (and dually for boxes). Then, for instance, $\mathfrak{D}(\langle(p+q)a\rangle p)$ may be defined as the set $\{\{p,\ \langle a\rangle p\},\ \{q,\ \langle a\rangle p\}\}$.

If done naively, however, this approach causes some complications that become apparent when we try to define expansions. Let $\langle a\rangle\langle\alpha\rangle s \in C$. Following our approach in the test-free case, we would like an expansion of $\langle a\rangle\langle\alpha\rangle s \mid C$ to consist of pairs $t \mid D$ where t is a *single formula* obtained using a decomposition of $\langle\alpha\rangle s$. This is essential for our approach since it allows us to give a meaningful definition of links, paths and loops. Now, however, diamond decompositions consist of *sets of formulas*. Suppose $\mathcal{D}$ is a decomposition of $\langle\alpha\rangle s$. Intuitively, we want $t \mid D$ be such that $t \in A \in \mathcal{D}$ and D is a DNF of $\mathcal{R}_a \cup A$. But how do we select $t \in A$? Consider, for instance, the formula $s = \langle((\langle a\rangle\langle a^*\rangle p)b^*\rangle p$. The set $\mathcal{D} = \{A_1, A_2\}$ where $A_1 = \{\langle a\rangle\langle a^*\rangle p,\ p\}$ and $A_2 = \{\langle a\rangle\langle a^*\rangle p,\ \langle b\rangle\langle b^*\rangle p\}$ looks like a reasonable decomposition of s. Which of the formulas do we select for an expansion of $\langle a\rangle s$? Is the choice "don't know" or "don't care"? If the choice is "don't care", the set $\mathcal{D}$ induces four possible expansions of $\langle a\rangle s \mid \{\langle a\rangle s\}$, one of which is $\{\langle a\rangle\langle a^*\rangle p \mid A_1,\ \langle b\rangle\langle b^*\rangle p \mid A_2\}$. If the choice is "don't know", the set $\mathcal{D}$ induces one expansion of $\langle a\rangle s \mid \{\langle a\rangle s\}$ that consists of four elements: $\{\langle a\rangle\langle a^*\rangle p \mid A_1,\ p \mid A_1,\ \langle a\rangle\langle a^*\rangle p \mid A_2,\ \langle b\rangle\langle b^*\rangle p \mid A_2\}$. It can be shown by an argument similar to that in Remark 5.4.7 that choosing the formulas "don't care" destroys the refutation-soundness of the expansion rules. Choosing "don't know", on the other hand, seems unnecessarily inefficient since, as we will see in the following (Example 7.3.1), the smaller set $\{p \mid A_1,\ \langle b\rangle\langle b^*\rangle p \mid A_2\}$ is already an expansion of $\langle a\rangle s \mid \{\langle a\rangle s\}$.

To address this issue, we define diamond and box decompositions as sets of guarded formulas, where a **guarded formula** is a pair At such that A is a set of formulas and t is a formula.

A set $\mathcal{D}$ of guarded formulas is a **decomposition of** $\langle\alpha\rangle s$ if it satisfies the following conditions:

1. If $At \in \mathcal{D}$, then:
   a) $A;t \subseteq \mathcal{F}$ for every formula universe $\mathcal{F}$ containing $\langle\alpha\rangle s$.
   b) $t = s$ or $t = \langle a\rangle\langle\beta_1\rangle\dots\langle\beta_n\rangle s$ for some action a and programs $\beta_1,\dots,\beta_n$ ($n \geq 0$).
   c) Every formula in A is a proper subformula of $\langle\alpha\rangle s$.
2. $\mathcal{L}\alpha = \bigcup_{B\langle\beta_1\rangle\dots\langle\beta_n\rangle s\in\mathcal{D}} \mathcal{L}B\cdot\mathcal{L}\beta_1\cdot\ldots\cdot\mathcal{L}\beta_n$

Similarly to the test-free case, condition (1.b) implies that every formula t such that $At \in \mathcal{D}$ is a literal or a proper subformula of $\langle\alpha\rangle s$. Additionally, every formula in A (where $At \in \mathcal{D}$) is required by condition (1.c) to be a proper subformula of $\langle\alpha\rangle s$.

The definition of a **decomposition of** $[\alpha]s$ is now obtained from the above definition by replacing every occurrence of $\langle\alpha\rangle s$, $\langle a\rangle\langle\beta_1\rangle\dots\langle\beta_n\rangle s$, and $\langle\beta_1\rangle\dots\langle\beta_n\rangle s$ by $[\alpha]s$, $[a][\beta_1]\dots[\beta_n]s$, and $[\beta_1]\dots[\beta_n]s$, respectively, and adding the following condition:

1. d) If $\mathcal{D}$ is a decomposition of $[\alpha]s$ and $At \in \mathcal{D}$,
 then $\{\, {\sim}u \mid u \in A \,\} \subseteq \mathcal{F}$ for every formula universe $\mathcal{F}$ containing $[\alpha]s$.

As in the test-free case, conditions (1.b) and (2) ensure that diamond and box decompositions have the intended semantics.

Proposition 7.2.1 Let $\mathcal{D}$ be a decomposition of $\langle\alpha\rangle s$ and $\mathcal{E}$ be a decomposition of $[\alpha]s$.
1. $\mathfrak{M}, w \vDash \langle\alpha\rangle s \iff \exists At \in \mathcal{D}\colon\ \mathfrak{M}, w \vDash t$ and $\forall u \in A\colon\ \mathfrak{M}, w \vDash u$
2. $\mathfrak{M}, w \vDash [\alpha]s \iff \forall At \in \mathcal{E}\colon\ \mathfrak{M}, w \vDash t$ or $\exists u \in A\colon\ \mathfrak{M}, w \vDash {\sim}u$

Proof We show the direction from left to right in claim (1) by adapting the corresponding argument for Proposition 6.3.1 (the other direction adapts similarly and claim (2) follows analogously). Let $\mathfrak{M}, w \vDash \langle\alpha\rangle s$. By Proposition 7.1.6 (2), there is some $\sigma \in \mathcal{L}\alpha$ such that $w \xrightarrow{\sigma}_{\mathfrak{M}} v$ and $\mathfrak{M}, v \vDash s$. By condition (2) for diamond decompositions, $\sigma \in \mathcal{L}A \cdot \mathcal{L}\beta_1 \cdot \ldots \cdot \mathcal{L}\beta_n$ for some $A\langle\beta_1\rangle\dots\langle\beta_n\rangle s \in \mathcal{D}$. By n applications of Proposition 7.1.6 (2) and Lemma 7.1.5, we obtain that there is some $\tau \in \mathcal{L}A$ and some u such that $w \xrightarrow{\tau}_{\mathfrak{M}} u$ and $\mathfrak{M}, u \vDash \langle\beta_1\rangle\dots\langle\beta_n\rangle s$. Since $\tau \in \mathcal{L}A$, we have $u = w$ and $\mathfrak{M}, w \vDash A$. The claim follows. ∎

How do we compute diamond and box decompositions? Since a decomposition no longer consists of single formulas, simply taking some restriction of the alpha-beta closure no longer does the trick. Instead, we work with the closure under the following rules for guarded formulas.

$$\frac{A\langle t\rangle s}{(A;t)s} \qquad \frac{A\langle\alpha+\beta\rangle s}{A\langle\alpha\rangle s,\ A\langle\beta\rangle s} \qquad \frac{A\langle\alpha\beta\rangle s}{A\langle\alpha\rangle\langle\beta\rangle s} \qquad \frac{A\langle\alpha^*\rangle s}{As,\ A\langle\alpha\rangle\langle\alpha^*\rangle s}$$

$$\frac{A[t]s}{(A;t)s} \qquad \frac{A[\alpha+\beta]s}{A[\alpha]s,\ A[\beta]s} \qquad \frac{A[\alpha\beta]s}{A[\alpha][\beta]s} \qquad \frac{A[\alpha^*]s}{As,\ A[\alpha][\alpha^*]s}$$

Given a set $\mathcal{G}$ of guarded formulas, we call the closure of $\mathcal{G}$ under the above rules the **guarded alpha-beta closure of** $\mathcal{G}$ (notation $\mathcal{A}^{g}\mathcal{G}$).

Note that for test-free formulas $\mathcal{A}^{g}$ coincides with the ordinary alpha-beta closure $\mathcal{A}$ as defined in § 6.3. More generally, $\mathcal{A}^{g}$ is related to $\mathcal{F}$ and $\mathcal{A}$ by the following lemma.

Lemma 7.2.2
1. If $A;s \subseteq \mathcal{F}$ and $Bt \in \mathcal{A}^{g}\{As\}$, then $B;t \subseteq \mathcal{F}$.
2. If $Bt \in \mathcal{A}^{g}\{A\langle\alpha\rangle s\}$, then $t \in \mathcal{A}\{\langle\alpha\rangle s\}$ and $B \setminus A \subseteq \mathcal{A}\{\langle\alpha\rangle s\}$.
3. If $Bt \in \mathcal{A}^{g}\{A[\alpha]s\}$, then $t \in \mathcal{A}\{[\alpha]s\}$ and $\{\, {\sim}u \mid u \in B \setminus A \,\} \subseteq \mathcal{A}\{[\alpha]s\}$.

Proof All three claims are shown by straightforward induction on the derivation of, respectively, $Bt \in \mathcal{A}^{g}\{As\}$, $Bt \in \mathcal{A}^{g}\{A\langle\alpha\rangle s\}$, and $Bt \in \mathcal{A}^{g}\{A[\alpha]s\}$. ∎

Given a formula s, we can restrict the above rules such that they do not apply to any guarded formula of the form As. We call the closure of a set $\mathcal{G}$ under the restricted version of the rules the **guarded alpha-beta closure of $\mathcal{G}$ halting at** s (notation $\mathcal{A}^{g}_{s}\mathcal{G}$). Clearly, for every s and $\mathcal{G}$, we have $\mathcal{A}^{g}_{s}\mathcal{G} \subseteq \mathcal{A}^{g}\mathcal{G}$. Let $s = \langle\alpha\rangle t$ or $s = [\alpha]t$. We define:

$$\mathfrak{D}(As) := \{\, Bu \in \mathcal{A}^{g}_{t}\{As\} \mid u = t \text{ or } u \text{ literal}\,\}$$

We abbreviate $\mathfrak{D}(\emptyset s)$ as $\mathfrak{D}s$.

Example 7.2.3 Consider the formulas $\langle(p+q)a\rangle p$ and $\langle(\langle a\rangle\langle a^*\rangle p)b^*\rangle p$ that we use as examples at the beginning of the section. We have:

$$\begin{aligned}
\mathcal{A}^{g}_{p}\{\emptyset\langle(p+q)a\rangle p\} &= \{\emptyset\langle(p+q)a\rangle p,\ \emptyset\langle p+q\rangle\langle a\rangle p,\ \emptyset\langle p\rangle\langle a\rangle p,\ \emptyset\langle q\rangle\langle a\rangle p,\\
&\qquad \{p\}\langle a\rangle p,\ \{q\}\langle a\rangle p\}\\
\mathfrak{D}(\langle(p+q)a\rangle p) &= \{\{p\}\langle a\rangle p,\ \{q\}\langle a\rangle p\}\\
\mathcal{A}^{g}_{p}\{\emptyset\langle(\langle a\rangle\langle a^*\rangle p)b^*\rangle p\} &= \{\emptyset\langle(\langle a\rangle\langle a^*\rangle p)b^*\rangle p,\ \emptyset\langle\langle a\rangle\langle a^*\rangle p\rangle\langle b^*\rangle p,\\
&\qquad \{\langle a\rangle\langle a^*\rangle p\}\langle b^*\rangle p,\ \{\langle a\rangle\langle a^*\rangle p\}p,\\
&\qquad \{\langle a\rangle\langle a^*\rangle p\}\langle b\rangle\langle b^*\rangle p\}\\
\mathfrak{D}(\langle(\langle a\rangle\langle a^*\rangle p)b^*\rangle p) &= \{\{\langle a\rangle\langle a^*\rangle p\}p,\ \{\langle a\rangle\langle a^*\rangle p\}\langle b\rangle\langle b^*\rangle p\}
\end{aligned}$$ ◂

We now show that $\mathfrak{D}(\langle\alpha\rangle s)$ is a decomposition of $\langle\alpha\rangle s$ and $\mathfrak{D}([\alpha]s)$ is a decomposition of $[\alpha]s$.

Condition (1.d) for box decompositions follows by Lemma 7.2.2 (3) since $\mathcal{A}\{[\alpha]s\} \subseteq \mathcal{F}$ whenever $[\alpha]s \in \mathcal{F}$. Since other conditions are shown analogously for diamond and box decompositions, we will only detail the argument for diamonds

Condition (1.a) is satisfied by Lemma 7.2.2 (1). Conditions (1.b) and (1.c) for diamond decompositions follow with the following adaptation of Lemma 6.3.4.

Lemma 7.2.4 If $Bt \in \mathcal{A}_s\{A\langle\alpha\rangle s\}$, then every formula in $B \setminus A$ is a proper subformula of $\langle\alpha\rangle s$, and $t = \langle\beta_1\rangle \ldots \langle\beta_n\rangle s$ for some $n \geq 0$ and $\beta_1, \ldots, \beta_n$.

The proof of the lemma proceeds analogously to the one for Lemma 6.3.4.

To show that $\mathfrak{D}(\langle\alpha\rangle s)$ satisfies condition (2) for diamond decompositions we adapt Lemmas 6.3.5 and 6.3.6 as follows.

Lemma 7.2.5 If $B\langle\beta_1\rangle \ldots \langle\beta_n\rangle s \in \mathcal{A}_s\{A\langle\alpha\rangle s\}$, then $\mathcal{L}B \cdot \mathcal{L}\beta_1 \cdot \ldots \cdot \mathcal{L}\beta_n \subseteq \mathcal{L}A \cdot \mathcal{L}\alpha$.

Proof We proceed by induction on the derivation of $B\langle\beta_1\rangle \ldots \langle\beta_n\rangle s \in \mathcal{A}_s\{A\langle\alpha\rangle s\}$. All cases proceed analogously to the respective cases in the proof of Lemma 6.3.5 (with Lemma 7.2.4 in place of Lemma 6.3.4 and Proposition 7.1.2 in place of Proposition 6.2.1) except for the case where $B\langle\beta_1\rangle \ldots \langle\beta_n\rangle s$ is obtained from $A'\langle t\rangle\langle\gamma_2\rangle \ldots \langle\gamma_m\rangle s$ where t is a test formula. In this case, $B = A'\,;t$, $m = n+1$, and, for all $i \in [1,n]$, $\beta_i = \gamma_{i+1}$. The claim follows from the inductive hypothesis, $\mathcal{L}A' \cdot \mathcal{L}t \cdot \mathcal{L}\beta_1 \cdot \ldots \cdot \mathcal{L}\beta_n \subseteq \mathcal{L}A \cdot \mathcal{L}\alpha$, by Proposition 7.1.2 (2) since $\mathcal{L}A' \cdot \mathcal{L}t = \mathcal{L}A' \cup \mathcal{L}t = \mathcal{L}(A'\,;t) = \mathcal{L}B$. ∎

Lemma 7.2.6 $\mathcal{A}_{\langle\beta\rangle s}\{A\langle\alpha\rangle\langle\beta\rangle s\} \subseteq \mathcal{A}_s\{A\langle\alpha\rangle\langle\beta\rangle s\}$

Proof Let $Bt \in \mathcal{A}_{\langle\beta\rangle s}\{A\langle\alpha\rangle\langle\beta\rangle s\}$. We show that $Bt \in \mathcal{A}_s\{A\langle\alpha\rangle\langle\beta\rangle s\}$ by induction on the derivation of $Bt \in \mathcal{A}_{\langle\beta\rangle s}\{A\langle\alpha\rangle\langle\beta\rangle s\}$. All cases proceed analogously to the respective cases in the proof of Lemma 6.3.6 except for the case when Bt is obtained from $A'\langle u\rangle\langle\gamma_1\rangle\dots\langle\gamma_n\rangle\langle\beta\rangle s$ where u is a test formula. In this case, $B = A';u$ and $t = \langle\gamma_1\rangle\dots\langle\gamma_n\rangle\langle\beta\rangle s$. The claim follows since, by the inductive hypothesis, $A'\langle u\rangle t \in \mathcal{A}_s\{A\langle\alpha\rangle\langle\beta\rangle s\}$. ∎

With the necessary prerequisites in place, condition (2) for diamond decompositions immediately follows from the following more general claim.

Proposition 7.2.7 $\mathcal{L}A \cdot \mathcal{L}\alpha = \bigcup_{B\langle\beta_1\rangle\dots\langle\beta_n\rangle s \in \mathfrak{D}(A\langle\alpha\rangle s)} \mathcal{L}B \cdot \mathcal{L}\beta_1 \cdot \ldots \cdot \mathcal{L}\beta_n$

Proof The inclusion $\bigcup_{B\langle\beta_1\rangle\dots\langle\beta_n\rangle s \in \mathfrak{D}(A\langle\alpha\rangle s)} \mathcal{L}B \cdot \mathcal{L}\beta_1 \cdot \ldots \cdot \mathcal{L}\beta_n \subseteq \mathcal{L}A \cdot \mathcal{L}\alpha$ follows with Lemma 7.2.5. As for the other inclusion, let α be a program. We show $\forall A\,\forall s\,\forall\sigma \in \mathcal{L}A \cdot \mathcal{L}\alpha$: $\sigma \in \mathcal{L}B \cdot \mathcal{L}\beta_1 \cdot \ldots \cdot \mathcal{L}\beta_n$ for some $B\langle\beta_1\rangle\dots\langle\beta_n\rangle s \in \mathfrak{D}(A\langle\alpha\rangle s)$ by induction on α. We distinguish five cases depending on the shape of α. Four of them proceed analogously to the corresponding cases for Proposition 6.3.7, with Lemma 7.2.6 being used in place of Lemma 6.3.6. The remaining case proceeds as follows.

Let $\alpha = t$. Let $\sigma \in \mathcal{L}A \cdot \mathcal{L}t$. We have $(A;t)s \in \mathcal{A}_s^g\{A\langle t\rangle s\}$ and, consequently, $(A;t)s \in \mathfrak{D}(A\langle t\rangle s)$. Hence, it suffices to show that $\sigma \in \mathcal{L}(A;t)$. This follows since $\mathcal{L}A \cdot \mathcal{L}t = \mathcal{L}A \cup \mathcal{L}t = \mathcal{L}(A;t)$. ∎

Finally, note that by Lemma 7.2.2 (2,3), if $s = \langle\alpha\rangle t$ or $s = [\alpha]t$, the set $|\mathcal{A}^g\{\emptyset s\}|$ is bounded by $2^{|\mathcal{A}\{s\}|} \cdot |\mathcal{A}\{s\}|$. Therefore, since $\mathcal{A}\{s\}$ is linear in the size of s, $\mathfrak{D}s$ can be computed in exponential time with respect to the size of s.

Remark 7.2.8 We call a guarded formula $As \in \mathcal{G}$ **minimal in** $\mathcal{G}$ if there is no $Bs \in \mathcal{G}$ such that $B \subsetneq A$. To obtain a more concise program DNF, instead of taking all guarded formulas in $\mathfrak{D}s$, we can take only guarded formulas that are minimal in $\mathfrak{D}s$. This suffices since $A \subseteq B$ implies $\mathcal{L}B \subseteq \mathcal{L}A$, and hence $\mathcal{L}B \cdot \mathcal{L}\alpha_1 \cdot \ldots \cdot \mathcal{L}\alpha_n \subseteq \mathcal{L}A \cdot \mathcal{L}\alpha_1 \cdot \ldots \cdot \mathcal{L}\alpha_n$ for every formula $\langle\alpha_1\rangle\dots\langle\alpha_n\rangle t \in \mathfrak{D}s$. This can significantly reduce the size of program DNFs returned by $\mathfrak{D}$. For instance, consider the formula $\langle(p+q)^*\rangle p$. We have $\mathfrak{D}(\langle(p+q)^*\rangle p) = \{\emptyset p, \{p\}p, \{q\}p, \{p,q\}p\}$. Clearly, $\emptyset p$ is the only guarded formula minimal in $\mathfrak{D}(\langle(p+q)^*\rangle p)$, and hence $\{\emptyset p\}$ is a program DNF of $\langle(p+q)^*\rangle p$. More generally, the cardinality of $\mathcal{D}_n = \mathfrak{D}(\langle(p_1 + \cdots + p_n)^*\rangle p)$ grows exponentially in n, while, for every n, $\emptyset p$ remains the unique guarded formula minimal in $\mathcal{D}_n$.

For clarity of presentation, this optimization is not included in the definition of $\mathfrak{D}$. All our results adapt to the optimized version in a straightforward way. ◂

Together with the usual decomposition rules for conjunctions and disjunctions, $\mathfrak{D}$ yields a terminating decomposition table for HPDL, which may be represented as shown in Figure 7.1. As discussed earlier, a diamond formula is now decomposed into a set

formula		constituents
$s \wedge t$	α	$\{s, t\}$
$s \vee t$	β	$\{s, t\}$
$[\gamma]s$, γ not an action	α_β	$\{\, \{\, {\sim}u \mid u \in A \,\} ; t \mid At \in \mathfrak{D}([\gamma]s) \,\}$
$\langle\gamma\rangle s$, γ not an action	β_α	$\{\, A ; t \mid At \in \mathfrak{D}(\langle\gamma\rangle s) \,\}$

Figure 7.1: Terminating decomposition table for HPDL

of sets of formulas where the overall set is seen as a disjunction while the element sets are interpreted conjunctively. Dually, a box formula is decomposed into a conjunction of disjunctions. The correctness of this decomposition follows from Proposition 7.2.1:

Corollary 7.2.9
1. $\mathfrak{M}, w \models \langle\alpha\rangle s \iff \exists At \in \mathfrak{D}(\langle\alpha\rangle s) \colon \mathfrak{M}, w \models t$ and $\forall u \in A \colon \mathfrak{M}, w \models u$
2. $\mathfrak{M}, w \models [\alpha]s \iff \forall At \in \mathfrak{D}([\alpha]s) \colon \mathfrak{M}, w \models t$ or $\exists u \in A \colon \mathfrak{M}, w \models {\sim}u$

As in the test-free case, the table is terminating because the constituents of every formula s are literals or proper subformulas of s (up to $\sim$).

The table induces the definition of **Hintikka sets** as nonempty sets H of formulas that satisfy the following properties.
- For every predicate p: $\{p, \neg p\} \nsubseteq H$.
- If $s \wedge t \in H$, then $s \in H$ and $t \in H$.
- If $s \vee t \in H$, then $s \in H$ or $t \in H$.
- If $[\alpha]s \in H$ where α is not an action, then for every $At \in \mathfrak{D}([\alpha]s)$:
  $\{\, {\sim}u \mid u \in A \,\} ; t \cap H \neq \emptyset$
- If $\langle\alpha\rangle s \in H$ where α is not an action, then for some $At \in \mathfrak{D}(\langle\alpha\rangle s)$: $A ; t \subseteq H$.

7.3 Branches and Evidence

The definitions of the upward closure and support induced by the new decomposition table adapt analogously to the definition of Hintikka sets. Given the new notion of support, the defining conditions for DNFs are unchanged compared to § 5.2. Their computation via Hintikka decompositions adapts in the intuitive way.

Let C be a clause and $\langle a\rangle s \in C$. An **expansion of** $\langle a\rangle s \,|\, C$ is again defined by case analysis on the shape of s:
- If s is not a diamond formula and $\mathcal{D}$ is a DNF of $\mathfrak{R}_a C ; s$, then $\{\, (s \,|\, D) \mid D \in \mathcal{D} \,\}$ is an expansion of $\langle a\rangle s \,|\, C$.
- If s is a diamond formula and, for every $At \in \mathfrak{D}s$, $\mathcal{D}_{At}$ is a DNF of $\mathfrak{R}_a C \cup A ; t$, then $\bigcup_{At \in \mathfrak{D}s} \{\, (t \,|\, D) \mid D \in \mathcal{D}_{At} \,\}$ is an expansion of $\langle a\rangle s \,|\, C$.

Example 7.3.1 Let $s = \langle(\langle a\rangle\langle a^*\rangle p)b^*\rangle p$. As we saw in Example 7.2.3:

$$\mathfrak{D}s = \{\{\langle a\rangle\langle a^*\rangle p\}p,\ \{\langle a\rangle\langle a^*\rangle p\}\langle b\rangle\langle b^*\rangle p\}$$

Since the formulas p, $\langle a\rangle\langle a^*\rangle p$, $\langle b\rangle\langle b^*\rangle p$ are all literals, the computation of a DNF is trivial and the set $\{p \mid \{\langle a\rangle\langle a^*\rangle p,\ p\},\ \langle b\rangle\langle b^*\rangle p \mid \{\langle a\rangle\langle a^*\rangle p,\ \langle b\rangle\langle b^*\rangle p\}\}$ is an expansion of $\langle a\rangle s \mid \{\langle a\rangle s\}$.

For an example where the DNF computation is not trivial, let $s = \langle p \vee q\rangle(\neg p \vee \neg q)$. Then:

$$\begin{aligned} \mathcal{A}^{\mathrm{g}}_{\neg p\vee\neg q}\{s\} &= \{s,\ \{p \vee q\}\neg p \vee \neg q\} \\ \mathfrak{D}s &= \{\{p \vee q\}\neg p \vee \neg q\} \end{aligned}$$

Consider the clause $C = \{\langle a\rangle s,\ [a]([p]r_1 \wedge [q]r_2)\}$. We have:

- $\mathfrak{D}([p]r_1) = \{\{p\}r_1\}$, and hence $\{\{\neg p\},\ \{r_1\}\}$ is a DNF of $\{[p]r_1\}$.
- $\mathfrak{D}([q]r_2) = \{\{q\}r_2\}$, and hence $\{\{\neg q\},\ \{r_2\}\}$ is a DNF of $\{[q]r_2\}$.
- The set $\{D_1, D_2\}$ where $D_1 = \{p, \neg q, r_1\}$ and $D_2 = \{\neg p, q, r_2\}$ is a DNF of $\{[p]r_1 \wedge [q]r_2,\ p \vee q,\ \neg p \vee \neg q\}(= \mathfrak{R}_a C \cup \{p \vee q\}\,;\neg p \vee \neg q)$.
- The set $\{\neg p \vee \neg q \mid D_1,\ \neg p \vee \neg q \mid D_2\}$ is an expansion of $\langle a\rangle s \mid C$. ◂

The definition of links adapts to the new notion of an expansion without changes to its formulation, as do the definitions of branches, realization, paths, runs, loops, and evidence. Also, the statement and the proof of Proposition 6.4.5 remain unchanged.

7.4 Demos

The notion of demos is adapted by extending the definition of syntactic transition relations $\xrightarrow{\alpha}_S$ as proposed in § 3.7.

As in the test-free case, the proof of demo existence adapts in a straightforward way. With Corollary 7.2.9 we conclude that $H_{\mathfrak{M},w}$ as defined in § 3.3.3 is a Hintikka set for every model $\mathfrak{M}$ and every $w \in |\mathfrak{M}|$. Demo existence then follows by adapting the proof of Lemma 3.3.12 as described in § 3.7.

The proof of demo satisfaction requires more work. In a nutshell, we combine the adaptations for tests in § 3.7 with those for terminating decomposition tables in § 6.5. Lemma 3.3.6 is replaced by Lemma 3.7.1. The proof of the lemma remains unchanged. Lemma 6.5.1 is modified as follows.

Lemma 7.4.1 Let S be a Hintikka system, $\{H, H'\} \subseteq S$, and $[\alpha_1]\dots[\alpha_n]s \in H$ $(n \geq 0)$ such that

- $H \xrightarrow{\sigma}_{\mathfrak{M}_S} H'$ for some $\sigma \in \mathcal{L}\alpha_1 \cdot \ldots \cdot \mathcal{L}\alpha_n$ and
- for all formulas t in σ and all $H \in S$, $\sim t \in H$ implies $\mathfrak{M}_S, H \vDash \sim t$.

Then $s \in H'$.

Proof We proceed by lexicographic induction on the length of σ and the decomposition order of $[\alpha_1]\dots[\alpha_n]s$. If $n = 0$, the claim is trivial. Let $n \geq 1$. Depending on the shape of α_1, we distinguish two cases.

If $\alpha_1 = a$, we have $\sigma = Aa\tau$. Hence, $\mathfrak{M}_S, H \vDash A$ and there is some set H'' such that $H \xrightarrow{a}_{\mathfrak{M}_S} H''$ and $H'' \xrightarrow{\tau}_{\mathfrak{M}_S} H'$. By the definition of $\xrightarrow{a}_{\mathfrak{M}_S}$, we have $[\alpha_2]\dots[\alpha_n]s \in H''$. Moreover, $\tau \in \mathcal{L}\alpha_2 \cdot \ldots \cdot \mathcal{L}\alpha_n$. The claim follows by the inductive hypothesis for τ and $[\alpha_2]\dots[\alpha_n]s$.

Otherwise, for every $At \in \mathfrak{D}([\alpha_1]\dots[\alpha_n]s)$ we have $\{\sim u \mid u \in A\}; t \cap H \neq \emptyset$. By condition (2) for box decompositions, there is some $A[\beta_1]\dots[\beta_m]s \in \mathfrak{D}([\alpha_1]\dots[\alpha_n]s)$ such that $\sigma \in \mathcal{L}A \cdot \mathcal{L}\beta_1 \cdot \ldots \cdot \mathcal{L}\beta_m$. Suppose, for some $u \in A$, we have $\sim u \in H$. Then, by assumption, $\mathfrak{M}_S, H \vDash \sim u$, which contradicts $\mathfrak{M}_S, H \vDash u$ and thus (since $u \in A$ and $\sigma \in \mathcal{L}A \cdot \mathcal{L}\beta_1 \cdot \ldots \cdot \mathcal{L}\beta_m$) $H \xrightarrow{\sigma}_{\mathfrak{M}_S} H'$. Consequently, $\{\sim u \mid u \in A\} \cap H = \emptyset$ and thus $[\beta_1]\dots[\beta_m]s \in H$. The claim follows by the inductive hypothesis for σ and $[\beta_1]\dots[\beta_m]s$ ($[\beta_1]\dots[\beta_m]s$ is smaller than $[\alpha_1]\dots[\alpha_n]s$ according to the decomposition order). ∎

From this, we immediately derive the original formulation of Lemma 3.7.2 (see the proof of Lemma 6.5.2). With Lemmas 3.7.1 and 3.7.2 in place, Lemma 3.3.8 adapts as described in § 3.7. Theorem 3.3.9 follows.

It remains to show that the core of an evident branch describes a demo. For this, we first need to adapt the definition of $\xrightarrow{\sigma}_S$ to account for tests. This is done as follows:

$$H \xrightarrow{A}_S H' \iff H = H' \text{ and } A \subseteq H$$

$$H \xrightarrow{Aa\sigma}_S H' \iff A \subseteq H \text{ and } \exists H''\colon\ H \xrightarrow{a}_S H'' \text{ and } H'' \xrightarrow{\sigma}_S H'$$

Proposition 6.5.3 can be shown for the new definition of $\xrightarrow{\sigma}_S$ by adapting the proof of Proposition 7.1.6 (1). The notion of $\diamond$-preadmissibility remains unchanged. Also, the formulation of Lemma 6.5.4 remains unchanged and its proof adapts in a straightforward way. Lemma 6.5.5 is reproved as follows.

Lemma 7.4.2 A Hintikka system is $\diamond$-preadmissible if and only if it is $\diamond$-admissible.

Proof The direction from right to left follows the same as for Lemma 6.5.5. So, let S be $\diamond$-preadmissible and let $\langle\alpha\rangle s \in H \in S$. If α is an action, the claim follows since S is $\diamond$-preadmissible. Otherwise, we have $A; t \subseteq H$ for some $At \in \mathfrak{D}(\langle\alpha\rangle s)$, where $t = \langle\beta_1\rangle\dots\langle\beta_n\rangle s$ for some $n \geq 0$ and $\beta_1, \dots, \beta_n$ (Lemma 7.2.4).

If $n = 0$, then $A \in \mathcal{L}\alpha$ (condition (2) for diamond decompositions), and hence the claim follows from $H \xrightarrow{A}_S H$ by Proposition 6.5.3.

If $n \geq 1$, then $\beta_1 = b$ (for some action b). Since S is $\diamond$-preadmissible, there is some $H' \in S$ such that $H \xrightarrow{b\beta_2\dots\beta_n}_S H'$ and $s \in H'$. It suffices to show $H \xrightarrow{\alpha}_S H'$. By Proposition 6.5.3 and since $A \subseteq H$, we have $H \xrightarrow{Ab\sigma}_S H'$ for some $\sigma \in \mathcal{L}\beta_2 \cdot \ldots \cdot \mathcal{L}\beta_n$. The claim follows by Proposition 6.5.3 since, by condition (2) for diamond decompositions, $Ab\sigma \in \mathcal{L}A \cdot \mathcal{L}b \cdot \mathcal{L}\beta_2 \cdot \ldots \cdot \mathcal{L}\beta_n \subseteq \mathcal{L}\alpha$. ∎

Theorem 6.5.6 then follows as before and we are done.

7.5 Expansion Rules

The definition of nc-DNFs from DNFs remains unaffected. Also, Proposition 5.5.9 still holds. Nominally consistent expansions are defined from nc-DNFs analogously to how we define ordinary expansions from DNFs in § 7.3. Assuming the updated notion of nc-expansions, the expansion rules for HPDL with tests still look as shown in Figure 6.2. Once again, a branch is evident if and only if it is maximal. Therefore, the rules are demonstration-sound by the results in § 7.4. To show their refutation-soundness, we once again adapt our argument from the test-free case.

The definition of $\delta_{\mathfrak{M}}$ remains unchanged (except that the variable σ now ranges over guarded strings rather than words and the notations $|\sigma|$ and $\xrightarrow{\sigma}_{\mathfrak{M}}$ are defined as in § 7.1). Proposition 6.6.2 follows as before, now using Proposition 7.1.6 and Lemma 7.1.5 in place of Proposition 6.2.3 and Lemma 6.2.2. Satisfaction of links and branches is defined as before.

While the proof of Proposition 6.6.4 remains unchanged, Lemma 6.6.5 requires some adaptations.

Lemma 7.5.1 Let Γ be a branch, $\langle a\rangle s \in C \in \Gamma$, and $\mathcal{E}$ be an nc-expansion of $\langle a\rangle s \,|\, C$ in Γ. Let $\mathfrak{M}$ be a model of all clauses in Γ. Then there is some $t \,|\, D \in \mathcal{E}$ such that $\delta_{\mathfrak{M}}Dt < \delta_{\mathfrak{M}}C(\langle a\rangle s)$.

Proof If s is not a diamond formula, the proof proceeds the same as for Lemma 6.6.5. So, let s be a diamond formula. Then we have $s = \langle\alpha_1\rangle \dots \langle\alpha_n\rangle t$ for some $n \geq 1$. By Proposition 6.6.2, $\delta_{\mathfrak{M}}C(\langle a\rangle s) < \infty$. Let $v, w \in |\mathfrak{M}|$ and $\sigma = A\omega \in \mathcal{L}\alpha_1 \cdot \dots \mathcal{L}\alpha_n$ be such that $v \xrightarrow{\emptyset a\sigma}_{\mathfrak{M}} w$, $\mathfrak{M}, v \vDash C$, $\mathfrak{M}, w \vDash t$, and $|\sigma| = \delta_{\mathfrak{M}}C(\langle a\rangle s) - 1$. Let $u \in |\mathfrak{M}|$ be such that $v \xrightarrow{a}_{\mathfrak{M}} u$ and $u \xrightarrow{\sigma}_{\mathfrak{M}} w$. By condition (2) for diamond decompositions, there is some $B\langle\beta_1\rangle \dots \langle\beta_m\rangle t \in \mathbb{D}s$ such that $\sigma \in \mathcal{L}B \cdot \mathcal{L}\beta_1 \cdot \dots \cdot \mathcal{L}\beta_m$. In particular, we have $B \subseteq A$. Since $\mathfrak{M}, u \vDash \mathcal{R}_a C$ and $\mathfrak{M}, u \vDash A$ (the latter following from $u \xrightarrow{\sigma}_{\mathfrak{M}} w$), we have

$$\delta_{\mathfrak{M}}(\mathcal{R}_a C \cup B)(\langle\beta_1\rangle \dots \langle\beta_m\rangle t) \leq \delta_{\mathfrak{M}}(\mathcal{R}_a C \cup A)(\langle\beta_1\rangle \dots \langle\beta_m\rangle t) \leq |\sigma|$$

and hence $\mathfrak{M}, u \vDash \mathcal{R}_a C \cup B;\langle\beta_1\rangle \dots \langle\beta_m\rangle t$ (Proposition 6.6.2). Let $\mathcal{D}$ be an nc-DNF of $\mathcal{R}_a C \cup B;\langle\beta_1\rangle \dots \langle\beta_m\rangle t$ such that $\{\, (\langle\beta_1\rangle \dots \langle\beta_m\rangle t \,|\, D) \mid D \in \mathcal{D} \,\} \subseteq \mathcal{E}$. By Proposition 5.5.9, there is some $D \in \mathcal{D}$ such that $\mathfrak{M}, u \vDash D$. Hence, $\langle\beta_1\rangle \dots \langle\beta_m\rangle t \,|\, D \in \mathcal{E}$ and $\delta_{\mathfrak{M}}D(\langle\beta_1\rangle \dots \langle\beta_m\rangle t) \leq |\sigma| = \delta_{\mathfrak{M}}C(\langle a\rangle s) - 1$. ∎

Theorem 6.6.6 follows as before with Lemma 7.5.1 in place of Lemma 6.6.5. Thus, the rules induce a decision procedure for HPDL. The procedure runs in NExpTime since the search tree explored by the procedure has exponential depth (the size of a branch is exponentially bounded in $|\mathcal{F}|$) and an exponential branching factor (the cardinality of an expansion is at most exponential in $|\mathcal{F}|$ since both diamond decompositions and DNFs are at most exponential in $|\mathcal{F}|$).

8 Conclusion

This thesis was dedicated to the study of incremental decision procedures for modal logics with nominals and eventualities, with the goal of developing an incremental procedure for hybrid PDL. In this section, we review how we achieved this goal and summarize our main contributions. Afterwards, we look at some possiliblities for future research.

8.1 Summary

We began our investigations by looking at abstract, nonincremental decision procedures as developed by Pratt [160] for PDL. We modularized Pratt's approach and adapted it to extensions of PDL with nominals and difference modalities. As a key technique in Pratt's approach we identified *pruning*. We observed that in the presence of nominals, pruning becomes incomplete. To remedy this problem, we complemented pruning by a *guessing* stage used to make the candidate models *nominally admissible*. To account for the existential difference modality, we extended the syntax by auxiliary nominals, while the universal difference modality was handled by suitably extending the guessing stage.

In the next step, we investigated the connections between Pratt-style nonincremental procedures and incremental graph procedures as developed by Goré and Widmann [93, 200]. We restricted our investigations to K*. We showed how Pratt's approach in [160] can be stepwise refined first to his tableau-based approach in [161] and then to a variant of Goré and Widmann's incremental procedure. To obtain a compact clausal representation, we introduced *graph tableaux* based on *normal*, *branching*, and *clashing clauses*. To simplify correctness arguments, we introduced *clausal demos* complemented by a notion of *support* used to infer the satisfaction of nonliteral formulas from literals. To achieve incrementality, we refined Pratt-style pruning to two complementary mechanisms, called *eager* and *cautious pruning*, which produce certificates for the satisfiability and the unsatisfiability of formulas.

Faced with difficulties in extending pruning to nominals while preserving incrementality, we turned our attention to tree procedures. We began by developing an incremental decision procedure for H* as the simplest logic that contains both nominals and eventualities. We observed that the clausal representation we had used for our graph procedures leads to incompleteness of tree search in the presence of eventualities. We solved this problem by separating modal reasoning from propositional reasoning, connecting the two levels via an abstract interface. To deal with eventualities via tree search, we introduced the notion of a *loop* and postulated the absence of loops as a consistency criterion for branches. To argue the correctness of this approach, we strengthened the notion of satisfaction for branches by extending satisfaction to links. We dealt with nominals by introducing *nominal propagation*. Unlike other approaches to nominals [29, 127, 39], nominal propagation is static in that it does not modify existing clauses and links. This makes nominal propagation compatible with our clausal representation and with our treatment of eventualities.

Finally, we extended the tree procedure for H* to HPDL, first without tests and then with tests. When extending tree search from H* to HPDL, the main complication turned out to be that the naive alpha-beta decomposition no longer terminates on all diamond and box formulas. This destroys the correspondence between clauses as used by our tableau methods and Hintikka sets used to relate the clauses to states of a model. To regain the correspondence, we devised a terminating decomposition table and proved its correctness with the help of a language-theoretic semantics of programs.

Adding tests causes two main complications. The first complication is the need for a language-theoretic semantics that can account for tests. We dealt with this complication by developing a new language-theoretic semantics based on a model for Kleene algebras with tests. The second complication is that the definition of a terminating decomposition table and the computation of diamond and box decompositions become more involved. We faced this complication by introducing *guarded formulas* and lifting the computation of diamond and box decompositions to guarded formulas.

8.2 Future Research

The work reported in this thesis can be extended in many directions. The following directions we consider of particular interest.

To evaluate the practical potential of our procedure for HPDL, an efficient implementation is required. We expect such an implementation to be comparable on hybrid logics without eventualities to state-of-the-art systems for hybrid logic [109, 98, 42]. On PDL (without nominals), we conjecture that Goré and Widmann's [93, 200] worst-case optimal procedure will be more efficient than our approach since our approach is not worst-case optimal.

To evaluate our procedure on HPDL, a base of benchmark formulas is needed. Ideally, the formulas should come from practically relevant problems modeled in HPDL. However, we are not aware of any application areas that would currently exploit the full power of HPDL. Identifying such areas and generating practical problems that can be solved by state-of-the-art or future systems is a task of great interest and practical importance.

Another interesting direction for further research is extending the procedures in this thesis to other expressive logics. We believe that our incremental graph procedure for K* can be extended to logics with converse, possibly following Goré and Widmann's approach in [94, 95, 200]. We presently do not know if our tree procedures for H* and HPDL can be extended to deal with converse.

We conjecture that our approach to HPDL can be adapted to the temporal logics UB [22] and CTL [44], and possibly even to hybrid extensions of these logics. An important difference of UB and CTL from K* and PDL is that the former logics have two kinds of eventualities, represented in UB by formulas of the form EFs and AFs. While formulas of the form EFs are essentially equivalent to eventualities in K*, formulas AFs have no counterpart in K* or PDL. A formula AFs holds at a state if every infinite path starting in that state contains a state satisfying s. Dealing with formulas of the form AFs will require a significant extension of our theory. It is not clear to us what happens if we try to extend our approach to even more expressive logics like CTL* [60].

Finally, our work leaves open the question if it is possible to obtain decision procedures for modal logics with nominals and eventualities that are both incremental and worst-case optimal. Our results in § 3.5 suggest that it is impossible to give a pruning relation that is both confluent and complete in the presence of nominals. Still, it is conceivable that an incremental decision procedure may be obtained by looking at nonconfluent pruning relations.

Bibliography

[1] Abate, P., and Goré, R. The tableau workbench. In *M4M-5* (2009), C. Areces and S. Demri, Eds., vol. 231 of *Electr. Notes Theor. Comput. Sci.*, Elsevier, pp. 55–67.

[2] Abate, P., Goré, R., and Widmann, F. One-pass tableaux for computational tree logic. In *LPAR 2007* (2007), N. Dershowitz and A. Voronkov, Eds., vol. 4790 of *LNCS*, Springer, pp. 32–46.

[3] Abate, P., Goré, R., and Widmann, F. An on-the-fly tableau-based decision procedure for PDL-satisfiability. In *M4M-5* (2009), C. Areces and S. Demri, Eds., vol. 231 of *Electr. Notes Theor. Comput. Sci.*, Elsevier, pp. 191–209.

[4] Areces, C., Blackburn, P., and Marx, M. A road-map on complexity for hybrid logics. In *CSL '99* (1999), J. Flum and M. Rodríguez-Artalejo, Eds., vol. 1683 of *LNCS*, Springer, pp. 307–321.

[5] Areces, C., Blackburn, P., and Marx, M. The computational complexity of hybrid temporal logics. *L. J. IGPL 8*, 5 (2000), 653–679.

[6] Areces, C., de Nivelle, H., and de Rijke, M. Prefixed resolution: A resolution method for modal and description logics. In *CADE-16* (1999), H. Ganzinger, Ed., vol. 1632 of *LNCS*, Springer, pp. 187–201.

[7] Areces, C., de Nivelle, H., and de Rijke, M. Resolution in modal, description and hybrid logic. *J. Log. Comput. 11*, 5 (1999), 717–736.

[8] Areces, C., and Gorín, D. Resolution with order and selection for hybrid logics. *J. Autom. Reasoning 46*, 1 (2011), 1–42.

[9] Areces, C., and Heguiabehere, J. HyLoRes 1.0: Direct resolution for hybrid logics. In *CADE-18* (2002), A. Voronkov, Ed., vol. 2392 of *LNCS*, Springer, pp. 156–160.

[10] Areces, C., and ten Cate, B. Hybrid logics. In *Handbook of Modal Logic*, P. Blackburn, J. van Benthem, and F. Wolter, Eds., vol. 3 of *Studies in Logic and Practical Reasoning*. Elsevier, 2007, pp. 821–868.

[11] Baader, F. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. Tech. Rep. RR-90-13, DFKI, 1990.

[12] Baader, F. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *IJCAI '91* (1991), J. Mylopoulos and R. Reiter, Eds., Morgan Kaufmann, pp. 446–451.

[13] Baader, F. Terminological cycles in a description logic with existential restrictions. In *IJCAI 2003* (2003), G. Gottlob and T. Walsh, Eds., Morgan Kaufmann, pp. 325–330.

[14] Baader, F., Brandt, S., and Lutz, C. Pushing the $\mathcal{EL}$ envelope. In *IJCAI 2005* (2005), L. P. Kaelbling and A. Saffiotti, Eds., Professional Book Center, pp. 364–369.

[15] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., Eds. *The Description Logic Handbook: Theory, Implementation and Applications*, 2nd ed. Cambridge University Press, 2007.

[16] Baader, F., Hladik, J., Lutz, C., and Wolter, F. From tableaux to automata for description logics. *Fund. Inform. 57*, 2–4 (2003), 247–279.

[17] Baader, F., Küsters, R., and Molitor, R. Computing least common subsumers in description logics with existential restrictions. In *IJCAI '99* (1999), T. Dean, Ed., Morgan Kaufmann, pp. 96–103.

[18] Baader, F., and Lutz, C. Description logic. In *Handbook of Modal Logic*, P. Blackburn, J. van Benthem, and F. Wolter, Eds., vol. 3 of *Studies in Logic and Practical Reasoning*. Elsevier, 2007, pp. 757–820.

[19] Baader, F., and Nipkow, T. *Term Rewriting and All That*. Cambridge University Press, 1998.

[20] Baader, F., and Nutt, W. Basic description logics. In *The Description Logic Handbook: Theory, Implementation and Applications*, F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, Eds., 2nd ed. Cambridge University Press, 2007, pp. 47–104.

[21] Beckert, B., Hähnle, R., and Schmitt, P. H., Eds. *Verification of Object-Oriented Software: The KeY Approach*, vol. 4334 of *LNCS*. Springer, 2007.

[22] Ben-Ari, M., Pnueli, A., and Manna, Z. The temporal logic of branching time. *Acta Inf. 20*, 3 (1983), 207–226.

[23] Berardi, D., Calvanese, D., De Giacomo, G., Hull, R., and Mecella, M. Automatic composition of transition-based semantic web services with messaging. In *VLDB 2005* (2005), K. Böhm, C. S. Jensen, L. M. Haas, M. L. Kersten, P.-Å. Larson, and B. C. Ooi, Eds., ACM, pp. 613–624.

[24] Berry, G., and Sethi, R. From regular expressions to deterministic automata. *Theor. Comput. Sci. 48*, 3 (1986), 117–126.

[25] Beth, E. W. Semantic entailment and formal derivability. *Mededelingen van de Koninklijke Nederlandse Akademie van Wetenschappen, Afdeling Letterkunde 18*, 13 (1955), 309–342.

[26] Blackburn, P. Tense, temporal reference, and tense logic. *J. Semantics 11*, 1–2 (1994), 83–101.

[27] Blackburn, P., de Rijke, M., and Venema, Y. *Modal Logic.* Cambridge University Press, 2001.

[28] Blackburn, P., and Seligman, J. Hybrid languages. *J. Log. Lang. Inf. 4*, 3 (1995), 251–272.

[29] Bolander, T., and Blackburn, P. Termination for hybrid tableaus. *J. Log. Comput. 17*, 3 (2007), 517–554.

[30] Bolander, T., and Braüner, T. Tableau-based decision procedures for hybrid logic. *J. Log. Comput. 16*, 6 (2006), 737–763.

[31] Bonatti, P. A., Lutz, C., Murano, A., and Vardi, M. Y. The complexity of enriched μ-calculi. In *ICALP 2006, Part II* (2006), M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, Eds., vol. 4052 of *LNCS*, Springer, pp. 540–551.

[32] Brand, S., Gennari, R., and de Rijke, M. Constraint methods for modal satisfiability. In *CSCLP 2003* (2004), K. R. Apt, F. Fages, F. Rossi, P. Szeredi, and J. Váncza, Eds., vol. 3010 of *LNCS*, Springer, pp. 66–86.

[33] Brandt, S. Polynomial time reasoning in a description logic with existential restrictions, CGI axioms, and – what else? In *ECAI 2004* (2004), R. L. de Mántaras and L. Saitta, Eds., IOS Press, pp. 298–302.

[34] Braüner, T. *Hybrid Logic and its Proof-Theory.* Springer, 2011.

[35] Brzozowski, J. A. Derivatives of regular expressions. *J. ACM 11*, 4 (1964), 481–494.

[36] Bull, R. A. An approach to tense logic. *Theoria 36*, 3 (1970), 282–300.

[37] Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., and Rosati, R. DL-Lite: Tractable description logics for ontologies. In *AAAI 2005* (2005), M. M. Veloso and S. Kambhampati, Eds., AAAI Press/The MIT Pres, pp. 602–607.

[38] Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., and Rosati, R. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. Autom. Reasoning 39*, 3 (2007), 385–429.

[39] Cerrito, S., and Cialdea Mayer, M. An efficient approach to nominal equalities in hybrid logic tableaux. *J. Appl. Non-Class. Log. 20*, 1-2 (2010), 39–61.

[40] Cerrito, S., and Cialdea Mayer, M. Nominal substitution at work with the global and converse modalities. In *Advances in Modal Logic* (2010), L. Beklemishev, V. Goranko, and V. Shehtman, Eds., vol. 8, College Publications, pp. 59–76.

[41] Chagrov, A., and Zakharyaschev, M. *Modal Logic*. Oxford University Press, 1997.

[42] Cialdea Mayer, M., and Cerrito, S. Herod and Pilate: Two tableau provers for basic hybrid logic. In *IJCAR 2010* (2010), J. Giesl and R. Hähnle, Eds., vol. 6173 of *LNCS*, Springer, pp. 255–262.

[43] Clarke, E. M., and Draghicescu, I. A. Expressibility results for linear-time and branching-time logics. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency* (1989), J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, Eds., vol. 354 of *LNCS*, Springer, pp. 428–437.

[44] Clarke, E. M., and Emerson, E. A. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logics of Programs* (1982), D. Kozen, Ed., vol. 131 of *LNCS*, Springer, pp. 52–71.

[45] Cîrstea, C., Kupke, C., and Pattinson, D. EXPTIME tableaux for the coalgebraic μ-calculus. In *CSL 2009* (2009), vol. 5771 of *LNCS*, pp. 179–193.

[46] D'Agostino, M. Are tableaux an improvement on truth-tables? cut-free proofs and bivalence. *Journal of Logic, Language and Information 1*, 3 (September 1992), 235–252.

[47] Dam, M. CTL* and ECTL* as fragments of the modal μ-calculus. *Theor. Comput. Sci. 126*, 1 (1994), 77–96.

[48] Damas, L., and Milner, R. Principal type-schemes for functional programs. In *POPL '80* (1982), ACM, pp. 207–212.

[49] Davis, M., Logemann, G., and Loveland, D. A machine program for theorem proving. *Commun. ACM 5*, 7 (1962), 394–397.

[50] Davis, M., and Putnam, H. A computing procedure for quantification theory. *J. ACM 7*, 3 (1960), 201–215.

[51] De Giacomo, G. Eliminating "converse" from converse PDL. *J. Log. Lang. Inf. 5*, 2 (1996), 193–208.

[52] De Giacomo, G., and Massacci, F. Combining deduction and model checking into tableaux and algorithms for converse-PDL. *Inf. Comput. 162*, 1–2 (2000), 117–137.

[53] de Nivelle, H., Schmidt, R. A., and Hustadt, U. Resolution-based methods for modal logics. *L. J. IGPL 8*, 3 (2000), 265–292.

[54] de Rijke, M. The modal logic of inequality. *J. Symb. Log. 57*, 2 (June 1992), 566–584.

[55] Dechter, R. *Constraint Processing*. Morgan Kaufmann, 2003.

[56] Emerson, E. A. Automata, tableaux and temporal logics: Extended abstract. In *Logics of Programs* (1985), R. Parikh, Ed., vol. 193 of *LNCS*, Springer, pp. 79–88.

[57] Emerson, E. A. Temporal and modal logic. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed., vol. B: Formal Models and Semantics. Elsevier, 1990, pp. 995–1072.

[58] Emerson, E. A., and Clarke, E. M. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Programming 2*, 3 (1982), 241–266.

[59] Emerson, E. A., and Halpern, J. Y. Decision procedures and expressiveness in the temporal logic of branching time. *J. Comput. System Sci. 30*, 1 (1985), 1–24.

[60] Emerson, E. A., and Halpern, J. Y. "Sometimes" and "not never" revisited: On branching versus linear time temporal logic. *J. ACM 33*, 1 (1986), 151–178.

[61] Emerson, E. A., and Jutla, C. S. The complexity of tree automata and logics of programs. *SIAM J. Comput. 29*, 1 (1999), 132–158.

[62] Emerson, E. A., and Sistla, A. P. Deciding full branching time logic. *Inf. Comput. 61*, 3 (1984), 175–201.

[63] Enjalbert, P., and Fariñas del Cerro, L. Modal resolution in clausal form. *Theor. Comput. Sci. 65*, 1 (1989), 1–33.

[64] Fattorosi-Barnaba, M., and De Caro, F. Graded modalities. I. *Stud. Log. 44*, 2 (1985), 197–221.

[65] Fine, K. In so many possible worlds. *Notre Dame J. Form. Log. 13*, 4 (1972), 516–520.

[66] Fischer, M. J., and Ladner, R. E. Propositional modal logic of programs. In *STOC '77* (1977), ACM, pp. 286–294.

[67] Fischer, M. J., and Ladner, R. E. Propositional dynamic logic of regular programs. *J. Comput. System Sci.* (1979), 194–211.

[68] Fitting, M. Tableau methods of proof for modal logics. *Notre Dame J. Form. Log. 13*, 2 (1972), 237–247.

[69] Fitting, M. Model existence theorems for modal and intuitionistic logics. *J. Symb. Log. 38*, 4 (1973), 613–627.

[70] Fitting, M. *Proof Methods for Modal and Intuitionistic Logics*. D. Reidel, 1983.

[71] Fitting, M. Modal proof theory. In *Handbook of Modal Logic*, P. Blackburn, J. van Benthem, and F. Wolter, Eds., vol. 3 of *Studies in Logic and Practical Reasoning*. Elsevier, 2007, pp. 85–138.

[72] Franceschet, M., and de Rijke, M. Model checking hybrid logics (with an application to semistructured data). *J. Appl. Log. 4*, 3 (2006), 279–304.

[73] Friedmann, O., and Lange, M. A solver for modal fixpoint logics. In *M4M-6* (2010), T. Bolander and T. Braüner, Eds., vol. 262 of *Electr. Notes Theor. Comput. Sci.*, Elsevier, pp. 99–111.

[74] Friedmann, O., and Lange, M. The modal μ-calculus caught off guard. In *TABLEAUX 2011* (2011), K. Brünnler and G. Metcalfe, Eds., vol. 6793 of *LNCS*, Springer, pp. 149–163.

[75] Gabbay, D. M., Pnueli, A., Shelah, S., and Stavi, J. On the temporal analysis of fairness. In *POPL '80* (1980), ACM, pp. 163–173.

[76] Gargov, G., and Goranko, V. Modal logic with names. *Journal of Philosophical Logic 22* (1993), 607–636.

[77] Gaschnig, J. *Performance Measurement and Analysis of Certain Search Algorithms.* PhD thesis, Carnegie Mellon University, 1979.

[78] Gentzen, G. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift 39* (1935), 176–210 and 405–431. In German.

[79] Gentzen, G. Investigation into logical deduction. In *The Collected Papers of Gerhard Gentzen*, M. E. Szabo, Ed. North-Holland, 1969, pp. 68–131. Originally published as [78].

[80] Gerth, R., Peled, D., Vardi, M. Y., and Wolper, P. Simple on-the-fly automatic verification of linear temporal logic. In *PSTV '95* (1996), P. Dembinski and M. Sredniawa, Eds., vol. 38 of *IFIP Conference Proceedings*, Chapman & Hall, pp. 3–18.

[81] Giunchiglia, E., Giunchiglia, F., and Tacchella, A. SAT-based decision procedures for classical modal logics. *J. Autom. Reasoning 28*, 2 (2002), 143–171.

[82] Giunchiglia, E., and Tacchella, A. System description: *SAT: A platform for the development of modal decision procedures. In *CADE-17* (2000), D. A. McAllester, Ed., vol. 1831 of *LNCS*, Springer, pp. 291–296.

[83] Giunchiglia, E., and Tacchella, A. A subset-matching size-bounded cache for testing satisfiability in modal logics. *Ann. Math. Artif. Intell. 33*, 1 (2001), 39–67.

[84] Giunchiglia, F., and Sebastiani, R. Building decision procedures for modal logics from propositional decision procedures: The case study of modal K(m). *Inf. Comput. 162*, 1-2 (2000), 158–178.

[85] Goldblatt, R. Mathematical modal logic: A view of its evolution. *J. Appl. Log. 1*, 5–6 (2003), 309–392.

[86] GORANKO, V. Modal definability in enriched languages. *Notre Dame J. Form. Log. 31*, 1 (1990), 81–105.

[87] GORANKO, V. Temporal logic with reference pointers. In *ICTL '94* (1994), D. M. Gabbay and H. J. Ohlbach, Eds., vol. 827 of *LNCS*, Springer, pp. 133–148.

[88] GORANKO, V., AND OTTO, M. Model theory of modal logic. In *Handbook of Modal Logic*, P. Blackburn, J. van Benthem, and F. Wolter, Eds., vol. 3 of *Studies in Logic and Practical Reasoning*. Elsevier, 2007, pp. 249–329.

[89] GORANKO, V., AND PASSY, S. Using the universal modality: Gains and questions. *J. Log. Comput. 2*, 1 (1992), 5–30.

[90] GORÉ, R. Tableau methods for modal and temporal logics. In *Handbook of Tableau Methods*, M. D'Agostino, D. M. Gabbay, R. Hähnle, and J. Posegga, Eds. Kluwer Academic Publishers, 1999, pp. 297–396.

[91] GORÉ, R., AND NGUYEN, L. A. EXPTIME tableaux with global caching for description logics with transitive roles, inverse roles and role hierarchies. In *TABLEAUX 2007* (2007), N. Olivetti, Ed., vol. 4548 of *LNCS*, Springer, pp. 133–148.

[92] GORÉ, R., AND POSTNIECE, L. An experimental evaluation of global caching for $\mathcal{ALC}$: System description. In *IJCAR 2008* (2008), A. Armando, P. Baumgartner, and G. Dowek, Eds., vol. 5195 of *LNCS*, Springer, pp. 299–305.

[93] GORÉ, R., AND WIDMANN, F. An optimal on-the-fly tableau-based decision procedure for PDL-satisfiability. In *CADE-22* (2009), R. A. Schmidt, Ed., vol. 5663 of *LNCS*, Springer, pp. 437–452.

[94] GORÉ, R., AND WIDMANN, F. Sound global state caching for $\mathcal{ALC}$ with inverse roles. In *TABLEAUX 2009* (2009), M. Giese and A. Waaler, Eds., vol. 5607 of *LNCS*, Springer, pp. 205–219.

[95] GORÉ, R., AND WIDMANN, F. Optimal tableaux for propositional dynamic logic with converse. In *IJCAR 2010* (2010), J. Giesl and R. Hähnle, Eds., vol. 6173 of *LNCS*, Springer, pp. 225–239.

[96] GORÍN, D. *Técnicas de razonamiento automático para lógicas híbridas (Automated Reasoning Techniques for Hybrid Logics)*. PhD thesis, Universidad de Buenos Aires, 2009.

[97] GÖTZMANN, D. *Spartacus: A Tableau Prover for Hybrid Logic*. M.Sc. thesis, Saarland University, 2009.

[98] GÖTZMANN, D., KAMINSKI, M., AND SMOLKA, G. Spartacus: A tableau prover for hybrid logic. In *M4M-6* (2010), T. Bolander and T. Braüner, Eds., vol. 262 of *Electr. Notes Theor. Comput. Sci.*, Elsevier, pp. 127–139.

[99] Haarslev, V., and Möller, R. RACER system description. In *IJCAR 2001* (2001), R. Goré, A. Leitsch, and T. Nipkow, Eds., vol. 2083 of *LNCS*, Springer, pp. 701–705.

[100] Haarslev, V., and Möller, R. High performance reasoning with very large knowledge bases: A practical case study. In *IJCAI 2001* (2001), B. Nebel, Ed., Morgan Kaufmann, pp. 161–168.

[101] Halpern, J. Y., and Moses, Y. A guide to completeness and complexity for modal logics of knowledge and belief. *Artif. Intell. 54* (1992), 319–379.

[102] Harel, D. *First-Order Dynamic Logic*, vol. 68 of *LNCS*. Springer, 1979.

[103] Harel, D. Dynamic logic. In *Handbook of Philosophical Logic*, D. M. Gabbay and F. Guenthner, Eds., vol. 2. D. Reidel, 1984, pp. 497–604.

[104] Harel, D., Kozen, D., and Tiuryn, J. *Dynamic Logic*. The MIT Press, 2000.

[105] Hintikka, K. J. J. Form and content in quantification theory. Two papers on symbolic logic. *Acta Philosophica Fennica 8* (1955), 7–55.

[106] Hoare, C. A. R. An axiomatic basis for computer programming. *Commun. ACM 12*, 10 (1969), 576–580,583.

[107] Hoffmann, G. Lightweight hybrid tableaux. *J. Appl. Log. 8*, 4 (2010), 397–408.

[108] Hoffmann, G. *Tâches de raisonnement en logiques hybrides*. PhD thesis, Université Henri Poincaré, Nancy 1, 2010.

[109] Hoffmann, G., and Areces, C. HTab: A terminating tableaux system for hybrid logic. In *M4M-5* (2009), C. Areces and S. Demri, Eds., vol. 231 of *Electr. Notes Theor. Comput. Sci.*, Elsevier, pp. 3–19.

[110] Hopcroft, J. E., Motwani, R., and Ullman, J. D. *Introduction to Automata Theory, Languages, and Computation*, 3rd ed. Prentice Hall, 2006.

[111] Horrocks, I., Hustadt, U., Sattler, U., and Schmidt, R. A. Computational modal logic. In *Handbook of Modal Logic*, P. Blackburn, J. van Benthem, and F. Wolter, Eds., vol. 3 of *Studies in Logic and Practical Reasoning*. Elsevier, 2007, pp. 181–245.

[112] Horrocks, I., Kutz, O., and Sattler, U. The even more irresistible $\mathcal{SROIQ}$. In *Proc. 10th Intl. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006)* (2006), P. Doherty, J. Mylopoulos, and C. A. Welty, Eds., AAAI Press, pp. 57–67.

[113] Horrocks, I., and Patel-Schneider, P. F. Optimizing description logic subsumption. *J. Log. Comput. 9*, 3 (1999), 267–293.

[114] Horrocks, I., and Sattler, U. A description logic with transitive and inverse roles and role hierarchies. *J. Log. Comput. 9*, 3 (1999), 385–410.

[115] Horrocks, I., and Sattler, U. Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic. In *IJCAI 2001* (2001), B. Nebel, Ed., Morgan Kaufmann, pp. 199–204.

[116] Horrocks, I., and Sattler, U. Decidability of $\mathcal{SHIQ}$ with complex role inclusion axioms. *Artif. Intell. 160*, 1-2 (2004), 79–104.

[117] Horrocks, I., and Sattler, U. A tableau decision procedure for $\mathcal{SHOIQ}$. *J. Autom. Reasoning 39*, 3 (2007), 249–276.

[118] Horrocks, I., Sattler, U., and Tobies, S. Practical reasoning for expressive description logics. In *LPAR'99* (1999), H. Ganzinger, D. A. McAllester, and A. Voronkov, Eds., vol. 1705 of *LNCS*, Springer, pp. 161–180.

[119] Hustadt, U., and Schmidt, R. A. Simplification and backjumping in modal tableau. In *TABLEAUX '98* (1998), H. de Swart, Ed., vol. 1397 of *LNCS*, Springer, pp. 187–201.

[120] Hustadt, U., and Schmidt, R. A. MSPASS: Modal reasoning by translation and first-order resolution. In *TABLEAUX 2000* (2000), R. Dyckhoff, Ed., vol. 1847 of *LNCS*, Springer, pp. 514–520.

[121] Hustadt, U., and Schmidt, R. A. Using resolution for testing modal satisfiability and building models. *J. Autom. Reasoning 28*, 2 (2002), 205–232.

[122] Hustadt, U., and Schmidt, R. A. A comparison of solvers for propositional dynamic logic. In *PAAR-2010* (2010), B. Konev, R. A. Schmidt, and S. Schulz, Eds., EasyChair Proceedings, pp. 60–69.

[123] Jungteerapanich, N. A tableau system for the modal μ-calculus. In *TABLEAUX 2009* (2009), M. Giese and A. Waaler, Eds., vol. 5607 of *LNCS*, Springer, pp. 220–234.

[124] Kaminski, M., Schneider, S., and Smolka, G. Terminating tableaux for graded hybrid logic with global modalities and role hierarchies. *Log. Methods Comput. Sci. 7*, 1:5 (2011).

[125] Kaminski, M., and Smolka, G. Terminating tableaux for hybrid logic with the difference modality and converse. In *IJCAR 2008* (2008), A. Armando, P. Baumgartner, and G. Dowek, Eds., vol. 5195 of *LNCS*, Springer, pp. 210–225.

[126] Kaminski, M., and Smolka, G. Hybrid tableaux for the difference modality. In *M4M-5* (2009), C. Areces and S. Demri, Eds., vol. 231 of *Electr. Notes Theor. Comput. Sci.*, Elsevier, pp. 241–257.

[127] Kaminski, M., and Smolka, G. Terminating tableau systems for hybrid logic with difference and converse. *J. Log. Lang. Inf. 18*, 4 (2009), 437–464.

[128] Kaminski, M., and Smolka, G. Terminating tableaux for hybrid logic with eventualities. In *IJCAR 2010* (2010), J. Giesl and R. Hähnle, Eds., vol. 6173 of *LNCS*, Springer, pp. 240–254.

[129] Kaminski, M., and Smolka, G. Terminating tableaux for $\mathcal{SOQ}$ with number restrictions on transitive roles. In *TCS 2010* (2010), C. S. Calude and V. Sassone, Eds., vol. 323 of *IFIP AICT*, Springer, pp. 213–228.

[130] Kaplan, D. M. Regular expressions and the equivalence of programs. *J. Comput. System Sci. 3*, 4 (1969), 361–386.

[131] Kazakov, Y., and Pratt-Hartmann, I. A note on the complexity of the satisfiability problem for graded modal logics. In *LICS 2009* (2009), IEEE Computer Society Press, pp. 407–416.

[132] Kesten, Y., Manna, Z., McGuire, H., and Pnueli, A. A decision algorithm for full propositional temporal logic. In *CAV'93* (1993), C. Courcoubetis, Ed., vol. 697 of *LNCS*, Springer, pp. 97–109.

[133] Kowalski, R. *Logic for Problem Solving.* North-Holland, 1979.

[134] Koymans, R. *Specifying Message Passing and Time-Critical Systems with Temporal Logic*, vol. 651 of *LNCS.* Springer, 1992.

[135] Kozen, D. Results on the propositional μ-calculus. *Theor. Comput. Sci. 27* (1983), 333–354.

[136] Kozen, D. Kleene algebra with tests and commutativity conditions. In *TACAS'96* (1996), T. Margaria and B. Steffen, Eds., vol. 1055 of *LNCS*, Springer, pp. 14–33.

[137] Kozen, D., and Smith, F. Kleene algebra with tests: Completeness and decidability. In *CSL'96* (1996), D. van Dalen and M. Bezem, Eds., vol. 1258 of *LNCS*, Springer, pp. 244–259.

[138] Kozen, D., and Tiuryn, J. Logics of programs. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed., vol. B: Formal Models and Semantics. Elsevier, 1990, pp. 789–840.

[139] Kripke, S. A. A completeness theorem in modal logic. *J. Symb. Log. 24*, 1 (1959), 1–14.

[140] Kripke, S. A. Semantical analysis of modal logic I: Normal modal propositional calculi. *Z. Math. Logik Grundlagen Math. 9* (1963), 67–96.

[141] Kumar, V. Algorithms for constraint satisfaction problems: A survey. *AI Magazine 13*, 1 (1992), 32–44.

[142] Ladner, R. E. The computational complexity of provability in systems of modal propositional logic. *SIAM J. Comput. 6*, 3 (1977), 467–480.

[143] Lange, M. Satisfiability and completeness of converse-PDL replayed. In *KI 2003* (2003), A. Günter, R. Kruse, and B. Neumann, Eds., vol. 2821 of *LNCS*, Springer, pp. 79–82.

[144] Lemmon, E. J., and Scott, D. *The 'Lemmon Notes': An Introduction to Modal Logic.* Blackwell, 1977.

[145] Lewis, C. I. *A Survey of Symbolic Logic.* University of California Press, 1918.

[146] Massacci, F. Single step tableaux for modal logics. *J. Autom. Reasoning 24*, 3 (2000), 319–364.

[147] Minsky, M. A framework for representing knowledge. In *The Psychology of Computer Vision*, P. H. Winston, Ed. McGraw-Hill, 1975, pp. 211–277.

[148] Mints, G. Resolution calculi for modal logics. *Amer. Math. Soc. Transl. 143* (1989), 1–14.

[149] Nadel, B. A. Tree search and arc consistency in constraint satisfaction algorithms. In *Search in Artificial Intelligence*, L. N. Kanal and V. Kumar, Eds. Springer, 1988, pp. 287–342.

[150] Nguyen, L. A., and Szałas, A. An optimal tableau decision procedure for converse-PDL. In *KSE 2009* (2009), N. T. Nguyen, T. D. Bui, E. Szczerbicki, and N. B. Nguyen, Eds., IEEE Computer Society Press, pp. 207–214.

[151] Nguyen, L. A., and Szałas, A. Checking consistency of an ABox w.r.t. global assumptions in PDL. *Fund. Inform. 102*, 1 (2010), 97–113.

[152] Pan, G., Sattler, U., and Vardi, M. Y. BDD-based decision procedures for the modal logic K. *J. Appl. Non-Class. Log. 16*, 1-2 (2006), 169–208.

[153] Passy, S., and Tinchev, T. PDL with data constants. *Inf. Process. Lett. 20*, 1 (1985), 35–41.

[154] Passy, S., and Tinchev, T. Quantifiers in combinatory PDL: Completeness, definability, incompleteness. In *FCT '85* (1985), L. Budach, Ed., vol. 199 of *LNCS*, pp. 512–519.

[155] Passy, S., and Tinchev, T. An essay in combinatory dynamic logic. *Inf. Comput. 93*, 2 (1991), 263–332.

[156] Perrin, D. Finite automata. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed., vol. B: Formal Models and Semantics. Elsevier, 1990, pp. 1–57.

[157] Pierce, B. C. *Types and Programming Languages.* The MIT Press, 2002.

[158] Pnueli, A. The temporal logic of programs. In *FOCS '77* (1977), IEEE Computer Society Press, pp. 46–57.

[159] Pratt, V. R. Semantical considerations on Floyd-Hoare logic. In *FOCS '76* (1976), IEEE Computer Society Press, pp. 109–121.

[160] Pratt, V. R. Models of program logics. In *Proc. 20th Annual Symp. on Foundations of Computer Science (FOCS'79)* (1979), IEEE Computer Society Press, pp. 115–122.

[161] Pratt, V. R. A near-optimal method for reasoning about action. *J. Comput. System Sci. 20*, 2 (1980), 231–254.

[162] Prendinger, H., and Schurz, G. Reasoning about action and change: A dynamic logic approach. *J. Log. Lang. Inf. 5*, 2 (1996), 209–245.

[163] Prior, A. N. *Past, Present and Future.* Oxford University Press, 1967.

[164] Prior, A. N. *Papers on Time and Tense, New Edition.* Oxford University Press, 2003. Edited by Per Hasle, Peter Øhrstrøm, Torben Braüner and Jack Copeland.

[165] Prosser, P. Hybrid algorithms for the constraint satisfaction problem. *Comput. Intell. 9*, 3 (1993), 268–299.

[166] Quillian, M. R. Word concepts: A theory and simulation of some basic capabilities. *Behavioural Science 12*, 5 (1967), 410–430.

[167] Rautenberg, W. *Klassische und nichtklassische Aussagenlogik.* Vieweg, 1983. In German.

[168] Rautenberg, W. Modal tableau calculi and interpolation. *J. Philos. Log. 12*, 4 (1983), 403–423.

[169] Reynolds, M. A tableau for CTL*. In *FM 2009* (2009), A. Cavalcanti and D. Dams, Eds., vol. 5850 of *LNCS*, Springer, pp. 403–418.

[170] Sain, I. Is 'some other time' sometimes better than 'sometime' for proving partial correctness of programs? *Stud. Log. 47* (1988), 279–301.

[171] Schild, K. A correspondence theory for terminological logics: Preliminary report. In *IJCAI '91* (1991), J. Mylopoulos and R. Reiter, Eds., Morgan Kaufmann, pp. 466–471.

[172] Schmidt, R. A. Decidability by resolution for propositional modal logics. *J. Autom. Reasoning 22*, 4 (1999), 379–396.

[173] Schmidt, R. A., and Hustadt, U. The axiomatic translation principle for modal logic. *ACM Trans. Comput. Log. 8*, 4 (2007).

[174] Schmidt-Schauss, M., and Smolka, G. Attributive concept descriptions with complements. *Artif. Intell. 48*, 1 (1991), 1–26.

[175] Schneider, S. *Terminating Tableaux for Modal Logic with Transitive Closure.* Bachelor's thesis, Saarland University, 2009.

[176] SCHRIJVER, A. *Theory of Linear and Integer Programming.* John Wiley & Sons, 1986.

[177] SCHWENDIMANN, S. A new one-pass tableau calculus for PLTL. In *TABLEAUX '98* (1998), H. de Swart, Ed., vol. 1397 of *LNCS*, Springer, pp. 277–291.

[178] SEBASTIANI, R., AND VESCOVI, M. Automated reasoning in modal and description logics via SAT encoding: The case study of $K_m/\mathcal{ALC}$-satisfiability. *J. Artif. Intell. Res. 35* (2009), 343–389.

[179] SEGERBERG, K. A note on the logic of elsewhere. *Theoria 46*, 2-3 (1980), 183–187.

[180] SINZ, C., LUMPP, T., SCHNEIDER, J., AND KÜCHLIN, W. Detection of dynamic execution errors in IBM system automation's rule-based expert system. *Information and Software Technology 44*, 14 (2002), 857–873.

[181] SIPSER, M. *Introduction to the Theory of Computation*, 2nd ed. Course Technology, 2005.

[182] SIRIN, E., PARSIA, B., GRAU, B. C., KALYANPUR, A., AND KATZ, Y. Pellet: A practical OWL-DL reasoner. *J. Web Sem. 5*, 2 (2007), 51–53.

[183] SISTLA, A. P., AND CLARKE, E. M. The complexity of propositional linear temporal logics. *J. ACM 32*, 3 (1985), 733–749.

[184] SMULLYAN, R. M. *First-Order Logic.* Dover, 1995. First published in 1968 by Springer.

[185] SPAAN, E. *Complexity of Modal Logics.* PhD thesis, ILLC, University of Amsterdam, 1993.

[186] THORNTON, C., AND DU BOULAY, B. *Artificial Intelligence Through Search.* Springer, 1992.

[187] TOBIES, S. PSPACE reasoning for graded modal logics. *J. Log. Comput. 11*, 1 (2001), 85–106.

[188] TSARKOV, D., AND HORROCKS, I. Ordering heuristics for description logic reasoning. In *Proc. 19th Intl. Joint Conf. on Artificial Intelligence (IJCAI'05)* (2005), L. P. Kaelbling and A. Saffiotti, Eds., Professional Book Center, pp. 609–614.

[189] TSARKOV, D., AND HORROCKS, I. FaCT++ description logic reasoner: System description. In *IJCAR 2006* (2006), U. Furbach and N. Shankar, Eds., vol. 4130 of *LNCS*, Springer, pp. 292–297.

[190] TSARKOV, D., HORROCKS, I., AND PATEL-SCHNEIDER, P. F. Optimizing terminological reasoning for expressive description logics. *J. Autom. Reasoning 39*, 3 (2007), 277–316.

[191] Tzakova, M. Tableau calculi for hybrid logics. In *TABLEAUX '99* (1999), N. V. Murray, Ed., vol. 1617 of *LNCS*, Springer, pp. 278–292.

[192] van der Hoek, W., and de Rijke, M. Counting objects. *J. Log. Comput. 5*, 3 (1995), 325–345.

[193] Vardi, M. Y. The taming of converse: Reasoning about two-way computations. In *Logics of Programs* (1985), R. Parikh, Ed., vol. 193 of *LNCS*, Springer, pp. 413–423.

[194] Vardi, M. Y., and Stockmeyer, L. Improved upper and lower bounds for modal logics of programs. In *STOC '85* (1985), ACM, pp. 240–251.

[195] Vardi, M. Y., and Wolper, P. Automata-theoretic techniques for modal logics of programs. *J. Comput. System Sci. 32*, 2 (1986), 183–221.

[196] von Wright, G. H. A modal logic of place. In *The Philosophy of Nicholas Rescher: Discussion and Replies*, E. Sosa, Ed., vol. 15 of *Philosophical Studies Series*. D. Reidel, 1979, pp. 65–73.

[197] Voronkov, A. Deciding K using ꓘ. In *KR 2000* (2000), A. G. Cohn, F. Giunchiglia, and B. Selman, Eds., Morgan Kaufmann, pp. 198–209.

[198] Voronkov, A. How to optimize proof-search in modal logics: New methods of proving redundancy criteria for sequent calculi. *ACM Trans. Comput. Log. 2*, 2 (2001), 182–215.

[199] Weidenbach, C., Schmidt, R. A., Hillenbrand, T., Rusev, R., and Topic, D. System description: SPASS version 3.0. In *CADE-21* (2007), F. Pfenning, Ed., vol. 4603 of *LNCS*, Springer, pp. 514–520.

[200] Widmann, F. *Tableaux-based Decision Procedures for Fixed Point Logics*. PhD thesis, Australian National University, 2010.

[201] Wolper, P. The tableau method for temporal logic: An overview. *Logique et Analyse 110-111* (1985), 119–136.