A MODAL LOGIC FORMALISM FOR EFFICIENT KNOWLEDGE REPRESENTATION AND REASONING

by

Jens Happe

M.Sc., Simon Fraser University, 1995

Dipl.Math, Technische Universität Clausthal, Germany, 1995

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
Spring 2005

APPROVAL

Name: Jens Happe

Degree: Doctor of Philosophy

Title of thesis: A modal logic formalism for efficient knowledge representation and reasoning

Examining Committee: Dr. Eugenia Ternovska
Chair

Dr. James Delgrande, Senior Supervisor

Dr. David Mitchell, Supervisor

Dr. Veronica Dahl, Supervisor

Dr. Ray Jennings, SFU Examiner

Dr. Bruce Spencer, External Examiner,
University of New Brunswick, N.B.

Date Approved: March 15, 2005

SIMON FRASER UNIVERSITY

PARTIAL COPYRIGHT LICENCE

Abstract

Modal and modal-like logics have become the focus of renewed attention in the field of knowledge representation and automated reasoning. They provide a good middle-ground between expressiveness and complexity, which makes them suitable candidates for representation of ontologies on the Semantic Web. Expressive modal logics are well-known to be decidable but intractable. This problem has been addressed mostly by optimizations to improve the efficiency of reasoners on top of existing formalisms and algorithms, which have largely been unchanged since the days of Kripke.

This work addresses the problem of intractability by providing an improved definition of models. Rather than on a per-world basis as in Kripke models, truth assignments are specified over set of worlds addressed by labels with variables. The set of worlds represented by a label is the set of all its ground instances. This allows satisfying models to be stored in a more compact form. We prove that our representation of models and formulas preserves the satisfiability of formulas.

Our model-finding algorithm can be understood as an extension of nogood reasoning for propositional logic, whereas the nogoods are parametrized by labels, i.e. sets of worlds in which they apply. Nogoods ensure termination of the algorithm. The search strategy is general enough to accommodate tableau-style or DPLL-style branching. The algorithm is specified in sufficient detail so as to make it readily implementable, although we do not actually provide an implementation.

For simplicity, we describe our theory only on the modal logic **K**. An adaptation to multi-modal logics, the description logic $\mathcal{ALC}$, Quantified Boolean Formulas, and Propositional Dynamic Logic is straightforward; we also outline how the approach extends to logics with global axioms, such as reflexivity, transitivity, symmetry etc. We also outline the connection with recent advances of tableau- and DPLL-style methods in first-order logic.

Acknowledgments

I would like to thank my Senior Supervisor, James Delgrande, for his support in my transition into the field of logic and reasoning and through the years of my Ph.D. program, and for the space he provided for me to develop my abilities and ideas into original research. Likewise, I express my gratitude to the members of my committee, who offered me much advice on questions which were out of my line of research, and helped me cope with the new intellectual and emotional challenges of doing independent work.

Due to the nature of him being my external examiner, I have only known Bruce Spencer for a few weeks at this point. But even within this short time span, he has broadened my horizon on my research field and shown me future directions and possibilities.

Of the many other researchers in the field I have been honoured to interact with, I wish to mention Ian Horrocks who has given me much input on my work through the KR 2004 Doctoral Consortium, and Peter Patel-Schneider who, unwittingly, through his 1998 visit at Simon Fraser University has invoked in me the passion for efficient automated reasoning.

I would also like to acknowledge my lab colleagues, first and foremost Diana Cukierman who I have had the pleasure to share an office with for most of my program, and who has been my forerunner and role model in her own walk towards her Ph.D. degree and beyond.

I am deeply grateful to Jan Walls, Director of the SFU Centre for International Communication, for supporting me financially and for nurturing my appetite for cross-cultural dialogue and philosophical questions, well beyond the boundaries of formal languages and logics.

I cannot name all the students, staff, and faculty at SFU who have encouraged, challenged, or otherwise motivated me to carry on through my studies at SFU. My experience has been one of mutual giving and receiving, beyond the classroom, research lab, or other concrete walls on this campus. But over all these, I thank my beloved wife, Qing Qian, for walking all the highs and lows with me, rejoicing and suffering through the process as much as myself.

Contents

Approval ii

Abstract iii

Acknowledgments iv

Contents v

List of Figures viii

1 Introduction 1

2 Extensions of Propositional Logic 11

2.1 Formal Definition 13

2.2 Classical Propositional Logic 31

2.3 Complexity Results 36

3 The Modal Logic $\mathbf{K_{NNF}}$ 41

3.1 Kripke Models 42

3.2 Formal Definition of $\mathbf{K_{NNF}}$ 43

3.3 Characterizations of Satisfiability 44

3.4 Strict Kripke Models 46

3.5 Complexity of Reasoning Tasks in $\mathbf{K_{NNF}}$ 48

4 Labelled Formulas 51

4.1 Introduction and Motivation 51

4.2 Simple Labels 57

4.3 The Logic of Labelled Formulas 60
4.4 Realizability and Clashes 63
4.5 Equivalence between Kripke and Labelled Semantics 75
4.6 Complexity Results for Model Checking 88
4.7 The Complexity of Reasoning with Labelled Formulas 100
4.8 Non-strict Kripke Models and Labellings 114

5 Strong models for $\mathcal{LBL}$ 122
5.1 Introduction and Motivation 122
5.2 Complex Labels 128
5.3 Exception-Generated Instances 137
5.4 Realizability Revisited 147
5.5 Weak and Strong Satisfiability 157
5.6 Talking about Utopia 162
5.7 From Weak to Strong Models and Back 166

6 A Model-Finding Algorithm 183
6.1 Introduction and Motivation 183
6.2 More Normal Forms 188
6.3 Original Assertions 198
6.4 Branching 201
6.5 Nogoods 210
6.6 Standing Inconsistencies 226
6.7 Realizability Propagation 234
6.8 Failed Clauses and Nogood Merging 238
6.9 The Main Loop 241
6.10 Soundness and Correctness 246
6.11 Termination and Completeness 250
6.12 Remarks on Implementability 256
6.13 Refinements 259

7 Discussion and Conclusion 263
7.1 Contributions of this Work 263
7.2 Comparison with Existing Approaches and Techniques 266

7.3 Directions for Future Research 272

A Tableaux and Optimizations **276**

A.1 General Definitions 277

A.2 Tableau Calculi for $\mathbf{K_{NNF}}$ 283

A.3 Soundness and Completeness 286

A.4 Tableaux and Kripke Models 292

Bibliography **300**

Glossary of Notation **309**

List of Figures

3.1 *A strict model, non-strict tree model, and cyclic model for* $\Phi = \{\Diamond\, p, \Diamond\, q\}$ . . 47
3.2 *A strict model (left) and a DAG model (right) for* $\Phi = \{\Diamond\, p, \Diamond\, \neg p, \Box \Diamond\, q\}$. . 48

4.1 The $n \times n \times n$ cube of worlds in Example 4.1.1. 53
4.2 The full Kripke model and labellings for Example 4.1.1. 54
4.3 *A strict and a cyclic model for* $\{p, \Diamond\, p, \Box \Diamond\, p, \ldots, \Box^{k-1} \Diamond\, p\}$ 115
4.4 *A DAG model for* $\{\Diamond\, p_1, \Diamond\, \neg p_1, \Box \Diamond\, p_2, \Box \Diamond\, \neg p_2, \ldots, \Box^{k-1} \Diamond\, p_k, \Box^{k-1} \Diamond\, \neg p_k\}$.116
4.5 *A "ladder graph" model for* Φ *in Example 4.8.5.* 117
4.6 *A cyclic model for* Φ *in Example 4.8.5.* 118
4.7 *A model for* Φ *in Example 4.8.7, for* $k = 3$. 119

5.1 The sets of cubes satisfying $y = 1$ and falsifying $y = 1$, respectively. 123
5.2 The label $*** - \{1**, *1*, **1\}$, expressed with simple labels 124
5.3 *How* $M_{\langle\sigma\rangle}$ *and* $M_{\langle\tilde{\sigma}\rangle}$ *are formed from some set* M*, for* $\sigma \sqsubseteq \tilde{\sigma}$. 138
5.4 The logics $\mathcal{CBL}_w$, $\mathcal{CBL}_x$, and $\mathcal{CBL}_s$, and how their models can be converted. 166
5.5 The label $*** - \{111\}$, expressed with simple labels. 173

6.1 *A Nassi-Shneiderman flow diagram of the algorithm* 242

A.1 *A minimal model for* $\Phi = \{p, \Diamond\, (\neg p \wedge q), \Diamond \Diamond\, (\neg p \wedge \neg q)\}$. 293
A.2 *A strict model and the model generated by* S *for the set* Φ *in Example A.4.7* . 299

Chapter 1

Introduction

Dicit ei Pilatus: "Quid est veritas?" *"What is truth?", Pilate asked Him.*

The Bible, John 18:38

Overview Over the last 15 years, expressive but decidable logics have received renewed attention. Though not researched as actively as first-order logic or classical propositional logic, they do fill an important niche between these two extremes of an undecidable and a very inexpressive logic. Propositional modal logics have played a dominant role in this area even well before their complexity had been studied and before algorithms had been devised. They have been studied for their versatility and applicability in various areas of philosophy such as knowledge, belief, introspection, temporal reasoning, causalities, conditional reasoning, to mention but a few.

Today, these logics also appeal also to an entirely different community, that of automated reasoning. First, as mentioned, they are known to be strictly more expressive (and of higher complexity class) than propositional logic, while still decidable. Secondly, model-based semantics and reasoning algorithms based on analytic tableaux [60], available for over forty years, have stimulated the development of automated reasoners. Thirdly, the intractability in reasoning with modal logics proved less of a hindrance in practice than it might appear in theory. While all known algorithms have exponential worst-case time complexity in terms of the problem size, the actual running time of good implementations

on practical problems today is quite satisfactory. Fourthly, modal logics[1] can be used to reason with *entities*, modelled by worlds, and *relationships*, modelled by relations between worlds, which are the building blocks of *ontologies*. It is these ontologies which are now providing the main motivation for research into even more efficient algorithms. Arguably one of the first large-scale ontologies, established in 1993, was a knowledge base of medical terms called GALEN [87]. The suitability of modal and description logics for representing and reasoning with this knowledge base, as well as the efficiency of automated reasoners in answering typical and useful queries, have been convincingly demonstrated [47, 48]. Since then, the amount of data in whatever format could be conceived of as ontologies, suitable for automated reasoning, has increased rapidly and continues to do so, thanks to the expansion of the World Wide Web. In 2001, the World Wide Web Consortium approved a standard representation language for ontologies [50, 59], now called OWL (Web Ontology Language). Under the popular but somewhat vaguely defined moniker "Semantic Web", large efforts are now underway both in academic research and industry, to facilitate representation of and reasoning with knowledge in the format of ontologies. One goal is to develop intelligent agents which can access these ontologies and perform useful reasoning tasks with them, ontologies, just as humans access vast quantities of knowledge on the Web today, structured for human consumption and reasoning.

While the demand for increasingly powerful reasoning technology for ontologies is obvious, research efforts have intriguingly shifted away from the basic logics into desirable but specialized add-ons, such as reasoning with numerals, inverse modalities, and compositions of relations [95, 43, 53]. We see two plausible reasons for this: First, the rapid progress that has been made 4–7 years ago in optimizing reasoners may leave people feeling that further progress is either unnecessary or impossible. Secondly, it is held that designing an improvement over existing reasoners not only requires a good idea for a general reasoning algorithm, but also a great number of optimization techniques on both the implementation and hardware levels. There have been promising proposals for better algorithms and/or techniques, such as [9, 37, 38, 36, 44], but to the best of our knowledge, they have not found their way beyond prototypical implementation, be it into any of the leading reasoners or

[1] In practice, description logics are normally preferred here. Due to the close relationship between modal and description logics, much of what we will say about modal logics also holds for description logics, and we will refer to the latter in some of our discussions. We choose modal logics because their definition is somewhat simpler.

into an entirely novel reasoner.

A similar sentiment existed four years ago in the SAT (propositional satisfiability) community. However, since then it has witnessed spectacular and unanticipated progress in efficiency on SAT solvers. Fairly simple ideas, cleverly implemented, have contributed to significant improvements on the bound of problem sizes which can be handled by state-of-the-art solvers [78, 96, 89]. This demonstrated that improvement *was* possible, even without numerous years of developing and fine-tuning large and sophisticated reasoners.

Could such a success be repeated in propositional modal logics? There is good reason to believe so. First, they incorporate propositional reasoning. In fact, the logic of quantified Boolean formulas (QBF), which is as expressive as the modal logic **K** but with much simpler syntax, essentially consists of statements over the sets of variable assignments satisfying a propositional formula. Some interesting progress has been made, and continues to be made, on the efficiency of reasoners for QBF, and research efforts in this field have increased, while it is commonly held that substantial further improvements are attainable. At the same time, the leading modal and description logic reasoners have not proven too successful in handling certain hard translations of QBF problems [74][2], and we will explain this shortly. Moreover, the basic tableau-based calculus, used in three of the four most efficient provers today, is essentially unchanged since 1963. Ingrained in the calculus, as we will argue, are fundamental shortcomings which make reasoning inefficient[3]. Many of the optimizations in today's provers take aim at overcoming these shortcomings; they are motivated chiefly by practical considerations such as pruning of the search tree, search heuristics, model caching, and others. They do not address the shortcomings themselves. We claim that a significant improvement over the original calculus is possible, and we will propose one. Naturally, we will have to specify how exactly we measure "improvement".

Alternatives to tableau-based approaches Tableau-based calculi, the way they are currently used, perform reasoning by manipulating sets of formulas, each of which is bound to one single world in a Kripke structure. In explicit or labelled tableau systems, this

[2]Conversely, QBF solvers have not been too successful at translations of "easy" **K**-problems either [10], which suggests, at the very least, that today's algorithms in either field are not sophisticated enough to see beyond the encoding and recognize the underlying structure of the problem.

[3]To be entirely fair, we acknowledge that Kripke's original proposal has been refined over the years by calculi such as prefixed tableaux and tableaux which use semantic branching. However, the shortcomings we will point out affect these calculi as well.

world is explicitly represented in the formula, usually by a label. In implicit or unlabelled tableaux, the world is implied by the current state of the tableau proof, and its location in the structure is determined by the tableau rule applications which led to the current state. (For the difference between the two systems, see [22, 39].) But in either case, the classical calculus handles (more or less) one formula at a time, and hence one world at a time. It never reasons over sets of worlds simultaneously, although there is no compelling reason for this restriction. After all, the $\Box$-operator expresses a statement over a set of worlds, namely the successors of the current world. We give two reasons why we consider reasoning one world at a time a serious bottleneck:

1. Multiple $\Diamond$-formulas in nodes give rise to branching in the Kripke structure. On problem classes of unbounded modal depth, a minimal Kripke model may be exponentially larger than the original formula in the worst case, so we may need to reason over exponentially many instances (worlds). We will encounter examples in Chapter 4.

2. Disjunctions within $\Box$-formulas lead to branching within the *search tree*. Worse still, branching must be repeated in *each* of the successor worlds of the current world. In combination with 1., one can state problem classes of unbounded modal depth with disjunctions giving rise to exponentially many branching points, devastatingly degrading performance.

The effect of 2. has been well testified and, worse still, observed to occur in real-world problems [54, 51], and this considered to be the more serious of these two bottlenecks. (Of course, the two sources of complexity are inseparable; any approach addressing 1. and reducing branching in the structure will also reduce the impact of 2.)
Another existing and reasonably efficient approach is to translate the problem into a problem in classical first-order logic [80]. An efficient algorithm exists in the optimized functional translation [81], and it is known that all resulting first-order problems belong to the two-variable guarded fragment which is decidable and for which a number of complexity results are known [40, 67, 82, 30]. However, little research has been done on efficiently solving the translated problems. Conventional wisdom suggests using existing "off-the-shelf" resolution-based provers, since they already incorporate generations of research into efficiency improvements; moreover, a variety of provers with different characteristics are available. The appeal of resolution is that it incorporates reasoning with variables, which makes it possible to reason over sets of worlds simultaneously, provided the first-order translation of the problem

represents $\Box$-formulas using variables (which the optimized functional translation does). On the downside, the resolution method itself provides poor guidance for an efficient search. Computing all resolvents of a set of formulas is usually prohibitively expensive; even if redundancies (subsumptions) are filtered out, it is a difficult task to devise a search strategy which is both complete and well-adapted to the specific structure of problems. In this regard, first-order provers are not optimized for use with translations of modal formulas. Besides, resolution is mostly used for proving theorems, or equivalently, for showing a problem unsatisfiable. They are not as strong in finding models for satisfiable problems. Nonetheless, there are some who defend the translation-based approach as superior to tableau methods. In one solver comparison on benchmark problems [74], the MSPASS system [57] which translates modal logic problems into FOL and solves them using the SPASS theorem prover, has been reported to scale better than the competing tableau-based provers as the problem size increases, while lagging behind in terms of absolute running time.

Apart from resolution, new first-order calculi have been devised and refined to the point of being competitive to resolution, and implementations have become available. Two interesting calculi here are the *model evolution calculus* [5] and the disconnection tableau calculus [12, 63, 65]. At the end of this work, we will briefly discuss them and see how they are related to our approach and why we consider them promising for solving translations of modal problems.

Back to the original modal logic problems, there has been some research and a prototypical implementation [8] on an adaptation of the *free-variable tableau calculus* [9] from the like-named calculus in first-order logic [23]. This idea elaborates on the use of labels in tableaux. The free-variable tableau calculus reasons with formulas preceded by labels which reflect the original modalities: each $\Diamond$-operator corresponds to a constant, and each $\Box$-operator to a *variable* in the label. This captures the universality of the $\Box$-operator: the variables in a label can be *instantiated* by constants, thus referring to any particular world over which the $\Box$-formula is scoped. Depending on the particular modal logic, different rules are in place to translate modalities into labels. Furthermore, the calculus defines how branching over labels with variables must be performed in order to preserve soundness and completeness. Free-variable tableau systems could be seen as specific instances of the very general *labelled deductive systems* [26].

To our knowledge, this work has not been continued recently. Many issues have not been satisfactorily answered. First, the free-variable calculus has chiefly been proposed as a

method for proving unsatisfiability, although it has been shown how the calculus can be extended to a decision procedure. But no characterization of models, or how to obtain one from a tableau for a satisfiable set of formulas, has been provided. Furthermore, there are no formal results on the reasoning complexity of this calculus or on problem classes which can be handled more efficiently. And finally, while the prototypical implementation has the favourable features of being lean and modular, it has not been optimized for handling large-scale problems. At its core it relies on a Prolog engine for branching, backtracking and search heuristics, which unacceptably restricts the degrees of freedom for optimizing[4].

Goal of this work Our work originated from an enhancement of simplification and subsumption detection methods [69, 70, 45]. However, the approach has since developed into a formalism quite similar to mentioned free-variable tableaux, so it should be duly considered a continuation of that research. However, our treatment will be markedly different: Rather than stating it as a calculus in normal modal logics, we present it in the setting of a novel logic which we name $\mathcal{LBL}$, for "logic of labelled formulas". The formulas of $\mathcal{LBL}$ are made of propositional atoms, propositional connectives, and labels which may occur not only in the prefix, but throughout the formula; they take the place of modalities. Instead of variables, we have a single anonymous variable or wildcard $*$. Labels with variables represent sets of ground labels instantiating them. As an important notational extension of labels with variables, we later introduce labels with *exceptions*, which allows us to express the set difference of two or more sets represented by labels. The use of labels with exceptions reflects common real-world reasoning scenarios, where a formula can be satisfied in a large set of instances (worlds) using some default assignment, but in some (sets of) *exceptional* instances a different assignment must be found. For instance, the statement "Birds fly or have short wings" cannot be instantiated universally over the sets of all instances which satisfy "is-a-bird". But for *all but a few exceptional* instances, a default assignment "flies" succeeds without inconsistencies, and only for a few subsets of instances (such as penguins, emus etc.) an alternate assignment "has-short-wings" must be chosen.
Unlike in most other approaches, our emphasis is less on proving theorems rather than *finding concise models* for satisfiable formulas, and doing so efficiently. We will specify a model-based semantics for $\mathcal{LBL}$ and show how $\mathcal{LBL}$-models correspond to conventional

[4]Besides, we found the published implementation [8] to be incorrect for problems of a certain irregular structure.

Kripke models[5]. Conciseness is an important issue: While the satisfiability of a formula φ in the modal logic **K** can be decided in polynomial space, a Kripke model for φ may take exponential space to store it. So lest we impose a serious handicap regarding the size of problems which our algorithm can solve, we must be concerned about representing models as efficiency as possible. We will motivate why $\mathcal{LBL}$-models are typically smaller than Kripke models, and provide a large and interesting class of problems for which they are *provably* smaller.

Abandoning the use of tableaux for model finding, we will instead specify an algorithm based on nogood reasoning, similar to the well-known *dynamic backtracking* algorithm [31] for constraint satisfaction problems. Our algorithm provides a decision procedure, and we will prove it sound and complete, so it can be used for theorem proving (via unsatisfiability checking) as well as model finding. We do not provide an implementation, but the algorithms in this work are clearly and precisely worked out; implementing them should be a straightforward though perhaps cumbersome task.

Our goal is not to provide a general mechanism for handling as many logics as possible, as done in [9], or even to offer a general framework for many and diverse logics such as [26]. Our presentation is restricted to the most general normal logic **K**. The main reason is that the algorithm requires many novel ideas as it is, and restriction to **K** helps keep the amount of detail manageable. However, it is not our intention to devise a highly specialized algorithm for **K** only; this logic in itself is very weak and does not offer the additional features so desirable for practical reasoning with real-world ontologies. Instead, we intend our logic $\mathcal{LBL}$ and model finding algorithm as a starting point for developing similar ideas for more powerful logics. Some of these ideas will go well beyond those presented in this thesis, and we will characterize them as future work. The ideas we do use for **K** will be presented so as to be re-usable for other logics. Our approach is modular, in that we distinguish between the reasoning stages of translation, model finding, and consistency checking. We claim that these stages naturally result from the structure of modal formulas and the interplay between the propositional and modal operators. Each stage falls into a clearly defined complexity class. In this sense, this work makes a modest contribution to the theory of modal logics, by helping to understand how different constructs within modal formulas are sources of different degrees of hardness.

[5] If we like to be conservative, we could think of these models merely as *representations* of Kripke models, instead of claiming an entirely novel way to specify the semantics of modal logics.

Structure of this thesis As we have described, this work can be viewed as a sophisticated treatment of labelled tableaux. Alternatively, we could translate it into first-order logic and consider it a refinement of first-order reasoning. Either approach would have the advantage of offering a reasonably solid, established theoretical foundation upon which we can build our results.

However, we found that stating the work in terms of the tableau method would be too restrictive. The algorithm could be stated in a tableau framework, but it has little in common with Kripke's, or Smullyan's, original idea. Besides manipulating formulas from one tableau node to another, we would need to carry a lot of extra information with us. Moreover, our search strategy of branching and dependency-directed backtracking does not correspond to that of the classical tableau calculus.

Stating our approach in first-order logic, on the other hand, never was our intention. We consider first-order logic as too general a framework for our problems, whose translations would cover only a small fragment [67]. So while our work may well be fully specifiable in FOL, it would not likely become any easier. Besides, the class of first-order problems our algorithm can solve would not be as well-defined as the class of problems in **K**.

Using a novel logic $\mathcal{LBL}$ offers the advantage of clearly displaying the anatomy of a **K**-problem and our method of solving it.

The disadvantage of this approach is its considerable overhead. We need to define syntax and semantics, present a translation from **K** to $\mathcal{LBL}$ and prove it satisfiability-preserving, and characterize properties of models in $\mathcal{LBL}$. Much of this is routine work, neither very deep nor groundbreaking; hence the considerable size of this work. While existing concepts in the field can be adapted to meet many of our needs, they are not close enough to let us simply cite them; instead we must prove that the adapted concepts serve their desired purpose, so as to warrant correctness of our research.

A point in case is Chapter 2, a reflection on propositional reasoning. Our overall approach requires that all formulas be in negation normal form (NNF). While conversion to NNF is convenient and standard practice, formal treatments of the NNF are sparse in the literature on propositional logic. Consequently, we will expend great efforts developing terminology and results for the restriction of propositional logics to negation normal form. As such, these results are novel to the best of our knowledge, but they do not provide groundbreaking insights over and beyond their well-known counterparts in unrestricted propositional logics. Only a small section of this chapter actually deals with classical propositional logic. For the

major part, we will be concerned with general statements and theorems for propositional logics, that is, for any logic in NNF which defines $\wedge$, $\vee$, $\top$, and $\bot$ in their usual meanings; to these belong the basic normal modal logics as well as our new logic $\mathcal{LBL}$. These results will free us from rediscovering the same facts about the propositional part in each individual logic later, and thus reduce redundancy in this work considerably. Arguably the main contributions of the chapter are the notion of *propositional covers*, on which the semantic structures for $\mathcal{LBL}$ are based, and an important principle we refer to as *monotonicity*. We also introduce a number of useful proof tools like induction on the structure of formulas, and homomorphisms.

Chapter 3 covers all aspects of the modal logic $\mathbf{K_{NNF}}$ ($\mathbf{K}$ restricted to NNF) needed for the purposes of this work. It gives an account of the semantics of $\mathbf{K_{NNF}}$ in terms of Kripke structures. We will derive an interesting correspondence between Kripke models and propositional covers, which will foreshadow a similar correspondence in Chapter 4. A byproduct is the class of *strict* models which characterizes models arising from the classical tableau calculus without optimizations.

In Chapter 4, we introduce labels and labelled formulas for the first time. We offer a semantics for formulas which is little more than a concise version of the semantics of $\mathbf{K}$. Based on the analogies between the two, we describe a translation function between $\mathbf{K_{NNF}}$ and $\mathcal{LBL}$ which preserves satisfiability. We will then demonstrate that the worst-case complexity of reasoning is the same as in $\mathbf{K}$, but also that a large and interesting problem class admits polynomial-size models in $\mathcal{LBL}$ but not $\mathbf{K}$. In turn, we will see that the worst-case complexity of model checking increases relative to the size of an $\mathcal{LBL}$-model, but not (significantly) relative to the size of the Kripke model represented by it.

As we will find, the logic $\mathcal{LBL}$ as defined in Chapter 4 is still to weak to accommodate our goal of reasoning efficiently over sets of instances; for instance, it does not provide for default assignments with exceptions. In Chapter 5, we will progressively develop the still familiar-looking semantics of $\mathcal{LBL}$ into a novel, workable form which actually utilizes the expressivity offered by labels with wildcards. While the language of $\mathcal{LBL}$ remains largely unchanged, we first introduce labels with exceptions, allowing for even more concise models, and then undergo a significant paradigm shift from reasoning "one modality at a time" (or one label element at a time) to reasoning "from defaults to exceptions", considering labels as a whole. Each of these two steps is motivated carefully, showing at each step how the emphasis changes. Of course, we will demonstrate that the satisfiability of formulas remains

invariant to all these changes in semantics. We will argue that the worst-case complexity results of Chapter 4 remain unchanged, but that the improvements in Chapter 5 lead to average-case improvements in practice.

Our efforts culminate in Chapter 6 in a detailed description of a model-finding algorithm, its reasoning strategy, and a proof of its soundness and completeness. Mostly for convenience we introduce a normal form in which to represent formulas, analogous to the clause normal form (CNF) commonly found in propositional reasoning. This *labelled clause normal form* is a byproduct which may well be of interest beyond its use for the algorithm.

We conclude this work with some considerations about implementability, an outline of important future work to be done, and a summary of our contributions, in Chapter 7.

The tableau calculus has already been mentioned a number of times. We pay due tribute to this important technique and introduce it. This serves two additional goals: demonstrating how the formulas in the tableau nodes naturally correspond to propositional covers, and showing how strict Kripke models arise from the classical tableau calculus without optimizations. We draw upon results up to Chapter 3 only, and our findings, though interesting, are not needed for the rest of this work, so we attach them as Appendix A.

The reader may not be all that interested in reading sequentially through this entire work with all its theorems and propositions. For this reason, each of the three main chapters 4, 5, and 6 begins with a section on the problems we address in that chapter, motivating the results on some examples, and introducing key concepts we will define and use from that chapter onwards. One may wish to study these sections first to get a general overview, and then focus on individual sections of interest. To facilitate this, the (substantial) interdependency of results among sections is clearly and extensively marked, by referencing results from earlier sections and chapters by number, wherever they are used. A list of all algorithms by name and number is provided; finally, a list of all non-standard notation used, with explanation and reference to the page on which they were first defined, can be found in a glossary at the end of this thesis.

Chapter 2

Extensions of Propositional Logic

> *It is difficult to say what truth is, but sometimes so easy to recognize a falsehood.*
>
> *Albert Einstein*

In establishing our results, we will make use of a variety of well-known as well as novel logical frameworks. These frameworks have many properties in common; notably, they are logics with a well-defined semantics, based on some notion of Tarski-style *structures* and *models*[1]; moreover, they are all closed under the propositional operators $\wedge$ and $\vee$, which are always defined as expected: for instance, a model for $\varphi \wedge \psi$ is a model for both φ and ψ and vice versa, independent of the internal structure of φ, ψ, or of the model. Another common feature, quite distinguished from usual treatments of propositional or modal logics (but common for publications in automated reasoning) is that the negation operator $\neg$ is restricted to occurences in front of propositional variables only. This amounts to formulas being in the so-called *negation normal form* (NNF). It does not restrict expressivity though, as long as the logic obeys the duality principle: that is, each operator $\circ(\cdot, \ldots, \cdot)$ has a dual operator $\bar{\circ}(\cdot, \ldots, \cdot)$ so that $\neg \circ (\varphi_1, \ldots, \varphi_n) = \bar{\circ}(\neg\varphi_1, \ldots, \neg\varphi_n)$. We refer to any logic satisfying the above properties as *a propositional logic* (in NNF), as opposed to the familiar *classical propositional logic* which is just one special instance of a propositional logic, albeit a very important one.

Section 2.1, which takes up most of this chapter, is devoted to characterizing common features of propositional logics in NNF, and generalizing well-known results of classical

[1] See the remarks on terminology further below in this section.

propositional logic to *any* such logic. These results will be applied to individual logics in later chapters, thus saving us the tedious work of establishing them over and over again. Admittedly, this makes for a very nonstandard introduction into well-known logics such as **K**, but we feel that the benefits of our systematic approach outweigh the inconvenience of an unfamiliar perspective. Someone entirely new to the field may wish to consult standard treatments [15, 56] first.

Let us introduce a few general notions at the outset. Throughout this work, we deal with *propositions*. Propositions are considered atomic, and they are represented by *variables* from a given set P which is not *a priori* bounded; but only a finite subset of P is used in any particular problem instance. A *logic* $\mathbf{L}$ is built upon a language $\mathcal{L}(\mathbf{L})$ of well-formed *formulas* (or $\mathbf{L}$-*formulas*, if we wish to emphasize the logic used), and the semantics according to which formulas are evaluated. $\mathcal{L}(\mathbf{L})$ consists of propositional variables from P, *propositional constants* ($\top$ and $\bot$), logical connectives (such as $\wedge$,$\vee$,$\neg$), auxiliary symbols such as parentheses, and other symbols for use in logics introduced in Chapter 4. The semantics of a logic is based on Tarski-style *semantic structures*, also known as *interpretations*—objects whose structure depends on the specific logic. Structures specify the way in which *truth values* are assigned to the propositional variables, usually depending on their context within the structure. The only truth values allowed are *true* and *false* (always written in slanted font); there is no third value such as *undefined*. A variable should never be assigned to both *true* and *false* in one and the same context, although this is not enforced by the definition of some of the logics—for good reasons, as we will see. In these logics, we distinguish between *consistent* and *inconsistent* structures; the former are also called *models*. We need not and will not always state complete truth assignments for all variables in all contexts, but implied in such *partial assignments* is that any unassigned variable may be assigned arbitrarily to *true* or *false*. Any two variables from P are considered distinct, and their truth assignments are *a priori* independent of one another.

Besides structures, the semantics of a logic is based on a relation $\models$ on (consistent) structures and formulas. Whenever $\mathcal{M} \models \varphi$, we say that φ is *satisfied* by $\mathcal{M}$, or that $\mathcal{M}$ is a *model for* φ. If the logic allows for inconsistent structures, we will not admit these as models, but we still desire some notion of "satisfiability". Therefore, we use a different relational operator $\models\!\!\!\sim$ for structures in general. Hence $\mathcal{M} \models \varphi$ iff M is consistent and $\mathcal{M} \models\!\!\!\sim \varphi$. We thus separate the concept of "structures satisfying a formula" (the $\models\!\!\!\sim$ relation) from that of consistency, a major tenet we hold in this work.

At this point, we must address a common source of confusion. In the eyes of many logicians, the term "model" is reserved to structures which satisfy a given specific formula. Our own use of the term "model", however, follows common practice in the field of modal logics, where a model is any object which satisfies the given structural and consistency requirements, without a particular formula in mind. If we mean "model" in the former sense, we will always speak of a "model for φ" or a "model satisfying φ" etc.

For reasons which will become evident throughout the work, we would also like to refer to objects satisfying the structural requirements, but not necessarily the consistency requirements, of the logic; we will simply call them "semantic structures". We define a relational operator $\models$ on structures and **L**-formulas, and whenever $\mathcal{M} \models \varphi$, we will say that $\mathcal{M}$ *verifies* φ; the $\models$ relation is derived as the restriction of the $\models$ relation to models.

The $\models$ relation is the most vital part of each logic. Standard reasoning tasks in any given logic can be expressed in terms of $\models$ as follows:

- model checking, that is, verifying whether a given pair $(\mathcal{M}, \varphi)$ is an element of the satisfiability relation $\models$,
- satisfiability checking, that is, verifying that $\models^{-1}(\{\varphi\}) \neq \emptyset$,
- model finding, that is, finding an element of $\models^{-1}(\{\varphi\})$.

Other tasks, such as proving unsatisfiability ($\models^{-1}(\varphi) = \emptyset$) or validity ($\mathcal{C}(\models^{-1}(\varphi)) = \emptyset$), can be derived from the above (provided the tasks are decidable, which will be the case for all logics considered in this work), so we will keep the above three tasks at the centre of our attention.

2.1 Formal Definition

Definition 2.1.1 *A* propositional logic **L** (in negation normal form) *is defined by a language* $\mathcal{L}(\mathbf{L})$ *which is the smallest language containing a given, fixed set A of* atoms *including the propositional constants* $\top$ *and* $\bot$*, and closed under the binary propositional connectives* $\wedge$ *and* $\vee$*; and by a semantic relation* $\models_g$ *on a given, fixed set of semantic structures on the one hand, and formulas or sets of formulas on the other hand, obeying the following principles (where* α_1*,* α_2*,* β_1*,* β_2*,* φ *are* **L***-formulas,* Φ *a set of formulas, and* $\mathcal{M}$ *a structure:*

$$
\begin{array}{lll}
\mathcal{M} \models_g \top & & \textit{for all } \mathcal{M} \\
\mathcal{M} \models_g \bot & & \textit{for no } \mathcal{M} \\
\mathcal{M} \models_g \alpha_1 \wedge \alpha_2 & \textit{iff} & \mathcal{M} \models_g \alpha_1 \textit{ and } \mathcal{M} \models_g \alpha_2 \\
\mathcal{M} \models_g \beta_1 \vee \beta_2 & \textit{iff} & \mathcal{M} \models_g \beta_1 \textit{ or } \mathcal{M} \models_g \beta_2. \\
\mathcal{M} \models_g \Phi & \textit{iff} & \mathcal{M} \models_g \varphi \textit{ for all } \varphi \in \Phi.
\end{array}
$$

Furthermore, the logic **L** *characterizes some or all semantic structures as consistent, or* models. *If* $\mathcal{M} \models_g \varphi$ *(or* $\mathcal{M} \models_g \Phi$*) for a semantic structure* $\mathcal{M}$*, we say that* $\mathcal{M}$ verifies φ *(or* Φ*); if additionally* $\mathcal{M}$ *is consistent, we write* $\mathcal{M} \models_g \varphi$ *(*$\mathcal{M} \models_g \Phi$*) and say that* $\mathcal{M}$ satisfies φ *(or* Φ*). Whenever a model exists for a given formula* φ *(or set* Φ*),* φ *(or* Φ*) is said to be* $\models_g$-satisfiable *(or usually just* satisfiable*, with* $\models_g$ *clear from the context). We call two sets of formulas* Φ_1 *and* Φ_2 equivalent *(*$\Phi_1 \equiv_g \Phi_2$*), if they are satisfied by exactly the same semantic structures.*

For the rest of this section, **L** is understood to be a "generic" propositional logic with $\models_g$ as its semantic relation. The absence of a negation operator in the definition is intentional: while a negation operator will exist in all incarnations of propositional logics we consider, it will form part of a propositional atom, together with the propositional variable it precedes; no other occurrence of $\neg$ anywhere else is allowed. We also point out that different logics differ on the sets of atoms they define, which is why we do not include them in the generic definition (apart from $\top$ and $\bot$ which are always present).
Definition 2.1.1 entails an obvious but useful property:

Proposition 2.1.2 *If* $\mathcal{M} \models_g \Phi$*, then* $\mathcal{M} \models_g \Phi'$ *for any* $\Phi' \subseteq \Phi$.

One easily verifies that the conjunction and disjunction operators are associative, just as in plain propositional logic. This allows us to omit parentheses, effectively turning $\wedge$ and $\vee$ into connectives of any finite arity. We write a term with n conjuncts $\alpha_1, \ldots, \alpha_n$, also called an *$\alpha$-formula*, as $\alpha = \alpha_1 \wedge \ldots \wedge \alpha_n$ or $\alpha = \bigwedge_{i=1}^{n} \alpha_i$ or also $\alpha = \bigwedge\{\alpha_i : i = 1, \ldots, n\}$. Likewise, we write a disjunction with m disjuncts $\beta_1, \ldots, \beta_m$, also called a *β-formula*, as $\beta = \beta_1 \vee \ldots \vee \beta_m$ or $\beta = \bigvee_{j=1}^{m} \beta_j$ or also $\beta = \bigvee\{\beta_j : j = 1, \ldots, m\}$.

Proposition 2.1.3 *With notation as introduced above, we have:*

$$\begin{array}{lll} \mathcal{M} \models_g \bigwedge_{i=1}^n \alpha_i & \textit{iff} & \mathcal{M} \models_g \alpha_i \textit{ for all } i = 1, \ldots, n. \\ \mathcal{M} \models_g \bigvee_{j=1}^m \beta_j & \textit{iff} & \mathcal{M} \models_g \beta_j \textit{ for at least one } j = 1, \ldots, m. \\ \mathcal{M} \models_g \{\alpha_1, \ldots, \alpha_n\} & \textit{iff} & \mathcal{M} \models_g \bigwedge_{i=1}^n \alpha_i. \\ & & \textit{(Hence } \{\alpha_1, \ldots, \alpha_n\} \equiv_g \bigwedge\{\alpha_1, \ldots, \alpha_n\}.\textit{)} \end{array}$$

Proof: The first two statements follow by applying Definition 2.1.1 $(n-1)$ and $(m-1)$ times, respectively. For the third statement, we observe that the last line in Definition 2.1.1 and the first statement of this proposition feature identical conditions. □

We see that sets and conjunctions of the same formulas are equivalent. Many definitions and results in this work can be sensibly expressed in terms of formulas as well as sets of formulas, but we will often save space and efforts by stating them in only one form. In those cases, it is understood that a result holds for a formula φ whenever it holds for the set $\{\varphi\}$, and that a result holds for a set Φ whenever it holds for the conjunction $\bigwedge\Phi$.

Corollary 2.1.4 *If* $\alpha = \bigwedge_{i=1}^n \alpha_i$ *is a conjunction of conjunctions* $\alpha_i = \bigwedge_{j=1}^{n_i} \alpha_{i,j}$, *then* $\alpha \equiv_g \bigwedge\{\alpha_{i,j} : j = 1, \ldots, n_i, i = 1, \ldots, n\}$. *If* $\beta = \bigvee_{j=1}^m \beta_j$ *is a disjunction of disjunctions* $\beta_j = \bigvee_{i=1}^{m_j} \beta_{j,i}$, *then* $\beta \equiv_g \bigvee\{\beta_{j,i} : i = 1, \ldots, m_j, j = 1, \ldots, m\}$.

Hence a nested conjunction can be converted into one single conjunction, and a nested disjunction into one single disjunction. This simplification is known as *flattening.*

Definition 2.1.5 *A formula* φ *is a* propositional subformula *of* φ' *if either* $\varphi = \varphi'$ *or* φ' *is of the form* $\alpha_1 \wedge \ldots \wedge \alpha_n$ *or* $\beta_1 \vee \ldots \vee \beta_m$, *so that recursively,* φ *is a propositional subformula of some* α_i *or* β_j.

A formula can have a multiple number of occurrences of the same subformula. We usually consider different occurrences as distinct, except in a set context: $\{\varphi, \varphi\} = \{\varphi\}$. But we will comment on this whenever it is crucial. In this chapter we will consider atoms as truly atomic, in that they do not have propositional subformulas. Although in most logics to be introduced later atoms will be built from constructs which themselves have propositional subformulas, the recursive propositional evaluation or transformation of a formula stops at the level of atoms, as it is ignorant of their internal make-up.

Since $\mathcal{L}(\mathbf{L})$ is a minimal closure of the set of atoms A under propositional concatenation, any $\mathbf{L}$-formula is a concatenation of finitely many atoms. So long as every atom in A is

finite, every formula is also finite in length. This allows us to perform induction proofs on the propositional structure of a formula, which we will use extensively. We state the principle formally as follows: Let $Q(\cdot)$ be a property defined on formulas in the logic **L**. We say that Q is *preserved under propositional concatenation*, if whenever $Q(\varphi_1)$ and $Q(\varphi_2)$ are true, then so is $Q(\varphi_1 \circ \varphi_2)$ for $\circ = \wedge, \vee$.

Theorem 2.1.6 *(Structural induction on φ) If Q is a property on* **L** *which is preserved under propositional concatenation, and $Q(a)$ holds for all atoms in* **L**, *then $Q(\varphi)$ holds for every formula φ in* **L**.

Proof: Every formula contains only finitely many propositional subformulas. Take any **L**-formula φ. Assume that $Q(\varphi)$ does not hold. Let φ' be a propositional subformula of φ so that $Q(\varphi')$ does not hold, but $Q(\varphi'')$ holds for all subformulas φ'' of φ'. Now φ' cannot be atomic, as Q holds for all atoms. Hence $\varphi' = \varphi_1 \circ \varphi_2$ for some subformulas φ_1, φ_2. However, the way we defined φ', we have both $Q(\varphi_1)$ and $Q(\varphi_2)$. Since Q is closed under propositional concatenation, we also have $Q(\varphi_1 \circ \varphi_2)$, contradicting the assumption that Q does not hold for φ'. □

Note that the induction principle is indifferent to the semantics of $\wedge$ and $\vee$. Moreover, the connotation of Q is entirely different from that of $\models_g$. We use Q as a meta-logical operator to state and prove properties on **L**, not satisfiability properties of formulas. In all proofs using structural induction, the property represented by Q—that is, the induction hypothesis—will always be clearly stated.

Let us consider a prominent example of a property—or here rather, a metaproperty—preserved under propositional concatenation:

Definition 2.1.7 *Assume that some partial order $\leq$ is defined on the semantic structures of* **L**. *A property $Q(\mathcal{M})$ defined on semantic structures (and possibly other parameters) is* monotone *wrt $\leq$, if whenever $Q(\mathcal{M})$ is true and $\mathcal{M} \leq \mathcal{M}'$, then $Q(\mathcal{M}')$ is also true.*

The semantic relation $\models_g$ can be viewed as a property on semantic structures, parametrized by an **L**-formula; it is not the only, but by far the most important property for wish we desire to prove monotonicity, and we will exert great efforts in defining a modal-like logic whose $\models$ relation is indeed monotone (see Chapter 5). But here we can state at least one thing about the monotonicity of $\cdot \models_g \varphi$ (which in turn is a property on **L**-formulas):

Theorem 2.1.8 *Monotonicity of the* $\models_g$ *relation on* **L** *is preserved under propositional concatenation.*

Proof: Assume $\mathcal{M} \leq \mathcal{M}'$, and $\mathcal{M} \models_g \alpha_1 \wedge \ldots \wedge \alpha_n$. Then $\mathcal{M} \models_g \alpha_i$ for all $i = 1, \ldots, n$. Now if $\cdot \models_g \alpha_i$ is monotone for all α_i, we can conclude that $\mathcal{M}' \models_g \alpha_i$ for all $i = 1, \ldots, n$, and hence $\mathcal{M}' \models_g \alpha_1 \wedge \ldots \wedge \alpha_n$. We have thus shown that $\cdot \models_g \alpha_1 \wedge \ldots \wedge \alpha_n$ is monotone. A similar argument shows that monotonicity is also preserved under disjunction. □

Corollary 2.1.9 *If* $\cdot \models_g a$ *is monotone for all atoms in* **L**, *then it is monotone (for all formulas in* **L***).*

Proof: This follows directly by structural induction according to Theorem 2.1.6, whereas the induction step is provided by Theorem 2.1.8. □

The contrapositive of Theorem 2.1.9 provides a criterion for $\models_g$ to not be monotone: there must be some atom a so that $\cdot \models_g a$ is not monotone. We will encounter logics where this is so. Indeed, the standard versions of $\models_g$ in classical propositional and modal logics are not monotone.

Definition 2.1.10 *Given an operator* $d(\cdot) : A \mapsto \mathbb{N}_0$ *(where* $\mathbb{N}_0$ *is the set of nonnegative integers), we define a* depth function $d(\cdot) : \mathcal{L}(\mathbf{L}) \mapsto \mathbb{N}_0$ *by setting* $d(\top) = 0$, $d(\bot) = 0$, *and* $d(\varphi_1 \circ \ldots \circ \varphi_n) = \max\{d(\varphi_i) : i = 1, \ldots, n\}$ *for* $\circ = \wedge, \vee$.

Note that $d(\cdot)$ is well-defined, since every formula is a propositional concatenation of atoms. Moreover, every formula has a finite depth. We also observe:

Proposition 2.1.11 *If* **L** *is a propositional logic with a depth function, then so is any restriction of* **L** *to formulas of maximum depth* k, *for any* $k \in \mathbb{N}_0$. *The atoms in this restricted logic are exactly the atoms in* **L** *of depth at most* k.

Proof: Observe that the depth of any propositional concatenation of formulas is no larger than the maximum depth of any of its components, and that $\top$ and $\bot$ are included in any restriction by virtue of having depth 0. The semantic relation $\models_g$ is simply inherited. □

The depth of a set Φ of formulas is defined in the obvious way: $d(\Phi) = \max\{d(\varphi) : \varphi \in \Phi\}$. The depth function will play a vital role in defining the language of many of our logics. Atoms of depth $k + 1$ will be constructed recursively from formulas of depth at most k. In accordance with these definitions, it makes sense to refine the principle of structural induction:

Theorem 2.1.12 *(Induction over the formula depth)*
Let $Q(\cdot)$ be a property on **L***-formulas. If $Q(a)$ is true for all atoms of depth 0, Q is preserved under propositional concatenation, and from $Q(\varphi)$ being valid for all* formulas *φ of depth at most k we can infer $Q(a)$ for any* atom *of depth $k+1$, then $Q(\varphi)$ is valid for all formulas $\varphi \in \mathbf{L}$.*

Proof: We reduce this principle to the "ordinary" induction principle on the integer $d(\varphi)$ and induction on the propositional structure of a formula.
First, let us prove $Q(\varphi)$ whenever $d(\varphi) = 0$. According to Proposition 2.1.11, the set of formulas with $d(\varphi) = 0$ forms a sublogic of **L** whose atoms a are exactly the **L**-atoms of depth 0. For these we already know that $Q(a)$ holds. Furthermore, $Q(\cdot)$ is preserved under propositional concatenation. Hence, by Theorem 2.1.6, Q holds for all formulas of depth 0. Now assume that $Q(\varphi)$ holds for all formulas of depth at most k. In particular, it holds for all atoms of depth at most k, but following from the induction hypothesis, Q holds for atoms of depth $k+1$. Now we apply Theorem 2.1.6 a second time to the sublogic of formulas of depth at most $k+1$, concluding that Q holds on all these formulas. By induction on k we get the desired result. □

Proposition 2.1.13 *For any propositional logic, we have the following substitution properties:*

1. *If $\alpha_i \equiv_g \alpha_i'$ for all $i = 1, \ldots, n$, then $\bigwedge_{i=1}^n \alpha_i \equiv_g \bigwedge_{i=1}^n \alpha_i'$.*
2. *If $\beta_j \equiv_g \beta_j'$ for all $j = 1, \ldots, m$, then $\bigvee_{j=1}^m \beta_j \equiv_g \bigvee_{j=1}^m \beta_j'$.*
3. *For any sets of formulas Φ, Φ_1, Φ_2, if $\Phi_1 \equiv_g \Phi_2$, then $\Phi \cup \Phi_1 \equiv_g \Phi \cup \Phi_2$.*

Proof: These are all direct consequences of Definition 2.1.1 and Proposition 2.1.3. □
These properties allow us to carry out substitutions anywhere within the nested structure of a formula or set of formulas. Later on, when more operators are introduced to obtain particular logics, we will state appropriate substitution properties as well. We will apply the above rules combined with flattening in order to convert a formula into a well-known normal form:

Definition 2.1.14 *A formula in* And/Or normal form *is recursively defined as:*

1. *Atoms are in And/Or normal form.*

2. *A* disjunction in And/Or normal form *is a formula of the form* $\varphi = \bigvee_{j=1}^{m} \beta_j$, *where* $m \geq 2$, *and each* β_j *is either an atom or a* conjunction in And/Or normal form.

3. *A* conjunction in And/Or normal form *is a formula of the form* $\varphi = \bigwedge_{i=1}^{n} \alpha_i$, *where* $n \geq 2$, *and each* α_i *is either an atom or a disjunction in And/Or normal form.*

4. *The smallest set of atoms, disjunctions and conjunctions constructed according to these rules constitutes the formulas in And/Or normal form.*

Note that without the restrictions $m \geq 2$ and $n \geq 2$, this definition would be uninteresting. For, instead of flattening a nested conjunction $\alpha_1 \wedge (\alpha_2 \wedge \alpha_3)$ into $\bigwedge_{i=1}^{3} \alpha_i$, we could simply insert a unary disjunction operator between the two levels of nesting, obtaining $\alpha_1 \wedge \bigvee(\alpha_2 \wedge \alpha_3)$. But unary disjunctions or conjunctions only complicate formulas and are undesirable, at least when efficiency is an issue. Only on the outermost level of nesting will a unary operator prove useful. For example, $\bigwedge_{i=1}^{3} \alpha_i$ can be treated as if it were a disjunction $\bigvee \bigwedge_{i=1}^{3} \alpha_i$, thus avoiding the need for lengthy case distinctions in the proof of our next corollary. So we remark that *any* formula in And/Or normal form can be written as $\bigvee_{j=1}^{m} \beta_j$ as well as $\bigwedge_{i=1}^{n} \alpha_i$ with the restrictions as in lines 2. and 3. above, except that (on the outermost level) m and n can be equal to 1, respectively.

Corollary 2.1.15 *For any set of formulas* Φ, *an equivalent set of atoms and disjunctions in And/Or normal form no larger than* Φ *can be found in* $O(|\Phi|)$ *steps, where* $|\Phi|$ *is the number of atoms in* Φ.

Proof: In the following conversion, we use flattening and other equivalence-preserving operations, making tacit use of Propositions 2.1.3 and 2.1.13 which show that the conversion preserves equivalence.

Algorithm 2.1.16 *Conversion to And/Or normal form*

- *Write a set* $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ *as a single formula* $\varphi = \varphi_1 \wedge \ldots \wedge \varphi_n$.
- *Transform* φ *into And/Or normal form as follows:*
  - *If* φ *is an atom, leave it unchanged.*
  - *If* $\varphi = \bigwedge_{i=1}^{n} \alpha_i$, $n \geq 2$, *convert each conjunct* α_i *recursively into And/Or normal form, obtaining* $\alpha_i' = \bigwedge_{k=1}^{h_i} \alpha_{i,k}$. *Finally, flatten the nested conjunction* $\alpha_1' \wedge \ldots \wedge \alpha_n'$ *into one conjunction* $\alpha_{1,1} \wedge \ldots \wedge \alpha_{n,h_n}$.

  - *If* $\varphi = \bigvee_{j=1}^{m} \beta_j$, $m \geq 2$, *convert each disjunct* β_j *recursively into And/Or normal form, obtaining* $\beta'_j = \bigvee_{l=1}^{h'_j} \beta_{j,l}$. *Finally, flatten the nested disjunction* $\beta'_1 \vee \ldots \vee \beta'_m$ *into one disjunction* $\beta_{1,1} \vee \ldots \vee \beta_{m,h'_m}$.

- *If* $\varphi'_1 \wedge \ldots \wedge \varphi'_{n'}$ *has been thus determined as the And/Or normal form of* φ, *write it as a set* $\{\varphi'_1, \ldots, \varphi'_{n'}\}$.

The recursive construction terminates within the given time complexity, since the formulas converted in each recursive step are strict subformulas of the formula in the previous step and thus subformulas of (formulas in) the original set Φ; furthermore, each subformula is converted only once. The only operation performed in each step is flattening, so formulas may only get shorter due to omitted parentheses. It is easy to verify that the converted formulas are indeed in And/Or normal form. (Note that the arity of conjunctions and disjunctions never decreases through flattening, as multiple occurrences of subformulas are preserved as such. It is slightly less trivial to show that all formulas and propositional subformulas in the final set satisfy the arity restrictions $m \geq 2$ and $n \geq 2$ of Definition 2.1.14.) □

We will now present a few general results on transformations between two logics, particularly those which will allow us to derive satisfiability results for a new logic from those for a known logic.

Definition 2.1.17 *Let* $\mathbf{L}_1$ *and* $\mathbf{L}_2$ *be two propositional logics. An* $\mathbf{L}_1 - \mathbf{L}_2$ homomorphism *is a transformation* $\tau : \mathcal{L}(\mathbf{L}_1) \mapsto \mathcal{L}(\mathbf{L}_2)$, *satisfying:*

- $\tau(\alpha_1 \wedge \alpha_2) = \tau(\alpha_1) \wedge \tau(\alpha_2)$,
- $\tau(\beta_1 \vee \beta_2) = \tau(\beta_1) \vee \tau(\beta_2)$,
- $\tau(\top) = \top$, *and* $\tau(\bot) = \bot$.

It follows straightforwardly from the definition that homomorphisms map non-atoms to non-atoms. If the converse is also true, i.e. τ maps all $\mathbf{L}_1$-atoms to $\mathbf{L}_2$-atoms, we say that τ *preserves atoms*. We extend the definition of τ to sets of formulas in the usual way, namely $\tau(\Phi) = \{\tau(\varphi) : \varphi \in \Phi\}$. We also mention without proof that any mapping τ' from atoms of $\mathbf{L}_1$ to formulas of $\mathbf{L}_2$ can be *uniquely* extended to a homomorphism, the *homomorphism induced by* τ'.

Proposition 2.1.18 *Let $\models_1$ and $\models_2$ be the semantic relations of two propositional logics $\mathbf{L}_1$ and $\mathbf{L}_2$. Let further τ be an $\mathbf{L}_1 - \mathbf{L}_2$ homomorphism and $\mathcal{M}_1$, $\mathcal{M}_2$ semantic structures. Then the following properties are preserved under propositional concatenation:*

(1) $\mathcal{M}_1 \models_1 \varphi \Rightarrow \mathcal{M}_2 \models_2 \tau(\varphi)$;

(2) $\mathcal{M}_2 \models_2 \tau(\varphi) \Rightarrow \mathcal{M}_1 \models_1 \varphi$;

(3) $\mathcal{M}_1 \models_1 \varphi \Leftrightarrow \mathcal{M}_2 \models_2 \tau(\varphi)$.

Proof: For the sake of illustration, we show that Property (1) is preserved under conjunction. Assume that $\mathcal{M}_1 \models_1 \alpha_1 \Rightarrow \mathcal{M}_2 \models_2 \tau(\alpha_1)$ and $\mathcal{M}_1 \models_1 \alpha_2 \Rightarrow \mathcal{M}_2 \models_2 \tau(\alpha_2)$. Then we show the same property for $\alpha_1 \wedge \alpha_2$ by performing these transformations:

$$
\begin{aligned}
& \mathcal{M}_1 \models_1 \alpha_1 \wedge \alpha_2 \\
\Leftrightarrow\ & \mathcal{M}_1 \models_1 \alpha_1 \text{ and } \mathcal{M}_1 \models_1 \alpha_2 \\
\Rightarrow\ & \mathcal{M}_2 \models_2 \tau(\alpha_1) \text{ and } \mathcal{M}_2 \models_2 \tau(\alpha_2) && \text{(hypothesis)} \\
\Leftrightarrow\ & \mathcal{M}_2 \models_2 \tau(\alpha_1) \wedge \tau(\alpha_2) \\
\Leftrightarrow\ & \mathcal{M}_2 \models_2 \tau(\alpha_1 \wedge \alpha_2) && (\tau \text{ is homomorphism}).
\end{aligned}
$$

For a disjunction, the proof is analogous. For Property (2), simply use the above transformation, with $\Rightarrow$ replaced by $\Leftarrow$. Property (3) is Properties (1) and (2) combined. □

Corollary 2.1.19 *Under the conditions of Proposition 2.1.18:*

If $\mathcal{M}_1 \models_1 a \Rightarrow \mathcal{M}_2 \models_2 \tau(a)$ for any atom a, then $\mathcal{M}_1 \models_1 \varphi \Rightarrow \mathcal{M}_2 \models_2 \tau(\varphi)$ for an arbitrary $\mathbf{L}_1$-formula φ and $\mathcal{M}_1 \models_1 \Phi \Rightarrow \mathcal{M}_2 \models_2 \tau(\Phi)$ for any set Φ of $\mathbf{L}_1$-formulas.

If $\mathcal{M}_2 \models_2 \tau(a) \Rightarrow \mathcal{M}_1 \models_1 a$ for any atom a, then $\mathcal{M}_2 \models_2 \tau(\varphi) \Rightarrow \mathcal{M}_1 \models_1 \varphi$ for an arbitrary $\mathbf{L}_1$-formula φ and $\mathcal{M}_2 \models_2 \tau(\Phi) \Rightarrow \mathcal{M}_1 \models_1 \Phi$ for any set Φ of $\mathbf{L}_1$-formulas.

Proof: This follows from Proposition 2.1.18 by applying structural induction on φ. □

We have thus shown how satisfiability in a homomorphic image can be derived from satisfiability in the preimage. A similar proposition can be stated regarding the depth of a formula:

Proposition 2.1.20 *Let $\mathbf{L}_1$,$\mathbf{L}_2$ be two logics with depth functions d_1,d_2, respectively, and τ an $\mathbf{L}_1 - \mathbf{L}_2$ homomorphism. If $d_1(a) = d_2(\tau(a))$ for all $\mathbf{L}_1$-atoms a, then $d_1(\varphi) = d_2(\tau(\varphi))$ for an arbitrary $\mathbf{L}_1$-formula φ.*

We say that a homomorphism satisfying the property in Proposition 2.1.20 is *depth-preserving.* Notice that the depth of $\top$ and $\bot$ is preserved in *any* homomorphism, since these atoms are mapped to themselves.

Proof: We just need to show that the property $d_1(\varphi) = d_2(\tau(\varphi))$ is preserved under propositional concatenation, then the result follows by structural induction on φ. The key observation is that homomorphisms preserve the maximum function used in computing the depth of a nested formula: Assume that $d_1(\varphi_i) = d_2(\tau(\varphi_i))$ for $i = 1, \ldots, n$, and consider $\varphi = \varphi_1 \circ \ldots \circ \varphi_n$, where $\circ = \wedge, \vee$. Then

$$\begin{aligned} d_1(\varphi_1 \circ \ldots \circ \varphi_n) &= \max\{d_1(\varphi_1), \ldots, d_1(\varphi_n)\} \\ &= \max\{d_2(\tau(\varphi_1)), \ldots, d_2(\tau(\varphi_n))\} \\ &= d_2(\tau(\varphi_1) \circ \ldots \circ \tau(\varphi_n)) \\ &= d_2(\tau(\varphi_1 \circ \ldots \circ \varphi_n)), \end{aligned}$$

which establishes the proof. □

The fundamental task of model finding is to process a set of formulas in order to find a satisfying model, or to show that no such model exists. Since a theorem prover is unaware of the semantics of a logic, an algorithm must operate on the syntactic structure of the formula, and the steps of the algorithm must reflect the intended semantics (that is, the algorithm must prove sound and complete). We must use some construct to bridge the chasm between the syntax of a formula and the semantic notion of satisfiability. *Hintikka sets* or *saturated sets* [23, 39] are a commonly used construct for propositional as well as modal logics. We find it more convenient, however, to use a different construct which we call a *cover.* Speaking on a semantical level, covers are sets of formulas witnessing that a formula φ is verified : If $\mathcal{M} \models_g \varphi'$ for all formulas φ' in the cover, then $\mathcal{M} \models_g \varphi$. This is certainly true for $\{\varphi\}$; indeed, we will consider this set a cover for φ. Likewise, if all elements in a cover are atoms and we know that $\mathcal{M} \models_g a$ for every atom in it, then we know $\mathcal{M} \models_g \varphi$. Thus the existence of such an *atomic* cover settles the propositional reasoning part in our logic **L**. Covers must reflect the semantic asymmetry between conjunctions and disjunctions: Any set $\{\beta_j\}$ should serve as a cover for a disjunction $\beta_1 \vee \ldots \vee \beta_m$, for if $\mathcal{M} \models_g \beta_j$, then $\mathcal{M} \models_g \beta_1 \vee \ldots \vee \beta_m$. Conversely, a cover for $\alpha_1 \wedge \ldots \wedge \alpha_n$ should include $\{\alpha_1, \ldots, \alpha_n\}$ (or some of their respective formulas), for only if $\mathcal{M} \models_g \alpha_i$ for all $i = 1, \ldots, n$, then $\mathcal{M} \models_g \alpha_1 \wedge \ldots \wedge \alpha_n$. One consequence of this asymmetry is that not all subformulas of a formula φ contribute to a cover of φ. For example, $\varphi = a \vee (b \wedge \neg c)$ has a cover $\{a\}$,

which does not mention $(b \wedge \neg c)$ or any of its subformulas at all. This highlights a common phenomenon in propositional logics: Small subformulas of a formula can make it trivially satisfiable (or unsatisfiable), and an algorithm which can detect trivial subformulas will run faster than an algorithm which processes the entire formula. This is one reason why heuristics play an important role in Automated Theorem Proving. Having thus discussed the semantic connotations of covers, we now turn to their pure syntactic definition:

Definition 2.1.21 *A set Ψ of propositional formulas is called a* cover *of a formula φ, if $\Psi = \{\varphi\}$, or recursively:*

Case 1: $\varphi = \alpha_1 \wedge \ldots \wedge \alpha_n$, then Ψ is the union of covers Ψ_1 to Ψ_n of α_1 through α_n, respectively.

Case 2: $\varphi = \beta_1 \vee \ldots \vee \beta_m$, then Ψ is a cover of some β_j, $j = 1, \ldots, m$.

A set Ψ is a cover of a finite set of formulas Φ, if it is the union of covers for each formula in Φ.
A cover Ψ of Φ is minimal, *if Φ has no cover Ψ' so that $\Psi' \subset \Psi$.*
If Ψ_1 and Ψ_2 are covers, then we say that Ψ_2 is finer *than Ψ_1 (written $\Psi_1 \preceq \Psi_2$) if Ψ_2 is a cover of Ψ_1 itself. A cover is* atomic *iff all its elements are atoms.*

Example 2.1.22 The set $\Phi = \{\varphi, \varphi \vee \psi\}$ has two atomic covers, namely $\{\varphi\}$ and $\{\varphi, \psi\}$. The latter is not minimal.

Let us observe a trivial consequence of this definition:

Corollary 2.1.23 *Any cover of φ consists of propositional subformulas of φ. Any formula φ has exactly one cover which contains φ itself, namely the set $\{\varphi\}$.*

Proof: For the first part, note that in the entirety of Definition 2.1.21 only subformulas of the original formula φ are mentioned in its recursive applications, and the cover is constructed simply by collecting some of these. The second part follows from the first: By Definition 2.1.21, $\{\varphi\}$ is a cover of φ. All other covers of φ are constructed by finding (and taking the union of) covers of proper subformulas of φ; all of these covers contain only subformulas of these proper subformulas, so none can mention φ itself. □

Proposition 2.1.24 *The "is cover of" relation is a partial order on finite sets of formulas of* **L**. *Likewise, the "is finer than" relation is a partial order on the set of all covers of any given formula.*

Proof: Since $\{\varphi\}$ is a cover of φ, it is also a cover of itself, which shows reflexivity.
Next we show that a cover of a cover of φ is a cover of φ, by structural induction on φ: The only cover of an atomic formula is the formula itself, which, applied twice, establishes the base case. Let us now assume the hypothesis is valid for any subformula of φ, and let Ψ be a cover of φ. We wish to show that any cover of Ψ is a cover of φ. If $\Psi = \{\varphi\}$, this is certainly true. Alternatively, if φ is a disjunction, then Ψ is a cover of one of its disjuncts β_j. By the induction hypothesis, any cover of Ψ is a cover of β_j and hence of φ. If φ is a conjunction, then Ψ is the union of covers Ψ_i for each conjunct α_i. Again by the induction hypothesis, any cover of Ψ_i is itself a cover for α_i. It follows that any cover of Ψ (which by definition is a union of covers of its elements and hence a union of covers of Ψ_i) is a union of covers for the α_i, and thus a cover of φ. This concludes the induction step.
Finally, let Ψ be a cover of a finite *set* Φ, which by Definition 2.1.21 is a union of covers Ψ_i of its elements φ_i. By what we have just shown, any cover of Ψ_i is a cover of φ_i. By the same argument we used above for conjunctions, a cover of Ψ is a union of covers of the Ψ_i, which is a union of covers of the φ_i or in other words, a cover of Φ. This shows the transitivity of the relation.
Showing antisymmetry is even harder. Assuming that Ψ_1 and Ψ_2 are covers of one another, we wish to show that $\Psi_1 = \Psi_2$. We use induction on the (finite) cardinality $\min(|\Psi_1|, |\Psi_2|)$. The base case is the trivial case in which one of these sets is empty. In order to be a cover of an empty set, the other set must also be empty. We prove the inductive case by showing that Ψ_1 and Ψ_2 have a common element φ, and that $\Psi_1 - \{\varphi\}$ and $\Psi_2 - \{\varphi\}$ are still covers of one another. By the induction hypothesis, these remainders must be equal, which establishes the theorem. Let us pick a formula φ whose length in both sets is maximal. Without loss of generality, suppose that $\varphi \in \Psi_1$. Now every element of Ψ_1 is contained in a cover of some $\varphi' \in \Psi_2$. However, Corollary 2.1.23 says that each such element is a subformula of φ'. Since there is no formula in Ψ_2 longer than φ, (of which φ could be a proper subformula), the only possible choice for φ' is $\varphi' = \varphi$. Furthermore, all the covers of the other elements in Ψ_2 do not contain φ as a subformula. Finally, Corollary 2.1.23 also states that the only cover of φ containing φ is $\{\varphi\}$. Applied to our situation, it means that all other elements of Ψ_1 arise

from covers of other formulas in Ψ_2. To summarize, we have shown that $\Psi_1 - \{\varphi\}$ is a cover of $\Psi_2 - \{\varphi\}$. By repeating this argument with the roles of Ψ_1 and Ψ_2 exchanged (after all, we have shown that φ is also in Ψ_2), we show that $\Psi_2 - \{\varphi\}$ is a cover of $\Psi_1 - \{\varphi\}$. This concludes the induction proof.

For the second part of the theorem, note that the "is finer than" relation is simply a restriction of the "is cover of" relation to the set of all covers of a particular formula, and thus itself a partial order. □

We remark that the set of covers of a formula φ is never empty, since it always contains $\{\varphi\}$ which is also its coarsest cover.

For simplicity, the remaining results of this section are stated for sets of formulas Φ only, but they hold for formulas φ just as they do for the singleton sets $\{\varphi\}$.

Definition 2.1.25 *A* refinement *of a set of formulas Φ is a set of the form $\Phi - \{\alpha_1 \wedge \ldots \wedge \alpha_n\} \cup \{\alpha_1, \ldots, \alpha_n\}$, where $\alpha_1 \wedge \ldots \wedge \alpha_n \in \Phi$, or $\Phi - \{\beta_1 \vee \ldots \vee \beta_m\} \cup \{\beta_j\}$ for some $j = 1, \ldots, m$, where $\beta_1 \vee \ldots \vee \beta_m \in \Phi$.*

Proposition 2.1.26 *If Ψ is a cover of Φ, then any refinement of Ψ is a finer cover of Φ.*

Proof: We only need to show that a refinement Ψ' of Ψ is a cover of Ψ, i.e. a union of covers of the elements of Ψ. By transitivity, this refinement is also a cover of Φ.

Considering the first case in Definition 2.1.25, Case 1. of Definition 2.1.21 states that any cover Ψ_1 of $\{\alpha_1, \ldots, \alpha_n\}$ is a cover of $\{\alpha_1 \wedge \ldots \wedge \alpha_n\}$. In particular $\Psi_1 = \{\alpha_1, \ldots, \alpha_n\}$, being a cover of itself, is such a cover. As to the second case of Definition 2.1.25, Case 2. of Definition 2.1.21 states that any cover Ψ_2 of some β_j is a cover of $\{\beta_1 \vee \ldots \vee \beta_m\}$. Again, we choose $\Psi_2 = \{\beta_j\}$. Each of the remaining elements φ in Ψ is included in Ψ' according to Definition 2.1.25, so we can simply choose $\{\varphi\}$ as a cover for φ. In summary, every element of Ψ has a cover in Ψ', and every element of Ψ' is used in a cover of one of those elements, so Ψ' is indeed a cover of Ψ. □

Proposition 2.1.27 *A cover is maximally fine iff it is atomic.*

Proof: The only cover of a set of atoms is the set itself. Therefore, no cover can be strictly finer than a set of atoms. For the converse, suppose a cover Ψ contains a nonatomic formula φ which must be either a disjunction or a conjunction. Then a refinement of Ψ according to Definition 2.1.25 can be found, and Proposition 2.1.26 shows that Φ is not maximally fine. □

Proposition 2.1.28 *Every set of formulas has an atomic cover.*

Proof: Note that any formula φ has only finitely many sets of subformulas. Hence, any strictly descending $\preceq$-chain of covers for φ must terminate in a maximally fine cover, which by Proposition 2.1.27 is atomic. But then the union of such covers for each $\varphi \in \Phi$ is also atomic, and it is a cover of Φ. □

Let us prove a converse of Proposition 2.1.26, establishing that the refinements of a cover Φ are the immediate $\preceq$-successors of Φ:

Proposition 2.1.29 *Every cover of a set Φ other than Φ itself is a cover of a refinement of Φ.*

Proof: Let Ψ be a cover of Φ, i.e. a union of covers of the formulas in Φ. If $\Psi \neq \Phi$, then at least one formula $\varphi \in \Phi$ must have a cover other than $\{\varphi\}$ in Ψ. By Definition 2.1.21, this cover must be a union of covers of $\{\alpha_1, \ldots, \alpha_n\}$ (if $\varphi = \alpha_1 \wedge \ldots \wedge \alpha_n$), or a cover of some β_j (if $\varphi = \beta_1 \vee \ldots \vee \beta_m$). Therefore, Ψ is a union of covers of formulas in (and hence a cover of) $\Phi - \{\varphi\} \cup \{\alpha_1, \ldots, \alpha_n\}$ or $\Phi - \{\varphi\} \cup \{\beta_j\}$, respectively. But these sets are refinements of Φ according to Definition 2.1.25. □

By virtue of this proposition, we have shown that the $\preceq$-relation is discrete. The theory of finite ordered sets gives us this important converse:

Corollary 2.1.30 *Every cover of a* finite *set Φ other than Φ itself is a refinement of some cover of Φ.*

Moreover, we have:

Corollary 2.1.31 *Every cover of a* finite *set Φ is obtained through a finite chain of successive refinements of Φ.*

So far we have stated that any cover can be obtained through a chain of refinements in a given fixed order. Considering the freedom we have in choosing a formula from Φ for refining our cover, the question arises whether *any* order of refinements arrives at the same set of atomic covers for Φ, a property called *confluence*. Quite surprisingly, the answer is negative. Let us explore what can go wrong:

Example 2.1.32 The set $\Phi = \{p \vee q, (p \vee q) \wedge r\}$ has a refinement $\{p, (p \vee q) \wedge r\}$, and two successive refinements of the second formula yield an atomic cover $\{p, q, r\}$. However, if we

refine Φ using the second formula first, we get $\{p \vee q, r\}$, and no refinement of this set can produce the atomic cover $\{p, q, r\}$. Observe that this atomic cover is not minimal.

The trouble was that the formula $p \vee q$ was refined twice in the first case; in each instance, we can choose a different cover, something we could not do in the second case, as $p \vee q$ was evaluated only once. We take this into consideration in our next proposition:

Proposition 2.1.33 *Any cover Ψ of $\Phi = \Phi_1 \cup \cdots \cup \Phi_n$ is the union of covers of $\Phi_1, \ldots, \Phi_n$. Conversely, if $\Psi_1, \ldots, \Psi_n$ are covers of $\Phi_1, \ldots, \Phi_n$, respectively, then $\Psi_1 \cup \cdots \cup \Psi_n$* contains *a cover of $\Phi_1 \cup \cdots \cup \Phi_n$.*

Proof: By Definition 2.1.21, Ψ is the union of covers Ψ_φ of the elements $\varphi \in \Phi$. By the same definition, the sets $\bigcup\{\Psi_\varphi : \varphi \in \Phi_i\}$ are covers of the Φ_i, $i = 1, \ldots, n$, respectively, and their union is Ψ. For the converse, each Ψ_i contains covers for all $\varphi \in \Phi_i$. Define Ψ_φ as the cover for φ in Φ_{i_0}, where $i_0 = \min\{i = 1, \ldots, n : \varphi \in \Phi_i\}$. Then $\bigcup\{\Psi_\varphi : \varphi \in \Phi\}$ is evidently a cover of Φ, and it is wholly contained in $\Psi_1 \cup \cdots \cup \Psi_n$. □

Proposition 2.1.34 *Let Φ be a set of formulas, and $\alpha_1, \ldots, \alpha_n$, $\beta_1, \ldots, \beta_m$ formulas.*

1. *Every cover of $\{\alpha_1, \ldots, \alpha_n\}$ is a cover of $\{\alpha_1 \wedge \ldots \wedge \alpha_n\}$.*
2. *Every cover of $\{\alpha_1 \wedge \ldots \wedge \alpha_n\}$ other than $\alpha_1 \wedge \ldots \wedge \alpha_n$ itself is a cover of $\{\alpha_1, \ldots, \alpha_n\}$.*
3. *Every cover of $\{\beta_j\}$, $j = 1, \ldots, m$, is a cover of $\{\beta_1 \vee \ldots \vee \beta_m\}$.*
4. *Every cover of $\Phi \cup \{\beta_1 \vee \ldots \vee \beta_m\}$ other than $\beta_1 \vee \ldots \vee \beta_m$ itself is a cover of $\{\beta_j\}$ for some $j = 1, \ldots, m$.*
5. *Every cover of $\Phi \cup \{\alpha_1, \ldots, \alpha_n\}$ is a cover of $\Phi \cup \{\alpha_1 \wedge \ldots \wedge \alpha_n\}$.*
6. *Every cover of $\Phi \cup \{\alpha_1 \wedge \ldots \wedge \alpha_n\}$ not containing $\alpha_1 \wedge \ldots \wedge \alpha_n$ contains a cover of $\Phi \cup \{\alpha_1, \ldots, \alpha_n\}$,*
7. *Every cover of $\Phi \cup \{\beta_j\}$ is a cover of $\Phi \cup \{\beta_1 \vee \ldots \vee \beta_m\}$.*
8. *Every cover of $\Phi \cup \{\beta_1 \vee \ldots \vee \beta_m\}$ not containing $\beta_1 \vee \ldots \vee \beta_m$ contains a cover of $\Phi \cup \{\beta_j\}$ for some $j = 1, \ldots, m$.*
9. *The* minimal *covers of $\Phi \cup \{\alpha_1, \ldots, \alpha_n\}$ are exactly the minimal covers of $\Phi \cup \{\alpha_1 \wedge \ldots \wedge \alpha_n\}$ which do not contain $\alpha_1 \wedge \ldots \wedge \alpha_n$.*

10. *The* minimal *covers of* $\Phi \cup \{\beta_j\}$, $j = 1, \ldots, m$, *are exactly the minimal covers of* $\Phi \cup \{\beta_1 \vee \ldots \vee \beta_m\}$ *which do not contain* $\beta_1 \vee \ldots \vee \beta_m$.

11. *If* $\alpha_i \in \Phi$ *for all* $i = 1, \ldots, n$, *then every cover of* Φ *is a cover of* $\Phi \cup \{\alpha_1 \wedge \ldots \wedge \alpha_n\}$; *hence all minimal covers of* $\Phi \cup \{\alpha_1 \wedge \ldots \wedge \alpha_n\}$ *are covers of* Φ.

12. *If* $\beta_j \in \Phi$ *for some* $j = 1, \ldots, m$, *then every cover of* Φ *is a cover of* $\Phi \cup \{\beta_1 \vee \ldots \vee \beta_m\}$; *hence all minimal covers of* $\Phi \cup \{\beta_1 \vee \ldots \vee \beta_m\}$ *are covers of* Φ.

Proof: Parts (1) through (4) follow immediately from Definition 2.1.21. From these, we get parts (5) through (8) by applying Proposition 2.1.33. In (9), let Ψ be a minimal cover of $\Phi \cup \{\alpha_1, \ldots, \alpha_n\}$. By (5), Ψ is a cover of $\Phi \cup \{\alpha_1 \wedge \ldots \wedge \alpha_n\}$. Furthermore, Ψ contains covers for $\alpha_1, \ldots, \alpha_n$, whose union already is a cover for $\alpha_1 \wedge \ldots \wedge \alpha_n$, so if Ψ contained $\alpha_1 \wedge \ldots \wedge \alpha_n$, it would not be minimal. Now suppose that $\Phi \cup \{\alpha_1 \wedge \ldots \wedge \alpha_n\}$ has a cover Ψ_1 which is a strict subset of Ψ. Then by (6), Ψ_1 contains a cover Ψ_2—still a strict subset of Ψ—which is a cover of $\Phi \cup \{\alpha_1, \ldots, \alpha_n\}$. This is a contradiction to the minimality of Ψ. Therefore Ψ is minimal among the covers of $\Phi \cup \{\alpha_1 \wedge \ldots \wedge \alpha_n\}$ as well. Conversely, let Ψ' be any minimal cover of $\Phi \cup \{\alpha_1 \wedge \ldots \wedge \alpha_n\}$ which does not contain $\alpha_1 \wedge \ldots \wedge \alpha_n$. Then by (6) a subset Ψ_1 of Ψ' is a cover of $\Phi \cup \{\alpha_1, \ldots, \alpha_n\}$. Now take any subset Ψ_2 which is also a cover of $\Phi \cup \{\alpha_1, \ldots, \alpha_n\}$. Now (5) states that Ψ_2 is a cover of $\Phi \cup \{\alpha_1 \wedge \ldots \wedge \alpha_n\}$. But since Ψ' was minimal, we must have $\Psi_2 = \Psi_1 = \Psi'$, which shows that Ψ' is a minimal cover of $\Phi \cup \{\alpha_1, \ldots, \alpha_n\}$. This completes the proof of (9). For (10) the proof is analogous. The first half of (11) is a special case of (5): Note that $\Phi \cup \{\alpha_1, \ldots, \alpha_n\} = \Phi$. For the second half, since $\Phi \subseteq \Phi \cup \{\alpha_1 \wedge \ldots \wedge \alpha_n\}$, any cover Ψ of $\Phi \cup \{\alpha_1 \wedge \ldots \wedge \alpha_n\}$ has a cover Ψ' of Φ as a subset. Since Ψ' itself is a cover of $\Phi \cup \{\alpha_1 \wedge \ldots \wedge \alpha_n\}$, Ψ can only be minimal if $\Psi = \Psi'$. In other words, the minimal covers are all found among the covers of Φ. This shows (11), and (12) follows by the same argument. □

Corollary 2.1.35 *Using the same notation as in Proposition 2.1.34, the minimal* atomic *covers of* $\Phi \cup \{\alpha_1, \ldots, \alpha_n\}$ *are exactly the minimal atomic covers of* $\Phi \cup \{\alpha_1 \wedge \ldots \wedge \alpha_n\}$, *and the minimal atomic covers of* $\Phi \cup \{\beta_j\}$, $j = 1, \ldots, m$, *are exactly the minimal atomic covers of* $\Phi \cup \{\beta_1 \vee \ldots \vee \beta_m\}$.

Proof: This follows from parts (9) and (10) of Proposition 2.1.34. Note that atomic covers do not contain any α- and β-formulas, which makes the extra conditions in parts (9) and (10) redundant. □

This corollary proves the confluence property for finding the *minimal* atomic covers of a set Φ: No matter which α- or β-formula we single out from Φ in order to obtain a refinement of Φ, all chains of refinements eventually result in the same set of minimal atomic covers. The next two results demonstrate how the $\preceq$-relation is preserved under homomorphisms in either direction:

Theorem 2.1.36 *Let τ be an $\mathbf{L}_1 - \mathbf{L}_2$ homomorphism, φ an $\mathbf{L}_1$-formula, and Φ, Ψ sets of $\mathbf{L}_1$-formulas.*

1. *If Ψ is a cover for φ (or Φ), then $\tau(\Psi)$ is a cover for $\tau(\varphi)$ (or $\tau(\Phi)$).*
2. *If τ preserves atoms and Ψ is an* atomic *cover for φ (or Φ), then $\tau(\Psi)$ is an atomic cover for $\tau(\varphi)$ (or $\tau(\Phi)$).*

Proof: We use structural induction over φ. For the coarsest cover $\{\varphi\}$, we see that $\tau(\{\varphi\}) = \{\tau(\varphi)\}$ is a cover of $\tau(\varphi)$. (This settles the proof if φ is an atom, as $\{\varphi\}$ is its only cover.) Now we show that the Theorem is preserved under disjunctions: A set Ψ is a cover of $\varphi = \beta_1 \vee \ldots \vee \beta_m$, iff it is a cover of one of its disjuncts β_j. By the induction hypothesis, $\tau(\Psi)$ is a cover of $\tau(\beta_j)$. But since $\tau(\varphi) = \tau(\beta_1) \vee \ldots \vee \tau(\beta_m)$, $\tau(\Psi)$ is also a cover of $\tau(\varphi)$. Finally, the theorem is preserved under conjunctions: Let Ψ be a cover of $\varphi = \alpha_1 \wedge \ldots \wedge \alpha_n$, i.e. the union of covers Ψ_i of its conjuncts α_i, $i = 1, \ldots, n$. By the induction hypothesis, each $\tau(\Psi_i)$ is a cover of $\tau(\alpha_i)$. But since $\tau(\varphi) = \tau(\alpha_1) \wedge \ldots \wedge \tau(\alpha_n)$, $\tau(\Psi) = \tau(\Psi_1) \cup \ldots \cup \tau(\Psi_n)$ is a cover of $\tau(\varphi)$. The proof follows by structural induction. For sets Φ, the proof is analogous to the induction step for conjunctions.
Part 2 of the proof follows mainly from part 1; additionally, if all elements in Ψ are atoms, then so are all elements in $\tau(\Psi)$, since τ preserves atoms. Therefore, $\tau(\Psi)$ is an atomic cover whenever Ψ is. □

We also get a "converse" of Theorem 2.1.36. Note that τ need not map atoms to atoms:

Theorem 2.1.37 *Let τ be an $\mathbf{L}_1 - \mathbf{L}_2$ homomorphism and Φ a set of $\mathbf{L}_1$-formulas. Then every (atomic) cover Ψ' of $\tau(\Phi)$ is the τ-image of some (atomic) cover Ψ of Φ.*

Proof: For $\Psi' = \tau(\Phi)$, the coarsest cover of $\tau(\Phi)$, the set $\Psi = \Phi$ is the desired cover of Φ. Next we show the theorem in case Ψ' is any *refinement* of $\tau(\Phi)$: Assume Ψ' has been obtained by replacing some conjunctive formula $\alpha'_1 \wedge \ldots \wedge \alpha'_n$ in $\tau(\Phi)$ with $\{\alpha'_1, \ldots, \alpha'_n\}$, and let $\varphi_1, \ldots, \varphi_m$ be the τ-preimages of $\alpha'_1 \wedge \ldots \wedge \alpha'_n$ which are elements of Φ. (Remember that

several preimages may be mapped to the same formula.) None of the φ_j can be atomic, since τ maps atoms to atoms only. Nor can they be disjunctions or conjunctions of different arities, since τ as a homomorphism maps disjunctions to disjunctions, and conjunctions to conjunctions of the same arity. So each φ_j must be of the form $\alpha_{j,1} \wedge \ldots \wedge \alpha_{j,n}$, and furthermore we have $\tau(\alpha_{j,1} \wedge \ldots \wedge \alpha_{j,n}) = \tau(\alpha_{j,1}) \wedge \ldots \wedge \tau(\alpha_{j,n}) = \alpha'_1 \wedge \ldots \wedge \alpha'_n$, giving us the identities $\alpha'_i = \tau(\alpha_{j,i})$, $i = 1, \ldots, n$[2]. Thus for each pre-image φ_j of $\alpha'_1 \wedge \ldots \wedge \alpha'_n$ the set $\{\alpha_{j,1}, \ldots, \alpha_{j,n}\}$ provides a refinement of $\{\varphi_j\}$. If we think of all other elements in Φ as covers for themselves, the set $\Psi = \Phi - \{\varphi_1, \ldots, \varphi_m\} \cup \bigcup_{j=1}^{m} \{\alpha_{j,1}, \ldots, \alpha_{j,n}\}$ is really a union of covers of the elements in Φ and hence a cover of Φ.

If Ψ' has been obtained by replacing a disjunction, the argument is similar. Now we apply Corollary 2.1.31 which guarantees that any cover Ψ' of $\tau(\Phi)$ is obtained through a finite chain of refinements Ψ'_k, starting from $\Psi'_0 = \tau(\Phi)$. For each step k, we can apply our result above: Assuming we have found a cover Ψ_k of Φ so that $\tau(\Psi_k) = \Psi'_k$, the refinement Ψ'_{k+1} has a preimage Ψ_{k+1} which is a cover of Ψ_k (and hence a cover of Φ by transitivity), so that $\tau(\Psi_{k+1}) = \Psi'_{k+1}$. We continue until $\Psi'_k = \Psi'$. Finally, if the final cover Ψ' consists only of atoms, then so does its preimage, since non-atoms are always mapped to non-atoms. □

We are now ready to substantiate our earlier intuitive discussion and establish the relationship between the syntactical notion of covers and the semantical notion of verifying structures and satisfiability:

Theorem 2.1.38 *A set of formulas Φ is verified by $\mathcal{M}$ iff it has an atomic cover Ψ so that $\mathcal{M} \models_g \Psi$.*

Proof: To show the "only if" part, consider the set of all covers of Φ which are verified by $\mathcal{M}$. Since $\mathcal{M} \models_g \Phi$, this set is nonempty. Now take a maximally fine cover Ψ in this set, and assume this cover is nonatomic. First, suppose there exists a conjunction $\alpha_1 \wedge \ldots \wedge \alpha_n$ in Ψ (which must be verified by $\mathcal{M}$). Then $\mathcal{M}$ also verifies each of the α_i. By replacing the conjunction with the set of all α_i, we obtain a refinement which is also verified by $\mathcal{M}$, a contradiction to the maximality assumption. Now suppose Ψ contains a disjunction

[2]Note that homomorphisms as we defined them preserve the *syntactic* structure of a formula. As such, the $\alpha_{j,i}$ must correspond to the α'_i in the given order, and even if some of the α'_i are identical, none of the multiple occurrences can be omitted. It is possible to define a weaker form of homomorphisms which would allow for commutativity as well as basic simplification of formulas according to idempotence etc.; we could show that this theorem still holds. However, such extensions are not needed and would unduly complicate the issue.

$\beta_1 \vee \ldots \vee \beta_m$ (which again must be verified by $\mathcal{M}$). By the definition of $\models_g$, $\mathcal{M}$ must satisfy at least one of the disjuncts β_j. Replacing $\beta_1 \vee \ldots \vee \beta_m$ with this β_j yields a refinement of Ψ verified by $\mathcal{M}$, contradicting the maximality assumption. Therefore, we must conclude that Ψ is atomic.

To prove the "if" part, we use structural induction over the formulas in Φ. If Φ consists entirely of atoms, then Φ itself is the only cover of Φ, so we know that $\mathcal{M} \models_g \Phi$. Now we show that the statement is preserved under conjunctions. So let $\varphi = \alpha_1 \wedge \ldots \wedge \alpha_n$, and Ψ be an atomic cover of φ verified by $\mathcal{M}$. Then by Definition 2.1.21, Ψ must be the union of covers $\Psi_1, \ldots, \Psi_n$ for $\alpha_1, \ldots, \alpha_n$, respectively. From our assumption $\mathcal{M} \models_g \Psi$ we conclude $\mathcal{M} \models_g \Phi_i$ by Corollary 2.1.2, and the induction hypothesis gives us $\mathcal{M} \models_g \alpha_i$, for $i = 1, \ldots, n$. But this proves $\mathcal{M} \models_g \varphi$. Secondly, if $\varphi = \beta_1 \vee \ldots \vee \beta_m$, then Φ is a cover for one of the β_j. By the induction hypothesis, we get $\mathcal{M} \models_g \beta_j$ for this disjunct, and hence $\mathcal{M} \models_g \varphi$. We have shown that the statement is also preserved under disjunctions, and the principle of structural induction completes the proof. □

By virtue of this theorem, we can show any formula φ satisfiable by providing a model for an atomic cover of φ. Thus the problem $\mathcal{M} \models_g \varphi$ ($\mathcal{M} \models_g \varphi$) is reduced to the problem $\mathcal{M} \models_g \Psi$ ($\mathcal{M} \models_g \Psi$), where Ψ is a set of atoms. We have thus encapsulated propositional reasoning in our logic **L**, so in any incarnation of propositional logic we encounter later, we can focus on the characteristic features of their elementary building blocks, namely the atoms.

2.2 Classical Propositional Logic

For a simple first application of our theory, let us consider classical propositional logic itself. This is not just an academic exercise but will provide us with important basic results to be used later. As we mentioned in the introduction to this chapter, we start out with a set P of propositional variables which we also call *positive literals.* Variables $p \in P$ may be negated, thus forming *negative literals* $\neg p$. Together with the two propositional constants $\top$ and $\bot$, the positive and negative literals, form the set A of *propositional atoms.* (There are no atoms other than these.) As discussed informally above, a structure must specify how atoms are mapped to truth values. We provide the following definition:

Definition 2.2.1 *A* valuation *is a set* $V \subseteq P$.

In the literature, valuations are commonly defined as total functions $v : P \mapsto \{true, false\}$. The equivalence between the two definitions is obvious and illustrates our intuition in defining V: Any variable $p \in V$ is assigned *true* (in v), and any variable $p \in P - V$ is assigned *false*. (We see that a variable can never be assigned both *true* and *false* at the same time, so all valuations are consistent.)

Definition 2.2.2 *Given a set of propositions P, the logic* $\mathbf{PL_{NNF}}$ *(restriction of propositional logic to formulas in negation normal form) is exactly the smallest propositional logic containing all propositional atoms, with valuations serving as models, and the semantic relation $\models_p$ extending $\models_g$ by defining it on positive and negative literals as follows:*

$$V \models_p p \quad \textit{iff} \quad p \in V$$
$$V \models_p \neg p \quad \textit{iff} \quad p \notin V.$$

Definition 2.2.2 provides an easy way of deciding whether V satisfies a given set Ψ_A of atoms: Every positive literal $p \in \Psi_A$ must be included in V, and for every negative literal $\neg p \in \Psi_A$, p must not be included. If Ψ_A contains $\bot$, then no valuation satisfies it. More precisely, we define:

Definition 2.2.3 *A valuation V is* consistent with *a set Ψ_A of propositional atoms (or, Ψ_A is V-*consistent*), if $\bot \notin \Psi_A$ and $\Psi_A \cap P \subseteq V \subseteq \{p \in P : \neg p \notin \Psi_A\}$. A set Ψ_A is called* consistent *whenever such a valuation V exists.*

Applying Definitions 2.1.1 and 2.2.2, we easily conclude that the consistent valuations are exactly the satisfying valuations for a set of atoms:

Corollary 2.2.4 *V is consistent with a set of atoms Ψ_A iff $V \models_p \Psi_A$. Hence, a set Ψ_A is consistent iff it is satisfiable.*

Lemma 2.2.5 *A valuation V which is consistent with sets $\Phi_1, \ldots, \Phi_n$ of propositional atoms is also consistent with $\bigcup_{i=1}^{n} \Phi_i$.*

Proof: If $\bot \notin \Phi_i$ for any $i = 1, \ldots, n$, then $\bot$ is also not in the union of these sets. Likewise, if the subset relationship in Definition2.2.3 holds for all sets, it also holds for their union. □

Recalling Theorem 2.1.38, we can decide whether a set of formulas Φ is satisfied by a given valuation V (which is the reasoning task of *model checking*), simply by finding an atomic cover satisfied by V. Combining this with Corollary 2.2.4 above, we get:

Proposition 2.2.6 *Given a set Φ of $\mathbf{PL_{NNF}}$-formulas and a valuation V, we have $V \models_p \Phi$ iff Φ has a V-consistent atomic cover.*

Proof: According to Theorem 2.1.38, $V \models_p \varphi$ if and only if $V \models_p \Psi$ for some atomic cover Ψ of φ, which, as shown in Corollary 2.2.4, is the case exactly if V is consistent with Ψ. □

Let us now turn to the reasoning task of *satisfiability checking*, i.e. showing that a $\mathbf{PL_{NNF}}$-formula φ has a valuation which satisfies it. Again, Theorem 2.1.38 allows us to reduce this task to satisfiability checking for sets of atoms, which has a very nice characterization:

Definition 2.2.7 *A* (propositional) clash *in a set S of propositional formulas is an occurrence of $\bot$, or of two complementary literals $p, \neg p$, in S.*

Proposition 2.2.8 *A set of atoms is satisfiable iff it is clash-free. Hence, a set Φ of $\mathbf{PL_{NNF}}$-formulas is satisfiable iff it has a clash-free cover of propositional atoms.*

Proof: For the first part of the proposition, take a set Ψ_A of atoms. Suppose Ψ_A contains $\bot$, then no valuation is consistent with it. Now suppose that Ψ_A contains a clash of the form $p, \neg p$. Then $\Psi_A \cap P \not\subseteq \{p \in P : \neg p \notin \Psi_A\}$, so no valuation satisfying the inclusion relation in Definition 2.2.3 can be found. To summarize, if Ψ_A contains a clash, then no valuation is consistent with it, and by Corollary 2.2.4 no valuation satisfies it. For the converse, suppose Ψ_A is clash-free. Then $\Psi_A \cap P \subseteq \{p \in P : \neg p \notin \Psi_A\}$, and any set V of propositions "in between" these two sets satisfies Definition 2.2.3 and thus, by Corollary 2.2.4, provides a satisfying valuation. (We are free to include or not include into V any $p \in P$ which is not mentioned in Ψ_A at all.)

For the second part, Theorem 2.1.38 stated that any model for Φ is a model for some atomic cover Ψ and vice versa. So Φ is satisfiable iff it has some satisfiable atomic cover Ψ. By the first part of this corollary, this is the case iff Ψ is clash-free. □

Corollary 2.2.9 *A consistent set Φ of atoms has at most $|P| + 1$ elements.*

Proof: For each variable $p \in P$, Φ may contain p or $\neg p$ but not both, or else it would not be clash-free. Likewise, Φ may not contain *false*, but it may contain $\top$, for a total of at most $|P| + 1$ elements. □

We will now use the results of this section to develop a characterization of satisfiability in $\mathbf{PL_{NNF}}$ which uses semantic structures different from valuations and more similar to atomic covers. First, we discovered that it was not necessary to specify a total valuation, but we only need to assign truth values to those variables which actually occur in an atomic cover. This gives rise to the notion of *partial valuations* [16]. However, we generalize even further: Since it is possible for atomic covers to contain clashes, we allow our semantic structures to contain clashes as well, and we can define a meaningful relation $\models$ even for these. (Semantic structures with clashes do not satisfy every formula, as one might suppose.) In order to get an equivalent logic to $\mathbf{PL_{NNF}}$, we will consider exactly the clash-free structures as models for a formula.

The advantage of considering $\models$ instead of $\models$ is that $\models$ is monotone wrt the $\subseteq$ relation on sets of atoms. Under certain assumptions, this idea can be generalized to more complex logics, such as the logic of labelled formulas we will consider later in this work. As we will see, the monotonicity property will pay dividends, enabling us to specify an efficient model finding algorithm and prove it sound and complete.

Theorem 2.2.10 *The logic* $\mathbf{PL_{NNF}}$ *is equivalent to the following logic, defined on the same language of propositional formulas in NNF, whose semantic structures are sets* Ψ_A *of propositional atoms, whose models are clash-free sets of atoms, with the relation* $\models_{pp}$ *defined as an extension of* $\models_g$ *as follows:*

$$\Psi_A \models_{pp} l \text{ iff } l \in \Psi_A,$$

and $\Psi_A \models_{pp} \varphi$ *iff* Ψ_A *is clash-free and* $\Psi_A \models_{pp} \varphi$.

Proof: We need to show that every $\models_p$-satisfiable set of $\mathbf{PL_{NNF}}$-formulas is $\models_{pp}$-satisfiable and vice versa. First we recall Theorem 2.1.38, saying that $\mathcal{M} \models_g \Phi$ iff Φ has an atomic cover Ψ so that $\mathcal{M} \models_g \Psi$. This theorem applies to both our logics, since they are propositional logics. Hence we can restrict ourselves to the case where Φ is a set of propositional atoms, in which case a semantic structure satisfies Φ iff it is clash-free and verifies all its atoms.

First, suppose that Φ has a satisfying valuation V according to Definition 2.2.2. We define $\Psi_A = V \cup \{\neg p : p \in P - V\}$. Since every propositional atom occurs only once in Ψ_A (and $\bot$ does not), this set must be clash-free. Now consider any positive literal $p \in \Phi$. We have $V \models_p p$ iff $p \in V$ iff $p \in \Psi_A$ iff $\Psi_A \models_{pp} p$. Similarly for any negative literal $\neg p \in \Phi$, we have $V \models_p \neg p$ iff $p \notin V$ iff $\neg p \in \Psi_A$ iff $\Psi_A \models_{pp} \neg p$. Finally, valuations on the one hand and sets

Ψ_A on the other hand always verify $\top$ and never verify $\bot$, so we have shown that $V \models_p a$ iff $\Psi_A \mathrel{\sim\!\!\!\!\!\!\models}_{pp} a$ for all atoms.

Conversely, suppose that $\Psi_A \models_{pp} \Phi$. Since Ψ_A is clash-free, it does not contain $\bot$. Take any valuation which is consistent with Ψ_A according to Definition 2.2.3. (Such a valuation can be found, since $\Psi_A \cap P \subseteq \{p \in P : \neg p \notin \Psi_A\}$ holds thanks to the fact that p and $\neg p$ never occur simultaneously in Ψ_A.) From the set inclusion in Definition 2.2.3 we infer that $p \in \Psi_A$ implies $p \in V$, and $\neg p \in \Psi_A$ implies $p \notin V$; since this holds for all propositions p in Φ, we infer that $\Psi_A \models_{pp} \Phi$ implies $V \models_p \Phi$. □

Let us make explicit the connection between sets Ψ_A verifying Φ and atomic covers of Φ:

Proposition 2.2.11 *We have $\Psi_A \mathrel{\sim\!\!\!\!\!\!\models}_{pp} \Phi$ iff Ψ_A contains all the literals occurring in some atomic cover of Φ which does not contain $\bot$.*

Proof: As in the last proof, we use the fact that $\Psi_A \mathrel{\sim\!\!\!\!\!\!\models}_{pp} \Phi$ iff $\Psi_A \mathrel{\sim\!\!\!\!\!\!\models}_{pp} \Psi$ for some atomic cover Ψ of Φ. If Ψ contains $\bot$, this can never be the case; and for a $\bot$-free set Ψ, $\Psi_A \mathrel{\sim\!\!\!\!\!\!\models}_{pp} \Psi$ iff Ψ_A contains all the literals in Ψ. □

Now use the subset relation as the partial order on semantic structures. With regard to this order, we can prove that $\mathrel{\sim\!\!\!\!\!\!\models}_{pp}$ is monotone, as suggested above:

Theorem 2.2.12 *The relation $\mathrel{\sim\!\!\!\!\!\!\models}_{pp}$ is monotone wrt the subset relation $\subseteq$ on semantic structures (sets of atoms).*

Proof: Let $\Psi_A \subseteq \Psi'_A$, and $\Psi_A \mathrel{\sim\!\!\!\!\!\!\models}_{pp} a$ for some propositional atom. This is always true (regardless of the semantic structure) if $a = \top$, and always false if $a = \bot$. For any literal, we have $\Psi_A \mathrel{\sim\!\!\!\!\!\!\models}_{pp} l$ iff $l \in \Psi_A$, which implies $l \in \Psi'_A$ and hence $\Psi'_A \mathrel{\sim\!\!\!\!\!\!\models}_{pp} l$. We have thus shown that $\cdot \mathrel{\sim\!\!\!\!\!\!\models}_{pp} a$ is monotone for all atoms; Theorem 2.1.8 showed that monotonicity is preserved under propositional concatenation, and the principle of structural induction (Theorem 2.1.6) entails the monotonicity of $\mathrel{\sim\!\!\!\!\!\!\models}_{pp}$ in general. □

Thanks to monotonicity, the lattice formed by all semantic structures (i.e. sets of propositional atoms) corresponds to the lattice of sets of $\mathbf{PL_{NNF}}$-formulas verified by the respective structures. The top and bottom elements are the complete structure (containing all atoms) and the empty structure, respectively; the top element verifies all formulas (except those all of whose atomic covers contain $\bot$), whereas the bottom element verifies only a few tautologies (e.g. disjunctions containing $\top$). A particularly useful application is this: Consider a set $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ (or similarly for a conjunction $\varphi_1 \wedge \ldots \wedge \varphi_n$). If $\Psi_1 \mathrel{\sim\!\!\!\!\!\!\models}_{pp} \varphi_1, \ldots, \Psi_n \mathrel{\sim\!\!\!\!\!\!\models}_{pp} \varphi_n$

have been found, then $\Psi = \bigcup_{i=1}^{n} \Psi_i \models_{pp} \Phi$. This principle is commonly called *model merging*. Moreover, it demonstrates the *extensibility* of structures: whenever another formula φ_{n+1} is added to Φ at a later time, Ψ can be extended simply by adding a set of atoms Ψ_{n+1}, and $\Psi \cup \Psi_{n+1}$ covers the new set formulas.

Of course there is no guarantee that the extension of a *model* is still clash-free and hence a model. However, it is much easier to test whether a set of propositional atoms is clash-free than to find a new model "from scratch". If the extension turns out to be clash-free, we have found a model with little effort. Moreover, even if the extension is not clash-free, there may be ways to repair the clash. For instance, beginning with this extension, one might perform a local search on structures verifying all formulas, by removing atoms participating in a clash and replacing them so as to find a structure which is clash-free. In our logic of labelled formulas, the model-finding algorithm we will propose in Chapter 6 is based on the same principle: iterating over a model for a subset of Φ, we will add elements to the model so as to cover more formulas in Φ, and then *deterministically* repair all ensuing clashes, so as to find a model for a larger subset, until all formulas in Φ are satisfied.

A similar approach underlies the connection calculus [11]: Our atomic covers correspond to *paths* which "satisfy" a formula; paths can have clashes (here called *connections*), and if they do, they do not qualify as models, so a procedure for removing clashes (or enumerating all possible paths in order to find a clash-free path) must be given.

2.3 Complexity Results

At the end of this chapter, we would like to state some complexity results for the various reasoning tasks in $\mathbf{PL_{NNF}}$. Most of these results are well-known (see e.g. [83] for an overview), but we would like to show that essentially the same complexities are attained using our definitions of satisfiability and models.

Before we start, we need to be clear about how we measure the complexity. Given a set Φ of formulas, we could measure the complexity of reasoning with Φ in terms of the number of formulas in Φ. But this measure is biased: for, why should a conjunction $\bigwedge_{i=1}^{n} \alpha_i$ be measured differently than a set $\{\alpha_i : i = 1, \ldots, n\}$, when these two are equivalent? Instead, we can measure the number of propositional atoms in Φ, counting multiplicities. The number of propositional subformulas Φ is an equivalent measure, as it is bounded by twice the number of atoms. The only disadvantage is that atoms themselves may need more

than constant space and time to be stored and reasoned with. This is especially true in logics we will discuss later, where propositional atoms are themselves made up of formulas. The total number of characters over some alphabet needed to store Φ would be the fairest measure, and it would be te closest to actual machine storage and processing requirements. But this is not always practical either: we should not really be concerned, for instance, about the number of characters needed to store the name of a propositional variable. And when we state results on general propositional logics, we really do not know anything about the size of the atoms we might reason with. For the purposes in this section, we decide upon the second choice: define $|\Phi|$ as the number of atoms in Φ.

The complexity of elementary set operations also depends on how the sets are implemented. For example, the set membership problem can be solved in linear, logarithmic or (almost) constant time, depending if, and how, the elements of the set are ordered. We should not be concerned about such optimization issues (although these play a vital role in actual implementations). Instead we declare:

- Set membership tests for atoms are considered elementary.
- Computing the union or intersection of n sets $S_1, \ldots, S_n$ takes $\Sigma_{i=1}^{n} |S_i|$ elementary steps, allowing one step for "reading" each element in each of the S_i.
- A subset test $S_1 \subseteq S_2$ takes $|S_1|$ steps, since it can be expressed as $|S_1|$ membership tests.

Verifying whether a given set of atoms constitutes an atomic cover for a formula is certainly an interesting question:

Proposition 2.3.1 *For any propositional logic, given a set Φ of formulas and a set of atoms Ψ, it takes $O(|\Phi|)$ set membership tests to verify whether Ψ contains an atomic cover for Φ.*

Proof: Let us state the following, easy-to-prove properties (compare with Definition 2.1.21):

- Ψ contains a cover for an atom a iff $a \in \Psi$.
- Ψ contains a cover for $\alpha_1 \wedge \ldots \wedge \alpha_n$ iff Ψ contains a cover for all α_i, $i = 1, \ldots, n$.
- Ψ contains a cover for $\beta_1 \vee \ldots \vee \beta_m$ iff Ψ contains a cover for at least one β_j, $j = 1, \ldots, m$.
- Ψ contains a cover for Φ, iff Ψ contains a cover for all $\varphi \in \Phi$.

Beginning with the atoms, we can thus decide "bottom up" for any subformula φ of any formula in Φ whether Ψ contains a cover of φ. Each subformula needs to be evaluated at most once, and we incur exactly one membership test for each atom in Φ. In total, no more than $|\Phi|$ set membership tests are required. □

In this work, we will never have to verify whether a set of atoms is *exactly* a cover of a given formula. Thus, the following proposition, though intriguing, is only of academic interest. We mention it here without proof:

Proposition 2.3.2 *For any propositional logic, given a set Φ of formulas and a set of atoms Ψ, verifying whether Ψ is an atomic cover for Φ is strongly NP-complete.*

Surprisingly, *finding* an atomic cover for a given formula is very easy:

Proposition 2.3.3 *For any propositional logic, given a set Φ of formulas, it takes $O(|\Phi|)$ steps to find an atomic cover for Φ.*

Proof: To obtain an atomic cover of a conjunction, we must take the union of covers of its conjuncts. For a cover of a disjunction, we must choose a disjunct and take its respective cover. It does not matter which disjunct we choose: any of them will give rise to an atomic cover. Finally, for an atom, we must include the atom itself into our cover. Hence, we can traverse each formula in Φ in top-down fashion, stepping into all conjuncts of a conjunction, and into the chosen disjunct in each disjunction, gathering the atomic subformulas along the way, which results in an atomic cover for Φ. Since we visit each subformula at most once, the entire procedure takes $O(|\Phi|)$ steps. □

If Φ is in clause normal form, finding a cover is particularly easy: Just pick a disjunct from each of the formulas in Φ. This cover is strongly reminiscent of the initial literal assignment in the Disconnection Calculus [12, 65], where it is known as a *path*.

This result seems to contradict the well-known NP-completeness property for satisfiability checking in propositional logic. However, we must remember that Corollary 2.2.8 warrants only the existence of a *clash-free* cover as a necessary and sufficient criterion for the satisfiability of a $\mathbf{PL_{NNF}}$-formula Φ. Therefore, the following result comes as no surprise at all:

Proposition 2.3.4 *Let Φ be a set of $\mathbf{PL_{NNF}}$-formulas. Finding whether Φ has a clash-free atomic cover is an NP-complete problem.*

Proof: To show that the problem is in NP, we propose a nondeterministic "guess" Ψ. All covers of Φ only contain atoms which actually occur in Φ, so we can easily restrict Ψ to be of size $O(\Phi)$. It is easy to verify that Ψ is clash-free: We just need to check that p and $\neg p$ do not both occur in Ψ, and that $\bot$ is absent from Ψ. This involves at most $O(\min(|P|, |\varphi|))$ membership tests. Furthermore, we showed in Proposition 2.3.1 that Ψ can be verified in linear time to *contain* an atomic cover for φ. In fact, the algorithm we sketched in Proposition 2.3.1 returns such an atomic cover. Being a subset of the clash-free set Ψ, it is also clash-free and hence provides a solution to the problem. Therefore, we proved in linear time, based on our initial guess, that Φ has a clash-free atomic cover. (Note that this cover need not be identical to our guess Ψ.) So our problem is indeed in NP.
On the other hand, Proposition 2.2.8 says that a clash-free atomic cover exists iff φ is satisfiable. Hence, the satisfiability problem in propositional logic (which is well-known to be NP-complete) can be polynomially reduced to the problem of finding a clash-free atomic cover, simply by converting a propositional formula into NNF. So our problem must be NP-hard. □

Finally, let us recall our result from Proposition 2.2.6 regarding model checking. Of course it would be hugely inefficient to perform model checking by using this proposition naïvely. We would not want to find all atomic covers for a given set Φ and check each of them against our valuation V, as there may be exponentially many different covers. The conventional method of evaluating the formulas in a bottom-up fashion is much faster. Moreover, we can modify this method so as to construct an atomic cover, using V to guide us in the search. Notice that we invariantly assign a cover to all subformulas satisfied by V and that V is consistent with each such cover:

- For any atom a in any formula in Φ, assign a cover $\Psi = \{a\}$ if $V \models_p a$; otherwise, assign no cover.
- For a conjunction $\alpha = \alpha_1 \wedge \ldots \wedge \alpha_n$ where $V \models_p \alpha_i$ for all $i = 1, \ldots, n$, a V-consistent cover Ψ_i can be found for each α_i. Assign the cover $\bigcup_{i=1}^{n} \Psi_i$ to α. (By Lemma 2.2.5, this cover is V-consistent.) If $V \not\models_p \alpha_i$ for all $i = 1, \ldots, n$, assign no cover.
- For any disjunction $\beta = \beta_1 \vee \ldots \vee \beta_m$ where $V \models_p \beta_j$ for some $j = 1, \ldots, m$, a V-consistent cover Ψ_j can be found. Assign this cover to β. To all other disjunctions assign no cover.

- V must satisfy every $\varphi \in \Phi$, so each φ must have a V-consistent cover Ψ_φ. By Lemma 2.2.5, $\Psi = \bigcup\{\Psi_\varphi : \varphi \in \Phi\}$ is a V-consistent atomic cover as desired.

- Corollary 2.2.9 states that $|\Psi| \leq |P| + 1$ for any consistent atomic cover. Therefore, each union operation can be performed in $O(|P|)$ steps. Since each subformula is evaluated only once, constructing a V-consistent atomic cover for Φ takes $O(|P| \times |\Phi|)$ steps, where $|\Phi|$ is the number of atoms in Φ, counting multiplicities.

Chapter 3

The Modal Logic $\mathbf{K_{NNF}}$

> *At the same time, God is supposed to be noncontingent, immaterial, infinite, nontemporal and unconditional. All we know, perceive, or experience of the things in this world is contingent, material, finite, temporal, conditioned. But God is everything ..., yet he is incomparable. ... At the same time he's supposed to be superior to the cosmos. Superior implies comparison, doesn't it? I'm too dumb to be an ontologist. ... But I do believe in the existence of God or some power I can't know.*
>
> *William Wharton, "Tidings"*

The modal logic **K** has been defined in a variety of ways. While there is agreement on the language $\mathcal{L}(\mathbf{K})$, the semantics are introduced depending on one's particular viewpoint. Someone interested in axiomatic theories would specify axioms and rules for deriving all the tautologies of **K** from these axioms. A classical treatment of a variety of axiomatizations can be found [15]. We however, following our interest in satisfying models, use models to define the semantics of **K**. Standard works here are [60] and [56]. We deviate from these standards in one important respect: We consider only formulas in negation normal form; the logic thus restricted shall be denoted $\mathbf{K_{NNF}}$. Thanks to the duality of the modal operators, this does not compromise expressivity: every **K**-formula can be converted (in linear time) to an equivalent formula in $\mathbf{K_{NNF}}$.

3.1 Kripke Models

Before we define $\mathbf{K_{NNF}}$ formally, let us look at the semantic structures themselves. As in the previous section, we assume a set P of propositional variables given. Our definition follows [60]:

Definition 3.1.1 *A* Kripke frame *is a tuple (W, R), where W is a finite, nonempty set of* worlds *(or* nodes*), and R is a binary relation on W, called* accessibility relation*. A* valuation *on W is a mapping $V : P \mapsto 2^W$. We call the tuple (W, R, V) a* Kripke model *or* Kripke structure.

As the definition suggests, we make no distinction between structures and models: there are no inconsistent Kripke structures. Since a Kripke frame can be viewed as a graph, we may use standard terminology from graph theory. In particular, a *path* from a node w_0 to a node w is a sequence of directed R-edges connecting w_0 with w. (w_0 is connected to itself via the empty path.) We say that a Kripke frame (W, R) is *rooted in* w_0, if every world $w \in W$ is connected to w_0 via a directed path. We will informally talk of worlds w at *depth* k in a model, by which we mean that k is the minimal length of any path from w_0 to w. A Kripke frame is a *tree*, if it is rooted in some world w_0, and every world has exactly one R-predecessor, except w_0 which has no predecessor. (Equivalently, we can say that every world $w \in W$ is connected to w_0 via a *unique* path.) A Kripke model is a *tree model*, if its Kripke frame is a tree.

We define in the usual fashion the kth power of R, $k \in \mathbb{N}_0$, by $R^0 = \mathrm{id}$, $R^1 = R$, and $R^{k+1} = R \circ R^k$; furthermore, we define the transitive closure $R^+ = \bigcup_{k=1}^{\infty} R^k$ and the reflexive-transitive closure $R^* = \bigcup_{k=0}^{\infty} R^k$. For any set W, any relation R on W, and $w \in W$, we denote the set of R-successors of w as $R(w) = \{w' : w\ R\ w'\}$.

For complexity analysis, we will define two measures for the size of a Kripke model (W, R, V). We set $\#K = |W|$, the number of worlds in K. By $|K|$ we denote the total size needed to store K, assuming that each world in W and each variable in P is of elementary size 1. An alternate and commonly seen characterization of the valuation V is a mapping $P \times W \mapsto \{true, false\}$. Similarly as in $\mathbf{PL_{NNF}}$, these two characterizations are equivalent, and ours can be understood as "in every world contained in $V(p)$, p is assigned *true*, and *false* everywhere else". Another useful view of the valuation function is how it defines a propositional valuation in each world $w \in W$:

Definition 3.1.2 *Given a Kripke structure (W, R, V) and a world $w \in W$, we define the propositional valuation $V_w = \{p \in P : w \in V(p)\}$.*

3.2 Formal Definition of $\mathbf{K_{NNF}}$

Definition 3.2.1 *The modal logic $\mathbf{K_{NNF}}$ is exactly the smallest propositional logic containing as atoms the propositional atoms of Definition 2.2.2, as well as any formulas of the form $\pi = \Diamond\ \pi_0$ (commonly called π-*formulas *or $\Diamond$-*formulas*) and $\nu = \Box\ \nu_0$ (ν-*formulas *or $\Box$-*formulas*)*[1], *where π_0 and ν_0 are $\mathbf{K_{NNF}}$-formulas.*
A model of $\mathbf{K_{NNF}}$ consists of a Kripke model $K = (W, R, V)$ and one of its worlds w. The semantic relation $\models_k$ is an extension of $\models_p$ (see Definition 2.2.2), defined on atoms as follows (a is a propositional atom and π_0, ν_0 are $\mathbf{K_{NNF}}$-formulas):

$$
\begin{aligned}
K, w \models_k a \quad & \textit{iff} \quad V_w \models_p a. \\
K, w \models_k \Box\ \nu_0 \quad & \textit{iff} \quad \textit{for all } w' \in R(w),\ K, w' \models_k \nu_0. \\
K, w \models_k \Diamond\ \pi_0 \quad & \textit{iff} \quad \textit{there exists } w' \in R(w) \textit{ such that } K, w' \models_k \pi_0.
\end{aligned}
$$

Finally, we define a depth function $d(\cdot)$ *on $\mathbf{K_{NNF}}$ as: $d(a) = 0$ for any propositional atom, $d(\nu) = d(\nu_0) + 1$ for $\nu = \Box\ \nu_0$, and $d(\pi) = d(\pi_0) + 1$ for $\pi = \Diamond\ \pi_0$, extended recursively to propositional concatenations as in Definition 2.1.10.*

In conjunction with Definition 3.1.2 we get the "classical" characterization for $\models_k$ on literals:

Corollary 3.2.2 *In the notation of Definition 3.2.1, we have for any $p \in P$:*

$$
\begin{aligned}
K, w \models_k p \quad & \textit{iff} \quad w \in V(p). \\
K, w \models_k \neg p \quad & \textit{iff} \quad w \notin V(p).
\end{aligned}
$$

We write $\varphi \equiv_k \psi$ iff φ and ψ are satisfied by the same models. The following substitution properties hold:

Proposition 3.2.3 *If $\varphi \equiv_k \psi$, then $\Diamond\ \varphi \equiv_k \Diamond\ \psi$ and $\Box\ \varphi \equiv_k \Box\ \psi$.*

---

[1] *ν- and π-formulas are collectively referred to as* modal atoms *by many authors, for the same reason we consider them as atoms, namely that they are atomic with regard to propositional reasoning.*

Proof: These are a direct consequence of Definition 3.2.1. □

Two Kripke structures $K_1 = (W_1, R_1, V_1)$ and $K_2 = (W_2, R_2, V_2)$ are *isomorphic* (written $K_1 \cong K_2$), iff there exists a bijection $\mu : W_1 \mapsto W_2$ so that R_2 and V_2 are obtained from R_1 and V_1 by replacing each world w with $\mu(w)$.

Corollary 3.2.4 *If K_1 and K_2 are isomorphic under the bijection μ and $w_2 = \mu(w_1)$, then $K, w_1 \models_k \varphi$ iff $K_2, w_2 \models_k \varphi$ for all $\mathbf{K_{NNF}}$-formulas φ.*

Proof: Informally stated, none of the conditions in the definition of $\models_k$ changes upon renaming the worlds in K. A formal proof, using induction on the structure of φ, is left to the reader. □

Definition 3.2.5 *Let $K = (W, R, V)$ and $K' = (W', R', V')$ be a Kripke structure, so that $W' \subseteq W$, $R' \subseteq R$, and $V'(p) = V(p) \cap W'$ for every $p \in P$. Then we say that K' is a* submodel *(or substructure) of K. In particular, the submodel where $R' = R \cap (W' \times W')$, is called the submodel* induced by *W', and written as $K|_{W'}$. Furthermore, we identify the substructure induced by a world $w \in W$ (and all its successors) as $K_w = K|_{R^*(w)}$.*

A well-known result is the following:

Proposition 3.2.6 *Let $K = (W, R, V)$ be a Kripke structure and $w_0 \in W$. Then $K, w_0 \models_k \varphi$ if and only if $K_{w_0}, w_0 \models_k \varphi$.*

In other words, any Kripke model for φ has a rooted submodel which also satisfies φ. According to this result, if a set Φ is satisfiable, not only can we always find a satisfying rooted Kripke model, but even one whose root world w_0 satisfies Φ. This greatly alleviates our search for satisfying Kripke frames.

3.3 Characterizations of Satisfiability

Having considered some of the particular notions of Kripke semantics, we now recall that we introduced $\mathbf{K_{NNF}}$ as a propositional logic. This allows us to apply Theorem 2.1.38 to get this important result:

Theorem 3.3.1 *Let a Kripke structure $K = (W, R, V)$, a world $w_0 \in W$, and a set of $\mathbf{K_{NNF}}$-formulas Φ be given. Then $K, w_0 \models_k \Phi$ iff Φ has an atomic cover $\Psi = \Psi_A \uplus \Psi_N \uplus \Psi_P$,*

whose elements are categorized into the sets Ψ_A, Ψ_N, and Ψ_P of propositional atoms, ν-formulas, and π-formulas of Ψ, respectively, so that

(1) *V_{w_0} is consistent with Ψ_A,*

(2) *for every $\pi \in \Psi_P$ there exists a world $w \in R(w_0)$ so that $K_w, w \models_k \pi_0$,*

(3) *and for every $\nu \in \Psi_N$ and every world $w \in R(w_0)$, $K_w, w \models_k \nu_0$.*

Proof: We first apply Proposition 2.1.38, stating that K, w_0 is a model for Φ iff it is a model for some atomic cover Ψ. The atoms in Ψ here fall into the three distinct categories *propositional atoms, ν-formulas, and π-formulas*; it is equivalent with $K, w_0 \models_k \Psi$ to say that

- $K, w_0 \models_k a$ for every $a \in \Psi_A$,
- *and* $K, w_0 \models_k \pi$ for every $\pi \in \Psi_P$,
- *and* $K, w_0 \models_k \nu$ for every $\nu \in \Psi_N$.

We show the equivalence of each of these three statements with the corresponding statement (1) through (3) above.

For the first statement, consider any propositional atom a. According to Definition 3.2.1, $K, w_0 \models_k a$ iff $V_{w_0} \models_p a$; therefore, $K, w_0 \models_k a$ for every $a \in \Psi_A$ is equivalent to saying that $V_{w_0} \models_p \Psi_A$, which in turn is equivalent to Statement (1) by Corollary 2.2.4.

For the second statement, Definition 3.2.1 states that $K, w_0 \models_k \pi$ iff there exists a world $w \in R(w_0)$ so that $K, w \models_k \{\pi_0\}$. Using Proposition 3.2.6, we can equivalently write $K_w, w \models_k \{\pi_0\}$, as stated in (2). The third statement is analogously shown equivalent with (3). □

We can use this theorem to give a recursive definition for the satisfiability of $\mathbf{K_{NNF}}$-formulas which does not depend on a particular Kripke model:

Corollary 3.3.2 *A set Φ of $\mathbf{K_{NNF}}$-formulas is satisfiable iff Φ has an atomic cover $\Psi = \Psi_A \uplus \Psi_N \uplus \Psi_P$, whose elements are categorized into the sets Ψ_A, Ψ_N, and Ψ_P of propositional atoms, ν-formulas, and π-formulas of Ψ, respectively, so that Ψ_A is clash-free and for every $\pi \in \Psi_P$, the set $\{\pi_0\} \cup \{\nu_0 : \nu \in \Psi_N\}$ is satisfiable.*

Proof: The set Φ is satisfiable iff there exists a Kripke model K so that $K, w_0 \models_k \Phi$, which holds according to Theorem 2.1.38 iff Φ has an atomic cover of the form above, satisfying statements (1) through (3). From (1) and Corollary 2.2.8 we obtain the assertion that Ψ_A is clash-free, and from (2) and (3) we infer that for every $\pi \in \Psi_P$, K has a world $w \in R(w_0)$ so that K_w, w satisfies all elements of $\{\pi_0\} \cup \{\nu_0 : \nu \in \Psi_N\}$, which witnesses that this set is satisfiable. Conversely, suppose that the conditions of this Corollary hold. Then for each $\pi \in \Psi_P$, the set $\{\pi_0\} \cup \{\nu_0 : \nu \in \Psi_N\}$, presumed satisfiable, must have a Kripke model $K(\pi)$ with root $w(\pi)$. Now introduce a new world w_0 distinct from the worlds in all the $K(\pi)$, and define a valuation V_{w_0} consistent with Ψ_A (which can be done, since Ψ_A is clash-free). Furthermore, define $(w_0, w(\pi)) \in R$ for all roots $w(\pi)$, and let W, R and V be the union of all worlds, accessibility relation pairs, and valuations created. Then it is easy to show that $K = (W, R, V)$ satisfies the conditions of Theorem 2.1.38, which witnesses the satisfiability of Φ and completes this direction of the proof. □

3.4 Strict Kripke Models

The constructive second half of the proof of Corollary 3.3.2 gives rise to a certain type of model for $\mathbf{K_{NNF}}$-formulas, namely one in which all submodels are distinct and pairwise disjoint. We define it formally like this (notice the similarity with Theorem 3.3.1):

Definition 3.4.1 *A Kripke model $K = (W, R, V)$ with root w_0 is a* strict model *for a $\mathbf{K_{NNF}}$-formula φ (or set Φ) (written $K, w_0 \models_c \varphi$, $K, w_0 \models_c \Phi$), if (W, R) is a tree and φ (or Φ) has an atomic cover $\Psi = \Psi_A \uplus \Psi_N \uplus \Psi_P$, whose elements are categorized into the sets Ψ_A, Ψ_N, and Ψ_P of propositional atoms, ν-formulas, and π-formulas of Ψ, respectively, so that*

1. *V_{w_0} is consistent with Ψ_A,*
2. *for each $\pi \in \Psi_P$ there exists a* distinct *$w \in R(w_0)$ so that $K_w, w \models_c \pi_0$,*
3. *and for every $\nu \in \Psi_N$ and every world $w \in R(w_0)$, $K_w, w \models_c \nu_0$.*

Let us reemphasize the two distinguishing features of strict models: first, they are tree models; secondly, each π-subformula corresponds to a distinct R-successor of the respective world where this formula holds. Thanks to Theorem 3.3.1, every strict model for a set Φ is

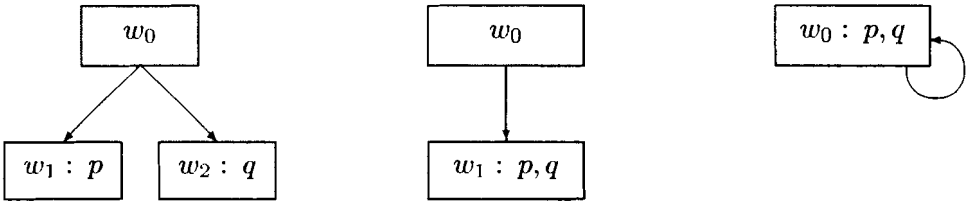

Figure 3.1: *A strict model (left), non-strict tree model (middle), and cyclic model (right) for* $\Phi = \{\Diamond\, p, \Diamond\, q\}$.

indeed a model for Φ. Conversely, we can state what is somewhat stronger than the usual *tree model property* commonly stated for **K**:

Theorem 3.4.2 *Every satisfiable* $\mathbf{K_{NNF}}$*-formula (or set of formulas) has a strict model.*

Proof: Let Φ be a satisfiable set of formulas. This means that the conditions of Corollary 3.3.2 hold. We prove our statement by induction on the depth $d(\Phi)$, using a slight alteration of the second half of Corollary 3.3.2. If Φ does not contain any modal subformulas, then it has a model with a Kripke frame of the form $(\{w_0\}, \emptyset)$. Such a model is trivially strict. Now assume the statement shown for formulas of depth at most k, and let $d(\Phi) = k + 1$. Corollary 3.3.2 says that for each $\pi \in \Psi_P$ the set $\{\pi_0\} \cup \{\nu_0 : \nu \in \Psi_N\}$ is satisfiable. By the induction hypothesis, this set (which is of depth at most d) must have a *strict* model. Now ensure (by renaming or similar means) that the sets of worlds in all the submodels arising from the different $\pi \in \Psi_P$ are pairwise disjoint, and construct K as before. (Remember that w_0 was also chosen distinct from any existing worlds.) Then K is easily shown to be a tree model, and it satisfies the modified criteria in Definition 3.4.1. Thus we have found a strict model for Φ, which finishes the induction step. □

As Theorem 3.4.2 shows, strict models are a "safe fall-back" in model finding: they always exist, as long as the set Φ is satisfiable. From the fact that each π-formula (in Ψ_P) needs to correspond to only one distinct successor world $w \in R(w_0)$, and any successor worlds beyond these are redundant, we implicitly get an upper bound for the smallest Kripke model for Φ. But there may exist smaller non-strict models:

Example 3.4.3 The set $\Phi = \{\Diamond\, p, \Diamond\, q\}$ has a strict model with root world w_0 and two successor worlds w_1, w_2, where p and q are assigned *true* in worlds w_1 and w_2, respectively (see Figure 3.1). However, there is no need to keep the successors w_1 and w_2 separate; a

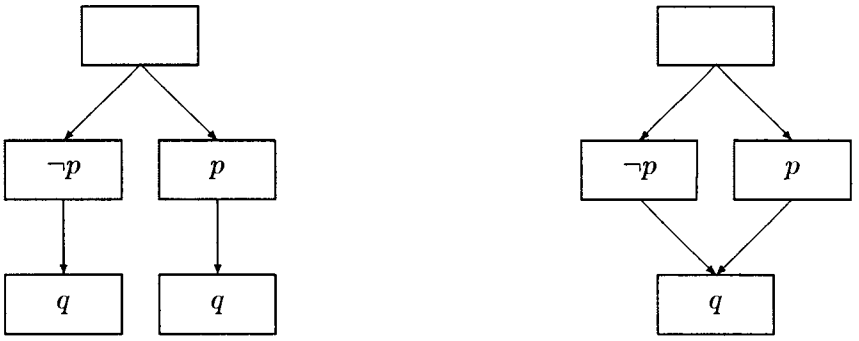

Figure 3.2: *A strict model (left) and a DAG model (right) for* $\Phi = \{\Diamond\, p, \Diamond\, \neg p, \Box \Diamond\, q\}$.

model with one successor world in which both p and q are assigned *true* also satisfies φ; note that it is still a tree model for φ. We can reduce the model even further to a one-world frame where w_0 is accessible to itself, with p and q assigned *true* in w_0. This is no longer a tree model.

Example 3.4.4 The set $\Phi = \{\Diamond\, p, \Diamond\, \neg p, \Box \Diamond\, q\}$ has a strict model as shown in Figure 3.2. We see that the two worlds at depth 2 have identical variable assignments, so they can be replaced by one world. This model is no longer a tree model, although it does not have any cycles. We will call it a DAG model. See Section 4.8 for further discussion.

We see that non-strict models, if they exist, can be much smaller than strict models. For this reason an algorithm which is capable of producing non-strict models should be preferred to one which only produces strict models. Some methods for finding non-strict and non-tree models as in Figure 3.2 are discussed in Appendix A.2.

3.5 Complexity of Reasoning Tasks in $\mathbf{K_{NNF}}$

The complexity of reasoning in **K** is well understood. Our subset $\mathbf{K_{NNF}}$, as we have shown, is no less expressive than full **K**, so the same results as shown in the literature [61] are true here, and we quote them without giving any proof. In fact, they will follow independently from similar theorems for labelled formulas we will state in the next chapter. For a definition of the complexity class PSPACE, see [92] or refer to Section 4.7.

Theorem 3.5.1 *Given a set Φ of $\mathbf{K_{NNF}}$-formulas, the problem of deciding whether Φ is unsatisfiable is PSPACE-complete.*

Note that in the case of PSPACE the complexity of deciding satisfiability and unsatisfiability are the same. Therefore, we have the standard properties:

- The space requirement for deciding whether Φ has a model is polynomial.
- There are algorithms for satisfiability, requiring at most exponential running time, but no known algorithms with less-than exponential worst-case running time complexity.

The polynomial space result only holds if we do not care about an actual model to be returned. If we do, then [61] also shows:

Proposition 3.5.2 *There are families of sets Φ of size $O(n)$ for which minimal Kripke models are of size $\Theta(2^n)$.*

We will present a large class of such sets later in Proposition 4.7.9. The fact that Kripke models can be exponentially larger than the original problem size is the driving factor of this work. In between the polynomial space requirement for showing satisfiability at the disadvantage of not obtaining a model, and the exponential space requirement for full-size Kripke models, our desire is to specify models in a more efficient way, so as to reduce their size in as many problem instances as possible, without losing any of the information needed to construct the model.
It turns out that the exponential bound is also a worst-case upper bound:

Theorem 3.5.3 *$\#K \leq 2^b$, where b is the number of $\Diamond$-operators in Φ.*

Let us conclude this brief section with a result on model checking. Since we will use it later, we provide a proof as well:

Theorem 3.5.4 *Given a set of $\mathbf{K_{NNF}}$-formulas Φ and a Kripke model K, deciding whether $K, w_0 \models_k \Phi$ takes $O(|K| \times |\Phi|)$ steps.*

Proof: We describe a simple systematic procedure as follows: For $k = 0, \ldots, d(\Phi)$ in ascending order, determine and store whether $K, w \models_k \varphi$ for every subformula φ in Φ of depth k, and every world w in K. We claim the following property (which immediately

implies the theorem): The procedure takes $O(|K| \times |\varphi|)$ time per formula φ. (Notice that we did not claim that the procedure takes $O(|\varphi|)$ time to decide $K, w \models_k \varphi$ for *one* world w, which is not true.)

The property obviously holds if φ is a propositional atom: In case $\varphi = \top, \bot$, it is evident whether $K, w \models_k \varphi$ or not; for $\varphi = p$ ($\varphi = \neg p$), $K, w \models_k \varphi$ iff $w \in V(p)$ ($w \notin V(p)$), which can be decided in one step per world w. The property is preserved under propositional concatenation: If $\varphi = \varphi_1 \circ \varphi_2$ ($\circ = \wedge, \vee$), we evaluate $K, w \models_k \varphi_1$ and maybe $K, w \models_k \varphi_2$ (which takes $O(|\varphi_1|)$ and $O(|\varphi_2|)$ steps, respectively), and computing $K, w \models_k \varphi_1 \circ \varphi_2$ is one Boolean operation on these results, for a total of $O(|\varphi|)$ steps. Finally, we show that the validity of this property for formulas of depth at most k implies its validity for all atoms of depth $k+1$. Let $\Box\ \nu_0$ and $\Diamond\ \pi_0$ be these atoms. We have already evaluated and stored $K, w' \models_k \nu_0$, $K, w' \models_k \pi_0$ for all worlds w' in k. Therefore, evaluating $K, w \models_k \Box\ \nu_0$ and $K, w \models_k \Diamond\ \pi_0$ involves looking up these Boolean values for all successors w' of w, and taking their conjunction or disjunction, respectively. We easily see that summarized over all worlds w in K, the number of lookups for each formula is at most equal to the number of arcs in the accessibility relation R. So we add $O(|K|)$ operations to the $O(|K| \times |\nu_0|)$ (or $O(|K| \times |\pi_0|)$) operations guaranteed by the induction hypothesis, for a total of $O(|K| \times |\nu|)$ (or $O(|K| \times |\pi|)$) steps. By the principle of induction over the formula depth in Theorem 2.1.12, the property holds for every $\mathbf{K_{NNF}}$-formula, which shows the theorem. □

Chapter 4

Labelled Formulas

書不盡言，言不盡意；
然則聖人之意，其不可見乎？
子曰：聖人立象以盡意，
設卦以盡情偽，
系辭焉以盡其言，
變而通之以盡利，
鼓之舞之以盡神。

Writings do not fully encompass words, and words do not fully encompass meaning. This being so, can the ideas of the Sages not be illustrated? The Master says: The Sages created labels in order to fully express meaning. They designed diagrams so as to exhaustively categorize what is true and what is false, tied them together with explanations to fully express their words, and discussed all their variations in order to obtain the utmost advantage from them. Through proclaiming them as with drums and dances, their spirit will be completely revealed.

Konfucius, "The Great Appendix" (to the Yi Jing), I, Ch. XII

4.1 Introduction and Motivation

As we have seen, a Kripke structure consists of three types of objects: possible worlds, pairs (w_1, w_2) of an accessibility relation, and pairs (p, w) of a valuation function. We make the assumption that every world and every pair are explicitly represented by some object[1].

[1]Of course, Kripke structures in which R and V are specified in some closed form rather than by enumerating all their pairs are imaginable, but we are not aware of any approach which uses such structures for

This often entails a large amount of redundancy, as we will now highlight. The following example may not lend itself naturally to modal reasoning, but it will serve in illustrating our concepts well, and we will encounter variations throughout the next two chapters:

Example 4.1.1 Consider a set of cubes, arranged according to a tuple (x, y, z) of coordinates, $x, y, z = 1, \ldots, n$. We can picture these as part of a larger $n \times n \times n$-cube, as in Figure 4.1. We would like to reason about properties on these cubes, such as "has coordinate $x = 2$", "is located at a corner", "is hidden from view" etc. Therefore, we would like to model them as a set of individuals or possible worlds, in each of which any of the above properties can be *true* or *false*. One possible way to accomplish this is shown in Figure 4.2: Beginning with a root world, we create three additional layers of worlds, the third of which represents the set of actual cubes. In each layer we add another dimension, thus creating successively a row of n worlds, a plane of $n \times n$ worlds, and finally a cube of $n \times n \times n$ worlds. The root world and the worlds in the first two layers do not actually have a physical interpretation, but this three-layer approach helps structure the problem.

One set Φ of formulas which has this arrangement of worlds as a Kripke model is the following:

$$\begin{array}{lll} \Diamond\Box\Box\,(x=1) & \Box\Diamond\Box\,(y=1) & \Box\Box\Diamond\,(z=1) \\ \quad\ldots & \quad\ldots & \quad\ldots \\ \Diamond\Box\Box\,(x=n) & \Box\Diamond\Box\,(y=n) & \Box\Box\Diamond\,(z=n) \end{array} \tag{4.1}$$

We treat the predicates $x = 1, \ldots, z = n$ as propositions and assume that variable assignments for each of x,y,z are mutually exclusive. (This means, at most one of $x = 1, \ldots, x = n$ can be *true* in any given world; same for y and z.) This effect can be easily achieved by introducing more formulas ("mutex clauses"), but we will omit those to keep things simple. Mutual exclusivity forces each $\Diamond$-formula to be satisfied in a distinct accessible world. These formulas not only span the model shown in Figure 4.1; they also specify x-, y- and z-coordinates for each cube, as annotated in the figure.

It is convenient—and common practice in the literature, see e.g. [22]—to use *labels* for addressing the worlds in a Kripke structure. Labels are strings of constants; they are

---

reasoning, except perhaps the free-variable tableaux [9]. But in these, a model is only implicitly contained in the structure of a tableau node, and we cannot easily extract information out of it.

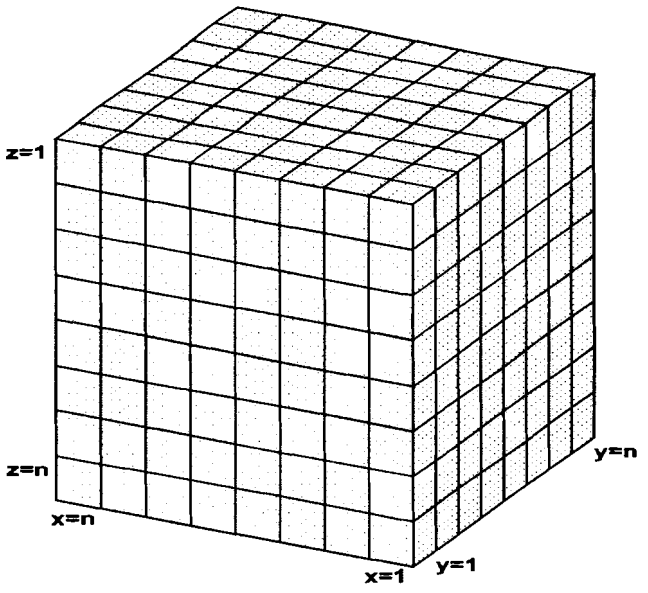

Figure 4.1: The $n \times n \times n$ cube of worlds in Example 4.1.1.

intended to capture the accessibility relation in the following way: The root world w_0 is assigned the empty string ε (or 1, as is commonly seen in the literature, perhaps to accommodate for more than one root world), and if a world is labelled γ, then the successors of the worlds are labelled $\gamma 1, \gamma 2, \ldots$. In our example, the coordinates of our cubes lend themselves naturally as constants for use in labels, and we assign labels to worlds in the other layers as well: The worlds on the first layer, being immediate successors of the root world, receive labels 1 through n, the worlds on the second layer labels 11 through nn, and our "real" cubes labels 111 through nnn.

So how do we understand the semantics of formulas in this setting? A ν-formula specifies that ν_0 is *true* in all successors of a world, and a π-formula that some successor exists in which π_0 is *true*. Thinking of worlds as *instances*, we can think of modalities as specifications over *all* or *one* successor instance. In this way, chains of modalities specify a set, or *subspace* if you will, of instances in which a given formula is *true*. For example, $\Diamond \Box \Box (x = 1)$ states that some instance (here: 1) exists, and $x = 1$ is satisfied in all worlds labelled $1c_y c_z$. Likewise, $\Box \Diamond \Box (y = 1)$ says that every instance c_x has a successor (here: $c_x 1$) whose successors $c_x 1 c_z$ satisfy $y = 1$, and so forth.

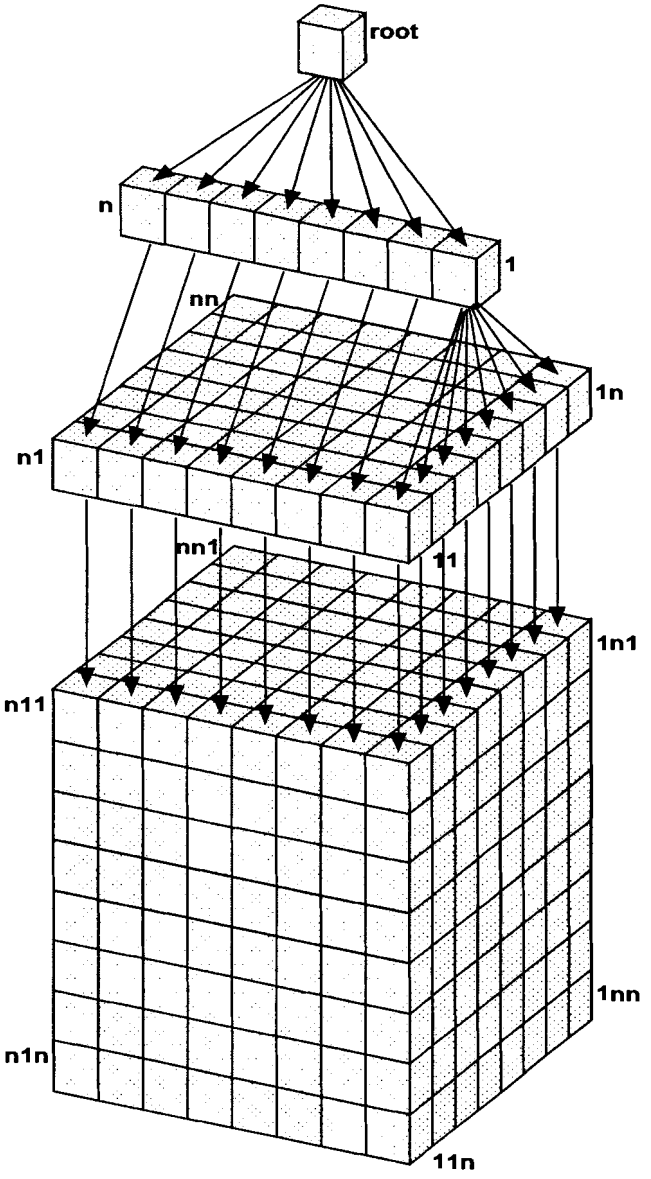

Figure 4.2: The full Kripke model and labellings for Example 4.1.1.

Now consider the variable assignments in the worlds. We observe that each cube (world on the third layer of the model) satisfies a different combination of propositions $x = c_x$, $y = c_y$, $z = c_z$. But this does not mean that there are no redundancies: For instance, all worlds labelled $1c_yc_z$ satisfy $x = 1$. Rather than specifying n^2 variable assignments, we could just specify one. To facilitate this, we will extend the notion of labels by introducing *variables* in place of constants. For the most part, it turns out that we do not actually need to *name* the variables; one anonymous variable (or *wildcard*) $*$ will do, understanding that different occurrences of $*$ can be instantiated differently. So the same variable assignments as above can be written as an *assertion* $1{*}{*}\,(x = 1)$. For the time being, let us forget about the need to specify negative assignments such as $\neg(x = 1)$ on all the other worlds; then we can state the set M of *all* variable assignments as follows:

$$\begin{array}{lll} 1{*}{*}\,(x=1) & *1{*}\,(y=1) & {*}{*}1\,(z=1) \\ \quad\ldots & \quad\ldots & \quad\ldots \\ n{*}{*}\,(x=n) & *n{*}\,(y=n) & {*}{*}n\,(z=n) \end{array} \tag{4.2}$$

These are only $3n$ assertions specifying $3n^3$ variable assignments in the n^3 cubes. If we generalize this three-dimensional problem to k dimensions, then kn assertions would suffice for specifying kn^k variable assignments in n^k worlds, a reduction from exponential to polynomial size in k.

How about the set W and relation R? As we said before, R is implicit in the structure of labels. And now we claim that even the set of labels W has become implicit in the assertions! Let us look at (4.2) carefully: We immediately see that all constants 1 through n occur in each of the three positions. And since the other two positions contain $*$, we can read an assertion such as $1{*}{*}\,(x = 1)$ as: "For whatever instances c_y, c_z of the two $*$ we can find in M, a cube labelled $1c_yc_z$ exists (and $x = 1$ is *true* in that world)." Now we can find instantiations of c_y and c_z such as $c_y = 2$, $c_z = n$ from other assertions, in this case $*2{*}\,(y = 2)$ and $**n\,(z = n)$. Together the three assertions witness that a world with label $12n$ exists. Observe that exactly this label $12n$ is the *unifier*, or common instance, of the labels $1{*}{*}$, $*2{*}$, and $**n$. So we will declare that W is the set of all *ground* labels which can be constructed as unifiers of labels in all the assertions. We call any such ground label *realized*. Some examples for labels which are not realized in this example are 1111 (because no assertion has a label with 4 positions) and $11(n+1)$ (because the constant $n+1$ does not exist). Realizability and related issues are described at length in Section 4.4. To summarize,

a set of assertions is all we need to specify a Kripke model implicitly, and as we have seen, these sets of assertions can be vastly more compact than explicit Kripke models.

We must point out that the use of assertions may give rise to inconsistencies. If for instance we added an assertion $1{*}{*}\,\neg(z = 1)$, then $z = 1$ would have to be both *true* and *false* in any instance of $1{*}1$, which is impossible. An occurrence of two contradictory assertions is called a *clash*. Only clash-free sets M will be acceptable as models, so we must be sure to avoid introducing clashes.

Finally, let us compare the original set of formulas (4.1) with our newly found set of assertions (4.2). We observe that these are almost identical, except that the former uses modal operators and the latter labels. The $\Box$-operator obviously corresponds to $*$, and each $\Diamond$-operator to each of the n constants we used in that respective position. What this shows is that we could have found a model for Φ simply by replacing its modal operators with constants and wildcards! Now in general this does not result in a set of assertions, for assertions may not contain propositional operators. But even in the general case, it makes sense to convert formulas by replacing *all* modal operators with constants and wildcards, anywhere in the nested structure of each formula. Then we only have to deal with labels, not modalities, and model finding simply becomes an algorithm to branch on the propositional structure of labelled formulas, with the goal of eliminating all propositional connectives and arriving at a set of assertions. In one final step, we must then check whether this set is clash-free.

To round up this introduction, let us briefly highlight the conceptual changes in the approach we are undertaking:

- The logic $\mathcal{LBL}$ does retain a lot of similarity with **K**. It is a propositional logic, so all results from Chapter 2 are applicable. Furthermore, the semantics of modal operators and labels are so similar that we can find a homomorphism between the two logics (see Definition 2.1.17) and show that it preserves satisfiability. See Section 4.5.

- The $\Diamond$-operator and the constants in a label have different connotations. A formula $\Diamond\ \pi_0$ says that π_0 is *true* in *some* successor, whereas $c\,F$ states that F is *true* in *one specific* successor. This is different, and we will point this out in several places.

- On the other hand, the use of labels in both formulas and semantic structures gives us a correspondence between assertions in the structure and labels in the formula: they correspond whenever they use the same constants.

- We ought to ensure that our logic of labelled formulas does not leave the reasoning class PSPACE. We will demonstrate this in Section 4.7.

- Model checking, on the other hand, does not retain its complexity class it occupied in **K**, where it was polynomial; here it becomes co-NP complete in the size of the model. One obvious reason is the additional requirement to ensure clash-freeness, which itself is a co-NP complete problem, as we will see. However, we will prove that model checking only becomes more expensive when our models are more compact than Kripke models, and that we can always resort to checking the Kripke model represented by our assertions, without significant overhead in the size of this Kripke model. All details can be found in Section 4.6.

- To retain soundness, we must replace $\Diamond$-modalities by *distinct* constants. We will see that this forces us to find a *strict* Kripke model. Usually the gain in compactness through our representation more than outweighs the loss in compactness through reduplication of worlds in a tree model versus a non-tree model. But we will expend Section 4.8 studying examples where this is not true, and suggesting some remedies.

4.2 Simple Labels

Let D be a fixed, countably infinite domain of *constants*, and $*$ a symbol not contained in D, the so-called *wildcard.*

Definition 4.2.1 *A* simple label *of* length $n \geq 0$ *is an n-tuple of symbols from $D \cup \{*\}$, written $\sigma = c_1 \cdots c_n$. For any $k = 1, \ldots, n$, the element c_k is called the kth* position *of σ, and the label $c_1 \cdots c_k$ is the k-*prefix *(or simply: a prefix) of σ. The* concatenation *of two labels $\sigma = c_1 \cdots c_n$ and $\sigma' = c'_1 \cdots c'_m$ is defined and denoted in the usual way: $\sigma\sigma' = c_1 \cdots c_n c'_1 \cdots c'_m$. The only label of length 0 is the* empty label, *denoted ε. We identify labels of length 1 with their one element, e.g. $\sigma = c$. Furthermore, we define the* undefined label $\emptyset$*; it has no length assigned to it. A label is called* ground *if it does not have $*$ in any of its positions. The* prefix closure $\mathrm{pref}(\Sigma)$ *of a set Σ of labels is simply the set of all prefixes of labels in Σ, and Σ is* closed under prefixing, *if $\Sigma = \mathrm{pref}(\Sigma)$.*

We will establish the convention of using σ, σ', etc. for arbitrary simple labels, γ, γ', etc. for ground labels, x, x_i for elements which may be constants or $*$, and c, c_i for constants

from D only. If we deviate from this, we will mention it.

Definition 4.2.2 *Two labels $\sigma = x_1 \cdots x_n$ and $\sigma' = x'_1 \cdots x'_m$* unify *if $m = n$ and for every $k = 1, \ldots, n$, if $x_k \in D$ and $x'_k \in D$, then $x_k = x'_k$. (That is, wherever σ and σ' both have constants in the same position k, these two constants are the same.) If σ and σ' unify, their* most general unifier *(mgu) (σ, σ') is the label $x''_1 \cdots x''_n$, where $x''_k = x'_k$ whenever $x_k = *$, and $x''_k = x_k$ for all other k. (If σ and σ' do not unify, we define $(\sigma, \sigma') = \emptyset$.) If $(\sigma, \sigma') = \sigma'$, then we say that σ' is an* instance *of σ, written $\sigma \sqsubseteq \sigma'$. If additionally $\sigma \neq \sigma'$, we also write $\sigma \sqsubset \sigma'$, saying that σ' is a* strict *instance of σ.*

It is easy to see that taking the mgu is a commutative and associative operation (as we would expect), so it is consistent to define the mgu of n labels $\sigma_1, \ldots, \sigma_n$ as $(\sigma_1, \ldots, \sigma_n) = ((\ldots(\sigma_1, \sigma_2), \ldots), \sigma_n)$. (Occasionally, we also write $\mathrm{mgu}(\{\sigma_1, \ldots, \sigma_n\})$ to avoid confusion.)
The usual lattice-theoretic properties regarding unifiers and the instance relation may be readily verified by the reader, justifying the choice of terms just defined:

Proposition 4.2.3 *The instance relation $\sqsubseteq$ is a partial order on the set of all labels, that is, it is reflexive, antisymmetric, and transitive. Two labels σ and σ' unify exactly if they have a common instance; every such instance is also an instance of (σ, σ').*

To motivate the above definitions, we associate a label with a set of ground instances from a given universe Γ. Such a universe cannot be chosen arbitrarily: Corresponding to the worlds of a Kripke tree, we must expect any predecessor (read: prefix) of a ground instance in Γ, as well as the root, which is ε, to belong to Γ.

Definition 4.2.4 *Let $\Gamma \subset D^*$ be a set of ground labels which contains ε and is closed under prefixing. To any given label σ, we assign the set $\mathcal{G}(\sigma, \Gamma)$ of all labels in Γ which are ground instances of σ. We call it the* domain of σ wrt Γ. *For the undefined label $\emptyset$, we set $\mathcal{G}(\emptyset, \Gamma) = \emptyset$, the empty set.*

The reader may understand a (non-ground) label σ as a *placeholder* for any of its instantiations in $\mathcal{G}(\sigma, \Gamma)$. Any statement involving σ is intended to hold for all ground instances of σ. While we can always choose $\Gamma = D^*$ as the universe, we will see an important application in Section 4.4 with a more restricted universe.
The domain of a label has the properties we expect from it:

Proposition 4.2.5 *For any two labels σ, σ' and any universe Γ as above, we have:*

(1) $\mathcal{G}((\sigma,\sigma'),\Gamma) = \mathcal{G}(\sigma,\Gamma) \cap \mathcal{G}(\sigma',\Gamma)$.

(2) *If $\sigma \sqsubseteq \sigma'$, then $\mathcal{G}(\sigma',\Gamma) \subseteq \mathcal{G}(\sigma,\Gamma)$.*

Proof: The first part follows from the fact that any ground instance of (σ,σ') is a ground instance of both σ and σ' and vice versa. This is trivially true for labels which do not unify, in which case the set of common ground instances is empty. For the second part, suppose $\sigma \sqsubseteq \sigma'$. Since $\sqsubseteq$ is transitive, any ground instance of σ' is also a ground instance of σ, which establishes the subset relationship. □

Definition 4.2.6 *Given a set of labels Σ and a constant or wildcard x, we define $\Sigma_{\langle x\rangle} = \{\sigma : x\sigma \in \Sigma \text{ or } *\sigma \in \Sigma\}$. For an arbitrary label $\sigma = x_1 \cdots x_n$, we define $\Sigma_{\langle\sigma\rangle} = (\ldots(\Sigma_{\langle x_1\rangle})\ldots)_{\langle x_k\rangle}$. Conversely, for a constant or wildcard x, we define $x\Sigma = \{x\sigma : \sigma \in \Sigma\}$.*

The following technical lemma lists some properties of this definition with respect to domains; we will build upon this result later:

Lemma 4.2.7 *For any label σ, constant c and universe Γ as before, we have:*

(1) $\Gamma = \{\varepsilon\} \cup \bigcup_{c\in\Gamma} c\Gamma_{\langle c\rangle}$.

(2) $\mathcal{G}(*\sigma,\Gamma) = \bigcup_{c'\in\Gamma} c'\mathcal{G}(\sigma,\Gamma_{\langle c'\rangle})$.

(3) *If $c \in \Gamma$, then $\mathcal{G}(c\sigma,\Gamma) = c\mathcal{G}(\sigma,\Gamma_{\langle c\rangle})$; $\mathcal{G}(c\sigma,\Gamma) = \emptyset$ otherwise.*

Proof: For part (1), consider Definition 4.2.4. The empty label is required to be in Γ; now take any nonempty label in Γ of the form $c\sigma$. Then $\sigma \in \Gamma_{\langle c\rangle}$, and $\Gamma_{\langle c\rangle}$ contains no labels beyond these. Also Γ is closed under prefixes, so we have $c \in \Gamma$, and the union operator correctly collects all nonempty labels of Γ.

For parts (2) and (3), take $x = *$ or $x = c$. For any ground label $c'\gamma \in \Gamma$, we get $c' \in \Gamma$ and $\gamma \in \Gamma_{\langle c'\rangle}$ from part (1). Furthermore, $x\sigma \sqsubseteq c'\gamma$ iff $x \sqsubseteq c'$ and $\sigma \sqsubseteq \gamma$. The second and fourth of these conditions together are equivalent to $\gamma \in \mathcal{G}(\sigma,\Gamma_{\langle c'\rangle})$. Now if $x = *$, then $* \sqsubseteq c'$ is always true, and c' ranges over all elements of Γ, which proves part (2). If $x = c$, then $c \sqsubseteq c'$ is true iff $c = c'$. So all ground instances are of the form $c\gamma$ where $\gamma \in \mathcal{G}(\sigma,\Gamma_{\langle c\rangle})$, provided $c \in \Gamma$. Otherwise there are no ground instances. Part (3) is thus shown. □

To conclude this section, we introduce a principle of induction over labels. Just as the induction principles of previous chapters, it will be frequently used in upcoming proofs:

Lemma 4.2.8 *(Induction on simple labels) Let Q be a property on the set of simple labels over a given domain D. If $Q(\varepsilon)$ is true and $Q(\sigma)$ implies*

- *either $Q(*\sigma)$ and $Q(c\sigma)$ for every $c \in D$*
- *or $Q(\sigma*)$ and $Q(\sigma c)$ for every $c \in D$*

for every simple label σ, then Q is true for all simple labels over D.

Proof: This principle reduces easily to induction over the integer k, where k is the length of the label. The only label of length 0 is ε, and if Q holds for all labels of length k, then all labels of length $k+1$ are of the form $\sigma*$ and σc (alternatively, $*\sigma$ and $c\sigma$), $c \in D$, where σ is some label of length k. □

4.3 The Logic of Labelled Formulas

Let D be a domain as in the previous section.

Definition 4.3.1 *The logic $\mathcal{LBL}$ of* labelled formulas *is the smallest propositional logic containing as its atoms the propositional atoms, plus formulas of the form $c\ F$ and $*\ F$, where F is a labelled formula and $c \in D$.*
The depth function $d(\cdot)$ is defined as: $d(a) = 0$ for any propositional atom, $d(c\ F) = d(*\ F) = d(F) + 1$, and $d(\cdot)$ is extended to propositional concatenations in the usual way (see Definition 2.1.10).
Let $\sigma = c_1 \cdots c_k$. Then we write the formula $c_1\ (\ldots\ (c_k\ F))$ simply as $(c_1 \ldots c_k)\ F$ or $\sigma\ F$[2]*.*
An assertion *is a labelled formula of the form $\sigma\ a$, where σ is a label and a is a propositional atom. The semantic structures of $\mathcal{LBL}$ are sets of assertions.*
For a set of $\mathcal{LBL}$-formulas S, we denote by $\#S$ the cardinality (number of formulas, not counting subformulas) in S, and by $|S|$ the absolute size of S. For any constant c, we further define $S_{\langle c\rangle} = \{\sigma\ F : c\sigma\ F \in S \text{ or } *\sigma\ F \in S\}$, and for a ground label $\gamma = c_1 \cdots c_k$, we set $S_{\langle\gamma\rangle} = ((S_{\langle c_1\rangle})\ldots)_{\langle c_k\rangle}$. Furthermore, for any label σ, we write $\sigma S = \{\sigma\sigma'\ F : \sigma'\ F \in S\}$.
The set pref(S) *of prefixes in S is defined as* pref(S) = pref$(\{\sigma : \sigma\ F \in S\})$. *Of any label $\sigma \in$* pref(S) *we say that σ is a* label in S *(or that σ* occurs *or* exists *in S).*
The semantic relation $\models_l$ is defined on sets of assertions M and labelled formulas F. It extends $\models_g$ (see Definition 2.1.1) by:

[2] *Evidently, $d(\sigma\ F) = k + d(F)$.*

$$M \models_l l \quad \textit{iff} \quad \varepsilon\, l \in M, \textit{ for all literals } l.$$
$$M \models_l c\, F \quad \textit{iff} \quad c \in \mathrm{pref}(M) \textit{ and } M_{\langle c \rangle} \models_l F.$$
$$M \models_l * F \quad \textit{iff} \quad M_{\langle c \rangle} \models_l F \textit{ for all constants } c \in \mathrm{pref}(M).$$

We read $M \models_l F$ as usual: "M verifies *F. The $\models_l$ relation is extended to sets S of formulas in the usual way.*

We have not yet characterized which sets of assertions we regard as consistent; we will elaborate on this in the next section.

Assertions form the "building blocks" of semantic structures in $\mathcal{LBL}$. The intent of an assertion $\sigma\, p$ (or $\sigma\, \neg p$) is to say that p is assigned *true* (or *false*) in all the ground labels which are represented by σ, that is, $\gamma \in \mathcal{G}(\sigma, \Gamma)$. In addition to this, the set Γ itself arises from the labels σ in the assertions of M, which capture certain "existence requirements". For instance, an assertion $\gamma\, a$ with a ground label implies that $\gamma \in \Gamma$. For non-ground labels, the situation is more complicated; we will make it precise in the next section, once we have the appropriate terminology.

One might think that if M verifies a formula F, then any larger set of assertions $M' \supseteq M$ should also verify F, which would be the monotonicity property we discussed in Chapter 2. However, this is not true. For example, while the set $\{c_1\, p\}$ verifies $F = *\, p$, the set $\{c_1\, p, c_2\, \neg p\}$ does not. This is one of the main reasons why we will define a *strong* semantic relation in Chapter 5 which will prove to be monotone wrt $\subseteq$. For $\models_l$, a somewhat weaker property does hold:

Proposition 4.3.2 *For any set M of assertions and any assertion $\sigma\, a \in M$, $a \neq \bot$, we have $M \models_l \sigma\, a$, i.e. M verifies any of its elements.*

Proof: We prove this proposition by induction over the label σ. For $\sigma = \varepsilon$, it follows directly from Definition 4.3.1 (provided a is a literal; for $a = \top$, it is vacuously true). Assume this proposition holds for a label σ (and any set M containing it). First, consider any assertion of the form $c\sigma\, a$, where c is a constant. Then $c \in \mathrm{pref}(M)$ and $\sigma\, a$ is in $M_{\langle c \rangle}$; the induction hypothesis implies $M_{\langle c \rangle} \models_l \sigma\, a$. Hence $M \models_l c\sigma\, a$ is true according to Definition 4.3.1. Now consider assertions of the form form $*\sigma\, a$. Then $\sigma\, a \in M_{\langle c \rangle}$ for any constant c, particularly for all constants which occur in M. Therefore, $M \models_l *\sigma\, a$ according to Definition 4.3.1. We have thus shown the induction step for all labels of the form $*\sigma$ and $c\sigma$, $c \in D$, and the proof follows by induction on σ. □

An immediate consequence of the Proposition 4.3.2 is this idempotence property:

Corollary 4.3.3 *For any set M of assertions none of which are of the form $\sigma\,\bot$, we have $M \models\!\!\!\sim_l M$.*

We see why it is desirable to avoid assertions containing the atom $\bot$. By contrast, they will be of use in the strong models of Chapter 5.
We derive the relation $\equiv_l$ from $\models\!\!\!\sim_l$ as usual, getting the following substitution properties for labelled formulas:

Proposition 4.3.4 *If $F \equiv_l G$, then $c\,F \equiv_l c\,G$ for any $c \in D$, and $* F \equiv_l * G$.*

Proof: These are a direct consequence of Definition 4.3.1. □
A useful nonrecursive definition of $S_{\langle\gamma\rangle}$ is easily shown equivalent to that in Definition 4.3.1:

Proposition 4.3.5 *For any set S of formulas and any ground label γ, we have*

$$S_{\langle\gamma\rangle} = \{\sigma'\,F : \sigma\sigma'\,F \in S \text{ and } \sigma \sqsubseteq \gamma\}.$$

We conclude this section with another principle of structural induction.

Lemma 4.3.6 *(Structural induction on labelled formulas) Let Q be a property defined on $\mathcal{LBL}$-formulas, closed under propositional concatenation, and $Q(a)$ true for all propositional atoms a. If one of the following holds for all $\mathcal{LBL}$-formulas F:*

(1) *$Q(F)$ implies $Q(* F)$ and $Q(c\,F)$ for all $c \in D$,*

(2) *$Q(F)$ implies $Q(\sigma\,F)$ for any label σ,*

then $Q(F)$ is true for all $\mathcal{LBL}$-formulas F.

Proof: This principle reduces to that of induction over the formula depth according to Lemma 2.1.12. We were given that $Q(a)$ is true for all atoms of depth 0 and preserved under concatenation. All atoms of depth $k > 0$ are of the form $* F$, $c\,F$ for some $c \in D$, where F is of depth $k-1$, or (in the second case) they are of the form $\sigma\,F$, where σ is some label of length $k' > 0$ and F is of depth $k - k'$. If $Q(F)$ is valid for all formulas of depth at most $k-1$, then any of the implications (1), (2) can be used to show $Q(F')$ for any atom F' of depth $k+1$. By the principle of induction over the formula depth, we conclude that $Q(F)$ is valid for all $\mathcal{LBL}$-formulas. □

4.4 Realizability and Clashes

The existence of a set of assertions M with $M \models_l S$ cannot be a sufficient criterion for calling the set S satisfiable. For even a set with complementary literals such as $S = \{p, \neg p\}$ has sets of assertions which verify it. In fact, we just showed in Corollary 4.3.3 that $S \models_l S$, since S consists of assertions not including $\perp$. The answer in this case is obvious: We should disqualify S as a model since it is not clash-free (in the propositional sense). But when do assertions with nonempty labels constitute a clash? To settle this question, we need to introduce some more theory, centred around the following definition:

Definition 4.4.1 *Given a set S of labelled formulas, the set $I(S)$ of* ground labels realized in S *(or* rgls in S *for short) is the smallest set obeying the following recursive definitions:*

1. $\varepsilon \in I(S)$.
2. *If $\gamma \in I(S)$ and c is a constant so that $\sigma c \in \mathrm{pref}(S)$ for some $\sigma \sqsubseteq \gamma$, then $\gamma c \in I(S)$.*

For a given label σ, the set $I(\sigma, S)$ of realized ground instances (rgis) of σ in S *is the set $\mathcal{G}(\sigma, I(S))$ according to Definition 4.2.4. A label σ is called* realizable in S *if $I(\sigma, S)$ is nonempty.*

The following properties follow immediately from this definition, ensuring that $I(S)$ is indeed a universe with the properties as required in Definition 4.2.4:

Proposition 4.4.2 *Any prefix of a realized label is realized, and any prefix of a realizable label is realizable. The empty label ε is realizable in any set. A ground label is realizable iff it is realized. If σ' is realizable in S and $\sigma \sqsubseteq \sigma'$, then σ is realizable in S.*

The next proposition is a non-recursive version of the definition:

Proposition 4.4.3 *A ground label $\gamma = c_1 \cdots c_k$ is realized in S iff for every $i = 1, \ldots, k$ there exists a label $\sigma_{i-1} c_i$ in S so that $\sigma_{i-1} \sqsubseteq c_1 \ldots c_{i-1}$.*

Example 4.4.4 For any ground label $\gamma = c_1 \cdots c_k$ and any atom a, the set $S = \{\gamma\, a\}$ has γ and all its prefixes $c_1 \ldots c_i$, $0 \leq i \leq k$, as its realized labels. For the set $S = \{F = c{*}c{*}\ldots p, G = {*}c{*}c\ldots q\}$, assuming these two labels are both of length k, exactly the labels of the form $ccc\ldots c$ (up to length k) are realized. This can be shown using Proposition 4.4.3,

where the labels of F and G alternately justify the odd- and even-numbered positions of $ccc\ldots c$. Consequently, the labels of both F and G are realizable. We make the observation that testing realizability may involve the same assertion multiple times.

Proposition 4.4.5 *Let $\sigma_1, \ldots, \sigma_n$ be labels in S so that their mgu $\sigma = (\sigma_1, \ldots, \sigma_n)$ exists. Then we have:*

(1) $I(\sigma, S) = I(\sigma_1, S) \cap \cdots \cap I(\sigma_n, S)$.

(2) *If σ is realizable in S, then $\sigma_1, \ldots, \sigma_n$ are all realizable in S.*

(3) *If σ is ground, then σ is a realized ground label in S.*

Proof: Part (1) is just the n-ary version of Corollary 4.2.5, part (1). In (2), if σ is realizable, then σ has an rgi γ in S; by (1), γ is also an rgi of every σ_i, $i = 1, \ldots, n$, which witnesses that these labels are realizable. For (3), let us write $\sigma = c_1 \cdots c_k$. We claim that σ is realized according to Proposition 4.4.3. To prove this, consider any position c_j in σ, $j = 1, \ldots, k$; c_j must match the jth position in some of the unifying labels σ_i; furthermore, the prefix $c_1 \cdots c_{j-1}$ instantiates the $(j-1)$-prefix of σ_i (since σ as a whole instantiates σ_i). Since this is true for every $j = 1, \ldots, k$, Proposition 4.4.3 shows that σ is realized in S. □

Example 4.4.6 Let S consist of four assertions with labels $\sigma_1 = c_1{*}{*}$, $\sigma_2 = {*}c_2{*}$, and the ground labels $c_1c_1c_1$ and $c_2c_2c_2$. In this example, σ_1 and σ_2 are both realizable in S, but their unifier $c_1c_2{*}$ is not. This shows that the converse of (2) above is not true.

Part (3) of Proposition 4.4.5 states that realized ground labels may arise as ground mgus of labels in S; while this outlines the general idea, not every realized ground label can be explained in this simple manner. For instance, the two labels in $S = \{c_1{*}c_1\, p, {*}c_2c_2\, q\}$ do not unify. Yet there exist realized ground labels in S: to be precise, $I(S) = \{\varepsilon, c_1, c_1c_2, c_1c_2c_2\}$. This can be shown e.g. using Proposition 4.4.3. Equipped with more tools, we will provide a necessary and sufficient criterion for finding realized ground labels in Proposition 5.4.8 in the next chapter.

The next lemma provides us with further characterizations of realizability, namely in terms of subscripted sets (see Definition 4.3.1). These will prove very useful in a number of upcoming correctness proofs.

Lemma 4.4.7 *Let S be a set of formulas, σ, σ' labels, and γ a ground label. In the following, c, c' are understood to be constants from D. We have:*

(1) *$c\gamma \in I(S)$ iff c exists in S and $\gamma \in I(S_c)$. In particular, $c \in I(S)$ iff c exists in S.*

(2) *If c exists in S, then $(I(S))_{\langle c\rangle} = I(S_{\langle c\rangle})$; otherwise $(I(S))_{\langle c\rangle} = \emptyset$.*

(3) *$I(S) = \{\varepsilon\} \cup \bigcup\{cI(S_{\langle c\rangle}) : c \text{ exists in } S\}$.*

(4) *$I(*\sigma, S) = \bigcup\{c'I(\sigma, S_{\langle c'\rangle}) : c' \text{ exists in } S\}$.*

(5) *$I(c\sigma, S) = cI(\sigma, S_{\langle c\rangle})$ if c exists in S; $I(c\sigma, S) = \emptyset$ otherwise.*

(6) *If c exists in S, then σ is realizable in $S_{\langle c\rangle}$ iff $c\sigma$ is realizable in S; if this is the case, then $*\sigma$ is also realizable in S.*

(7) *If σ has a realized ground instance γ in S, then σ' is realizable in $S_{\langle\gamma\rangle}$ iff $\gamma\sigma'$ is realizable in S; if this is the case, then $\sigma\sigma'$ is also realizable in S.*

Proof: To show (1), we use Proposition 4.4.3, reasoning as follows on $\gamma = c_1 \cdots c_k$:

$c_1 \cdots c_k \in (I(S))_{\langle c\rangle}$

iff $cc_1 \cdots c_k \in I(S)$

iff c exists in S, and some $\sigma_{i-1}c_i$ ex. in S, $\sigma_{i-1} \sqsubseteq cc_1 \ldots c_{i-1}$ $(i = 1, \ldots, k)$ (Prop. 4.4.3)

iff c ex. in S, and $*\sigma'_{i-1}c_i$ or $c_1\sigma'_{i-1}c_i$ ex. in S, $\sigma'_{i-1} \sqsubseteq c_1 \ldots c_{i-1}$ $(i = 1, \ldots, k)$

iff c ex. in S, and $\sigma'_{i-1}c_i$ ex. in $S_{\langle c\rangle}$, $\sigma'_{i-1} \sqsubseteq c_1 \ldots c_{i-1}$ $(i = 1, \ldots, k)$

iff c ex. in S, and $c_1 \cdots c_k \in I(S_{\langle c\rangle})$ (Prop. 4.4.3)

The special case $\gamma = \varepsilon$ follows because ε is always an rgl.

To get (2), note that $I(S)$ is a set of ground labels, so $(I(S))_c$ is the set of ground labels γ so that $c\gamma \in I(S)$. The proof follows from (1). In particular, if c does not exist in S, then $I(S)$ does not contain a label of the form $c\gamma$, so $(I(S))_c$ is empty.

For (3), we first use Lemma 4.2.7, part (1), and then apply (1) and (2) above:

$$I(S) = \{\varepsilon\} \cup \bigcup_{c \in I(S)} c(I(S))_{\langle c\rangle} = \{\varepsilon\} \cup \bigcup_{c \text{ ex. in } S} cI(S_{\langle c\rangle}).$$

For (4) and (5), we use the definition $I(\sigma, S) = \mathcal{G}(\sigma, I(S)$, Lemma 4.2.7, parts (2) and (3), and (1) and (2) above:

$$I(*\sigma, S) = \bigcup_{c' \in I(S)} c'\mathcal{G}(\sigma, (I(S))_{\langle c' \rangle}) = \bigcup_{c' \text{ ex. in } S} c'\mathcal{G}(\sigma, I(S_{\langle c \rangle})) = \bigcup_{c' \text{ ex. in } S} c'I(\sigma, S_{\langle c \rangle}).$$

According to Lemma 4.2.7, part (3), $I(c\sigma, S) = \mathcal{G}(c\sigma, I(S))$ is empty iff $c \notin I(S)$, that is, c does not exist in S. Otherwise we have:

$$I(c\sigma, S) = c\mathcal{G}(\sigma, (I(S))_{\langle c \rangle}) = c\mathcal{G}(\sigma, I(S_{\langle c \rangle})) = cI(\sigma, S_{\langle c \rangle}).$$

For (6), just re-write "σ is realizable in $S_{\langle c \rangle}$" as "$I(\sigma, S_{\langle c \rangle})$ is nonempty", then the precondition and LHS of (6) are sufficient for the set $I(*\sigma, S)$ in (4) to be nonempty, and necessary and sufficient for the set $I(c\sigma, S)$ in (5) to be nonempty; again, this is just the definition for $*\sigma$ and $c\sigma$, respectively, to be realizable.

Finally, (7) is an iterative version of (6). To get more practice with our definitions, we provide a detailed induction proof for the first part, the "iff" statement. For $\sigma = \gamma = \varepsilon$, the statement is trivial. For our induction hypothesis, we assume it holds for σ. First consider a label $c\sigma$. The precondition of (7) says that $c\sigma$ has an rgi, which implies that c exists in S (see (1)). In (5), we read that all elements of $I(c\sigma, S)$ are of the form $c\gamma$, where γ is an rgi of σ in $S_{\langle c \rangle}$. Hence, we can apply the induction hypothesis to $S_{\langle c \rangle}$ which says that σ' is realizable in $S_{\langle c\gamma \rangle}$ iff $\gamma\sigma'$ is realizable in $S_{\langle c \rangle}$; by (6) this is equivalent to $c\gamma\sigma'$ being realizable in S. This completes the first case. Now take a label $*\sigma$. Again, $I(*\sigma, S)$ is assumed nonempty; by (4), this means that at least one constant c exists in S so that $I(\sigma, S_{\langle c \rangle})$ is nonempty. Moreover, every rgi of $*\sigma$ is of the form $c\gamma$, where c exists in S and $\gamma \in I(\sigma, S_{\langle c \rangle})$. With any $c\gamma$ thus chosen, we proceed with proving the induction step as in the first case. The induction proof is hereby complete. For the second "if" case, we use the fact that $\gamma\sigma'$ is realizable, and that it is an instance of $\sigma\sigma'$; by Proposition 4.4.2, $\sigma\sigma'$ is also realizable in S. □

For a formula $\sigma\ F$ to have a realizable label σ in a set of assertions M intuitively means that it "materializes" somewhere in M. that is, the formula F gets evaluated on some nonempty set of ground labels, namely the realized ground instances of σ. So $M \models_l \sigma\ F$ should necessarily entail $M \models_l \gamma\ F$ for any realized ground instance γ of σ. (We will give a necessary *and* sufficient condition for $M \models_l \sigma\ F$ later in Proposition 4.4.21.) Conversely, if F is unsatisfiable and σ is realizable, then $\sigma\ F$ should also be unsatisfiable. Spinning this

thought further, if two formulas F, G are not simultaneously satisfiable and their labels σ and σ' share a realized ground instance, then $\{\sigma\ F, \sigma'\ G\}$ should be unsatisfiable. These considerations give rise to our refined notion of a clash:

Definition 4.4.8 *A* clash *in a set of labelled formulas is an occurrence of an assertion $\sigma.\bot$ where σ is realizable, or of a pair of assertions $\sigma'\ p, \sigma''\ \neg p$, where (σ', σ'') exists and is realizable. The label σ, or the unifier (σ', σ''), is called the* clash witness.
A clash-free set of assertions verifying a formula F (or set of formulas S) is called a model *of F (or S). As usual, we write $M \models_l F$ ($M \models_l F$) and call F (or S)* satisfiable[3].

We observe that this definition indeed extends our previous definition of a propositional clash, and our next result is the equivalent of Proposition 2.2.8:

Corollary 4.4.9 *For any set M of propositional atoms (i.e. assertions with empty labels), Definitions 4.4.8 and 2.2.7 coincide; furthermore, M is satisfiable iff it is clash-free.*

Proof: The first part follows from the fact that ε is always realizable, so the "where"-requirements in Definition 4.4.8 are vacuously true. Now if M contains $\bot$, this assertion is a clash; but by Definition 2.1.1, $\bot$ is not verified by any set of assertions, nor is the set M. Secondly, assume M contains two complementary literals $p, \neg p$. It follows from Definition 4.3.1 that any set M' verifying M must contain every propositional atom in M. In particular, M' must contain p and $\neg p$ and hence does not qualify as a model. Finally, in case M does not contain any clash, it cannot contain $\varepsilon\ \bot$; but then Corollary 4.3.3 applies, stating that M verifies itself, and since M is clash-free, it is a model (of itself). □

Proposition 4.4.10 *If S is clash-free, then so is $S_{\langle c \rangle}$ for any constant c occurring in S. More generally, if S is clash-free and γ is a realized ground label in S, then $S_{\langle \gamma \rangle}$ is clash-free.*

Proof: By Lemma 4.4.7, part (1), c is realized iff it occurs in S, so the first part is a special case of the second. We prove this second part by the contrapositive. Assume $S_{\langle \gamma \rangle}$ contains two complementary assertions $\sigma'\ p$ and $\sigma''\ \neg p$ with clash witness $\sigma = (\sigma', \sigma'')$, and σ is realizable. Then S must contain two assertions $\sigma'_0\sigma'\ p$ and $\sigma''_0\sigma''\ \neg p$, and both σ'_0 and σ''_0 have γ as a common instance. This shows that these two labels have an mgu (call it σ_0), and σ_0 too is instantiated by γ; furthermore $(\sigma'_0\sigma', \sigma''_0\sigma'') = \sigma_0\sigma$. Now we apply part

[3] *See our remarks on terminology at the beginning of Chapter 2.*

(7) of Lemma 4.4.7: Since γ is an rgi of σ_0 and the clash witness σ is realizable in $S_{\langle\gamma\rangle}$, $\sigma_0\sigma$ is realizable and hence witnesses a clash in S. For a clash of the form $\sigma\ \bot$, the proof is analogous. □

Corollary 4.4.11 *If $M \models_l S$ and c exists in S, then $M_{\langle c\rangle} \models_l S_{\langle c\rangle}$. If M is a model for S and c exists in S, then $M_{\langle c\rangle}$ is a model for $S_{\langle c\rangle}$.*

Proof: Assume as proposed that $M \models_l S$. The set $S_{\langle c\rangle}$ consists of all formulas F, F' so that $*\,F \in S$ or $c\,F' \in S$, whereas S has at least one formula of the latter form. Since $M \models_l c\,F'$, c also exists in M and $M_{\langle c\rangle} \models_l F'$. But if c exists in M, the fact that $M \models_l *F$ implies $M_{\langle c\rangle} \models_l F$ for all F of the former type. Hence $M_{\langle c\rangle} \models_l S_{\langle c\rangle}$. The second statement follows from the first, whereas Proposition 4.4.10 shows that $M_{\langle c\rangle}$ is clash-free and hence a model. □

To prepare for the main theorem of this section, we need a small technical lemma, motivated as follows: Normally we can safely say that $M \models_l F$ implies $cM \models_l c\,F$, since c exists in cM and $(cM)c = M$. If however $F = \top$ and M is the empty set, then cM is likewise empty and c does not exist in cM, so $cM = \emptyset$ does *not* verify $c\,\top$. To avoid this pathological case, we wish to guarantee that a *nonempty* model M can always be found for any satisfiable formula:

Lemma 4.4.12 *A set of assertions M verifies a set S of formulas iff $M \cup \{\varepsilon\ \top\}$ does.*

Proof: The extra element $\varepsilon\ \top$ is "neutral" in that it cannot form part of a clash, nor does it verify any literal or other formula, nor does it introduce a new realized ground label into M. This result can be formally established through a simple induction proof over the structure of F, which we leave to the reader. □

Corollary 4.4.13 *Every satisfiable set of formulas has a* nonempty *model.*

Proposition 4.4.14 *Let S be a set of propositional atoms and M a set of assertions. Then $M \models_l S$ iff $S \subseteq M \cup \{\varepsilon\ \top\}$.*

Proof: The "if" part follows from Proposition 4.3.2: If M is a superset of S (except perhaps for $\varepsilon\ \top$), it verifies all elements of S, and $\varepsilon\ \top$ is verified by any set of assertions. For the converse, notice that $\varepsilon\ \bot \notin S$, otherwise S would not be verified by *any* set M. Any positive or negative literal in S must be included in M according to Definition 4.3.1. Finally, we

included $\varepsilon\ \top$ on the right-hand side $M \cup \{\varepsilon\ \top\}$, which shows the subset relationship and completes the proof. □

We are now ready to state a result similar to that of Theorem 3.3.1 for modal formulas:

Theorem 4.4.15 *Given a set S of $\mathcal{LBL}$-formulas and a set of assertions M not containing $\varepsilon\ \bot$, we have $M \models_l S$ iff S has an atomic cover $T = T_A \uplus T_N$ (where T_A and T_N are the atoms in T with empty and nonempty labels, respectively), so that*

(1) $T_A \subseteq M \cup \{\varepsilon\ \top\}$,

(2) *every c occurring in T_N also occurs in M,*

(3) *and for every c' occurring in M, we have $M_{\langle c'\rangle} \models_l (T_N)_{\langle c'\rangle}$.*

Proof: As before, we apply Proposition 2.1.38; stated in $\mathcal{LBL}$, it says that a set of assertions verifies S iff it verifies some atomic cover T of S. We split this cover into the two sets T_A and T_N, whereas T_A contains all atoms with empty labels, i.e. the propositional atoms. In Proposition 4.4.14, we showed that $M \models_l T_A$ iff $T_A \subseteq M \cup \{\varepsilon\ \top\}$, which is (1). We now have to show that $M \models_l T_N$ is equivalent to statements (2) and (3):

Suppose that $M \models_l \sigma\ F$ for every $\sigma\ F \in T_N$. Consider any c occurring in some atom $c\sigma'\ F$ in T_N. Since $M \models_l c\sigma'\ F$, Definition 4.3.1 stipulates that c occur in M, which shows statement (2). Now take *any* c' which occurs in M, regardless whether or not it also occurs in T_N. Note that $(T_N)_{\langle c'\rangle}$ consists of all formulas $\sigma'\ F$ such that $x\sigma'\ F \in T_N$, where x can be either c' or $*$. Since $M \models_l x\sigma'\ F$, Definition 4.3.1 (for either choice of x) entails that $M_{\langle c'\rangle} \models_l \sigma'\ F$. (In case $x = *$, we recall that c' was supposed to exist in M.) We have shown that every formula $\sigma'\ F$ in $(T_N)_{\langle c'\rangle}$ is verified by $M_{\langle c'\rangle}$, proving statement (3).

For the converse direction, assume that statements (2) and (3) hold. For formulas in T_N of the form $c\sigma\ F$, statement (2) guarantees that c exists in M, and we get as a special case of (3) that $M_{\langle c\rangle} \models_l (T_N)_{\langle c\rangle}$, and specifically $M_{\langle c\rangle} \models_l \sigma\ F$, since $\sigma\ F \in (T_N)_{\langle c\rangle}$. By Definition 4.3.1, this shows $M \models_l c\sigma\ F$. For formulas in T_N of the form $*\sigma\ F$, we have $\sigma\ F \in (T_N)_{\langle c'\rangle}$ for *any* c'. Since $M_{\langle c'\rangle} \models_l (T_N)_{\langle c'\rangle}$ for any such c' which occurs in M, we get in particular $M_{\langle c'\rangle} \models_l \sigma\ F$. By Definition 4.3.1 we obtain $M \models_l *\sigma\ F$, which completes the proof that M verifies all elements in T_N. □

Just as Theorem 3.3.1 had a Corollary 3.3.2 characterizing satisfiability in $\mathbf{K_{NNF}}$, Theorem 4.4.15 has such a corollary:

Corollary 4.4.16 *A set S of $\mathcal{LBL}$-formulas is satisfiable iff it has an atomic cover $T = T_A \uplus T_N$ (where T_A and T_N are defined as above) so that T_A is clash-free and for every c occurring in T_N, $(T_N)_{\langle c \rangle}$ is satisfiable.*

Proof: Just like Corollary 3.3.2, in our proof of the "if" part we provide a recipe for constructing a clash-free set of assertions M obeying the conditions of Theorem 4.4.15 for the given atomic cover, which witnesses $M \models_l S$. So consider any constant c occurring in T_N. Since the set $(T_N)_{\langle c \rangle}$ is presumed satisfiable, it has a *nonempty* model $M(c)$ in accordance with Corollary 4.4.13. Now construct $cM(c)$ (by prepending c to the labels of all assertions in $M(c)$), and take $M = T_A \cup \bigcup_c cM(c)$, where the union is taken over all c occurring in T_N.

Let us first prove that M is clash-free: Note that T_A and all $M(c)$ were presumed clash-free within themselves; evidently, so are the $cM(c)$. Any two assertions from two different sets $cM(c)$, $c'M(c')$ begin with different constants, and the assertions of T_A have the empty label. Thus, no two labels from different sets unify, so a clash cannot possibly occur in M. In particular, $\varepsilon \perp$ cannot be an assertion in M.

Now we will verify statements (1) to (3) of Theorem 4.4.15. Statement (1) is true since $T_A \subseteq M$. For statement (2), we remark that M was constructed so that the constants c occurring in T_N are exactly the constants occurring in M. (We ensured that $M(c)$ was nonempty, so c does indeed occur in M.) Furthermore, for each such c, $M_{\langle c \rangle} = (cM(c))_{\langle c \rangle} = M(c)$, as no other assertions in M begin with c or $*$. And $M_{\langle c \rangle}$ is known to satisfy $(T_N)_{\langle c \rangle}$, which shows statement (3). By Theorem 4.4.15, we conclude that $M \models_l S$, so M is shown to be a model for S.

For the "only if" part, take a model M for S. M is clash-free and hence does not contain $\varepsilon \perp$. Theorem 4.4.15 provides us with an atomic cover $T = T_A \cup T_N$ for which statements (1) through (3) hold. Statement (1) says that all the propositional atoms of T_N (except perhaps $\varepsilon \top$) are contained in M. Since M is clash-free, so is T_A. (The extra atom $\varepsilon \top$ never forms part of a clash.) Now take any c occurring in T_N. Because of (2), c also occurs in M, and by (3), we have $M_{\langle c \rangle} \models_l (T_N)_{\langle c \rangle}$. Since M is clash-free, so is $M_{\langle c \rangle}$, thanks to Proposition 4.4.10. Hence we have found a model for $(T_N)_{\langle c \rangle}$, proving this set satisfiable as claimed. □

We remark that this construction does not make use of the expressive power of labels with wildcards; all the labels constructed in the proof are ground. Our approach in Chapter 5 will

be quite different, allowing us to utilize them much more productively. In preparation for that, and for other sections, we will now take the "one label position at a time" characterization of $M \models_l \sigma F$ from Definition 4.3.1 (which only provides for $M \models_l * F$ and $M \models_l c F$) and transform it into a "closed-form" characterization. We continue on our earlier intuition that $M \models_l \sigma F$ should somehow entail $M \models_l \gamma F$, i.e. $M_{\langle\gamma\rangle} \models_l F$, for any realized ground instance γ of σ in M. But how do we characterize which ground labels *should* be realized in M? In other words, how do we test whether the existence requirements encapsulated in *any* of the constants of σ are met by the realized ground labels of M? This will be the intent of the following notion which is similar to that of realizability:

Definition 4.4.17 *A label σ is* constant-wise realizable *(cwr) in a set S of labelled formulas if for any prefix of σ of the form $\sigma' c$ and any ground instance γ of σ', whenever γ is realized in S, then γc is also realized in S.*

Example 4.4.18 Take the second set of Example 4.4.4: $S = \{c*c* \ldots p, *c*c \ldots q\}$. Every label σ consisting of $*$ and c, in any order, is constant-wise realizable in S; this is because the only realized ground instance γ of σ' in any prefix σ' consists only of c's, and then γc is also realizable.
In Example 4.4.6, the label c_1c_2* is constant-wise realizable, although, as we have seen, it is not realizable. One can show that $M \models_l c_1c_2* F$ for *any* formula F. We will see that this is true for any label which is constant-wise realizable but not realizable.

We state a few corollaries of Definition 4.4.17, the proof of which is easy and left to the reader:

Corollary 4.4.19 *If σ is constant-wise realizable in S, then so is any prefix of σ. If σ occurs in S, then σ is constant-wise realizable in S. Ground labels are constant-wise realizable iff they are realized, whereas $*^k$ is constant-wise realizable for any k in any S. If two labels σ_1, σ_2 unify, then (σ_1, σ_2) is constant-wise realizable iff σ_1 and σ_2 are both constant-wise realizable.*

As we have done for realizability, we study the behaviour of our new definition with respect to subscripted sets:

Proposition 4.4.20 *A label $c'\sigma$ is constant-wise realizable in S iff c' exists in S and σ is constant-wise realizable in $S_{\langle c'\rangle}$. A label $*\sigma$ is constant-wise realizable in S iff σ is constant-wise realizable in $S_{\langle c'\rangle}$ for all c' which exist in S.*

Proof: To prove this, it helps to restate Definition 4.4.17 slightly more formally: A label σ is constant-wise realizable in S, if:

$$\text{For every prefix of the form } \sigma' c \text{ and every } \gamma \in I(\sigma', S),\ \gamma c \in I(S). \tag{4.3}$$

Consider first the label $c'\sigma$. We will prove the first equivalence as follows: First we observe that the first instance of (4.3), namely $\sigma' = \varepsilon$, $c' = c$, is equivalent to "c' exists in S". We then use the established fact that c' exists in S to show the equivalence of the remaining instances of (4.3) with the statement "σ is cwr in $S_{\langle c' \rangle}$".
So take the smallest prefix ending in a constant, namely $\sigma' c = \varepsilon c'$. Then (4.3) reads: "If ε (the only ground instance of ε) is realized in S (which is always true), then c' is realized in S", or equivalently: c' exists in S, as desired.
Now consider the remaining instances of (4.3) which read:

$$\text{For all prefixes of the form } c'\sigma' c \text{ and all } c'\gamma \in I(c'\sigma', S),\ c'\gamma c \in I(S). \tag{4.4}$$

Since c' is known to exist in S, we can apply Lemma 4.4.7, parts (3) and (5), and replace $I(c'\sigma', S)$ by $c' I(\sigma', S_{\langle c' \rangle})$, and $c'\gamma c \in I(S)$ by $c'\gamma c \in c' I(S_{\langle c' \rangle})$. We can further simplify (4.4) to:

$$\text{For all prefixes (of } c'\sigma) \text{ of the form } c'\sigma' c \text{ and } \gamma \in I(\sigma', S_{\langle c' \rangle}),\ \gamma c \in I(S_{\langle c' \rangle}). \tag{4.5}$$

Finally, $c'\sigma' c$ is a prefix of $c'\sigma$, iff $\sigma' c$ is a prefix of σ. Now our statement reads:

$$\text{For all prefixes (of } \sigma) \text{ of the form } \sigma' c \text{ and } \gamma \in I(\sigma', S_{\langle c' \rangle}),\ \gamma c \in I(S_{\langle c' \rangle}). \tag{4.6}$$

which matches (4.3) once again, so it is equivalent to "σ is cwr in $S_{\langle c' \rangle}$", as claimed.
For the second part of the proposition, consider a label $*\sigma$. Any prefix partaking of (4.3) is of the form $*\sigma' c$. According to Lemma 4.4.7, part (4), the elements of $I(*\sigma', S)$ are exactly the ground labels of the form $c'\gamma$, where c' exists in S and $\gamma \in I(\sigma', S_{\langle c' \rangle})$. Likewise, we have $c'\gamma c \in I(*\sigma' c, S)$ iff $\gamma c \in I(\sigma' c, S_{\langle c' \rangle})$. So we restate (4.3) as:

$$\text{For every prefix of } \sigma \text{ of the form } \sigma' c, \text{ every } c' \text{ existing in } S, \text{ and every } \gamma \in I(\sigma', S_{\langle c' \rangle}), \text{ we have } \gamma c \in I(S_{\langle c' \rangle}). \tag{4.7}$$

This again matches (4.3), quantified over all c' existing in S. We conclude that $*\sigma$ is cwr in S iff σ is cwr in $S_{\langle c' \rangle}$ for all c' which exist in S, which is what we had to show. □
We use Proposition 4.4.20 to get our closed-form characterization of $M \models_l \sigma\ F$:

Theorem 4.4.21 *Given a set M of assertions and a formula $\sigma\, F$, we have $M \models_l \sigma\, F$ iff σ is constant-wise realizable in M and $M_{\langle\gamma\rangle} \models_l F$ for every realized ground instance γ of σ in M.*

Proof: We show this theorem by induction on σ. Note that ε is always cwr and it is its own (sole) rgi, so the statement reduces to "$M \models_l F$ iff $M_{\langle\varepsilon\rangle} \models_l F$", which is trivially true. Now assume the theorem holds for a label σ (and for any set M). Our goal is to show it for $c\sigma$ and $*\sigma$.

By definition, $M \models_l c\sigma\, F$ iff (a) c exists in M, and (b) $M_{\langle c\rangle} \models_l \sigma\, F$. By the induction hypothesis, (b) is true iff (b1) σ is cwr in $M_{\langle c\rangle}$ and (b2) $M_{\langle c\gamma\rangle} \models_l F$ for every $\gamma \in I(\sigma, M_{\langle c\rangle})$. By Proposition 4.4.20, (a) and (b1) combined are equivalent to "$c\sigma$ is cwr in M". In (b2), note that $\gamma \in I(\sigma, M_{\langle c\rangle})$ iff $c\gamma \in I(c\sigma, M)$ (by Lemma 4.4.7, part (5)); equivalently, $M_{\langle c\gamma\rangle} \models_l F$ for every rgi $c\gamma$ of $c\sigma$ in M. These two facts match exactly the right-hand side of the theorem for the label $c\sigma$.

As to the second case, $M \models_l *\sigma\, F$ iff (c) $M_{\langle c\rangle} \models_l \sigma\, F$ for all c which exist in M. We apply the induction hypothesis to (c), rewriting it as (c1) σ is cwr in $M_{\langle c\rangle}$, and (c2) $M_{\langle c\gamma\rangle} \models_l F$ for every $\gamma \in I(\sigma, M_{\langle c\rangle})$. Since (c1) holds for all c existing in M, Proposition 4.4.20 proves it equivalent to "$*\sigma$ is cwr in M". Statement (c2) in turn is quantified over all $c\gamma \in \bigcup\{cI(\sigma, M_{\langle c\rangle}) : c \text{ exists in } M\}$; but the latter set is just $I(*\sigma, S)$, according to Lemma 4.4.7, part (4). So (c2) is equivalent to "$M_{\langle c\gamma\rangle} \models_l F$ for every rgi $c\gamma$ of $*\sigma$ in M". Again we match the right-hand side of the theorem, which completes the induction proof on σ. □

Corollary 4.4.22 *Given a set S of $\mathcal{LBL}$-formulas and $M \models_l S$, if γ is a realized ground label in S, then γ is also a realized ground label in M.*

Proof: The label $\gamma = \varepsilon$ is an rgl in any set, so there is nothing to show. Write any non-empty label $\gamma = c_1 \cdots c_k$, and assume that the label $c_1 \cdots c_{i-1}$ is an rgl in M, $i = 1, \ldots k$. Since γ is an rgl in S, Proposition 4.4.3 asserts that some label of the form $\sigma_{i-1}c_i$ exists in S so that $c_1 \cdots c_{i-1}$ instantiates σ_{i-1}. $\sigma_{i-1}c_i$ in turn must be a prefix in the label σ of some formula $\sigma\, F$. Now Theorem 4.4.21 stipulates that σ be cwr in M; its prefix $\sigma_{i-1}c_i$ matches the form given in Definition 4.4.17, and $c_1 \cdots c_{i-1}$ is an rgi of σ_{i-1} in M (which was our induction hypothesis), so $c_1 \cdots c_i$ is an rgl in M. The proof now follows by induction on i. □

We see that the set of realized ground labels of any model for S must include at least the realized ground labels of S, but it is entirely possible (and sometimes necessary) to

introduce more realized ground labels. A simple example is $S = \{c_1\, p \vee c_2\, q\}$ whose only realized ground label is ε. But every model must have c_1 or c_2 as an additional realized ground label.

Let us now reconsider our earlier intuitions about the ground labels and variable assignments represented by a set of assertions. Equipped with the terminology of this section, we can specify clearly what a set M of assertions entails. First, M entails the *existence* of a label γ iff γ is a realized ground label in M. Secondly, it entails that a variable p is assigned *true* (*false*) in a label γ, iff there exists an assertion $\sigma\, p$ ($\sigma\, \neg p$) so that γ is a realized ground instance of σ in M. We now see why M must be clash-free in order to specify a consistent variable assignment. Moreover, we will see in the next section that a model M gives rise to a Kripke model $K(M)$ whose worlds are exactly the labels entailed by M and so that the variable assignments entailed by M match variable assignments in $K(M)$. This shows that these intuitively motivated definitions are consistent with our theory and with that of $\mathbf{K_{NNF}}$. Another proof of "consistency" with our theory is that the entailed facts about M can themselves be expressed as assertions, and that these assertions are indeed entailed by M:

Corollary 4.4.23 *For any set M of assertions and any ground label γ, M entails the existence of γ iff $M \models_l \gamma\, \top$. Furthermore, for any $p \in P$, M entails that p is assigned true (false) in γ iff $M \models_l \gamma\, p$ ($M \models_l \gamma\, \neg p$).*

Proof: Theorem 4.4.21 states that $M \models_l \gamma\, a$ iff γ is cwr in M (whereas Corollary 4.4.19 says that "cwr" and "rgl" coincide for ground labels), and $M_{\langle\gamma\rangle} \models_l a$ (here we have already applied the fact that a is realized, and it is the only rgi of itself). By Definition 4.3.1, the second fact is always true for $a = \top$, whereas for $a = p$ and $a = \neg p$ it is true iff $p \in M_{\langle\gamma\rangle}$ ($\neg p \in M_{\langle\gamma\rangle}$). In Proposition 4.3.5 we have characterized the elements of $M_{\langle\gamma\rangle}$ and found $a \in M_{\langle\gamma\rangle}$ iff $\sigma\, a \in M$ for some $\sigma \sqsubseteq a$.

We summarize our findings: $M \models_l \gamma\, \top$ iff γ is an rgl in M, and $M \models_l \gamma\, a$ (where $a = p$ or $a = \neg p$) iff γ is an rgl in M and there exists an assertion $\sigma\, a \in M$ so that $\sigma \sqsubseteq a$. But these are exactly the stated entailments that γ exist, and that p be assigned *true* (*false*) in γ, respectively. □

Another important consequence of Corollary 4.4.22 is the following:

Corollary 4.4.24 *If a set S of $\mathcal{LBL}$-formulas contains a clash, it is unsatisfiable.*

Proof: Let $\sigma\ \bot \in S$ or $\sigma'\ p, \sigma''\ \neg p \in S$ be the clash, and γ be the clash witness, i.e. an rgi of σ or of both σ' and σ''. Suppose there existed a clash-free model M so that $M \models_l S$. Since γ is realized in S, it must also be realized in M. Then Proposition 4.4.10 (iterated over the elements of γ) applies, implying that $M_{\langle\gamma\rangle}$ is clash-free. However, Theorem 4.4.21 asserts that $M_{\langle\gamma\rangle} \models_l \bot$ (which contradicts Definition 2.1.1), or that $M_{\langle\gamma\rangle} \models_l p$ and $M_{\langle\gamma\rangle} \models_l \neg p$. By Definition 4.3.1, $M_{\langle\gamma\rangle}$ must contain both p and $\neg p$, but then it is not clash-free, contradicting our finding above. □

This corollary justifies a shortcut similar to those used in tableau proofs: Assume our method of finding a model for a given set S consists of transforming S into equivalent but successively "simpler" sets. If along the process we find a set S' which contains a clash, we know that S' and hence S is unsatisfiable, without the need to transform S' further into a set of assertions. Now what if we can transform S all the way into an equivalent set of assertions, without encountering a clash? Then the following converse to Corollary 4.4.24 asserts that S is satisfiable:

Corollary 4.4.25 *A clash-free set M of assertions is a model for itself.*

Proof: This is not subsumed by Corollary 4.3.3 where we could not handle assertions of the form $\sigma\ \bot$. For these, Theorem 4.4.21 comes to our rescue. Since M is clash-free, σ cannot have an rgi, so the requirement $M_{\langle\gamma\rangle} \not\models_l \bot$ is vacuously true for all rgi γ. The requirement that σ be cwr in M is met, since σ is a label in M (Corollary 4.4.19). To all other labels $\sigma\ a \in M$ we can apply Proposition 4.3.2 to show $M \not\models_l \sigma\ a$. In summary, we get $M \not\models_l M$, and since M is clash-free, $M \models_l M$. □

4.5 Equivalence between Kripke and Labelled Semantics

We are now going to establish the relationship between the conventional Kripke semantics of $\mathbf{K_{NNF}}$ and the semantics of $\mathcal{LBL}$. Syntactically, $\mathbf{K_{NNF}}$-formulas φ are converted into labelled formulas simply by replacing $\Box$ with $*$ and $\Diamond$ with constants. (We call the result a *labelling* of φ.) However, this approach presents us with a number of obstacles to reasoning with these formulas in a useful and efficient manner.

First, since we did not specify *what* constants to substitute for $\Diamond$-operators, our way of labelling is highly nondeterministic, which makes it hard to describe as a function on

$\mathcal{L}(\mathbf{K_{NNF}})$. We overcome this obstacle by using a trick: the *inverse* of a labelling is a well-defined function on $\mathcal{L}(\mathcal{LBL})$.

Next, not every labelling preserves satisfiability. The simplest example for this is the set $\Phi = \{\Diamond\ p, \Diamond\ \neg p\}$. If we replace all $\Diamond$ with the constant c, we get $S = \{c\,p, c\,\neg p\}$, which cannot have a model since it is not clash-free. This impossibility is explained by the fact that any Kripke model for Φ must have (at least) two distinct w_0-successors, one of which satisfies p and the other $\neg p$; only if we replace the two $\Diamond$-operators by two distinct constants will the resulting labelled formula admit a model.

We will discover that choosing distinct constants for distinct $\Diamond$-operators always preserves satisfiability. (We will call such labellings *strict*, in analogy to strict Kripke models.) However, this strategy may result in unreasonably large models. For example, the set $\{\Diamond\ p, \Diamond\,(p \vee q)\}$ can be satisfied by a model with just one w_0-successor in which p is assigned *true* (compare Example 3.4.3). However, if we label the two $\Diamond$-operators with different constants c_1, c_2, we forfeit the chance to satisfy both formulas with the same assertion $c\,p$, since both c_1 and c_2 must occur in a satisfying model. We will discuss this problem at length in Section 4.8.

Definition 4.5.1 *The function* $\kappa : \mathcal{L}(\mathcal{LBL}) \mapsto \mathcal{L}(\mathbf{K_{NNF}})$ *is defined as*

- $\kappa(a) = a$ *for any propositional atom,*
- $\kappa(F_1 \circ \ldots \circ F_n) = \kappa(F_1) \circ \ldots \circ \kappa(F_n)$ *for* $\circ = \wedge, \vee$,
- $\kappa(*\,F) = \Box\ \kappa(F)$,
- $\kappa(c\,F) = \Diamond\ \kappa(F)$ *for any* $c \in D$.

We say that the $\mathcal{LBL}$-formula F is a labelling *of the $\mathbf{K_{NNF}}$-formula φ if $\varphi = \kappa(F)$. F is a* strict labelling *of φ if the restriction of κ to subformulas of F is one-to-one and for any two subformulas $c_1\,F_1$, $c_1\,F_2$ of F with the same constant c_1, we have $F_1 = F_2$. Labellings and strict labellings are defined analogously for sets of formulas.*

The advantage of κ being one-to-one on subformulas of F is that identical $\Diamond$-subformulas are assigned the same label. This is already an optimization, and, as we will see, a safe one, in that the labelling of a satisfiable formula remains satisfiable.

We immediately see that κ is a homomorphism which preserves atoms. Some more interesting properties of κ are stated in the next two propositions:

Proposition 4.5.2 *κ is depth-preserving.*

Proof: We restate this proposition as:

$$d(F) = d(\kappa(F)) \text{ for all } F \in \mathcal{L}(\mathcal{LBL}).$$

Note that this is a property stated on $\mathcal{LBL}$-formulas (not $\mathbf{K_{NNF}}$-formulas). We prove it by induction over the depth of F, according to Theorem 2.1.12. It is obviously true for all propositional atoms, i.e. all atoms of depth 0, since $\kappa(a) = a$. Assume it is true for all formulas of depth at most k. Now the $\mathcal{LBL}$-atoms of depth $k+1$ are exactly of the form $*F$ and $c\,F$, $c \in D$, where F is an $\mathcal{LBL}$-formula and $d(F) = k$. By the induction hypothesis, $d(\kappa(F))$ is also k, and $\kappa(*\,F) = \Box\ \kappa(F)$ and $\kappa(c\,F) = \Diamond\ \kappa(F)$ are both of depth $k+1$. We obtain the proof by induction on the depth of F. □

Proposition 4.5.3 *κ is onto. In fact, every $\mathbf{K_{NNF}}$-formula has a* strict *labelling.*

Proof: We prove the stronger version of this proposition directly. Define some one-to-one characteristic function $\chi : \mathcal{L}(\mathbf{K_{NNF}}) \mapsto D$ on the formulas of $\mathbf{K_{NNF}}$. (Such a function exists, since $\mathcal{L}(\mathbf{K_{NNF}})$ is countable, so the countable set D can accommodate all-distinct values of χ.) We now define an "inverse" κ' on $\mathbf{K_{NNF}}$. For propositional atoms, we set $\kappa'(a) = a$; furthermore, we set $\kappa'(\varphi_1 \circ \varphi_2) = \kappa'(\varphi_1) \circ \kappa'(\varphi_2)$ for $\circ = \wedge, \vee$, and $\kappa'(\Box\ \nu_0) = *\kappa'(\nu_0)$. Finally, and most interestingly, we set $\kappa'(\Diamond\ \pi_0) = \chi(\pi_0)\ \kappa'(\pi_0)$. The function κ' is clearly well-defined, and all values of κ' are $\mathcal{LBL}$-formulas; furthermore, we easily see that $\kappa(\kappa'(\varphi)) = \varphi$ for any $\mathbf{K_{NNF}}$-formula φ, which not only shows that κ is onto; also, κ, restricted to range(κ'), is a bijection. We conclude two things: first, range(κ') contains a labelling $F = \kappa'(\varphi)$ for every formula φ in $\mathbf{K_{NNF}}$, and secondly, κ is one-to-one on range(κ'), so it is certainly one-to-one on all subformulas of F. Finally, take any two formulas $c_1\ F_1 = \kappa'(\Diamond\ \pi_1)$, $c_1\ F_2 = \kappa'(\Diamond\ \pi_2)$. Then according to our definition of κ', $\chi(\pi_1) = c_1 = \chi(\pi_2)$, and since χ was one-to-one, π_1 and π_2 must be the same; hence $F_1 = F_2$. This shows that any labelling $\kappa'(\varphi)$ is strict. □

The characteristic function χ we used in the proof is just an enumeration of all $\mathbf{K_{NNF}}$-formulas. Such enumerations are also known as *Gödelizations* [9]. In fact, we could simply choose $D = \mathcal{L}(\mathbf{K_{NNF}})$, were it not for the fact that labels with such constants would be cumbersome to write out.

Having thus stated how to translate *formulas*, we now turn our attention to *models*. Given a model M for a labelled formula, how do we obtain a Kripke model for the corresponding

$\mathbf{K_{NNF}}$-formula? The idea is simple: Use the ground labels defined by M to "name" the worlds of W, and the propositional atoms inside the assertions to define the valuation. More precisely, this is how we translate a set M of assertions—syntactically, for now—into a Kripke structure:

Algorithm 4.5.4 *convert-to-kripke-model*

Parameters:

M: a *clash-free* set of assertions

Returns:

$K(M) = (W, R, V)$ a Kripke model with root world ε

begin

set $W := \{\varepsilon\}$

set $R := \emptyset$

foreach $p \in P$ **do**

if p occurrs as an unlabelled (positive) atom

set $V(p) = V_0(p) = \{\varepsilon\}$

else

set $V(p) = V_0(p) = \emptyset$

end if

done

foreach c occurring in M **do**

set $(W_c, R_c, V_c) :=$ *convert-to-kripke-model*$(M_{\langle c \rangle})$

set $W := W \cup \{c\sigma : \sigma \in W_c\}$

set $R := R \cup \{(\varepsilon, c)\} \cup \{(c\sigma, c\sigma') : (\sigma, \sigma') \in R_c\}$

foreach $p \in P$ **do**

set $V(p) := V(p) \cup \{c\sigma : \sigma \in V_c(p)\}$

done

done

return (W, R, V)

We easily see that Algorithm 4.5.4 constructs a well-formed Kripke structure whose worlds W are ground labels from D^*. The much harder part, which we now set out to accomplish, is to show that M and $K(M)$ are models for corresponding satisfiable formulas in $\mathcal{LBL}$ and $\mathbf{K_{NNF}}$, respectively.

Proposition 4.5.5 *For any set M of assertions and any c occurring in M, we have $K(M_{\langle c\rangle}) \cong K(M)_{\langle c\rangle}$.*

Proof: We notice that c is the root world of $K(M_{\langle c\rangle})$ after prefixing all its world labels with c. As stated in Definition 3.2.5, $K(M)_{\langle c\rangle}$ consists of the subset W' of worlds in W which are successors of c, with R and V accordingly restricted. But these are exactly the worlds from $K(M_{\langle c\rangle})$, introduced into W by prefixing all their labels with the constant c. Hence, the bijection $\mu : W_c \mapsto W'$ with $\mu(c\sigma) = \sigma$ establishes the isomorphism. A formal proof that $K(M_{\langle c\rangle})$ and $K(M)_{\langle c\rangle}$ are isomorphic wrt μ is left to the reader. □

Our original idea is reflected in the next proposition. Recall the definitions of realized ground labels and realized ground instances in Definition 4.4.1:

Proposition 4.5.6 *Let M be a set of assertions. An alternate characterization of $K(M)$ is $K(M) = (W', R', V')$, where*

- $W' = I(M)$,
- $R' = \{(\sigma, \sigma c) : \sigma c \in W'\}$,
- $V'(p) = \bigcup\{I(\sigma, M) : \sigma\, p \in M\}$.

Proof: Given any model M, let (W, R, V) be the Kripke model obtained by Algorithm 4.5.4 (with W_c, R_c and V_c as defined therein), and (W', R', V') the Kripke model as characterized above. We will prove these two models identical using induction on the depth of M. If M is of depth 0, then all assertions are propositional atoms; furthermore, $W' = I(M) = \{\varepsilon\}$, and R' is the empty relation, which corresponds exactly to the initial settings of W and R in Algorithm 4.5.4. (Since no constant occurs in M, these are the "final" settings of W and R.) Furthermore, for any $p \in P$, $V(p)$ and $V'(p)$ both contain ε exactly if $\varepsilon\, p \in M$, and they both contain no other label. (Note that $I(\varepsilon, M) = \mathcal{G}(\varepsilon, \{\varepsilon\}) = \{\varepsilon\}$.)

Now assume the proposition shown for any model of depth less than k, and let M be of depth k. To re-state the definition of W, we have $W = \{\varepsilon\} \cup \bigcup_{c \text{ in } M}\{c\sigma : \sigma \in W_c\}$. As part of the induction hypothesis we have $W_c = I(M_{\langle c\rangle})$, so we can write $W = \{\varepsilon\} \cup \bigcup\{cI(M_{\langle c\rangle}) : c \text{ exists in } M\}$, which is equal to $I(M) = W'$, by Lemma 4.4.7, part (3).

To show that $R = R'$, we prove that any label in W of the form σc is the successor of label σ, that $\sigma \in W$, and that there are no other related pairs in R: First, as we have just shown, every non-empty label in W is of the form $c'\sigma'$, $\sigma' \in W_{c'}$, and c' exists in M. If

$\sigma' = \varepsilon$ (and hence $c' = c$), then the required pair (ε, c) has been explicitly introduced into R. Otherwise, σ is of the form $c'\sigma''$, and $\sigma' = \sigma''c$ is in $W_{c'}$. Since $W_{c'} = I(M_{\langle c' \rangle})$ and the set of rgl is closed under prefixing, σ'' is also in $W_{c'}$, and by the induction hypothesis $(\sigma'', \sigma''c) \in R_{c'}$. According to the construction in Algorithm 4.5.4, $\sigma = c'\sigma'' \in W$, and $(\sigma, \sigma c) \in R$. Conversely, every pair introduced into R is of the form (ε, c') or $(c'\sigma, c'\sigma')$ where c' exists in R. The first case matches the definition of R' directly, whereas in the second case the induction hypothesis (on $M_{\langle c' \rangle}$) states that σ' is of the form σc and $\sigma c \in W$, which also matches the definition of R'. which concludes the induction step.

To show that $V(p) = V'(p)$ for all p, we re-state $V(p)$ in closed form as $V(p) = V_0(p) \cup \bigcup_{c \text{ in } M}\{cV_c(p)\}$, where $V_0(p)$ is the initial valuation in Algorithm 4.5.4. We also re-state $V'(p)$ by taking the union over all labels, split into three categories:

$$V'(p) = \bigcup_{\varepsilon\, p \in M} I(\varepsilon, M) \cup \bigcup_{*\sigma\, p \in M} I(*\sigma, M) \cup \bigcup_{c \text{ ex. in } M} \bigcup \{I(c\sigma, M) : c\sigma\, p \in M\}.$$

(The extra "$\bigcup_{c \text{ ex. in } M}$" operator has been introduced as "syntactic sugar"; the condition "c in M" is already implied by $c\sigma\, p \in M$.) Now observe that the first term is equivalent to $V_0(p)$, whereas the other terms can be rewritten using Lemma 4.4.7, parts(4) and (5):

$$V'(p) = V_0(p) \cup \bigcup_{*\sigma\, p \in M} \bigcup_{c \text{ ex. in } M} \{cI(\sigma, M_{\langle c \rangle})\} \cup \bigcup_{c \text{ ex. in } M} \bigcup_{c\sigma\, p \in M} \{cI(\sigma, M_{\langle c \rangle})\}.$$

Since all sets are finite and c is not a bound variable, we can reverse the order of the '$\bigcup$' operators in the second term. Furthermore, all labels in the second and third term are preceded by c, so we can pull it out of the set union:

$$V'(p) = V_0(p) \cup \bigcup_{c \text{ ex. in } M} c\left(\bigcup_{*\sigma\, p \in M} \{I(\sigma, M_{\langle c \rangle})\} \cup \bigcup_{c\sigma\, p \in M} \{I(\sigma, M_{\langle c \rangle})\} \right).$$

Now the inner terms are identical, and the two criteria $*\sigma\, p \in M$ and $c\sigma\, p \in M$ comprise exactly the elements of M which contribute an assertion $\sigma\, p$ to $M_{\langle c \rangle}$:

$$V'(p) = V_0(p) \cup \bigcup_{c \text{ ex. in } M} c\left(\bigcup_{\sigma\, p \in M_{\langle c \rangle}} \{I(\sigma, M_{\langle c \rangle})\} \right).$$

On comparing the *inner* union with our original definition of $V'(p)$, we discover that it describes the valuation of $K(M_{\langle c \rangle})$ which is V_c by the induction hypothesis. So we can write:

$$V'(p) = V_0(p) \cup \bigcup_{c \text{ ex. in } M} \{cV_c(p)\},$$

which is identical to our re-written definition of $V(p)$. This concludes the induction step, and the proof follows by induction on $d(M)$. □

Corollary 4.5.7 *For any set of assertions M, $K(M)$ is a tree.*

Proof: First, we know that $W = I(M)$ is closed under prefixes. Secondly, we showed that each rgi in M of length $k > 0$ becomes an R-successor of its own $k-1$-prefix, and there are no other R-pairs. From this it is easy to infer that (W, R) is a tree. □

Now we will show that $K(M)$ delivers on what we defined it for, that is whenever M is a model for a labelling of Φ, then $K(M)$ is a model for the original formula Φ. Let us consider the base case first:

Proposition 4.5.8 *Let $M = F = \Phi$ be a* clash-free *set of propositional atoms. Then $K(M) \models_k \Phi$.*

Proof: Obviously Φ is its own atomic cover. The model $K(M)$ is of the form $(\{\varepsilon\}, \emptyset, P \cap \Phi)$. As a special case of Theorem 3.3.1, $K(M) \models_k \Phi$ iff V_ε is consistent with Φ. But this is the case, since $\bot \notin \Phi$, and whenever $p \in \Phi$, then $p \in V_\varepsilon$, and in all other cases (including all cases where $\neg p \in \Psi$), $p \notin V_\varepsilon$. (Compare Definition 3.1.2.) □

And now we prove the general case:

Theorem 4.5.9 *Let Φ be a set of $\mathbf{K_{NNF}}$-formulas, S a labelling of Φ, and M a* clash-free *set of assertions. If $M \models_l S$, then $K(M)$ is a Kripke tree model for Φ.*

Proof: Although this theorem is originally a statement on sets of $\mathbf{K_{NNF}}$-formulas and their labellings, we recall that $\Phi = \kappa(S)$ and that κ is onto, so we can consider it a statement on sets of $\mathcal{LBL}$-formulas instead, whence it reads:

$$\text{If } M \models_l S, \text{ then } K(M) \models_k \kappa(S). \tag{4.8}$$

(From Corollary 4.5.7 we already know that $K(M)$ is a tree.) Now (4.8) matches Property (1) in Proposition 2.1.18. Since κ is a homomorphism, we infer that (4.8) is preserved under propositional concatenation. Thus understood, we prove it using the already familiar principle of induction over the depth of F (which is also the depth of φ, since κ is depth-preserving).

In Proposition 4.5.8 we have already shown the theorem for sets of atoms of depth 0. Assuming the theorem is true for all formulas of depth at most k (and *any* clash-free set of

assertions M), we wish to show it for atoms of depth $k+1$. As before, we distinguish the two cases $*\,F$ and $c\,F$, $c \in D$, where F is of depth k.

By definition we have $M \mid\!\!\sim_l c\,F$ iff c is a label in M and $M_{\langle c\rangle} \mid\!\!\sim_l F$. Then by the construction in Algorithm 4.5.4, the world ε has a direct successor c in $K(M)$. Proposition 4.5.5 showed that $K(M)_{\langle c\rangle} \cong K(M_{\langle c\rangle})$. Now since M is clash-free, so is $M_{\langle c\rangle}$ according to Proposition 4.4.10, and by the induction hypothesis (which was to hold for *any* clash-free set of assertions), $M_{\langle c\rangle} \models_l F$ implies $K(M_{\langle c\rangle}), \varepsilon \models_k \kappa(F)$. Thanks to Corollary 3.2.4 we get $K(M)_{\langle c\rangle}, c \models_k \lambda(F)$ (c is the isomorphic image of the root world of $K(M_{\langle c\rangle})$), and further by Proposition 3.2.6, $K(M), c \models_k \kappa(F)$. This establishes world c as a witness showing $K(M), \varepsilon \models_k \Diamond\, \kappa(F) = \kappa(c\,F)$.

As to the second case, we have $M \mid\!\!\sim_l *\,F$ iff $M_{\langle c\rangle} \mid\!\!\sim_l F$ for all constants c occurring in M. In $K(M)$, these c are exactly the worlds created as successors of ε. We apply the same steps as in the first case, arriving at $K(M), c \models_k \kappa(F)$ for all successors c of ε. This proves $K(M), \varepsilon \models_k \Box\, \kappa(F) = \kappa(*\,F)$. The theorem now follows by induction on $d(F)$. □

Now we would like to derive a similar result for the converse. Suppose we are given a set Φ of $\mathbf{K_{NNF}}$-formulas and a labelling S of Φ, and we know a Kripke model K so that $K, w_0 \models_k \Phi$ for some (root) world w_0 of K. How can we "convert" K into a model for S? Before we formally state and prove the theorem, let us discuss the task at hand and the main idea for solving it. The challenge we face is that a Kripke model does not specify how the successors of w_0 are to be labelled. Instead, the labelling of worlds is largely dependent on the labelling S, and we must somehow find a mapping from labels of S to worlds in K. Note that K may have redundant worlds; these may not need to correspond to any label in S (as they do not correspond to any formula in Φ either). Likewise, S may have labels which do not need to be mapped to worlds, such as labels occurring in disjuncts which are never evaluated. Furthermore, an accessible world may attest the truth of more than one π-formula, but different π-formulas receive different labels in a strict labelling. So the mapping may not be one-to-one.

Our proof idea is that K witnesses the satisfiability of an atomic cover Ψ of Φ, and we can construct a corresponding cover T for S. Now every formula of the form $c\,F$ in T, $c \in D$, is the labelling of a π-formula $\Diamond\, \pi_0$ in Ψ, and this existential formula must "materialize" in some accessible world w in K; furthermore, the submodel K_w must satisfy π_0 (as well as the ν_0 for any $\Box\, \nu_0 \in \Psi$). So we "map" the constant c to the world w and proceed recursively through the submodel K_w. In doing this for all c existing in T, we find the ground labels

from which we construct our model M. The assertions to be stated in M are obtained from the propositional atoms in the atomic covers as we encounter them.

Theorem 4.5.10 *Let Φ be a set of $\mathbf{K_{NNF}}$-formulas, S a* strict *labelling of Φ, and $K = (W, R, V)$ a Kripke model (not necessarily a tree) for Φ. Then a model M exists for S.*

Proof: Our proof will be constructive; as we motivated above, we will also construct a partial mapping $f : D^* \mapsto W$. Our proof will be by induction over $k = d(\Phi)$, the induction hypothesis stating that we can find a model M and mapping f for any formula of depth less than k.

We choose $w_0 \in W$ so that $K, w_0 \models_k \Phi$. Theorem 3.3.1 warrants the existence of an atomic cover of Φ of the form $\Psi = \Psi_A \uplus \Psi_P \uplus \Psi_N$, so that K and w_0 provide models for these components as specified further below. Since S is a labelling of Φ, we have $\Phi = \kappa(S)$. Thanks to Theorem 2.1.36 and the fact that κ is a homomorphism which preserves atoms, S also has an atomic cover T such that $\Psi = \kappa(T)$. We write $T = T_A \uplus T_N$, where T_A and T_N comprise the propositional atoms and the formulas with nonempty labels, respectively. We also distinguish $T_* \subseteq T_N$, the set of formulas whose labels begin with $*$. It is not hard to see that κ maps T_A to Ψ_A (in fact we have $T_A = \Psi_A$, as atoms are mapped to themselves), T_* to Ψ_N, and $T_N - T_*$ to Ψ_P.

The first property in Theorem 3.3.1 states that V_{w_0} is consistent with Ψ_A. By Corollaries 2.2.4 and 2.2.8, this is only possible if $\Psi_A = T_A$ is consistent, i.e. clash-free. Furthermore, thanks to Corollary 4.3.3, $T_A \not\models_l T_A$, which shows $T_A \models_l T_A$. We define $M = T_A$ and $f(\varepsilon) = w_0$.

Now T_N consists of formulas with nonempty labels. If $d(S) = 0$, then T_N must be empty, so $T = T_A$, and we just showed that $M = T_A$ is clash-free and provides a model for itself. (This is the base case of the induction proof.) Otherwise, reconsider the other two conditions in Theorem 3.3.1:

$$\text{For every } \pi \in \Psi_P \text{ there exists a world } w \in R(w_0) \text{ so that } K_w, w \models_k \pi_0, \quad (4.9)$$

$$\text{and for every } \nu \in \Psi_N \text{ and every world } w \in R(w_0),\ K_w, w \models_k \nu_0. \quad (4.10)$$

Any of these π_0 and ν_0 are of depth less than k; so is any set containing these formulas. This will later allow us to apply the induction hypothesis.

Now take any constant c occurring in T_N. Since F is a *strict* labelling, there is exactly one formula in T_N of the form $c\sigma' F'$ (the definition of strictness included that any two formulas

preceded by the same constant are identical, and T_N as a set does not contain duplicates); its image is $\kappa(c\sigma'\ F') = \Diamond\ \kappa(\sigma'\ F') \in \Psi_P$. So condition (4.9) applies, which means that w_0 has an R-successor w in K so that $K_w, w \models_k \kappa(\sigma'\ F')$; secondly, for every $*\sigma''\ F'' \in T_*$, we have $\kappa(*\sigma''\ F'') = \Box\ \kappa(\sigma''\ F'') \in \Psi_N$, so by means of condition (4.10) we get $K_w, w \models_k \kappa(\sigma''\ F'')$. Now consider the set $T(c) = \{\sigma'\ F'\} \cup \{\sigma''\ F'' :\ *\sigma''\ F'' \in T_*\}$ containing all the formulas just mentioned. As seen above, the submodel K_w is a Kripke model for $\kappa(T(c))$. Since each label in T contributes at most one (shortened) label to $T(c)$, $T(c)$ is still a strict labelling of $\kappa(T(c))$. Therefore, the induction hypothesis grants that we can find a model for $T(c)$ which we augment using Lemma 4.4.12, if necessary, to a *nonempty* model $M(c)$. (This augmentation does not introduce new rgis into $M(c)$.) We further obtain a mapping f_c. Now set $M = M \cup cM(c)$ (recall that $cM(c)$ is a shorthand for $\{c\sigma\ a :\ \sigma\ a \in M(c)\}$), and $f(c\gamma) = f_c(\gamma)$ for all $\gamma \in \mathrm{Dom}\, f_c$, and repeat this for all c occurring in T_N[4]. In order to prove that M is a model, we first demonstrate $M \models_l S$, or equivalently: M meets the preconditions of Theorem 4.4.15:

- $T_A \subseteq M \cup \{\varepsilon\ \top\}$ because T_A is part of M.
- Every c occurring in T_N also occurs in M: $M(c)$ was chosen nonempty, so c occurs in $cM(c)$ which is part of M.
- For every c' occurring in M, we have $M_{\langle c'\rangle} \models_l (T_N)_{\langle c'\rangle}$: We did not introduce any constants other than the c occurring in T_N, and we did not make use of $*$ in M at all, so $M_{\langle c\rangle} = M(c)$. Furthermore, the only assertions in T_N which begin with c or $*$ are those in T_*, plus the formula $c\sigma'\ F'$; from this we see that $(T_N)_{\langle c\rangle} = T(c)$. But $M(c) \models_l T(c)$, as we stated before.

Secondly, M is clash-free: For T_A we have already stated this, and for any of the $M(c)$, clash-freeness is warranted by the induction hypothesis. But then each constituent $cM(c)$ is clash-free in itself as well. These constituents are distinguished from one another and from T_A, in that their labels have nonempty labels beginning with a distinct constant. Thus there can be no two unifying labels anywhere across all the assertions in M, so M must be clash-free. □

[4]If no constants exist in T_N, as is the case in any labelling of $\Phi = \{\Box\ \bot\}$, for instance, this part is vacuous. In particular, all assertions in M will have empty labels, which corresponds to a Kripke model for Φ with only one world w_0.

We remark that a strict labelling was required only in order to ensure distinctness of labels across different constituents in M, which we needed in order to guarantee clash-freeness. If S is a non-strict labelling and we can show that M as constructed above is clash-free, then M is still a model for S (but possibly smaller than any model for a strict labelling). Once again we observe a potential for more compact models by choosing non-strict labellings, at the risk of losing satisfiability for this labelling.
We summarize the previous two theorems in one fundamental equisatisfiability result:

Theorem 4.5.11 *Let Φ be a set of $\mathbf{K_{NNF}}$-formulas and S a strict labelling of Φ. Then Φ is satisfiable iff S is satisfiable.*

Apart from this, Theorem 4.5.10 also gives rise to an upper bound for the size of a model M. Let us state a number of properties of the mapping f:

Proposition 4.5.12 *The function f defined in the proof of Theorem 4.5.10 is well-defined, and we have* $\mathrm{Dom}\, f = I(M)$ *and* $\mathrm{Range}\, f \subseteq R^*(w_0)$. *Furthermore, if $\gamma = c_1 \cdots c_k$, $k \geq 0$, then the sequence $f(\varepsilon), f(c_1), \ldots, f(c_1 \cdots c_k)$ describes a path from w_0 to $f(\gamma)$ in K.*

Proof: We show that these properties hold initially and that they are preserved in the induction step, so we can make them part of the same induction proof as in Theorem 4.5.10. First, we have $f(\varepsilon) = w_0$. For a formula of depth 0, M has ε as its only rgi, and $w_0 \in R^0(w_0)$. For the last statement, the only existing label ε defines the empty path from w_0 to w_0 in K. All other values of f are introduced through one of the f_c, that is, each non-empty label $\gamma \in \mathrm{Dom}\, f$ begins with a constant c occurring in T. By the induction hypothesis, f_c is well-defined and satisfies the following properties: $\mathrm{Dom}\, f_c = I(M(c))$ and $\mathrm{Range}\, f \subseteq R^*(w)$. ($w$ is the successor of w_0 which was found to correspond to c.) Furthermore, if $\gamma' = c_2 \cdots c_k$, $k \geq 1$, then $f_c(c_2), \ldots, f_c(c_2 \cdots c_k)$ describes a path from w to $f_c(\gamma')$.
Let $c\gamma$ be any nonempty ground label. If c does not occur in M (so $c\gamma$ is not realized in M), then c does not occur in T either. (Each $M(c)$ was chosen nonempty, so c occurs in $cM(c)$ which is a subset of M.) But all arguments we added to f were elements $c'\gamma'$, where c' occurs in T, so we are certain that $c\gamma \notin \mathrm{Dom}\, f$. If c does occur in M but $\gamma \notin I(M(c))$ (so $c\gamma$ is still not realized in M), then $\gamma \notin \mathrm{Dom}\, f_c$ and hence $c\gamma \notin \mathrm{Dom}\, f$. Finally, if c occurs in M and $\gamma \in I(M(c))$, then $c\gamma \in I(M)$ but also $\gamma \in \mathrm{Dom}\, f_c$ and hence $c\gamma \in \mathrm{Dom}\, f$. Since each constant occurs only once in T, the argument $c\gamma$ is added to f only once during the construction of f, so f is well-defined. Since all values of f_c are direct or indirect successors

of w and w is a successor of w_0, all values of f are successors of w_0 as claimed. For the last statement, we have $f(c_1) = f_{c_1}(\varepsilon) = w$ which is a successor of w_0, and the desired path from w_0 to $f(\gamma)$ is obtained by extending the path from w to $f(c_2 \cdots c_k)$ provided by the induction hypothesis, with the initial arc from w_0 to w. □

Corollary 4.5.13 *If $K = (W, R, V)$ is a* strict *Kripke model for Φ and S a strict labelling of S, then a model M for S can be constructed so that the function f in the proof of Theorem 4.5.10 is one-to-one.*

Proof: We prove this by slightly modifying the proof of Theorem 4.5.10. As before, let w_0 be the root of K. We recall Definition 3.4.1 which says that whenever κ is strict, Φ has an atomic cover $\Psi = \Psi_A \uplus \Psi_N \uplus \Psi_P$ as above, but each $\pi \in \Psi_P$ corresponds to a *distinct* successor w of w_0 so that K_w is a strict Kripke model for π_0 and for each $\nu \in \Psi_N$ every K_w is a strict model for ν_0. We determine the atomic cover T for S as before. Now for each c occurring in T, we take the (unique) formula $c\sigma'\, F'$ in which c occurs, and let $\Diamond\ \pi_0^c = \kappa(c\sigma'\, F')$. Since κ is one-to-one, all the π_0^c are distinct. Since K is strict, we find distinct successors w^c of w_0 so that $K, w^c \models_s \pi_0^c$; of course we also have $K, w^c \models_s \nu_0$ for every $\Box\ \nu_0 \in \Psi_N$. Just as in the proof of Theorem 4.5.10 we apply the induction hypothesis to the set $T(c) = \{\sigma'\, F'\} \cup \{\sigma''\, F'' : *\sigma''\, F'' \in T_*\}$ and the model K_{w^c} which, as we have just seen, is a strict model for $\kappa(T(c))$. This gives us a model $M(c)$ for $T(c)$ and a one-to-one function f_c. As before, we define $f(c\gamma) = f_c(\gamma)$ for all $\gamma \in \mathrm{Dom}\, f_c$ and for all c occurring in T_N.

Let us now prove that f is one-to-one. First we observe that $f(c) = w^c \neq w^{c'} = f(c')$ for any two distinct $c, c' \in \mathrm{Dom}\, f$. Now we recall that every strict model is a tree. These two facts will be sufficient to show that any two distinct labels are mapped to distinct worlds. First, if we have an empty label ε and a nonempty label $c\gamma$, then by Proposition 4.5.12 there exists a path from w_0 to $f(c\gamma)$ via $f(c) = w^c$; in particular, this path is nonempty. Since trees do not have cycles, it cannot end in w_0, so $f(c\gamma) \neq w_0$. Now consider two labels $c\gamma$ and $c\gamma'$ with identical initial constant and $\gamma \neq \gamma'$. Then $f(c\gamma) = f_c(\gamma) \neq f_c(\gamma') = f(c\gamma')$, where the inner inequality holds because f_c is one-to-one. Finally, consider two labels $c\gamma$ and $c'\gamma'$ with distinct initial constants. By Proposition 4.5.12 there exist two paths from w_0 to $f(c\gamma)$ and $f(c'\gamma')$ via $w^c = f(c)$ and $w^{c'} = f(c')$, respectively. Since these two worlds are distinct, the two paths are not equal, and the fact that K is a tree model implies that $f(c\gamma)$ and $f(c'\gamma')$ are distinct. This finishes the proof. □

Corollary 4.5.14 *Let $K = (W, R, V)$ be a* strict *Kripke model for a set Φ of $\mathbf{K_{NNF}}$-formulas. Then any strict labelling S of Φ has a model M so that $|I(M)| \leq |W|$ and whose overall size is $O((|P| + 1) \times d(\Phi) \times |W|)$. Here P is to be understood as the (finite) set of propositional variables used in Φ.*

Proof: The first claim follows directly from Proposition 4.5.13 and the facts that $\mathrm{Dom}\, f = I(M)$, $\mathrm{Range}\, f \subseteq W$, and that $|\mathrm{Dom}\, f| = |\mathrm{Range}\, f|$ for any one-to-one function. For the second claim, recall that all assertions introduced into M arise from consistent sets T_A of propositional atoms. Any such set mentions each propositional variable and $\top$ at most once, and this does not change when T_A is augmented by $\varepsilon\, \top$. So for each label in $I(M)$, at most $|P| + 1$ assertions are introduced. In the course of constructing M the labels of each assertion are prefixed, but only to a maximum length equal to the depth of Φ, bounding the length of each assertion. □

This result guarantees that S always has a model with no more ground labels than the number of worlds in a minimal strict Kripke model for Φ. But do the other factors ($|P|+1$) and $d(\Phi)$ result in an "absolute" size of M which could be much larger than K? This depends somewhat on the exact representation of K. If we assume that the worlds of W are represented by labels similar to those in M (so their average size is $\Theta(d(\Phi))$), and that V is not sparse (that is, $|V(p)| = \Theta(|W|)$ for any variable $p \in P$), then the size of K itself is of order $\Theta(|P| \times d(\Phi) \times |W|)$, and M is not significantly larger. In the worst case, the factors $d(\Phi)$ and $|P|$ can never exceed the length of the original formula, so $|M|$ is polynomial in $|S| + |W|$; but in practice $d(\Phi)$ and $|P|$ are logarithmic in $|S|$, whereas $|W|$ is of size $\Theta(|S|)$ and can be exponentially larger than $|S|$. Hence $|W|$ must be considered the dominant factor in the size of $|M|$. To summarize, we can say that $|M|$ is bounded by the size of K, albeit not quite by a constant factor in some extreme cases. On the other hand, we have not utilized the expressive power of labels with wildcards yet. (The model M above has only ground labels.) In the next two sections, we will show how they can be used to obtain lower bounds for $|M|$ which are much smaller than the size of K.

Corollary 4.5.15 *Any lower bound on the size of models for a (strict or non-strict) labelling of Φ is a lower bound on the size of strict Kripke models for Φ.*

Proof: For strict labellings, this is Corollary 4.5.14. For a non-strict labelling S of Φ, we only sacrifice the guarantee that S has a model, but for any model it does have, the argument in Corollary 4.5.14 is still valid. □

4.6 Complexity Results for Model Checking

In Section 2.3, we measured the complexity of some reasoning tasks in terms of membership checks. As we take these considerations further by studying our various types of atoms, we find that simple membership checks are no longer sufficient. Let us begin with the simple task of deciding whether a set M of assertions entails the existence of a ground label γ. As we know, this is equivalent to showing that γ is a realized ground label in M. Now the "rgl check" is not elementary, but it is easily expressed in terms of *instance checks* which in turn are string matches; if k is the length of a label, then this operation can be performed in $O(k)$ steps.

Proposition 4.6.1 *Given a set S of labelled formulas and a ground label $\gamma = c_1 \cdots c_k$, it involves $O(|S|)$ elementary steps to verify whether γ is realized in S.*

Proof: According to Proposition 4.4.3, we must verify for every $i = 1, \ldots, k$ that some label in S has a prefix of the form $\sigma_{i-1}c_i$, so that $\sigma_{i-1} \sqsubseteq c_1 \ldots c_{i-1}$. We can perform these checks in a smart way, so that each element of S needs to be checked only once, as shown in the algorithm given below. □

Algorithm 4.6.2 *is-realized*

Parameters:

$\gamma = c_1 \cdots c_k$: a ground label

S: a set of formulas

Returns: *true*, if γ is realized, *false* otherwise

begin

set $found_1$:= *false*, ..., $found_k$ = *false*.

foreach formula $\sigma\ F$ in S **do**

Find the maximal j-prefix σ' of σ, $j \leq k$, so that $\sigma' \sqsubseteq c_1 \cdots c_j$

foreach i from 1 to j **do**

if the ith position of σ' is a constant (i.e. c_i)

set $found_i$:= *true*

end if

done

done

if $found_i = true$ for all $i = 1, \ldots, k$
  return *true*
else **return** *false*
end if end

Next, how hard is it to decide whether M entails a variable assignment of p to *true* or *false* in an instance γ? We know that M does so iff γ is a realized ground instance of some label σ in M, so that $\sigma\, p \in M$ or $\sigma\, \neg p \in M$, respectively. This condition is no harder to verify than the previous one; beyond verifying that γ is realizable, we have only one more instance check, namely $\sigma \sqsubseteq \gamma$. In fact, we can integrate it into the above algorithm without incurring any extra instance check: At the point where we find the maximal j-prefix in the label of $\sigma\, a \in M$, if we find that $a = p$ ($a = \neg p$) and $j = |\sigma| = |\gamma|$, we have obtained an assertion entailing that p is assigned *true* (*false*) in γ. Then we only need to continue verifying whether γ is realized or not. We summarize these considerations as:

Corollary 4.6.3 *Given a set of assertions M and an assertion $\gamma\, a$ with a ground label, checking whether $M \models_l \gamma\, a$ takes $O(|M|)$ instance checks.*

Not surprisingly, finding whether a label σ is *realizable* in a set S is also an important reasoning task. We obviously do not want to construct all realized ground labels of S and check whether any of them instantiates σ, as S may have exponentially many realized ground labels. The surprise is that realizability is a hard problem even for simple-looking labels:

Proposition 4.6.4 *Given a positive integer k, consider the problem of deciding whether the label $*^k$ is realizable in a set S of $O(k)$ labelled formulas of size $O(k)$. This problem is NP-hard in the worst case.*

Proof: This proposition has an analogue in the theory of satisfiability in description logics, namely a theorem that unsatisfiability in the description logic $\mathcal{ALE}$ is NP-hard [20]. This theorem in turn can be proved by a polynomial-size reduction to the NP-complete problem 1-in-3-SAT [27]. To keep our treatment self-contained—we have not introduced description logics—we adapt the same reduction idea to the setting of $\mathcal{LBL}$.
Given a set of propositions $P = \{p_1, \ldots, p_n\}$ and a set of clauses $C_1, \ldots, C_m$, each of which contains at most three of the p_j as (positive) literals, the 1-in-3-SAT problem asks whether there is a variable assignment V so that exactly one variable per clause is assigned *true*.

For example, if $P = \{p_1, p_2, p_3, p_4\}$, $C_1 = \{p_1, p_4\}$, $C_2 = \{p_2, p_3\}$, $C_3 = \{p_3, p_4\}$, then the assignment $V(p_2) = V(p_4) = \mathit{true}, V(p_1) = V(p_3) = \mathit{false}$ provides a solution, as does the assignment $V(p_2) = V(p_4) = \mathit{false}, V(p_1) = V(p_3) = \mathit{true}$.

We now encode this problem into a variable-free formula in $\mathcal{LBL}$. We need n constants from D, written for simplicity of notation as integers 1 through n. Let $k = 2m$, and $S = \{\sigma_1\sigma_1 \top, \ldots, \sigma_n\sigma_n \top\}$, where σ_j corresponds to p_j in the following way:

$$\sigma_j = x_{1,j} \cdots x_{m,j}, \text{ where } x_{i,j} = \begin{cases} c_j & p_j \in C_i \\ * & \text{otherwise.} \end{cases}$$

It is easy to see that this translation makes the problem at most polynomially larger. All labels are doubled; we will see later why. In the example above, S contains the following assertions:

$$1{*}{*}\ 1{*}{*}\ \top$$

$$*2{*}\ *2{*}\ \top$$

$$*33\ *33\ \top$$

$$4{*}4\ 4{*}4\ \top$$

We can see that $*^{2m}$ indeed has realized ground instances in S, namely $\gamma = 424424$ and $\gamma = 133133$. Note how the constants in these ground instances correspond to the propositions set *true* in the variable assignments above. This is also the idea of the equivalence proof.

Claim: $I(*^{2m}, S)$ is nonempty iff the problem instance has a satisfying assignment.

Assume that a variable assignment V for 1-in-3-SAT exists. Pick $\Sigma = \{\sigma_j : V(p_j) = \mathit{true}\}$. We claim that $\gamma = \mathrm{mgu}(\{\sigma_j : V(p_j) = \mathit{true}\})$ exists and is ground. (Proposition 4.4.5 then implies that $\gamma\gamma$ is an rgi of $*^{2m}$.) To prove our claim, note that the ith of the m positions in all the σ_j (the ith column in the above matrix, if you like) corresponds to the clause C_i. As required, exactly one of the variables in C_i is set to *true*, i.e. exactly one of those σ_j which have $x_{j,i} = c_j$ is chosen into the set Σ. So among the labels in Σ, exactly one label has a constant in the ith position. Since this is true for all $i = 1, \ldots, m$, the unifier of all σ_j in Σ exists (there can be no two different constants at any position) and is ground (each position is instantiated by exactly one constant found at that position).

Let us now show the converse. Suppose $\gamma\gamma' = z_1 \cdots z_m z_{m+1} \cdots z_{2m}$ is an rgi of $*^{2m}$. Define $\Sigma = \{\sigma_j : j \text{ occurs in some position of } \gamma\gamma'\}$. We claim that $\gamma = \mathrm{mgu}(\Sigma)$ and that $\gamma = \gamma'$.

To show this, take any j occurring in some position z_{m+i}. According to Proposition 4.4.3, S must have a label with constant j at its $(m+i)$th position. Since only σ_j contains the constant j at all, the label in question can only be (an $(m+i)$-prefix of) $\sigma_j\sigma_j$. Also by Proposition 4.4.3, the $(m+i-1)$-prefix of $\gamma\gamma'$ instantiates that of $\sigma_j\sigma_j$. In particular, we must have $\sigma_j \sqsubseteq \gamma$. Also, since σ_j evidently has constant j in its i-th position, the corresponding position in γ, namely z_i, must also be j. Since this holds for any $i = 1, \ldots, m$, we get $\gamma = \gamma'$. Thirdly, we showed that γ is a common instance of all σ_j where j occurs in γ, i.e. of all $\sigma_j \in \Sigma$. Hence the mgu exists, and γ instantiates it. But now observe that in any position $i = 1, \ldots, m$, the label σ_{z_i} has a constant in this position; therefore the mgu must have the same constant, i.e. it is ground.

To obtain the satisfying valuation for 1-in-3-SAT, we simply assign $V(p_j) = \mathit{true}$ whenever $\sigma_j \in \Sigma$, and $V(p_j) = \mathit{false}$ otherwise. By following the first part of the proof in the reverse direction, we show that exactly one variable per clause (which corresponds to the one label per i with a constant in its ith position) is set to *true*, as required. □

A number of reasoning tasks involve checking realizability, which determines their respective complexity:

Proposition 4.6.5 *Given a set S of labelled formulas and a label σ, deciding whether σ is realizable in S (i.e. $I(\sigma, S)$ is nonempty) is NP-complete.*

Proof: In Proposition 4.6.4 we showed NP-hardness for a subproblem of the given problem. To show that the general problem is in NP, we take any problem instance and guess a ground label nondeterministically. As we showed in Proposition 4.6.1, it takes linear time to verify that γ is indeed an rgi of σ. □

Corollary 4.6.6 *Given a set M of assertions, deciding whether M has a clash is NP-complete.*

Proof: Proving that $\sigma\ \bot$ constitutes a clash in M involves a check whether σ is realizable, so this problem is at least as hard as realizability. To show that it is NP-easy, note that if M has $2n$ elements, then there can be at most n^2 pairings of the form $\sigma\ p, \sigma'\ \neg p$. In each of these pairs, we need to check, whether (σ, σ') exists and is realizable in M; computing an mgu takes $O(\max(|\sigma|, |\sigma'|)$ steps, and checking realizability is NP-complete. Since all these $O(n^2)$ realizability checks are independent of each other, the entire problem too is NP-complete. □

Theorem 4.6.7 *Given a formula F and a set M of assertions, deciding whether $M \models_l F$ is coNP-hard.*

Proof: Let $F = *^k \bot$. According to Theorem 4.4.21, $M \models_l \bot$ iff $*^k$ is cwr (which is always the case, as we already know), and $M_{\langle\gamma\rangle} \models_l \bot$ (which is never the case) for every rgi γ of $*^k$ in M. So the only way to make it true is by ensuring $*^k$ has no rgi in M. In other words, $M \not\models_l \bot$ iff $*^k$ is realizable, a problem we already showed NP-hard in Proposition 4.6.4. □
The other direction is more interesting. As usual, whenever $M \not\models_l F$ we must be able to guess a "witness" of polynomial size and then prove in polynomial time, given this information, that $M \not\models_l F$. Let us specify what we mean by "witness":

Definition 4.6.8 *Given a set of assertions M, a* counterinstance against $M \models_l F$ *(or simply:* against F*, if M is understood from the context) is a* nonempty *set M^- of assertions with ground labels, where:*

(1) *For every assertion $\gamma\ \top \in M^-$, γ is not a realized ground label in M;*

(2) *For every assertion $\gamma\ l \in M^-$ (where l is a literal), either γ is not a realized ground label in M, or for any assertion of the form $\sigma\ l \in M$, γ is not a realized ground instance of σ;*

(3) *If $F = a$ is atomic but $a \neq \bot$, then M^- contains $\varepsilon\ a$;*

(4) *If $F = F_1 \wedge F_2$, then M^- is a counterinstance against at least one of F_1 and F_2.*

(5) *If $F = F_1 \vee F_2$, then M^- is a counterinstance against both of F_1 and F_2.*

(6) *If $F = c\,F'$, then either c exists in M^- but not in M, or $(M^-)_{\langle c\rangle}$ is a counterinstance against $M_{\langle c\rangle} \models_l F'$.*

(7) *If $F = *\,F'$, then some c exists in M so that $(M^-)_{\langle c\rangle}$ is a counterinstance against $M_{\langle c\rangle} \models_l F'$.*

We call a counterinstance lean, *if each occurrence of a propositional atom a in F corresponds to at most one assertion $\gamma\ a$ or $\gamma\ \top$ in M^- and every assertion in M^- is thus accounted for. This is to say, there is a one-to-one, right-total (but not necessarily left-total) relation from occurrences of propositional atoms in F to elements of M^-.*

Example 4.6.9 We have $M = \{1{*}\,p, {*}1\,\neg q\} \not\models_l 1(1\,q \vee 2\,p)$. A lean counterinstance against M is $M^- = \{11\,q, 12\,p\}$. We obtain it by "reading off" one assertion from each of the alternatives in F, so that each assertion is not entailed by M. The first assertion is not entailed since q must be assigned *false* in instance 11; the second assertion is not entailed because 12 is not realized in M.

Intuitively, M^- specifies instances (or worlds of $K(M)$) which must be realized, given the requirements stated in F, the labels used in M, and the variable assignments which must hold in these instances; conditions (1) and (2) state that these stipulated instances either do not exist in M, or the variable assignments in these instances are not entailed by M. No matter how we branch on disjunctions in F, we find counterinstances within M^- for each alternative. Although counterinstances look somewhat akin to models, they should really not be considered as such; in particular they may contain clashes. The leanness condition guarantees that M^- is only by a factor of at most $d(F)$ larger than F, i.e. polynomial in F. We now show that the existence of a (lean) counterinstance is characteristic for $M \not\models_l F$.

Lemma 4.6.10 *Let M and F be as above, and M^- any counterinstance against F, then any superset of M^- which satisfies conditions (1) and (2) above is also a counterinstance against F.*

Proof: Condition (3) is certainly true for any superset of F, and conditions (4) through (7) are "robust" with respect to supersets. These facts can be easily shown by induction on the structure of F. □

Lemma 4.6.11 *Given M and F, if F has a counterinstance, then $M \not\models_l F$.*

Proof: We prove this indirectly. Assume $M \models_l F$. If $F = \bot$, then $M \not\models_l F$ for any M, so the assumption is contradictory from the start. If $F = a \neq \bot$, then because of condition (3) above, M^- must contain $\varepsilon\, a$. However, ε is ground and realized in any set, so condition (1) (in case $a = \top$) and the first alternative of condition (2) are violated. If $a \neq \top$, then M must also contain $\varepsilon\, a$ because $M \models_l a$, and ε as an rgi of itself violates the second alternative of condition (2). Now suppose this lemma has been shown for any subformula of F (and any set of assertions M). We study the different cases:

- $F = F_1 \wedge F_2$. Then according to our assumption, $M \models_l F_1$ and $M \models_l F_2$. By virtue of condition (4), M^- is a counterinstance against at least one of these, and the induction hypothesis yields the contradictory statement $M \not\models_l F_i$ for $i = 1$ or $i = 2$.

- $F = F_1 \vee F_2$. Then according to our assumption, $M \models_l F_1$ or $M \models_l F_2$. But condition (5) states that M^- is a counterinstance against both F_1 and F_2, and the induction hypothesis implies $M \not\models_l F_1$ and $M \not\models_l F_2$, which contradicts the assumption.

- $F = c\, F'$. Then c must exist in M (so the first alternative in condition (6) cannot be met), and $M_{\langle c \rangle} \models_l F'$. But the second alternative in condition (6) warrants the existence of a counterinstance against $M_{\langle c \rangle} \models_l F'$, which by the induction hypothesis leads to the contradictory result $M \not\models_l F'$.

- $F = * F'$. In condition (7), it is claimed that some c exists in M so that $(M^-)_{\langle c \rangle}$ is a counterinstance against $M_{\langle c \rangle} \models_l F'$; together with the induction hypothesis we infer $M_{\langle c \rangle} \not\models_l F'$. This is a contradiction, for if $M \models_l * F'$ and c exists in M, then $M_{\langle c \rangle} \models_l F'$.

The proof of the lemma follows by induction on the formula F. □

Lemma 4.6.12 *If $M \not\models_l F$, then there exists a* lean *counterinstance against F.*

Proof: First, if $F = a$ is atomic, then we must find a counterinstance with at most (in fact: exactly) one atom. Since $M \models_l \top$ for every M, we cannot have $a = \top$; for any other atomic formula we simply set $M^- = \{\varepsilon\, a\}$, so (3) is clearly satisfied and M^- is lean. In case $a = \bot$, neither condition (1) nor (2) needs to be verified in M'. For literals, we must establish condition (2); we do so by showing its second half. Notice that the only literal which has ε as an rgi is ε itself. But $\varepsilon\, a \notin M$, for otherwise we would have $M \models_l F$, contradicting our assumption.

As usual, we suppose the proposition holds for any subformula of F and show that it also holds for F. First, if $F = F_1 \wedge F_2$, then M does not verify one of F_1 and F_2; using the induction hypothesis, we get a lean counterinstance M^- for this conjunct, which is sufficient to show (4). Being a counterinstance, M^- satisfies conditions (1) and (2) with respect to one of the conjuncts; but since M and M^- both remain unchanged, both conditions hold true for F. To demonstrate that M^- remains lean, we retain the correspondence between atoms in the one conjunct and assertions in M^- as they were, and leave the atoms in the other conjunct unrelated to assertions in M^-. Clearly no more than one assertion corresponds to any propositional atom in F.

Secondly, consider $F = F_1 \vee F_2$. If $M \not\models_l F$, then neither F_1 nor F_2 are verified by M, so by the induction hypothesis they both have counterinstances M_1^- and M_2^- respectively. We

now take $M^- = M_1^- \cup M_2^-$. Conditions (1) and (2) hold for every assertion in both M_1^- and M_2^-, so they surely hold in $M_1^- \cup M_2^-$. (M remains unchanged.) Since M^- is also a superset of each of these, Lemma 4.6.10 applies, showing that M^- is a counterinstance against both F_1 and F_2. This establishes condition (5) for F. Finally, we derive the leanness of M^- from that of M_1^- and M_2^- by relating the atoms in the subformula F_1 to the assertions from M_1^-, and the atoms in F_2 corresponding to assertions from $M_2^- - M_1^-$ to those assertions. Atoms in F_2 corresponding to assertions from $M_2^- \cap M_1^-$ remain unrelated in order to guarantee that the relation remains one-to-one.

Thirdly, let $F = c\,F'$. Then $M \not\models_l F$ either because c does not exist in M, or because $M_{\langle c \rangle} \not\models_l F'$. In the first case, $M^- = \{c\,\top\}$ is a counterinstance. (Condition (1) is satisfied as c is not an rgl in M, and (6) is satisfied because c does exist in M^-.) This counterinstance is obviously lean—simply relate any of the atoms in F' to $c\,\top$. In the second case, the induction hypothesis warrants a lean counterinstance M'^- against $M_{\langle c \rangle} \models_l F'$. We claim that $M^- = cM'^-$ is a counterinstance: It is obvious that $(M^-)_{\langle c \rangle} = M'^-$, so M^- satisfies condition (6). Conditions (1) and (2) follow from their counterparts for M'^-, because a label $c\gamma$ from M^- is realized in M iff the corresponding label γ in M'^- is realized in $M_{\langle c \rangle}$, and $c\gamma$ is an rgi of some $c\sigma$ or $*\sigma$ in M iff γ is an rgi of σ in $M_{\langle \sigma \rangle}$ (see Lemma4.4.7, parts (3)-(5); we assumed that c exists in M). Finally we derive the leanness of M^- from that of M'^- by relating each atom in F' to assertion $c\gamma\,a$ whenever it is related to assertion $\gamma\,a$ in M'^-. (Note that the atoms of F and F' are identical.)

Finally take $F = *\,F'$. Then some constant c must exist in M so that $M_{\langle c \rangle} \not\models_l F'$. The induction hypothesis provides us with a lean counterinstance M'^- against $M_{\langle c \rangle} \models_l F'$, and we construct $M^- = cM'^-$. In the same way as we did in the previous case, we show that M^- satisfies conditions (7), (1), and (2), and that M^- is lean.

We have completed our discussion of all cases; thus we obtain the proof by induction on the formula F. Note that all sets we have claimed to be counterinstances are nonempty. □

Lemma 4.6.13 *For any set S of n assertions with ground labels of maximal depth d, $S_{\langle \gamma \rangle}$ is empty for all but at most $nd + 1$ ground labels γ.*

Proof: The set $S_{\langle \gamma \rangle}$ is empty, unless at least one of the assertions in S has a label of which γ is a prefix. There are at most d distinct nonempty prefixes per label in S, plus the ubiquitous ε. □

Theorem 4.6.14 *Given a formula F and a set M of assertions, deciding whether $M \not\models_l F$ is in coNP and hence coNP-complete.*

Proof: We have provided a property characteristic for $M \not\models_l F$, namely the existence of a nonempty, lean (i.e. polynomial in the size of F) counterinstance. To prove $M \not\models_l F$, we nondeterministically guess a hypothetical lean counterinstance M^-. Now we must be able to verify our claim in polynomial time. First we notice that for each subformula of F, only one of conditions (3) through (7) must be checked, and only once. Condition (3) involves a simple membership check, whereas (4) and (5) are simple recursive checks. Condition (6) involves checking whether c exists in M, which is also a membership check, and then (if necessary) computing $M_{\langle c \rangle}$ and $(M^-)_{\langle c \rangle}$. In condition (7), we may have to test several candidates for c. However, empty sets $(M^-)_{\langle c \rangle}$ do not qualify as counterinstances, so we recurse only when $(M^-)_{\langle c \rangle}$ is nonempty, or equivalently c exists in M^-. Over the course of the entire verification, each step can be regarded as a check whether some nonempty $M^-_{\langle \gamma \rangle}$ is a counterinstance against $M_{\langle \gamma \rangle} \not\models_l F'$ for some subformula F'. Lemma 4.6.13 states that there are at most $|M^-|\ d + 1$ distinct nonempty sets $(M^-)_{\langle \gamma \rangle}$ (wlog d can be chosen as the depth of M), and there are at most $|F|$ subformulas of F. So as long as we do not unnecessarily repeat identical steps, the total number of steps is polynomial, and each step requires only polynomial effort in terms of $|M|$. Finally, conditions (1) and (2) involve checking whether a ground label is realized, as well as instance checks; the former involves $O(|M|)$ instance checks (see Proposition 4.6.1), each of which can be performed in $O(d)$ time. In total, verifying whether M^- is a counterinstance requires polynomial time in M and F. (Remember that $|M^-| = O(|F|)$ since M^- is lean.) □

Corollary 4.6.15 *Model checking in $\mathcal{LBL}$ is coNP-complete.*

Proof: To verify $M \models_l F$, we first need to prove $M \not\models_l F$ (which we have shown in Theorem 4.6.14 to be coNP-complete) and then show that M is clash-free (which by the converse of Corollary 4.6.6 is also coNP-complete). These two steps are independent. □

The coNP-result for model checking sounds discouraging and may lead one to prematurely discard the idea of doing model checking in $\mathcal{LBL}$; after all, model checking in $\mathbf{K_{NNF}}$ is only polynomial *in the size of the Kripke model.* However, we already mentioned that labelled models may be a lot more concise than equivalent Kripke models (a fact we will substantiate and characterize more formally in the next section). In terms of this much smaller size, it

should come as no surprise that model checking is harder in the worst case. We do want to ensure, of course, that model checking in $\mathcal{LBL}$ is not harder in *absolute terms* than model checking in $\mathbf{K_{NNF}}$; this is easily confirmed:

Proposition 4.6.16 *Given an $\mathcal{LBL}$-formula F and a set of assertions M, checking $M \models_l F$ according to Definition 4.3.1 involves at most one recursive check $M_{\langle\gamma\rangle} \models_l F'$ for each combination of $\gamma \in I(M)$ and subformula F' of F, treating multiple occurrences of subformulas as distinct, and no such checks for any non-realizable γ or any non-subformula F'.*

Proof: This is shown by induction simultaneously over γ and the formula structure of F. Think of Definition 4.3.1 as a recursive procedure which is called with M and F as its parameters and returns *true* or *false*. With the one-time initial call $M \models_l F$ serving as the base case, we assert for our induction hypothesis that the recursive call $M_{\langle\gamma\rangle} \models_l F'$ is executed only once for any given (occurrence of) F' and any γ, and that γ must be an rgl in M and F' a subformula of F. Let us go through the cases in Definition 4.3.1. If F' is a literal, then the recursion ends, and the procedure returns *true* or *false*. If F' is a propositional formula, a number of recursive calls will be performed, at the end of which we have one recursive call for each atom in an atomic cover for F', with $M_{\langle\gamma\rangle}$ and the respective atom as parameters; each of these atoms is a distinct subformula of F'. (The number of intermediate calls does not exceed the number of intermediate propositional subformulas of F'.) Next, let $F' = c\,F''$. If c does not exist in $M_{\langle\gamma\rangle}$, then we immediately return *false*. Otherwise, we recursively verify $M_{\langle\gamma c\rangle} \models_l F''$. The induction hypothesis assured that γ is an rgi of M; by Lemma 4.4.7, part (7), c exists (i.e. is realizable) in $M_{\langle\gamma\rangle}$ iff γc is realizable in M. (Remember that "realizable" and "realized" coincide for ground labels.) Thus we are guaranteed that $M_{\langle\gamma c\rangle} \models_l F''$ is called only when γc is an rgl. Finally, let $F' = *\,F''$. Then the recursive call $M_{\langle\gamma c\rangle} \models_l F''$ is made once for each distinct c existing in $M_{\langle\gamma\rangle}$. The same argument as before asserts that γc is an rgl of M. This completes the case distinction, and the proof follows by induction on γ and F. □

This proof demonstrates an application of Definition 4.3.1 which is smart in one sense: every subformula F' of F is evaluated at most once in each realized ground label γ of M, and the result of $M_{\langle\gamma\rangle} \models_l F'$ is cached for later use. But the evaluation is naïve in another sense: Suppose F is a conjunction, one conjunct of which is $*^k p$. If M also contains $*^k p$, we know immediately that $M \models_l *^k p$. Definition 4.3.1 however forces us to evaluate $M_{\langle\gamma\rangle} \models_l p$ over

every realized ground instance of $*^k$, of which there may be exponentially many. For this reason, the evaluation is not very efficient. But instead of trying to find more efficient ways to evaluate $M \models_l F$ now, we defer this issue to the next chapter, where a more powerful semantic relation $\models_s$ is used to decide formulas like $M \models_s *^k p$ in one step.

Corollary 4.6.17 *Given an $\mathcal{LBL}$-formula F and a set of assertions M, let W be the set of worlds in $K(M)$, the Kripke model obtained from M by Algorithm 4.5.4[5]. Clash-freeness of M can be decided in $O(|M|^2|W|)$ steps, and $M \models_l F$ can be decided in $O((|M| + |F|)|W|)$ steps.*

Proof: We recall from Proposition 4.5.6 that W is equal to $I(M)$. Moreover, the number of subformulas in F is bounded by the size of F. Therefore, the statement about $M \models_l F$ follows immediately from Proposition 4.6.16. (We allow $|M| \times |W|$ steps for computing $M_{\langle \gamma c \rangle}$ from $M_{\langle \gamma \rangle}$ for each $\gamma \in W$, which is reasonable since $M_{\langle \gamma \rangle}$ can never be larger than M.) For the second part, consider any two assertions from M. If any of them contains $\sigma\ \bot$, check whether σ is realizable. If they contain complementary assertions $\sigma_1\ p$ and $\sigma_2\ \neg p$, compute (σ_1, σ_2), and if it exists, check whether this mgu is realizable. To check realizability of σ, it suffices to test every rgl from $I(M)$ and see whether it instantiates σ. This yields the above worst-case estimate. Needless to say, this method of finding an rgi is very crude and could be improved by using efficient storage and lookup methods for candidate rgl in $I(M)$. □

Let us discuss this complexity result and its implications for model checking in practice. A typical context would be the following: We are given a set Φ of $\mathbf{K_{NNF}}$-formulas. A theorem prover converts Φ into a set of $\mathcal{LBL}$-formulas S, finds a model M for S, and returns it. How can we check that M is indeed a model? There are two possible scenarios: Either we check M on the same machine which returned the result, and the labelling S is still stored or otherwise known to us; or we do not know the labelling S.

In the first scenario, we can choose whether to check $M \models_l S$ directly, or rather compute $K(M)$ and then check $K(M) \models_k \Phi$. In the next section we will see that $K(M)$ may be exponentially larger than M, so the second route may not always be available to us, given a limited amount of storage space. But suppose we can represent $K(M)$. As we have seen in Theorem 3.5.4, checking $K(M) \models_k \Phi$ takes $O(|\Phi| \times |K(M)|)$ steps; on the other hand, the

[5] *We formally required that M be already clash-free in order to run the algorithm. However, here we are only interested in the set of worlds W, and the algorithm can still be run if M contains clashes, if we skip the computation of valuation functions.*

complexity of checking $M \models_l S$ as stated in Corollary 4.6.17 is comparable up to a factor $|M|^2$. Allowing a polynomial factor in $|M|$ for processing the model while checking it seems reasonable. Also, if M is not larger than $K(M)$, then checking M is not significantly more expensive than checking the Kripke model represented by M. But is our assumption "M is not larger than $K(M)$" reasonable? The fact of the matter is that M can be arbitrarily inflated by adding assertions such as $\sigma\, a$ with an unrealizable label σ. These assertions can be crafted so as to make it non-obvious whether any particular assertion has a realizable label or not. However, once we have fully determined $I(M)$, we can easily find all assertions whose labels are not realizable and delete them. Determining $I(M)$ takes no greater effort than determining $K(M)$ along the alternate route, and we will certainly have to go linearly through all assertions in M while doing this, so another linear scan through M in order to delete redundant assertions is not very costly. In our practical context above this will hardly be necessary anyway, for our model M arises as the answer of a theorem prover to the question whether S (or Φ) is satisfiable. We already showed in Corollary 4.5.14 that every strict labelling of Φ has a model M not essentially larger than the smallest strict Kripke model for Φ. To require of our prover to return a model within these—still generous—size bounds is not unreasonable. Since $K(M)$ itself is strict, we are then guaranteed that M is no larger than $K(M)$.

Now what if we only know Φ and M but not the labelling S? Of course we can still compute $K(M)$ and check $K(M) \models_k \Phi$. Notice however that this may give rise to false positives: We will see in Section 4.8 that some sets Φ have non-strict models K which are smaller than any models arising from any labelling of Φ. In other words, no labelling of Φ has a model M so that $K = K(M)$. Yet it is easy to find *some* model M' so that $K = K(M')$. While K is indeed a Kripke model for Φ, a given candidate M' may not actually be a model for any labelling of Φ. Admittedly we may not care much about this in practice, if all we are interested in is information on *some* satisfying model for Φ. But there remains the problem that $K(M)$ might be prohibitively large and not computable in practice.

So can we reconstruct S and check $M \models_l S$ in a reasonable amount of time, provided we know M and Φ? The answer depends on how we measure the complexity. Expressed in terms of Φ and M, the problem becomes Σ_2^p-complete. We will not present a formal proof here, for lack of relevance to the rest of this work. But intuitively, this is true because upon "guessing" a labelling S of Φ (which has a polynomial-size representation in $|\Phi|$: specify a constant $c \in D$ for each $\Diamond$-operator occurring in Φ), we are left with the coNP-complete

problem of checking $M \models_l S$. But one can also show that satisfiability of $\mathbf{QBF}_2$ (see next section for a definition) can be translated polynomially into the above problem.
In terms of Φ, M and W, where W is the set of worlds of $K(M)$, reconstructing S and checking $M \models_l S$ can be done in $O(|\Phi||M||W|)$ steps. Again we omit a proof for lack of relevance to our work. The bound crucially depends on the labelling information contained in M. Recall that the constants assigned to each $\Diamond$-operator denote R-arcs in $K(M)$. If M is given, the labelling of all arcs is known; in order to label a $\Diamond$-operator in a subformula $\Diamond\ \pi_0$, we just need to find a successor in which π_0 holds; if the arc to this successor is labelled c, then c can be used to label this particular $\Diamond$-operator.

4.7 The Complexity of Reasoning with Labelled Formulas

Arguably the most common characterization of problems in the class PSPACE is by translating them into quantified Boolean formulas (QBF). This is no different with the modal logic **K**; several satisfiability-preserving translation methods have been proposed [61, 90], and translations of random QBF have been used as hard benchmark problems for automated reasoners in **K**[72, 74].
Complexity results about reasoning with labelled formulas can be aptly stated in terms of QBF as well. To facilitate this, we must introduce a satisfiability-preserving translation method from QBF into $\mathcal{LBL}$. We could simply translate a given QBF into **K** (the resulting formula from using any of the methods proposed in [61, 90] is in NNF), and then quote Theorem 4.5.11 to translate the $\mathbf{K}_{\mathbf{NNF}}$-formula into an equisatisfiable $\mathcal{LBL}$-formula. Instead, a direct translation suits our purposes better.
We remark that such a translation is not only of academic interest. Given an efficient method for translating QBF-formulas into labelled formulas, the model-finding algorithm we are presenting in this work will provide an alternative for solving QBF problems, an area of active current research [32, 62, 10].

Definition 4.7.1 *The class* $\mathbf{QBF}_k$ *of* QBF-formulas in standard form *is defined as the set of all formulas of the form*

$$\varphi_0 = Q_1 \ldots Q_n\, \varphi(p_1, \ldots, p_n), \tag{4.11}$$

where $\varphi(p_1, \ldots, p_n)$ *is a* $\mathbf{PL}_{\mathbf{NNF}}$*-formula in the variables* $p_1, \ldots, p_n$, $n \in \mathbb{N}$, *and* $Q_j = \exists p_j$ *or* $Q_j = \forall p_j$, *whereas* $Q_1 = \exists p_1$, *and the sequence of quantifiers alternates between* $\exists$ *and*

$\forall$ *exactly* $k-1$ *times. The class* **QBF** *is simply the union of all* $\mathbf{QBF}_k$*, i.e. the set of all Quantified Boolean Formulas with no restriction on quantifier alternation. A QBF is decided valid or not valid, as follows:*
Let $\varphi_j(p_1,\ldots,p_j) = Q_{j+1}\ldots Q_n\varphi(p_1,\ldots,p_n)$*. (Thus,* $\varphi_n(p_1,\ldots,p_n) = \varphi(p_1,\ldots,p_n)$*;* $\varphi_0()$ *is consistent with* φ_0 *as given in (4.11), and* $\varphi_j(p_1,\ldots,p_j) = Q_{j+1}\varphi_{j+1}(p_1,\ldots,p_{j+1})$ *for all* $i = 0,\ldots,n-1$*.) Then a valuation on* $p_1,\ldots p_j$ satisfies $\varphi_j(p_1,\ldots,p_j)$ *iff it also satisfies*

- $\varphi_{j+1}(p_1,\ldots,p_{j+1})|_{p_{j+1}=\mathit{true}}$ *and* $\varphi_{j+1}(p_1,\ldots,p_{j+1})|_{p_{j+1}=\mathit{false}}$*, if* $Q_j + 1 = \forall p_j$*,*

- $\varphi_{j+1}(p_1,\ldots,p_{j+1})|_{p_{j+1}=\mathit{true}}$ *or* $\varphi_{j+1}(p_1,\ldots,p_{j+1})|_{p_{j+1}=\mathit{false}}$*, if* $Q_j + 1 = \exists p_j$*.*

Then $\varphi_0()$ *is valid if the empty valuation satisfies it.*
The complexity class Σ_k^p *is defined as the class of all problems which can be translated in polynomial time into a validity problem in* $\mathbf{QBF}_k$*. We define* PSPACE $= \bigcup_{k=1}^{\infty}\Sigma_k^p$*. A problem is called* Σ_k^p-hard *if any formula* φ_0 *in* $\mathbf{QBF}_k$ *can be translated in polynomial time into an instance of the given problem, so that* φ_0 *is valid iff the corresponding problem instance has a solution. A problem is called* Σ_k^p-complete *if it is in* Σ_k^p *and* Σ_k^p*-complete.* PSPACE-completeness *and* PSPACE-hardness *are defined analogously.*

A more intuitive way of describing the validity of a QBF is by picturing the quantifiers Q_j as first-order quantifiers over the truth values of p_j. Thus, $\forall p_j\,\varphi$ says that φ must be valid for both truth values of p_j, and $\exists p_j\,\varphi$ expresses that φ is valid for some truth value of p_j. If all quantifiers are existential, then the QBF is valid iff there is some truth assignment for $p_1,\ldots p_n$ so that φ is satisfied, which is to say φ is satisfiable. This shows that validity in $\mathbf{QBF}_1$ is equivalent to the SAT problem, which means $\mathbf{QBF}_1 \cong \mathbf{PL_{NNF}}$ and hence $\Sigma_1^p = NP$.

Example 4.7.2 The formula

$$\exists p_1 \forall p_2 \forall p_3 \exists p_4\, (p_1 \wedge (p_2 \vee p_3 \vee p_4))$$

is in $\mathbf{QBF}_3$, as there are two alternations of quantifiers ($\exists$ to $\forall$ and again to $\exists$). This formula is valid, as we can informally see: Any valuation in which p_1 and p_4 are assigned *true* satisfies $p_1 \wedge (p_2 \vee p_3 \vee p_4)$, no matter how p_2 and p_3 are assigned.

It is easy to show that the dual of a QBF takes a form similar to a QBF:

$$\overline{\varphi}_0 = \overline{Q}_1 \ldots \overline{Q}_n \overline{\varphi}(p_1,\ldots,p_n),$$

where $\overline{Q}_j = \exists p_j$ if $Q_j = \forall p_j$ and vice versa, and $\overline{\varphi}$ is the propositional dual of φ. It is easy to show that $\overline{\varphi}_0$ is valid iff φ_0 is invalid. Thus, a QBF and its dual are complementary. We denote the class of complements of $\mathbf{QBF}_k$ by $\overline{\mathbf{QBF}}_k$; the associated complexity classes are commonly denoted Π_k^p. It is easy to show that Π_k^p and Σ_k^p are contained in *both* Π_{k+1}^p and Σ_{k+1}^p. For detailed treatments on QBF and the polynomial hierarchy, we refer to [92].
We now demonstrate that the satisfiability problem in $\mathcal{LBL}$ is PSPACE-hard, by constructing a translation from **QBF** into $\mathcal{LBL}$. For every variable p_j, $j = 1, \ldots, n$, we designate three constants from D which we label $c_\top^{(j)}$, $c_\bot^{(j)}$, and $c^{(j)}$. Note that the (j) is clear from the context and the position in a given label, so we will henceforth omit it and simply write $c_\top$, $c_\bot$, and c.

Definition 4.7.3 *Given a QBF $\varphi_0 = Q_1 \ldots Q_n \, \varphi(p_1, \ldots, p_n)$ as before, we define the standard translation of φ_0 into $\mathcal{LBL}$ as $\lambda(\varphi_0) = \{G_1, \ldots, G_n, G_0\}$, where*

$$\begin{aligned} G_i &= \begin{cases} *^{i-1}\,(c_\top *^{n-i} p_i \wedge c_\bot *^{n-i} \neg p_i) & \text{for } Q_i = \forall p_i \\ *^{i-1} c\,(*^{n-i} p_i \vee *^{n-i} \neg p_i) & \text{for } Q_i = \exists p_i \end{cases} \\ G_0 &= *^n \varphi(p_1, \ldots, p_n). \end{aligned}$$

It is easy to see that this translation is polynomial. Before we prove it satisfiability-preserving, though, let us state a few useful facts on sets of labelled formulas. Recall the shortcut notation $\sigma\, S$ for $\{\sigma\sigma'\, F : \sigma'\, F \in S\}$.

Lemma 4.7.4 *Given three nonempty sets S, S' and S'' of labelled formulas, two formulas F' and F'', and distinct constants $c, c', c'' \in D$:*

(1) *$\{F' \vee F''\} \cup S$ is satisfiable iff $\{F'\} \cup S$ or $\{F''\} \cup S$ is satisfiable.*

(2) *$\{F\} \cup S$ is satisfiable iff $\{c\, F\} \cup *\, S$ is satisfiable.*

(3) *$S \cup S'$ and $S \cup S''$ are both satisfiable iff $* S \cup c'\, S' \cup c''\, S''$ is satisfiable.*

Proof: This proof will be constructive, in that we derive models for the right-hand side from models of the left-hand side of (2) and (3).
A model of a set of formulas satisfies all its elements. Hence, if $M \models_l \{F' \vee F''\} \cup S$, then M satisfies all formulas in S, and it satisfies F' or F''. Consequently, M satisfies one of $\{F'\} \cup S$ and $\{F''\} \cup S$. The converse of (1) follows just by the same argument in reverse.

Now let M be a model for the set $\{c\,F\} \cup * S$ on the right-hand side of (2). Since c exists in this set, Corollary 4.4.11 can be applied, showing that $M_{\langle c \rangle}$ satisfies $(\{c\,F\} \cup * S)_{\langle c \rangle}$. But the latter is exactly the set $\{F\} \cup S$ on the left-hand side. Conversely, given a model M' for S, we create $\widehat{M}$ by prefixing each assertion in M' with either c or $*$, whereas at least one assertion is prefixed with c, to ensure c is the one and only constant which exists in $\widehat{M}$. We clearly have $\widehat{M}_{\langle c \rangle} = M'$, and $\widehat{M}$ satisfies $c\,F$ (because $\widehat{M}_{\langle c \rangle} \models_l F$), as well as any formula $* F_0 \in * S$ (because $\widehat{M}_{\langle c \rangle} \models_l F_0$ and c is the only constant existing in $\widehat{M}$). Thus $\widehat{M}$ is a model for $\{c\,F\} \cup * S$. Note that if M is clash-free, then $M_{\langle c \rangle}$ is also clash-free (by Proposition 4.4.10), and if M' is clash-free, then so is $\widehat{M}$ (which is easy to see because two labels in $\widehat{M}$ unify iff their counterparts in M' unify.

To show (3), let $\overline{M}$ be a model for $T = * S \cup c'\,S' \cup c''\,S''$. Now both c' and c'' exist in T, so we can apply Corollary 4.4.11 twice, showing that $\overline{M}_{\langle c' \rangle} \models_l T_{\langle c' \rangle}$ and $\overline{M}_{\langle c'' \rangle} \models_l T_{\langle c'' \rangle}$. But $T_{\langle c' \rangle}$ and $T_{\langle c'' \rangle}$ are exactly the sets $S \cup S'$ and $S \cup S''$ on the left-hand side. To show the converse, take two *non-empty* models M' for $S \cup S'$ and M'' for $S \cup S''$, and let M be the set of the following assertions:

- $*\sigma\ a$ for any $\sigma\ a$ which occurs in both M' and M'',
- $c'\sigma\ a$ for any $\sigma\ a$ which occurs only in M',
- $c''\sigma\ a$ for any $\sigma\ a$ which occurs only in M'',
- $c'\ \top$ and/or $c''\ \top$, if c' or c'' does not precede any assertion taken so far.

Let us first observe that c' and c'' exist in M and that $M_{\langle c' \rangle} = M'$ and $M_{\langle c'' \rangle} = M''$, except perhaps for an additional, irrelevant $\varepsilon\ \top$. These models verify all formulas in S' and S'', respectively, and they both verify all formulas in S. No other constant exists in M, so we have shown that M verifies (all formulas in) $* S$, $c'\ S'$, and $c''\ S''$. Next we observe that whenever $x\sigma$ is realizable in M (where x is one of c',c'',$*$), σ is realizable in one of M' or M''. To see this, consider Lemma 4.4.7, parts (4) and (5), which say that $I(c'\sigma, M) = c'I(\sigma, M')$, $I(c''\sigma, M) = c''I(\sigma, M'')$, and $I(*\sigma, M) = c'I(\sigma, M') \cup c''I(\sigma, M'')$; so in order for $I(x\sigma, M)$ to be nonempty, either $I(\sigma, M')$ or $I(\sigma, M'')$ must be nonempty. We use this result to show that M is clash-free. Suppose M contains some label $c'\sigma\ \bot$ so that $c'\sigma$ is realizable in M, then we would have $\sigma\ \bot \in M'$ and σ is realizable in M', contradicting the fact that M' as a model is clash-free. If the clash is $c''\sigma\ \bot$, we get the same contradiction in M'', whereas if it is of the form $*\sigma\ \bot$, then $\sigma\ \bot$ is in both M' and M'' and σ is realizable in at least

one of them, which also results in a contradiction. Now suppose the clash is of the form $x_1\sigma_1\, p$, $x_2\sigma_2\, \neg p$. If $x_1 = x_2 = c'$, then $\sigma_1\, p$ and $\sigma_2\, \neg p$ come from the same set M', and since the mgu $(x_1\sigma_1, x_2\sigma_2)$ is supposedly realizable in M, (σ_1, σ_2) is realizable in M'. This is a contradiction since M' is clash-free. The same contradiction arises in M'' if $x_1 = x_2 = c''$. Otherwise at least one of x_1 and x_2 must be $*$, say x_1. But then $\sigma_1\, p$ occurs in both M' and M'' and would clash with $\sigma_2\, \neg p$ in at least one of them, which again is a contradiction. Therefore, M must be clash-free, and we have proved the converse of (3). □

In Definition 4.7.1 we defined the validity of a QBF recursively; likewise, most of our results around satisfiability so far are of a recursive nature. By contrast, Definition 4.7.3 introduces a closed-form translation of φ_0 into $\mathcal{LBL}$. A matching recursive definition, more suitable for our equivalence proofs, is readily given. We extend $\lambda(\cdot)$ by defining $\lambda(\varphi_j(p_1, \ldots, p_j)) = \{G_{j+1}^{(j)}, \ldots, G_n^{(j)}, G_0^{(j)}\}$, where

$$\begin{aligned} G_i^{(j)} &= \begin{cases} *^{i-j-1}\,(c_\top *^{n-i}\, p_i \wedge c_\bot *^{n-i}\, \neg p_i) & \text{for } Q_i = \forall p_i \\ *^{i-j-1} c\,(*^{n-i}\, p_i \vee *^{n-i}\, \neg p_i) & \text{for } Q_i = \exists p_i \end{cases} \\ G_0^{(j)} &= *^{n-j}\, \varphi(p_1, \ldots, p_n). \end{aligned} \tag{4.12}$$

Proposition 4.7.5 *Given a propositional valuation V on $p_1, \ldots p_j$, $j \leq n$, define $l_i = p_i$ if $V(p_i) = true$, and $l_i = \neg p_i$ if $V(p_i) = false$, for all $i \leq j$. Then $\varphi_j(p_1, \ldots, p_j)$ is valid under V iff $\lambda(\varphi_j(p_1, \ldots, p_j)) \cup \{*^{n-j}\, l_i : i \leq j\}$ is satisfiable.*

Proof: We prove this recursively, beginning with $j = n$. In this case, $\lambda(\varphi_n(p_1, \ldots, p_n)) = \{\varphi_n(p_1, \ldots, p_n)\}$, and the set in question is $\{\varphi_n(p_1, \ldots, p_n), l_1, \ldots, l_n\}$. It is satisfiable, by Corollary 2.2.8, iff it has a clash-free cover of propositional atoms. Now it is not hard to see that any such cover contains all the literals $l_1, \ldots, l_n$ and is clash-free if and only if it contains no other literals. In other words, $l_1, \ldots, l_n$ (and possibly $\top$) already contain a cover for $\varphi_n(p_1, \ldots, p_n)$, which is just another way of stating that $\varphi_n(p_1, \ldots, p_n)$ is satisfied under the valuation V (see Proposition 2.2.6). This establishes the base case.

Now suppose this proposition shown for $j + 1$, and let V be a given valuation on $p_1, \ldots p_j$. We write $V_\top$ and $V_\bot$ for the two extensions of V to $p_1, \ldots p_{j+1}$, with $V_\top(p_{j+1}) = true$ and $V_\bot(p_{j+1}) = false$ respectively. We need to distinguish two cases, depending on whether Q_{j+1} is universal or existential. First assume $\varphi_j(p_1, \ldots, p_j) = \exists p_{j+1}\, \varphi_{j+1}(p_1, \ldots, p_{j+1})$. According to Definition 4.7.1, $\varphi_j(p_1, \ldots, p_j)$ is valid under V iff $\varphi_{j+1}(p_1, \ldots, p_{j+1})$ is valid under one of $V_\top$ or $V_\bot$. By the hypothesis, this is the case iff $\lambda(\varphi_{j+1}(p_1, \ldots, p_{j+1})) \cup$

$\{*^{n-j-1} l_i : i \leq j+1\}$ is satisfiable, where l_{j+1} is either p_{j+1} (under $V_\top$) or $\neg p_{j+1}$ (under $V_\bot$). We write this set out in full, according to (4.12):

$$\begin{array}{ll} *^{i-j-2}(c_\top *^{n-i} p_i \wedge c_\bot *^{n-i} \neg p_i) & \text{whenever } Q_i = \forall p_i, i \geq j+2 \\ *^{i-j-2} c\,(*^{n-i} p_i \vee *^{n-i} \neg p_i) & \text{whenever } Q_i = \exists p_i, i \geq j+2 \\ *^{n-j-1}\varphi(p_1,\ldots,p_n) & \\ *^{n-j-1} l_i & i \leq j \\ *^{n-j-1} p_{j+1} \vee *^{n-j-1} \neg p_{j+1}. & \end{array} \tag{4.13}$$

(We obtained this system by applying statement (1) of Lemma 4.7.4: Thinking of the last line as a disjunction $F' \vee F''$ and of the remaining lines as S, the system is satisfiable iff $\{F'\} \cup S$ or $\{F''\} \cup S$ is satisfiable.) Now let us write out $\lambda(\varphi_j(p_1,\ldots,p_j)) \cup \{*^{n-j} l_i : i \leq j\}$ via (4.12) and compare:

$$\begin{array}{ll} *^{i-j-1}(c_\top *^{n-i} p_i \wedge c_\bot *^{n-i} \neg p_i) & Q_i = \forall p_i, i \geq j+2 \\ *^{i-j-1} c\,(*^{n-i} p_i \vee *^{n-i} \neg p_i) & Q_i = \exists p_i, i \geq j+2 \\ *^{n-j}\varphi(p_1,\ldots,p_n) & \\ *^{n-j} l_i & i \leq j \\ c(*^{n-j-1} p_{j+1} \vee *^{n-j-1} \neg p_{j+1}). & \end{array} \tag{4.14}$$

For clarity $G^{(j)}_{j+1}$ has been written out separately in the last line. We observe that each line in (4.13) is identical to the corresponding line in (4.14), with an extra $*$ prefixed in all but the last line, and c prefixed in the last line of (4.14). Thus system (4.13) matches the pattern $S \cup \{F\}$ and system (4.14) the pattern $\{c\,F \cup *\,S\}$. As shown in Lemma 4.7.4, part (2), these two sets are equisatisfiable.

Now consider the case where Q_{j+1} is universal. Definition 4.7.1 states that $\varphi_j(p_1,\ldots,p_j)$ is valid under V iff $\varphi_{j+1}(p_1,\ldots,p_{j+1})$ is valid under both $V_\top$ and $V_\bot$. Using the hypothesis, this is true iff $\lambda(\varphi_{j+1}(p_1,\ldots,p_{j+1})) \cup \{*^{n-j-1} l_i : i \leq j+1\}$ is satisfiable in both cases $l_{j+1} = p_{j+1}$ and $l_{j+1} = \neg p_{j+1}$. Again we write this out in full, pointing out that we have

two sets this time—one each for $l_{j+1} = p_{j+1}$ and $l_{j+1} = \neg p_{j+1}$:

$$\begin{array}{ll} *^{i-j-2}\,(c_\top *^{n-i}\, p_i \wedge c_\bot *^{n-i}\, \neg p_i) & Q_i = \forall p_i, i \geq j+2 \\ *^{i-j-2} c\,(*^{n-i}\, p_i \vee *^{n-i}\, \neg p_i) & Q_i = \exists p_i, i \geq j+2 \\ *^{n-j-1}\, \varphi(p_1, \ldots, p_n) & \\ *^{n-j-1}\, l_i & i \leq j \\ *^{n-j-1}\, l_{j+1} & l_{j+1} = p_{j+1} \text{ or } l_{j+1} = \neg p_{j+1} \text{ fixed.} \end{array} \tag{4.15}$$

And this is the set $\lambda(\varphi_j(p_1, \ldots, p_j)) \cup \{*^{n-j} l_i : i \leq j\}$:

$$\begin{array}{ll} *^{i-j-1}\,(c_\top *^{n-i}\, p_i \wedge c_\bot *^{n-i}\, \neg p_i) & Q_i = \forall p_i, i \geq j+2 \\ *^{i-j-1} c\,(*^{n-i}\, p_i \vee *^{n-i}\, \neg p_i) & Q_i = \exists p_i, i \geq j+2 \\ *^{n-j}\, \varphi(p_1, \ldots, p_n) & \\ *^{n-j}\, l_i & i \leq j \\ c_\top *^{n-j-1}\, p_{j+1} & \\ c_\bot *^{n-j-1}\, \neg p_{j+1}. & \end{array} \tag{4.16}$$

This time $G^{(j)}_{j+1}$ is a conjunction of two formulas; we have written it out in the last *two* lines. As before, we observe the analogy between these two systems: With $c' = c_\top$, $c'' = c_\bot$, $F' = \{*^{n-j-1} p_{j+1}\}$, $F'' = \{*^{n-j-1} \neg p_{j+1}\}$, and S being all but the last line in system 4.15, statement (3) of Lemma 4.7.4 applies, showing that system 4.16 is satisfiable iff (both instantiations of) system 4.15 is satisfiable.

We have thus derived the proposition for j from the assumption that it is valid for $j+1$. Since it is valid for n, it follows for all $j = 0, \ldots, n$ as claimed. □

Corollary 4.7.6 *A QBF $\varphi_0 = Q_1 \ldots Q_n\, \varphi(p_1, \ldots, p_n)$ is valid iff $\lambda(\varphi_0)$ is satisfiable.*

Proof: This is just Proposition 4.7.5, written out for $j = 0$. □

Corollary 4.7.7 *Satisfiability in $\mathcal{LBL}$ is PSPACE-hard.*

Proof: In λ we have a polynomial-size translation function mapping **QBF** into the satisfiability problem for labelled formulas, and Corollary 4.7.6 shows that this translation preserves solutions. □

Theorem 4.7.8 *Satisfiability in $\mathcal{LBL}$ is PSPACE-complete.*

Proof: We just showed PSPACE-hardness. To show that satisfiability is in PSPACE, we recall that the problem of **K**-satisfiability is in PSPACE. In the transformation κ from Definition 4.5.1 we have a translation function which maps any set of labelled formulas into an at most polynomially larger set of **K**-formulas, and we showed in Theorem 4.5.11 that this translation is satisfiability-preserving. Thus we have reduced satisfiability in $\mathcal{LBL}$ to a PSPACE-complete problem. □

Note that this theorem settles only the problem whether a model for a given $\mathcal{LBL}$-formula *exists*; we did not claim that models could be written out in polynomial space. We will now show that it is impossible to find polynomial-size models in general, but possible in a well-defined subclass of problems.

Proposition 4.7.9 *If S is the translation (via λ) of a QBF with m existential and $n-m$ universal quantifications, then $*^n$ has exactly 2^{n-m} distinct realized ground instances in S; these realized ground instances take all Boolean combinations of $c_\top$ and $c_\bot$ in the positions corresponding to the $\forall$ quantifiers, and c in all other positions.*

Proof: These rgis are spanned by the G_i, $i = 1, \ldots, n$, (see Definition 4.7.3) whose labels introduce constants in the ith position. If $I(*^{i-1}, S)$ has been determined (and has 2^j labels where j universal quantifiers have been "encountered" so far), then $I(*^i, S)$ is determined as follows: If $Q_i = \exists p_i$, then G_i has a label beginning with $*^{i-1}c$, and it is the only label in S with a constant in the ith position. So $I(*^i, S)$ consists of all ground labels from $I(*^{i-1}, S)$, with c appended. If $Q_i = \forall p_i$, then G_i introduces two constants in the ith position, and $I(*^i, S)$ consists of all ground labels from $I(*^{i-1}, S)$, with either $c_\top$ or $c_\bot$ appended. Hence, $I(*^i, S)$ consists of 2^{j+1} ground labels. The proposition follows from here by an easy induction proof. □

This proposition has one important consequence: If we translate a QBF with $n-m$ universal quantifiers via λ into an $\mathcal{LBL}$-formula, then any model for this formula must span all the above ground instances. If we translate this formula further via κ into a $\mathbf{K_{NNF}}$-formula, then by Corollary 4.5.15 a strict Kripke model for this translated formula must have $\Theta(2^{n-m})$ distinct worlds, one for each realized ground instance. In fact, we can show that any two different ground instances of $*^n$ feature incompatible variable assignments. (If, say, in position i one label has $c_\top$ and the other has $c_\bot$, then p_i must be assigned to *true* in the

former instance, and to *false* in the latter.) Therefore, *any* Kripke model must have $\Theta(2^{n-m})$ distinct worlds[6]. Since even in $\mathbf{QBF}_2$ the number of universal quantifiers is unbounded, Kripke models are definitely not a space-efficient way to witness a valid QBF. One might hold this to be an artifact of our two-stage translation. However, the common methods [61, 90] for translating QBF problems directly into satisfiability problems for **K** are very similar in nature, and the resulting formulas in **K** give rise to exponential-size models. (This is why translations of QBF formulas are used as benchmark problems for modal logics [74]—they tend to be hard[7].)

For labelled formulas we have a similar but slightly diminished negative result, in that the problem class is chosen *outside* $\mathbf{QBF}_2$:

Proposition 4.7.10 *There is a family of valid formulas in* $\overline{\mathbf{QBF}}_2$ *of size* $O(n)$ *whose translations via* λ *have only models of size* $\Theta(2^n)$.

Proof: Take the family of formulas:

$$\varphi_n = \forall p_1 \ldots \forall p_n \exists q_1 \ldots \exists q_n \, \varphi(p_1, \ldots, p_n, q_1, \ldots, q_n),$$

where $\varphi(p_1, \ldots, p_n, q_1, \ldots, q_n)$, written in set form, is:

$$\begin{aligned} p_1 &\leftrightarrow q_1 \\ p_2 \leftrightarrow q_1 &\leftrightarrow q_2 \\ &\ldots \\ p_n \leftrightarrow q_{n-1} &\leftrightarrow q_n. \end{aligned}$$

The $\leftrightarrow$ operator is used for convenience; each formula above can be replaced by a set of up to four disjunctions with up to three literals each. Using $\leftrightarrow$ better conveys the setup: Variable q_i is *true* iff an even number of variables out of $p_1, \ldots, p_n$ is *false*. Since suitable assignments for the q_i can be found for any variable assignment to $p_1, \ldots, p_n$, this QBF is valid. However, we will show that these assignments cannot be expressed concisely as a model in $\mathcal{LBL}$. The main idea of the proof is that q_n is assigned *true* whenever an even

[6] In fact, a model with *exactly* 2^{n-m} worlds can be found if we allow worlds to be accessible to themselves.

[7] This is also the reason why a theorem prover should implement optimization techniques which allow it to find non-strict Kripke models, as it may then be able to "break" the exponential-space barrier in some problem instances.

number of p_i is *true*. That means, in any two valuations which differ in only one of the variable assignments to the p_i, q_n has opposite truth assignments. This makes it impossible to use $*$ in any of the label positions corresponding to the p_i, but we have to use ground labels for each of the 2^n different combinations of variable assignments to the $p - i$.

By Proposition 4.7.9, $\lambda(\varphi_n)$ has 2^n ground instances of length $2n$, namely $x_1 \cdots x_n c \ldots c$, where each x_i takes one of the constants $c_\top$ or $c_\bot$. Assuming we have a model M for $\lambda(\varphi_n)$, let us explore its properties. As we know, all rgl of $\lambda(\varphi_n)$ must also be realized in M. Furthermore, M must satisfy all formulas in $\lambda(\varphi_n)$, particularly the formula $*^{2n}\, \varphi(p_1, \ldots, p_n, q_1, \ldots, q_n)$. We apply Theorem 4.4.21 to conclude that for every rgi γ of $*^{2n}$ we must have $M_{\langle\gamma\rangle} \models_l \varphi(p_1, \ldots, p_n, q_1, \ldots, q_n)$; since $\varphi(p_1, \ldots, p_n, q_1, \ldots, q_n)$ is propositional, this is the case iff $M_{\langle\gamma\rangle}$ provides a (partial) propositional truth assignment satisfying this formula. Finally, $\lambda(\varphi_n)$ contains formulas $*^{i-1} c_\top *^{2n-i}\, p_i$ and $*^{i-1} c_\bot *^{2n-i}\, \neg p_i$, for $i = 1, \ldots n$. Again by Theorem 4.4.21, we conclude that $M_{\langle\gamma\rangle}$ contains $\varepsilon\, p_i$ iff the ith position of γ is $c_\top$, and $\varepsilon\, \neg p_i$ iff this position is $c_\bot$.

Now look at the setup of $\varphi(p_1, \ldots, p_n, q_1, \ldots, q_n)$; to determine its truth value, all variables $p_1, \ldots, p_n, q_1, \ldots, q_n$ must be assigned. Hence, every propositional variable must be mentioned in some assertion in $M_{\langle\gamma\rangle}$. We focus particularly on q_n. Consider a fixed γ, wlog so that $M_{\langle\gamma\rangle}$ contains the assertion $\varepsilon\, q_n$. This assertion must correspond to an original assertion in M of the form $\sigma\, q_n$ with $\sigma \sqsubseteq \gamma$. Assume σ has $*$ in some position $i \leq n$. Hence σ has another ground instance γ' which differs from γ only in position i, where one of them has the constant $c_\top$ and the other $c_\bot$. Now since q_n was assumed to be *true* in $M_{\langle\gamma\rangle}$, an even number of p_i must be *false* in $M_{\langle\gamma\rangle}$. From the correspondence we have found above between the first n positions of γ and the assignments to the p_i, we conclude that an even number out of the first n positions in γ is $c_\bot$. Since γ' differs only in one position, it has an odd number of $c_\bot$ in its first n positions. By the same argument in reverse, an odd number of p_i is assigned *false* in $M_{\langle\gamma'\rangle}$, so q_n must be *false* also. Thus M must contain another assertion $\sigma'\, \neg q_n$, and γ' is an instance of both σ and σ'. But this means that M is not clash-free, contradicting the fact that M is a model. Hence we must drop our assumption that σ has $*$ in any of its first n positions.

This argument applies to any of the 2^n different ground instances γ. Each of these must give rise to a distinct $\sigma\, q_n$ or $\sigma\, \neg q_n$, so M must contain $\Theta(2^n)$ assertions mentioning q_n alone. □

As for proving a positive result, finding a large class of formulas which do offer a polynomial-size model, we face one difficulty: So far, we have only been concerned about the existence of a model, regardless of its size. Now we would like to make a statement like "If Φ has a model at all, then it has a polynomial-size model." We will not be able to prove something like: "If Φ has a model, then this model has to be at most this large." As we know, models can be arbitrarily large. For translations of QBF, however, we can exploit the regular problem structure in order to find "small" models which are guaranteed to exist (if the formula is satisfiable at all). Let us revisit Proposition 4.7.5:

Proposition 4.7.11 *Given a propositional valuation V on $p_1, \ldots p_j$, $j \leq n$, define l_i from p_i for all $i \leq j$, as in Proposition 4.7.5. If $\varphi_j(p_1, \ldots, p_j)$ is valid under V, then there exists a model for $\lambda(\varphi_j(p_1, \ldots, p_j)) \cup \{*^{n-j}\, l_i : i \leq j\}$ containing exactly the following types of assertions:*

- $*^{i-j-1}c_\top *^{n-i}\, p_i$ *and* $*^{i-j-1}c_\bot *^{n-i}\, \neg p_i$, *where* $Q_i = \forall p_i$, $i \geq j+1$
- $x_1 \cdots x_{i-j-1} c\, *^{n-i}\, p_i$ *and/or* $x_1 \cdots x_{i-j-1} c\, *^{n-i}\, \neg p_i$, *where* $Q_i = \exists p_i$, $i \geq j+1$, *and* $x_1 \cdots x_{i-j-1}$ *is a label made up of constants and* $*$ *which is realizable in* $\lambda(\varphi_j(p_1, \ldots, p_j))$,
- $*^{n-j}\, l_i$ *for any* $i \leq j$.

Note that there are at most $2n$ assertions of the first and third kind combined; but of the second kind there may be exponentially many in any given model.

Proof: Look at the relevant parts of the proof of Proposition 4.7.5. In the case $j = n$, we seek a model for the set $S = \{\varphi_n(p_1, \ldots, p_n), l_1, \ldots, l_n\}$. Theorem 4.4.15, applied to a formula with empty labels only, grants that for any such model M, $M \cup \{\varepsilon\, \top\}$ contains an atomic cover T_A for S, and of course it must be clash-free. But as we stated in Proposition 4.7.5, any cover for S must contain $l_1, \ldots, l_n$. We easily see that $\{\top, l_1, \ldots, l_n\}$ is already a maximal clash-free set of atoms, and since a model was guaranteed to exist, $\{\varepsilon\, l_1, \ldots, \varepsilon\, l_n\}$ must be a model.

Now look at the proof of the recursive step from $j+1$ to j. In the existential case, our hypothesis states that system (4.13)—which we found to be of the form $S \cup \{F\}$—has a model M' consisting of assertions of the form

- $*^{i-j-2}c_\top *^{n-i}\, p_i$ and $*^{i-j-2}c_\bot *^{n-i}\, \neg p_i$, where $Q_i = \forall p_i$, $i \geq j+2$

- $x_1 \cdots x_{i-j-2} c \, *^{n-i} \, p_i$ and/or $x_1 \cdots x_{i-j-2} c \, *^{n-i} \, \neg p_i$, where $Q_i = \exists p_i$, $i \geq j+2$, and $x_1 \cdots x_{i-j-1}$ is realizable in $\lambda(\varphi_{j+1}(p_1, \ldots, p_{j+1}))$,
- $*^{n-j-1} \, l_i$ for any $i \leq j$,
- one of $*^{n-j-1} \, p_{j+1}$ or $*^{n-j-1} \, \neg p_{j+1}$.

In accordance with the proof of Lemma 4.7.4, part (2), we define a model $\widehat{M}$ for system (4.14)—which was known to be of the form $* \, S \cup \{c \, F\}$: Prefix the last assertion in the above enumeration with c, and prefix all other assertions with $*$. As shown in the proof of Lemma 4.7.4, this is a model for system (4.14), and we check easily that $\widehat{M}$ also contains exactly the types of assertions proposed.

Now consider the universal case. By the hypothesis, the two sets in system (4.15)—which were found to be of the form $S \cup S'$ and $S \cup S''$, respectively— have respective models M' and M'', consisting of:

- $*^{i-j-2} c_\top *^{n-i} \, p_i$ and $*^{i-j-2} c_\bot *^{n-i} \, \neg p_i$, where $Q_i = \forall p_i$, $i \geq j+2$
- $x_1 \cdots x_{i-j-2} c \, *^{n-i} \, p_i$ and/or $x_1 \cdots x_{i-j-2} c \, *^{n-i} \, \neg p_i$, where $Q_i = \exists p_i$, $i \geq j+2$, and $x_1 \cdots x_{i-j-1}$ is realizable in $\lambda(\varphi_{j+1}(p_1, \ldots, p_{j+1}))$,
- $*^{n-j-1} \, l_i$ for any $i \leq j$,

plus $*^{n-j-1} \, p_{j+1}$ in M', and $*^{n-j-1} \, \neg p_{j+1}$ in M''. Now the proof of Lemma 4.7.4 part (3) provides us with a model M of system (4.16) which has the following properties:

- Since $*^{n-j-1} \, p_{j+1}$ and $*^{n-j-1} \, \neg p_{j+1}$ are unique to their respective models, M contains $c_\top *^{n-j-1} \, p_{j+1}$ and $c_\bot *^{n-j-1} \, \neg p_{j+1}$;
- therefore, $c_\top$ and $c_\bot$ already exist in M, and we need not add $c_\top \, \top$ or $c_\bot \, \top$;
- For every universal Q_i, the two assertions $*^{i-j-2} c_\top *^{n-i} \, p_i$ and $*^{i-j-2} c_\bot *^{n-i} \, \neg p_i$ are identical in M' and M'', so the corresponding assertions in M are $*^{i-j-1} c_\top *^{n-i} \, p_i$ and $*^{i-j-1} c_\bot *^{n-i} \, \neg p_i$.
- Likewise, $*^{n-j-1} \, l_i$, for $i \leq j$, is in both M' and M'', so $*^{n-j} \, l_i$ is in M.
- For the existential Q_i the various labels $x_1 \cdots x_{i-j-2}$ in M' and M'' may differ, so the corresponding assertions may be prefixed by any of $c_\top$, $c_\bot$, or $*$, yielding the

assertions included in M. However, $x_1 \cdots x_{i-j-2}$ is realizable in $\lambda(\varphi_{j+1}(p_1, \ldots, p_{j+1}))$ according to the hypothesis; now the rgis of $\lambda(\varphi_j(p_1, \ldots, p_j))$ are exactly those of $\lambda(\varphi_{j+1}(p_1, \ldots, p_{j+1}))$, with $c_\top$ and $c_\bot$ prepended. This guarantees that the extended labels $x_0 x_1 \cdots x_{i-j-2}$ are always realizable, no matter whether x_0 is $*$, $c_\top$, or $c_\bot$.

To summarize, the model M contains only the types of assertions stated in the proposition, which completes the proof of the recursive case. □

Corollary 4.7.12 *Given a valid QBF $\varphi_0 = Q_1 \ldots Q_n \, \varphi(p_1, \ldots, p_n)$, the translation $\lambda(\varphi_0)$ has a model M containing exactly the following types of assertions:*

- $*^{i-1} c_\top *^{n-i} \, p_i$ *and* $*^{i-1} c_\bot *^{n-i} \, \neg p_i$, *where* $Q_i = \forall p_i$, $i = 1, \ldots, n$,
- $x_1 \cdots x_{i-1} c \, *^{n-i} \, p_i$ *and/or* $x_1 \cdots x_{i-1} c \, *^{n-i} \, \neg p_i$, *where* $Q_i = \exists p_i$, $i = 1, \ldots, n$, *and* $x_1 \cdots x_{i-1}$ *is realizable in* $\lambda(\varphi_0)$.

Proof: Again this is just Proposition 4.7.11, written out for $j = 0$. □

A few observations on these models are in order. First, all labels in M are realizable in $\lambda(\varphi)$. In fact, we easily see that $I(M) = I(\lambda(\varphi))$. In other words, M does not have any realized ground label other than those "prescribed" by the problem. Secondly, for the universal quantifiers Q_i, M defines only the two assertions $*^{i-1} \, c_\top *^{n-i} \, p_i$ and $*^{i-1} \, c_\bot *^{n-i} \, \neg p_i$, as required by $\lambda(\varphi)$. The only "slack" in M is in how the $x_1 \cdots x_{i-1}$ in all the existential Q_i are defined, and how many of these labels are needed to specify M completely. These labels are also the only part of M which may be exponentially larger than φ. (As a matter of fact, exactly this happened in our example in Proposition 4.7.10.) We already know from Proposition 4.7.9 that $\lambda(\varphi)$ has 2^{n-m} realized ground labels of length n, where $n - m$ is the number of universal quantifications in φ_0. The size of a model can be bounded in a similar way:

Corollary 4.7.13 *If φ_0 is a valid QBF with $n - m$ universal and m existential quantifications, then $\lambda(\varphi)$ has a model with at most $m2^{n-m} + 2(n - m)$ assertions.*

Proof: Consider a fixed i so that Q_i is existential. (If no such index exists, we can skip this first part.) In the proof of Proposition 4.7.11, consider the maximal number of assertions mentioning p_i in a model of $\varphi_j(p_1, \ldots, p_j)$ over all valuations on $p_1, \ldots, p_j$. As long as $j \geq i$, only one assertion mentions p_i, namely $*^{n-j} \, l_i$. For each step $j < i$ of Proposition 4.7.11

where Q_j is existential, the number of assertions does not increase, whereas for universal Q_j it doubles at worst, as two models are combined into one. In total 2^{n-m} assertions mention p_i, and there are m indices i so that Q_i is existential, for a total of at most $m2^{n-m}$ assertions. Each universal Q_i contributes exactly 2 assertions as we see from Corollary 4.7.12, which contributes another $2(n-m)$ assertions. □

We are now ready to address our original quest for classes of formulas with *polynomial-size* models:

Proposition 4.7.14 *If φ is a valid formula in $\mathbf{QBF}_2$ with m existential and $n-m$ universal quantifiers, then $\lambda(\varphi)$ has a model of size $O(n^2)$.*

Proof: Consider the possible labels in Corollary 4.7.12. In $\mathbf{QBF}_2$ the existential quantifiers are all located at the beginning of the formula, so the only constant occurring in the first m positions of any label is c. Therefore, the labels $x_1 \cdots x_{i-1}c$ preceding the assertions which mention any existentially quantified variable p_i have only one rgi, namely the label c^i. In fact, any $*$ in these positions $x_1 \cdots x_{i-1}$ can be replaced by c, thus obtaining an equivalent model for $\lambda(\varphi)$. In this model there exists only one assertion mentioning each of $p_1, \ldots, p_m$. For the *universally* quantified p_i, we have two assertions each, for a total of $O(n)$ assertions, each with a label of length n. □

To summarize, we have demonstrated that the class of translations of $\mathbf{QBF}_2$ gives rise to polynomial-size models for all satisfiable formulas, whereas in the comparable class of translations of $\mathbf{QBF}_2$ into $\mathbf{K}$ only worst-case exponential-size models exist. We have thus shown how for a large and interesting class of problems, reasoning with translations into $\mathcal{LBL}$ is more powerful than reasoning with translations into $\mathbf{K}$. (For $\mathbf{QBF}_3$ and higher, as well as for the duals $\overline{\mathbf{QBF}}_2$ and higher, we demonstrated that there are no general polynomial bounds for translations into $\mathcal{LBL}$.)

This completes our discussion of theoretical upper bounds for the size of models. Although the strong models we define in the next chapter are more expressive than the weak models discussed here, they will not provide an improvement on these general upper bounds. Instead, they will serve our goal of developing a powerful algorithm for *finding* models efficiently. But as it turns out, their greater expressivity *will* often result in smaller models *in practice*. We will even exhibit families of problems whose models are reduced in size from exponential to polynomial.

4.8 Non-strict Kripke Models and Labellings

In our discussion so far, we have compared the conciseness of models in $\mathcal{LBL}$ to that of *strict* Kripke models only. In particular, Corollary 4.5.14 gives an upper bound for the size of models M for strict labellings of Φ in terms of the size of strict Kripke models for Φ. This restriction is not fair though; as we have already pointed out, it unduly limits the potential for reducing the size of a satisfying model. We already discussed this in Section 3.3. On the other hand, we should not limit ourselves to strict *labellings* either, which would restrict the potential of finding small models. However, as we have already noted, it may be hard to guarantee whether a non-strict labelling preserves the satisfiability of the problem.

Apart from that issue, we will find that models for non-strict labellings do not correspond as nicely to non-strict Kripke models; furthermore, we will not be able to give a polynomial upper bound for the model of *any* labelling of Φ, in terms of a minimal-size non-strict Kripke model for Φ. But as we will see, the counterexamples are quite contrived, so there is reason to believe that this worst-case behaviour is rather theoretical, and that models for $\mathcal{LBL}$-formulas are not worse (but on the contrary, more concise) than their Kripke model counterparts in practice. The examples will provide a deep understanding of the difference between labels and modal operators.

It is helpful to categorize non-strict Kripke models as follows:

Cyclic models: models which have cycles such as loops or backward arcs. See Example 4.8.1 and Figure 4.3.

DAG models: acyclic models in which multiple paths exist from w_0 to some world w. See Example 4.8.4 and Figure 4.4.

Non-strict trees: tree models which do not match Definition 3.4.1. See Example 4.8.7 and Figure 4.7.

Example 4.8.1 The set $\Phi = \{p, \Diamond\, p, \Box \Diamond\, p, \ldots, \Box^{k-1} \Diamond\, p\}$ (see Figure 4.3) has a cyclic model $K = (\{w_0\}, R, V)$, where $w_0\, R\, w_0$ and $V(p) = \{w_0\}$. The size of K is $O(1)$, whereas the smallest acyclic model has $k + 1$ worlds.

Our labelling algorithm does not allow us to assign the same label to formulas at different modal depth. In Example 4.8.1, a labelling using a minimal number of constants is $S =$

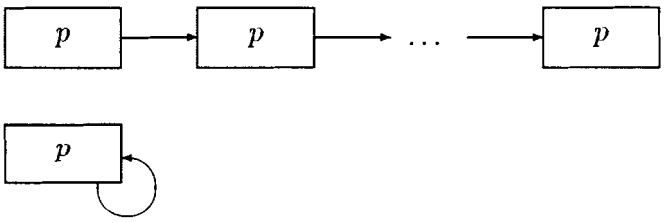

Figure 4.3: *A strict (top) and a cyclic model (bottom) for* $\{p, \Diamond p, \Box \Diamond p, \ldots, \Box^{k-1} \Diamond p\}$.

$\{p, c\, p, *c\, p, \ldots, *^{k-1} c\, p\}$, and it is easy to show that S is not only a model for itself, but also that it is minimal. So any model for any labelling is of size $\Theta(k^2)$.

As we can see, the size of a minimal cyclic model for a set Φ may be smaller than that of a minimal acyclic model for Φ. However, the size reduction is at most linear—or quadratic, if we account for a linear amount of space to store world labels—in $k = d(\Phi)$. Here is how a cyclic model $K = (W, R, V)$ can be translated into an acyclic model $K' = (W', R', V')$ with $(k+1)|W|$ worlds:

- $W' = \{(w, i) : w \in W, i = 0, \ldots, k\}$.
- $R' = \{((w, i), (w', i+1)) : (w, w') \in R, i = 0, \ldots, k-1\}$.
- $V'(p) = \{(w, i) : w \in V(p), i = 0, \ldots, k\}$.

Proposition 4.8.2 *If* $j = d(\Phi)$ *and* $K, w \models_k \Phi$, *then* $K', (w, i) \models_k \Phi$ *for* $i = 0, \ldots, k - j$.

Proof: This is easily shown using induction over $d(\Phi)$. Whenever in the course of evaluating $K, w \models_k \Box\, \nu_0$ or $K, w \models_k \Diamond\, \pi_0$ we evaluate $K, w' \models_k \nu_0, \pi_0$ (where $w\ R\ w'$), respectively, we can analogously evaluate $K', (w', i+1) \models_k \nu_0, \pi_0$ in order to determine whether $K', (w, i) \models_k \Box\, \nu_0, \Diamond\, \pi_0$; since $d(\nu_0)$ and $d(\pi_0)$ are at most $j-1$, we are guaranteed that $i+1 \leq k-(j-1)$, which allows us to apply the proposition as an induction hypothesis. Full details of the proof are left to the reader. □

From this construction we immediately obtain what we have claimed above:

Corollary 4.8.3 *If* Φ *has a cyclic model* K, *then* Φ *has an acyclic model of size* $O((d(\Phi))^2 \times |K|)$.

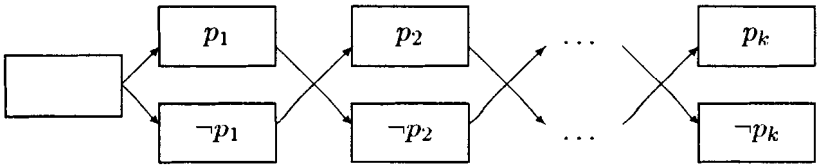

Figure 4.4: *A DAG model for* $\{\Diamond\, p_1, \Diamond\, \neg p_1, \Box \Diamond\, p_2, \Box \Diamond\, \neg p_2, \ldots, \Box^{k-1} \Diamond\, p_k, \Box^{k-1} \Diamond\, \neg p_k\}$.

So the space overhead for obtaining an acyclic model is low-order polynomial in $d(\Phi)$, whereas $d(\Phi)$ is typically—but not necessarily—logarithmic in $|\Phi|$. If we could obtain small models for labellings of Φ from any acyclic model for Φ, these models would still be reasonably small compared to cyclic models for Φ[8].

DAG models, on the other hand, offer the potential for a substantial space reduction:

Example 4.8.4 Consider $\Phi = \{\Diamond\, p_1, \Diamond\, \neg p_1, \Box \Diamond\, p_2, \Box \Diamond\, \neg p_2, \ldots, \Box^{k-1} \Diamond\, p_k, \Box^{k-1} \Diamond\, \neg p_k\}$. A DAG model for Φ consisting of $2k+1$ worlds is given in Figure 4.4. A minimal tree model for Φ has 2^k worlds at modal level k, for a total of $2^{k+1} - 1$ worlds.

If we construct a labelling S from Φ by replacing each $\Diamond$ with a distinct constant, then S provides a model for itself, as we can easily see. Thus S has a model with $2k$ assertions.

Notice that Φ is just a translation of the QBF $\forall p_1 \cdots \forall p_k\, \top$. So the result about S can also be obtained from Theorem 4.7.11. It is also worth pointing out that S is a *strict* labelling. Thus, by using $*$ in the model, even a strict labelling gives rise to a model which is comparable to the size of the smallest DAG model for Φ.

Unfortunately, this promising demonstration of the expressivity of $\mathcal{LBL}$ does not generalize. So far we did not mention that labels are slightly more rigid than the modal operators: While $\Diamond\, \varphi$ says that φ is *true* in *some* successor of the current world w, the formula $c\, F$ states that F is *true* in *the* successor of the current label γ which is labelled γc. Here we do not have the flexibility to switch roles between the constants c. This fact has one very useful consequence, namely that e.g. $c\,(F \wedge G)$ and $c\, F \wedge c\, G$ are equivalent, whereas $\Diamond\, (\varphi \wedge \psi)$

[8]Note though that a polynomially smaller model may translate into an exponentially smaller time complexity for finding a model. Assume all the p in the formula Φ of Example 4.8.1 were replaced by $p \vee q$. Then we may have 2^{k+1} combinations of variable assignments in K', but only 2 variable assignments in K. For this reason it is a useful endeavour to find an equivalent of cyclic models for labellings. We will sketch a few ideas in Chapter 7.

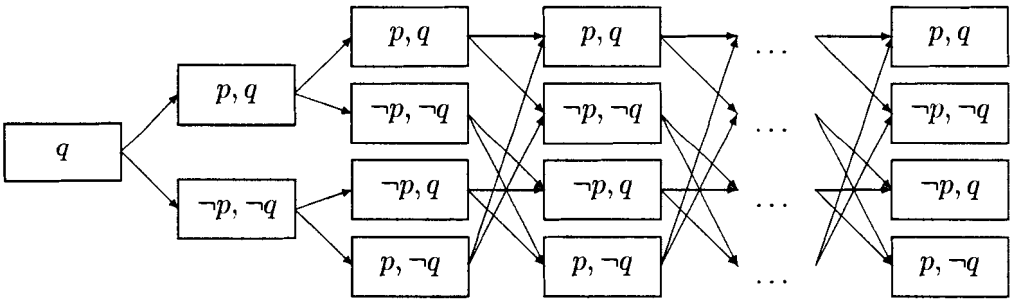

Figure 4.5: *A "ladder graph" model for Φ in Example 4.8.5.*

and $\Diamond\, \varphi \wedge \Diamond\, \psi$ are not. (The latter corresponds more closely to the labelling $c\, F \wedge c'\, G$.) But this convenience comes at a steep price: We can construct families of $\mathbf{K_{NNF}}$-formulas which have polynomial-size DAG models, but any labelling (whether strict or not) has only exponential-size models. The construction is somewhat intricate, but it reveals how the rigidity of labels stands in the way of finding small models:

Example 4.8.5 Consider the set

$$\Phi = \{q\} \cup \bigcup_{i=0}^{k} \Phi_i,$$

where $\Phi_i = \{\Box^i \Diamond\, p, \Box^i \Diamond\, \neg p, \Box^i\, (q \rightarrow \Box\, (p \leftrightarrow q)), \Box^i\, (\neg q \rightarrow \Box\, (p \leftrightarrow \neg q))\}$. (The expressions involving $\rightarrow$ and $\leftrightarrow$ can be replaced by suitable expressions in NNF.) We can describe this set of formulas as follows: In the root world, q must be *true*. Furthermore, each world w has one successor in which p is *true*, and one in which p is *false*. In the successor world where p is *true*, q has the same truth value as in w, whereas in the other successor q has the opposite truth value. We can easily show that Φ has a DAG model with $4k - 1$ worlds as shown in Figure 4.5. We notice that worlds are distinguished by the truth values on p and q, which allows for only 4 distinct combinations. Furthermore, the truth values of a successor of w depend only on the truth value (of q) in w and no other world in its ancestry. Hence, from the third level onwards we can re-use successors, which prohibits further combinatorial explosion. (In fact, we can construct a cyclic model with only 4 worlds, as shown in Figure 4.6, where the root world can be chosen among the two left-hand nodes in the graph.)

A translation of this example into labelled form is straightforward, so we do not write it out. The only $\Diamond$-operators of Φ occur in $\Box^i \Diamond\, p$, $\Box^i \Diamond\, \neg p$. A replacement of these two $\Diamond$ by the

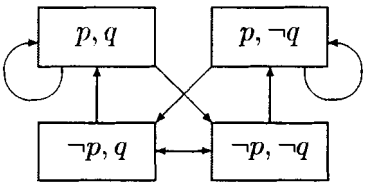

Figure 4.6: *A cyclic model for* Φ *in Example 4.8.5.*

same constant would result in a clash, so we must label them with two different constants $c_\top$ and $c_\bot$ respectively. (The same constants can be reused for different i; no clash can arise from that.) Thus obtaining a labelling S of Φ, we see that S spans 2^i realized ground labels of length i. But this in itself does not prove that any model of S must be exponential in size. Instead, we consider the truth values of q in the different instances:

Claim 4.8.6 *For any model M of S and any ground label $c_1 \cdots c_i$, we have $M_{\langle c_1 \cdots c_i \rangle} \models_l q$ iff an even number of constants in $c_1 \cdots c_i$ is $c_\bot$.*

Proof: As usual, this is proved by induction on i. For $i = 0$, $M \models_l q$ is evidenced by the fact that q is an atom in S. Assume the claim is valid for some $i < k$. (We will only discuss the case $M_{\langle c_1 \cdots c_i \rangle} \models_l q$, i.e. an even number of constants in $c_1 \cdots c_i$ is $c_\bot$. The opposite case is analogous.) From the induction hypothesis, as well as the labellings of the formulas in Φ_i, we get the following requirements:

$$M_{\langle c_1 \cdots c_i \rangle} \models_l \{q,\ c_\top\, p,\ c_\bot\, \neg p,\ q \rightarrow *(p \leftrightarrow q),\ \neg q \rightarrow *(p \leftrightarrow \neg q)\}.$$

From these we infer that both $c_\top$ and $c_\bot$ must exist in $M_{\langle c_1 \cdots c_i \rangle}$, and $M_{\langle c_1 \cdots c_i c_\top \rangle}$ and $M_{\langle c_1 \cdots c_i c_\bot \rangle}$ must contain p and $\neg p$, respectively. While the last formula is verified (vacuously) by q being *true* in $M_{\langle c_1 \cdots c_i \rangle}$, the second-last formula entails $M_{\langle c_1 \cdots c_i \rangle} \models_l *(p \leftrightarrow q)$. Hence, $p \leftrightarrow q$ is necessarily verified in both $M_{\langle c_1 \cdots c_i c_\top \rangle}$ and $M_{\langle c_1 \cdots c_i c_\bot \rangle}$. In the former, p is already *true*, so we get $M_{\langle c_1 \cdots c_i c_\top \rangle} \models_l q$. (Note that the subscript $c_1 \cdots c_i c_\top$ still contains an even number of $c_\bot$.) In the latter, p is *false*, which entails $M_{\langle c_1 \cdots c_i c_\bot \rangle} \models_l \neg q$. (This subscript contains an odd number of $c_\bot$.) In either case, the induction hypothesis has been verified for all successors of length $i + 1$, so the proof follows by induction on i. □

From here the argument follows that of Proposition 4.7.10: Suppose $\sigma\, q$ or $\sigma\, \neg q$ are assertions in M. If σ contains $*$, then it has at least one instantiation with an odd number of $c_\bot$ and one with an even number of $c_\bot$, so it either produces a clash, or M does not verify S. Hence

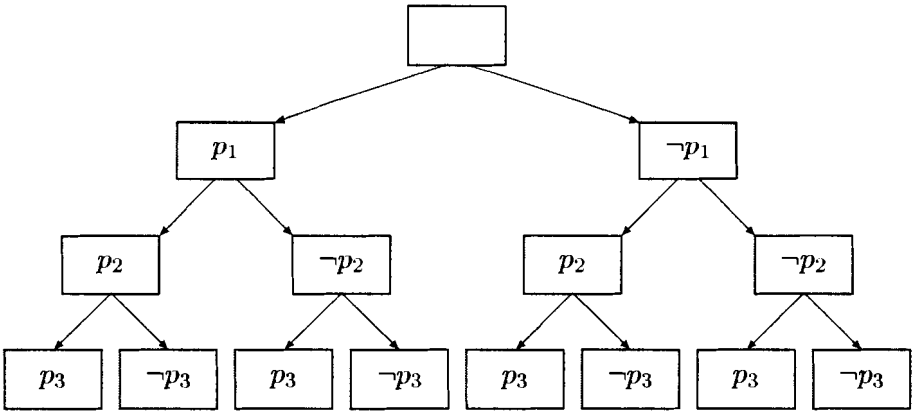

Figure 4.7: *A model for Φ in Example 4.8.7, for $k = 3$.*

any model M for S must specify the truth assignments for q using ground labels only, which results in $\Theta(2^k)$ assertions, making M exponentially larger than the Kripke model of Figure 4.8.5.

What if we restrict our choice of non-strict Kripke models to the third kind, namely tree models? It turns out that even then it is impossible to give a polynomial bound for models of labellings:

Example 4.8.7 Using the propositional variables $p_1, \ldots, p_k$ and q, we construct a set Φ containing the following formulas:

$$\begin{array}{ll}
\Box^{i-1} \Diamond \Box^{k-i}\ p_i & i = 1, \ldots, k \\
\Box^{i-1} \Diamond \Box^{k-i}\ \neg p_i & i = 1, \ldots, k \\
\Box^{i-1} \Diamond \Box^{k-i}\ (p_j \leftrightarrow p_i) & i = 2, \ldots, k, j = 1, \ldots, i-1 \\
\Box^{i-1} \Diamond \Box^{k-i}\ (p_j \leftrightarrow \neg p_i) & i = 2, \ldots, k, j = 1, \ldots, i-1 \\
\Box^{k}\ ((p_1 \wedge \ldots \wedge p_k) \leftrightarrow q) &
\end{array}$$

This set has $k^2 + k + 1$ formulas. The first two lines "span" a Kripke tree K with 2^i nodes at depth $i \leq k$, similar to the strict model for Example 4.8.4, except that the truth assignments to all the variables p_i refer to the worlds at depth k. Notice that p_i is uniformly assigned *true* or *false* in all successors of the same world at depth i; in our discussion below, we say that this world *establishes* the truth value of p_i in all its successors at depth k. For $k = 3$

the tree is shown in Figure 4.7. To preserve space, we have written p_i or $\neg p_i$ into the worlds at depth i which establish the value of p_i, rather than into all its successors at depth k. The fifth line can be satisfied by assigning q to *true* in exactly the one node where all the p_i are *true*. It turns out that the formulas in lines 3 and 4 are also satisfied by this model, which we can see as follows: Every node at depth $i-1$ has one successor which establishes p_i and one which establishes $\neg p_i$. Therefore, for any variable assignment $p_j, j < i$ already established in its ancestor nodes, there is always one successor in which p_j and p_i are assigned the same truth value, as well as one in which p_j and p_i are assigned opposite truth values. (It seems plausible that a theorem prover with optimizations would find this Kripke model.) We do not have the same flexibility in choosing successors in $\mathcal{LBL}$. In fact, we cannot replace any two $\Diamond$-operators in the $2i$ formulas on the same level i by one and the same constant. To illustrate this, let us say we assigned the same constant in the formulas containing $p_j \leftrightarrow p_i$ and $p_l \leftrightarrow \neg p_i$:

$$*^{i-1}c*^{n-i}p_j \leftrightarrow p_i,$$
$$*^{i-1}c*^{n-i}p_l \leftrightarrow \neg p_i.$$

If $j = l$, the contradiction is obvious, so assume $j \neq l$. The variables p_j and p_l themselves are also found in two labellings of formulas in the first two lines, say:

$$*^{j-1}c_j*^{n-j}p_j$$
$$*^{l-1}c_l*^{n-l}p_l$$

Observe that the three labels $*^{j-1}c_j*^{n-j}$, $*^{l-1}c_l*^{n-l}$, and $*^{i-1}c*^{n-i}$ unify; in any of its realized ground instances, both p_j and p_l must be *true*. Moreover, p_i must have the same truth assignment as p_j and opposite truth assignment from p_l, which is impossible. For any other pair of formulas, a similar contradiction can be obtained. Thus a labelling S of Φ using a minimal number of constants is:

$$*^{i-1}c_0^{\top}*^{k-i}p_i \qquad i = 1, \ldots, k$$
$$*^{i-1}c_0^{\bot}*^{k-i}\neg p_i \qquad i = 1, \ldots, k$$
$$*^{i-1}c_j^{\top}*^{k-i}(p_j \leftrightarrow p_i) \qquad i = 2, \ldots, k, j = 1, \ldots, i-1$$
$$*^{i-1}c_j^{\bot}*^{k-i}(p_j \leftrightarrow \neg p_i) \qquad i = 2, \ldots, k, j = 1, \ldots, i-1$$
$$*^{k}(p_1 \wedge \ldots \wedge p_k \leftrightarrow q)$$

Let us first calculate the number of realized ground instances of $*^k$. We see that exactly $2i$ constants are introduced at each level i, namely $c_0^\top, c_0^\bot, \ldots, c_{i-1}^\top, c_{i-1}^\bot$. The ground labels of length k are arbitrary combinations of constants at each level, which results in $2 \times 4 \times \ldots \times 2k = 2^k k!$ different realized ground instances.

Models may not necessarily be that large, as the placeholder $*$ may be used in some instances. (In fact, as a combinatorial curiosity, the truth assignments for $p_1, \ldots, p_k$ can be specified by a total of $3^k - 1$ assertions, which is still polynomial in 2^k, the number of worlds in K.) But we can give a superpolynomial lower bound for the number of assertions mentioning q: Let γ be one of the realized ground instances above, then q is *true* in $M_{\langle\gamma\rangle}$ exactly when all p_i are *true*. We can easily see from the formulas above that not all p_i can be *true* if any of the constants in γ is $c_j^\bot$. Conversely, in all labels consisting only of $c_j^\top$ (where j may vary), p_1 is assigned *true*, and all other variables are either directly assigned *true* (in case $j = 0$), or they have the same truth value as one of the variables before, i.e. *true* also. So the assertions $\sigma\, q$ in M must cover exactly all realized ground instances γ consisting only of constants $c_j^\top$. (There are exactly $k!$ of these.) If σ contained $*$, then $*$ would be instantiated by any $c_j^\bot$ as well, thus including ground instances in which q is not verified and leading to M either containing a clash or not verifying S. Hence, all these σ must be ground, so M must include $\Theta(k!)$ assertions which mention q alone.

Let us summarize our observations. Since $k!$ is not polynomially bounded by 2^k—though growing much less than exponentially—we have provided a family of satisfiable formulas any of whose labellings, if satisfiable, has only superpolynomially larger models than the smallest tree models for each original formula. In our example, the tree model K is of size 2^k which is not polynomial in the size of Φ to begin with; but one can always "make it" polynomial by "padding" Φ with 2^k "dummy" formulas which do not affect the model.

These last two examples do appear rather contrived. They both feature an exponential number of realized ground instances, and truth assignments were based on some notion of parity in the labels themselves; this prohibits the applicability of $*$. (This holds unchanged even for the strong models we will introduce in the next chapter.) In more "typical" cases one should expect that $*$ will be of greater use.

Chapter 5

Strong models for $\mathcal{LBL}$

El camino es largo y desconocido en parte; conocemos nuestras limitaciones. Haremos el hombre del siglo XXI: nosotros mismos. Nos forjaremos en la acción cotidiana, creando un hombre nuevo con una nueva técnica.

The road is long and partly hidden; we know our own limitations. We will create the man of the 21st century: ourselves. We will forge ourselves into a new shape, through our everyday action, creating a new man using a new technique.

Ernesto (Ché) Guevara, "El hombre nuevo"

5.1 Introduction and Motivation

Let us follow up on Example 4.1.1, where we have not yet specified the worlds in which, say, $y = 1$ is *false*. Of course, these are the worlds with labels $c_x c_y c_z$, where $c_y \neq 1$. (The mutex constraints forced us to assign $y = 1$ to *false* on these; we will not be concerned about why, but rather about how these value assignments can be represented.) The assertions $*2* \neg(y = 1), \ldots, *n* \neg(y = 1)$ accurately capture these worlds. But this representation is still fairly redundant, especially when n is large. If we did this for all propositions in the example, then we would need $3n^2$ assertions in total, counting those with positive literals ($*1* y = 1$ etc.) as well.

This example reflects a very common situation in reasoning. (For another example, recall the birds example from the introduction chapter.) In the vast majority of instances, a proposition has one default truth value assigned to it, such as "$y = 1$ is *false*", or "birds

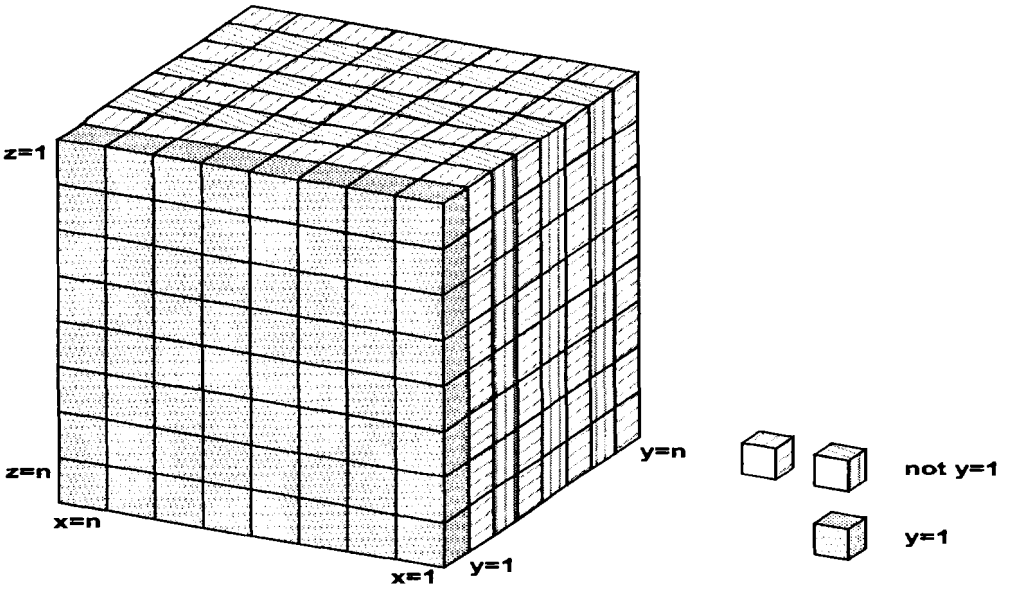

Figure 5.1: The sets of cubes satisfying $y = 1$ and falsifying $y = 1$, respectively.

fly", and in a comparatively small number of instances, this truth assignment would lead to a clash (because we know that penguins cannot fly, for instance), so we must assign opposite truth values in these instances.

We will see that both the entire space of instances (e.g. those satisfying *bird*) and the sets of exceptional instances (e.g. those satisfying *penguin*, *emu* etc.) usually form a block structure. By this we mean that the instances can be represented in closed form by just one label with wildcards. Many of our results in the previous chapter rely on the block structure afforded by simple labels with wildcards. For example, a label $x_1 \cdots x_k$ can be viewed as the concatenation of its positions $x_1, \ldots, x_k$; the set of ground instances represented by $x_1 \cdots x_k$ is the Cartesian product of the sets represented by the x_i—either some $\{c_i\}$, or D. The *set difference* of all default instances minus their exceptions (e.g. *flying-bird*), however, does not have a block structure—usually, it cannot be represented by one label alone. In the example above, we re-establish a block structure by "slicing up" the set of instances satisfying $\neg(y = 1)$ into $n - 1$ blocks(planes): each block is represented by a label with two $*$, namely $*c_y*$, $c_y = 2, \ldots, n$. But this approach is not generalizable, as we will see in our next example. Instead, we desire to redefine our logic, allowing us to represent set

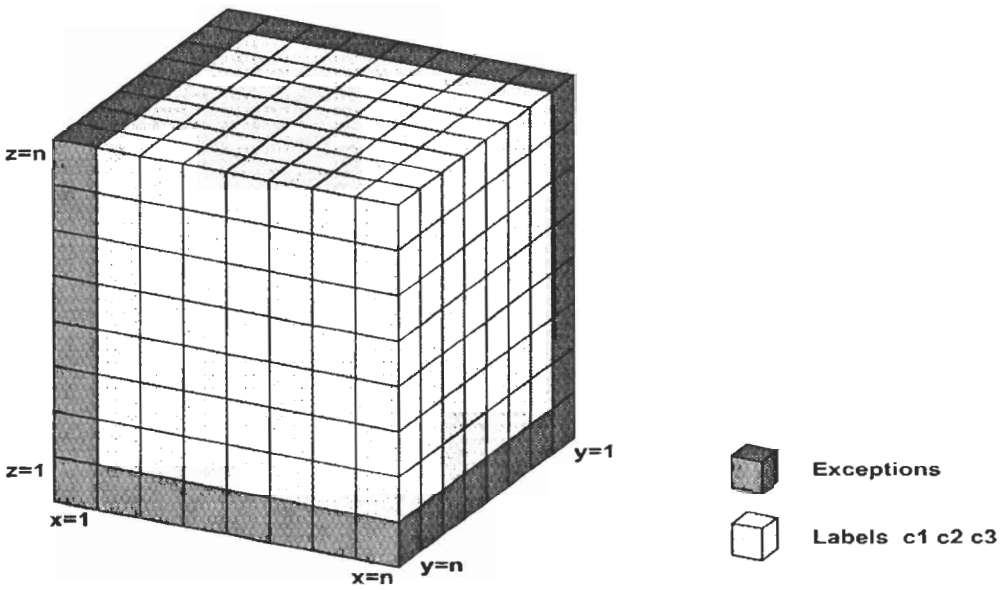

Figure 5.2: The label $*** - \{1**, *1*, **1\}$, expressed with simple labels. The cube has been rotated compared to Figure 5.1, to make the instances visible.

differences within labels, so we need not establish a block structure. Hence we introduce labels with exceptions. Labels can have several exceptions, so we represent them in the form $\sigma - \Sigma$, where Σ is a set of instances (usually) of σ. Let us apply this notation to specify all negative assertions for Example 4.1.1:

$$\begin{array}{lll} (*** - \{1**\})\ \neg(x = 1) & (*** - \{*1*\})\ \neg(y = 1) & (*** - \{**1\})\ \neg(z = 1) \\ \ldots & \ldots & \ldots \\ (*** - \{n**\})\ \neg(x = n) & (*** - \{*n*\})\ \neg(y = n) & (*** - \{**n\})\ \neg(z = n) \end{array} \tag{5.1}$$

Together with the positive assertions in (4.2) which remain unchanged, $6n$ assertions in total define a model for Example 4.1.1 obeying the implicitly stated mutual exclusivity constraints. In this example, the space gain over labels without exceptions is another linear factor in n, which is not as spectacular as our previous exponential gain over ground labels in Section 4.1, but still considerable. This is quite typical for many problems. In some cases however, models do become exponentially more concise:

Example 5.1.1 In addition to the formulas in Example 4.1.1, we will introduce another formula specifying a predicate h to characterize the set of all elementary cubes which are invisible in Figure 5.1. In Figure 5.2, our $n \times n \times n$ cube has been rotated to reveal these hidden cubes. It is easy to see that a cube is hidden unless any of its coordinates is equal to 1, so we write it out as

$$***\Big(h \leftrightarrow \big(\neg(x = 1) \wedge \neg(y = 1) \wedge \neg(z = 1)\big)\Big).$$

The same formula in negation normal form reads:

$$F_1 = *** \Big(\neg h \vee \big(\neg(x = 1) \wedge \neg(y = 1) \wedge \neg(z = 1)\big)\Big)$$
$$F_2 = *** \Big(h \vee (x = 1) \vee (y = 1) \vee (z = 1)\Big).$$

The formula F_1 is easily satisfied by adding three assertions $1** \neg h$, $*1* \neg h$, $**1 \neg h$ to the set M; for whenever $x \neq 1$, $y \neq 1$, and $z \neq 1$, the conjunction is *true*, and we specified $\neg h$ on all other instances. To satisfy F_2, we need to assign h to *true* in all ground instances where none of the positions are equal to 1. These instances form the lightly shaded $(n-1) \times (n-1) \times (n-1)$-subcube shown in Figure 5.2. This subcube does not contain a subspace (plane or row) of the entire cube; so without the use of exceptions, wildcards cannot be used in any assertions. Instead we must use $(n-1)^3$ assertions $\gamma\, h$, where γ ranges over all ground instances of $***$ which do not contain the constant 1. If generalized to d dimensions, this number would be exponential in d, the depth of M. Using exceptions however, we can easily write out the variable assignments as one assertion, namely $A = (*** - \{1**, *1*, **1\})\, h$, and include A in M as well.

Now how do we efficiently *check* that F_1 and F_2 are satisfied? One way is to adapt the $\models_l$ and $\models_l$ relations of Chapter 4 to labels with exceptions. We will do this, but chiefly in order to transfer some results into this chapter, not to facilitate practical reasoning. Recall that at least a naïve implementation of $\models_l$, such as we used in the proof of Proposition 4.6.16, forces us to reason individually over each ground instance (this remains unchanged with or without exceptions), which is unacceptable. Instead, one merit of our notion of labels with exceptions is that they lend themselves to a novel way of reasoning "universally" over labels. Let us illustrate how M verifies F_2 above:

Ignoring for a moment that A specifies exceptions, let us "pretend" that $A = *** h$. Then the first disjunct in F_2 is universally verified on the label $***$ preceding F_2, which is sufficient for showing that $\{A\}$ verifies F_2. Now we must account for the exceptions specified in A,

that is, we must show that $(h \vee (x = 1) \vee (y = 1) \vee (z = 1))$ of F_2 is also verified over the instances $1{**}$, $*1*$, and $**1$. For this, we can cite the assertion $1{**}\,(x = 1)$ which also exists in M, so the second disjunct is verified on this instance. Likewise, the third and fourth disjunct are verified on the instances $*1*$ and $**1$, respectively. We have thus accounted for all exceptions in A, which shows that M verifies F_2. We will generalize this procedure into a definition of a new semantic relation, written $\models_s$.

In our description, we glossed over one inconspicuous but very important detail: How can we be certain that the assertions verifying F_2 on the various instances of $***$ can simply be "thrown together" into M, so that M still verifies F_2? To do so requires the property we have introduced in Chapter 2 as *monotonicity*, namely that any superset of a set of assertions verifying F still verifies F. This was not true for the relation $\models_l$ in Chapter 4, but we will prove it for $\models_s$, which constitutes a fundamental result of this chapter.

In order to rightfully call M a model, we also have to show that M is clash-free, which becomes slightly more involved on labels with exceptions. Let us demonstrate the principle on A and one of the other assertions in M, $1{**}\ \neg h$. The literals in these two assertions are complementary, and the main label in A, $***$, unifies with $1{**}$, (and their mgu is realizable, of course). So at first glance there appears to be a clash. But now observe that the mgu $1{**}$ is an exception in A, so the assertion does not "hold" on this instance. Given this, the two assertions do not form an actual clash.

Now how do we handle cases where variable assignments have "exceptions to exceptions"?

Example 5.1.2 Consider a predicate e stating that an even number of faces of an elementary cube are visible. This predicate is *true* in all cubes except those on the three front faces (with at least one coordinate equal to 1), but among these, it is again *true* in all cubes on the three front edges (with two coordinates equal to 1), except the cube 111. We need no recursive constructs to handle these. The predicate e can be stated using a set of assertions:

$$
\begin{aligned}
(*** - \{1{**}, *1*, **1\})\, e \\
(11* - \{111\})\, e \\
(1*1 - \{111\})\, e \\
(*11 - \{111\})\, e
\end{aligned}
$$

How large, or how redundant, can this set of assertions get? If we generalized this example to an analogous problem in d dimensions, we would be disappointed to find that a minimal

set of assertions would specify $\Theta(2^d)$ labels. We infer that a set specifying deeply nested recursions of exceptions is expensive to represent. But we maintain that PSPACE-problems of such a structure are intrinsically hard[1], so we do not expect our own formalism to represent these too efficiently either. One known example is the class of parity problems which we have already encountered in the previous chapter. In fact, the example above *is* an instance of the parity problem.

Before we begin our formal treatment, let us introduce some more terminology which will be used, and give an overview of the structure of this chapter:

- Complex labels are only used in assertions, not in labelled formulas in general. Therefore, the language of $\mathcal{CBL}$ remains *almost* unchanged (see Section 5.2). However, the relation $\models_l$ needs to be redefined for sets of assertions with complex labels. To distinguish our different flavours of logics, we subscript the "old" logic $\mathcal{CBL}$ of chapter 4 as $\mathcal{CBL}_w$ (for "weak"), and the logic admitting complex labels in semantic structures, but with an adapted version of the old relation $\models_l$ as $\mathcal{CBL}_x$ (for "complex"). The necessary extensions to the definitions of labels, instances, and most general unifiers are presented in Section 5.2. Finally, the logic defined by semantic structures with complex labels and the new "monolithic" relation $\models_s$ will be denoted as $\mathcal{CBL}_s$ (for "strong").

- In a complex label $\sigma - \Sigma$, we can reasonably expect that the exceptions in Σ are instances of σ; we do not formally require this, but we will single out such labels as *normalized* and formally show that assertions are adequately represented using normalized labels alone.

- Section 5.3 in this chapter is devoted to precisely describing recursion on exceptions and explaining how it works. Two notions from the previous chapter, *constant-wise realizability* and satisfiability itself, will be redefined using recursion on exceptions, in Sections 5.4 and 5.5 respectively. Section 5.5 also contains the new definitions for $\models_l$ and $\models_s$, respectively. In Section 5.6, we will make a necessary adjustment on how to treat assertions with falsities so they can be used in verifying formulas such as $\Box \bot$. Finally, in Section 5.7 we establish theorems regarding the equisatisfiability of

[1] At least they are hard for any reasoning algorithm which can be polynomially simulated by resolution.

formulas between the three logics $\mathcal{CBL}_w$, $\mathcal{CBL}_x$, and $\mathcal{CBL}_s$, and provide algorithms for converting models of one logic into models of another.

5.2 Complex Labels

As motivated above, we require a few extensions of earlier definitions and notions. Our first task is to establish a distinction between *existential* and *conditional* constants:

Definition 5.2.1 *Given a domain D of constants, a* simple label *is a string σ over the alphabet $\{c, [c] : c \in D\} \cup \{*\}$. We call $[c]$ a* conditional *and c an* existential *constant. A label is* ground *if it consists only of existential constants. Given a simple label σ, we let $[\sigma]$ denote the simple label obtained by replacing all constant positions c with conditional constants $[c]$.*

Let us illustrate the intended meaning of conditional versus existential constants: As we know, the assertion $c\,p$ states that c *exists* (or: is realized) and p is *true* in this instance c. By contrast, the assertion $[c]\,p$ only states that *if* c is realized, *then* p must be *true* in this instance c. Thus the distinction between conditional and existential constants is only relevant when we discuss realizability issues. Up to Section 5.4 we can think of $[c]$ and c as identical. In particular, they behave the same regarding instantiation: $[c]$ instantiates c and vice versa, and both instantiate $*$. As to the mgu (σ_1, σ_2), we introduce the following convention: in any position where both σ_1 and σ_2 have the same constant c, if at least one is existential, then that position in (σ_1, σ_2) is c, otherwise $[c]$.
The next two definitions are extensions of the $\sqsubseteq$ relation which will allow us to compare two labels of different length:

Definition 5.2.2 *For two simple labels σ, σ', we write $\sigma \sqsubseteq_p \sigma'$ if σ is instantiated by a prefix of σ'. Conversely, we write $\sigma \,{}_p\!\sqsubseteq \sigma'$ if a prefix of σ is instantiated by σ'.*

Definition 5.2.3 *Given two labels σ_1 and $\sigma_2 = \sigma_2'\sigma_2''$ so that (σ_1, σ_2') exists, we define $\lfloor \sigma_1, \sigma_2 \rfloor = \lfloor \sigma_2, \sigma_1 \rfloor = (\sigma_1, \sigma_2')$ and $\lceil \sigma_1, \sigma_2 \rceil = \lceil \sigma_2, \sigma_1 \rceil = (\sigma_1, \sigma_2')\sigma_2''$ as the* lower *and* upper *mgu of σ_1 and σ_2, respectively.*

In other words, $\lfloor \sigma_1, \sigma_2 \rfloor$ is the longest and most general label σ so that $\sigma_1 \,{}_p\!\sqsubseteq \sigma$ and $\sigma_2 \,{}_p\!\sqsubseteq \sigma$, and $\lceil \sigma_1, \sigma_2 \rceil$ is the shortest and most general label σ so that $\sigma_2 \sqsubseteq_p \sigma$ and $\sigma_1 \sqsubseteq_p \sigma$. Although

it is true that $\lfloor \sigma_1, \sigma_2 \rfloor = \lceil \sigma_1, \sigma_2 \rceil = (\sigma_1, \sigma_2)$ whenever σ_1 and σ_2 are of the same length, these notions are not an extension of $(\cdot, \cdot)$: If σ_1 and σ_2 are of different length, (σ_1, σ_2) does not exist, and we wish to keep it that way. The old and the new notions of mgus will be used in different contexts: Lower and upper mgus will often be used when σ_1 is a simple label and σ_2 is an exception of a complex label, or vice versa. The more restrictive unifier $(\cdot, \cdot)$ will be generalized to complex labels and used to determine if a label is realizable.

Definition 5.2.4 *A* complex label, *or* label with exceptions, *is of the form* $\sigma - \Sigma$, *where* Σ *is a set of labels called* exceptions*;* σ *is called the* main label. *We usually denote complex labels by the letter* λ, *and elements of* Σ *by the letter* ξ. *The* length $|\lambda|$ *of a label is defined as the length of its main label.*
A complex label is called trivial *if* $\varepsilon \in \Sigma$, *and* normalized *if* $\sigma \sqsubseteq_p \xi$ *for every* $\xi \in \Sigma$.

We motivate the intension of complex labels by introducing domains as in Definition 4.2.4:

Definition 5.2.5 *Given a simple label* σ' *and a complex label* $\lambda = \sigma - \Sigma$, *we say that* σ' instantiates a prefix *of* λ *(*$\lambda \sqsubseteq_p \sigma'$*), if* $\sigma \sqsubseteq_p \sigma'$ *but* $\xi \not\sqsubseteq_p \sigma'$ *for any* $\xi \in \Sigma$. *In case* σ *and* σ' *are of the same length, we also write* $\lambda \sqsubseteq \sigma'$ *and call* σ' *an* instance *of* λ. *Finally, we define* $\lambda \sqsubseteq_p \sigma'$, *if some prefix of* σ' *is an instance of* λ.
Let Γ *be a set of ground labels which contains* ε *and is closed under prefixing. We define the* domain of λ wrt Γ, $\mathcal{G}(\lambda, \Gamma)$, *and the* prefix domain of λ wrt Γ, $\mathcal{G}_p(\lambda, \Gamma)$, *as follows:*

$$\mathcal{G}(\lambda, \Gamma) = \{\gamma \in \Gamma : \lambda \sqsubseteq \gamma\}.$$
$$\mathcal{G}_p(\lambda, \Gamma) = \{\gamma \in \Gamma : \lambda \sqsubseteq_p \gamma\}.$$

The $\not\sqsubseteq_p$ in $\xi \not\sqsubseteq_p \sigma'$ is intentional: Exceptions shorter than the main label σ are important in order to express exceptions to the *realizability* of σ (see Section 5.4), but in terms of instantiation, they are to be treated as if they were of the same length as σ (with the remaining positions filled up with $*$). Therefore, σ' does not instantiate λ when some exception ξ is instantiated by a *prefix* of σ'. As a special case, labels with ε as an exception do not have any instance, as we will show below. This is why we call such labels *trivial*: they are vacuous and can be ignored.
From Definition 5.2.5 follow a number of facts analogous to those in Lemma 4.2.7:

Lemma 5.2.6 *Let σ be a simple label, Σ a set of exceptions to σ, λ a complex label, c a constant, and Γ a set of ground labels as in Definition 5.2.5.*

(1) *$\mathcal{G}_p(\lambda,\Gamma)$ is closed under prefixing, and $\mathcal{G}(\lambda,\Gamma)$ is the set of all ground labels in $\mathcal{G}_p(\lambda,\Gamma)$ of length $|\lambda|$.*

(2) *If $\varepsilon \in \Sigma$, then $\mathcal{G}(\sigma - \Sigma,\Gamma) = \mathcal{G}_p(\sigma - \Sigma,\Gamma) = \emptyset$.*

(3) *If $\varepsilon \notin \Sigma$, then $\mathcal{G}(*\sigma - \Sigma,\Gamma) = \bigcup\{c'\mathcal{G}(\sigma - \Sigma_{\langle c'\rangle},\Gamma_{\langle c'\rangle}) : c' \in \Gamma\}$.*

(4) $\mathcal{G}(c\sigma - \Sigma,\Gamma) = \mathcal{G}([c]\sigma - \Sigma,\Gamma) = \begin{cases} c\mathcal{G}(\sigma - \Sigma_{\langle c\rangle},\Gamma_{\langle c\rangle}) & \text{if } \varepsilon \notin \Sigma \text{ and } c \in \Gamma \\ \emptyset & \text{otherwise.} \end{cases}$

Proof: Part (1) can be easily derived from Definition 5.2.5: Consider $\gamma \in \mathcal{G}_p(\lambda,\Gamma)$ and a prefix γ' of γ. Since Γ is closed under prefixing, we can be sure that $\gamma' \in \Gamma$. Now since $\sigma \ {}_p\!\sqsubseteq \gamma$, we certainly have $\sigma \ {}_p\!\sqsubseteq \gamma'$. Furthermore, no prefix of γ (including γ' and all its prefixes) instantiates any of the exceptions in λ, which shows $\gamma' \in \mathcal{G}_p(\lambda,\Gamma)$. The second statement in (1) is obvious.

To show (2), notice that $\varepsilon \sqsubseteq_p \gamma$ for any label γ, so if ε is an exception in Σ, no ground label can be an instance of $\sigma - \Sigma$.

If $\varepsilon \notin \Sigma$, then all exceptions are of length at least 1; we keep this in mind when we transform (for $x = c$ or $x = *$):

$$\begin{aligned}
\mathcal{G}(x\sigma - \Sigma,\Gamma) &= \{c'\gamma \in \Gamma : x\sigma - \Sigma \sqsubseteq c'\gamma\} \\
&= \bigcup_{c'\in\Gamma} \{c'\gamma \in c'\Gamma_{\langle c'\rangle} : x\sigma \sqsubseteq c'\gamma, x'\xi \not\sqsubseteq_p c'\gamma \text{ for any } x'\xi \in \Sigma\} \\
&= \bigcup_{c'\in\Gamma} \{c'\gamma \in c'\Gamma_{\langle c'\rangle} : x \sqsubseteq c', \sigma \sqsubseteq \gamma, x' \not\sqsubseteq c' \text{ or } \xi \not\sqsubseteq_p \gamma \text{ for any } x'\xi \in \Sigma\} \\
&= \bigcup_{c'\in\Gamma} \{c'\gamma \in c'\Gamma_{\langle c'\rangle} : x \sqsubseteq c', \sigma \sqsubseteq \gamma, \xi \not\sqsubseteq_p \gamma \text{ for any } \xi \in \Sigma_{\langle c'\rangle}\}.
\end{aligned}$$

The last step is justified by considering the types of exceptions $x'\xi$: First, exactly when x' is neither c' nor $*$, $x' \not\sqsubseteq c'$ is true, so these exceptions can be disregarded. On the other hand, the exceptions $x'\xi \in \Sigma$ where $x' = c$ or $x' = *$ are exactly the exceptions so that $\xi \in \Sigma_{\langle c'\rangle}$. Now if $x = *$, then $* \sqsubseteq c'$ is always true, so we have:

$$\begin{aligned}
\mathcal{G}(*\sigma - \Sigma,\Gamma) &= \bigcup_{c'\in\Gamma} c'\{\gamma \in \Gamma_{\langle c'\rangle} : \sigma \sqsubseteq \gamma, \xi \not\sqsubseteq_p \gamma \text{ for any } \xi \in \Sigma_{\langle c'\rangle}\} \\
&= \bigcup_{c'\in\Gamma} c'\mathcal{G}(\sigma - \Sigma_{\langle c'\rangle},\Gamma_{\langle c'\rangle}),
\end{aligned}$$

which shows (3).

Finally, if $x = c$ then $x \sqsubseteq c'$ iff $c = c'$, so all labels in $\mathcal{G}(c\sigma - \Sigma, \Gamma)$ must begin with c. If $c \notin \Gamma$, then no other label in Γ can begin with c, since Γ is closed under prefixing; hence $\mathcal{G}(c\sigma - \Sigma, \Gamma) = \emptyset$. Otherwise we write:

$$\mathcal{G}(c\sigma - \Sigma, \Gamma) = c\{\gamma \in \Gamma_{\langle c \rangle} : \sigma \sqsubseteq \gamma, \xi \not\sqsubseteq_p \gamma \text{ for any } \xi \in \Sigma_{\langle c \rangle}\} = c\mathcal{G}(\sigma - \Sigma_{\langle c \rangle}, \Gamma_{\langle c \rangle}).$$

This shows (4). □

In (2) we showed that trivial labels are redundant. Likewise, non-normalized labels specify redundancies: their exceptions have instances which are not even instances of the main label. We would like to eliminate these redundancies and ensure that labels are normalized. We accomplish this through the following procedure, which takes a label $\lambda = \sigma - \Sigma$ and returns its *normalization* $\|\lambda\| = \sigma - \|\Sigma\|_\sigma$:

Algorithm 5.2.7 *Normalization of a label*

Parameters:

σ: the main label

Σ: the set of exceptions to normalize

Returns:

$\|\Sigma\|_\sigma$: the normalization of Σ

begin

foreach exception ξ **do**

if $|\xi| \leq |\sigma|$ **and** $\lfloor \sigma, \xi \rfloor$ exists

set $\xi := \lfloor \sigma, \xi \rfloor$

else

delete ξ

end if

done

return Σ

For instance, the normalization of the label $c_1c_2 - \{c_1c_1, *\}$ is $c_1c_2 - \{c_1\}$. Of course, we must give justice to the name of our algorithm:

Proposition 5.2.8 *The normalization of a complex label is normalized. Normalization does not change a label which is already normalized. Furthermore,* $\mathcal{G}_p(\lambda, \Gamma) = \mathcal{G}_p(\|\lambda\|, \Gamma)$ *for any domain* Γ *as in Definition 5.2.5.*

Proof: Following Definition 5.2.3, we noted that $\sigma \ {}_p\sqsubseteq \lfloor \sigma, \xi \rfloor$ and $\xi \ {}_p\sqsubseteq \lfloor \sigma, \xi \rfloor$. Since all exceptions remaining in $\|\lambda\|$ are of the form $\lfloor \sigma, \xi \rfloor$, the first of these instance relations shows that $\|\lambda\|$ is normalized. Conversely, if λ is normalized, then we already have $\sigma \ {}_p\sqsubseteq \xi$ and trivially $\xi \ {}_p\sqsubseteq \xi$. There obviously cannot be any longer or more general label for which this holds, so we conclude $\xi = \lfloor \sigma, \xi \rfloor$. Furthermore, $\sigma \ {}_p\sqsubseteq \xi$ implies that ξ is no longer than σ, which prevents it from deletion. Altogether every exception in Σ remains unchanged, hence $\Sigma = \|\Sigma\|_\sigma$.

Finally, let us prove the set equality $\mathcal{G}_p(\lambda, \Gamma) = \mathcal{G}_p(\|\lambda\|, \Gamma)$. All ground instances of either set must instantiate the main label: $\sigma \ {}_p\sqsubseteq \gamma$, and they are excluded iff they instantiate an exception in Σ and $\|\Sigma\|_\sigma$, respectively. So we will show that out of $\mathcal{G}_p(\sigma, \Gamma)$, Σ and $\|\Sigma\|_\sigma$ exclude the same set of ground instances. So assume that $\sigma \ {}_p\sqsubseteq \gamma$ and $\xi \sqsubseteq_p \gamma$ for some $\xi \in \Sigma$. First, this implies that $|\xi| \leq |\gamma| \leq |\sigma|$, so any exception ξ which is strictly longer than σ does not exclude any ground instances. Secondly, let γ' be the prefix of γ of length $|\xi|$. Then $\sigma \ {}_p\sqsubseteq \gamma'$ and $\xi \sqsubseteq \gamma'$, so from the remark following Definition 5.2.3, we infer that $\lfloor \sigma, \xi \rfloor$ exists and $\lfloor \sigma, \xi \rfloor \sqsubseteq \gamma'$, which further implies $\lfloor \sigma, \xi \rfloor \sqsubseteq_p \gamma$. It necessarily follows that all ξ deleted by the algorithm (because $\lfloor \sigma, \xi \rfloor$ does not exist) do not have any ground instances; and for those ξ not deleted, all ground instances of ξ are instances of $\lfloor \sigma, \xi \rfloor$, the exception replacing them. Therefore, all ground instances in Σ are ground instances in $\|\Sigma\|_\sigma$. Conversely, consider any ground instance γ in $\|\Sigma\|_\sigma$. Then $\lfloor \sigma, \xi \rfloor \sqsubseteq_p \gamma$ and $|\xi| \leq |\sigma|$ for some $\xi \in \Sigma$. Now the second instance relation mentioned at the beginning of this proof can be sharpened to $\xi \sqsubseteq \lfloor \sigma, \xi \rfloor$, and using transitivity, we conclude $\xi \sqsubseteq_p \gamma$. Hence, γ is also a ground instance in Σ. This concludes the proof that $\mathcal{G}_p(\lambda, \Gamma) = \mathcal{G}_p(\|\lambda\|, \Gamma)$. □

From now on, we will implicitly assume that all complex labels are normalized and trivial labels are removed from any set of labels as they occur. (Some of the properties we will define must be explicitly verified to be unaffected by—or invariant wrt—normalization and removing trivial labels. We will not fail to do so.)

Definition 5.2.9 *The* most general unifier (mgu) *of two labels* $\lambda_1 = \sigma_1 - \Sigma_1$, $\lambda_2 = \sigma_2 - \Sigma_2$, *so that* (σ_1, σ_2) *exists, is defined and denoted by* $(\lambda_1, \lambda_2) = (\sigma_1, \sigma_2) - \|\Sigma_1 \cup \Sigma_2\|_{(\sigma_1, \sigma_2)}$. *If additionally* $\mathcal{G}((\lambda_1, \lambda_2), D^*)$ *is nonempty, we say that* λ_1 *and* λ_2 unify.

Notice that even if (σ_1, σ_2) exists, the mgu (λ_1, λ_2) may be vacuous, i.e. $\mathcal{G}((\lambda_1, \lambda_2), D^*) = \emptyset$. Consider $\lambda_1 = * - \{c\}$ and $\lambda_2 = c$, then $(\lambda_1, \lambda_2) = (*, c) - \{c\} = c - \{c\}$ which, though normalized, has no ground instances.

As expected, the mgu of two labels represents the intersection of their domains:

Proposition 5.2.10 *For any two labels $\lambda_1 = \sigma_1 - \Sigma_1$, $\lambda_2 = \sigma_2 - \Sigma_2$ and any Γ as in Definition 5.2.5, we have $\mathcal{G}((\lambda_1, \lambda_2), \Gamma) = \mathcal{G}(\lambda_1, \Gamma) \cap \mathcal{G}(\lambda_2, \Gamma)$.*

Proof: The common ground instances γ of both λ_1 and λ_2 satisfy $\sigma_1 \sqsubseteq \gamma$, $\sigma_2 \sqsubseteq \gamma$, $\xi \not\sqsubseteq_p \gamma$ for any $\xi \in \Sigma_1$, and $\xi \not\sqsubseteq_p \gamma$ for any $\xi \in \Sigma_2$. This can be written equivalently as: $(\sigma_1, \sigma_2) \sqsubseteq \gamma$, and $\xi \not\sqsubseteq_p \gamma$ for any $\xi \in \Sigma_1 \cup \Sigma_2$, which says exactly that γ is a ground instance of $(\sigma_1, \sigma_2) - \Sigma_1 \cup \Sigma_2)$. Now (λ_1, λ_2) is obtained by normalizing this label; as we showed in Proposition 5.2.8, normalization leaves the set of ground instances unchanged, whence the theorem follows. □

Complex labels are designed for use in the assertions constituting the semantic structures, and we are now ready to modify the definition of assertions accordingly. We also extend the notion of subscripted sets, which not only includes the familiar $M_{\langle c \rangle}$ and $M_{\langle \gamma \rangle}$, but also $M_{\langle * \rangle}$ and $M_{\langle \sigma \rangle}$, for arbitrary simple labels σ:

Definition 5.2.11 *An* assertion (with a complex label) *is a formula of the form $\lambda\, a$, where λ is a complex label and a is a propositional atom. A constant c* exists *(or* occurs*) in a set of assertions M, if M contains an assertion with a* nontrivial *label of the form $c\sigma - \Sigma$ (where c is an existential constant).*

For a set M of assertions without trivial labels and a constant or wildcard *x, we define $M_{\langle x \rangle}$ as the set $\{(\sigma - \Sigma_{\langle x \rangle})\, a : (x\sigma - \Sigma)\, a \in M$ or $(*\sigma - \Sigma)\, a \in M, \varepsilon \notin \Sigma_{\langle x \rangle}\}$. For $\sigma = x_1 \cdots x_k$ and an* arbitrary *set M of assertions, we define $M_{\langle \varepsilon \rangle}$ as M without trivial labels, and $M_{\langle \sigma \rangle} = (((M_{\langle \varepsilon \rangle})_{\langle x_1 \rangle}) \ldots)_{\langle x_k \rangle}$.*

Notice that $M_{\langle \varepsilon \rangle}$ is formally, but not practically, distinct from M, as labels with trivial labels contain no information.

Subscripting is affected by exceptions, in that they can make an assertion disappear from $M_{\langle c \rangle}$. For instance, consider the set $M = \{(*1 - \{2\})\, a\}$. (That is, $\Sigma = \{2\}$.) Upon subscripting by the constant 1, we get $M_{\langle 1 \rangle} = \{1\, a\}$. But upon subscripting by 2, we have $\Sigma_{\langle 2 \rangle} = \{\varepsilon\}$, so the assertion $(1 - \{\varepsilon\})\, a$ has a trivial label and must be omitted, which yields $M_{\langle 2 \rangle} = \emptyset$. This is in line with the intended reading of the original assertion which

says: "In all instances *but 2*, the constant 1 exists and a is *true* in it." We will examine the generalization to $M_{\langle\sigma\rangle}$ below in Proposition 5.2.15.
A simple observation states that subscripting is invariant under normalization:

Proposition 5.2.12 *Let M' be the set obtained from M by normalizing all its labels according to Algorithm 5.2.7. Then for any simple label σ, the set $M'_{\langle\sigma\rangle}$ is identical to the set obtained by normalizing all labels in $M_{\langle\sigma\rangle}$.*

Proof: It is easy to see that the general result follows iteratively from the special case where we have a set M without trivial labels, and $\sigma = x$ is a one-element label. According to Definition 5.2.11, $M_{\langle x\rangle}$ collects labels $(\sigma - \Sigma_{\langle x\rangle})\ a$ so that $(x'\sigma - \Sigma)\ a \in M$, whereas $x' = x$ or $x' = *$, and $\varepsilon \notin \Sigma_{\langle x\rangle}$. These labels get normalized to $(\sigma - \|\Sigma_{\langle x\rangle}\|_\sigma)\ a$. On the other hand, if the label in M gets normalized first, it becomes $(x'\sigma - \|\Sigma\|_{x'\sigma})\ a$, and upon subscripting, the corresponding label in $M'_{\langle x\rangle}$ is $(\sigma - (\|\Sigma\|_{x'\sigma})_{\langle x\rangle})\ a$. We remark that $\varepsilon \notin \Sigma_{\langle x\rangle}$ iff $\varepsilon \notin (\|\Sigma\|_{x'\sigma})_{\langle x\rangle}$, which follows from the equivalent observation that neither x nor $*$ are exceptions in Σ iff they are not exceptions in $\|\Sigma\|_{x'\sigma}$. (These two one-element exceptions could not have been erased in the normalization of Σ, because the main label $x\sigma$ must be of length at least 1 itself, and both x and $*$ unify with $x\sigma$. The only change which may occur is when $x' = x = c$ and $*$ is an exception in Σ; this exception would be instantiated to c, which is still equal to x.) It is now clear that corresponding assertions are collected into $M_{\langle x\rangle}$ and $M'_{\langle x\rangle}$. It only remains to be shown that $\|\Sigma_{\langle x\rangle}\|_\sigma = (\|\Sigma\|_{x'\sigma})_{\langle x\rangle}$. Thus consider any exception $x''\xi$ in Σ. A corresponding label ξ is found in $\Sigma_{\langle x\rangle}$ iff $x'' = x$ or $x'' = *$. Let us compare the criteria for retaining and modifying ξ and $x''\xi$ upon normalizing Σ and $\Sigma_{\langle x\rangle}$, respectively:

- $|\xi| \leq |\sigma|$; this is the case iff $x''\xi \leq |x'\sigma|$.
- $\lfloor\sigma, \xi\rfloor$ exists; which is iff $\lfloor x'\sigma, x''\xi\rfloor$ exists, as both x' and x'' are instantiated by x.
- The normalized exception becomes $\lfloor\sigma, \xi\rfloor$; compare this with $\lfloor x'\sigma, x''\xi\rfloor = (x', x'')\lfloor\sigma, \xi\rfloor$.

Upon subscripting by x, this latter exception gets shortened to $\lfloor\sigma, \xi\rfloor$, which completes the proof of $\|\Sigma_{\langle x\rangle}\|_\sigma = \|\Sigma\|_{x'\sigma})_{\langle x\rangle}$. □
As a consequence, normalized labels remain normalized after subscripting:

Proposition 5.2.13 *If all labels of M are normalized, then so are all labels of $M_{\langle\sigma\rangle}$.*

Proof: Let M' be the normalization of M as in Proposition 5.2.12; then $M'_{\langle\sigma\rangle}$ is established as the normalization of $M_{\langle\sigma\rangle}$. By Proposition 5.2.8, if M is normalized, then $M = M'$. But then also $M_{\langle\sigma\rangle} = M'_{\langle\sigma\rangle}$, so $M_{\langle\sigma\rangle}$ is invariant under normalization, which means that it is normalized. □

Subscripting M by $*$ or by any other label σ involving wildcards seems to give rise to novel sets $M_{\langle\sigma\rangle}$—different from any sets $M_{\langle\gamma\rangle}$ we can produce with ground labels γ. But as it turns out, there are no novel sets at all:

Proposition 5.2.14 *Any simple label σ has a ground instance γ so that $M_{\langle\sigma\rangle} = M_{\langle\gamma\rangle}$.*

Proof: We find the ground label γ simply by replacing all occurrences of $*$ in σ with a constant $c \in D$ which does not occur anywhere in M. We do the proof for $\sigma = *$; it generalizes to arbitrary labels in an obvious way. First, since c is new to M, no assertion $(c\sigma - \Sigma)\, a$ exists in M, so all "candidates" in M are of the form $(*\sigma - \Sigma)\, a$. Secondly, we have $\Sigma_{\langle c\rangle} = \Sigma_{\langle *\rangle}$, since no exception in M contains c either. This shows that $M_{\langle c\rangle}$ and $M_{\langle *\rangle}$ in Definition 5.2.11 are identical. □

As the proof demonstrates, $M_{\langle *\rangle}$ can be viewed as the set of assertions which have to hold in a "generic" instance of $*$, or in an instance that may be introduced into M at a later time. (Note that the same is true for the exceptions of a label: only the "generic" exceptions beginning with $*$ are carried over into $M_{\langle *\rangle}$, with the initial $*$ omitted. "Specialized" exceptions beginning with constants c have no influence on $M_{\langle *\rangle}$. Instead, these exceptions will require special consideration by taking $M_{\langle c\rangle}$. (We touched upon this issue in Example 5.1.1; we will address it in generality in the next section.)

Many of our previous informal remarks on subscripted sets are summarized in the following important closed-form characterization of $M_{\langle\sigma\rangle}$ (see the end of this section for an illustrative example):

Proposition 5.2.15 *For any set M of assertions without trivial labels and any simple label σ, $M_{\langle\sigma\rangle}$ is the set of assertions $(\sigma'' - \Sigma'')\, a$ for which all of the following are true:*

(1) *M contains an assertion of the form $(\sigma'\sigma'' - \Sigma)\, a$.*

(2) *$\sigma' \sqsubseteq \sigma$.*

(3) *$\xi \not\sqsubseteq_p \sigma$ for any $\xi \in \Sigma$.*

(4) $\Sigma'' = \Sigma_{\langle\sigma\rangle} = \{\xi'' : \xi'\xi'' \in \Sigma, \xi' \sqsubseteq \sigma\}$.

Proof: For $M_{\langle\varepsilon\rangle}$ all these are trivially true, with $\sigma = \sigma' = \varepsilon$, $\Sigma'' = \Sigma$, and $\xi' = \varepsilon$. Notice that condition (3) is trivially true, since M is devoid of assertions with trivial labels.
Now assuming the proposition valid for σ, we will prove it for σx. By Definition 5.2.11, $M_{\langle\sigma x\rangle}$ is the set of assertions $(\sigma'' - \Sigma''_{\langle x\rangle})\, a$ so that $\varepsilon \notin \Sigma''_{\langle x\rangle}$, and $(x\sigma'' - \Sigma'')\, a \in M_{\langle\sigma\rangle}$ or $(*\sigma'' - \Sigma'')\, a \in M_{\langle\sigma\rangle}$. By applying the induction hypothesis, we reason:

- M has an assertion $(\sigma' x \sigma'' - \Sigma)\, a$ or $(\sigma' * \sigma'' - \Sigma)\, a$, and $\sigma' \sqsubseteq \sigma$. We substitute $\tilde{\sigma}$ in place of $\sigma' x$ or $\sigma' *$, and write the assertion as $(\tilde{\sigma}\sigma'' - \Sigma)\, a$, and $\tilde{\sigma} \sqsubseteq \sigma x$. (Note that all such $\sigma' x$ and $\sigma' *$ are the only labels in M which are instantiated by σx, and no other assertions in M can give rise to any other assertion $(\sigma'' - \Sigma''_{\langle x\rangle})\, a$ in $M_{\langle\sigma x\rangle}$. So we have established conditions (1) and (2).

- Using Definition 4.2.6, we transform:

$$\begin{aligned}\Sigma''_{\langle x\rangle} = (\Sigma_{\langle\sigma\rangle})_{\langle x\rangle} &= \{\xi'' : x\xi'' \in \Sigma_{\langle\sigma\rangle} \text{ or } *\xi'' \in \Sigma_{\langle\sigma\rangle}\} \\ &= \{\xi'' : \xi' x \xi'' \in \Sigma \text{ or } \xi' * \xi'' \in \Sigma, \xi' \sqsubseteq \sigma\} \\ &= \{\xi'' : \tilde{x} i \xi'' \in \Sigma, \tilde{x} i \sqsubseteq \sigma x\}.\end{aligned}$$

  This shows condition (4).

- $\xi \not\sqsubseteq_p \sigma$ for any $\xi \in \Sigma$. This is the case iff $\xi \not\sqsubseteq_p \sigma x$ for all exceptions ξ of length at most $|\sigma|$ in Σ. We need to show this ($\tilde{\xi} \not\sqsubseteq \sigma x$) also for all exceptions $\tilde{\xi} \in \Sigma$ of length $|\sigma| + 1 = |\sigma x|$. But we stipulated that $\varepsilon \notin \Sigma''_{\langle x\rangle}$; by (4), this is equivalent to $\tilde{\xi} \not\sqsubseteq \sigma x$ for all exceptions $\tilde{\xi} \in \Sigma$, which is exactly what we needed to show. (The length requirement $|\tilde{\xi}| = |\sigma x|$ is implicit in the $\sqsubseteq$ relation.) This establishes condition (3).

We have thus shown that the proposition holds for $M_{\langle\sigma x\rangle}$, provided it does for $M_{\langle\sigma\rangle}$; the principle of induction on simple labels entails the proof. □

Remark 5.2.16 *In the settings of Proposition 5.2.15, we say that the assertion* $(\sigma'\sigma'' - \Sigma)\, a$ contributes *an assertion, namely* $(\sigma'' - \Sigma'')\, a$, *to* $M_{\langle\sigma\rangle}$. *By joining conditions (2) and (3) together, we see that an assertion* $\lambda\, a$ *contributes to* $M_{\langle\sigma\rangle}$ *exactly if* $\lambda \sqsubseteq_p \sigma$.

Example 5.2.17 The characterization of Proposition 5.2.15 is best understood from the point of the original set M and its assertions contributing into $M_{\langle\sigma\rangle}$. Consider the set $M = \{(** - \{*1\})\, p, *1\ \neg p, (*** - \{**2\})\, q, **2\ \neg q, (** - \{11\})\, r, 11\ \neg r\}$ and $\sigma = *1$:

- Since $11 \not\sqsubseteq \sigma$ in (2), $11\ \neg r$ does not contribute to $M_{\langle\sigma\rangle}$.
- Since $*1 \sqsubseteq \sigma$, $*1\ \neg p$ contributes the assertion $\varepsilon\ \neg p$ to $M_{\langle\sigma\rangle}$. (We have $\sigma'' = \varepsilon$.)
- Since $** \sqsubseteq \sigma$, $**2\ \neg p$ contributes the assertion $2\ \neg q$ to $M_{\langle\sigma\rangle}$. (We have $\sigma'' = 2$.)
- Although $** \sqsubseteq \sigma$, $(** - \{*1\})\ p$ does not contribute to $M_{\langle\sigma\rangle}$, because its label has an exception $*1 \sqsubseteq_p \sigma$, violating (3).
- By contrast, $11 \not\sqsubseteq_p \sigma$, so $(** - \{11\})\ r$ contributes the assertion $\varepsilon\ r$ to $M_{\langle\sigma\rangle}$.
- Likewise, $**2 \not\sqsubseteq_p \sigma$, because σ is shorter than $**2$. However, this exception is of the form $\xi'\xi''$ so that $\xi' = ** \sqsubseteq \sigma$, so the assertion contributed by $(*** - \{**2\})\ q$ is $(* - \{2\})\ q$, with an exception 2 according to (4).

To summarize, the subscripted set is $M_{\langle\sigma\rangle} = \{\varepsilon\ \neg p, (* - \{2\})\ q, 2\ \neg q, \varepsilon\ r\}$. See next section for some more examples.

5.3 Exception-Generated Instances

In this section, we explore the "geometry" of labels with exceptions and how they lend themselves to reasoning "monolithically". By considering generic operators and properties defined on formulas and sets of assertions, we will establish general principles in regard to subscripting and instantiating, as well as the important property of monotonicity. The results of this section will be applied throughout the remainder of this chapter.

Proposition 5.2.15, a closed-form characterization of subscripted sets, already steers us away from the position-wise characterizations of Chapter 4 (e.g. for the $\models_l$ relation). We observe that subscripting is a monotone operation, at least in one sense:

Corollary 5.3.1 *For any two sets of assertions M, M', if $M \subseteq M'$, then $M_{\langle\sigma\rangle} \subseteq M'_{\langle\sigma\rangle}$ for any simple label σ.*

This is because $M_{\langle\sigma\rangle}$ is generated "element-wise": Whether an assertion M contributes an assertion into $M_{\langle\sigma\rangle}$ does not depend on any other assertions which exist in M, or which may later be added to M. But is subscripting also monotone with respect to σ, that is, if $\sigma \sqsubseteq \sigma'$, does $M_{\langle\sigma'\rangle}$ include all assertions in $M_{\langle\sigma\rangle}$? For sets of assertions with simple labels, the answer is yes:

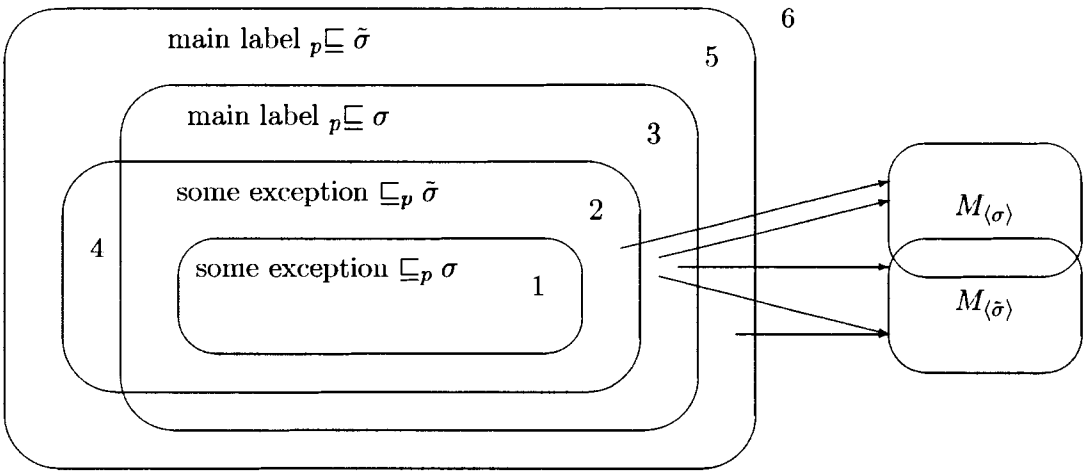

Figure 5.3: *How $M_{\langle\sigma\rangle}$ and $M_{\langle\tilde{\sigma}\rangle}$ are formed from some set M, for $\sigma \sqsubseteq \tilde{\sigma}$.*

Proposition 5.3.2 *If M is a set of assertions with* simple *labels (that is, with empty sets of exceptions in every assertion), and $\sigma \sqsubseteq \tilde{\sigma}$, then $M_{\langle\sigma\rangle} \subseteq M_{\langle\tilde{\sigma}\rangle}$.*

Proof: The conditions in Proposition 5.2.15 for $\sigma''\ a$ to be an assertion in $M_{\langle\sigma\rangle}$ reduce to:

- M contains an assertion of the form $\sigma'\sigma''\ a$.
- $\sigma' \sqsubseteq \sigma$.

(It is obvious that Σ'' is empty, i.e. all labels in $M_{\langle\sigma\rangle}$ are simple.) From $\sigma' \sqsubseteq \sigma$ and $\sigma \sqsubseteq \tilde{\sigma}$ we also get $\sigma' \sqsubseteq \tilde{\sigma}$ by transitivity, so $\sigma''\ a$ is also an assertion in $M_{\langle\tilde{\sigma}\rangle}$. This proves the proposition. □

In sets of assertions with exceptions, however, the monotonicity property is lost. Let us consider a moderately intricate example:

Example 5.3.3 For $M = \{*1\ p, *2\ \neg p, 1*\ q, 2*\ \neg q, (** - \{*1\})\ r, (** - \{1*\})\ s\}$, we have:

$$M_{\langle**\rangle} = \{r, s\} \qquad M_{\langle1*\rangle} = \{q, r\}$$
$$M_{\langle*1\rangle} = \{p, s\} \qquad M_{\langle11\rangle} = \{p, q\}.$$

None of these four sets is comparable with any other.

Now consider the general case, a set M of assertions and two labels $\sigma \sqsubseteq \tilde{\sigma}$. The assertions in M are categorized as shown in the Venn diagram of Figure 5.3; recall also the characterization of $M_{\langle\sigma\rangle}$ provided in Proposition 5.2.15. We analyze which categories of assertions in M contribute to $M_{\langle\sigma\rangle}$ and which to $M_{\langle\tilde{\sigma}\rangle}$:

- If the main label $\sigma'\sigma''$ does not have σ instantiating its prefix σ' (Areas 4–6), it does not contribute to $M_{\langle\sigma\rangle}$. (Condition (2) in Proposition 5.2.15 is violated.)
- If $\sigma' \sqsubseteq \sigma$ but an exception $\xi \in \Sigma$ is instantiated by a prefix of σ (Area 1), the assertion does not contribute to $M_{\langle\sigma\rangle}$ either. (Condition (3) in Proposition 5.2.15 is violated.)
- In all other cases (Areas 2, 3), the assertion contributes to $M_{\langle\sigma\rangle}$.
- Presuming that all labels are normalized and that $\sigma \sqsubseteq \tilde{\sigma}$, we get the set inclusions shown in Figure 5.3 with regard to the above three types of assertions, and the corresponding types of assertions according to $\tilde{\sigma}$.

The assertions which contradict our assumption $M_{\langle\sigma\rangle} \subseteq M_{\langle\tilde{\sigma}\rangle}$ are those found in Area 2. Their main label (or rather, a prefix of it) is instantiated by both σ and $\tilde{\sigma}$, and some assertion has an exception instantiated by a prefix of $\tilde{\sigma}$ which is not general enough to be instantiated by a prefix of σ.

Finally, the same assertion may contribute different exceptions into the corresponding assertion in $M_{\langle\sigma\rangle}$ and $M_{\langle\tilde{\sigma}\rangle}$, respectively. To see the difference, consider Condition (4) in Proposition 5.2.15. The exceptions ξ'' included in Σ'' stem from those exceptions $\xi'\xi''$ in Σ whose prefix ξ' is instantiated by σ. Since there may be exceptions where ξ' is instantiated by $\tilde{\sigma}$ but not σ, the corresponding assertion in $M_{\langle\tilde{\sigma}\rangle}$ may have exceptions not found in the assertion in $M_{\langle\sigma\rangle}$. In Example 5.3.3, we find this exemplified by the assertion $(** - \{1*\})\, s$: it contributes the assertion $* s$ (without exceptions) into $M_{\langle *\rangle}$, and the (vacuous) assertion $(* - \{*\})\, s$ into $M_{\langle 1\rangle}$.

In either case, these violations of the monotonicity property stem from exceptions occurring in the labels in M. Therefore, the nature of exceptions deserves a thorough study which we will provide in the remainder of this section. We introduce the following terminology:

Definition 5.3.4 *Given a set M of assertions and a simple label σ, the set of* exception-generated instances (egi) *of σ in M is defined recursively as the smallest set for which the following is true:*

- *σ is an exception-generated instance of σ in M.*
- *If σ' is an exception-generated instance of σ in M, and ξ is an exception occurring in some assertion in M, truncated to the length of σ[2], then $\lceil\sigma', \xi\rceil$ is an exception-generated instance of σ in M.*

This recursive definition gives rise to another induction principle, that of *induction over the exception-generated instances of a label σ*:

Corollary 5.3.5 *Given a property $Q(\sigma')$ defined on a label σ and its exception-generated instances in M, if $Q(\sigma)$ is true, and for all exception-generated instances σ', $Q(\sigma')$ implies $Q(\lceil\sigma', \xi\rceil)$ for all exceptions ξ occurring in M, truncated to length $|\sigma|$, so that $\lceil\sigma', \xi\rceil$ exists, then $Q(\sigma')$ is true for all exception-generated instances σ'.*

Due to the nature of mgus we obtain the following easy conclusions:

Corollary 5.3.6 *With σ and M as above:*

(1) *Every exception-generated instance of σ in M can be written as the iterated upper mgu $\lceil\sigma, \xi_1, \ldots, \xi_n\rceil$, where ξ_i, $i = 1, \ldots, n$, are exceptions in M, truncated to the length of σ, and $n \geq 0$. Conversely, every existing such unifier is an exception-generated instance.*

(2) *If σ_1 and σ_2 are exception-generated instances of σ in M, then so is (σ_1, σ_2), provided this mgu exists.*

(3) *Every instance σ' of σ has a uniquely defined least general exception-generated instance $\hat{\sigma}$ so that $\hat{\sigma} \sqsubseteq \sigma'$; that is, for all exception-generated instances $\tilde{\sigma}$ with $\tilde{\sigma} \sqsubseteq \sigma'$, we have $\tilde{\sigma} \sqsubseteq \hat{\sigma}$.*

(4) *For σ' and its unique exception-generated instance $\hat{\sigma}$ in (3), $M_{\langle\hat{\sigma}\rangle} \subseteq M_{\langle\sigma'\rangle}$*

Proof: Part (1) is just an iterative version of the recursive Definition 5.3.4. For (2), write $\sigma_1 = \lceil\sigma, \xi_1, \ldots, \xi_n\rceil$ and $\sigma_2 = \lceil\sigma, \xi_{n+1}, \ldots, \xi_{n+m}\rceil$ according to (1). Then we can easily show that $(\sigma_1, \sigma_2) = \lceil\sigma, \xi_1, \ldots, \xi_{n+m}\rceil$, so the mgu itself is also an egi, as it matches (1). In (3), σ' instantiates at least one egi of σ, namely σ itself. Now find all egis $\tilde{\sigma}$ instantiated by σ', and take their mgu $\hat{\sigma}$ (which exists since all $\tilde{\sigma}$ have a common instance); we claim this to

[2] *That is, either ξ is an exception of length at most $|\sigma|$, or it is the length-$|\sigma|$ prefix of a longer exception.*

be the sought least general egi. First, being the mgu of egis of σ, $\hat{\sigma}$ is also an egi of σ by (2). Secondly, since all $\tilde{\sigma}$ are instantiated by σ', so is $\hat{\sigma}$. Finally, every egi instantiated by σ' is instantiated by their mgu $\hat{\sigma}$, which shows that $\hat{\sigma}$ is the least general egi.

To prove (4), we will show that every assertion in $M_{\langle\hat{\sigma}\rangle}$ according to Proposition 5.2.15 is also in $M_{\langle\sigma'\rangle}$. Let $(\hat{\sigma}''-\Sigma_{\langle\hat{\sigma}\rangle})\,a \in M_{\langle\hat{\sigma}\rangle}$ be such an assertion, and $(\hat{\sigma}'\hat{\sigma}''-\Sigma)\,a$ a corresponding assertion in M which contributed it. Then $\hat{\sigma}' \sqsubseteq \hat{\sigma} \sqsubseteq \sigma'$, and $\xi \not\sqsubseteq_p \hat{\sigma}$ for any $\xi \in \Sigma$. If Σ contained an exception so that $\xi \sqsubseteq_p \sigma'$, then $\lceil\sigma', \xi\rceil$ would exist and be an egi of σ'. We cannot have $\lceil\sigma', \xi\rceil \sqsubseteq \hat{\sigma}$ as this would imply $\xi \sqsubseteq_p \hat{\sigma}$, contradicting our assumption. But then $\hat{\sigma}$ cannot be the least general egi, which contradicts our choice of $\hat{\sigma}$ in (3). Therefore, $\xi \not\sqsubseteq_p \sigma'$ for all exceptions $\xi \in \Sigma$. Finally, $\Sigma_{\langle\hat{\sigma}\rangle}$ is the set of all ξ'' stemming from some exception $\xi'\xi'' \in \Sigma$ so that $\xi' \sqsubseteq \hat{\sigma}$, but then also $\xi' \sqsubseteq \sigma'$, which shows that $\xi'' \in \Sigma_{\langle\sigma'\rangle}$. Conversely, assume there existed an exception ξ'' in $\Sigma_{\langle\sigma'\rangle}$ but not $\Sigma_{\langle\hat{\sigma}\rangle}$. This exception must derive from an exception $\xi'\xi''$ in Σ so that $\sigma' \sqsubseteq \xi'$ but $\hat{\sigma} \not\sqsubseteq \xi'$. Then once again we would have found an egi $\lceil\sigma', \xi'\rceil = \xi'$ which is not instantiated by $\hat{\sigma}$, contradicting our choice in (3). Hence $\Sigma_{\langle\sigma'\rangle}$ and $\Sigma_{\langle\hat{\sigma}\rangle}$ must be identical, and we have verified in accordance with Proposition 5.2.15 that $(\hat{\sigma}'\hat{\sigma}'' - \Sigma)\,a$ contributes the same assertion $(\hat{\sigma}'' - \Sigma_{\langle\hat{\sigma}\rangle})\,a$ to $M_{\langle\sigma'\rangle}$ as it does to $M_{\langle\hat{\sigma}\rangle}$. □

Part (4) is actually quite remarkable: it gives us a "local" monotonicity property in the hierarchy of instances of σ. Let us illustrate this on our Example 5.3.3. For instance, we have $M_{\langle *2\rangle} = \{\neg p, r, s\}$ which, as claimed, is a superset of $M_{\langle **\rangle}$, as $**$ is the only and hence least general exception-generated instance of $**$ instantiated by $*2$. In the same way, $M_{\langle 33\rangle} = \{r, s\} \supseteq M_{\langle **\rangle}$; notice that the constant 3 does not occur anywhere in M, so this example also illustrates (the proof of) Proposition 5.2.14 which says that $M_{\langle 33\rangle}$ and $M_{\langle **\rangle}$ are identical.

Proposition 5.3.7 *If $\sigma'\sigma''$ is an exception-generated instance of $\sigma'_0\sigma''_0$ in M and $|\sigma'| = |\sigma'_0| = k$, then σ' is an exception-generated instance of σ'_0 in M, and σ'' is an exception-generated instance of σ''_0 in $M_{\langle\sigma'\rangle}$.*

Proof: Write this egi as $\sigma'\sigma'' = \lceil\sigma'_0\sigma''_0, \xi'_1\xi''_1, \ldots, \xi'_m\xi''_m, \xi'_{m+1}, \ldots, \xi'_n\rceil$, according to Corollary 5.3.6, part (1). If $m = n = 0$, then $\sigma'\sigma'' = \sigma'_0\sigma''_0$, and the proposition is trivially true (even when $M_{\langle\sigma'_0\rangle}$ is empty!). So let us assume that $\sigma'\sigma''$ is generated using at least one exception. Wlog we sort the exceptions so that $\xi'_1\xi''_1, \ldots, \xi'_m\xi''_m$ are all of length greater than k, and $\xi'_{m+1}, \ldots, \xi'_n$ are of length at most k. Furthermore, we split the first m exceptions so that $|\xi'_i| = k$, $\xi''_i \neq \varepsilon$ for $i \leq m$, and we declare $\xi''_i = \varepsilon$ for $i > m$. It follows that

$\sigma' = \lceil \sigma'_0, \xi'_1, \ldots, \xi'_n \rceil$ (hence σ' is an egi of σ'_0 in M), and $\sigma'' = \lceil \sigma''_0, \xi''_1, \ldots, \xi''_m \rceil$. Furthermore, since $\xi'_i \xi''_i \sqsubseteq_p \sigma'\sigma''$ for all the exceptions, we obtain $\xi'_i \sqsubseteq \sigma'$ for $i = 1, \ldots, m$. This shows that $\xi''_1, \ldots, \xi''_m$ all exist as exceptions in $M_{\langle \sigma' \rangle}$. We conclude that σ'' is an egi of σ''_0 in $M_{\langle \sigma' \rangle}$, as claimed. □

The converse is not true, for a good reason: Consider a set M containing $**-\{11\}$ as its only label, and let $\sigma' = 1$, $\sigma'' = *$. Obviously $\sigma'\sigma''$ is not an exception-generated instance of $**$ in M. However, 1 is an exception-generated instance of $*$ in M, and $*$ is trivially an exception-generated instance of itself in $M_{\langle 1 \rangle}$. The deeper reason behind this is that the exception-generated instances of a label do not constitute a block structure in the sense described in the introductory section: not every combination of 1 and $*$ in the above example results in an exception-generated instance. Accordingly, the set of exception-generated instances of $\sigma'_0\sigma''_0$ is not necessarily the Cartesian product of the sets of exception-generated instances of σ'_0 and σ''_0.

Next, we will state our fundamental principle of reasoning over the hierarchy of exception-generated instances of a given label σ in a given set of assertions M. Consider a generic property $Q(\sigma, M)$ satisfying the following recursion:

> $Q(\sigma, M)$ iff $Q(\varepsilon, M_{\langle \sigma \rangle})$ and for all exceptions ξ in M, truncated to the length of σ, $\lceil \sigma, \xi \rceil$ does not exist, or $\lceil \sigma, \xi \rceil = \sigma$, or $Q(\lceil \sigma, \xi \rceil, M)$. (5.2)

We say that $Q(\sigma, \cdot)$ is *monotone (wrt $\subseteq$)* if whenever $M \subseteq M'$, $Q(\sigma, M)$ implies $Q(\sigma, M')$. (See Definition 2.1.7.) We understand this to be universally qualified over all labels σ, unless explicitly stated otherwise.

First of all, let us ensure that the evaluation of $Q(\sigma, M)$ is terminating:

Proposition 5.3.8 *If $Q(\varepsilon, M)$ is finitely decidable for every set M of assertions, then so is $Q(\sigma, M')$ for any simple label σ and any set M'.*

Proof: If σ is ground, it is its only instance (modulo an irrelevant distinction between existential and conditional constants). Furthermore, any exception ξ is truncated to the length of σ, so if the upper mgu $\lceil \sigma, \xi \rceil$ exists at all, it is equal to σ, and no recursive evaluations ensue according to (5.2). If σ does contain wildcards and none of them are instantiated in any (truncated) exception ξ so that $\lceil \sigma, \xi \rceil$ exists, then this unifier is again equal to σ, and no recursive evaluation is performed. Hence all recursive evaluations of $Q(\lceil \sigma, \xi \rceil, M)$ occur on strict instances $\lceil \sigma, \xi \rceil$ of σ, so the evaluation must terminate. □

Theorem 5.3.9 *Statements (1) through (3) below are equivalent; so are (4) and (5). If $Q(\sigma, \cdot)$ is monotone, all statements (1) through (5) below are equivalent:*

(1) *$Q(\sigma, M)$ is true according to (5.2).*

(2) *$Q(\hat{\sigma}, M)$ is true according to (5.2), for every exception-generated instance $\hat{\sigma}$ of σ.*

(3) *$Q(\varepsilon, M_{\langle\hat{\sigma}\rangle})$ for every exception-generated instance $\hat{\sigma}$ of σ.*

(4) *$Q(\varepsilon, M_{\langle\sigma'\rangle})$ for every instance σ' of σ.*

(5) *$Q(\varepsilon, M_{\langle\gamma\rangle})$ for every ground instance γ of σ.*

Proof: Clearly (1) is a special case of (2). To prove (2) from (1), we use induction over the egis of σ according to Corollary 5.3.5: $Q(\sigma, M)$ is given, and given any label σ', $Q(\sigma', M)$ implies $Q(\lceil\sigma', \xi\rceil, M)$ for all exceptions ξ in M, truncated to the length of σ', so that $\lceil\sigma', \xi\rceil$ exists, according to (5.2). Then the said induction principle shows (2).

Statement (2) implies (3) by (5.2). The converse is obtained by induction over the hierarchy of egis of σ, this time in the converse direction: We begin with the case where $\hat{\sigma}$ has no egis (of σ) as strict instances. For these, no recursive evaluation is performed in (5.2), and $Q(\hat{\sigma}, M)$ follows immediately. For all other $\hat{\sigma}$, assume (as the induction hypothesis) $Q(\tilde{\sigma}, M)$ for all egis $\tilde{\sigma}$ (of σ) which are strict instances of $\hat{\sigma}$; these include all $\lceil\hat{\sigma}, \xi\rceil$ in (5.2), and since $Q(\varepsilon, M_{\langle\hat{\sigma}\rangle})$ was given, we conclude $Q(\hat{\sigma}, M)$.

Statement (5) is a weakening of (4). For the converse, we use Proposition 5.2.14: Given an arbitrary instance σ', we can find a ground instance γ of σ' so that $M_{\langle\sigma'\rangle} = M_{\langle\gamma\rangle}$; hence $Q(\varepsilon, M_{\langle\sigma'\rangle}) = Q(\varepsilon, M_{\langle\gamma\rangle})$ must always be true.

Statement (3) is just a weakening of (4). For the converse, assume that $Q(\sigma, \cdot)$ is monotone. Take an arbitrary instance σ', and find the least general egi of σ instantiated by *sigma'*, as in part (3) of Corollary 5.3.6; call this label $\hat{\sigma}$. Part (4) therein shows that $M_{\langle\hat{\sigma}\rangle} \subseteq M_{\langle\sigma'\rangle}$. Since $Q(\varepsilon, \cdot)$ is monotone, $Q(\varepsilon, M_{\langle\hat{\sigma}\rangle})$ implies $Q(\varepsilon, M_{\langle\sigma'\rangle})$, which is thus shown for every instance σ' of σ. □

It is worth mentioning that the proof required only $Q(\varepsilon, \cdot)$ to be monotone, not $Q(\sigma, \cdot)$ in general. But there is really no difference: monotonicity for $Q(\sigma, \cdot)$ will follow from that of $Q(\varepsilon, \cdot)$ by Proposition 5.3.13.

Another desired property of $Q(\sigma, M)$ is that it be possible to evaluate it "partially": if we split a label into $\sigma_1\sigma_2$ and know that $Q(\sigma_2, M_{\langle\sigma'\rangle})$ holds for all exception-generated instances

σ' of σ_1, then we would want to derive $Q(\sigma_1\sigma_2, M)$. If $Q(\sigma, \cdot)$ is monotone, this is indeed possible:

Theorem 5.3.10 *If $Q(\sigma, \cdot)$ is monotone, then $Q(\sigma_1\sigma_2, M)$ is true iff $Q(\sigma_2, M_{\langle\sigma'\rangle})$ is true for all exception-generated instances σ' of σ_1 in M.*

Proof: Assume $Q(\sigma_1\sigma_2, M)$ is true. Then by Theorem 5.3.9, Statement (4), $Q(\varepsilon, M_{\langle\sigma'\sigma''\rangle})$ is true for all instances σ' of σ_1 and all instances σ'' of σ_2. Using Theorem 5.3.9 again, we obtain that $Q(\sigma_2, M_{\langle\sigma'\rangle})$ is true for all instances σ' of σ_1; *a fortiori*, it is true for all σ' which are egis of σ_1. Conversely, if $Q(\sigma_2, M_{\langle\tilde{\sigma}\rangle})$ holds for all egis $\tilde{\sigma}$ of σ_1 and σ' is an arbitrary instance of σ_1, then find the least general egi $\hat{\sigma}$ instantiated by σ'. According to Corollary 5.3.6, part (4), we have $M_{\langle\hat{\sigma}\rangle} \subseteq M_{\langle\sigma'\rangle}$, and since $Q(\sigma, \cdot)$ was assumed monotone, the known fact $Q(\sigma_2, M_{\langle\hat{\sigma}\rangle})$ implies $Q(\sigma_2, M_{\langle\sigma'\rangle})$; we thus showed $Q(\sigma_2, M_{\langle\sigma'\rangle})$ for any instance σ' of σ_1. The remainder of the proof is the converse of the first few steps. □

Neither direction of Theorem 5.3.10 can be upheld when $Q(\sigma, \cdot)$ is not monotone, as the following two examples will demonstrate:

Example 5.3.11 Let $M = \{(** - \{11\})\, p, (** - \{*2\})\, q, 12\ \bot\}$. The label 12 is not an exception-generated instance of $**$, as the only exception-generated instances other than $**$ itself are 11 and $*2$. (These two exception-generated instances do not unify.) However, the label 2 *is* an exception-generated instance of $*$ in $M_{\langle 1\rangle}$. Now let $Q(\sigma, M)$ be the property that $\bot$ does not exist in $M_{\langle\sigma\rangle}$. It is easy to see that $Q(\sigma, M)$ satisfies (5.2) but is not monotone: for if $Q(\varepsilon, M)$ is true for some M, upon adding an extra assertion $\varepsilon\ \bot$ to M we no longer have $Q(\varepsilon, M \cup \{\varepsilon\ \bot\})$. Now take $\sigma_1 = *$ and $\sigma_2 = 2$. Indeed we have $Q(*2, M)$, as $M_{\langle *2\rangle} = \{p\}$ does not contain $\bot$. But 1 is an egi of $*$ in M, and $Q(2, M_{\langle 1\rangle})$ does not hold (since $M_{\langle 1\rangle}$ contains $2\ \bot$), violating one direction of Theorem 5.3.10.

Example 5.3.12 Take $M = \{(**-\{1, *2\})\, p, 12\, \bot\}$. The label 12 is an exception-generated instance of $1*$ (as it is obtained through unifying with the exception $*2$). With the property $Q(\sigma, M)$ as in the previous example, $Q(1*, M)$ is not satisfied. But $M_{\langle 1\rangle} = \{2\ \bot\}$ does not have any exceptions, so the only exception-generated instance of $*$ is $*$ itself. Now $M_{\langle 1*\rangle}$ is empty, so it does not contain $\bot$, which makes $Q(*, M_{\langle 1\rangle})$ true, violating the other direction of Theorem 5.3.10.

It is not reasonable to restrict $Q(\sigma, M)$ so as to make Theorem 5.3.10 always true. The property $Q(\sigma, M)$: "$M_{\langle\sigma\rangle}$ does not contain $\bot$", used in the examples, seems reasonably

interesting—so much so that we will spend part of Section 5.6 studying it. Nonetheless, Theorems 5.3.9 and 5.3.10 are very powerful results. For this reason, it is desirable to prove monotonicity of a property whenever we can. Fortunately, this is easier than one might think:

Proposition 5.3.13 *If $Q(\varepsilon, \cdot)$ is monotone, then so is $Q(\sigma, \cdot)$ for any simple label σ.*

Proof: Consider two sets $M \subseteq M'$ and an arbitrary simple label σ; we wish to show that $Q(\sigma, M)$ implies $Q(\sigma, M')$. As in the previous proof, we rely heavily on Theorem 5.3.9 and Corollary 5.3.6. Thus $Q(\sigma, M)$ is true iff $Q(\varepsilon, M_{\langle\tilde{\sigma}\rangle})$ is true for all egis $\tilde{\sigma}$ of σ in M. Now consider an arbitrary egi σ' of σ *in* M', and find the least general egi in M instantiated by σ'; call it $\hat{\sigma}$. (If σ' itself is an egi in M, then $\sigma' = \hat{\sigma}$.) Now part (4) of Corollary 5.3.6 gives us $M_{\langle\hat{\sigma}\rangle} \subseteq M_{\langle\sigma'\rangle}$, but since $M \subseteq M'$, we also have $M_{\langle\sigma'\rangle} \subseteq M'_{\langle\sigma'\rangle}$ (Cor. 5.3.1) and hence $M_{\langle\hat{\sigma}\rangle} \subseteq M'_{\langle\sigma'\rangle}$; since we know that $Q(\varepsilon, M_{\langle\hat{\sigma}\rangle})$ is true, we further get $Q(\varepsilon, M'_{\langle\sigma'\rangle})$ because of monotonicity; since this holds for all egis σ' of σ in M', we have shown (again by Theorem 5.3.9) that $Q(\sigma, M')$ is true. □

Two other important results shall be established in this section. Upon analyzing at the recursion (5.2) which defines $Q(\sigma, M)$, we find that $Q(\sigma, M)$ is expensive to verify in practice, especially if σ contains many wildcards and M contains many exceptions which may unify with σ. The recursive nature of the definition further increases the potential for combinatorial explosion. Even in the most "economical" form of Theorem 5.3.9, part (3), we need to check $Q(\varepsilon, M_{\langle\hat{\sigma}\rangle})$ for every exception-generated instance $\hat{\sigma}$ of σ, and these exception-generated instances are mgus taken over subsets of exceptions in M, as we saw in Corollary 5.3.6, part (1). So can we reduce work in verifying $Q(\sigma, M)$? It usually turns out that many of the recursive computations are redundant. Suppose that a small subset M' of M already satisfies $Q(\sigma, M')$ (which entails $Q(\varepsilon, M'_{\langle\sigma\rangle})$) and that M contains an exception-generated instance σ' which does not instantiate any exception-generated instance of σ in M' (except σ itself). In the likely event that M has many exceptions beyond those in M', this will be the case very often. If $Q(\sigma, \cdot)$ is monotone (which we assume here), we see immediately that $Q(\sigma, M)$ is true. Verifying it by (5.2), however, would require us to check $Q(\varepsilon, M_{\langle\hat{\sigma}\rangle})$ for every exception-generated instance $\hat{\sigma}$ of σ in M. But for $\hat{\sigma} = \sigma'$ above, part (4) of Corollary 5.3.6 shows that $M'_{\langle\sigma\rangle} \subseteq M'_{\langle\sigma'\rangle}$. Thus $M'_{\langle\sigma\rangle}$, which has already been used as a witness for $Q(\varepsilon, M_{\langle\sigma\rangle})$, is also a subset of $M_{\langle\sigma'\rangle}$ and serves as a witness for $Q(\varepsilon, M_{\langle\sigma'\rangle})$. In short, the same fact $Q(\varepsilon, M'_{\langle\sigma\rangle})$ is used—and possibly derived at great cost—over and over again.

Toward reducing the amount of redundant work, we propose a new recursion for $Q(\sigma, M)$ which will allow us to use smaller sets M' as "witnesses" for $Q(\varepsilon, M'_{\langle\sigma\rangle})$:

Theorem 5.3.14 *If $Q(\varepsilon, M)$ is monotone, then $Q(\sigma, M)$, defined by (5.2), is equivalent to $\overline{Q}(\sigma, M)$, defined by:*

> *$\overline{Q}(\sigma, M)$ iff there exists a set $M' \subseteq M$ so that $Q(\varepsilon, M'_{\langle\sigma\rangle})$ and for all exceptions ξ in M', truncated to the length of σ, $\lceil\sigma, \xi\rceil$ does not exist, or $\lceil\sigma, \xi\rceil = \sigma$, or $\overline{Q}(\lceil\sigma, \xi\rceil, M)$.* (5.3)

Proof: It is clear that $Q(\sigma, M)$ implies $\overline{Q}(\sigma, M)$, as we can always choose $M' = M$, which reduces (5.3) to (5.2). Now let us show the converse: Given a label σ, suppose that $\overline{Q}(\sigma, M)$ is true, established using some subset M'. We would like to show that $Q(\sigma, M)$ is also true, which we will do by establishing $Q(\varepsilon, M_{\langle\sigma'\rangle})$ for all egis σ' of σ in M (Statement (3) of Theorem 5.3.9). We do this inductively, assuming that $\overline{Q}(\lceil\sigma, \xi\rceil, M)$ and $Q(\lceil\sigma, \xi\rceil, M)$ are equivalent for all exceptions ξ in M' truncated to the length of σ, whenever $\lceil\sigma, \xi\rceil$ exists and is not equal to σ. This entails that $Q(\varepsilon, M_{\langle\sigma'\rangle})$ holds for all instances σ' of $\lceil\sigma, \xi\rceil$. (The base case is included in this, which is when no such exceptions ξ exist.)
So let σ' be an arbitrary egi of σ in M (but not necessarily in M'). If σ' instantiates $\lceil\sigma, \xi\rceil$ for some exception ξ in M', then $Q(\varepsilon, M_{\langle\sigma'\rangle})$ holds by the induction hypothesis, as detailed above. Otherwise, σ itself is the only egi of σ in M' which is instantiated by σ'. We infer (Corollary 5.3.6, part (4)) that $M'_{\langle\sigma\rangle} \subseteq M'_{\langle\sigma'\rangle}$, and further $M'_{\langle\sigma'\rangle} \subseteq M_{\langle\sigma'\rangle}$ by Corollary 5.3.1 (M' is a subset of M). The fact $Q(\varepsilon, M'_{\langle\sigma\rangle})$ was given in (5.3), and by monotonicity we conclude $Q(\varepsilon, M_{\langle\sigma'\rangle})$. Since σ' was an arbitrary egi of σ in M, $Q(\sigma, M)$ follows as usual by Theorem 5.3.9. The entire proof follows by the induction principle. □
Note that the "otherwise" case in the last paragraph describes exactly the scenario we outlined as a motivation for the recursion (5.3). The redundancy we observed there has been identified as such in the proof and eliminated through use of (5.3) in lieu of (5.2).
The second and last result is about the invariance of $Q(\sigma, M)$ with regard to normalization of labels and elimination of trivial labels:

Proposition 5.3.15 *If $Q(\varepsilon, \cdot)$ is invariant wrt normalization and elimination of trivial labels, then so is $Q(\sigma, \cdot)$.*

Proof: Statement 4 in Theorem 5.3.9 says that $Q(\sigma, M)$ iff $Q(\varepsilon, M_{\langle\sigma'\rangle})$ for all instances σ' of σ. The proof now simply follows from the invariance of subscripting (by σ') wrt normalization and elimination of trivial labels. □

In practice, it is too tedious to evaluate $Q(\sigma, M)$ using Statement 4. If instead we evaluate it by the usual (5.3) or (5.2), it makes a difference when M is normalized (although the result, of course, is the same): A non-normalized set, or a set with trivial labels, may have more exception-generated instances than a normalized set. (The trivial label may specify, beside ε, any other label as exceptions.) These additional exception-generated instances may introduce redundancies in the evaluation. So we see that it is not only convenient but beneficial to work with normalized labels.

5.4 Realizability Revisited

In this section we will reconsider issues of realizability and constant-wise realizability in sets of assertions with complex labels. Starting with generalizations of theorems in Section 4.4, we apply the general properties studied in Section 5.3 to derive important novel results.

Definition 5.4.1 *Given a set M of assertions, the set $I(M)$ of* ground labels realized in M (rgls in M) *is the smallest set obeying the following recursive definition:*

1. $\varepsilon \in I(M)$.
2. *If $\gamma \in I(M)$ and M contains some assertion of the form $(\sigma c \sigma' - \Sigma)\, a$ such that $\sigma \sqsubseteq \gamma$, in which $c \in D$ is an* existential *constant, and $\xi \not\sqsubseteq_p \gamma$ for all $\xi \in \Sigma$, then $\gamma c \in I(M)$.*

For a given label λ, the set $I(\lambda, M)$ of realized ground instances (rgis) of λ in M *is the set $\mathcal{G}(\lambda, I(M))$ according to Definition 5.2.5, and λ is called* realizable in M *if $I(\lambda, M)$ is nonempty.*

We can re-state the condition in 2. as: "There exists a label λ in M such that $\lambda\, {}_p\!\sqsubseteq \gamma$, and the main label of λ contains c as an existential constant in its $(|\gamma|+1)$st position." Largely in analogy to Corollary 4.4.2 we immediately obtain these facts:

Proposition 5.4.2 *Any prefix of a realized label is realized, and any prefix of a realizable label is realizable. The empty label ε is realizable in any set. A ground label is realizable iff it is realized. If a simple label σ' is realizable in M and $\sigma \sqsubseteq_p \sigma'$, then σ is realizable in M.*

Furthermore, a non-recursive version of the definition just like Proposition 4.4.3 is useful:

Corollary 5.4.3 *A ground label $\gamma = c_1 \cdots c_k$ is realized in M iff for every $i = 1, \ldots, k$ there exists an assertion $(\sigma_{i-1} c_i \sigma'_{i-1} - \Sigma_{i-1})\, a$ in M so that $\sigma_{i-1} \sqsubseteq c_1 \ldots c_{i-1}$ and $\xi \not\sqsubseteq_p c_1 \ldots c_{i-1}$ for all $\xi \in \Sigma_{i-1}$.*

Let us illustrate this interpretation and the role exceptions play in it:

Example 5.4.4 Let $M = \{(1* - \{11\})\, q, (*1 - \{11\})\, \neg q, (*2 - \{1\})\, r, (** - \{13\})\, p, *[4]\, s\}$. We claim that $I(M) = \{\varepsilon, 1, 11\}$: First, the label 1 is witnessed by the first assertion, whose main label begins with this constant; none of its exceptions instantiates a prefix of ε. Likewise, the label 11 is witnessed by the second assertion because $*1 \sqsubseteq 11$. (The realizability of 11 is *not* negated by the fact that 11 is an exception in that assertion, which merely indicates that q need not be assigned *false* in this instance 11.) By contrast, 12 is not a realized ground instance, since the exception $\xi = 1$ in the third assertion is instantiated by the first position 1, which disqualifies the third assertion as a witness. We see how this interpretation is different from that of the second assertion. If an exception is shorter than the main label, all instances it represents are exempted from any *existence requirement* stated in the remaining positions of the main label (through existential constants). This third assertion translates to: "All labels c have a successor label $c2$, but 1 *may not*." (Again we stress the "may not": Exceptions never *prohibit* labels from existing. If M contained another assertion such as $12\ \neg r$, then 12 would be realized by virtue of that assertion.) Next, 13 is not a realized ground instance because the constant 3 never occurs in a *main label* in M. Just being stated as an exception does not render a label realized. (There is no practical use for stating an exception with constants never used in a main label in M; we just stated this example for illustration.) Finally, 14 is not a realized ground instance because [4] is only a conditional constant.

In our example we pointed out that realized ground instances do not get "disqualified" as new assertions with exceptions are added to M. We took this fact for granted in Chapter 4, but here it is worth noting:

Corollary 5.4.5 *If $M \subseteq M'$, then $I(M) \subseteq I(M')$ and $I(\lambda, M) \subseteq I(\lambda, M')$ for any label λ. Hence the properties of being a realized ground instance, and of being a realized ground label of λ, are monotone.*

Proof: All assertions in M witnessing (in the sense of Corollary 5.4.3) that a ground label γ is realized in M remain in M', so γ is realized in M'. For rgls, the proof is analogous. □
We also wish to ensure that normalization and removal of trivial labels do not affect the realized ground instances of a set. To move toward this goal, we suggest an alternative to Definition 5.4.1:

Corollary 5.4.6 *Given a set M of assertions, define $I_k(M)$ and $\Gamma_k(M)$, $k \in \mathbb{N}_0$, recursively as follows:*

- $I_0(M) = \{\varepsilon\}$;
- *for $k \in \mathbb{N}_0$, $\Gamma_k(M) = \bigcup_{i=0}^{k} I_i(M)$;*
- *for each assertion $\lambda\, a$ in M whose main label has an existential constant c in its $(k+1)$st position, include into $I_{k+1}(M)$ the set $\{\gamma c : \gamma \in I_k(M) \cap \mathcal{G}_p(\lambda, \Gamma_k(M))\}$.*

Then $I(M) = \Gamma_{d(M)}(M)$.

Proof: This definition and Definition 5.4.1 are more similar than they appear. Let $I(M)$ be as in Definition 5.4.1; then we claim that $I_k(M)$ is exactly the set of rgls in $I(M)$ of length k. We will prove this by induction over k. For $k = 0$, the statement is trivial: ε, the only label of length 0, is always realized. Now assume the statement shown for k. We introduced $\Gamma_k(M)$ in order to have a set closed under prefixes, which we use to define $\mathcal{G}_p(\lambda, \Gamma_k(M))$, the set of all rgls γ so that $\lambda \,{}_p\!\sqsubseteq \gamma$ (see Definition 5.2.5). Hence $I_k(M) \cap \mathcal{G}_p(\lambda, \Gamma_k(M))$ identifies all such γ of length k. Since we assumed the $(k+1)$st position in the main label of λ to be c, we can write $\lambda = \sigma c \sigma' - \Sigma$. This corresponds to Definition 5.4.1, whereas $|\sigma| = |\gamma| = k$. Given this, $\lambda \,{}_p\!\sqsubseteq \gamma$ is equivalent to $\sigma \sqsubseteq \gamma$, $\xi \not\sqsubseteq_p \gamma$ for any $\xi \in \Sigma$. But this is exactly the condition given in Definition 5.4.1 for including γc in $I(M)$. We have thus shown that the labels included into $I_{k+1}(M)$ are exactly the labels in $I(M)$ of length $(k+1)$. By induction on k, we conclude that this is true for all k. Finally we observe that no rgl can be of length greater than $d(M)$ because c, the $(k+1)$st and last position in any of the γc above must occur in the $(k+1)$st position of some main label in M. Note that the maximal length of any main label in M is equal to $d(M)$. Therefore, $\Gamma_{d(M)}(M)$ indeed includes all rgls in M and must be identical with $I(M)$. □

Proposition 5.4.7 *If M' is obtained from a set of assertions M by removing all assertions with trivial labels and normalizing all other labels, then $I(M) = I(M')$ and $I(\lambda', M) = I(\lambda', M')$ for any label λ'.*

Proof: Look at the construction in Corollary 5.4.6. If $\lambda\, a$ is an assertion with a trivial label, then by part (2) of Lemma 5.2.6, $\mathcal{G}_p(\lambda, \Gamma)$ is empty for any Γ, so this assertion does not give rise to including any ground instances in $I_{k+1}(M)$, for any k. Furthermore, Proposition 5.2.8 states that $\mathcal{G}_p(\lambda, \Gamma)$ remains unchanged when λ is normalized, and the main label of an assertion is unaffected by normalization. The proof of $I(M) = I(M')$ follows from this formally by induction over k: Assuming that $I_k(M) = I_k(M')$ and $\Gamma_k(M) = \Gamma_k(M')$, we conclude that corresponding labels $\lambda\, a$, $\|\lambda\|\, a$ in M, M' satisfy Corollary 5.4.6, and we get $\mathcal{G}_p(\lambda, \Gamma_k(M)) = \mathcal{G}_p(\|\lambda\|, \Gamma_k(M)) = \mathcal{G}_p(\|\lambda\|, \Gamma_k(M'))$, which shows $I_{k+1}(M) = I_{k+1}(M')$ and $\Gamma_{k+1}(M) = \Gamma_{k+1}(M')$. □

A very important test we will have to perform frequently is whether a given label σ is realizable in a given set M of assertions. In Section 4.6, we showed that this problem is NP-complete for assertions with simple labels, a result which holds unchanged if we allow complex labels. But we did not state a constructive algorithm. Trying all ground labels of σ one by one and testing if it is realized is unacceptably tedious. But we already have all the tools at hand in order to specify a more efficient way:

Proposition 5.4.8 *A label λ is realizable iff there exist some n assertions of the form $(\sigma_i \sigma_i' - \Sigma_i)\, a$ in M, $i = 1, \ldots, n$, so that $\gamma = \lceil \sigma_1, \ldots, \sigma_n \rceil$ exists and is ground, $\lambda \sqsubseteq \gamma$, and $\xi \not\sqsubseteq_p \gamma$ for any exception $\xi \in \Sigma_i$ of length less than $|\sigma_i|$, for $i = 1, \ldots, n$.*

Proof: If λ is realizable, then it has some rgi which we write as $\gamma = c_1 \cdots c_k$. Corollary 5.4.3 warrants the existence of—not necessarily all distinct—assertions $(\bar{\sigma}_{i-1} c_i \sigma_{i-1}' - \Sigma_{i-1})\, a$, $i = 1, \ldots, k$, so that $\bar{\sigma}_{i-1} \sqsubseteq c_1 \ldots c_{i-1}$ and $\xi \not\sqsubseteq_p c_1 \ldots c_{i-1}$ for any $\xi \in \Sigma_{i-1}$. In particular, $\bar{\sigma}_{i-1} c_i \sqsubseteq_p \gamma$ for all $i = 1, \ldots, k$, which shows that the upper mgu $\sigma = \lceil c_1, \bar{\sigma}_1 c_2, \ldots, \bar{\sigma}_{k-1} c_k \rceil$ exists and $\sigma \sqsubseteq_p \gamma$. Now let us show that $\sigma = \gamma$: Since $\bar{\sigma}_{k-1} c_k$ is of length k, $\bar{\sigma}_{k-1} c_k \sqsubseteq_p \gamma$ requires that σ be of length at least (and hence exactly) k. Furthermore, each of the k positions in σ corresponds to a constant in one of the unifying labels, so the unifier must be ground. This shows $\sigma = \gamma$. Furthermore, $\xi \not\sqsubseteq_p c_1 \ldots c_{i-1}$ implies $\xi \not\sqsubseteq_p \gamma$, in case the exception ξ is of length less than i. With $\sigma_i = \bar{\sigma}_i c_i$, $i = 1, \ldots, n$, this shows one direction of the proposition.

We prove the converse direction by finding suitable assertions to apply Corollary 5.4.3 in reverse. We are given that $\gamma = \lceil \sigma_1, \ldots, \sigma_n \rceil$ is ground; again write $\gamma = c_1 \cdots c_k$. Now we must have $\sigma_i \sqsubseteq_p \gamma$ for every $i = 1, \ldots, n$, and each position $j = 1, \ldots, k$ must be equal to c_j in at least one of the σ_i, whereas obviously $j \leq |\sigma_i|$. We take $\bar{\sigma}_{j-1}$ to be the prefix of σ_i of length $j-1$, and write $\sigma_i = \bar{\sigma}_{j-1} c_j \sigma'_{j-1}$. We easily verify that $\bar{\sigma}_{j-1} \sqsubseteq c_1 \ldots c_{j-1}$ and $\xi \not\sqsubseteq_p c_1 \ldots c_{j-1}$ for any exception in Σ_i (for $\xi \sqsubseteq_p c_1 \ldots c_{j-1}$ would imply $\xi \sqsubseteq_p \gamma$ while $|\xi| < |\sigma_i|$, contradicting our assumption). Since this holds for $j = 1, \ldots, k$, Corollary 5.4.3 can be applied, showing that γ is realized. Thus instantiated by an rgi, λ is shown realizable. □

The labels $(\sigma_i \sigma'_i - \Sigma_i)\ a$ are called *realizability witnesses* of λ. In practice, one would reasonably compute $\lceil \sigma_1, \ldots, \sigma_n \rceil$ iteratively; in each step i we must find a suitable realizability witness in M with prefix σ_i, so that the unifier $\lceil \sigma_1, \ldots, \sigma_i \rceil$ has instances in common with λ, while not instantiating any exception in $\Sigma_1, \ldots, \Sigma_i$. Once all positions in $\lceil \sigma_1, \ldots, \sigma_i \rceil$ are constants, we are done. Conversely, if no assertion can be found for constructing the next $\lceil \sigma_1, \ldots, \sigma_i \rceil$, we must backtrack to some $i' < i$ and try another $\sigma_{i'}$, from a different assertion in M.

The next lemma, analogous to the first 5 parts of Lemma 4.4.7, provides the foundation for a closed-form characterization of the $\models_l$ relation. It also prepares the way for defining the $\models_s$ relation:

Lemma 5.4.9 *Let M be a set of assertions and σ a simple label, Σ a set of simple labels (exceptions), c, c' constants from D, and γ, γ_1, γ_2 ground labels. Then:*

(1) $c \in I(M)$ *iff c exists in M.*

(2) $\gamma_1\gamma_2 \in I(M)$ *iff* $\gamma_1 \in I(M)$ *and* $\gamma_2 \in I(M_{\langle\gamma_1\rangle})$. *In particular,* $\gamma c \in I(M)$ *iff* $\gamma \in I(M)$ *and c exists in $M_{\langle\gamma\rangle}$.*

(3) *If c exists in M, then* $(I(M))_{\langle c\rangle} = I(M_{\langle c\rangle})$.

(4) $I(M) = \{\varepsilon\} \cup \bigcup\{cI(M_{\langle c\rangle}) : c \text{ exists in } M\}$.

(5) $I(*\sigma - \Sigma, M) = \bigcup\{c' I(\sigma - \Sigma_{\langle c'\rangle}, M_{\langle c'\rangle}) : c' \text{ exists in } M\}$.

(6) $I(c\sigma - \Sigma, M) = I([c]\sigma - \Sigma, M) = \begin{cases} cI(\sigma - \Sigma_{\langle c\rangle}, M_{\langle c\rangle}) & c \text{ exists in } M \\ \emptyset & \text{otherwise.} \end{cases}$

Proof: Parts of the proof follow the same pattern as that of Lemma 4.4.7, but there are some important differences.

As for Part (1), Corollary 5.4.3 for a one-element label $\gamma = c$ reads: $c \in I(M)$ iff there exists an assertion $(c\sigma_0' - \Sigma_0)\, a \in M$ so that ($\varepsilon \sqsubseteq \varepsilon$ and) $\xi \not\sqsubseteq \varepsilon$ for all $\xi \in \Sigma_0$. In other words, there exists an assertion in M with a *nontrivial* label, whose main label begins with c. This is exactly the condition for c existing in M, stated in Definition 5.2.11.

For (2), we write $\gamma_1 = c_1 \cdots c_{k_1}$ and $\gamma_2 = c_{k_1+1} \cdots c_k$. Corollary 5.4.3 provides characteristic conditions for $\gamma_1\gamma_2$ being realizable in M, which we split up into several parts:

For all $i = 1, \ldots, k_1$, there exists a witness $(\sigma_{i-1} c_i \sigma_{i-1}' - \Sigma_{i-1})\, a \in M$ so that

1. $\sigma_{i-1} \sqsubseteq c_1 \cdots c_{i-1}$
2. $\xi \not\sqsubseteq_p c_1 \cdots c_{i-1}$ for all $\xi \in \Sigma_{i-1}$,

and for $i = k_1 + 1, \ldots, k$, there ex. a witness $(\sigma_{i-1}^1 \sigma_{i-1}^2 c_i \sigma_{i-1}' - \Sigma_{i-1})\, a \in M$ so that

3. $\sigma_{i-1}^1 \sqsubseteq c_1 \cdots c_{k_1} = \gamma_1$
4. $\sigma_{i-1}^2 \sqsubseteq c_{k_1+1} \cdots c_{i-1}$
5. $\xi \not\sqsubseteq_p c_1 \cdots c_{k_1} = \gamma_1$ for all $\xi \in \Sigma_{i-1}$ of length at most k_1
6. $\xi_1 \not\sqsubseteq c_1 \cdots c_{k_1} = \gamma_1$ or $\xi_2 \not\sqsubseteq_p c_{k_1+1} \cdots c_{i-1}$ for all other $\xi_1\xi_2 \in \Sigma_{i-1}$.

Now we can apply Corollary 5.4.3 backwards to 1. and 2., which becomes $\gamma_1 \in I(M)$. Also, by Proposition 5.2.15, the second witness together with 3. and 5. are equivalent to

$$(\sigma_{i-1}^2 c_i \sigma_{i-1}' - (\Sigma_{i-1})_{\langle\gamma_1\rangle})\, a \in M_{\langle\gamma_1\rangle},$$

and 6. is equivalent to

7. $\xi_2 \not\sqsubseteq_p c_{k_1+1} \cdots c_{i-1}$ for all $\xi_2 \in (\Sigma_{i-1})_{\langle\gamma_1\rangle}$

We apply Corollary 5.4.3 backwards a second time: on 7. and 4., to get the equivalent statement $\gamma_2 \in I(M_{\langle\gamma_1\rangle})$. The particular case $\gamma_2 = c$ follows by applying part (1).

Part (3) can be obtained directly from parts (1) and (2):

$$\begin{aligned}
& c_1 \cdots c_k \in (I(M))_{\langle c\rangle} \\
\text{iff}\quad & cc_1 \cdots c_k \in I(M) \\
\text{iff}\quad & c \in I(M) \text{ and } c_1 \cdots c_k \in I(M_{\langle c\rangle}) \qquad (2) \\
\text{iff}\quad & c \text{ ex. in } M \text{ and } c_1 \cdots c_k \in I(M_{\langle c\rangle}). \quad (1)
\end{aligned}$$

As to (4), Proposition 5.4.2 provides the result that $I(M)$ contains ε and is closed under prefixing. Hence Lemma 4.2.7 applies, and then we use (2) and (3), just as in part (3) of Lemma 4.4.7:

$$I(M) = \{\varepsilon\} \cup \bigcup_{c \in I(M)} c(I(M))_{\langle c \rangle} = \{\varepsilon\} \cup \bigcup_{c \text{ ex. in } M} cI(M_{\langle c \rangle}).$$

For (5) and (6), knowing that $\varepsilon \notin \Sigma$, we can apply Lemma 5.2.6, parts (3) and (4). Using (1) and (3) above, the proof is entirely analogous to that of Lemma 4.4.7, parts (4) and (5), respectively. □

Finally we need to extend the definition of constant-wise realizable labels. Our first attempt is to state the equivalent of Definition 4.4.17, except that M contains complex labels, of course:

Definition 5.4.10 *Given a set M of assertions and a simple label σ', denote by $P_c(\sigma', M)$ the property that for any realized ground instance γ of σ' in M the label γc is also realized in M. A simple label σ is* constant-wise realizable *(cwr) in a set M of assertions, if for any prefix of σ of the form $\sigma' c$ with an existential constant c the property $P_c(\sigma', M)$ is satisfied.*

Both the property $P_c(\sigma', M)$ and the property of being constant-wise realizable in M are invariant wrt normalization of labels and removing trivial labels in M. This is because these properties are defined using the properties of being a realized ground label and being a realized ground instance, both of which were shown invariant in Proposition 5.4.7.
Next, we have a familiar-looking result:

Proposition 5.4.11 *If σ is a simple label, c a constant, and M a set of assertions, then:*

(1) *$c'\sigma$ is constant-wise realizable in M iff c' exists in M and σ is constant-wise realizable in $M_{\langle c' \rangle}$.*

(2) *$[c']\sigma$ is constant-wise realizable in M iff c' does not exist in M or σ is constant-wise realizable in $M_{\langle c' \rangle}$.*

(3) *$*\sigma$ is constant-wise realizable in M iff σ is constant-wise realizable in $M_{\langle c' \rangle}$ for all c' which exist in M.*

Proof: The proof of (1) and (3) can be copied verbatim from Proposition 4.4.20. (Just replace the references to Lemma 4.4.7 with the corresponding results of Lemma 5.4.9.) We only need to show part (2) about conditional constants, which we will do using the new terminology introduced in this section. So $[c']\sigma$ is cwr in M iff $P_c(\hat{\sigma}, M)$ holds for all prefixes $\hat{\sigma}c$ of $[c']\sigma$ with an existential constant c. Since $[c']$ is not existential, it does not qualify as such a prefix, so all candidate prefixes are of the form $[c']\sigma' c$. We write out $P_c([c']\sigma', M)$ explicitly:

$$\text{For all } c'\gamma \in I([c']\sigma', M),\ c'\gamma c \in I(M). \tag{5.4}$$

Let us apply part (6) of Proposition 5.4.9. If c' does not exist in M, then $I([c']\sigma', M) = \emptyset$, so (5.4) is vacuously true for all prefixes, and hence $[c']\sigma$ is cwr in M. Otherwise we wish to show that $P_c([c']\sigma', M)$ and $P_c(\sigma', M_{\langle c'\rangle})$ are equivalent for all prefixes $\sigma' c$ of σ. Observe that $I([c']\sigma', M) = c'I(\sigma', M_{\langle c'\rangle})$, so $c'\gamma \in I([c']\sigma', M)$ iff $\gamma \in I(\sigma', M_{\langle c'\rangle})$. Furthermore, the ground label $c'\gamma c$ must be found in the subset $c'(I(M))_{\langle c'\rangle}$ of $I(M)$ (as shown in Lemma 5.4.9, part (4)), and since c' exists in M, $(I(M))_{\langle c'\rangle}$ is equal to $I(M_{\langle c'\rangle})$ (by Lemma 5.4.9, part (3)). To summarize, we rewrite (5.4) equivalently as:

$$\text{For all } \gamma \in I(\sigma', M_{\langle c'\rangle}),\ \gamma c \in I(M_{\langle c'\rangle}). \tag{5.5}$$

But this is just $P_c(\sigma', M_{\langle c'\rangle})$ written out, so we showed that $P_c([c']\sigma', M)$ iff $P_c(\sigma', M_{\langle c'\rangle})$ for all prefixes $\sigma' c$ of σ as claimed, which proves (2). prefixes $\sigma' c$ of σ. This shows that σ is cwr in $M_{\langle c'\rangle}$. □

The various facts stated in Corollary 4.4.19 still hold, except that a label which exists in M may not necessarily be constant-wise realizable: Take $M' = \{(*c_2 - \{c_1\})\, p, c_1\ \top\}$. The label c_1 is realized in M', but $c_1 c_2$ is not, so $*c_2$ is not constant-wise realizable.

Apart from this minor difference, a more fundamental weakness is that constant-wise realizability is not a monotone property. For instance, let M consist of only the first formula in the above set M'. The label $*c_2$ is trivially constant-wise realizable in M, for no instance of $*$ is realized. But as we have seen, $*c_2$ is not constant-wise realizable in the larger set M'. Hence we launch a second attempt towards a more rigid definition of constant-wise realizable labels. The key to making this work is that we admit *arbitrary* ground instances γ of σ', not just realized ground instances:

Definition 5.4.12 *Given a set M of assertions and a simple label σ', denote by $Q_c(\sigma', M)$ the property that for* any *ground instance γ of σ' the label γc is realized in $M \cup \{\gamma\ \top\}$. A*

simple label σ is strongly constant-wise realizable *(scwr) in a set M of assertions, if for any prefix of σ of the form $\sigma' c$ with an existential constant c the property $Q_c(\sigma', M)$ is satisfied.*

Both the property $Q_c(\sigma', M)$ and the property of being strongly constant-wise realizable in M are invariant wrt normalization of labels and removing trivial labels in M. The argument is the same as for $P_c(\sigma', M)$ and constant-wise realizable labels: the properties are defined using the property of being realized, which is also invariant. Next, it is easy to see that $Q_c(\sigma', \cdot)$, and with it the property of being strongly constant-wise realizable, are monotone:

Proposition 5.4.13 *Given two sets M and M' of assertions, $M \subseteq M'$, if $Q_c(\sigma', M)$ is true, then so is $Q_c(\sigma', M')$. Furthermore, if σ is strongly constant-wise realizable in M, then it is strongly constant-wise realizable in M' also.*

Proof: If γc is realized in $M \cup \{\gamma\, \top\}$, then it is also realized in the larger set $M' \cup \{\gamma\, \top\}$ (Corollary 5.4.5). The proof follows immediately from this. □

We state some expected, easy-to-verify properties similar to Corollary 4.4.19:

Proposition 5.4.14 *If σ is strongly constant-wise realizable in M, then so is any prefix of σ. Ground labels are strongly constant-wise realizable iff they are realized, whereas any label containing only $*$ and conditional labels is strongly constant-wise realizable. A constant c is strongly constant-wise realizable in M iff it exists in M as an existential constant.*

Proposition 5.4.15 *If σ_1 and σ_2 are both strongly constant-wise realizable in M, then so are $\lfloor \sigma_1, \sigma_2 \rfloor$ and $\lceil \sigma_1, \sigma_2 \rceil$, provided these unifiers exist.*

Proof: Every prefix of $\lceil \sigma_1, \sigma_2 \rceil$ of the form $\sigma' c$ instantiates a prefix $\sigma'' c$ in one of σ_1 and σ_2, Since γc is realized in $M \cup \{\gamma\, \top\}$ for all ground instances γ of σ'' and these instances include all ground instances of σ', we have shown that $\lceil \sigma_1, \sigma_2 \rceil$ is scwr in M. The same holds for $\lfloor \sigma_1, \sigma_2 \rfloor$ because it is a prefix of $\lceil \sigma_1, \sigma_2 \rceil$. □

Let us now show that $Q_c(\sigma', M)$ is an instance of the property studied intensively in the previous section:

Proposition 5.4.16 *For any constant $c \in D$, the property $Q_c(\sigma', M)$ satisfies (5.2).*

Proof: Similarly as in Theorem 5.3.14, we let $\overline{Q}_c(\sigma', M)$ be the property defined from $Q_c(\varepsilon, M)$ using (5.2), and show that $\overline{Q}_c(\sigma', M)$ and $Q_c(\sigma', M)$ are equivalent. First, we

observe that $Q_c(\varepsilon, M)$ is the property that c is realized in M, which is monotone. By Proposition 5.3.13, so is $\overline{Q}_c(\sigma', M)$. Then Statement (5) of Theorem 5.3.9 shows that $\overline{Q}_c(\sigma', M)$ is true iff $Q_c(\varepsilon, M_{\langle\gamma\rangle})$ is true for all ground instances γ of σ'. If we can show the same for $Q_c(\sigma', M)$, then the proposition follows.

According to Definition 5.4.12, $Q_c(\sigma', M)$ holds iff $\gamma c \in I(M \cup \{\gamma\ \top\})$ for every ground instance γ of σ'. By Lemma 5.4.9, part (2), this is equivalent to γ being realized in $M \cup \{\gamma\ \top\}$ (which is trivially true, as $\gamma\ \top$ witnesses it), and c existing in $(M \cup \{\gamma\ \top\})_{\langle\gamma\rangle} = M_{\langle\gamma\rangle} \cup \{\varepsilon\ \top\}$. By part (1) of said Lemma 5.4.9, c exists in a set iff it is realizable in it, so upon applying Definition 5.4.12 again, we equivalently get $Q_c(\varepsilon, M_{\langle\gamma\rangle})$. Exactly this was to be shown, for all instances γ of σ'. □

This important result makes available to us all the results of the previous section and opens up a whole host of properties, among which we will only mention a few. The first one qualifies witnesses of strongly constant-wise realizable labels. The assertion $11\ a$ witnesses that $*1$ is constant-wise realizable in $M = \{11\ a\}$, for 1 is the only constant occurring in the first position. As long as the realizability of $c1$ is shown for every constant c existing in M, $P_1(*, M)$ is shown, and $*1$ is proven constant-wise realizable. By contrast, $11\ a$ does not serve as a witness for $*1$ being strongly constant-wise realizable in M. (In fact, $*1$ is not strongly constant-wise realizable in M.) Instead, a label witnessing $Q_1(*, M)$ must have a prefix which is instantiated by $*$. More generally, any $*$ in σ' must be matched by a $*$ in the witness:

Proposition 5.4.17 *If the label σ is strongly constant-wise realizable in M, then for any prefix of σ of the form $\sigma' c$, c exists in $M_{\langle\sigma'\rangle}$, and there exists an assertion $(\sigma_0 c \sigma_1 - \Sigma)\ a$ in M so that $\sigma_0 c \sigma_1 - \Sigma\ {}_p\sqsubseteq \sigma'$.*

Proof: If σ is scwr, then any prefix $\sigma' c$ of σ satisfies $Q_c(\sigma', M)$. Now the first condition in (5.2) states that $Q_c(\varepsilon, M_{\langle\sigma'\rangle})$ is true, so c must exist in (some assertion A in) $M_{\langle\sigma'\rangle} \cup \{\varepsilon\ \top\}$. The assertions in $M_{\langle\sigma'\rangle}$ can be characterized as usual using Proposition 5.2.15, and the assertion in M which contributes A to $M_{\langle\sigma'\rangle}$ (according to Remark 5.2.16) must be of the form $(\sigma_0 c \sigma_1 - \Sigma)\ a$, with $(\sigma_0 c \sigma_1 - \Sigma)\ {}_p\sqsubseteq \sigma'$, which is what we claimed. □

Also, we can finally do justice to our term *strongly constant-wise realizable*:

Proposition 5.4.18 *If σ is strongly constant-wise realizable in M, then σ is constant-wise realizable in M.*

Proof: We only need to show that $Q_c(\sigma', M)$ implies $P_c(\sigma', M)$; the theorem follows directly from this. So assume that $Q_c(\sigma', M)$ holds, and let γ be an arbitrary rgi of σ' in M. By Statement (5) of Theorem 5.3.9, γ as a ground instance of σ' satisfies $Q_c(\varepsilon, M_{\langle\gamma\rangle})$, that is, c exists in $M_{\langle\gamma\rangle}$. Since γ as an rgi of σ' is realized in M, part (2) of Lemma 5.4.9 implies that γc is realized, which shows Property $P_c(\sigma', M)$. □

5.5 Weak and Strong Satisfiability

Having developed the necessary prerequisites, we are now ready to re-state the $\models_l$ relation to fit the extensions in our logic $\mathcal{LBL}_x$. . We remind the reader that all labels used in (sets of) $\mathcal{LBL}$-formulas must be simple (with existential or conditional constants but no exceptions), whereas assertions may be preceded by complex labels of any kind.

Definition 5.5.1 *The logic $\mathcal{LBL}_x$ is defined on the language of labelled formulas, using sets of assertions (with complex labels) as semantic structures, and defining the semantic relation $M \models_l S$ on sets of assertions M and sets of labelled formulas S as an extension of $\models_g$ (see Definition 2.1.1) by:*

$$
\begin{array}{lll}
M \models_l l & \textit{iff} & (\varepsilon - \Sigma)\, l \in M \textit{ and } \varepsilon \notin \Sigma, \textit{ for any literal } l \\
M \models_l [c]\, F & \textit{iff} & c \textit{ does not occur in } M \textit{ or } M_{\langle c\rangle} \models_l F. \\
M \models_l c\, F & \textit{iff} & c \textit{ occurs in } M \textit{ and } M_{\langle c\rangle} \models_l F. \\
M \models_l *\, F & \textit{iff} & \textit{for all constants } c \textit{ which occur in } M,\ M_{\langle c\rangle} \models_l F.
\end{array}
$$

A clash *in a set of assertions M is an occurrence of an assertion $\lambda\, \bot$ where λ is realizable, or of a pair of assertions $\lambda'\, p, \lambda''\, \neg p$, where (λ', λ'') exists and is realizable in M. The label λ, or the unifier (λ', λ''), is called the* clash witness.
If $M \models_l S$, we say that M (weakly) verifies *S. If additionally M is clash-free, we say that M* (weakly) satisfies *S, or is a* (weak) model *of S, written $M \models_l S$. If S has a model, it is* (weakly) satisfiable[3].

Among many possible generalizations of results from Chapter 4, a closed-form characterization analogous to Theorem 4.4.21 will be of prominent importance for the work to follow:

[3] *See our remarks on terminology at the beginning of Chapter 2.*

Theorem 5.5.2 *Given a set M of assertions and a formula σF, $M \models_l \sigma F$ iff σ is constant-wise realizable in M and $M_{\langle\gamma\rangle} \models_l F$ for every realized ground instance γ of σ in M.*

Proof: The proof is almost identical to that of Theorem 4.4.21; for the needed preliminaries we call upon Proposition 5.4.11 and Lemma 5.4.9. In addition to $*\sigma'$ and $c\sigma'$, we need to consider the extra case $\sigma = [c]\sigma'$ in the induction step: If c does not exist in M, then $M \models_l [c]\, F$ holds trivially; but according to Proposition 5.4.11, $[c]\sigma'$ is also cwr, and "$M_{\langle\gamma\rangle} \models_l F$ for every rgi γ" is vacuously true, as $[c]\sigma'$ has no rgi. If c exists in M, this case is identical to the case $\sigma = c\sigma'$. □

Corollary 5.5.3 *The properties $M \models_l S$ and $M \models_l S$ are invariant wrt normalization of labels and removal of trivial labels.*

Proof: A full proof uses structural induction on F. We will only sketch the important steps: For the base case, note that $M \models_l l$ requires $(\varepsilon - \Sigma)\, l$ to be in M; this assertion was explicitly required to be non-trivial, and upon normalization it will simply become $\varepsilon\, l \in M'$. This shows that $M' \models_l l$. Assuming the invariance shown for F, we use Theorem 5.5.2 and the induction hypothesis (on $M_{\langle\gamma\rangle} \models_l F$) to show it for $\sigma\, F$; for, σ being cwr is invariant, and γ being an rgi of σ is also an invariant, wrt normalization and removing trivial labels. □

Furthermore, we will make use of this analogue of Proposition 4.4.10:

Proposition 5.5.4 *If M is clash-free and γ is a realized ground label in M, then $M_{\langle\gamma\rangle}$ is clash-free.*

We obtain a proof either in analogy to Proposition 4.4.10, or from the following generalization:

Proposition 5.5.5 *A set M of assertions contains a clash iff it has a realized ground label γ so that $M_{\langle\gamma\rangle}$ contains $\varepsilon\, \bot$ or two complementary assertions $\varepsilon\, p$, $\varepsilon\, \neg p$.*

Proof: Assume that an rgl γ exists so that $\varepsilon\, p$ and $\varepsilon\, \neg p$ are assertions in $M_{\langle\gamma\rangle}$. They must have been contributed by two assertions $\lambda\, p, \lambda'\, \neg p \in M$ so that $\lambda \sqsubseteq \gamma$ and $\lambda' \sqsubseteq \gamma$, as we easily see from Remark 5.2.16. This shows that λ and λ' have a common (realized ground) instances, so their mgu exists and has γ as an rgi, which witnesses a clash in M. The argument can be reversed, in which case the existence of $\lambda\, p$ and $\lambda'\, p$, and of a common rgi

γ of λ and λ', is warranted by the clash. Therefore, $\lambda\, p$ and $\lambda'\, p$ both contribute assertions to $M_{\langle\gamma\rangle}$, namely $\varepsilon\, p$ and $\varepsilon\, \neg p$. The case where the clash is of the form $\lambda\bot$ is easier and left to the reader. □

We already observed in the previous chapter that the $\models_l$ relation is not monotone: Provided $*$ is used in the labels of a formula $\sigma\, F$, we can extend an existing set verifying $\sigma\, F$ by introducing new realizable labels and assertions which falsify $\sigma\, F$. In essence, this is the same problem we encountered on constant-wise realizable labels in the previous section. A partial answer, as we might expect, is to replace "constant-wise realizable" by "strongly constant-wise realizable" in the closed-form characterization; however, this alone is not sufficient.

In addition to this, recall the shortcomings of the $\models_l$ relation we outlined in the introduction to this chapter, namely that the naïve evaluation of $M \models_l \sigma\, F$ forces us to evaluate $M_{\langle\gamma\rangle} \models_l F$ for every realized ground instance γ of σ. We suggested that we may overcome these shortcomings by a semantic relation which is evaluated "monolithically": Establish that $M_{\langle\sigma\rangle}$ verifies F, and recursively check whether M also verifies $\xi\, F$ for all exceptions ξ encountered. The time has now come to make this more precise, giving rise to our new logic $\mathcal{LBL}_s$:

Definition 5.5.6 *The logic $\mathcal{LBL}_s$ is defined on the language of labelled formulas, using sets of assertions (with complex labels) as semantic structures, and defining the* strong semantic relation $\models_s$ *on sets of assertions and formulas as an extension of $\models_g$ in the following way:*

- *$M \models_s l$ iff $(\varepsilon - \Sigma)\, l \in M$ and $\varepsilon \notin \Sigma$, for any literal l.*
- *For a label σ without existential constants, $M \models_s \sigma\, F$ iff $M_{\langle\sigma\rangle} \models_s F$, and for all exceptions ξ occurring in the labels of M, truncated to length at most $|\sigma|$, either $\lceil\sigma, \xi\rceil$ does not exist, or $\lceil\sigma, \xi\rceil = \sigma$, or $M \models_s \lceil\sigma, \xi\rceil\, F$.*
- *For a label σ with existential constants, $M \models_s \sigma\, F$ iff σ is strongly constant-wise realizable in M and $M \models_s [\sigma]\, F$.*

(Strongly) satisfying sets (models), the $\models_s$ relation, and (strongly) satisfiable formulas are defined just like their weak counterparts in Definition 5.5.1.

Thanks to the second condition in this definition, the property $M \models_s \sigma\, F$ (for a label σ without existential constants) satisfies (5.2) in Section 5.3. Therefore, we will also write it as $Q_F(\sigma, M)$. Naturally, our concern is to establish this property as monotone:

Proposition 5.5.7 *If $M \models_s \sigma\, F$, then $M' \models_s \sigma\, F$ for any set of assertions $M' \supseteq M$.*

Proof: This proposition is an excellent candidate for structural induction according to Lemma 4.3.6. We have already shown it for propositional atoms (Proposition 2.2.12), and we know it to be preserved under propositional concatenation (Theorem 2.1.8). Next, if σ is a nonempty label without existential constants and $Q_F(\varepsilon, \cdot)$ is monotone, then by Proposition 5.3.13, $Q_F(\sigma, \cdot)$ is monotone for any σ without existential constants. Finally, if σ does have existential constants, then $M \models_s \sigma\, F$ requires that σ be scwr in M, but since this property itself is monotone, σ is also scwr in M', and we obtain $M' \models_s \sigma\, F$ as before. The proof now follows by structural induction on F. □

Once again, we can draw upon a host of results from Section 5.3. Our first corollary is based on part (5) of Theorem 5.3.9 plus Definition 5.5.6, and it yields a "strong" equivalent of Theorem 5.5.2:

Corollary 5.5.8 *Given a set M of assertions and a formula $\sigma\, F$, $M \models_s \sigma\, F$ iff σ is strongly constant-wise realizable in M and $M_{\langle\gamma\rangle} \models_s F$ for every ground instance γ of σ.*

From this we get our usual invariance wrt normalization and removing trivial labels:

Corollary 5.5.9 *The properties $M \models_s S$ and $M \models_s S$ are invariant wrt normalization of labels and removal of trivial labels.*

Proof: This is analogous to the proof of Corollary 5.5.3, except that here we use the property of being scwr, which is invariant wrt normalization and removing trivial labels, to prove the induction step (invariance under labelling). □

We now understand in what way $\models_s$ is more rigid than $\models_l$: Even if $M_{\langle\gamma\rangle}$ verifies p, say, for any *realized* ground instance γ of σ, this does not show $M_{\langle\gamma\rangle} \models_s p$ for an *arbitrary* ground instance γ. Hence $M \models_l S$ does not always entail $M \models_s S$. On the other hand, $\models_s$ entails $\models_l$; this is similar to the result that strongly constant-wise realizable labels are constant-wise realizable, shown in the previous section:

Theorem 5.5.10 *If $M \models_s F$, then $M \models_l F$. Hence strong models for F are weak models.*

Proof: The property expressed in the first part matches the first property of Proposition 2.1.18. (The two logics here use the same language; they are distinguished only by the semantic relations $\models_s$ and $\models_l$, respectively. The homomorphism is the identical mapping

$F \mapsto F$.) Hence our theorem is preserved under propositional concatenation. Since Definitions 5.5.1 and 5.5.6 agree on propositional atoms, the theorem holds for these. We will now show it preserved under labelling.

Assume the theorem shown for some formula F (over all sets of assertions). Consider a fixed but arbitrary set M and a simple label σ so that $M \models_s \sigma\, F$. By Theorem 5.3.9, Statement (5), we infer that $M_{\langle\gamma\rangle} \models_s F$, and by the induction hypothesis, $M_{\langle\gamma\rangle} \models_l F$ for all instances γ of σ. This is true particularly for all γ which are rgis of σ. Furthermore, σ is scwr and hence cwr, by Corollary 5.4.18. We apply Theorem 5.5.2 to conclude that $M \models_l \sigma\, F$.

The first part of our theorem now follows by structural induction according to Lemma 4.3.6. The second part follows from the fact that both strong and weak models are defined using the same version of clash-freeness. □

This theorem will serve as part of our proof, to be completed in Section 5.7, that $\mathcal{LBL}_w$-satisfiability as defined in Chapter 4, weak satisfiability as in Definition 5.5.1, and strong satisfiability as in Definition 5.5.6 are equivalent. Before we do this, however, we need to spend the next section addressing a shortcoming in our definition.

We end this section with two more useful results. First, we recall the alternate characterization of Theorem 5.3.14 which, as we have seen, helps eliminate much redundancy. Let us restate it the present context of checking $M \models_s \sigma\, F$:

Corollary 5.5.11 *For any formula of the form $\sigma\, F$ where σ does not contain existential constants, $M \models_s \sigma\, F$ iff there exists a set $M' \subseteq M$ so that $M'_{\langle\sigma\rangle} \models_s F$, and for all exceptions ξ in M', truncated to the length of σ, $\lceil\sigma,\xi\rceil$ does not exist, or $\lceil\sigma,\xi\rceil = \sigma$, or $M \models_s \lceil\sigma,\xi\rceil\, F$.*

Secondly, we present an easy way to construct a set of assertions M for a labelled formula $\sigma\, F$. To ensure that all instances of σ are covered, we can simply use the same label σ in the assertions in M:

Proposition 5.5.12 *If M is nonempty and $M \models_s F$, then $\sigma M \models_s \sigma\, F$, and $\sigma M \models_s [\sigma']\, F$ for every instance σ' of σ. More specifically, $\{\sigma\ a\} \models_s \sigma\ a$, and $\{\sigma\ a\} \models_s [\sigma']\, a$ for every instance σ' of σ.*

Proof: Take an arbitrary instance σ' of σ. Then $(\sigma M)_{\langle\sigma'\rangle} = M \models_s F$. Furthermore, all exceptions ξ which may exist in M become exceptions $\sigma\xi$ in σM, and truncated to the length of σ, these exceptions are all equal to σ, so no recursive evaluations in Definition 5.5.6 ensue, which shows that $\sigma M \models_s [\sigma']\, F$.

In case $\sigma' = \sigma$, we will show that σ is scwr in σM. (This looks obvious, but it is not quite that trivial.) Together with the above, this proves that $\sigma M \models\!\!\!\sim_s \sigma F$. So take any set $M' = \{(\sigma\sigma_0 - \sigma\Sigma)\, a\}$ consisting of just one assertion from σM, and let *šigmac* be any prefix of σ ending in an existential constant. We easily see that c exists in $M'_{\langle\tilde{\sigma}\rangle}$ (which is $Q_c(\varepsilon, M'_{\langle\tilde{\sigma}\rangle})$), and that all exceptions in M', truncated to the length of $\tilde{\sigma}$, are equal to $\tilde{\sigma}$. Hence no recursive evaluations ensue, and $Q_c(\tilde{\sigma}, M')$ is established according to Proposition 5.4.16. Since $\tilde{\sigma}c$ is an arbitrary prefix of σ, we have shown that σ is scwr in M', and hence in σM by monotonicity.

The second half of the proposition represents the special case $F = a$ and $M = \{\varepsilon\, a\}$. □

This can be applied in combination with the previous Corollary 5.5.11. If we have several formulas of the form σF or σa in a larger set S, we can make σM or $\{\sigma a\}$ the subset M' of a larger set of assignments verifying S; then M' verifies σF or σa according to Corollary 5.5.11. In this fashion, a set of assertions for S can be pieced together by sets verifying its individual formulas. Note that this is not guaranteed to produce an overall *model* for S, as the union of all the M' may not be clash-free. But it may provide a good starting point, a set which may be turned into a model for S through clash repairs.

5.6 Talking about Utopia

In the previous section, we succeeded in defining a monotone semantic relation. Recall that monotonicity establishes a relationship between the lattice of all sets M of assertions and the lattice of sets of formulas verified by M (see Section 2.2). However, as Definition 5.5.6 stands, some formulas are not verified by any set M. Consider the formula $F = *\,\bot$. In analogy to the formula $\Box\,\bot$ which is satisfied by any Kripke model in which the root node w_0 has no successor, this formula ought to be satisfied by any model M in which $*$ is not realizable (i.e. where no constant exists in M). Indeed, any such model satisfies Definition 5.5.1, showing that $M \models_l F$. However, we easily see that no set M exists so that $M \models_s F$, or even $M \models\!\!\!\sim_s F$: Definition 5.5.6 requires that $M_{\langle * \rangle} \models\!\!\!\sim_s \bot$ which is impossible, as $\bot$ is not verified by any set. This discrepancy is unacceptable, as we want the logics $\mathcal{LBL}_x$ and $\mathcal{LBL}_s$ to define the same set of satisfiable formulas. Moreover, for reasons which will become evident later, we desire for *every* set S to have a set of assertions M so that $M \models\!\!\!\sim_s S$. (If S is unsatisfiable, any such set M must have a clash, of course.)

We remedy the situation by a slight modification on the role of the propositional constant

$\perp$ in assertions. We declare that a set M containing $\perp$ verifies *every* $\mathcal{LBL}$-formula; we aptly call such a set *utopian*[4]. Hence a utopian set M can be used as a trivial witness for $M \models_s S$ whenever S has no "proper" set of assertions verifying it. Note that $\perp$ constitutes a clash with witness ε in M, so a utopian set can never be a model[5].

Since any superset of a utopian set also contains $\perp$, the monotonicity of $\models_s$ is not affected by this change in definition. However, we will need to adjust some other definitions in order to accommodate the properties of utopian sets. First, in the same way as every formula is verified by a utopian set M, we consider all ground labels realized, and all labels realizable, in M. Upon subscripting a utopian set M, we would like $M_{\langle\sigma\rangle}$ to "inherit" the property of being utopian, but to be the same as before in all other aspects. Therefore, we extend Definition 5.2.11 to utopian sets as follows:

Definition 5.6.1 *For any set M of assertions and any constant or wildcard x, let $M^{\text{old}}_{\langle x\rangle}$ be the subscripted set according to Definition 5.2.11. If M is utopian, we set $M_{\langle x\rangle} = \{\perp\} \cup M^{\text{old}}_{\langle x\rangle}$, otherwise $M_{\langle x\rangle} = M^{\text{old}}_{\langle x\rangle}$. $M_{\langle\sigma\rangle}$ is defined iteratively from $M_{\langle x\rangle}$ in the usual way.*

Utopian sets can arise as a result of subscripting non-utopian sets:

Proposition 5.6.2 *If $M^{\text{old}}_{\langle\sigma\rangle}$ is the subscripted set of Definition 5.2.11, then a closed form for $M_{\langle\sigma\rangle}$ in Definition 5.6.1 is:*

$$M_{\langle\sigma\rangle} = \begin{cases} \{\varepsilon \perp\} \cup M^{\text{old}}_{\langle\sigma\rangle} & \text{if } \lambda \perp \in M \text{ and } \lambda \sqsubseteq_p \sigma \\ M^{\text{old}}_{\langle\sigma\rangle} & \text{otherwise.} \end{cases}$$

Proof: Consider the shortest σ' for which some $\lambda \perp$ exists in M such that $\lambda \sqsubseteq \sigma'$. Then $M^{\text{old}}_{\langle\sigma'\rangle}$ contains $\varepsilon \perp$, and according to our new definition, the set $M_{\langle\sigma'\sigma''\rangle}$ is utopian for any label $\sigma'\sigma''$ extending σ'. If no such label σ' exists, then $M_{\langle\sigma\rangle}$ and $M^{\text{old}}_{\langle\sigma'\rangle}$ are clearly identical. □

---

[4] We could also re-declare the role of the empty model in such a way that $\emptyset$ does not verify *any* formula. For instance, we could require for $M \models_s \top$ that $\varepsilon \top \in M$. Then the above relationship would have the following nice property: for every set of formulas S we can find minimal sets of assertions which verify all formulas in S, as well as maximal sets of assertions which verify no formulas in S. However, the aesthetic value alone does not seem to justify the efforts in changing our definition yet again.

[5] We could save ourselves this non-standard definition and just declare that every *inconsistent* set trivially verifies every formula, following an *ex falso quodlibet* philosophy. However, testing for inconsistency (clashes) is computationally expensive. Admitting only a minimal number of utopian sets necessary proves to be much more elegant for our purposes. Most importantly, we can immediately decide whether a set is utopian, without the need for realizability tests and the like.

The label σ' in the proof marks the transition from a proper set into a utopian set. The same transition needs to be taken into account as we adjust the definition of realized ground labels in M:

Definition 5.6.3 *Given a set M of assertions, $I(M)$ is defined as the smallest set satisfying the recursive definitions in Definition 5.4.1, plus:*

3. *If $\gamma \in I(M)$ and $\bot \in M_{\langle\gamma\rangle}$, then $\gamma c \in I(M)$ for all $c \in D$.*

The elements of $I(M)$ are called realized ground labels (rgls)*; as before, a label is* realizable *if at least one realized ground label instantiates it. A realized ground label is* non-trivial *if it is a realized ground label according to Definition 5.4.1 proper (that is, without using 3. to justify any of its positions), and* trivial *otherwise.*

Corollary 5.6.4 *If $\gamma \in I(M)$ and $\bot \in M_{\langle\gamma\rangle}$, then every ground label γ' which has γ as a prefix is realized. More generally, if M contains $\lambda\, \bot$, then all ground labels γ' which have a prefix $\gamma \in I(\lambda, M)$ are realized.*

Proof: If $\bot \in M_{\langle\gamma\rangle}$, then for every ground label γ' extending γ, $M_{\langle\gamma'\rangle}$ is also utopian, so if γ' is realized, then so is $\gamma' c$ for any $c \in D$, as stated in Definition 5.6.3. The label γ itself was presumed realized, so by way of induction we conclude that any label extending γ is also realized. For the second part, observe that $\lambda\, \bot \in M$ implies $\bot \in M_{\langle\gamma\rangle}$ for all ground instances γ of λ; given this, the statement follows from the first part of the proof. □

We observe that the entry point of this "lapse" into triviality is the shortest ground label γ_0 so that $M_{\langle\gamma_0\rangle}$ contains $\bot$, and we can say:

Corollary 5.6.5 *Every trivial realized ground label γ of M has a smallest prefix γ_0 so that $\bot \in M_{\langle\gamma_0\rangle}$; this prefix γ_0 is a nontrivial realized ground label.*

Hence a necessary precondition for a trivial realized ground label to occur is that some assertion $\lambda\, \bot$ exist in M so that γ_0 is a (nontrivial) realized ground instance of λ. This means that M cannot be clash-free. By the contrapositive, whenever M is clash-free, it has no trivial realized ground labels; therefore, our transition from Definition 5.4.1 to Definition 5.6.3 leaves $I(M)$ unchanged. However, even for clash-free sets M, Definition 5.6.3 impacts our notion of *strongly constant-wise realizable labels.* While the definitions of constant-wise realizable and strongly constant-wise realizable labels (Definitions 5.4.10 and 5.4.12)

themselves do not change, their applicability changes with our extended notion of realized ground labels:

Example 5.6.6 In the model $M = \{* \; \bot\}$, every label of the form $*\sigma$ is strongly constant-wise realizable. For instance, $*c'$ is shown strongly constant-wise realizable as follows: Given any constant c, $(M \cup \{c \; \top\})_{\langle c \rangle}$ is utopian, and since c is realizable in $M \cup \{c \; \top\}$, so is cc' by Definition 5.6.3. This proves that $*c'$ is strongly constant-wise realizable.

The result in Example 5.6.6 looks bizarre at first glance. But as a matter of fact, the labels $*\sigma$ have been "weakly" constant-wise realizable all along, since $*$ has no realized ground instances. Thus the new definition of realized labels brings the property of being strongly constant-wise realizable closer to that of being constant-wise realizable (yet there still exist constant-wise realizable labels which are not strongly constant-wise realizable). We can state the following general facts:

Proposition 5.6.7 *Every label which is constant-wise realizable (strongly constant-wise realizable) in the old sense (allowing only nontrivial realized ground labels in the definition) is also constant-wise realizable (strongly constant-wise realizable) in the new sense (allowing any realized ground label). Any strongly constant-wise realizable label is constant-wise realizable.*

Proof: We prove this by analyzing the properties $P_c(\sigma', M)$ and $Q_c(\sigma', M)$ in Definitions 5.4.10 and 5.4.12, respectively: $Q_c(\sigma', M)$ holds iff γc is realized in $M \cup \{\gamma \; \top\}$ for every instance γ of σ'. But whenever γc is nontrivially realized, it certainly is realized in the new, more general sense. For $P_c(\sigma', M)$, we need to distinguish between the two types of rgis γ: If γ is *nontrivially* realized, then γc is realized, as we assumed that $P_c(\sigma', M)$ holds in the old sense. Now if γ is *trivially* realized, then by Corollary 5.6.4 any extension γc is also trivially realized. Having thus shown that $\gamma c \in I(M)$ for all types of realizable labels γ, we have verified $P_c(\sigma', M)$ (in its new sense). For the third statement which originally was Corollary 5.4.18, we add a twist to its original proof: Using the same γ, finding that $Q_c(\varepsilon, M_{\langle\gamma\rangle})$ is true allows us to conclude that either c exists in $M_{\langle\gamma\rangle}$ (as in the original proof) or that $M_{\langle\gamma\rangle}$ is utopian. But in the latter case, Corollary 5.6.4 shows that γc is trivially realized. We derive property $P_c(\sigma', M)$ and conclude as in the original proof of Corollary 5.4.18. □

As the last task of this section, we now modify the relation $\models_s$ in Definition 5.5.6. That is, we formally define what we discussed at the beginning:

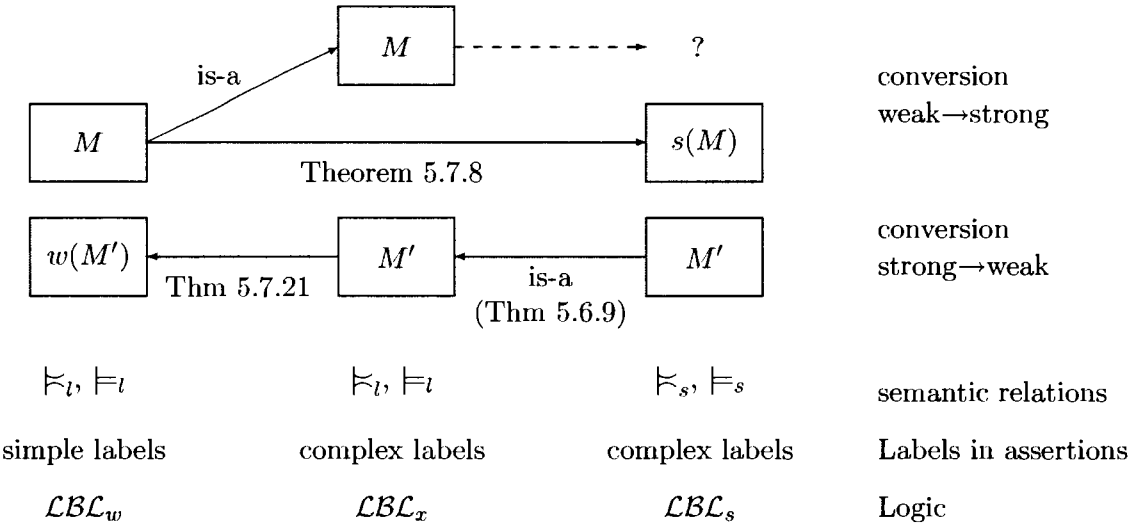

Figure 5.4: The logics $\mathcal{CBL}_w$, $\mathcal{CBL}_x$, and $\mathcal{CBL}_s$, and how their models can be converted.

Definition 5.6.8 *If M is utopian, then $M \models_s F$ for all $\mathcal{CBL}$-formulas F. For all other sets of assertions, $M \models_s F$ if M meets the conditions stated in Definition 5.5.6.*

Theorem 5.6.9 *If M is a* clash-free *set of assertions and $M \models_s F$, then $M \models_l F$. In other words, all strong models for F are weak models for F.*

Proof: A clash-free set cannot contain $\bot$, so $M \models_s F$ is determined according to Definition 5.5.6. We prove the theorem using the same structural induction proof as in Theorem 5.5.10. In proving the theorem preserved under labelling, we encounter a difficulty though: σ may have instances γ so that $M_{\langle\gamma\rangle}$ is utopian, preventing us from applying the induction hypothesis. But, recalling that we are only interested in *realized* instances γ, and aided by Proposition 5.5.4, we find that $M_{\langle\gamma\rangle}$ remains clash-free for any such rgi γ. So the induction hypothesis does apply, and the rest of the proof follows as in Theorem 5.5.10. □

5.7 From Weak to Strong Models and Back

Let us recall our progress so far. In Chapter 4 we provided a logic $\mathcal{CBL}_w$ whose models are clash-free sets of assertions with simple labels and whose semantic relation is $\models_l$. In Section 5.2 we extended the definition of models, allowing for complex labels in assertions.

In Section 5.5 we adjusted the $\models_l$ relation to accommodate these complex labels, which resulted in the logic $\mathcal{LBL}_x$. (We also extended the language of $\mathcal{LBL}_x$ slightly, to allow conditional constants which we will need in the next chapter.) However, we found that $\mathcal{LBL}_x$ is laden with problems preventing efficient model checking. So we proposed a new relation $\models_s$ which is better suited for handling labels with exceptions in model checking, resulting in $\mathcal{LBL}_s$. We will now study how the three logics relate to one another, claiming that exactly the same formulas are satisfiable in each of them[6]. This is an important result, since it ties into the results of Chapter 4: ultimately, satisfiability of a $\mathbf{K_{NNF}}$-formula can be decided by giving a decision procedure for $\mathcal{LBL}_s$. Just as in Chapter 4, our proofs are constructive: we will show how a model for a formula F in one logic can be converted to a model in another. We summarize all results in Figure 5.4. Since the $\models_l$ relation remains unchanged on models whose labels are simple and do not contain conditional constants, all $\mathcal{LBL}_w$-models (left-hand side of Figure 5.4) are $\mathcal{LBL}_x$-models (middle). Furthermore, we showed in Theorem 5.6.9 that $\mathcal{LBL}_s$-models (right-hand side) are also $\mathcal{LBL}_x$-models. We are yet to specify the other two conversions marked by solid arrows in Figure 5.4; they are non-trivial in that assertions must be added to the original model or replaced with equivalent assertions. We term these conversions *saturation* and *expansion*, respectively. To illustrate how they work, we will motivate each of them on an example before we describe the details. The assertions in a weak model M define the set of ground instances which are realized in it, and they provide variable assignments on subsets of these, sufficient for showing a formula satisfied. In the case of a labelled formula $\sigma\, F$ where σ contains $*$, weak satisfiability entails that F must be verified in all realized ground instances γ of σ. By contrast, a strong model must contain assertions whose main label is σ or some other label instantiated by σ; these assertions express universal statements over a scope at least as large as σ, needed to verify F. Such universal statements may be absent from weak models. In the process of saturation, we will introduce additional "dummy" assertions—$\bot$ preceded by any number of $*$—into the model; obviously, any label σ instantiates some $*^i$ of the same length, which provides a universal assertion matching the universal statement expressed by $\sigma\, F$. Furthermore, the atom $\bot$ ensures that F is trivially verified . This is true for *any* $\mathcal{LBL}$-formula $\sigma\, F$. To ensure that these extra assertions do not introduce a clash, we must ensure that none of their labels has any realized ground instances; we achieve this by specifying all existing labels in M as

[6]To be precise, a formula is equally satisfiable in each logic where it is defined. In $\mathcal{LBL}_w$, we do not have formulas whose labels involve conditional constants.

exceptions to the new labels $*^i$. We call the resulting model $s(M)$ saturated, because every ground label γ is either realized in M, or the set $M_{\langle\gamma\rangle}$ is utopian. Consequently, whenever F is *true* in all realized ground instances of σ in M (that is, $M \models_l \sigma F$, in $s(M)$ this F becomes universally *true* in all ground instances of σ (which establishes $s(M) \models_s \sigma F$). Thus we have generated a strong model from a weak one.

Example 5.7.1 The model $M = \{1\,p, 21\,q\}$ is a weak model for $F = *(p \vee *q)$. To obtain $s(M)$, we add the assertions $A_1 = (* - \{1,2\})\,\bot$, $A_2 = (** - \{21\})\,\bot$, and $A_3 = ***\,\bot$. Let us ascertain that $s(M) \models_s F$: First, we must have $(s(M))_{\langle * \rangle} \models_s p \vee *q$. Since A_1 contributes the assertion $\bot$ into $(s(M))_{\langle * \rangle}$, this is trivially true. In fact, $\{A_1\}_{\langle * \rangle} \models_s p \vee *q$, so when recursively checking exceptions, we only need to consider all exceptions in A_1. So next we show $(s(M))_{\langle 1 \rangle} \models_s p \vee *q$. This is witnessed by the assertion p contributed by the original set M, which verifies the first disjunct p. Secondly, we show $(s(M))_{\langle 2 \rangle} \models_s p \vee *q$, claiming that $(s(M))_{\langle 2 \rangle} \models_s *q$. So we construct $(s(M))_{\langle 2* \rangle} = \{\bot, *\bot\}$. Again this set trivially verifies anything, so $(s(M))_{\langle 2 \rangle} \models_s *q$ is established. The only exception in $(s(M))_{\langle 2 \rangle}$ is the label 1. Now $(s(M))_{\langle 21 \rangle} = \{q, *\bot\}$ verifies q, so $(s(M))_{\langle 2 \rangle} \models_s *q$ and hence $s(M) \models_s F$ have been verified. We leave it to the reader to check that none of the three labels $* - \{1,2\}$, $** - \{21\}$, $***$ is realizable, showing that $S(M)$ is clash-free and hence a model for F.

Definition 5.7.2 *Given a set M of assertions with simple labels, with $d(M)$ denoting the depth of M as usual, define the* saturation *of M as*

$$s(M) = M \cup \{\underbrace{(*^i - \Sigma_i)}_{\lambda_i}\bot : i = 1, \ldots, d(M)+1\},$$

where $\Sigma_i = \{\sigma c : \sigma c \text{ ex. in } M, |\sigma| = i-1, c \in D\}$.

Notice that i ranges up to $d(M)+1$ so as to introduce a label $*^{d(M)+1}\,\bot$, which guarantees that $(s(M))_{\langle\gamma\rangle}$ is utopian for any ground label γ of length greater than $d(M)$. For all other i from 1 to $d(M)$, the set Σ_i collects all prefixes of labels in M which end in a constant at position i. Our first observation on our new definition is:

Proposition 5.7.3 *A ground label $\gamma \in D^*$ is nontrivially realized in M iff it is nontrivially realized in $s(M)$.*

Proof: The "only if" part is obvious, as $s(M) \supseteq M$ and being nontrivially realized is a monotone property. We show the "if" part via the contrapositive: If γ is not nontrivially

realized in M, then it must have some prefix $\gamma' c$ so that no corresponding σc exists in M with $\sigma \sqsubseteq \gamma'$. But this is also true for $s(M)$, since all new assertions in $s(M)$ have main labels of the form $*^i$ without constants. Therefore, γ is not nontrivially realized in $s(M)$. □

Proposition 5.7.4 *The λ_i introduced into $s(M)$ have no nontrivial realized ground instances in $s(M)$. Furthermore, M is clash-free iff $s(M)$ is clash-free.*

Proof: Assume to the contrary that some λ_i has a nontrivial rgi. Since $i \geq 1$, this rgi cannot be ε, so we can write it as γc. The constant c (which is in the ith position) must be witnessed by some label σc with $\sigma \sqsubseteq \gamma$. For lack of constants in the main labels of $s(M) - M$, this label σc must exist in M. Since σc is of length i, it is included in Σ_i. In other words, γc instantiates an exception of λ_i, so it cannot be an instance of λ_i, contradicting our assumption.

The "if" part of the second proposition is trivial, since every subset of a clash-free set is clash-free. As to the "only if" part, suppose $s(M)$ contains a clash which does not exist in M. Since all labels introduced are of the form $\lambda_i \perp$, the clash must be of this form also, with λ_i being realizable in $s(M)$. Because of the first part of this proposition, λ_i does not have any nontrivial rgi, so it has some trivial rgi. Then according to the remark after Corollary 5.6.5, $s(M)$ must contain another assertion $\lambda \perp$ so that λ has a nontrivial rgi. By Proposition 5.7.3 above, this rgi is also a nontrivial rgi in M. Again none of the new labels in $s(M)$ have nontrivial rgis, so $\lambda \perp$ must have already existed in M. This shows that M has a clash, which completes the proof. □

Proposition 5.7.5 *For any $\gamma \notin I(M)$, $(s(M))_{\langle\gamma\rangle}$ is utopian.*

Proof: If γ is not realized, then (just as in the proof of Proposition 5.7.3) γ has a prefix $\gamma' c$ so that $\sigma \not\sqsubseteq \gamma'$ for any σc which may occur in M. Let k be the length of $\gamma' c$, and consider the labels $\sigma c'$ in Σ_k. We either have $c \neq c'$ or $\sigma \not\sqsubseteq \gamma'$, as we just stated. In either case, $\sigma c' \not\sqsubseteq \gamma' c$, that is, $\gamma' c$ does not instantiate any exception in Σ_k, and hence $\lambda_k \sqsubseteq \gamma' c$. From Corollary 5.6.4 and its proof we obtain that $(s(M))_{\langle\gamma' c\rangle}$ is utopian and that the same holds for any ground label extending $\gamma' c$, so $(s(M))_{\langle\gamma\rangle}$ in particular is utopian. □

Upon combining this result and Proposition 5.7.3, we see that every nontrivial realized ground label in M is a nontrivial realized ground label in $s(M)$ and all other ground labels

are trivial realized ground labels in $s(M)$. We have thus covered every ground label in D^*. Let us now study the implications on constant-wise realizability:

Proposition 5.7.6 *A simple label $\overline{\sigma}$ is constant-wise realizable in M iff it is strongly constant-wise realizable in $s(M)$.*

Proof: This property follows easily provided $P_c(\sigma, M)$ and $Q_c(\sigma, s(M))$ are shown equivalent for all prefixes σc of $\overline{\sigma}$; we prove this equivalence for *any* σc.
As to the "if" part, Proposition 5.6.7 shows that $Q_c(\sigma, s(M))$ implies $P_c(\sigma, s(M))$. To prove $P_c(\sigma, M)$, take $\gamma \in I(\sigma, M)$. First, note that γ is also an rgi of σ in the bigger set $s(M)$. Then because of $P_c(\sigma, s(M))$ we get $\gamma c \in I(s(M))$. Assume γc is a trivial rgl, then Corollary 5.6.5 entails the existence of a prefix γ' which is a nontrivial rgl and instantiates a clash witness in $s(M)$. Propositions 5.7.3 and 5.7.4 further show that γ' is a nontrivial rgl in M and cannot instantiate any label in $s(M) - M$. Therefore, the clash—of the form $\sigma' \perp$ with $\sigma' \sqsubseteq \gamma'$—exists in M, and by Corollary 5.6.4, all extensions of γ', including γc, are realized in M. Secondly, if γc is a nontrivial rgl, then Proposition 5.7.3 shows directly that γc is a (nontrivial) rgl in M also. In either case we have verified $\gamma c \in I(M)$. Since this is true for all rgis γ, we derive $P_c(\sigma, M)$.
For the converse, suppose $P_c(\sigma, M)$, and take *any* instance γ of σ. First, if γ is an rgi of σ in M, then γc is also realized in M thanks to $P_c(\sigma, M)$, and since being an rgl is a monotone property, γc is an rgl in $s(M) \cup \{\gamma\ \top\}$. Secondly, if γ is not an rgi of σ in M, then by Proposition 5.7.5 $(s(M))_{\langle\gamma\rangle}$ is utopian, and so is $(s(M) \cup \{\gamma\ \top\})_{\langle\gamma\rangle}$. Since γ is nontrivially realized in $s(M) \cup \{\gamma\ \top\}$, Condition 3. of Definition 5.6.3 applies, showing that $\gamma c \in I(s(M) \cup \{\gamma\ \top\})$. We have thus shown this for all ground instances γ of σ, and $Q_c(\sigma, s(M))$ follows. □
As a final preparatory result, we state that saturation and subscripting by some realized ground label are interchangeable:

Lemma 5.7.7 *For any realized ground label γ in M, we have $s(M_{\langle\gamma\rangle}) = (s(M))_{\langle\gamma\rangle}$.*

Proof: Set $d = d(M)$, $k = |\gamma|$, as well as

$$s(M_{\langle\gamma\rangle}) = M_{\langle\gamma\rangle} \cup \{(*^i - \Sigma'_i)\ \perp, i = 1, \ldots, d - k + 1\}$$

(where we have already used the fact that $d(M_{\langle\gamma\rangle}) = d - k$, and Σ'_i will be specified later),

and

$$\begin{aligned}(s(M))_{\langle\gamma\rangle} &= (M \cup \{(*^j - \Sigma_j)\,\bot, j = 1, \ldots, d+1\})_{\langle\gamma\rangle} \\ &= M_{\langle\gamma\rangle} \cup \{(*^{j-k} - (\Sigma_j)_{\langle\gamma\rangle})\,\bot, j = k+1, \ldots, d+1\}.\end{aligned}$$

To justify the latter, obvious-looking equality, we must ascertain that $\varepsilon\ \bot$ does not "accidentally" get introduced into $(s(M))_{\langle\gamma\rangle}$. First, if $\varepsilon\ \bot \notin M_{\langle\gamma\rangle}$, then γ and all its prefixes γ' are *nontrivial* rgls. By Proposition 5.7.4 we can infer that $\lambda_i \not\sqsubseteq \gamma'$ and further $\lambda_i \not\sqsubseteq_p \gamma$. Now Proposition 5.6.2 shows that $(s(M))_{\langle\gamma\rangle}$ does not contain $\varepsilon\ \bot$. On the other hand, if $\varepsilon\ \bot \in M_{\langle\gamma\rangle}$ already, we need not care whether it is also contributed into $(s(M))_{\langle\gamma\rangle}$ by one of the assertions $\lambda_j\ \bot$, $j \leq k$.

On comparing $s(M_{\langle\gamma\rangle})$ and $(s(M))_{\langle\gamma\rangle}$ and setting $j = i + k$, we see that we only need to check $(\Sigma_j)_{\langle\gamma\rangle} = \Sigma'_i$. Observe that the labels in Σ_j are exactly the labels of length j occurring in M which end in a constant, and upon subscripting by γ, their length gets reduced to $j - k$ (and the final constant never gets chopped off, since $j - k \geq 1$). Likewise, the labels in Σ'_i are of length i and end in a constant. Furthermore, we have:

$$\begin{aligned}\sigma c \in \Sigma'_i \quad &\text{iff} \quad \sigma c \text{ ex. in } M_{\langle\gamma\rangle} \\ &\text{iff} \quad \text{some } \sigma_0\sigma c \text{ ex. in } M,\ \sigma_0 \sqsubseteq \gamma \\ &\text{iff} \quad \text{some } \sigma_0\sigma c \text{ ex. in } \Sigma_j,\ \sigma_0 \sqsubseteq \gamma \\ &\text{iff} \quad \sigma c \in (\Sigma_j)_{\langle\gamma\rangle}.\end{aligned}$$

This shows that indeed $(\Sigma_j)_{\langle\gamma\rangle} = \Sigma'_i$ for all $i = 1, \ldots, d - k + 1$, $j = i + k$; which completes the proof of $s(M_{\langle\gamma\rangle}) = (s(M))_{\langle\gamma\rangle}$. □

And finally, here is our equisatisfiability result:

Theorem 5.7.8 *For any clash-free set of assertions M, $M \models_l F$ iff $s(M) \models_s F$.*

Proof: By Proposition 5.7.4, M is clash-free iff $s(M)$ is. We will now show that $M \models_l F$ iff $s(M) \models_s F$, given clash-freeness. First, note that M cannot be utopian (nor can $s(M)$), so $M \models_l F$ is never true because of the trivial condition stated in Definition 5.6.8. Furthermore, clash-freeness implies that all rgls of M are nontrivial. We prove the theorem by induction on the structure of F, only doing the induction step for labellings and leaving propositional concatenation and the base case (propositional atoms) to the reader.

So assume the theorem shown for a given formula F and any model M, and consider $\sigma\ F$, where for now σ does not have any existential constants. Then we have:

$s(M) \models_s \sigma F$

iff $(s(M))_{\langle\gamma\rangle} \models_s F$ for all ground instances γ of σ (Thm. 5.3.9, Statement 5)

iff $(s(M))_{\langle\gamma\rangle} \models_s F$ for all $\gamma \in I(\sigma, s(M))$ (trivial when γ is not realized, as $(s(M))_{\langle\gamma\rangle}$ is utopian, Proposition 5.7.5)

iff $s(M_{\langle\gamma\rangle}) \models_s F$ for all $\gamma \in I(\sigma, M)$ (Lemma 5.7.6, Prop. 5.7.3)

iff $M_{\langle\gamma\rangle} \models_l F$ for all $\gamma \in I(\sigma, M)$ (induction hypothesis)

iff $M \models_l \sigma\, F$ (Theorem 5.5.2: σ is cwr, since it has no existential constants).

For labels σ with existential constants, we reason as follows:

$s(M) \models_s \sigma\, F$

iff σ is scwr in $s(M)$, and $s(M) \models_s [\sigma]\, F$

iff σ is cwr in M (Prop. 5.7.6), and $M_{\langle\gamma\rangle} \models_l F$ for all $\gamma \in I(\sigma, M)$ (first part)

iff $M \models_l \sigma\, F$ (Theorem 5.5.2).

The proof follows by induction on the structure of F. □

This theorem makes a remarkable contribution to our theory. Not only did we convert a weak model M into a strong model $s(M)$ for exactly the same set of formulas; but we did so with only moderate size increase, namely $O(d)$ extra formulas, each of which specifies $O(\#M)$ exceptions in its label. This shows that minimal-size strong models are not significantly larger than minimal-size weak models. Another remarkable fact is that $s(M)$ satisfies *all* formulas which M satisfies. In practice however, the strong model we constructed in this fashion is of rather poor quality: $s(M)$ is deprived of most of the extensibility typical for strong models: normally, if we add another formula F to S, a strong model for S can be extended into a strong model for $S \cup \{F\}$ through additional assertions verifying F; only sometimes do clashes occur, and usually they can be repaired locally. But in $s(M)$, an additional assertion which introduces a new realized ground label to $s(M)$ always leads to a clash. We can obtain even smaller strong models for a *specific* set of $\mathcal{LBL}$-formulas S[7] which are nonetheless superior in terms of extensibility, in a different way:

Example 5.7.9 Given a set $S = \{*\,(p \vee q), *\,(\neg p \vee \neg q), 1\,\top, 2\,\top\}$ and a model $M = \{1\,p, 2\,q\}$, we can extend this model to $M' = \{1\,p, 2\,q, (* - \{2\})\,p, (* - \{2\})\,\neg q, 2\,\neg p\}$; we easily see

[7] Of course, we will eventually be able to generate models directly from S, once we have developed our algorithm. The ability to create a strong model from a weak model is mostly of theoretical interest.

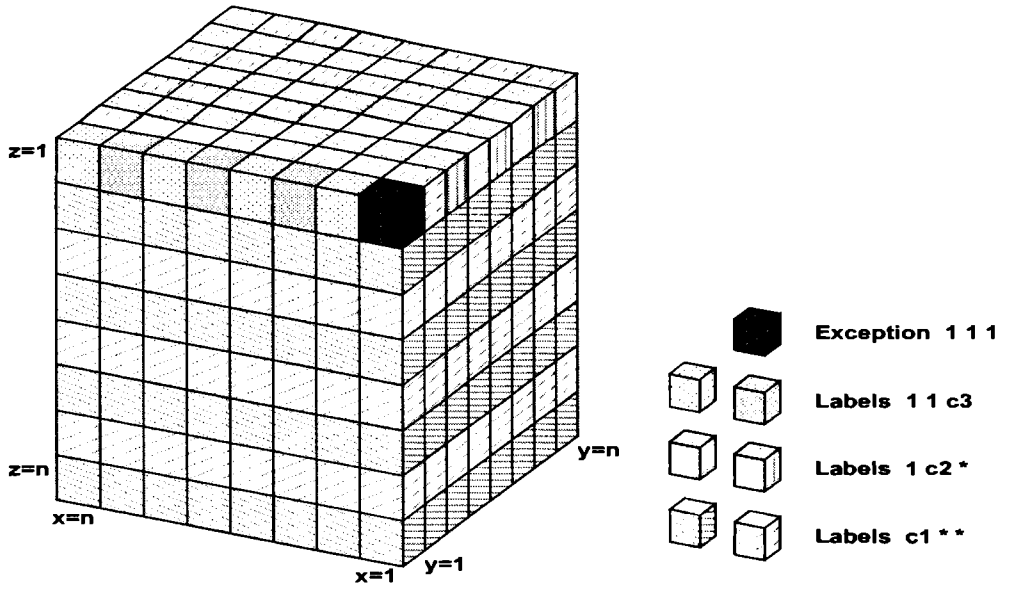

Figure 5.5: The label $*** - \{111\}$, expressed with simple labels.

that M' is clash-free and $M' \models_s S$. If we now add a formula $3\ \top$ or even $4\,p$ to S, we can get a strong model simply by adding the same formulas to M' as well. In the case of a new formula $5\ \neg p$, we can do the same; however, we will then have to repair the ensuing clash by adding 5 as another exception to $(* - \{2\})\,p$ and $(* - \{2\})\ \neg q$, and including another assertion $5\,q$ in order to verify $*\,(p \vee q)$. These repairs go beyond simple extensions of M', but they are local to assertions involving the instance 5.

The reason why we can construct strong models with less overhead is that they are tailored to satisfy one *particular* set of formulas. By contrast, the model $s(M)$ is designed to satisfy *all* formulas weakly satisfied by M. In doing so, extensibility is sacrificed.
Now for the second task: providing an algorithm for converting an $\mathcal{LBL}_x$-model into an $\mathcal{LBL}_w$-model. How can the same set of realized ground instances expressed by labels with exceptions be expressed using simple labels only?

Example 5.7.10 Consider a cubes model M as in Example 4.1.1, in which all ground labels of length 3 with constants 1 through n are realized, thanks to some assertions in M with labels $i**$, $*i*$, $**i$, $i = 1, \ldots, n$. Suppose further that M contains an assertion $\lambda\,p$ with the

label $\lambda = *** - \{111\}$, representing the entire $n \times n \times n$ cube minus the elementary cube at position $(1,1,1)$, as illustrated in Figure 5.5. Using simple labels, we cannot express this set difference. Instead, we "slice up" the entire block into subspaces (planes, rows, or single cubes) whose set union represents the same set as that represented by λ. For instance, the set of assertions $w(M) = \{2** \; a, \ldots, n** \; a, 12* \; a, \ldots, 1n* \; a, 112 \; a, \ldots, 11n \; a\}$ adequately replaces $\lambda \; a$, and it contains only simple labels. We can regard $w(M)$ as an *expansion* of the model M.

In the general case, we also need to eliminate conditional constants $[c]$ from all labels σ; this can be accomplished by unifying these labels with all other labels in M which have a matching existential constant c in the same position.

In our algorithm, we do both steps (dividing up a complex label into blocks, and eliminating conditional constants) simultaneously, and in increasing order of positions: With $d = d(M)$, we define $M_0 = w_0(M) = M$, $M_i = w_i(\ldots(w_1(M)) = w_i(M_{i-1})$, $i = 1, \ldots, d$, and $w(M) = M_d$, where each $w_i(\cdot)$ "works" on the ith position of all labels in M_{i-1}. We presume that M is clash-free and devoid of assertions with trivial labels and that all labels are normalized. We assume the following invariant, to be proved inductively later: If $(\sigma - \Sigma) \; a$ is an assertion in M_i, then all exceptions in Σ are of length at least $i + 1$, and their first i positions are all identical to those of σ. Furthermore, none of the first i positions in σ are conditional constants. Finally, M_i is clash-free. Note that the invariant holds for $i = 0$, since M contains no trivial labels. The algorithm computes $w_i(M_{i-i})$ for a model M_{i-1} satisfying the above invariant (for $i - 1$). We will use the following notational conventions: Any simple label (main label or exception) processed by the algorithm is written in the form $\sigma_0 x \sigma_1$, where x denotes the ith position of the entire label; the algorithm does not touch any labels of length *less* than i.

Algorithm 5.7.11 *Elimination of exceptions and conditional constants*

Parameters:

M': the current model

Returns:

$w_i(M')$: a one-step conversion of M'

1. *For each assertion* $(\sigma_0 * \sigma_1 - \Sigma)\ a \in M'$ *so that* Σ *contains at least one label with a (conditional or existential) constant in its* i*th position:*

   Replace this assertion by $\sigma_0\ \top$*, and introduce new assertions*

   $$A = ((\sigma_0, \sigma_0')c\sigma_1 - (\sigma_0, \sigma_0')c\Sigma_{\langle \sigma_0 c \rangle})\ a,$$

   where $\sigma_0' c$ *ranges over all labels of this form which exist as prefixes of main labels in* M'*, so that* (σ_0, σ_0') *exists;* $(\sigma_0, \sigma_0')c\Sigma_{\langle \sigma_0 c \rangle}$ *is the usual shorthand notation.*

2. *For each assertion* $(\sigma_0[c]\sigma_1 - \Sigma)\ a \in M'$*:*

   Replace this assertion by $\sigma_0\ \top$*, and introduce new assertions*

   $$A = ((\sigma_0, \sigma_0')c\sigma_1 - (\sigma_0, \sigma_0')c\Sigma_{\langle \sigma_0 c \rangle})\ a,$$

   where $\sigma_0' c$ *ranges over all labels of this form which exist as prefixes of main labels in* M' *so that* (σ_0, σ_0') *exists (*c *is existential but otherwise the same constant as in the label we replace).*

3. *For every assertion* $(\sigma_0 x \sigma_1 - \Sigma)\ a \in M'$*:*

   If Σ *contains a label* $\sigma_0 x$*, replace the entire assertion by* $\sigma_0 x\ \top$[8]*.*

It is easy to see that all assertions in M_i remain normalized. Furthermore, thanks to Step 2, all conditional constants in the ith position get replaced by existential constants. All replaced assertions in Steps 1, 2, and 3 have simple labels, whereas in each new assertion A in Steps 1 and 2, the first i positions in all exceptions are identical to those in its main label. Finally, an assertion remains unchanged:

- if the ith position in its main label is $*$, and the ith position in all exceptions (if any) is also $*$,
- if the ith position in its main label is c. Since the label of A is normalized, the ith position in each exception (all of which are of length at least i, as our invariant states!) must instantiate c, that is, be equal to c.

[8] *The label of this assertion can be truncated even more: any trailing* $*$ *and conditional constants can also be deleted. An assertion* $\sigma\ \top$ *with existential constants can be shortened or even deleted, so long as* σ *remains constant-wise realizable (for instance when* σ *exists somewhere else in* M'*). In our examples of this section, we will point out which labels are redundant and can be omitted, but we will refrain from establishing a general result in order to keep things simple.*

In either case, the ith position in all exceptions is identical to that in the main label of A. This is also true for all positions less than i, since the invariant was assumed to hold for $i-1$. Finally, any assertion whose label has an exception of length at most i (which at this point is necessarily identical to a prefix of the main label) gets truncated to an assertion with a simple label through Step 3. This shows that the invariant is preserved in M_i.
Now notice that the invariant for $i = d$ states that all exceptions are of length at least $d+1$. Since the algorithm does not increase the length of any main label or exception and all labels in M_d are normalized, no such exceptions can exist, which shows that $w(M) = M_d$ has only simple labels. Furthermore, in all positions 1 through d, all conditional constants have been eliminated, so $w(M)$ is indeed a well-formed set of assertions in $\mathcal{LBL}_w$.
Before we proceed, let us illustrate the algorithm on Example 5.7.10: We are given the model M containing the simple labels $i{*}{*}$,$*i{*}$, $**i$, $i = 1, \ldots, n$, and the sole assertion with a complex label, $(*** - \{111\})\, p$. Here is how the algorithm will proceed:

- For $i = 1$, Step 1 applies and $(*** - \{111\})\, p$ gets replaced by $\varepsilon\ \top$ (which is redundant), $(1** - \{111\})\, p$, and $c_1{*}{*}\, p$, for $c_1 = 2, \ldots, n$.
- For $i = 2$, Step 1 applies again to the new assertion $(1** - \{111\})\, p$ which then gets replaced by $1\ \top$ (redundant), $(11* - \{111\})\, p$, and $1c_2{*}\, p$, for $c_2 = 2, \ldots, n$.
- For $i = 3$, Step 1 applies once more to the new assertion $(11* - \{111\})\, p$, which gets replaced by $11\ \top$ (again redundant), $(111 - \{111\})\, p$, and $11c_3\, p$, for $c_3 = 2, \ldots, n$.
- Now Step 3 applies to the assertion $(111 - \{111\})\, p$, truncating it to $111\ \top$.
- We see that all assertions combined constitute exactly the set $w(M)$ we suggested in Example 5.7.10. Apart from a few assertions we found redundant, they represent the various shaded blocks in Figure 5.5. Unlike the other new assertions which specify that p holds in these blocks, $111\ \top$ merely states that cube 111 exists.

We will now show that weak satisfiability is preserved throughout the algorithm, that is: for all $i = 1, \ldots, d$, if M satisfies the above invariant, then $M_{i-1} \models_l F$ iff $M_i \models_l F$. We will obtain this through a sequence of propositions. Note that some of these propositions will specifically refer to the sets M_{i-1}, M_i, etc. in the sequence from M to $w(M)$, whereas others will refer to "some" set of assertions M' satisfying the invariant for $w_i(\cdot)$, just as in

the algorithm. We will need to use the latter, more general sets in some of the recursive proofs to come.

Proposition 5.7.12 *For assertions with empty labels, we have $\varepsilon\, a \in M_{i-1}$ iff $\varepsilon\, a \in M_i$, except perhaps for an extra $\varepsilon\, \top$ in M_1. Hence $M_{i-1} \models_l a$ iff $M_i \models_l a$ for any propositional atom a.*

Proof: Assertions with label ε are not affected by any step of the algorithm, since i is always greater than 0; neither does their presence give rise to any new assertions in M in Steps 1 and 2, as ε is never of the form $\sigma_0 c$. Nor do new assertions with empty labels ever get produced by the algorithm, except perhaps for $\varepsilon\, \top$, which might be produced in Steps 1 or 2 for $i = 1$. The second part is true because the presence of an extra $\varepsilon\, \top$ in M_i does not affect the $\models_l$ relation, as we showed in Lemma 4.4.12. □

Proposition 5.7.13 *$M_{i-1} \models_l F \Leftrightarrow M_i \models_l F$, viewed as a property of F, is preserved under propositional concatenation.*

Proof: Like many times before, we call upon Theorem 2.1.18 showing this. □

Proposition 5.7.14 *If $(\sigma_0, \sigma_0')c\sigma_1 - (\sigma_0, \sigma_0')c\Sigma_{\langle \sigma_0 c\rangle}) \sqsubseteq \gamma$ for the label in the assertions introduced in Steps 1. and 2. and for a ground label γ, then γ also instantiates the original label $\sigma_0{*}\sigma_1 - \Sigma$ or $\sigma_0[c]\sigma_1 - \Sigma$.*

Proof: The label γ must be of the form $\gamma_1\gamma_2$ so that $(\sigma_0, \sigma_0')c \sqsubseteq \gamma_1$ and $\sigma_1 \sqsubseteq \gamma_2$. Since $\sigma_0{*} \sqsubseteq (\sigma_0, \sigma_0')c$, we see that γ instantiates the main label of $\sigma_0{*}\sigma_1 - \Sigma$. Now because of the invariant, all exceptions in Σ have σ_0 as a prefix. Furthermore, γ_1 does not instantiate any prefixes of exceptions of the form $\sigma_0 c'$, $c' \neq c$ (since its ith position is c), and for any $\sigma_0 x\xi \in \Sigma$ so that $\sigma_0 x \sqsubseteq \gamma_1$, we have $\xi \in \Sigma_{\langle \sigma_0 c\rangle}$, hence $\xi \not\sqsubseteq_p \gamma_2$, which shows that $\sigma_0 x\xi \not\sqsubseteq_p \gamma$. To summarize, no exception in Σ instantiates a prefix of γ, so we have proved that $\sigma_0{*}\sigma_1 - \Sigma \sqsubseteq \gamma$. For $\sigma_0[c]\sigma_1 - \Sigma \sqsubseteq \gamma$, the proof is analogous. □

Proposition 5.7.15 *A constant c exists in M' iff it exists in $w_i(M')$.*

Proof: Consider first the case $i = 1$. Then Steps 1 and 2 do not apply to any of the assertions in M' whose main label begins with an existential constant, whereas Step 3, if it applies, preserves the constant $x = c$ in the first position. Conversely, any existential

constant in the new assertions in $w_i(M')$, introduced through Steps 1 and 2, must have existed before (in some other assertion) in M'.

For $i > 1$, the constant c is the first position in the prefix σ_0 of $\sigma_0 x \sigma_1$. Upon truncating any label through Steps 1, 2, or 3, σ_0 is always preserved, and with it the constant c. Conversely, any existential constant in a new assertion in $w_i(M')$ is introduced through unification of two labels σ_0 and σ_0' in M', one of which has this existential constant in its first position; so the constant must already have existed in M'. □

The next result is rather technical, but key to our proof. It states that applying the algorithm and subscripting are "orthogonal":

Proposition 5.7.16 *For any set M' satisfying the invariant for $i-1$, and any constant $c' \in D$ which exists in M', $(w_i(M'))_{\langle c' \rangle}$ and $w_{i-1}(M'_{\langle c' \rangle})$ are identical, except perhaps for an extra $\varepsilon\ \top$ in $(w_i(M'))_{\langle c' \rangle}$.*

Proof: We presumed that M' is a model, so neither M' nor $M'_{\langle c' \rangle}$ for any c' existing in M' can be utopian, which excludes trivial cases. Let us consider $i = 1$ first, in which case we must show $(w_1(M'))_{\langle c' \rangle} = w_0(M'_{\langle c' \rangle}) = M'_{\langle c' \rangle}$:

We consider each of Steps 1, 2, and 3 separately. Note that $\sigma_0 = \sigma_0' = \varepsilon$. In Step 1 the assertion $(*\sigma_1 - \Sigma)\ a$ gets replaced by the trivial $\varepsilon\ \top$ plus $(c\sigma_1 - c\Sigma_{\langle c \rangle})\ a$ for all c existing in M', which includes c'. Upon subscripting by c', only assertions whose main labels begin with c' may contribute to $M'_{\langle c' \rangle}$ and $(w_1(M'))_{\langle c' \rangle}$, respectively. If c' itself is an exception in Σ, then it is also an exception in $c'\Sigma_{\langle c' \rangle}$, and neither the original assertion $(*\sigma_1 - \Sigma)\ a$ nor any of the replacement assertions contribute to $M'_{\langle c' \rangle}$ and $(w_1(M'))_{\langle c' \rangle}$, respectively. Otherwise, they both contribute the same assertion $(\sigma_1 - \Sigma_{\langle c' \rangle})\ a$ into either set.

In Step 2 the reasoning is analogous, except only when $c = c'$ (and $c' \notin \Sigma$ as before) will $([c]\sigma_1 - \Sigma)\ a$ and its replacements $\varepsilon\ \top$ and $(c\sigma_1 - c\Sigma_{\langle c \rangle})\ a$ contribute the (same) assertion $(\sigma_1 - \Sigma_{\langle c' \rangle})\ a$ to $M'_{\langle c' \rangle}$ and $(w_1(M'))_{\langle c' \rangle}$, respectively.

Finally, the label in Step 3 is of the form $(x\sigma_1 - \Sigma)\ a$, and x is an exception in Σ. If x is neither c' nor $*$, $(x\sigma_1 - \Sigma)\ a$ does not contribute to $M'_{\langle c' \rangle}$; nor does the replacement assertion $x\ \top$. But if $x = c'$ or $x = *$, c' instantiates an exception in Σ, so again the label does not contribute to $M'_{\langle c' \rangle}$. On the other hand, the replacement assertion $x\ \top$ in $w_1(M')$ contributes the assertion $\varepsilon\ \top$ to $(w_1(M'))_{\langle c' \rangle}$.

Now consider $i > 1$. Observe that in all three steps both the original assertion $(\sigma_0 x \sigma_1 - \Sigma)\ a$ and its replacement begin with σ_0, as do all exceptions in Σ, in accordance with our invariant.

And the positions in σ_0 do not play an active part in determining whether any of the steps should be applied. Neither assertion contributes anything to $M'_{\langle c'\rangle}$ and $(w_i(M'))_{\langle c'\rangle}$, respectively, unless σ_0 begins with c' or $*$. For the rest of the proof, assume this to be the case. Then x is found in the ith position in $(\sigma_0 x \sigma_1 - \Sigma)\ a$ iff it is found in the $(i-1)$st position in the corresponding assertion in $M'_{\langle c'\rangle}$. The same is true for a constant in the ith position of a label in Σ (the criterion for applying Step 1), so each step applies in the same way to corresponding assertions in M' and $M'_{\langle c'\rangle}$. Subscripting only eliminates the first position from σ_0 and from all exceptions in Σ, which is clearly interchangeable with the replacement (or truncation, if you will) of assertions in Steps 1, 2, and 3.

For the new assertions $A = ((\sigma_0, \sigma'_0)c\sigma_1 - (\sigma_0, \sigma'_0)c\Sigma_{\langle \sigma_0 c\rangle})\ a$ introduced in Steps 1 and 2, notice that A contributes an assertion $A_{c'}$ to $(w_i(M'))_{\langle c'\rangle}$ iff (σ_0, σ'_0) begins with c' or $*$, that is, both σ_0 and σ'_0 begin with c' or $*$. But then the assertion A' in which the label $\sigma'_0 c$ occurred contributes an assertion $A'_{c'}$ to $M'_{\langle c'\rangle}$ (and c is found in the $(i-1)$st position in the label of A'). So the assertion $A_{c'}$ in $(w_i(M'))_{\langle c'\rangle}$ also arises as an additional assertion from Step 1 or 2, applied to $M'_{\langle c'\rangle}$ while computing $w_{i-1}(M'_{\langle c'\rangle})$. (Here $A'_{c'}$ provides the unifying label in $M'_{\langle c'\rangle}$.) This completes the proof that these two sets are equal. □

Corollary 5.7.17 *For any constant $c \in D$ which exists in M, $(w(M))_{\langle c\rangle}$ and $w(M_{\langle c\rangle})$ are identical, except perhaps for an extra $\varepsilon\ \top$ in $(w(M))_{\langle c\rangle}$.*

Proof: This is just Proposition 5.7.16, iterated over $i = 1, \ldots, d$. □

Proposition 5.7.18 *For any set M' satisfying the invariant for $w_i(\cdot)$, M' and $w_i(M')$ have the same set of realized ground labels.*

Proof: The empty label ε is realized in all sets. For a nonempty ground label $c\gamma$, we use induction on i. Assuming the proposition shown for $i-1$, we reason as follows:

$c\gamma \in I(w_i(M'))$

iff c exists in $(w_i(M'))$, and $\gamma \in I((w_i(M'))_{\langle c\rangle})$ (Lemma 5.4.9, parts (2),(3))

iff c exists in M', and $\gamma \in I((w_{i-1}(M'_{\langle c\rangle}))_{\langle c\rangle})$ (Prop. 5.7.15, Prop. 5.7.16)

iff c exists in M', and $\gamma \in I(M'_{\langle c\rangle})$ ($i > 1$: Ind. Hyp.; $i = 1$: $w_0(M_{\langle c\rangle}) = M_{\langle c\rangle}$)

iff $c\gamma \in I(M')$ (Lemma 5.4.9, parts (2),(3)).

The extra assertion $\varepsilon\ \top$ which may be found in $(w_i(M'))_{\langle c\rangle}$ but not $w_{i-1}(M'_{\langle c\rangle})$ does not affect the set of their rgl. The proof follows by induction on i. □

Corollary 5.7.19 *If M_{i-i} is clash-free, so is M_i.*

Proof: Assume M_i contains a clash. Clashes are characterized by the existence of an rgl γ in M_i—which by Proposition 5.7.18 is also an rgl in M_{i-1}—so that M_i contains $\lambda_1 \perp$ or two complementary assertions $\lambda_1 p$, $\lambda_2 \neg p$ so that $\lambda_1 \sqsubseteq \gamma$ (and $\lambda_2 \sqsubseteq \gamma$). Other than assertions containing $\top$ which do not participate in clashes and assertions taken over unchanged from M_{i-i}, the only new assertions in M_i are those introduced in Steps 1. and 2. For these, Proposition 5.7.14 warrants the existence of some label(s) $\lambda_1' \sqsubseteq \gamma$ (and $\lambda_2' \sqsubseteq \gamma$) so that $\lambda_1' \perp$ (or $\lambda_1' p$ and $\lambda_2' \neg p$) are assertions in M_{i-i}. But since γ is realized, these assertions constitute a clash in M_{i-1} as well. □

Proposition 5.7.20 *If $M \models_l F \Leftrightarrow w(M) \models_l F$ for all clash-free sets M of assertions (viewed as a property on formulas) holds for a formula F, then it also holds for $* F$, $c F$, and $[c] F$, $c \in D$.*

Proof: For existential constants, we reason as follows (over an arbitrary model M):

$M \models_l c F$

iff c exists in M and $M_{\langle c \rangle} \models_l F$

iff c exists in $w(M)$, and $w(M_{\langle c \rangle}) \models_l F$ (Prop. 5.7.15 iterated, Ind. Hyp.)

iff c exists in $w(M)$, and $(w(M))_{\langle c \rangle} \models_l F$ (Cor. 5.7.17)

iff $w(M) \models_l c F$.

Again the extra assertion $\varepsilon \top$ which may be found in $(w(M))_{\langle c \rangle}$ but not $w(M_{\langle c \rangle})$ does not affect the equivalence of $w(M_{\langle c \rangle}) \models_l F$ and $(w(M))_{\langle c \rangle} \models_l F$.
For $x = [c]$ and $x = *$, we reason analogously, except for $[c]$ we have nothing to show if c does not exist in M and $w(M)$, and for $*$ we reason over all constants which exist in M, which are exactly the constants existing in $w(M)$. □

Theorem 5.7.21 *For any $\mathcal{CBL}$-formula F, the set M is an $\mathcal{CBL}_x$-model for F iff $w(M)$ is an $\mathcal{CBL}_w$-model for F.*

Proof: This follows by induction on the depth of F from shown facts: the equivalence $M \models_l F \Leftrightarrow w(M) \models_l F$ holds for propositional atoms (Proposition 5.7.12, iterated over $i = 1, \ldots, d$), it is preserved under propositional concatenation (Proposition 5.7.13, iterated

over $i = 1, \ldots, d$), and it follows for all atoms of depth $k+1$ provided it holds for all formulas of depth up to k (Proposition 5.7.20). Finally, we showed that clash-freeness of M_i is preserved as an invariant, and since M is clash-free, so is $w(M)$. □

Needless to say, the model $w(M)$ may expand greatly compared to M. After all, assertions cannot be represented as concisely using only simple labels as with complex labels. We illustrate this on the model we found for Example 5.1.1:

Example 5.7.22 Consider the assertion $(*** - \{1**, *1*, **1\})\ h$ in the model M for Example 5.1.1. Recall that all ground labels of length up to 3 with constants 1 through n are realized through labels $i**$, $*i*$, $**i$, existing in M for all $i = 1, \ldots, n$. We also said that $(n-1)^3$ assertions $\gamma\ h$ with ground labels γ are needed to represent all instances in which h is *true*. Our algorithm indeed produces all these assertions:

- For $i = 1$, Step 1. applies, and $(*** - \{1**, *1*, **1\})\ h$ gets replaced by $\varepsilon\ \top$ (which is redundant) and assertions $(1** - \{1**, 11*, 1*1\})\ h$ and $(c_1** - \{c_1 1*, c_1 *1\})\ h$ added, for $c_1 = 2, \ldots, n$.

- For $i = 2$, Step 1. applies again to the new assertions which then get replaced by $1\ \top$ and $c_1\ \top$ (redundant), and assertions $(11* - \{11*, 111\})\ h$, $(1c_2* - \{1c_2*, 1c_2 1\})\ h$, $(c_1 1* - \{c_1 1*, c_1 11\})\ h$, $(c_1 c_2* - \{c_1 c_2 1\})\ h$ added, for $c_1, c_2 = 2, \ldots, n$.

- For $i = 3$, Step 1. applies a third time to any of the new assertions which get replaced by $c_1 c_2\ \top$ ($c_1, c_2 = 1, \ldots, n$; all redundant) and eight kinds of new assertions added; the first seven of these are of the form $(c_1 c_2 c_3 - \{c_1 c_2 c_3\})\ h$, where $c_1 = 1$ or $c_2 = 1$ or $c_3 = 1$, and the last is of the form $c_1 c_2 c_3\ h$, for $c_1, c_2, c_3 = 2, \ldots, n$.

- Now Step 3. applies to all new assertions where $c_1 = 1$ or $c_2 = 1$ or $c_3 = 1$, truncating these to $c_1 c_2 c_3\ \top$.

To conclude this section, let us reflect on Figure 5.4. We accomplished the task of converting an $\mathcal{LBL}_w$-model into an $\mathcal{LBL}_s$-model. We also provided an algorithm for converting an $\mathcal{LBL}_x$-model (which includes all $\mathcal{LBL}_s$-models) into an $\mathcal{LBL}_w$-model. We are left with one missing link—the dotted arrow in Figure 5.4. Can a procedure similar to that of $s(\cdot)$ be constructed, converting an $\mathcal{LBL}_x$-model M into an $\mathcal{LBL}_s$-model? One could, of course, convert M to an $\mathcal{LBL}_w$-model $w(M)$ and then produce a strong model $s(w(M))$ from it.

But as we have seen, the conversion to $w(M)$ hugely increases the size of the model, so $s(w(M))$, though formally an $\mathcal{LBL}_s$-model, will hardly be satisfactory.

Instead we suggest an adaptation of the algorithm for computing $s(M)$ to handle labels with exceptions. The basic idea is to introduce extra assertions $(\xi - \Sigma)\ \bot$, but here ξ does not only range over the universal labels $*^i$ but also over all exceptions in the assertions of M, so as to render $(s(M))_{\langle\gamma\rangle}$ utopian whenever γ is not a realized ground label. The "exceptions to the exceptions" in Σ must be suitably defined so that no realized ground label in M will instantiate any of the $\xi - \Sigma$. For lack of relevance to the rest of this work, we will not develop this idea further. We do point out that this conversion will result in a similarly moderate size increase as in the conversion $s(\cdot)$.

Chapter 6

A Model-Finding Algorithm

That's not magic—it's logic.

J.K. Rowling, "Harry Potter and the Philosopher's Stone"

6.1 Introduction and Motivation

Equipped with the syntax and strong model semantics of $\mathcal{LBL}$, we are now ready to give an algorithm which finds strong models for a given set S of formulas whenever S is satisfiable, or rejects S whenever it is unsatisfiable. Keeping in mind that models themselves are sets of formulas of a special type (namely, they do not contain the propositional connectives $\wedge$ and $\vee$), we could view model finding as a transformation of S which eliminates all propositional connectives. As we will see shortly, labels distribute over conjunctions, so conjunctions occurring outside any disjunctions are easily eliminated from an $\mathcal{LBL}$-formula. This is a novelty which is not true for the modalities in **K**. The problematic issue is how to eliminate *disjunctions*. The obvious answer, just as in propositional logic, is to perform a tree search over all combinations of disjuncts, commonly referred to as *branching*. This could be done in various ways: breadth-first, branch-and-bound, depth-first, etc. Notwithstanding other approaches being feasible, we will advocate a "branch-and-repair" approach, which is depth-first in that it keeps only one partial solution at a time, thus aiming for optimal use of memory resources; but it does not share the backtracking behaviour of an ordinary depth-first search: a failed branch in a search tree is usually not completely undone; only the parts of the branch which contributed to the failure are repaired, and the rest left untouched.

We could successfully state an algorithm which handles arbitrary $\mathcal{LBL}$-formulas. But a less cluttered description can be given if formulas are preprocessed into a normal form we call *labelled clause normal form* (LCNF), in analogy to the clause normal form frequently used for stating the propositional satisfiability problem. In Section 6.2, we will define the LCNF and how to convert a formula into it.
To illustrate how the search algorithm works, let us reconsider Example 5.1.1:

Example 6.1.1 We are given a set S of formulas containing:

- the formulas listed in (4.2) and (5.1) which define the x-, y-, and z-coordinates:

$$\begin{array}{rrr}
1{*}{*}\,(x=1) & *1{*}\,(y=1) & {*}{*}1\,(z=1) \\
\ldots & \ldots & \ldots \\
n{*}{*}\,(x=n) & *n{*}\,(y=n) & {*}{*}n\,(z=n) \\
(***-\{1{*}{*}\})\,\neg(x=1) & (***-\{*1{*}\})\,\neg(y=1) & (***-\{{*}{*}1\})\,\neg(z=1) \\
\ldots & \ldots & \ldots \\
(***-\{n{*}{*}\})\,\neg(x=n) & (***-\{*n{*}\})\,\neg(y=n) & (***-\{{*}{*}n\})\,\neg(z=n)
\end{array}$$

- the formulas F_1 and F_2 from Example 5.1.1:

$$\begin{aligned}
F_1 &= *** \Big(\neg h \vee (\neg(x=1) \wedge \neg(y=1) \wedge \neg(z=1))\Big) \\
F_2 &= *** \Big(h \vee (x=1) \vee (y=1) \vee (z=1)\Big).
\end{aligned}$$

The first kind of formulas are already assertions. We have shown that assertions satisfy themselves, and we will also show that they can safely be added into any model which does not already contain them, so we will simply include them in the model M we are going to construct. But we must branch on the disjunctions in F_1 and F_2. We see that these disjunctions are within the scope of the label $***$; to be most efficient, we will try to branch simultaneously on as many instances as possible. Let us begin with F_2, which we will try to satisfy *universally* by adding $*** h$ to M. This succeeds (M remains clash-free), so we do the same with F_1, adding $*** \neg h$ to M. As a result, we have a clash with witness $***$, so we undo this choice and try the other disjunct, a conjunction. As claimed, we can distribute the label $***$ over it, adding the assertions $*** \neg(x=1)$, $*** \neg(y=1)$, and $*** \neg(z=1)$ to M. Now M contains three clashes with the existing assertions $1{*}{*}\,(x=1)$, $*1{*}\,(y=1)$, and

$**1$ ($z = 1$), respectively. Notice that each clash witness is a *strict* instance of the branch label $***$, so we need not undo the entire branch. Instead, we add the clash witnesses as exceptions to the branch label, which gives us $(*** - \{1**, *1*, **1\})\ \neg(x = 1)$; same for $y = 1$ and $z = 1$. (These new assertions are subsumed by existing assertions and can be deleted.)

Now we have unsuccessfully tried all disjuncts in formula F_1, so we need to backtrack to our first branching point in F_2—but only over the three clash witnesses, not the label $***$. To keep things simple, let us only pursue the instance $1**$ further: We first introduce this instance as an exception to $*** h$. Now we branch again on F_2, choosing the second disjunct and adding $1**\ (x = 1)$ to M. (In fact, this assertion is already in M.) The clash is hereby removed, but we need to go back to F_1 and satisfy it over the instance $1**$. Now branching on the first disjunct and introducing $1**\ \neg h$ no longer leads to a clash, so we do this and are done for this instance. On repeating this in a similar way for the other two instances $*1*$ and $**1$, we arrive at a satisfying model M containing the following additional assertions:

$$(*** - \{1**, *1*, **1\})\ h \qquad 1**\ \neg h \qquad *1*\ \neg h \qquad **1\ \neg h.$$

This is exactly the model we suggested in Example 5.1.1.

The search can be considerably shortened if we employ techniques like Boolean constraint propagation. For instance, we could mark the disjunct $\neg(x = 1) \wedge \neg(y = 1) \wedge \neg(z = 1)$ in F_1 with the instances $1**$, $*1*$, $**1$, in which it is not satisfiable because complementary assertions already exist in M. Then we can immediately derive $1**\ \neg h$ etc., for $\neg h$ is the only other disjunct available. We can propagate these new assertions further into F_2 where we find a complementary literal in the disjunct h and mark it with these instances. Now when we first branch on F_2, we would select h only over the unmarked part of $***$, that is $(*** - \{1**, *1*, **1\})\ h$. We thus avoid the occurrence of clashes altogether. In Section 6.13, we will outline how Boolean constraint propagation can be incorporated into our general algorithm.

In our example, we observed how some branches were partially undone, whereas others were undone and later revisited on subinstances. If we are not careful, the algorithm may get caught in a nonterminating loop, undoing and redoing the same branch(es) without making any progress towards a satisfying model or towards exhausting the search space. To avoid this, we will have to use additional data structures to guide the search and keep track of its state:

- We consider each disjunct in each formula a potential branch; to each branch, we assign a positive integer, its *branch index*. We do not need to branch in ascending order of indices to ensure termination; but whenever a clash occurs or a formula runs out of disjuncts to branch on, we must undo the branch with maximal index among all branches participating in the clash or causing the failed formula. We describe branches in detail in Section 6.4.

- Whenever we have exhausted a part of the search space without any solution, we block it by rules called *nogoods*. These rules list all branch assignments defining that part of the search space. At any time during the search, we ensure that no nogood is violated, so the algorithm always works in a part of the search space we have not visited before. We describe nogoods in Section 6.5.

- As we have demonstrated in the example, the search is chiefly driven by the need to repair occurrences of complementary assertions and formulas in which no branch can be selected. We call these *standing inconsistencies* and *failed clauses*, respectively. Whenever they occur on a *realizable* label, they must be repaired, or else the current set of assertions cannot be extended into a model. In Section 6.6, we will classify all types of inconsistencies, and in Section 6.8 we will discuss how to repair failed clauses.

Nogood calculi, also known as *dependency-directed backtracking*, are a well-established family of techniques in Constraint Satisfaction Problems [19, 31]. They include techniques such as backjumping [28] and dynamic backtracking [31]; and they have been successfully adapted to propositional logic [24], where they are also referred to as conflict-driven clause learning [89, 77]. All nogood calculi can be viewed as resolution calculi [68, 6].
In adapting a nogood calculus to $\mathcal{LBL}$, we are faced with additional challenges arising from the use of labels, as we will illustrate on the next example:

Example 6.1.2 Find a model for the set S containing the following clauses:

$$C_1 : *(p \vee q), \quad C_2 : (c_1\, \neg p) \vee f, \quad C_3 : (c_2\, \neg q) \vee f, \quad C_4 : \neg f.$$

The variable f is to represent an unsatisfiable formula. It could also represent a whole set of formulas which imply $\neg f$, so the fact that the second disjunct in C_2 and C_3 is unsatisfiable may not be obvious. We assign branch indices to the disjuncts as follows:

C_1	Branch 1: p	Branch 2: q
C_2	Branch 3: $c_1\ \neg p$	Branch 4: f
C_3	Branch 5: $c_2\ \neg q$	Branch 6: f

The algorithm proceeds in the following way:

1. (C_1, Branch 1 on $*$:) Add $*\,p$ to M.

2. (C_2, Branch 3 on ε:) Add $c_1\ \neg p$ to M.

   Now M contains a clash with witness c_1. Notice that the main label of the second formula is ε, not c_1.

3. (Undo Branch 3 on ε:) Remove $c_1\ \neg p$ from M.

   Add a nogood c_1: $1, \varepsilon$: $3 \rightarrow \varepsilon\ \bot$, expressing that Branches 1 and 3 cannot be chosen simultaneously on the labels c_1 and ε, respectively, in any model.

4. (C_2, Branch 4 on ε:) Add $\varepsilon\ f$ to M.

   This results in a clash with C_4 on the witness ε.

5. (Undo Branch 4 on ε:) Remove $\varepsilon\ f$ from M.

   We introduce a nogood ε: $4 \rightarrow \varepsilon\ \bot$. Now C_2 fails on all disjuncts, and we *propagate* the two nogoods witnessing the failure into a common nogood, not including the branches 3 and 4 on C_2 itself. So the propagated nogood becomes c_1: $1 \rightarrow \varepsilon\ \bot$.

6. (Undo Branch 1 on c_1:) Add c_1 as an exception to $*\,p$.

   Nogoods may not remain violated, so we must undo Branch 1. If the nogood mentioned more than one branch, we would undo the branch with maximal index, and if it mentioned no branches at all, we would have shown S unsatisfiable.

7. (C_1, Branch 2 on c_1:) Add $c_1\ q$ to M.

8. (Redo Branch 3 on ε:) Add $c_1\ \neg p$ to M.

   Note that this does not violate the nogood c_1: $1, \varepsilon$: $3 \rightarrow \varepsilon\ \bot$ (which continues to exist!), because Branch 1 no longer includes the instance c_1.

9. (C_3, Branch 5 on ε:) Add $c_2\ \neg q$ to M.

The algorithm returns the set $M = \{(* - \{c_1\})\, p, c_1\, \neg p, c_1\, q, c_2\, \neg q, \neg f\}$ as a model for S. In Step 3, it was crucial to restrict the nogood premise to c_1: 1. If we had carelessly written $*$: 1, then in Step 7 we would have to extend Branch 2 to $*$, which would lead to a clash after branching on C_3, similar to Steps 3–5, and we would erroneously determine S unsatisfiable. This is just one of the caveats we must pay attention to when designing the algorithm.

6.2 More Normal Forms

Early on in Definition 2.1.14, we presented the *And/Or normal form* for propositional formulas, in which no conjunctions are nested inside other conjunctions and no disjunctions inside disjunctions, and all conjunctions or disjunctions are of arity at least 2.
While this is achievable in purely propositional concatenations of subformulas, we usually cannot flatten conjunctions into conjunctions or disjunctions into disjunctions, when the inner conjunctions or disjunctions are preceded by modal operators: Only $\Diamond$ distributes over disjunctions, and $\Box$ over conjunctions. Thus the **K**-formula $\Diamond\,(p \wedge q) \wedge \Diamond\, r$ is *not* equivalent to $\Diamond\, p \wedge \Diamond\, q \wedge \Diamond\, r$, but it is permissible to flatten $\Box\,(p \wedge q) \wedge \Box\, r$ to $\Box\, p \wedge \Box\, q \wedge \Box\, r$.
For $\mathcal{LBL}$, perhaps surprisingly, the rules are different:

Proposition 6.2.1 *For any simple label σ and any conjunction $\alpha_1 \wedge \ldots \wedge \alpha_n$, we have $\sigma\,(\alpha_1 \wedge \ldots \wedge \alpha_n) \equiv \sigma\,\alpha_1 \wedge \ldots \wedge \sigma\,\alpha_n$, where $\equiv$ is $\equiv_l$ or $\equiv_s$.*

Proof: Let M be a set of assertions. According to Corollary 5.5.8, $M \models_s \sigma\,(\alpha_1 \wedge \ldots \wedge \alpha_n)$ iff σ is scwr and $M_{\langle\gamma\rangle} \models_s \alpha_1 \wedge \ldots \wedge \alpha_n$ for all ground instances γ of σ. Now we distribute $\models_s$ over the conjunction to obtain $M_{\langle\gamma\rangle} \models_s \alpha_i$ for all $i = 1, \ldots, n$. This holds for all ground instances of σ, so we can apply Corollary 5.5.8 backwards to get $M \models_s \sigma\,\alpha_i$ for all $i = 1, \ldots, n$, which is equivalent to $M \models_s \sigma\,\alpha_1 \wedge \ldots \wedge \sigma\,\alpha_n$.
The proof for the $\models_l$ case is similar, using Theorem 5.5.2 instead of Corollary 5.5.8, and reasoning over all rgis of σ in M instead of all ground instances. □
One easily verifies that this result is invalid for disjunctions. For instance, $*\,(p \vee q)$ ("One of p or q holds in every instance of $*$") is not equivalent to $*p \vee *q$ ("The *same* p or q holds in *all* instances of $*$"). A (weak) countermodel is $\{c_1\, p, c_2\, q\}$: it satisfies the former formula but not the latter.
Proposition 6.2.1 allows us to define what we call the *expanded And/Or normal form* (ENF for short), because labels in conjunctions get expanded into all the conjuncts:

Definition 6.2.2 *The set of $\mathcal{LBL}$-formulas in* expanded And/Or normal form (ENF) *is the smallest set of formulas in the following recursive form:*

$\sigma_0\,(\beta_1 \vee \ldots \vee \beta_m)$, *where each* β_j, $j = 1, \ldots, m$, *is a conjunction of formulas of the form* $\sigma_{j,i}\alpha_{j,i}$, $i = 1, \ldots, n_j$, *in which each* $\alpha_{j,i}$ *is*

- *a propositional atom or*
- *a disjunction of at least two terms, and* $\sigma_{j,i}\alpha_{j,i}$ *is in ENF.*

We convert a formula F into a formula enf(F) as follows:

Algorithm 6.2.3 *Conversion to expanded And/Or normal form*

Parameters:

F: an $\mathcal{LBL}$-formula in And/Or NF

Returns:

a formula enf(F) in ENF

enf($\sigma\,(\beta_1 \vee \ldots \vee \beta_m)$) = $\sigma\,(\text{enf}(\beta_1) \vee \ldots \vee \text{enf}(\beta_m))$,

enf($\sigma\,(\alpha_1 \wedge \ldots \wedge \alpha_m)$) = $\bigwedge\{\sigma\,\alpha_i' : \alpha_i' \text{ is a conjunct in enf}(\alpha_i), i = 1, \ldots, n\}$,

enf($\sigma\,a$) = $\sigma\,a$, for any propositional atom a.

Beyond propagating the conversion into inner formulas, the rule for conjunctions performs two additional steps: moving the label σ inside the conjunctions, and flattening conjunctions to the greatest extent possible. Take $F = \sigma_1\,(\sigma_2\,(p \wedge q) \wedge r)$, for instance. According to our definition, enf($\sigma_2\,(p \wedge q)$) = $\sigma_2\,p \wedge \sigma_2\,q$. The flattening step ensures that enf(F) = $\sigma_1\sigma_2\,p \wedge \sigma_1\sigma_2\,q \wedge \sigma_1\,r$, not $\sigma_1(\sigma_2\,p \wedge \sigma_2\,q) \wedge \sigma_1\,r$ which would not be in ENF.
Let us verify that the result of this algorithm is indeed in ENF:

Proposition 6.2.4 *For any $\mathcal{LBL}$-formula F, the result* enf(F) *of applying Algorithm 6.2.3 is a conjunction of one or more formulas in ENF, and* $F \equiv \text{enf}(F)$, *for* $\equiv$ *in* $\{\equiv_l, \equiv_s\}$.

Proof: Both statements are easily shown using structural induction on F. The only interesting part is to show that equivalence is preserved under conjunction of formulas: in this case, we assume by the induction hypothesis that $\alpha_i \equiv \text{enf}(\alpha_i)$ for all $i = 1, \ldots, n$. Hence, $\alpha_1 \wedge \ldots \wedge \alpha_m \equiv \text{enf}(\alpha_1) \wedge \ldots \wedge \text{enf}(\alpha_m)$. The equivalence is preserved under flattening of this conjunction (Corollary 2.1.4), so we get $\sigma\,\alpha_1 \wedge \ldots \wedge \alpha_m \equiv \sigma\,\bigwedge\{\alpha_i'\,\alpha_i' \text{ is a conjunct in}$

$\text{enf}(\alpha_i), i = 1, \ldots, n\}$, and finally Proposition 6.2.1 shows that σ can be distributed over all conjuncts, resulting in the equivalent term $\bigwedge\{\sigma\,\alpha_i' : \alpha_i'$ is a conjunct in $\text{enf}(\alpha_i), i = 1, \ldots, n\}$, which is $\text{enf}(\sigma\,(\alpha_1 \wedge \ldots \wedge \alpha_m))$. Further details of this proof are left to the reader. □

Corollary 6.2.5 *For any set S of $\mathcal{LBL}$-formulas, a set S' of $\mathcal{LBL}$-formulas in ENF can be found so that $S \equiv S'$, for $\equiv$ in $\{\equiv_l, \equiv_s\}$.*

Proof: By Proposition 2.1.15, S can first be converted into an equivalent set $\overline{S}$ of atoms (propositional atoms or formulas of the form $\sigma\,F$, $\sigma \neq \varepsilon$) and at least binary disjunctions in And/Or NF. Write the latter as $\varepsilon\,F$. Now let S' be the set of conjuncts in $\text{enf}(\sigma\,F)$, for all $\sigma\,F \in S$. Proposition 6.2.4 shows that S' is a set of formulas in ENF, and that equivalence is preserved in computing $\text{enf}(\sigma\,F)$. And equivalence is also preserved in flattening the conjunctions $\text{enf}(\sigma\,F)$ into S'. □

By converting S from And/Or NF into ENF, we have eliminated all labels preceding conjunctions, and possibly flattened conjunctions in S further. Notice that $d(S)$ does not change under this procedure. The size of S may increase due to the duplication of labels, but only marginally: any given subformula may be augmented by some or all of the labels preceding the cascade of formulas in which it is a subformula; the concatenation of these labels is of length no greater than $d(S)$. Hence S may increase by a factor $O(d(S))$ in the worst case.

The next conversion step does not preserve equivalence, but it does preserve satisfiability. The idea for this conversion is taken from a normal form for **S5**-formulas [14, 56][1]. It resembles the well-known polynomial-size conversion algorithm for propositional formulas into CNF which also preserves satisfiability but not equivalence. What we call *labelled CNF* is much like the ENF in Definition 6.2.2, except without recursion:

Definition 6.2.6 *A set of $\mathcal{LBL}$-formulas is in* labelled clause (or conjunctive) normal form *(LCNF), if its formulas, called* clauses, *are of the form $\sigma_0\,(\beta_1 \vee \ldots \vee \beta_m)$, and each β_j, called a* term, *is a conjunction of labelled atoms of the form $\sigma_{j,i}\,a_{j,i}$, $i = 1, \ldots, n_j$, $j = 1, \ldots, m$.*

Note that unlike in propositional CNF, $a_{j,i}$ may not always be a literal: we cannot avoid the use of $\top$ and $\bot$ in an LCNF, as these constants are used to make statements on the

[1]The term "modal conjunctive normal form" (MCNF) used in [56] is a misnomer, for only the positions of the modal operators are normalized; no steps are taken to flatten the formula propositionally. The axioms of **S5** are needed for this equivalence-preserving transformation which does not generalize to other normal modal logics. To the best of our knowledge, a more general, *satisfiability-preserving* transformation has never been proposed before.

existence and nonexistence of labels. We could get rid of the inner conjunctions, to obtain a form even closer to propositional CNF. However, we cannot eliminate the inner label $\sigma_{j,i}$ or distribute the outer σ_0, since labels do not distribute over disjunctions. And eliminating the inner conjunctions would introduce extra variables while not providing significant gains, so we content ourselves with the LCNF as defined[2].

To convert a formula from ENF into LCNF, all we need is to introduce new propositional variables and replace the inner conjuncts $\sigma_{j,i}\ \alpha_{j,i}$ suitably. Here is how:

Algorithm 6.2.7 *Conversion to labelled clause normal form*

Parameters:

S: a set of formulas in ENF

Returns:

lcnf(S): a set of formulas in LCNF

while there exists a formula $\sigma_0\ F \in S$ which is not in LCNF **do**

Choose a conjunct $\sigma_{j,i}\alpha_{j,i}$ from the terms β_j in F so that $\alpha_{j,i}$ is not an atom.

Choose some $p_{j,i} \in P$ which is not used in S.

In F, replace $\alpha_{j,i}$ by $p_{j,i}$.

set $S := S \cup \{\sigma_0[\sigma_{j,i}]\ (\neg p_{j,i} \vee \alpha_{j,i})\}$

done

return S

We remarked in Definition 6.2.6 that any non-atom $\alpha_{j,i}$ in an ENF-formula must be a disjunction of at least two terms. It is understood that these terms are flattened into the above disjunction $\neg p_{j,i} \vee \alpha_{j,i}$. We easily see that the number of nested non-atomic conjuncts $\sigma_{j,i}\alpha_{j,i}$ strictly decreases upon applying this procedure, so the algorithm terminates, and according to the termination criterion in the **while** loop, the returned set is in LCNF. We only need to show that satisfiability is preserved in each iteration of the **while** loop. The two directions of the proof are markedly different, so we state them separately. We will use the same identifiers as in the algorithm above, and denote by S' the set obtained from S in one iteration of the **while** loop. In the first direction, given a set of assertions M for S', we

[2]If we had used labels with proper variables instead of the anonymous $*$, then labels would distribute over disjunctions as well as conjunctions, and we could obtain a "proper" CNF. In fact, the optimized functional translation [81] accomplishes exactly that. However, if we did this, the entire problem would collapse into a notational variant of a first-order problem; this was not our goal, as we already stated in the introduction.

eliminate all occurrences of $p_{j,i}$ in a way that the resulting set satisfies S. We will establish a few general results first:

Proposition 6.2.8 *Given a set of assertions M and propositional variable p, we denote by $M|_p$ the set obtained from M by replacing all occurrences $\lambda_1\, p$ and $\lambda_2\, \neg p$ with $\lambda_1\, \top$ and $\lambda_2\, \top$, respectively, and adding $(\lambda_1, \lambda_2)\, \bot$ for any pair of such occurrences in which λ_1 and λ_2 unify. Then for any simple label σ we have:*

(1) $(M_{\langle\sigma\rangle})|_p \subseteq (M|_p)_{\langle\sigma\rangle} \subseteq (M_{\langle\sigma\rangle})|_p \cup \{\varepsilon\, \bot\}$.

(2) *If a label σ' is strongly constant-wise realizable in $M_{\langle\sigma\rangle}$, then it is strongly constant-wise realizable in $(M|_p)_{\langle\sigma\rangle}$.*

(3) *If F is an $\mathcal{LBL}$-formula not mentioning p, then $M_{\langle\sigma\rangle} \models_s F$ implies $(M|_p)_{\langle\sigma\rangle} \models_s F$.*

(4) *If $M|_p$ contains a clash, then so does M.*

Proof: It suffices to show part (1) for a one-element label $\sigma = x$; the general proof follows iteratively. Assume first that M does not contain a clash with witness ε. Then neither M nor $M|_p$ are utopian. It is clear that each original assertion $\lambda\, a$ in M contributes an assertion $\lambda'\, a$ into $M_{\langle x\rangle}$ exactly when the corresponding assertion $\lambda\, a'$ in $M|_p$ (where $a' = \top$ in case $a = p$ or $a = \neg p$, and $a' = a$ otherwise) contributes $\lambda'\, a'$ into $(M|_p)_{\langle x\rangle}$; the same assertion $\lambda'\, a'$ is found in $(M_{\langle x\rangle})|_p$.
Secondly, any of the additional assertions $(\lambda_1, \lambda_2)\, \bot$ in $M|_p$ contributes an assertion $\lambda'\, \bot$ into $(M|_p)_{\langle x\rangle}$ iff $(\lambda_1, \lambda_2)\ {}_p\sqsubseteq x$, which holds iff both $\lambda_1\ {}_p\sqsubseteq x$ and $\lambda_2\ {}_p\sqsubseteq x$. Exactly when this is the case, these two assertions contribute two assertions $\lambda_1'\, p$ and $\lambda_2'\, \neg p$ into $M_{\langle x\rangle}$, and one easily verifies that (λ_1', λ_2') exists and is equal to λ'. Therefore, exactly then the assertion $\lambda'\, \bot$ is also found in $(M_{\langle x\rangle})|_p$.
Next, if M is utopian, then so is $M|_p$, and subscripting introduces an extra $\varepsilon\, \bot$ into both $(M|_p)_{\langle x\rangle}$ and $M_{\langle x\rangle}$, which remains in $(M_{\langle x\rangle})|_p$. Finally, if M contains a clash of the form $\varepsilon\, p$, $\varepsilon\, \neg p$, then only $M|_p$ is utopian, and $(M|_p)_{\langle x\rangle}$ may contain the extra $\varepsilon\, \bot$ which is not in $(M_{\langle x\rangle})|_p$, the only case where these two sets are not identical.
Part (2) is easy because the labels of $M_{\langle\sigma\rangle}$ are all present in $(M|_p)_{\langle\sigma\rangle}$. This is sufficient for applying the monotonicity principle to the property of being scwr, even though the subset relation does not strictly hold.

For part (3), it suffices to consider the special case $\sigma = \varepsilon$. (In the general case, define $M' = M_{\langle\sigma\rangle}$. Then $M'|_p = (M_{\langle\sigma\rangle})|_p$ is a subset of $(M|_p)_{\langle\sigma\rangle}$ according to (1). If we have shown that $M' \models_s F$ implies $M'|_p \models_s F$, then $(M|_p)_{\langle\sigma\rangle} \models_s F$ follows by monotonicity.) The case where $M \models_s F$ because M is utopian is trivial. (As we said, $M|_p$ is also utopian.) So we will assume $\bot \notin M$. Now suppose for the base case that $M \models_s a$, where a is an atom; a cannot be $\bot$, and since $M|_p \models_s \top$ is always true, the case $a = \top$ is trivial. If a is a literal, then $\varepsilon\, a$ must be in M. This assertion is found unchanged in $M|_p$, unless $a = p$ or $a = \neg p$; but a was explicitly assumed not to mention p. Therefore, $M|_p \models_s a$. Next, assuming part (3) shown for F, suppose that $M \models_s \sigma\, F$. Then σ is scwr in M, and by (2) it is also scwr in $M|_p$. Furthermore, $M_{\langle\sigma'\rangle} \models_s F$ for all instances σ' of σ, and the induction hypothesis, applied to $M_{\langle\sigma'\rangle}$ and F, implies that $(M|_p)_{\langle\sigma'\rangle} \models_s F$, showing that $M|_p \models_s \sigma\, F$. We leave it as an exercise to the reader to show this property preserved under propositional concatenation; from here the proof follows by structural induction.

As for part (4), the only type of clash in $M|_p$ not involving labels already in M is when one of the new assertions $(\lambda, \lambda')\ \bot$ has a realizable witness (λ, λ'). However, $\lambda\, p_{j,i}$ and $\lambda'\, \neg p_{j,i}$ were assertions in M, and since (λ, λ') exists and is realizable, these two assertions form a clash in M. □

Proposition 6.2.9 *Let M be a model for S'. Then $M|_{p_{j,i}}$ is a model for S.*

Proof: We prove this "bottom-up"—from the term level to the entire set S. We point out that $p_{j,i}$ is a new variable which does not occur in S, so for any subformula of S satisfied by some $M_{\langle\sigma\rangle}$, Proposition 6.2.8 shows that it is also satisfied by $(M|_{p_{j,i}})_{\langle\sigma\rangle}$.

1. For any instance $\sigma\sigma'$ of $\sigma_0\sigma_{j,i}$ so that $M_{\langle\sigma\sigma'\rangle} \models_s p_{j,i}$, we have $(M|_{p_{j,i}})_{\langle\sigma\sigma'\rangle} \models_s \alpha_{j,i}$.

   Since $\sigma_0[\sigma_{j,i}]\,(\neg p_{j,i} \vee \alpha_{j,i})$ is a formula in S', $M_{\langle\sigma\sigma'\rangle}$ must satisfy the disjunction $\neg p_{j,i} \vee \alpha_{j,i}$ for any instance $\sigma\sigma'$ of $\sigma_0\sigma_{j,i}$. If $M_{\langle\sigma\sigma'\rangle}$ satisfies (some disjunct in) $\alpha_{j,i}$, then so does $(M|_{p_{j,i}})_{\langle\sigma\sigma'\rangle}$ (since $\alpha_{j,i}$ is a subformula of S), and we are done. Otherwise $M_{\langle\sigma\sigma'\rangle}$ must satisfy $\neg p_{j,i}$, but according to our assumption, $M_{\langle\sigma\sigma'\rangle}$ also satisfies $p_{j,i}$. This means, two assertions $\lambda\, p_{j,i}$ and $\lambda'\, \neg p_{j,i}$ must exist in M, and $\sigma\sigma'$ instantiates both λ and λ'. Hence these two labels unify, and $(\lambda, \lambda') \sqsubseteq \sigma\sigma'$. But in this case, $(\lambda, \lambda')\ \bot$ has been included in $M|_{p_{j,i}}$. It follows that $(M|_{p_{j,i}})_{\langle\sigma\sigma'\rangle}$ is utopian and satisfies anything, including $\alpha_{j,i}$.

2. For any instance σ of σ_0, if $M_{\langle\sigma\rangle} \models_s \sigma_{j,i}\, p_{j,i}$, then $(M|_{p_{j,i}})_{\langle\sigma\rangle} \models_s \sigma_{j,i}\, \alpha_{j,i}$.

A label $\sigma\sigma'$ instantiates $\sigma_0\sigma_{j,i}$ iff $\sigma_0 \sqsubseteq \sigma$ and $\sigma_{j,i} \sqsubseteq \sigma'$. Hence this result follows from Statement 1 by quantifying over all instances σ' of $\sigma_{j,i}$, citing part (4) of Theorem 5.3.9, and noting that $\sigma_{j,i}$ being scwr in $M_{\langle\sigma\rangle}$ implies that $\sigma_{j,i}$ is also scwr in $(M|_{p_{j,i}})_{\langle\sigma\rangle})$, by part (2) or Proposition 6.2.8.

3. Let $F = \beta_1 \vee \ldots \vee \beta_j \vee \ldots \vee \beta_m$ be the formula changed into $F' = \beta_1 \vee \ldots \vee \beta'_j \vee \ldots \vee \beta_m$ by replacing the conjunct $\sigma_{j,i}\, \alpha_{j,i}$ in β_j with $\sigma_{j,i}\, p_{j,i}$. Then for any instance σ of σ_0, if $M_{\langle\sigma\rangle} \models_s F'$, then $(M|_{p_{j,i}})_{\langle\sigma\rangle} \models_s F$.

   If $M_{\langle\sigma\rangle}$ satisfies any disjunct $\beta_{j'}$ other than β'_j, this disjunct is a subformula in S, and Proposition 6.2.8 shows $(M|_{p_{j,i}})_{\langle\sigma\rangle} \models_s \beta_{j'}$. Otherwise, $M_{\langle\sigma\rangle}$ must satisfy the remaining disjunct β'_j, along with all its conjuncts $\sigma_{j,i'}\, \alpha_{j,i'}$ in β'_j, $i' = 1, \ldots, n_j$. For $i' \neq i$, the $\sigma_{j,i'}\, \alpha_{j,i'}$ are subformulas in S, so $(M|_{p_{j,i}})_{\langle\sigma\rangle} \models_s \sigma_{j,i'}\, \alpha_{j,i'}$. When $i' = i$, we have $M_{\langle\sigma\rangle} \models_s \sigma_{j,i}\, p_{j,i}$, and by Statement 2, $(M|_{p_{j,i}})_{\langle\sigma\rangle} \models_s \sigma_{j,i}\, \alpha_{j,i}$. This proves that $(M|_{p_{j,i}})_{\langle\sigma\rangle} \models_s \beta_j$, which in turn proves our statement.

4. $M \models_s S'$ implies $M|_{p_{j,i}} \models_s S$.

   For any formula in S other than $\sigma_0\, F$, this is evident. And for the formula F' modified by the algorithm, $M \models_s \sigma_0\, F'$ entails $M_{\langle\sigma\rangle} \models_s F'$ for any instance σ of σ_0; Statement 3 shows that $(M|_{p_{j,i}})_{\langle\sigma\rangle} \models_s F$. Since this is true for every instance σ, and since σ_0 being scwr is inherited into $M|_{p_{j,i}}$ (as before), we derive $M|_{p_{j,i}} \models_s \sigma_0\, F$ as claimed.

5. If M is clash-free, then so is $M|_{p_{j,i}}$.

   This is just the contrapositive of Proposition 6.2.8, part (4).

□

Let us now show that satisfiability is also preserved in the converse direction. Assume that S has a set of assertions M satisfying it. Wlog, we can assume that M does not contain $p_{j,i}$. (If it does, it can be eliminated by replacing M with the set M' as in Proposition 6.2.9.) We seek to obtain a set $\widehat{M}$ which satisfies S'. All we need is to find suitable assertions laying out the truth assignments for the extra variable $p_{j,i}$:

Proposition 6.2.10 *Given a model M for S, classify the exception-generated instances σ of σ_0 into two sets Σ^+, Σ^-, as follows:*

If $M_{\langle\sigma\rangle} \models_s \beta_j$, then $\sigma \in \Sigma^+$, otherwise $\sigma \in \Sigma^-$.

Now construct $\widehat{M}$ *by adding the following assertions to* M *:*

- $(\sigma\sigma_{j,i} - \{\xi \in \Sigma^- : \sigma \sqsubseteq \xi\})\, p_{j,i}$ *for every* $\sigma \in \Sigma^+$;
- $(\sigma[\sigma_{j,i}] - \{\xi \in \Sigma^+ : \sigma \sqsubseteq \xi\})\, \neg p_{j,i}$ *for every* $\sigma \in \Sigma^-$.

Then $\widehat{M}$ *is a model for* S'.

The asymmetry, turning the constants in $\sigma_{j,i}$ into conditional constants for "negative" assertions only, is intentional: If $\sigma \in \Sigma^+$, then $M_{\langle\sigma\rangle}$ satisfies $\sigma_{j,i}\, \alpha_{j,i}$ as part of β_i, which entails that $\sigma_{j,i}$ must be existential; for $\sigma \in \Sigma^-$, it need not.

Proof: We make the following observations:

- If M is clash-free, then so is $\widehat{M}$.

  This follows from the way we carefully separate the labels so that any pair involving a "positive" and a "negative" assertion have disjoint scopes: The atoms introduced into $\widehat{M}$ only mention $p_{j,i}$, a variable proposed to be absent from M. Therefore, the only possible sources of "new" clashes in $\widehat{M}$ involve two assertions $(\sigma^+\sigma_{j,i} - \Sigma_1^-)\, p_{j,i}$ and $(\sigma^-[\sigma_{j,i}] - \Sigma_1^+)\, \neg p_{j,i}$. If their two labels are to unify, the main labels must have an mgu, which exists iff (σ^+, σ^-) exists; this mgu, the main label of the clash witness, is of the form $(\sigma^+, \sigma^-)\sigma_{j,i}$. But σ^+ and σ^- are both egis of σ_0, and from Corollary 5.3.6, part (2), we know that (σ^+, σ^-) itself is an egi of σ_0, so it must also have been classified into either Σ^+ or Σ^-. If $(\sigma^+, \sigma^-) \in \Sigma^+$, then it is also an exception in Σ_1^+ (since $\sigma^- \sqsubseteq (\sigma^+, \sigma^-)$), so $(\sigma^+, \sigma^-)\sigma_{j,i}$ is not actually an instance of $\sigma^-[\sigma_{j,i}] - \Sigma_1^+$, and the two (complex) labels above do not unify. Likewise, if $(\sigma^+, \sigma^-) \in \Sigma^-$, it must be an exception in Σ_1^-, and again the two complex labels do not unify. We conclude that no two new labels in $\widehat{M}$ form a clash, which completes the proof.

- For any exception-generated instance σ of σ_0, if $M_{\langle\sigma\rangle} \models_s \beta_j$, then $\widehat{M}_{\langle\sigma\rangle} \models_s \beta'_j$.

  From $M_{\langle\sigma\rangle} \models_s \beta_j$ we know that $M_{\langle\sigma\rangle} \models_s \sigma_{j,i'}\, \alpha_{j,i'}$ for all $i' \neq i$. By monotonicity, the larger set $M'_{\langle\sigma\rangle}$ also satisfies $\sigma_{j,i'}\, \alpha_{j,i'}$. Now since $M_{\langle\sigma\rangle} \models_s \beta_j$, we classified σ into Σ^+, so $\widehat{M}$ contains a label of the form $(\sigma\sigma_{j,i} - \Sigma)\, p_{j,i}$, and Σ contains only proper instances of σ. Therefore, this label contributes the assertion $\sigma_{j,i}\, p_{j,i}$ into $\widehat{M}_{\langle\sigma\rangle}$, and using Proposition 5.5.12, we conclude that $\widehat{M}_{\langle\sigma\rangle} \models_s \sigma_{j,i}\, p_{j,i}$. Therefore, $\widehat{M}_{\langle\sigma\rangle}$ satisfies β'_j since we have shown that it satisfies all its conjuncts.

- If $M \models_s \sigma_0\, F$, then $\widehat{M} \models_s \sigma_0\, F'$ (where F' is F with β_j replaced by β'_j, as before).

  Let σ be any egi of σ_0. If $M_{\langle\sigma\rangle} \models_s \beta_j$, then $\widehat{M}_{\langle\sigma\rangle} \models_s \beta'_j$ as we just showed, and hence $\widehat{M}_{\langle\sigma\rangle} \models_s F'$. Otherwise, $M_{\langle\sigma\rangle}$ must satisfy some other disjunct $\beta_{j'}$, $j' \neq j$; but this disjunct also occurs in F', and $\widehat{M}_{\langle\sigma\rangle}$ is a superset of $M_{\langle\sigma\rangle}$, so $\widehat{M}_{\langle\sigma\rangle} \models_s \beta_{j'}$, and once again $\widehat{M}_{\langle\sigma\rangle} \models_s F'$. We have shown this for every egi of σ_0, so part (3) of Theorem 5.3.9 implies $\widehat{M} \models_s \sigma_0\, F'$.

- For every exception-generated instance $\sigma\sigma'$ of $\sigma_0\sigma_{j,i}$, we have $\widehat{M}_{\langle\sigma\sigma'\rangle} \models_s (\neg p_{j,i} \vee \alpha_{j,i})$.

  We make use of Proposition 5.3.7 which says: If $\sigma\sigma'$ is an egi of $\sigma_0\sigma_{j,i}$ in M, then σ is an egi of σ_0 in M, and σ' is an egi of $\sigma_{j,i}$ in $M_{\langle\sigma\rangle}$. We distinguish two cases: If σ is classified into Σ^-, then a new assertion $(\sigma[\sigma_{j,i}] - \Sigma_1^+)\, \neg p_{j,i}$ has been introduced into $\widehat{M}$; upon subscripting, it contributes $[\sigma_{j,i}]\, \neg p_{j,i}$ to $\widehat{M}_{\langle\sigma\rangle}$. Upon subscripting even further, we see that $\neg p_{j,i} \in \widehat{M}_{\langle\sigma\sigma'\rangle}$ (as σ' instantiates $\sigma_{j,i}$); this shows that $\widehat{M}_{\langle\sigma\sigma'\rangle} \models_s \neg p_{j,i}$. If instead σ has been classified into Σ^+, this is because $M_{\langle\sigma\rangle} \models_s \beta_j$. But then $M_{\langle\sigma\rangle}$ satisfies all conjuncts in β_j, in particular $\sigma_{j,i}\alpha_{j,i}$. By our familiar part (3) of Theorem 5.3.9, $M_{\langle\sigma\sigma'\rangle} \models_s \alpha_{j,i}$ because σ' is an egi of $\sigma_{j,i}$ in $M_{\langle\sigma\rangle}$. Finally, $\widehat{M}_{\langle\sigma\sigma'\rangle} \models_s \alpha_{j,i}$ also holds because of monotonicity. In either case we conclude that $\widehat{M}_{\langle\sigma\sigma'\rangle}$ satisfies the disjunction $\neg p_{j,i} \vee \alpha_{j,i}$ as claimed.

- $\widehat{M} \models_s \sigma_0[\sigma_{j,i}]\, (\neg p_{j,i} \vee \alpha_{j,i})$.

  The label σ_0 is scwr in M, and since being scwr is a monotone property, σ_0 is scwr in $\widehat{M}$ as well. The label $\sigma_0[\sigma_{j,i}]$ has no existential constants beyond those in σ_0, so it is also scwr in $\widehat{M}$. And we just showed that $\widehat{M}_{\langle\sigma\sigma'\rangle} \models_s (\neg p_{j,i} \vee \alpha_{j,i})$ for every egi $\sigma\sigma'$ of this label, which altogether proves our claim.

To summarize, we showed that $\widehat{M}$ satisfies the modified formula $\sigma_0\, F'$ as well as the new formula in S'. And being a superset of M, it also satisfies any unchanged formula from S. So we showed $\widehat{M} \models_s S'$. Furthermore, since M is clash-free, so is $\widehat{M}$, which finishes the proof that $\widehat{M}$ is a model for S'. □

A few remarks on this conversion are in order:

- In analogy to Theorem 5.3.14, it would suffice to classify into Σ^+, Σ^- only the exception-generated instances of a suitable submodel M' of M which contribute to the satisfiability of $\sigma_0\, F$. This reduces the number of exception-generated instances for which new labels are introduced into $\widehat{M}$.

- Furthermore, it is not necessary to introduce a new assertion for a label σ in Σ^+ if it has an immediate predecessor $\hat{\sigma}$ in the hierarchy of all exception-generated instances of σ_0 which is also classified in Σ^+; likewise for Σ^-. Similarly, any exception ξ in Σ_1^- (Σ_1^+) can be omitted, if it instantiates another exception $\xi' \in \Sigma_1^-$ (Σ_1^+). This too will reduce the number and size of new assertions in $\widehat{M}$.

- Even despite these efficiency measures, we cannot prevent the size of $\widehat{M}$ from growing exponentially in the worst case, especially considering that Proposition 6.2.10 covers only one step of the algorithm. In Example 5.1.2, we encountered a variable e representing a parity check on some of the other propositional variables; we saw that representing the truth assignments to e takes an exponential number of assertions; specifying all other variable assignments takes only $O(d)$ assertions. Now imagine a formula $*^d(F_o \vee F_e)$, where F_o and F_e are *true* iff e is *false* and *true*, respectively, but do not mention e. This formula is *true* in all instances of $*^d$, so the model need only specify the $O(d)$ assertions for the original variables. But upon converting to LCNF, we must represent e (and possibly other auxiliary variables) in the model as well, which results in an exponential increase in the model size.

  So why would we consider conversion to LCNF, given this potentially devastating increase in model size? First, we derive support from the analogous situation in propositional satisfiability: Arguably, the overwhelming majority of SAT algorithms are specified only for problems in CNF, because it is easier to do so—despite the well-known fact that introducing new propositional variables may lead to an exponential increase in the time complexity of the algorithm, at least if those variables are poorly handled. Secondly, if it really takes an exponential number of assertions to express the truth assignment for a subformula $\alpha_{j,i}$, then a model-finding algorithm would spend an exponential amount of time satisfying this subformula over all instances of $*^k$ anyway, whether we explicitly represent them or not. So the exponential increase in model size would only occur in problems which are difficult to start with. Finally, the main purpose of assuming that all formulas are converted to LCNF is that we can present our model finding algorithm with much greater clarity. Whether to represent the $p_{j,i}$ explicitly or through pointers to the formula $\alpha_{j,i}$ is an issue we leave open as an implementation decision.

We summarize our results on the LCNF by iterating them over the course of the algorithm:

Theorem 6.2.11 *A set S of formulas is satisfiable iff* $\mathrm{lcnf}(S)$ *is satisfiable.*

6.3 Original Assertions

Thanks to the results of the previous section, we are now in a position to convert any $\mathcal{LBL}$-formula (and hence any $\mathbf{K_{NNF}}$-formula) or set thereof, into a set S of formulas in LCNF. Let us reflect on the anatomy of these formulas and how we can proceed in finding a (strong) model for them.

Each formula C (or clause, as we called it) is composed of a simple label σ_0 (which we will call the *outer label* of C), and a disjunction of one or more terms β_j, $j = 1, \ldots, m$. Each term in turn is a conjunction of labelled atoms $\sigma_{j,i}\, a_{j,i}$, $i = 1, \ldots, n_j$. (The $\sigma_{j,i}$ are the *inner labels* of C.) In case there is only one disjunct ($m = 1$), the clause simply becomes a conjunction of labelled atoms; we can flatten it into a set of assertions, namely $\{\sigma_0\sigma_{1,i}\, a_{1,i} : i = 1, \ldots, n_1\}$, according to Proposition 6.2.1. (In fact, the algorithm for conversion into ENF has already taken care of this, and all clauses with one disjunct have been merged with the assertions.) We refer to the assertions existing in S after this preprocessing step as *original assertions*; they must be satisfied by every model for S; thus they constitute the deterministic part of S.

The easiest way of satisfying an original assertion $\sigma\, a$ is to include $\sigma\, a$ in the model M: Proposition 5.5.12 states that any model containing $\sigma\, a$ strongly satisfies $\sigma\, a$. However, we must show that this is a "safe" approach. In other words, there must not exist a model M without $\sigma\, a$, for which $\sigma\, a$ would introduce a clash. We set out by proving an auxiliary result containing two cases: a simple one, verifying the assertions in S as described, and a second, more involved one intended for use with the terms $\sigma_{j,i}\, a_{j,i}$ in the disjuncts. In essence, we would like to ensure that $M_{\langle\sigma'\rangle}$ satisfies a disjunct β_j in one of the clauses. This is easily accomplished by adding the assertions $(\sigma'\sigma_{j,i} - \Sigma)\, a_{j,i}$, $i = 1, \ldots, n_j$, to M. (We will later see what we need the exceptions in the set Σ for, but suffice it for now that they are all of length at most $|\sigma'|$.) Again we need a condition which guarantees that no clash is introduced and our model is preserved.

Lemma 6.3.1 *Let M be a model for a set of formulas S and $\lambda = \sigma - \Sigma$ a complex label so that σ is strongly constant-wise realizable in M and $M_{\langle\sigma\rangle} \models_s a$. Then $M \cup \{\lambda\, a\}$ is also a model for S.*

Now let $\lambda = \sigma_1\sigma_2 - \Sigma$ be a complex label so that all exceptions in Σ are of maximum length $|\sigma_1|$, σ_1 is scwr in M, and for all instances σ of $\sigma_1 - \Sigma$, $M_{\langle\sigma\rangle} \not\models_s \sigma_2\, a$. Then $M \cup \{\lambda\, a\}$ is also a model for S.

Proof: Notice that the first statement is a special case of the second, with $\sigma_2 = \varepsilon$. However, we will prove at least part of the first statement separately, and use it to prove the second. Since any extension of M satisfies S due to monotonicity, the only interesting fact to prove is that the extension is clash-free. First, we ensure that no additional rgls are introduced through the new assertion $\lambda\, a$, otherwise a non-realizable potential clash witness in M could become realizable. Secondly, we verify that the assertion $\lambda\, a$ itself cannot participate in a clash, unless M already contains a clash.

Consider the simple case first. Assume that $M \cup \{(\sigma - \Sigma)\, a\}$ does contain "new" rgls, and choose one, γc, of minimal length. Hence $\gamma \in I(M)$, and σ must have a prefix of the form $\sigma' c$ with $\sigma' \sqsubseteq \gamma$, and $\xi \not\sqsubseteq_p \gamma$ for any $\xi \in \Sigma$. (If the witness $\sigma' c$ existed in M instead, γc would be realized in M, contradicting our assumption.) But σ is scwr and hence cwr in M; accordingly, for any rgi γ of σ' we have $\gamma c \in I(M)$, which contradicts our assumption. We conclude that all rgls in $M \cup \{(\sigma - \Sigma)\, a\}$ are rgls in M.

In the second case, suppose $M \cup \{(\sigma_1\sigma_2 - \Sigma)\, a\}$ contains a "new" rgl of length at most $|\sigma_1|$. This leads to a contradiction exactly as in the first case, using the fact that σ_1 is scwr.

Now assume there exist "new" rgls of the form $\gamma_1\gamma_2 c$. As before, we pick one of minimal length, so $\gamma_1\gamma_2 \in I(M)$. We apply part (2) of Lemma 5.4.9 three times, which gives us $\gamma_1 \in I(M)$, $\gamma_2 \in I(M_{\langle\gamma_1\rangle})$, and $\gamma_2 c \in I((M \cup \{(\sigma_1\sigma_2 - \Sigma)\, a\})_{\langle\gamma_1\rangle})$, but $\gamma_2 c \notin I(M_{\langle\gamma_1\rangle})$. This is only possible when $\{(\sigma_1\sigma_2 - \Sigma)\, a\}_{\gamma_1}$ is nonempty, so γ_1 must instantiate $\sigma_1 - \Sigma$. With this information, we compute $I((M \cup \{(\sigma_1\sigma_2 - \Sigma)\, a\})_{\langle\gamma_1\rangle}) = M_{\langle\gamma_1\rangle} \cup \{\sigma_2\, a\}$. (Recall that no exception in Σ is longer than σ_1.) Furthermore, our prerequisites imply $M_{\langle\gamma_1\rangle} \not\models_s \sigma_2\, a$, which entails that σ_2 is scwr in $M_{\langle\gamma_1\rangle}$ and $M_{\langle\gamma_1\sigma_2\rangle} \not\models_s a$. Now we can apply the first case a second time, which shows that $M_{\langle\gamma_1\rangle}$ and $M_{\langle\gamma_1\rangle} \cup \{\sigma_2\, a\}$ have the same set of rgls, contradicting what we found above, namely $\gamma_2 c \in I(M_{\langle\gamma_1\rangle} \cup \{(\sigma_2\, a\})$ and $\gamma_2 c \notin I(M_{\langle\gamma_1\rangle})$.

Now assume the new assertion $\lambda\, a$ forms part of a clash. We consider the complex case directly, which includes the simple case. Since the clash witness is realizable, λ too must be realizable (in $M \cup \{\lambda\, a\}$; but from the first case, we see that this is the same as λ being realizable in M). Let $\gamma_1\gamma_2$ be the clash witness; in particular, γ_1 instantiates $\sigma_1 - \Sigma$. Therefore, $M_{\langle\gamma_1\rangle} \not\models_s \sigma_2\, a$, and since $\sigma_2 \sqsubseteq \gamma_2$, we conclude $M_{\langle\gamma_1\gamma_2\rangle} \not\models_s a$. In case $a = \bot$,

$M_{\langle\gamma_1\gamma_2\rangle}$ obviously contains a clash. In case $a = l$ is a literal, $M_{\langle\gamma_1\gamma_2\rangle}$ must contain l, and there must be another assertion $\lambda'\,\bar{l} \in M$ so that (λ, λ') exists. The clash witness $\gamma_1\gamma_2$ instantiates both λ and λ', so $\lambda'\,\bar{l}$ contributes the literal $\bar{l}$ into $M_{\langle\gamma_1\gamma_2\rangle}$. Together with the already existing l, this too forms a clash in $M_{\langle\gamma_1\gamma_2\rangle}$. Now $\gamma_1\gamma_2$ is realizable, so Proposition 5.5.4 entails that M contains a clash, which contradicts the fact that M is a model. Therefore, our assumption that $M \cup \lambda\, a$ contain a clash must be dropped, which completes the proof. □

Corollary 6.3.2 *Let S be a set of formulas including the assertion $\sigma\, a$. If the set M is a strong model for S, then so is $M \cup \{\sigma\, a\}$.*

Proof: The fact that M satisfies $\sigma\, a$ implies that σ is scwr in M, and $M_{\langle\sigma\rangle} \models_s a$. So we can apply Lemma 6.3.1 (with $\lambda = \sigma$), showing that along with M the set $M \cup \{\sigma\, a\}$ too is a model for S. □

Another, much less conspicuous type of original assertions is implicit within the main labels of the clauses: in any set M verifying a clause $\sigma_0\, F$, the label σ_0 must be strongly constant-wise realizable. The implicit original assertion which expresses this is $\sigma_o\, \top$:

Proposition 6.3.3 *Let S be a set of formulas including the clause $\sigma_0\, F$. If the set M is a strong model for S, then so is $M \cup \{\sigma_0\, \top\}$.*

Proof: Since $M \models_s \sigma_0\, F$, σ_0 is scwr in M, and $M_{\langle\sigma_0\rangle} \models_s \top$ is trivially true. So Lemma 6.3.1 shows that with M the set $M \cup \{\sigma\, \top\}$ is a model for S. □

We could normally do without these extra assertions. Later on, when we branch on the clause $\sigma_0\, F$, we will introduce some assertions of the form $\sigma_0\sigma_{j,i}\, a_{j,i}$, which will witness that σ_0 is strongly constant-wise realizable in M. However, there is an important conceptual distinction: while the assertions $\sigma_0\sigma_{j,i}\, a_{j,i}$ can be undone if the branch fails to produce a model, the fact that σ_0 is strongly constant-wise realizable cannot be undone—just like the original assertions, it is a property of the problem, not of one particular branch[3].

[3]We have not investigated this further, but including $\sigma_0\, \top$ may offer a significant practical advantage as well: upon branching, we can replace the existential constants in σ_0 by conditional constants, thus introducing assertions of the form $[\sigma_0]\sigma_{j,i}\, a_{j,i}$. This reduces the number of occurrences of existential constants in M, which in turn reduces the number of realizability checks for labels.

6.4 Branching

Having thus settled the unary clauses, the "proper" clauses constitute the much greater challenge in model finding. Just as in SAT, our suggested approach is *branching*: For each clause C, we successively try out each term in C in a suitable order, until a term gives rise to a satisfying model, or until there are no more terms left in C, in which case we must backtrack to an earlier branch in the search tree. This simplistic description captures the general principle. However, the fact that clauses are scoped over labels results in complications not encountered in SAT:

- We obviously do not wish to branch on every ground instance of σ_0 separately. (Otherwise our algorithm would not be much different from conventional calculi; besides, we would not obtain a *strong* model.) Instead, we will branch *universally* over σ_0, picking some term β_j and adding the assertions $\sigma_0\sigma_{j,i}\, a_{j,i}$, $i = 1, \ldots, n_j$, to our current set of assertions M. This conforms with our definition of strong satisfiability: $M \models_s \sigma_0\, F$ entails $M_{\langle\sigma_0\rangle} \models_s F$, or $M_{\langle\sigma_0\rangle} \models_s \beta_j$ for some $j = 1, \ldots, m$; this is evidently satisfied through the above assertions $\sigma_0\sigma_{j,i}\, a_{j,i}$. In fact, if we branch in this fashion on any one term per clause in S and collect all assertions into M, then in the end M strongly verifies S. Of course, M will likely not be clash-free; it is the occurrence of clashes during the construction of M which will cause us to repair branches.

- As demonstrated in the introductory section, we assign a unique positive integer called *branch index* to each term in each clause. Branch indices can be assigned at once at the beginning of the search, or dynamically to each term branched upon in the course of the algorithm, but once assigned, they must remain fixed. We introduce the notion λ: b to refer to a branch with index b over the label λ. Branch indices define an order on the terms in S; they will be used to avoid repetition and ensure completeness of the search, and they allow us to perform dependency-directed backtracking. More on this in the next section.

- If we were to branch only "globally" over the outer labels σ_0 of each clause, we would not obtain a sound algorithm. We already observed this in Example 6.1.2 at the beginning of this chapter: Upon picking either branch $*$: 1 or $*$: 2, we obtain assertions $*\, p$ and $*\, q$, respectively; neither can be extended to a satisfying model. While σ_0 is a good *initial* label to branch over, we will undo branches only on the instance(s) ξ

of σ_0 which lead(s) to inconsistencies. These instances become *exceptions* to the label σ_0 in the original branch σ_0: b_1. We can then try another branch on C over ξ, for instance ξ: b_2, and so on.

- For reasons which will become evident later, we insist that branch labels never become vacuous, that is: all exceptions must *strictly* instantiate the main label. Therefore, if a branch leads to an inconsistency over the entire label σ_0, the entire label is erased from the branch. But note that we never erase the branch itself or unassign its branch index.

- Some time after a branch has been undone, we may reinstate all or part of that branch. We have seen this in Example 6.1.2 in Step 7. In the general case, let ξ be an exception on some branch b, and we wish to reinstate b on an instance σ of ξ. Since we have no notion of "exceptions to exceptions", we will simply make σ a new label on b. Hence a branch can range over a *set* of complex labels[4].

- Branching often introduces new labels into the set M, namely the labels $\sigma_0\sigma_{j,i}$. As $\sigma_{j,i}$ may have new existential constants, one side effect of branching is that new instances may become realized. This forces us to actively manage "near-clashes"—assertions with $\bot$ or complementary literals—in M; if a constant is introduced, making their witness realizable, they will form a clash which must then be repaired.

- It will prove useful to bundle all original assertions in S, including those arising from the main labels of clauses according to Proposition 6.3.3, into one branch with index 0. This branch cannot be undone. If this branch contains a clash, then S is unsatisfiable.

Motivated by these considerations, we specify the data structure related to a branch in the algorithm:

Definition 6.4.1 *A* branch *is a tuple* (b, C, β, Λ), *consisting of:*

- *a positive integer* b, *the* branch index

[4]To avoid multiple labels on a branch, we could declare a new branch σ: b' on the same term. However, it would then be harder to ensure that a finite total number of branch indices is introduced. By contrast, if each occurrence of a term in S corresponds to only one branch index, their number is necessarily finite. Moreover, we will later define additional information stored with a branch, which is best shared among all labels branching on the same term.

- *a clause* $C = C(b) = \sigma_0\, F$ *and a term* $\beta = \beta(b)$ *in* C
- *a set* $\Lambda = \Lambda(b)$ *of complex labels, also called the* scope *of the branch. Each complex label* $\sigma - \Sigma \in \Lambda$ *must satisfy* $\sigma_0 \sqsubseteq \sigma$ *and* $\sigma \sqsubset \xi$ *for each exception* $\xi \in \Sigma$.

For ease of writing, we will consider a branch (b, C, β, Λ) and its index b synonymous, denoting either by the integer b. We will say that we *branch on a clause* C, if we pick a branch b so that $C = C(b)$. Henceforth the letter B shall denote the set of all branches at the current stage of the algorithm. In our upcoming code sections, we will assume that B is implemented as an array $(\Lambda(b))_{b\in B}$, and that $C(b)$ and $\beta(b)$ are given and fixed over the course of the algorithm. We observe that B implicitly specifies the set $M_0(B)$ of all assertions $(\sigma\sigma_i - \Sigma)\, a_i$, for each conjunct $\sigma_i\, a_i$ in the term $\beta(b)$, $\sigma - \Sigma \in \Lambda(b)$, and $b \in B$. Since all original assertions in S were grouped into branch 0, $M_0(B)$ includes *all* assertions relevant to the problem; indeed, when the algorithm terminates on a satisfiable problem, the assertions in $M_0(B)$ will "almost" form a model for S:

Proposition 6.4.2 *For any branch* (b, C, β, Λ) *in* B *and all instances* σ' *of any* $\sigma - \Sigma \in \Lambda$, $(M_0(B))_{\langle\sigma'\rangle} \models_s \beta$.

Proof: Writing β as $\bigwedge_{i=1}^n \sigma_i\, a_i$ as usual, and taking any branch label $\lambda = \sigma - \Sigma \in \Lambda$, let $M' \subseteq M_0(B)$ be the set of assertions $(\sigma\sigma_i - \Sigma)\, a_i$ contributed by the branch b over label λ. We stipulated that $\sigma - \Sigma \sqsubseteq \sigma'$, so σ' instantiates no exception in Σ, and upon subscripting M' becomes $M'_{\sigma'} = \{\sigma_i\, a_i : i = 1, \ldots, n\}$. We see that $M'_{\sigma'}$ contains, and hence satisfies, every conjunct in β, which shows $M'_{\sigma'} \models_s \beta$ and further $(M_0(B))_{\langle\sigma'\rangle} \models_s \beta$ by monotonicity. □

As outlined above, we initially branch over the outer label σ_0 of clause C. Later on, we may have to undo a branch over some instance ξ of σ_0; if ξ is realizable, we must eventually branch over it again, using some other term in C. We now claim that these operations of branching over σ_0, undoing a branch, and branching again over instances, actually have much in common: in a sense, they shift labels between branches in C and a set of "unbranched" labels. We can even consider this set a branch in its own right. Consider the following:

Proposition 6.4.3 *The clause* $C = \sigma_0\, (\beta_1 \vee \ldots \vee \beta_m)$ *and its extension* $\sigma_0\, (\bot \vee \beta_1 \vee \ldots \vee \beta_m)$ *are equivalent. Furthermore, if a set* M *contains the assertion* $\sigma_0\, \bot$, *then* $M \models_s C$.

Proof: We have $M \models_s C$ iff $M_{\langle\sigma\rangle} \models_s \beta_1 \vee \ldots \vee \beta_m$ for all instances σ of σ_0. If $M_{\langle\sigma\rangle}$ is utopian, then it satisfies anything, with or without an additional term $\bot$. Otherwise

$M_{\langle\sigma\rangle} \not\models_s \bot$, so $M_{\langle\sigma\rangle}$ satisfies $\bot \vee \beta_1 \vee \ldots \vee \beta_m$ iff it satisfies one of the "proper" disjuncts β_j, that is, iff it satisfies $\beta_1 \vee \ldots \vee \beta_m$. Since this is true for all instances σ, we conclude that $M \models_s C$ iff $M \models_s \sigma_0\ (\bot \vee \beta_1 \vee \ldots \vee \beta_m)$. To show the second part, we use Corollary 5.5.11, choosing $M' = \{\sigma_0\ \bot\}$. Then M'_{σ_0} is utopian, so it satisfies $\beta_1 \vee \ldots \vee \beta_m$; the label in M' contains no exception for which to verify anything recursively, so we have shown that $M \models_s C$. □

Because of this result, we can think of C as a clause with $m+1$ branches; we assign to the $(m+1)$st term $\bot$ the *default branch*, written $U(C)$. If no term in C has been branched upon (yet), we can consider it as branched on $U(C)$ over σ_0. In fact, if we add $\sigma_0\ \bot$ to M, then M verifies C as shown. Likewise, we can think of undoing a branch b over ξ as transferring the label ξ from b to the default branch $U(C)$. Again, if we add $\xi\ \bot$ to M, then M verifies the instance $\xi\ (\beta_1 \vee \ldots \vee \beta_m)$ of C. These considerations motivate the following:

Definition 6.4.4 *For each clause C, we define $U(C)$, the* default branch *or* unbranched labels, *of C. The set $\Lambda(U(C))$ must satisfy the same requirements as for a normal branch in Definition 6.4.1. All sets $U(C)$ are assigned the branch index ∞. We define $M(B) = M_0(B) \cup \{\lambda\ \bot :\ \lambda \in U(C), C \in S\}$.*

In an implementation, we need not represent the default branches explicitly; but for the purpose of describing our algorithm, we will think of unbranched labels λ as "branched on $\bot$ by default". This view is reflected in the set $M(B)$ in which we included the assertion $\lambda\ \bot$. This is not a mere formality. If $\lambda = \sigma - \Sigma$ is not realizable, we need not choose a "real" branch on $C = \sigma_0\ F$ over this label; but then we must include $\lambda\ \bot$ in the model M, so as to ensure that $M_{\langle\sigma\rangle} \models_s F$, which in turn is needed as part of showing $M \models_s \sigma\ F$. In the same way, if σ_0 is not realizable at all, we can satisfy the entire clause C without branching, by including $\sigma_0\ \bot$ in M. If we let $\sigma_0 \in U(C)$ for every clause C in S (which is in fact the initial state of the algorithm), then $M(B) \models_s S$. For a few sets like $S = \{*\,(p \vee q), *\,(\neg p \vee q), *\,(p \vee \neg q), *\,(\neg p \vee \neg q)\}$, $M(B) = \{*\,\bot\}$ is indeed a model for S. Usually however, we can expect at least some of the σ_0 to be realizable, so this default set $M(B)$ contains a clash. And repairing clashes due to a realizable $\lambda \in U(C)$ is tantamount to branching on λ; conceptually, this is not much different from repairing a clash arising from complementary literals, say.

Branching, or redoing a branch, can be regarded as just the opposite move: removing a label from the default branch $U(C)$, and adding it to another branch b on the clause. Here,

assigning branch index ∞ to all default branches is consistent with our idea of undoing maximal branches first. In essence we say: "If it is possible to branch, prefer branching to undoing any other branch."

Our first piece of code for the model-finding algorithm performs branching as described. At this point, let us introduce some conventions we will adhere to throughout the code sections: B is considered an array over the branch indices whose elements $B[b]$ are the sets $\Lambda(b)$ of branch labels, $B[0]$ is the default branch, and $M(B)$, the set of assertions in all branches including the default branches, is specified implicitly—we will not include code for computing it. The vectors $(C(b))_{b\in B}$ and $\beta(b)_{b\in B}$ are considered given and fixed. The sets B, S and $U(C)$, as well as the set of nogoods are considered global in scope, they may be freely accessed by any of the procedures. Unlike B, the original set S is considered read-only—we do not add clauses, not even original assertions, to S.

Algorithm 6.4.5 *simple-branching*

Parameters:

b: the branch

λ: the label branched upon

begin

remove λ from $U(C(b))$

add λ to $B[b]$

end

As we will see shortly, branching involves more than the two steps of this algorithm: we need to ensure that the branching step does not "intrude" into blocked, previously exhausted parts of the search space. We will reserve the remaining steps of the branching algorithm to Section 6.9, by which time we will have worked out those concepts more precisely.

The following definition will allow us to distinguish a set of branches obtained from the current set B by branching on unbranched labels alone, without undoing or repairing any branches in B:

Definition 6.4.6 *A set B' of branches is called an* extension *of B, if it is obtained from B by a finite number of applications of* simple-branching.

Corollary 6.4.7 *If B' is an extension of B, then $M_0(B) \subseteq M_0(B')$.*

Proof: This is a straightforward conclusion from the definition of $M_0(B)$ above: The set $M_0(B)$ includes all assertions introduced by branching on any label on any branch in B. All these branches and labels are also found in B', so $M_0(B)$ must be a subset of $M_0(B')$. □
Thanks to this result, any monotone property is preserved under extensions:

Corollary 6.4.8 *Any extension B' of B inherits the following properties, if satisfied for B:*

- *$M_0(B) \models_s F$ for any formula F,*
- *λ is realizable in $M_0(B)$, for any label λ,*
- *σ is strongly constant-wise realizable in $M_0(B)$, for any simple label σ,*
- *$M_0(B)$ contains a clash.*

Using these facts, we now show that branching can only introduce, never remove, realized ground instances:

Proposition 6.4.9 *For any extension B' of B, Every realized ground instance γ in $M(B)$ is an realized ground instance in $M(B')$.*

Proof: If all realizability witnesses for γ are (prefixes of) labels in $M_0(B)$, then they are also in the larger set $M(B)$. The only interesting case are the witnesses which are prefixes of some λ in a default branch $U(C)$. But observe that any extension of B is obtained by branching alone, and if branching removes λ from $U(C)$, it introduces the same label λ into some "real" branch on C. Branching may introduce additional exceptions into λ, but these are added back into $U(C)$. This shows that all witnesses for γ are preserved through branching. □
We also specify a procedure for *undoing* a branch:

Algorithm 6.4.10 *undo-branch*

Parameters:

b: the index of the branch to be undone

σ: the label on which branch b is to be undone

if $b = 0$

```
        exit (unsatisfiable)
else
        foreach label σ' − Σ ∈ Λ(b) do
                if σ ⊑ σ'
                        remove σ' − Σ from Λ(b)
                else if (σ, σ') exists and ξ ⋢ (σ, σ') for any ξ ∈ Σ
                        foreach exception ξ ∈ Σ do
                                if (σ, σ') ⊑ ξ
                                        remove ξ from Σ
                                end if
                        done
                        add (σ, σ') to Σ
                end if
        done
        if (b is not a default branch)
                add σ to U(C(b))
        end if
end if
```

This procedure will be called in various places in the algorithm where inconsistencies have occurred. It may even be called on instances of a default branch to be selected for branching. Notice that it includes one of the two termination criteria, which is the only place where the model-finding algorithm may terminate *abnormally*: if the procedure is called with branch index 0, the algorithm exits and returns *unsatisfiable*, since branch 0 cannot be undone. For the sake of greater efficiency, the procedure also performs a few simplifications:

- Any labels on b which instantiate σ are completely exempted, so they get erased.
- In any other label $\sigma' - \Sigma$ on b, the instances it has in common with σ are exempted. By adding (σ, σ') to Σ instead of σ, the procedure ensures that the exceptions in Σ remain strict instances of σ', which also entails that all branch labels are normalized.
- Any existing exceptions on such a label which instantiate the new exception (σ, σ') are redundant and get erased.

- Unless *undo-branch* has been called on the default branch itself, σ is added back to it[5].

Let us summarize these observations:

Proposition 6.4.11 *After a call to undo-branch on b and σ returns, we have $\lambda \not\sqsubseteq \sigma'$ for any label $\lambda \in \Lambda(b)$ and any instance σ' of σ.*

Proposition 6.4.12 *Let B' be a set of branches obtained from B by either of the following operations:*

(1) *deleting labels λ from some $\Lambda(b)$ on B,*

(2) *adding exceptions to some label $\lambda \in \Lambda(b)$ on B,*

(3) *calling* undo-branch.

Let further σ be a simple label. If σ instantiates a label on some branch b on B', it also instantiates a label on branch b on B.

Proof: Let $\lambda = \sigma' - \Sigma' \in \Lambda(b)$ in B' satisfy $\lambda \sqsubseteq \sigma$ as stated. Operation (1) leaves all labels unchanged in B', so the same λ exists on the same branch in B. Operation (2) adds exceptions to labels, so λ may correspond to a label $\lambda' = \sigma' - \Sigma''$ in $\Lambda(b)$ on B so that $\Sigma'' \subseteq \Sigma'$. Since $\lambda \sqsubseteq \sigma$ implies $\xi \not\sqsubseteq \sigma$ for any $\xi \in \Sigma'$, this is particularly true for all $\xi \in \Sigma''$, which shows $\lambda' \sqsubseteq \sigma$. Operation (3) just consists of a series of deletions of labels and additions of exceptions, so the proof follows from that for operations (1) and (2). □

We mentioned above that branching may introduce new constants, which in turn may lead to new realized ground instances in $M(B)$. To manage the realizability of labels over the course of the algorithm, we define the following:

Definition 6.4.13 *Let B be a set of branches. A label λ is* level-b-realizable *in $M(B)$, if it is realizable in $M(B')$, where B' is the subset of all branches in B with index less than or equal to b. Henceforth, whenever we speak of realized ground labels, realizable labels, or level-b-realizable labels, this implies $M(B)$ as the set of reference, where B is either explicitly specified or the current set of branches.*

[5]To be more efficient, σ should not be added back to $U(C(b))$ if it is already completely on the default branch: that is, if there exists $\sigma' - \Sigma' \in U(C(b))$ so that $\sigma' \sqsubseteq \sigma$ but (ξ, σ) does not exist for any $\xi \in \Sigma'$.

A level-0-realizable label is always realizable, no matter what the current set of branches B is, because its realizability witnesses are all taken from the original assertions on branch 0, which is always included in B. We formally declare a procedure for testing whether a label λ is level-b-realizable:

Algorithm 6.4.14 *is-realizable*

Parameters:

λ: a complex label

b: a branch index

Returns:

Δ: a set of branches and labels (see below), if λ is level-b-realizable;

false: otherwise.

We will not provide any code for this procedure, since an implementation is straightforward: one possible way is to determine the subset M' of assertions in $M(B)$ contributed by branches with index at most b, and then searching for a set of realizability witnesses for λ in M' according to Proposition 5.4.8. The set Δ is constructed as follows: for each witness (assertion) $(\sigma_i\sigma_i' - \Sigma_i)\ a$ (in the notation of Proposition 5.4.8), find a branch b_i and label $\sigma' - \Sigma_i$ which contributed this assertion into M', and include $(\sigma' - \Sigma_i)$: b_i into Δ.

Let us elaborate further on the choice of witnesses $(\sigma_i\sigma_i' - \Sigma_i)\ a$. In Proposition 5.4.8, it is the label σ_i which contributes some existential constants towards a ground instantiation γ. When viewed from the branch b_i over the label $(\sigma' - \Sigma_i)$, which introduced it, the same assertion is of the form $(\sigma'\sigma'' - \Sigma_i)\ a$. Now notice that it is only the existential constants in σ'', but not those in σ', which really get introduced through branch b_i. The branch label σ' must have already existed before we branched over it, and it is through some other branch that its constants have been introduced—possibly branch 0, which introduced the labels of all original assertions, as well as all outer clause labels σ_0. (Notice that $\sigma' = \varepsilon$ for $b = 0$.) From this discussion we infer that σ_i and σ'' must have a position (with an existential constant) in common; this entails that σ_i must be strictly longer than σ'. If this is not the case, the assertion $(\sigma_i\sigma_i' - \Sigma_i)\ a$ is unsuitable as a witness and should be replaced by other witnesses (which must exist, as we argued) introducing the same constants.

Since *is-realizable* returns *false* if no witnesses can be found, it can be used in an **if** statement to simply test whether λ is level-b-realizable or not. The parameter b can be omitted, in

which case the procedure searches for witnesses in all of $M(B)$; it thus tests whether λ is realizable. Finally, we recall that ground labels are realized iff they are realizable, so the procedure can also be used to find the witnesses of a realized ground instance.

Before we move on to the next section, let us write out a number of conventions for our algorithm: B is considered an array over the branch indices whose elements are sets of branch labels, $B[0]$ is the default branch, and $M(B)$, the set of assertions in all branches including the default branches, is specified implicitly—we will not include code for computing it. The vectors $(C(b))_{b\in B}$ and $\beta(b)_{b\in B}$ are considered given and fixed. The sets B, S and $U(C)$, as well as the set of nogoods are considered global in scope, they may be freely accessed by any of the procedures. We do consider S read-only, and we also restrict nogoods from being modified or deleted; we may only add new nogoods.

6.5 Nogoods

Nogood reasoning, in the form of *dynamic backtracking*, has been heralded as a powerful strategy in Constraint Satisfaction Problems [31]. Although it is not efficiently applicable to SAT in its pure form [24], adaptations to SAT have been conceived and demonstrated to improve the search efficiency drastically. On problems in **K**, a somewhat weaker but much simpler technique called backjumping [28] has led to similarly spectacular improvements [47].

The basic idea of nogood reasoning is that parts of the search space which are known to contain no satisfiable assignments should be blocked from future access in the search. Nogoods store the information about the particular choice of branches which define the "forbidden" parts of the search space. As the search proceeds without finding satisfying models, progressively more and/or larger chunks of the search space will be blocked, until eventually the entire search space becomes inaccessible, in which case we have shown the problem unsatisfiable. Nogoods can aid a reasoning algorithm in several ways:

- to ensure termination. If the search space is finite and discrete, and if we can show that each nogood blocks a nonvanishing part of hitherto accessible search space, the search must terminate after a finite number of nogoods have been introduced.

- to ensure soundness. If we the blocked parts of the search space provably do not contain any solution, then once the entire search space is blocked and the algorithm

returns *unsatisfiable*, we can be sure that no model exists.

- to ensure completeness. If we can show that a so-called *empty nogood*[6]—a nogood which blocks the entire search space—can be derived from any unsatisfiable formula, this empty nogood will block the entire search space and lead the algorithm to return *unsatisfiable*.

It follows that a sound, complete, and terminating algorithm could be given based on nogood reasoning alone. In fact, reasoning with nogoods is a derivation of resolution [77, 68, 6], as we will discuss in Section 7.2. However, we will use nogoods only as a tool to ensure termination and completeness of our *model finding* algorithm.
As in the previous section on branches, we will informally discuss what information should be associated with a nogood:

- Similarly as in the propositional case, nogoods should be characterized by a *dependency set*: the set of assertions branched upon, which led to unsatisfiability. In order to avoid redundancies, we do not duplicate the assertions themselves but refer to the branches which contributed them, via their branch index.

- We must pinpoint as precisely as possible the instances (labels) of each participating branch which caused the inconsistency. Let us refresh our memory of Example 6.1.2: When we picked Branch 1 on the clause $* (p \vee q)$ and Branch 3 on c_1 $(\neg p \vee f)$, we discovered that $* p$ and c_1 $\neg p$ form a clash. While Branch 1 was scoped over $*$ at that time, only the instance c_1 actually contributed to the clash. So in the dependency set, rather than including the Branch $b = 1$ over $*$, we only include it over the instance c_1.

- Now does this requirement for precisely specifying the cause of a nogood extend to exceptions as well? Suppose a branch b_2 over $*$ introduced two assertions $* p$ and $* q$, and a clash with another assertion 1 $\neg p$, resulting from an earlier branch b_1, has led us to revise the branch label to $* - \{1\}$. If we encounter a clash with an existing assertion $* \neg q$ later on, the clash witness, strictly speaking, is $* - \{1\}$. (We assume that this witness is realizable.) However, we argue that, had the scope of branch b_2 been $*$, the clash witness would extend to the entire label $*$. The exception 1 has nothing to do with the inconsistency. We will see that in establishing the dependency sets of nogoods,

[6] Unlike in SAT, empty nogoods in $\mathcal{LBL}$ are not unique. See our discussion below.

it is always safe to disregard exceptions in the current scope of the branches, so we will use only simple labels in dependency sets. In fact, storing exceptions in nogoods would lead to inefficiencies: If the earlier branch b_1 is undone, we would rediscover the same clash and introduce the same nogood as before, but on the instance 1. Therefore,

- Nogoods are a static piece of information. They do not change as the algorithm branches on new instances or repairs instances. Even if we completely undo one of the branch labels participating in a nogood, the nogood remains valid, so it will serve in blocking the algorithm from re-establishing this branch. However, nogoods may become redundant or useless, and since they accumulate during the course of the algorithm, we may need to actively manage them in practice. We will briefly talk about this in Section 6.12. In the algorithm we present here, however, nogoods are restricted from being modified or deleted; we may only add new nogoods.

Our definition of nogoods takes these considerations into account, as well as others which we will leave for later discussion:

Definition 6.5.1 *A* nogood-like expression *is an expression of the form* $\Delta \rightarrow \Omega$, *where* Δ *is a finite set of* premises *of the form* σ_i: b_i *and* Ω *is a finite set of* conclusions $\sigma\ a$. *The labels* σ_i *are the* participating branch labels, *and the labels* σ *are the* witnesses. *(We refer to both kinds collectively as "nogood labels"; their syntax extends that of simple labels and will be more precisely characterized later.) If all conclusions in* Ω *are of the form* $\sigma\ \bot$, *the expression* $\Delta \rightarrow \Omega$ *is called a* nogood; *if additionally* $\Delta = \emptyset$, *we speak of an* empty nogood. *We use NG to denote nogoods.*

For most of this and upcoming sections, we will only consider nogoods; but nogood-like expressions will prove useful in stating nogood propagation rules, and in the final section of this chapter we will outline how they can be used to implement an extension of Boolean Constraint Propagation. For now, we can make the simplifying assumption that a nogood is of the form σ_1: $b_1, \ldots, \sigma_n$: $b_n \rightarrow \sigma\ \bot$. In the informal example earlier this section, the nogood arising from the clash between branches b_1 and b_2 is 1: b_1, 1: $b_2 \rightarrow 1\ \bot$, whereas the nogood resulting from the second clash is $*_t$: b_2, $*_t$: b_3, $\rightarrow *_t\ \bot$. The $*_t$ obey the generalized syntax for labels in nogoods and will be explained further below.

Many of the operations on nogoods depend on a linear order between the nogood premises. The ascending order of branch indices mostly provides for this; however, we will demonstrate that the branch indices b_i need not all be distinct, in which case we need to be more precise.

Definition 6.5.2 *Assume a linear order $\leq$ on the domain D of constants for labels given. We define a total order $\preceq$ on simple labels as the lexicographic order obtained from the order $\leq$ on their positions, defining $* \leq c$ for all $c \in D$. We extend this to a total order $\preceq$ on branch premises by: σ_1: $b_1 \preceq \sigma_2$: b_2 iff $b_1 < b_2$, or $b_1 = b_2$ and $\sigma_1 \preceq \sigma_2$.*

For instance, $*2$: $b \preceq 1*$: b, but $*1$: $b \preceq *2$: b. It is easy to verify that $\preceq$ defines a total order on simple labels, and hence on branch premises. We also point out the useful fact that $\sigma \sqsubseteq \sigma'$ implies $\sigma \preceq \sigma'$. Without loss of generality, we will henceforth assume that the nogood premises are sorted in ascending order. (In our algorithm, we will make provisions for that.) Premises with branch index 0 need not be mentioned in any nogood, since assertions from this branch are always present and cannot be undone anyway.

The maximal premise σ_n: b_n on the nogood is special in two ways: b_n is the (first) branch to be undone, namely, on (some instance of) σ_n, in order to repair the inconsistency represented by this nogood. Furthermore, whenever we try to re-branch on b_n over some instance of σ, we will first check whether this is sanctioned by the nogood (which requires that another branch b_i has been undone since the nogood was introduced). To mark the special role of this premise we add a reference to *NG* onto the branch b_n, and we call *NG* a *term nogood* of b_n (or of the term $\beta(b_n)$). Referencing term nogoods from branches is merely a matter of convenience, so we can easily look up all nogoods in which b_n is the maximal participating branch, to test whether branching on b_n is sanctioned. For empty nogoods, we do not have any participating branch index, so we declare $b_n = 0$. (Empty nogoods do not become term nogoods, however.)

During our algorithm, we will encounter two types of *basic nogoods* and two types of *propagated nogoods* which are derived from others:

- Basic nogoods are created in response to clashes introduced by branching. The two types of basic nogoods correspond to the two types of clashes—$\bot$ and complementary literals; the former contains one nogood premise, while the latter usually has two. We will cover the two basic types of nogoods in Section 6.6.

- One type of propagated nogoods arises from *failed clauses*. These are clauses $C = \sigma_0 F$ in which we have tried branching on every one of its terms, and in some instances of σ_0 none of the branches produces a clash-free set of assertions. At this point, term nogoods exist for every term in C. In the SAT equivalent, this is where backtracking

occurs. In Section 6.8 we will define a procedure called *nogood merging*, which combines the participating term nogoods into a new nogood and uses it to undo other branches, so that the failed clause can be branched upon again.

The fact that different term nogoods may have different witnesses requires us to copy all conclusions as well. This is why we defined nogoods with multiple witnesses, and they only arise through nogood merging. For instance, if two nogoods with respective witnesses $1*$ and $2*$ are merged, the resulting nogood is violated only when both $1*$ and $2*$ are realizable, and we cannot express this fact by one nogood witness alone. Nonetheless, nogoods with multiple witnesses are awkward in practice, so they should be avoided whenever possible.

- Another type of propagated nogoods has to do with nogood witnesses. First, we will need nogoods to block branches which might make a witness σ of another nogood realizable; in other situations, we may wish to derive a nogood with fewer or shorter witnesses. In Section 6.7, we will demonstrate how branches and labels witnessing the realizability of σ can be merged into an existing nogood in order to accomplish this. We call this *realizability propagation.*

Example 6.5.3 *In this example, we will demonstrate how nogood merging can lead to multiple occurrences of the same branch index in a nogood. Consider the following set of clauses:* $S = \{*\,(p \vee s), 1\,\neg p \vee q, 2\,\neg p \vee r, \neg q \vee \neg r\}$. *Let us assign branch indices* $b_{i,1}$, $b_{i,2}$ *to the first and second term in the* i*th formula,* $i = 1, \ldots, 4$. *Assume we branch on* $b_{1,1}$ *first, obtaining the assertion* $* p$. *Then in the second and third formula, branching on* $1\,\neg p$ *and* $2\,\neg p$ *leads to clashes; the corresponding nogoods obtained are* $NG_1 = 1\,b_{1,1}, 1\,b_{2,1} \rightarrow \bot$ *and* $NG_2 = 2\,b_{1,1}, 1\,b_{3,1} \rightarrow \bot$. *Furthermore, we must branch on* q *and* r*, which causes the fourth clause to fail. Skipping some intermediate steps, we derive the nogood* $NG_3 = b_{2,2}, b_{3,2} \rightarrow \bot$. *Now the third clause fails, since both terms have a nogood associated with them:* NG_2 *and* NG_3. *We merge them into* $NG_4 = 2\,b_{1,1}, b_{2,2} \rightarrow \bot$. *Now in turn the second clause fails, causing us to merge the two term nogoods* NG_1 *and* NG_4 *into* $NG_5 = 1\,b_{1,1}, 2\,b_{1,1} \rightarrow \bot$.

Since we have defined a total order on branch indices as well as branch labels, undoing branches in response to such a nogood is deterministic. This is important: we do not know at this point whether undoing $1\,b_{1,1}$ or $2\,b_{1,1}$ would lead to a solution. In the above example, either one does; but in the presence of other clauses, one of the assertions $1\,p$ or $2\,p$ may be

mandatory for a solution. Given the natural order of integer branch labels, we would undo $2\,b_{1,1}$ first; if this does not lead to a solution, we would (obtain a nogood $1\,b_{1,1} \to \bot$,) undo branch $1\,b_{1,1}$, and at that point be free to branch on $2\,b_{1,1}$ without incurring any nogood violation. Specifying a well-defined order for undoing nogood premises will form an essential part of our termination proof[7].

Let us now motivate the already mentioned important refinement on the syntax of nogood labels:

Example 6.5.4 *Consider a clause $C = *p \vee *q$. Suppose at some point during the search we branched on $* \neg p$ (with index b_1) and on $* \neg q$ (with index b_2), occurring elsewhere in S. This would result in the clause C failing on its outer label ε (not $*$!). Consequently, we would introduce a nogood $* b_1, * b_2 \to \varepsilon\,\bot$. We read this nogood as: "Branching on b_1 on* any *instance of $*$ and branching on b_2 on* any *instance of $*$ leads to failure."*

Example 6.5.5 *Now consider the slightly different clause $C = *(p \vee q)$, and assume that we branched on the same terms $* \neg p$ (index b_1) and $* \neg q$ (index b_2). Then the clause $*(p \vee q)$ also fails on its outer label, which is $*$ here. The corresponding nogood has the intended reading: "Whenever we branch on b_1 and b_2 on* the same *instance of $*$, this leads to failure on that same instance." This reading is markedly different, not just because the $*$ must be found in the nogood witness, but also because there is a dependency between the $*$ in the branches and the witness; this reflects the different role wildcards play in the outer versus the inner labels of a clause.*

What makes matters worse is that such nogoods will be propagated through nogood merging or realizability propagation. The implicit dependency between the $*$ would quickly get lost, resulting in unsoundness. Therefore, here (and only here) will we abandon the anonymity of the wildcards and turn them into "proper" variables[8], by indexing the $*$ with positive integers. To illustrate this, the nogood in example 6.5.5 might now read: $*_1\colon b_1, *_1\colon b_2 \to$

[7]The complications described here, due to multiple occurrences of branches with the same index, are also known to those in the QBF reasoning community who work on resolution, whether as a proof method or as an engine for clause learning. In QBF, multiple occurrences of the same branch (there: propositional variable) in nogoods (or:resolvents), and in fact the need to store some form of "labels" (there: a context of previous variable assignments) with a branch, can be avoided by a method called *Q-resolution*. For details, see [13, 62]. We believe it possible to derive an equivalent of Q-resolution for $\mathcal{LBL}$; however, as in QBF, this will put a restriction on the order in which we can perform branching, which may be undesirable.

[8]In a similar way, the free-variable calculus [7, 9] distinguishes between *universal* variables (corresponding to our $*$), and *free* variables (such as we need here).

$*_1 \perp$. With each new nogood introduced which requires indexing of wildcards, we will use a "fresh" index, that is, an index greater than the maximal index which has been assigned before. We only introduce at most one new index per nogood; if there is more than one position in a label which needs to be indexed, it will be understood that occurrences of the same $*_t$ in different positions can be instantiated by different constants. However, occurrences of the same $*_t$ in the same position in different nogood labels must henceforth be instantiated uniformly, either by the same constants or by a new $*_{t'}$, $t' > t$. In the example, if we branch on b_1 over the instance 1, the nogood will block application of branch b_2 over the same instance 1 only, but we could still branch on b_2 over another instance 2, or even on $* - \{1\}$. The number of positions to be indexed depends on the structure of the nogood and the dependencies between branches it expresses, as demonstrated in Examples 6.5.4 and 6.5.5. We will work out clear rules for all types of nogoods in the sections to come. Nogood indices are relevant only when propagating nogoods and checking for nogood violations. In undoing a branch and creating exceptions to branch labels, the indices are irrelevant and will be omitted. Finally, we add to Definition 6.5.2 and to our definition of the mgu of two labels: Let t, t' be positive integers, and $c \in D$. Then $* \leq *_t \leq c$, and $*_t \leq *_{t'}$ iff $t \leq t'$; furthermore, assuming $t \leq t'$, we have $(*_{t'}, *_t) = (*_t, *_{t'}) = *_{t'}$, $(*, *_t) = (*_t, *) = *_t$, and $(c, *_t) = (*_t, c) = c$. The $\sqsubseteq$ relation extends accordingly[9].

Definition 6.5.6 *Consider a nogood* $NG = \Delta \rightarrow \Omega$. *A* nogood substitution *is a function* ρ *which simultaneously instantiates the labels* $\sigma_i \in Delta$ *by* $\sigma_i\rho$ *and the* $\sigma \in \Omega$ *by* $\sigma\rho$, *whereas:*

- *each* $*_t$ *(occurring in the same position in any of the nogood labels) is instantiated uniformly by the* same *element* c, [c] $(c \in D)$, $*$, *or* $*_{t'}$, $t' > t$,
- *each* $*$ *is instantiated by an arbitrary such element,*
- *each conditional constant* [c], $c \in D$, *is instantiated uniformly by* [c] *or* c,
- *each existential constant is instantiated by itself.*

A nogood substitution ρ *is* more general *than* ρ' *if* $\sigma\rho \sqsubseteq \sigma\rho'$ *and* $\sigma_i\rho \sqsubseteq \sigma_i\rho'$ *for all* $i = 1, \ldots, n$.

[9]Stretching our terminology, we will talk of an "instantiation" of $*_t$ by $*$, although strictly speaking, $*$ does not instantiate $*_t$.

We introduce a few types of nogood substitution which we deem elementary and use frequently in upcoming nogood rules. In our discussion below, σ_i can stand in for any of the nogood labels, *including* any of the witnesses σ:

- the *substitution induced by uniform instantiation*: Consider a label σ_i' (usually but not necessarily an instance of σ_i). Then we define by $\rho(\sigma_i \mapsto \sigma_i')$ the most general substitution ρ so that $\sigma_i' \sqsubseteq \sigma_i\rho$. Such a substitution only exists if σ_i and σ_i' unify, and we easily see that $\rho(\sigma_i \mapsto \sigma_i')$ instantiates labels as follows: Replace σ_i by (σ_i, σ_i'), and in each position of σ_i where $*_t$ has been instantiated by an element x from (σ_i, σ_i'), replace $*_t$ uniformly by x in all $\sigma_{i'}$, $i' \neq i$. Here x can be an existential or conditional constant or an indexed or unindexed $*$.

- In the same manner, we define $\rho(\sigma_{i_1} \mapsto \sigma_{i_1}', \ldots, \sigma_{i_n} \mapsto \sigma_{i_n}') = \rho(\sigma_{i_1} \mapsto \sigma_{i_1}') \cdots \rho(\sigma_{i_n} \mapsto \sigma_{i_n}')$, the substitution induced by successive uniform instantiation of $\sigma_{i_1}, \ldots, \sigma_{i_n}$ with $\sigma_{i_1}', \ldots, \sigma_{i_n}'$, respectively. Note that even if all σ_{i_j}' instantiate σ_{i_j}, $j = 1, \ldots, n$, this substitution may not exist. For instance, two $*_t$ in the same positions in σ_{i_1} and σ_{i_2} may be instantiated to different constants c_1, c_2 in σ_{i_1}' and σ_{i_2}'. Upon the first instantiation $\sigma_{i_1} \mapsto \sigma_{i_1}'$, the $*_t$ is uniformly instantiated by c_1, so now $\sigma_{i_2}\rho(\sigma_{i_1} \mapsto \sigma_{i_1}')$ no longer unifies with σ_{i_2}', which has c_2 in the position in question. Therefore, $\rho(\sigma_{i_1} \mapsto \sigma_{i_1}', \sigma_{i_2} \mapsto \sigma_{i_2}')$ does not exist.

  We mention two important properties of successive instantiations without proof:

  - $\rho(\sigma_i \mapsto \sigma_i', \sigma_i' \mapsto \sigma_i'') = \rho(\sigma_i \mapsto \sigma_i'')$, provided the instantiations exist.
  - If $i \neq j$, then $\rho(\sigma_i \mapsto \sigma_i', \sigma_j \mapsto \sigma_j') = \rho(\sigma_j \mapsto \sigma_j', \sigma_i \mapsto \sigma_i')$, modulo indexing of $*_t$.

- As a special case of successive uniform instantiation, we define the *substitution induced by re-indexing* $\rho(* \mapsto *_t, \sigma_i^p)$, where σ_i^p is a prefix of σ_i, and t is a fresh index. In this substitution, all occurrences of $*$ and $*_{t'}$ in σ_i^p, as well as all corresponding occurrences of the same $*_{t'}$ in the other labels $\sigma_{i'}$, $i' \neq i$, are replaced by $*_t$.

- Finally, we have the *index-eliminating substitution* $\rho(*)$, which replaces all occurrences of indexed $*_b$ with $*$. This is useful for obtaining "ordinary" labels from nogood labels. (In order to keep things simple, we will usually omit this substitution. It is implicitly used, for instance, when a branch in undone over a label which originates from a nogood and thus may have indexed $*$.)

Definition 6.5.7 *Consider a nogood-like expression* $NG = \Delta \rightarrow \Omega$ *with* σ_1: $b_1, \ldots, \sigma_n$: b_n *as premises and* σ *being one of the witnesses in* Ω. *Assume further that* $\lambda_i = \sigma_i' - \Sigma_i \in \Lambda(b_i)$, $i = 1, \ldots, n$, *are labels on the current set of branches* B, *and let* σ' *be a simple label. A* nogood-violating substitution *(ngvs) is a substitution* $\rho(\sigma \mapsto \sigma', \sigma_1 \mapsto \sigma_1', \ldots, \sigma_n \mapsto \sigma_n')$ *so that* $\lambda_i \sqsubseteq \sigma_i\rho$ *for all* $i = 1, \ldots, n$. *The* most general *nogood-violating substitution is that where* $\sigma' = \sigma$. *If such a substitution exists for all witnesses* σ *in* Ω, B *is said to* violate *(or* match, *in case NG is not a nogood) NG with most general nogood-violating substitution* $\rho = \rho(\sigma_1 \mapsto \sigma_1', \ldots, \sigma_n \mapsto \sigma_n')$; *the* $\sigma_i\rho$ *are the* violating instances *of* b_i, $i = 1, \ldots, n$; *and* $\sigma\rho\rho(*)$ *the* witness *of the violation. Additionally, if for every witness* σ *in* Ω *there exists a realized ground instance* γ *of* σ *in* $M(B)$ *so that* $\rho(\sigma \mapsto \gamma, \sigma_1 \mapsto \sigma_1', \ldots, \sigma_n \mapsto \sigma_n')$ *is a nogood-violating substitution, we say that* B realizably violates *NG. If the branches witnessing the realized ground instance* γ *all have indices no larger than* b, *we say that* B level-b-realizably violates *NG.*

It may sound confusing, but even if B violates a nogood NG and the witness $\sigma_i\rho$ of the violation is realizable, NG may not be realizably violated:

Example 6.5.8 Consider a nogood $NG = \sigma_1$: $b \rightarrow \sigma\ \bot$ with $\sigma_1 = *_t$ and $\sigma = *_t*$, and suppose that an assertion 11 $\top$ exists on branch 0, 11 constituting the only realized ground instance of $*_t*$. Then picking branch b over the label $* - \{1\}$ violates NG with witness $**$, but not realizably so: The substitution $\rho = \rho(*_t \mapsto *)$ results in $\sigma_1\rho = *$ instantiating $* - \{1\}$; hence ρ is a nogood-violating substitution. However, any substitution ρ' which maps σ to the only available rrgl 11 entails $\sigma_1\rho' = 1$, which does not instantiate $* - \{1\}$. We see how the exceptions in the branch labels λ_i can cause the substitution ρ' to not violate the same nogood NG, even though it is less general than ρ. If we had branched over $*$ instead, NG would be realizably violated.

Despite this, it still suffices to characterize a realizable nogood violation by the most general ngvs ρ and a ground instance γ of each nogood witness $\sigma\rho$, since the substitution $\rho(\sigma \mapsto \gamma, \sigma_1 \mapsto \sigma_1', \ldots, \sigma_n \mapsto \sigma_n')$ required by Definition 6.5.7 is simply the substitution $\rho(\sigma \mapsto \gamma)\rho$. Hence, we can simultaneously find the most general ngvs for a nogood, as well as the witnesses γ for its realizable violation.

We will formally declare two procedures implementing Definition 6.5.7, for use in later sections of the algorithm. One procedure is used for "ordinary" nogood violations, the

other for level-b-realizable nogood violations. We will not provide pseudocode for actually computing the nogood-violating substitutions and witnesses, which is fairly straightforward.

Algorithm 6.5.9 *nogood-violated*

Parameters:

$NG = \Delta \rightarrow \Omega$: the nogood to be tested

Returns:

ρ: a most general nogood-violating substitution, if one exists

false otherwise

Algorithm 6.5.10 *realizably-violated*

Parameters:

$NG = \Delta \rightarrow \Omega$: the nogood to be tested

b: a branch index

Returns:

a tuple $(\rho, (\gamma_\sigma, \Delta_\sigma))$, if NG is level-b-realizably violated

ρ is the most general nogood-violating substitution

γ_σ is an rgi of σ

Δ_σ is a set of level-b-realizability witnesses of σ

(for each σ in Ω)

false otherwise

Similarly as in *is-realizable*, the parameter b may be omitted, in which case the procedure checks whether NG is realizably violated, with no restriction on the branch index; either procedure can be used inside an **if** statement to simply test whether a nogood is (level-b-)realizably violated. In parts of our algorithm later on, we will often perform a combination of tests on the same nogood; since testing for nogood violations is a fairly expensive procedure, an optimized implementation should perform these tests simultaneously to the greatest extent possible, reusing partial results from earlier tests. In our algorithm description, we prefer simplicity instead: we will simply call the above procedures anew, each time we need to test for a nogood violation.

A realizable nogood violation should indicate the presence of some inconsistency in B, so unless we remove the cause of the nogood violation by undoing branches, B cannot be extended to a satisfying set of branches. Let us state this formally as a paradigm which we must prove for all types of simple or propagated nogoods we will define. We will include the more general nogood-like expressions as well:

Paradigm 6.5.11 *Let a nogood-like expression $NG = \Delta \rightarrow \Omega$ and current set of branches B be given. Then for the most general nogood-violating substitution ρ, one of the conclusions $\sigma\, a$ in Ω satisfies:*

(1) *For all prefixes of $\sigma\rho$ of the form $\hat{\sigma}c$, c exists in $(M(B))_{\langle\hat{\sigma}\rangle}$.*

(2) *If $a = \bot$: $(M(B))_{\langle\sigma\rho\rangle}$ contains a clash.*

(3) *If a is a literal: $(M(B))_{\langle\sigma\rho\rangle}$ contains $\varepsilon\, a$.*

Defying appearances, we do not claim $\sigma\rho$ to be strongly constant-wise realizable in $M(B)$. Condition (1) given in the paradigm mentions $Q_c(\varepsilon, (M(B))_{\langle\hat{\sigma}\rangle}$ alright (see Definition 5.4.12), but this only forms the "first step" of testing whether $\sigma\rho$ is strongly constant-wise realizable; if some of the branch labels λ_i have exceptions, the nogood does not and cannot guarantee that every strict exception-generated instance of $\sigma\rho$ is also strongly constant-wise realizable. We do infer though that condition (1) is a monotone property. Since any extension B' of a set of branches B contains all branch labels on B, and since all the conditions given above are monotone properties, we conclude (compare Corollary 6.4.8):

Corollary 6.5.12 *Assuming Paradigm 6.5.11, if B violates a nogood σ_1: $b_1, \ldots, \sigma_n$: $b_n \rightarrow \sigma\, \bot$ with nogood-violating substitution ρ, then so does any set of branches B' extending B.*

As we mentioned, violated nogood do not necessarily pose a problem, but realizably violated nogoods always do. We now see why this is so. We can live with a clash in $(M(B))_{\langle\sigma\rho\rangle}$, as long as $\sigma\rho$ is not realizable. But:

Corollary 6.5.13 *If B realizably violates a nogood with nogood-violating substitution ρ, then $M(B)$ contains a clash, and so does $M(B')$ for any set of branches B' extending B.*

Proof: According to Paradigm 6.5.11, $(M(B))_{\langle\sigma\rho\rangle}$ contains a clash for one of the witnesses σ in Ω. Furthermore, $\gamma = \sigma\rho$ is ground and realized in $M(B)$, for *all* witnesses σ, including

the one above. So the converse of Proposition 5.5.4 shows that $M(B)$ itself contains a clash. The same follows for $M(B')$, because $\sigma\rho$ continues to be an rgi in $M(B')$, according to Proposition 6.4.9. □

A slightly modified version of Corollary 6.5.13 takes into account the branch level at which a witness becomes realizable:

Corollary 6.5.14 *If B level-b-realizably violates a nogood, where b is greater than or equal to the maximum participating branch index b_n, then for any set of branches B' identical with B on all branches with index up to b, $M(B')$ contains a clash.*

Proof: Let ρ be a nogood-violating substitution and σ the nogood witness of Corollary 6.5.13, so that the witness $\sigma\rho$ is an rgl witnessed by branches with index up to b alone. Likewise, the branches b_i all have branch indices at most b. Therefore, the realizable nogood violation persists on any set B' which is identical with B on all branches up to b, and the *first* part of Corollary 6.5.13 shows that $M(B')$ contains a clash. □

These results clarify in what way a realizable nogood violation in a set of branches is related to other forms of clashes we encountered so far, namely: clashes in $M_0(B)$ caused by assertions contributed from branches, and clashes in $M(B) - M_0(B)$ caused by realizable unbranched labels. By contrast, clashes due to nogood violations are not *caused* by the nogood. Corollaries 6.5.13 and 6.5.14 only state that whenever a nogood is realizably violated, $M(B)$ must have a clash; this clash can be of one or the other type. Of clashes in $M_0(B)$, we know from Corollary 6.4.8 that these clashes persist in all extensions of B, whereas clashes due to unbranched labels are typically removed by branching over that label. If a nogood is violated, however, Corollaries 6.5.13 and 6.5.14 state that even if the actual clash in $M(B)$ is due to an unbranched label λ, no extension of B will be clash-free, so we need not waste efforts trying to branch over λ. In this sense, a nogood blocks the search space represented by all extensions of B. (The purpose of nogoods, after all, is to "remember" previously found inconsistencies which are not obvious, so we need not search the blocked space unsuccessfully again.)

We will now encounter our first rule for propagating a nogood, for use in upcoming sections. We write these rules schematically, in a similar style to tableau rules (see the Appendix), or to other rule-based systems in the literature. In the top part of a rule, we write out all nogoods and nogood-like expressions which must currently exist; at the bottom, we list the nogood-like expression(s) we can derive. Additionally, a rule may have side conditions: for

instance, the term $\beta(b)$ of a branch may be required to satisfy certain properties in order to include branch b in a nogood. We must formally verify each rule by showing that whenever all nogood-like expressions at the top satisfy Paradigm 6.5.11 (and the side conditions hold), then the expressions at the bottom also satisfy Paradigm 6.5.11. Once this is established, we know that it is correct to add any of the bottom nogoods whenever all the top nogoods are already present. Note that this does *not* mean that whenever the nogoods at the top are violated, so is (are) the nogood(s) at the bottom. We will need to verify this separately, using further side conditions, for the nogood propagations we will discuss in Sections 6.7 and 6.8.

Proposition 6.5.15 *We have the following* instantiation rule *for nogood-like expressions:*

$$\frac{NG}{NG\rho(\sigma_i \mapsto \sigma_i')} \quad ,$$

where σ_i is one of the participating branch labels in NG.

Proof: Let B any set of branches with labels $\lambda_j = \sigma_j'' - \Sigma_j \in \Lambda(b_j)$, $j = 1, \ldots, n$, so that $\rho = \rho(\sigma_1 \mapsto \sigma_1'', \ldots, \sigma_i' \mapsto \sigma_i'', \ldots, \sigma_n \mapsto \sigma_n'')$ is the most general nogood-violating substitution for $NG' = NG\rho(\sigma_i \mapsto \sigma_i')$. We transform the composite substitution $\rho(\sigma_i \mapsto \sigma_i')\rho$ using the results we stated informally for successive uniform instantiations:

$$\begin{aligned}
& \rho(\sigma_i \mapsto \sigma_i')\rho(\sigma_1 \mapsto \sigma_1'', \ldots, \sigma_i' \mapsto \sigma_i'', \ldots, \sigma_n \mapsto \sigma_n'') \\
= \quad & \rho(\sigma_i \mapsto \sigma_i', \sigma_1 \mapsto \sigma_1'', \ldots, \sigma_i' \mapsto \sigma_i'', \ldots, \sigma_n \mapsto \sigma_n'') \\
= \quad & \rho(\sigma_1 \mapsto \sigma_1'', \ldots, \sigma_i \mapsto \sigma_i', \sigma_i' \mapsto \sigma_i'', \ldots, \sigma_n \mapsto \sigma_n'') \\
= \quad & \rho(\sigma_1 \mapsto \sigma_1'', \ldots, \sigma_i \mapsto \sigma_i'', \ldots, \sigma_n \mapsto \sigma_n'').
\end{aligned}$$

Since $\lambda_i \sqsubseteq \sigma_i'\rho = \sigma_i\rho(\sigma_i \mapsto \sigma_i')\rho$, the substitution $\rho(\sigma_i \mapsto \sigma_i')\rho$ is the most general nogood-violating substitution for NG with the same witness $\sigma\rho(\sigma_i \mapsto \sigma_i')\rho$. We apply Paradigm 6.5.11 to NG and obtain all its properties for NG' as well. □

It is not practical to store instantiations of nogoods separately, since they are so easily derivable from the original nogood. This proposition is merely a conceptual aid which will make it easier for us to verify Paradigm 6.5.11 for more complicated nogood propagation rules.

Note that the instantiation rule does not sanction a nogood instantiation of the form $NG\rho(\sigma \mapsto \sigma')$ for any of the nogood witnesses. This substitution may introduce existential

constants, and we cannot guarantee in general that the nogood premises alone cause these constants to "exist" in any branch, so we cannot prove condition (1) in Paradigm 6.5.11. In section 6.7, we will show how we can integrate "witnesses" for constants into a nogood.
Having said enough about nogood violations, we will now turn towards a method for repairing nogoods. Our procedure is intended for use with a nogood which is known to be violated:

Algorithm 6.5.16 *repair-nogood-violation*

Parameters:

$NG = \sigma_1 \colon b_1, \ldots, \sigma_n \colon b_n \rightarrow \Omega$: the nogood

if *realizably-violated* (NG, b_n)

set $\rho :=$ *nogood-violated*(NG)

undo-branch (b_n, $\sigma_n\rho$)

else

undo-realizability (NG)

end if

When repairing nogood violations, we remain faithful to our strategy of undoing largest branches (wrt $\preceq$) first: If all nogood witnesses are level-b_n-realizable, the **if** part is executed, and we undo the largest branch b_n on its violating instance. If some nogood witness is not level-b_n-realizable, we will see that NG can be made non-realizably violated (and hence "harmless") by undoing branches with index greater than b_n which contributed to its realizability, which again corresponds to our strategy above. The procedure *undo-realizability* accomplishes this; it is used in several places, and we will describe it in Section 6.7.
For now, let us just consider the **if** part, and let us also assume that NG is not an empty nogood. We will make a number of remarks, showing that the nogood repair delivers on our expectations we addressed informally earlier this section: it pinpoints, in some sense, the "minimal" instance on which to undo branch b_n so that NG is no longer violated; and henceforth NG blocks the algorithm from re-branching on b_n over that instance.

- Let ρ be a nogood substitution on NG. If there exists a branch b_i so that $\lambda_i \not\sqsubseteq \sigma_i\rho$ for every $\lambda_i \in \Lambda(b_i)$, then ρ is not nogood-violating.

  This is exactly the definition of a ngvs, stated in the contrapositive.

- After *undo-branch* is executed on branch b_i over $\sigma_i\rho$, ρ is no longer nogood-violating, nor is any less general substitution ρ'.

  This follows from the previous remark and from Proposition 6.4.11, which warrants that $\lambda_i \not\sqsubseteq \sigma_i\rho'$ for every $\lambda_i \in \Lambda(b_i)$. (Notice that $\sigma_i\rho'$ instantiates $\sigma_i\rho$.)

  It is possible that several labels on b_i "cause" nogood violations, and the instances $\sigma_i\rho$ for the corresponding most general ngvs may even be disjoint. In this case, undoing branch b_i on σ_i will not make the entire nogood non-violated. We have shown, however, that the substitution $\rho(\sigma_1 \mapsto \sigma'_1, \ldots, \sigma_n \mapsto \sigma'_n)$ obtained for the specific tuple $\lambda_1, \ldots, \lambda_n$ will no longer be a nogood-violating substitution. If we repeatedly call *repair-nogood-violation* for all (finitely many) combinations of branch labels $\lambda_1, \ldots, \lambda_n$ which violate *NG*, then eventually *NG* will no longer be violated.

  A similar result for the *undo-realizability* procedure in the **else** part will be shown in Section 6.7; in this case, *NG* will turn out to be no longer *realizably* violated.

- For a given most general ngvs ρ, unless some b_i is undone on $\sigma_i\rho$, the nogood remains violated. (This is to say: undoing b_i on a strict instance of $\sigma_i\rho$ is not sufficient.)

  This again follows directly from the definition of ngvs: undoing b_i on a strict instance of $\sigma_i\rho$ cannot "undo" the fact that $\lambda_i \sqsubseteq \sigma_i\rho$, so ρ continues to be a ngvs.

- By the same argument, we see that after we have undone b_n on $\sigma_n\rho$ appropriately, the nogood blocks us from re-branching on b_n over $\sigma_n\rho$ or any other label of which $\sigma_n\rho$ is an instance, as long as the other branches $b_1, \ldots, b_{n-1}$ are left untouched. Only if we undo some of these branches may we re-establish b_n over $\sigma_n\rho$.

Let us formalize this for the maximal branch b_n, of which, according to our definition, *NG* is a term nogood:

Definition 6.5.17 *Let NG be a term nogood on σ_n: b_n which is violated in the set B' obtained by adding λ: b_n to B, with most general nogood-violating substitution ρ. Then we say that NG* blocks *(branching on) b_n over a label λ on an instance $\sigma_n\rho$, with most general nogood-violating substitution ρ. (We do not mention $\sigma_n\rho$ if it is identical with the main label of λ.) If NG is realizably violated in B', we say that NG blocks b_n* realizably *over λ (on $\sigma\rho$).*

In this sense, by introducing the (term) nogood (on b_n), we have reduced the unblocked search space, the set of labels over which b_n can be branched. The blocked search space may only be unblocked when some branch lower than b_n (wrt $\preceq$) is undone; but this will entail a term nogood reducing the search space for that lower branch. We will develop this idea into a completeness proof in Section 6.11. We make an intentional distinction between term nogoods blocking b_n over λ entirely—these prevent branching entirely— and term nogood blocking b_n only on the instance $\sigma_n\rho$, in which case we are still allowed to branch on b_n, provided we add σ_n as an extra exception to λ.

We provide a declaration for a procedure which implements Definition 6.5.17—again without giving actual code. Note that the branch b_n is implicit in the nogood *NG*. As usual, the second procedure (for a term nogood blocking b_n realizably) can be supplied with an optional parameter b for the branch level.

Algorithm 6.5.18 *blocks*

Parameters:

NG: the term nogood to be tested

lambda: a branch label

Returns:

false if *NG* does not block its branch on λ

the most general ngvs ρ, otherwise

Algorithm 6.5.19 *blocks-realizably*

Parameters:

NG: the term nogood to be tested

lambda: a branch label

b: a branch level

Returns:

false if *NG* does not block its branch level-*b*-realizably on λ

the most general ngvs ρ, otherwise

One may object to our approach of "pinpointing" the minimal instance which causes the nogood violation, raising the following concern: b_n may have had strict instances of $\sigma_n\rho$ on which *NG* is not violated, as we mentioned earlier; these instances could have been left

branched upon. For an efficient implementation, we would indeed desire to retain these branch labels. But are we in danger of "overlooking" a possible solution if we undo them? It turns out that the soundness of the algorithm is not jeopardized, because the nogood does not block re-branching on b_n over these strict instances.
We now return to the one type of nogood violation which cannot be undone at all:

Corollary 6.5.20 *Every set of branches B violates an empty nogood.*

Proof: Definition 6.5.7 is trivially satisfied; the substitution ρ in this case is the identical mapping. □

In case of an empty nogood, *repair-nogood-violation* will be called with branch index $b_n = 0$. If the nogood witness is always realizable, an empty nogood is level-0-realizable, in which case *undo-branch* will be called on branch 0, leading to abnormal termination. Corollaries 6.5.20 and 6.5.13 show that this situation correctly characterizes an unsatisfiable set S. We have thus almost completed the soundness proof for our algorithm; we will finish the argument in Section 6.10. Notice that empty nogoods may well arise from running the algorithm on a satisfiable set S, but in that case we can always undo the realizability of the witness σ, so the nogood violation remains benign.

Remark 6.5.21 *Nogood repairs—or any other operation involving only undoing and deleting of branches—cannot cause any nogood violations which did not exist before. To see this, consider a set B' obtained from B by undoing and deleting branches, and assume that ρ violates a nogood NG with premises σ_i: b_i, $i = 1, \ldots, n$ in B'. So the branches b_i in B' have labels λ_i so that $\lambda_i \sqsubseteq \sigma_i\rho$. Proposition 6.4.12 shows that corresponding labels λ_i' on the b_i in B can be found, so that $\lambda_i' \sqsubseteq \sigma_i\rho$. This shows that ρ is a nogood-violating substitution in B as well; therefore, NG has been violated in B.*

This last property guarantees that repair operations for nogoods do not interfere with one another. Once a nogood is repaired, it remains non-violated, until we perform another branching step. This will ensure that our branch repair operations terminate finitely.

6.6 Standing Inconsistencies

In the sections so far, we have encountered various types of inconsistencies which warrant introduction of nogoods and/or branch repair. In order to treat them formally and consistently, we will classify and discuss them, using a number of parameters as follows:

- a complex label λ, the *witness* of the inconsistency,
- a dependency set Δ (the branches and labels which caused the inconsistency)
- the branch b in Δ with the highest index.

These are the types of inconsistencies which occur:

- An assertion $(\sigma_1\sigma - \Sigma)\ \bot$ introduced by a branch b over the label $\sigma_1 - \Sigma$, where $\sigma\ \bot$ is a conjunct in the term $\beta(b)$. Here we have $\Delta = \{(\sigma_1 - \Sigma)\colon b\}$ and $\lambda = \sigma_1\sigma - \Sigma$.
- An occurrence of complementary assertions $\lambda_1\ p$ and $\lambda_2\ \neg p$ introduced by branches b_1 and b_2 over labels λ_1' and λ_2', respectively (possibly even the same branch on the same label), so that λ_1 and λ_2 unify. Here we have $\lambda = (\lambda_1, \lambda_2)$, $\Delta = \{\lambda_1'\colon b_1, \lambda_2'\colon b_2\}$, and $b = \max\{b_1, b_2\}$.
- Any unbranched label λ in the default branch $U(C)$ of a clause C. In the special case when C has not been branched on at all, we have $\lambda = \sigma_0$. In terms of $M(B)$, we could classify unbranched labels among the first type of inconsistency, where not some branch b but the default branch $U(C)$ introduced $\lambda\ \bot$ into $M(B)$. (See our discussion in Section 6.4.) However, unbranched labels are significantly distinct from the other sources of inconsistencies, in that they are "repaired" by branching, whereas the other inconsistencies are repaired by undoing a branch. We declare that the dependency set Δ of an unbranched label is empty and $b = \infty$.
- A violated nogood NG, or more precisely: $NG\rho$, where ρ is the most general nogood-violating substitution. Then λ is any of the nogood witnesses $\sigma\rho$ which satisfies condition (2) in Paradigm 6.5.11, Δ is the set of nogood premises of $NG\rho$, and $b = b_n$ is the maximal branch participating in NG. We already discussed nogood violations and how to repair them.

We will refer to the first and second type above as *standing inconsistencies*. All four types of inconsistencies have in common that they only require repair when their witness is or becomes realizable in $M(B)$; in that case, we will lazily speak of a *realizable inconsistency*. Let us ascertain that standing inconsistencies and unbranched labels are indeed the only inconsistencies on B which may introduce a clash:

Proposition 6.6.1 *If B contains no realizable standing inconsistency, then $M_0(B)$ is clash-free. If additionally B contains no realizable unbranched label, then $M(B)$ is clash-free.*

Proof: We will show that each type of clash which may occur in $M_0(B)$ corresponds to a standing inconsistency, and that λ correctly identifies the clash witness. Since the clash witness is realizable in $M_0(B)$ by definition, it is also realizable in $M(B)$, which shows the proposition via the contrapositive. We do the same for clashes which occur in $M(B)$.
Recall that all assertions on $M_0(B)$ are of the form $(\sigma_1\sigma - \Sigma)\ a$, arising from some "real" branch b (including branch 0) over label $\sigma - \Sigma$. Any clash of the form $(\sigma_1\sigma - \Sigma)\ \bot$ has been correctly identified above by its witness $\lambda = \sigma_1\sigma - \Sigma$. In clashes consisting of two complementary assertions $\lambda_1\ p$ and $\lambda_2\ \neg p$ on $M_0(B)$ where λ_1 and λ_2 unify, $\lambda_1\ p$ and $\lambda_2\ \neg p$ arise from two (possibly identical) branches b_1 and b_2. Again we have correctly identified $\lambda = (\lambda_1, \lambda_2)$ as the clash witness.
For all clashes in $M(B)$ arising from assertions in $M_0(B)$ (whose witness may not be realizable in $M_0(B)$ but in $M(B)$), we argue analogously as above. Other than these, clashes in $M(B)$ can only be introduced through assertions $\lambda\ \bot$ with a realizable unbranched label λ. This label λ has been correctly characterized as the witness in the third type of inconsistency above. Our discussion has thus covered all sources of clashes in $M(B)$ and characterized them as standing inconsistencies and unbranched labels. □
Nogood violations are absent from the list in Proposition 6.6.1. For these, we already argued in the previous section that they do not *introduce* but merely *indicate* a clash.
We will now show how to repair clashes from standing inconsistencies. As one might guess, we accomplish this by undoing some branch from the depencency set Δ, or by undoing witnesses for the realizability of λ. Now since we have already covered these strategies extensively in the last section, we will not perform the repairs directly. Instead, we will derive an appropriate nogood from Δ and λ which will initially be violated by the current set B, so that the correct type of action will be implicitly triggered to not only repair the nogood violation, but also the inconsistency.
For standing inconsistencies, and only for these, we introduce nogoods as soon as the inconsistencies occur, even if their witness is not realizable. The reason is that a standing inconsistency may give rise to a clash at some later point, when other branches introduce constants making the witness λ realizable. We would need to store and monitor a separate list of standing inconsistencies, periodically checking for realizable witnesses. By contrast,

introducing nogoods for standing inconsistencies causes no harm and only little bookkeeping overhead[10]; and the list of nogoods is already actively monitored with regards to witnesses becoming realizable.

The following *assertion introduction rule* characterizes not only the first type of inconsistency, but allows us to infer a nogood-like expression from *any* assertion on B:

$$\frac{}{(\sigma_0\, b \rightarrow \sigma_0\sigma\, a)\rho(* \mapsto *_t, \sigma)}\ ,$$

whereas $\sigma\, a$ is a conjunct on $\beta(b)$, and σ_0 is the main label of the clause $C(b)$.

We easily show this rule correct:

Proposition 6.6.2 *The nogood-like expression NG in the bottom part of the assertion introduction rule satisfies Paradigm 6.5.11.*

Proof: Assume NG is violated; that is, branch b has a label $\sigma_1' - \Sigma$ and let $\rho = \rho(\sigma_0 \mapsto \sigma_1')$ be the most general nogood-violating substitution. Since σ_1' is an instance of σ_0, we have $\sigma_0\rho = \sigma_1'$, and $\sigma_0\sigma\rho = \sigma_1'\sigma$. Furthermore, the requirement $\sigma_1' - \Sigma \sqsubseteq \sigma_0\rho = \sigma_1'$ ensures that $\sigma_1' - \Sigma$ is not a vacuous label. Now according to the side condition of the rule, b introduces the assertion $(\sigma_1'\sigma - \Sigma)\, a$ into $M(B)$. The fact that $\sigma_1'\sigma$ is a main label in $M(B)$ shows the condition (1) of Paradigm 6.5.11; furthermore, we get $\varepsilon\, a \in (M(B))_{\langle\sigma_1'\sigma\rangle}$, which shows condition (3) in case a is a literal, and yields a clash in case $a = \bot$, showing condition (2). □

Corollary 6.6.3 *If $a = \bot$, the above nogood NG is violated by branch b on $\sigma_1' - \Sigma$, iff there is a standing inconsistency with $\Delta = \{\sigma_1' - \Sigma)\colon b\}$, so that all exceptions in Σ are strict instances of σ_1', and witness $\lambda = \sigma_1'\sigma - \Sigma$. Furthermore, NG is realizably violated iff all the above holds and λ is realizable.*

Proof: One direction of the proof follows directly from Proposition 6.6.2. For the other direction, observe that the condition $\sigma_1' - \Sigma \sqsubseteq \sigma_0\rho$ is warranted by our requirement that $\sigma_1' - \Sigma$ not be vacuous. For the second part, a realizable nogood violation always entails the existence of an rgi of the nogood witness. Conversely, let γ be an rgi of λ. Because of this,

[10] One may argue that standing inconsistencies *are* nogoods in and of themselves. (Simply ignore the exceptions in the witness λ and in the labels of the dependency set.) Taking this point of view, we could simply store the inconsistencies alongside with the nogoods or, as we will suggest in Section 6.12, infer them implicitly from the assertions on the branches.

$\rho(\sigma_0 \mapsto \sigma_1')$ and $\rho(\sigma_0\sigma \mapsto \gamma)$ are "compatible" substitutions, so $\rho(\sigma_0\sigma \mapsto \gamma, \sigma_0 \mapsto \sigma_1')$ exists and maps σ_0 to an instance of $\sigma_1' - \Sigma$, namely a prefix of γ. □

This corollary proves that *NG* may rightfully take the place of the standing inconsistency. Whenever a standing inconsistency of type 1 exists, the nogood introduced according to the above rule is violated as long as the inconsistency exists, and realizably violated as long as it constitutes a clash. Conversely, when the nogood is repaired, we are guaranteed that the inconsistency no longer exists.

To cover the second type of inconsistency, consider the following *complementary assertion rule*, in the special case where the two witnesses are identical:

$$\frac{\sigma_1\, b_1 \to \sigma\, p \quad \sigma_2\, b_2 \to \sigma\, \neg p}{(\sigma_1\, b_1, \sigma_2\, b_2 \to \sigma\, \bot)\rho(* \mapsto *_t, \lceil \sigma_1, \sigma_2 \rceil)} \ ,$$

Note that the indexing substitution $\rho(* \mapsto *_t, \lceil \sigma_1, \sigma_2 \rceil)$ ensures that all wildcards in both σ_1 and σ_2, as well as in the corresponding positions in σ, receive the same index. We easily show this rule correct as well:

Proposition 6.6.4 *If the nogood-like expressions NG_1, NG_2 in the top part of the complementary assertion rule satisfy Paradigm 6.5.11, then so does the nogood NG at the bottom.*

Proof: Let $\sigma_1' - \Sigma_1$ and $\sigma_2' - \Sigma_2$ be the labels on b_1 and b_2 which violate *NG*, and $\rho = \rho(\sigma_1 \mapsto \sigma_1', \sigma_2 \mapsto \sigma_2')$ the most general ngvs. Thanks to indexing the $*$, a nogood-violating substitution ρ of *NG* must instantiate corresponding $*_t$ in σ_1 and σ_2 by the same elements. Therefore, we have $\sigma_1\rho(\sigma_1 \mapsto \sigma_1') = \sigma_1\rho(\sigma_1 \mapsto \sigma_1', \sigma_2 \mapsto \sigma_2')$; likewise for σ_2. This shows that ρ is also the most general ngvs for NG_1 and NG_2, so the precondition of Paradigm 6.5.11 is met for *NG*, NG_1 and NG_2, with the same witness $\sigma\rho$. Therefore, condition (1) is shown for *NG* because it holds for NG_1 and NG_2. Now condition (3) for NG_1 and NG_2 shows that $M(B)_{\langle\sigma\rho\rangle}$ contains both $\varepsilon\, p$ and $\varepsilon\neg p$, which constitutes a clash, thus showing condition (2) for *NG*. This establishes that Paradigm 6.5.11 is valid. □

Note that in the case of two complementary assertions resulting from the same branch over the same label, the two premises in the dependency set are identical; it is correct to merge them into one premise.

From the special case, we derive the general *complementary assertion rule*, which reads as follows:

$$\frac{\sigma_1\, b_1 \to \sigma\, p \quad \sigma_2\, b_2 \to \sigma'\, \neg p}{(\lfloor \sigma_1, \sigma' \rfloor\, b_1, \lfloor \sigma, \sigma_2 \rfloor\, b_2 \to (\sigma, \sigma')\, \bot)\rho(* \mapsto *_t, \lceil \sigma_1, \sigma_2 \rceil)} \ ,$$

Proposition 6.6.5 *If the nogood-like expressions NG_1, NG_2 in the top part of the general complementary assertion rule satisfy Paradigm 6.5.11, then so does the nogood NG at the bottom.*

Proof: Apply the nogood instantiation rule from Proposition 6.5.15, using the substitution $\rho(\sigma_1 \mapsto \lfloor \sigma_1, \sigma' \rfloor)$, to NG_1. (This substitution instantiates the constants in (σ, σ') into corresponding positions where σ_1 has $*$.) The resulting nogood is $NG_1' = NG_1\rho(\sigma_1 \mapsto \lfloor \sigma_1, \sigma' \rfloor) = \lfloor \sigma_1, \sigma' \rfloor\ b_1 \rightarrow (\sigma, \sigma')\ p$. We do the same for NG_2 with $\rho(\sigma_2 \mapsto \lfloor \sigma, \sigma_2 \rfloor)$, obtaining NG_2'. Now NG is obtained from NG_1' and NG_2' by applying the special case of the complementary assertion rule, which establishes Paradigm 6.5.11 for NG. □

Corollary 6.6.6 *Let b_1 and b_2 be two branches on clauses with main labels σ_0 and σ_0', and $\sigma\, p$, $\sigma'\, \neg p$ two conjuncts on $\beta(b_1)$ and $\beta(b_2)$, and assume that b_1 and b_2 currently branch over $\sigma_1' - \Sigma_1$ and $\sigma_2' - \Sigma_2$, respectively. Then the nogood $NG = \lfloor \sigma_0, \sigma_0'\sigma' \rfloor$: b_1, $\lfloor \sigma_0', \sigma_0\sigma \rfloor$: $b_2 \rightarrow (\sigma_0\sigma, \sigma_0'\sigma')\ \bot$ is violated by the above branches iff there is a standing inconsistency with $\Delta = \{\sigma_1' - \Sigma_1)$: $b_1, \sigma_2' - \Sigma_2)$: $b_2\}$ (whereas the exceptions are strict instances of their respective main labels) and witness $\lambda = (\lambda_1, \lambda_2)$, whereas $\lambda_1\, p$, $\lambda_2\, \neg p$ are the assertions introduced by b_1 and b_2, respectively. Furthermore, NG is realizably violated iff all the above holds and λ is realizable.*

Proof: The proof is messy but essentially similar to that of Corollary 6.6.3; we will only sketch it here. First, we use the assertion introduction rule for each of the two assertions introduced by b_1 and b_2, respectively, to introduce nogood-like expressions NG_1 and NG_2 on σ_0 and σ_0', respectively. Then Corollary 6.6.3 shows that these expressions are matched (violated) by b_1 on $\sigma_1' - \Sigma_1$ and b_2 on $\sigma_2' - \Sigma_2$, respectively. Using the complementary assertion rule, the expressions can be joined into a nogood, which is of the form given in NG above. Paradigm 6.5.11 holds for NG, as evidenced by Proposition 6.6.5. So if NG is violated with most general ngvs ρ, then $(M(B))_{\langle(\sigma_0\sigma, \sigma_0'\sigma')\rho\rangle}$ contains a clash, which witnesses a standing inconsistency in $M(B)$. The converse and the case of a realizable violation are dealt with similarly as in Corollary 6.6.3. □

Corollaries 6.6.3 and 6.6.6 show how the two types of standing inconsistencies can be represented by nogoods. We showed in either case that the presence of the inconsistency due to some branch labels is equivalent to a nogood violation by these branch labels, and that the presence of a clash is equivalent to a realizable nogood violation. We have thus reduced the

task of repairing inconsistencies to the task of repairing nogoods. We used the main labels σ_0, σ_0' so as to introduce nogoods as general in scope as possible.

Our algorithm for repairing an inconsistency covers both kinds of standing inconsistencies. It assumes that the nogood premises and witness have already been computed (we will provide the code shortly). It only introduces the appropriate nogood and repairs it, provided it is realizably violated.

Algorithm 6.6.7 *repair-standing-inconsistency*

Parameters:

σ: the witness of the inconsistency

$\Delta = \{\sigma\colon b_1, \ldots, \sigma_n\colon b_n\}$: the premises of the inconsistency

begin

sort Δ by ascending branch index $b_1 < \ldots < b_n$

set $\rho := \rho(* \mapsto *_t, \lceil\sigma_1, \ldots, \sigma_n\rceil)$

set $NG = \sigma_1\rho\colon b_1, \ldots, \sigma_n\rho\colon b_n \rightarrow \sigma\rho\ \bot$

Add *NG* as a nogood

if *realizably-violated (NG)*

repair-nogood-violation(NG)

end if

end

The next procedure, *repair-branch*, is called after each branching step. We can observe that *repair-standing-inconsistency* is called on all types of standing inconsistencies, with the correct parameters for a nogood representing the inconsistency:

Algorithm 6.6.8 *repair-branch*

Parameters:

b_1: the index of the branch introduced

$\lambda_1 = (\sigma_1 - \Sigma_1)$ the new branch label

begin

foreach nogood *NG* **do**

if *is-realizably-violated(NG)*

repair-nogood-violation (NG)
end if
done
set σ_0 := the main label of $C(b_1)$
foreach conjunct $\sigma\, a$ in $\beta(b_1)$ **do**
if $a = \bot$
repair-standing-inconsistency($\sigma_0\sigma$, $\{\sigma_0\colon b_1\}$)
else if $a = l$ (some literal)
foreach assertion $(\sigma_2\sigma' - \Sigma_2)\,\bar{l}$ in $M(B)$ **do**
if $(\sigma_1\sigma - \Sigma_1, \sigma_2\sigma' - \Sigma_2)$ exists
set b_2 := the branch which introduced the assertion
set σ_0' := the main label of $C(b_2)$
repair-standing-inconsistency($(\sigma_1\sigma, \sigma_2\sigma')$,
$\{\lfloor\sigma_0, \sigma_0'\sigma'\rfloor\colon b_1, \lfloor\sigma_0', \sigma_0\sigma\rfloor\colon b_2\}$)
end if
done
end if
done
end

Since only finitely many new assertions were introduced through branching, and since there are only finitely many combinations of assertions which can form standing inconsistencies, all standing inconsistencies can be removed in finitely many steps.
The procedure also incorporates the mentioned "nogood monitor". All existing nogoods, of which there can be only finitely many at any given time, are systematically checked for realizable violations, and all violations repaired. We will show in the next section that *repair-nogood-violation* takes only finitely many steps for each[11]. We see that *repair-branch* repairs all types of clashes and nogood violations, so we conclude:

Proposition 6.6.9 *The procedure repair-branch returns after finitely many steps. When it returns, the current branch contains no clashes other than unbranched labels.*

[11] In a real implementation, we may wish to invoke the nogood monitor only sporadically, or use a good filtering algorithm so we do not waste time checking nogoods whose witness cannot have become realizable, using suitable criteria.

Hence the algorithm is ready to proceed to the next step, which is branching on another realizable unbranched label. Or if none remains, the entire algorithm is done. (See Section 6.9.)

6.7 Realizability Propagation

As part of the procedure for repairing a nogood violation with maximal branch index b, we encountered the case where the witness σ of the nogood violation is realizable but not level-b-realizable. In this case, undoing branch b would violate our principle of undoing branches with maximal index first. So we suggested to proceed by undoing branches which contribute realizability witnesses, until $\sigma\rho$ is no longer realizable. But we would also like to uphold another established principle: whenever we undo a branch, we put a nogood in its place, which prohibits us from re-branching on the same label or any of its instances. So we somehow need to combine the violated nogood with the realizability witness, which will result in another nogood corresponding to the branch to undo. We call this step *realizability propagation.*

A different application for realizability propagation will arise from the next section where we merge nogoods due to failed clauses. In lieu of a full explanation, we present an example here: Suppose we have a nogood with witness $*$ which we know to be realizable; can we shorten it into a nogood with witness ε?

We motivate our approach by the following:

Proposition 6.7.1 *We have the following* constant elimination rule*:*

$$\frac{\Delta \rightarrow \Omega, \sigma c \perp}{\Delta \rightarrow \Omega, \sigma \perp} \ .$$

Proof: By Paradigm 6.5.11, $(\sigma c)\rho$ satisfies condition (1) in $M(B)$, and $(M(B))_{\langle(\sigma c)\rho\rangle}$ contains a clash. We obviously have $(\sigma c)\rho = (\sigma\rho)c$, and since this label satisfies condition (1), so does its prefix $\sigma\rho$; furthermore, c exists in $(M(B))_{\langle\sigma\rho\rangle}$ (Proposition 5.4.17), and as we said, $(M(B))_{\langle\hat{\sigma}c\rangle}$ contains a clash for all instances $\hat{\sigma}$ of $\sigma\rho$; then so does $(M(B))_{\langle\hat{\sigma}\rangle}$ (Proposition 5.5.4). We have thus established Paradigm 6.5.11 for the bottom nogood, which establishes the validity of this rule. □

In this fashion, we can remove all trailing existential constants from any nogood witness. But this trick clearly does not work for nogood witnesses of the form $\sigma*$ or $\sigma[c]$. As we should

expect, these positions must be instantiated by existential constants in order to witness a clash. Of course we already know that they are: after all, the nogood is realizably violated. So we just need to find a suitable realizability witness.

The algorithm given below can propagate realizability into our nogood by instantiating any $*$ or conditional constants in the nogood witness; they do not have to be in the trailing position. Its usage will depend on what exactly we would like to do: If our purpose is to shorten the nogood witness, we probably want to shorten the nogood witness from the end. But if we are to make the witness non-realizable in $M(B)$, it does not matter which position we instantiate: instead, we would like to be faithful to our strategy and propagate a realizability witness arising from a branch with maximal index. No matter which one we decide to pick, the realizability witness will be of the form $\sigma'\sigma'' - \Sigma$, a prefix of some assertion $(\sigma'\sigma''\sigma''' - \Sigma)\ a$ arising from some branch b on the label $\sigma' - \Sigma$. As in the rule above, our original nogood is written as: $NG = \Delta \rightarrow \Omega, \sigma\ \bot$. It is safe to assume that $\sigma'\sigma''$ is no longer than σ (any positions beyond that would not instantiate anything in σ). Furthermore, as explained at the end of Section 6.4, we can assume that $|\sigma'| < |\sigma|$, and that some position in σ will be instantiated by some constant from σ''. Given these parameters, the realizability propagation rule is as follows:

$$\frac{\Delta \rightarrow \Omega, \sigma\ \bot \quad \sigma'\colon b \rightarrow \sigma'\sigma''\ a}{\Delta\rho(\sigma \mapsto \sigma'\sigma''), \sigma'\rho(\sigma' \mapsto \lfloor sigma, \sigma' \rfloor)\colon b \rightarrow \Omega, \lceil \sigma, \sigma' \rceil\ \bot}\ .$$

We used $\lfloor \sigma, \sigma' \rfloor$ instead of σ' in order to pinpoint the instance of the realizability witness which actually contributes to the propagation. Note that we need not bother with possible exceptions to σ', since we have chosen this label with the knowledge that a ground instance exists.

The witness $\sigma'\sigma''$ may instantiate more than one position in σ with constants. Usually the nogood premises will also be instantiated. Therefore, if we repeatedly propagate witnesses into a nogood, this nogood will quickly become too specialized and much less useful. Realizability propagation should be used with caution and only where necessary for guaranteeing completeness of the search. Whenever we have a choice between realizability witnesses to propagate, it is usually better to use witnesses with few constants. The "cleanest" witnesses are of the form $*^k c$—they instantiate only the one position they need to, in order to shorten the nogood witness.

Proposition 6.7.2 *The propagated nogood NG' satisfies Paradigm 6.5.11, provided NG does.*

Proof: We argue using the following nogood:

$$\begin{aligned} NG_1 &= NG\rho(\sigma \mapsto \lceil \sigma, \sigma'\sigma'' \rceil) \\ &= \lceil \sigma_1, \sigma'\sigma'' \rceil\colon b_1, \ldots, \lceil \sigma_n, \sigma'\sigma'' \rceil\colon b_n, \lfloor \sigma, \sigma' \rfloor\colon b \rightarrow \lceil \sigma, \sigma'\sigma'' \rceil \perp. \end{aligned}$$

Let $b_1, \ldots, b_n, b$ be a set of branches on B with labels $\lambda_j = \tilde{\sigma}_j - \Sigma_j$, $j = 1, \ldots, n$ and $\tilde{\lambda} = \tilde{\sigma} - \tilde{\Sigma}$, violating NG_1. Then B, by virtue of $b_1, \ldots, b_n$, also violates NG, and the nogood-violating substitution ρ for NG_1 is a concatenation of a nogood-violating substitution ρ_0 for NG_0 and the substitution $\rho(\sigma' \mapsto \tilde{\sigma})$. Since the instantiations $\tilde{\sigma}_j$ instantiate $\lceil \sigma_j, \sigma'\sigma'' \rceil$, so do the $\sigma_j\rho_0$. By 6.5.11, we conclude that $\sigma\rho_0$ is scwr. Since $\lceil \sigma_j, \sigma'\sigma'' \rceil\rho$ always instantiates $\sigma\rho_0$, we can conclude that $(M(B))_{\langle\hat{\sigma}\rangle}$ contains a clash for every instance of $\lceil \sigma_j, \sigma'\sigma'' \rceil\rho$. However, $\sigma\rho_0$ may not instantiate $\lceil \sigma, \sigma'\sigma'' \rceil$, so it does not by itself show that $\lceil \sigma_j, \sigma'\sigma'' \rceil\rho$ is scwr. Instead, we observe that $\lceil \sigma, \sigma'\sigma'' \rceil\rho$ is a concatenation of $\lceil \sigma, \sigma'\sigma'' \rceil\rho_0$ with a substitution replacing the first $|\sigma'|$ positions of σ with $\tilde{\sigma}$, an instance of $\lfloor \sigma', \sigma$. But if we change the order of concatenation and do this substitution first, we obtain $\lceil \sigma, \sigma'\sigma'' \rceil\rho = \lceil \sigma, \tilde{\sigma}\sigma'' \rceil\rho_0 = \lceil \sigma\rho_0, \tilde{\sigma}\sigma'' \rceil$. Now we see that both $\sigma\rho_0$ (thanks to 6.5.11 and $\tilde{\sigma}\sigma''$ (because $\tilde{\sigma}$ exists as a branch label in $M(B)$) are scwr in $M(B)$. Therefore, $\lceil \sigma, \sigma'\sigma'' \rceil\rho$ is scwr in $M(B)$, and Paradigm 6.5.11 is verified for the nogood NG_1. From this, the paradigm also follows for the nogood NG' with shortened witness, because of Proposition 6.7.1. □

The corresponding algorithm is very simple:

Algorithm 6.7.3 *propagate-realizability*

Parameters:

$NG = \sigma_1\colon b_1, \ldots, \sigma_n\colon b_n \rightarrow \Omega, \sigma \perp$: the nogood

$\sigma'\sigma''$: the label to be propagated

b: the branch which introduced the label

return a nogood

$NG' = \lceil \sigma_1, \sigma'\sigma'' \rceil\colon b_1, \ldots, \lceil \sigma_n, \sigma'\sigma'' \rceil\colon b_n, \lfloor \sigma, \sigma' \rfloor\colon b \rightarrow \Omega, \lceil \sigma, \sigma'\sigma'' \rceil \perp.$

obtained by deleting any number

of trailing existential constants from $\lceil \sigma, \sigma'\sigma'' \rceil$.

The next procedure, *undo-realizability*, has already been mentioned. It is used for undoing all branches which contribute realizability witnesses for a nogood violation. Since undoing branches must always be accompanied by introducing appropriate nogoods, the original nogood is given as a parameter, and realizability-propagated nogoods are derived from it:

Algorithm 6.7.4 *undo-realizability*

Parameters:

NG : a nogood

while (ρ, γ, Δ) = *realizably-violated*(NG,B) is not *false* **do**

select $(\sigma'\sigma'' - \Sigma)$: $b \in \Delta$ so that b is maximal

set $\sigma'\sigma_p$:= a minimal prefix of $\sigma'\sigma''$ which ends in a witnessing constant

set NG' := *propagate-realizability* (NG, $\sigma'\sigma_p$, b)

undo-branch $(b, \lfloor \sigma, \sigma' \rfloor)$

done

It is most efficient if the *is-realizable* function returns sets of witnesses whose maximal branch index b is as small as possible. This witness may be part of several other sets of realizability witnesses with higher index b'; if branch b has already been undone (it must be undone anyway), all these sets are simultaneously invalidated, which minimizes the number of realizability propagations needed. Similarly, the choice of a "shortest witnessing prefix" in the algorithm above ensures that the propagated nogood remains as general as possible, while still invalidating the set Δ.

In order to repair the propagated nogood, we call *undo-branch* instead of *repair-nogood-violation*. We already know the nogood to be level-b-realizably violated, as b was the maximal index among the realizability witnesses), so *repair-nogood-violation* would go into the **if** part, where it would call *undo-branch* with the same parameters as we do here. In particular, the nogood violation is properly undone. We also point out that *undo-realizability* may have been called from inside *repair-nogood-violation*, so avoiding calling it again keeps our procedure recursion-free, a favourable feature.

When a propagated nogood is introduced, it is initially violated because the original nogood NG is realizably violated. This means that the original nogood NG is violated because of the labels on $b_1, \ldots, b_n$, and the additional branch b is scoped over a label σ' so that the unifier $\lceil \sigma, \sigma'\sigma'' \rceil$ exists; but the witness of NG' is the same label, possibly shortened, which shows that NG' is also violated.

Now in response to the violation of NG', the branch b is undone on the label $\lfloor \sigma, \sigma' \rfloor$ over which it is scoped in NG', so the propagated nogood is no longer violated. In this way, none of the new nogoods created in *undo-realizability* is violated when the procedure terminates. Furthermore, branch b on label σ' no longer functions as a witness for the realizability of NG, because it now has $\lfloor \sigma, \sigma' \rfloor$ as an exception. In this way, every witness for the realizability of the violation of NG is "deactivated". Since finitely many branches allow us to state only finitely many sets of witnesses, and since the procedures do not introduce new branches, we conclude:

Corollary 6.7.5 *The* **while** *loop in undo-realizability terminates after finitely many realizability propagations. When the loop terminates, the original nogood NG is no longer realizably violated.*

6.8 Failed Clauses and Nogood Merging

The ability to merge several nogoods into a new nogood is not only an essential technique to ensure completeness of nogood reasoning; but we also need it in order to handle failed clauses correctly. When term nogoods on every branch b in a clause C prevent us from branching over a realizable label σ, it is the premises on all term nogoods combined which prevent us from branching on any of the b over σ. Therefore, we should merge all the premises, minus those mentioning any of the branches b themselves, into a new nogood; in response to this nogood, we can undo one of its premises, which will make one of the term nogoods on the branches in C ineffective: it no longer blocks the branch from being chosen. In the propositional case, nogood merging is easy: take the dependency set Δ_j of the term nogood on each branch b_j, except for b_j itself, and take the union of these dependency sets for all b_j on C. In $\mathcal{LBL}$, however, we encounter the difficulty that different nogoods may have different witnesses. If we merge two nogoods with witnesses $*1$ and $*2$, say, then does the violation of the merged nogood entail a clash in $(M(B))_{*1}$ or in $(M(B))_{*2}$? Since we cannot tell from the merged nogood itself, we must keep both options. This is why nogoods may have a *set* of witnesses. So in the above example, the nogood conclusions are $\Omega = \{*1, *2\}$. If the nogood is violated and both $*1$ and $*2$ are realizable, then the nogood is realizably violated, and a clash will always occur.

In the previous section, we have learned a technique for instantiating and shortening nogood witnesses. We could apply it when the sets of witnesses get too large. If two witnesses

become identical due to instantiations, they can be written as one witness; we can thus reduce the number of witnesses.

One might wonder why we do not always preprocess all term nogoods and only merge them when their witness is ε? In doing so, we would forfeit the advantage of reasoning jointly over sets of instances by using nogoods with wildcards. The more unnecessary instantiations we create, the more nogoods and calls of *undo-branch* we will incur; in fact, we may well end up reasoning over ground labels again. So we must strive to do as little preprocessing as absolutely necessary. In our algorithm, we refrain from realizability propagation for this purpose.

Before we formally state the algorithm, let us briefly outline the steps:

1. Only nogoods which prohibit branching on clause C participate in nogood merging. But these are exactly the term nogoods. In other words, for each branch b_j on term β_j there exists a term nogood NG_j which prohibits branching on b_j over σ. We write these nogoods as Δ_j, σ_j: $b_j \rightarrow \Omega_j$, $j = 1, \ldots, m$.

2. To be true to our nogood paradigm, we must pinpoint the scope of nogoods as precisely as possible. So we apply the nogood substitution $\rho_j = \rho(\sigma_j \mapsto \sigma)$ to NG_j, where σ is the mgu of all branch labels.

3. If any σ_j among the nogood witnesses is not level-b_j-realizable, then we should undo realizability witnesses for NG_j rather than merge the nogoods; for if σ_j is no longer realizable, we can branch on b_j over σ in the regular way; this will violate NG_j, but not realizably.

4. Now simply and take the union of all Δ_j as the premises, and the union of all nogood conclusions, plus the main label σ_0 of C, as the conclusion of the merged nogood.

Steps 1–3 above could be seen as preprocessing steps for each nogood. In fact, they are best performed as part of the *satisfy-unbranched-label* procedure. The nogood instantiations in Step 2 must be computed in order to find whether the term nogoods are violated in the first place; if some b_j does not realizably violate any term nogood on σ, we can simply branch on it. If b_j does violate some term nogood NG_j, we would verify if it is level-b_j-realizably violated. If not, we will still eventually perform branching on σ. Only when we have found term nogoods for all m terms whose witnesses are level-$b_{\max}$-realizable do we start the nogood merging procedure, which thus covers only steps 4 and 5 above:

Algorithm 6.8.1 *merge-nogood*

Parameters:

C: the clause, with main label σ_0

$NG_j = \Delta_j, \sigma_j$: $b_j \rightarrow \Omega_j$, $j = 1, \ldots, m$ the nogood instantiations

begin

set $\sigma := (\sigma_1, \ldots, \sigma_m)$

set $\rho_j = \rho(\sigma_j \mapsto \sigma)$, $j = 1, \ldots, m$

set $\rho' = \rho(* \mapsto *_t)$

set $NG_0 = (\Delta_1\rho_1\rho', \ldots, \Delta_m\rho_m\rho' \rightarrow \Omega_1, \ldots, \Omega_m, \sigma_0\rho' \perp$

add NG_0 as a new nogood

repair-nogood-violation (NG_0)

end

Proposition 6.8.2 *Paradigm 6.5.11 holds for NG_0, provided it holds for all nogoods NG_j.*

Proof: Consider a set B which violates NG_0, and take $\tilde{\sigma} = \lfloor \sigma, \sigma_0 \rfloor$, which is a label on which the clause C fails. Note that $\sigma \ {}_p\sqsubseteq \sigma_0$, so $\tilde{\sigma}$ is a prefix of σ. If $\tilde{\sigma} \in U(C)$, then $\tilde{\sigma}$ is scwr in $M(B)$ and $(M(B))_{\langle\tilde{\sigma}\rangle}$ is utopian; then this is trivially true for any longer label including σ, so Condition (1) of Paradigm 6.5.11 is satisfied for σ and $(M(B))_{\langle\sigma\rangle}$ contains a clash, showing Paradigm 6.5.11. Otherwise, $\tilde{\sigma}$ must instantiate a branch label in one of the b_j. But then the nogood NG_j is violated; let ρ be the nogood-violating substitution obtained from the branch labels of $b_{j,1}, \ldots, b_{j,n_j}$ and b_j participating in the nogood. Since NG_j satisfies Paradigm 6.5.11, $(M(B))_{\langle\hat{\sigma}\rangle}$ contains a clash for all instances $\hat{\sigma}$ of $\sigma_j\rho$; but this includes all instances of $(\sigma_1, \ldots, \sigma_m)\rho$. Finally, $(\sigma_j)\rho$ is scwr, and this argument can be repeated for all $j = 1, \ldots, m$, which shows that $(\sigma_1, \ldots, \sigma_m)\rho$ is also scwr. We have thus verified Paradigm 6.5.11 for the nogood NG_0. □

Since all premises of NG_0 are branched upon when NG_0 is introduced, we also obtain:

Corollary 6.8.3 *At the time nogood NG_0 is introduced, it is realizably violated.*

Accordingly, the procedure *repair-nogood-violation* will undo the branch with maximum index among the nogood premises on NG_0, after which one original nogood NG_j will no longer be violated, which will allow us to branch on b_j again. If all Δ_j are empty, however, we have obtained an empty nogood, in which case there is no branch we can undo. Unless this empty nogood is not realizably violated, we have found the problem unsatisfiable.

6.9 The Main Loop

We are now ready to specify the main routine of the algorithm for finding a model for a given set S of clauses in LCNF:

Algorithm 6.9.1 *model-finder*

Parameters:

S: a set of clauses

Returns:

B: a set of branches representing a satisfying model, or *unsatisfiable*

begin

branching (0,ε)

foreach clause $C = \sigma_0\ F$ **do**

set $U(C) := \sigma_0$

done

while (some $U(C)$ contains a label λ

and *is-realizable*(λ)) **do**

set *satisfy-unbranched-label* (C, σ)

done

return B

end

The first four lines are there to initialize branch 0 and the default branches $U(C)$. The **while** loop, with the function *satisfy-unbranched-label* inside, is the "engine" of the algorithm; it iterates as long as $U(C)$ contains realizable unbranched labels which need to be branched upon (indicating that $M(B)$ contains a clash which needs repair). As we already discussed in the previous section, we should do one of three things with an unbranched label λ, in descending priority:

1. If there exists a branch b_j on which no term nogood realizably blocks branching on λ (entirely), then branch on b_j over λ. This could be viewed as undoing an instance of a default branch with index ∞.

2. If every branch b_j is realizably but not level-b_j-realizably blocked by some term nogoods, undo realizability witnesses until the violation is no longer realizable, then

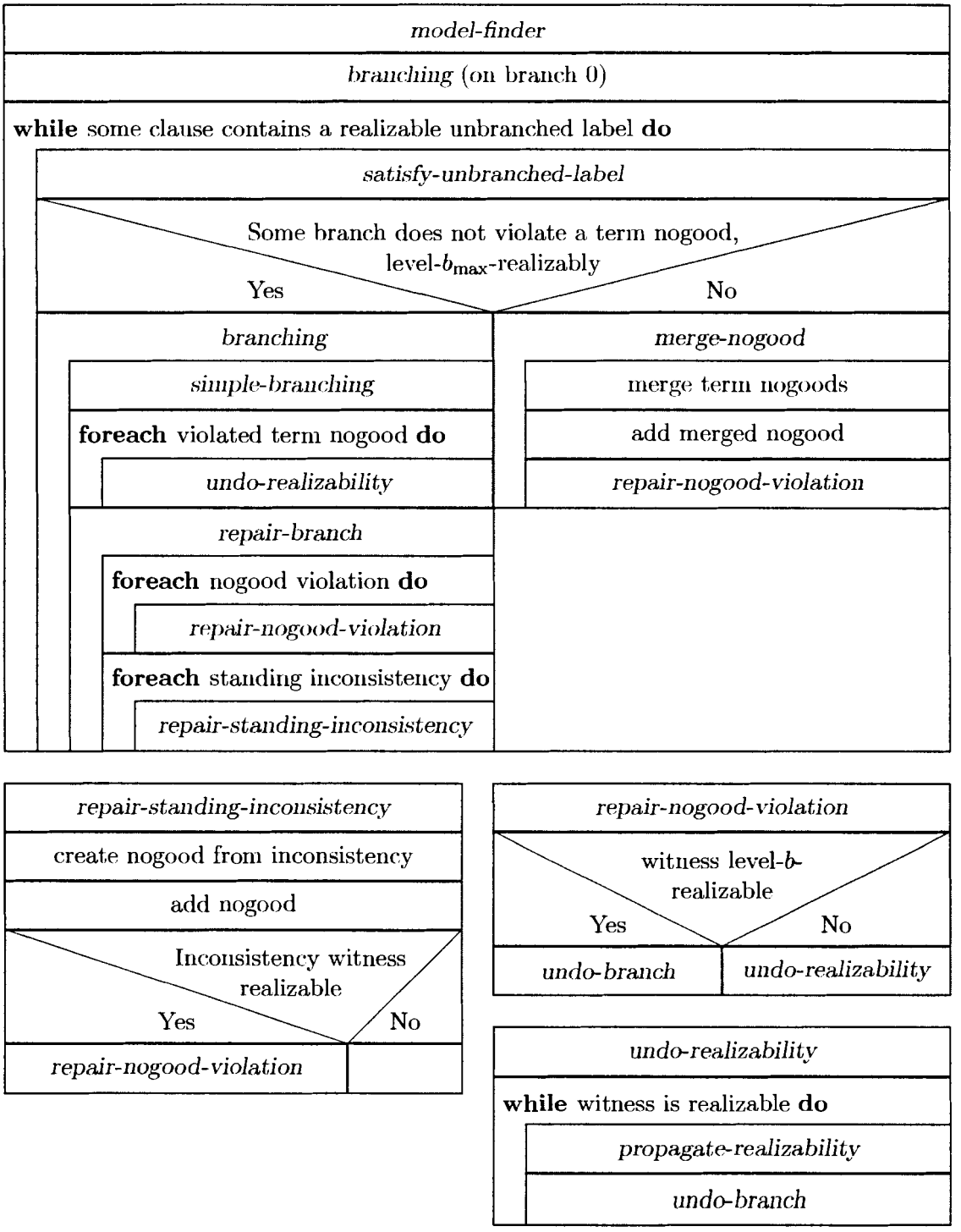

Figure 6.1: *A Nassi-Shneiderman [79] flow diagram of the algorithm. For conciseness, most procedures are shown inside the main procedure and indicated by the procedure name typeset in slanted. Some minor procedures have been omitted for clarity.*

branch on b_j over λ as in 1. Notice that all branches undone have indices greater than $b_{\max}$.

3. Otherwise the clause C fails; merge the term nogoods into a new nogood. This nogood will be violated, and repairing it entails undoing some branch with index less than or equal to $b_{\max}$, the maximum branch index on C.

We see that this prioritization matches our strategy of undoing branches with maximal index first. Note that we need not adhere to any such strategy for picking a branch, e.g. in ascending order. We are free to pick any branch that is not blocked by a term nogood, possibly according to some suitable heuristic. Here is the code for the above steps:

Algorithm 6.9.2 *satisfy-unbranched-label*

Parameters:

$C = \sigma_0\,(\beta_1 \vee \ldots \vee \beta_m)$: a clause

$\lambda = \sigma - \Sigma$: an unbranched label from $U(C)$

begin

set b_j := the branch index of β_j in C, for $j = 1, \ldots, m$

set $b_{\max} := \max\{b_j : j = 1, \ldots, m\}$

foreach $j = 1, \ldots, m$ **do**

if (there ex. a term nogood NG on b_j

s.th. ρ = *blocks-realizably* (NG, λ, b_j) is not *false*

and $\sigma\rho = \sigma$

set $NG_j := NG\rho$

else

branching (b_j,λ)

return

end if

done

merge-nogood (C, $(NG_j)_{(j=1,\ldots,m)}$)

end

Steps 1 and 2 of our informal description are covered by the **else** part, where we call *branching*, then the procedure returns prematurely. Undoing of realizability witnesses,

which is the additional part in Step 2 versus Step 1, is performed inside the *branching* function below. Otherwise we perform Step 3, which just consists of calling *merge-nogood* in the last line. The preprocessing steps for each term nogood, mentioned in the previous section, have already been performed in the **if** part, while iterating over the branches on C. Finally, here is the *branching* function:

Algorithm 6.9.3 *branching*

Parameters:

b: the branch

$\lambda = \sigma - \Sigma$: the label branched upon

begin

simple-branching (b,λ)

foreach term nogood *NG* **do**

if ρ = *blocks-realizably* (*NG*, λ, b_j) is not *false*

and $\sigma\rho = \sigma$

undo-realizability (*NG*)

end IF

if ρ = *blocks* (*NG*, λ) is not *false*

add $\sigma\rho$ as an exception to λ

add *sigma*ρ to $U(C(b))$

end if

done

add λ to $B[b]$

repair-branch (b,λ)

end

Let us discuss the various direct and side effects of branching, and how these get repaired through *branching* and *repair-branch*:

- *branching* is called only when it is not level-b-realizably blocked by a term nogood, which ensures that branching will succeed at least on the main label σ of λ.
- However, some term nogoods may still realizably (but not level-b-realizably) block branching on λ *entirely*, which is not acceptable. We resolve this by undoing some

realizability witnesses. (Since we have already branched on b, *undo-realizability* can handle this case just like an ordinary nogood violation.) These term nogoods are identified by the call to *blocks-realizably*.

- Branching on b over λ may still be blocked (even level-b-realizably so) on *strict instances* of λ. To reduce the need for branch repairs, we "undo" them immediately by introducing them as new exceptions into λ, and writing them back to $U(C)$. As a result, no more term nogoods realizably block the chosen branch b. These term nogoods are identified by the call to *blocks*.

- Other nogoods *NG* which are not term nogoods of b may be violated by branching on b over λ. But the way of repairing these violations is not to prohibit branching on b, but to undo the branch in the maximal nogood premise on b', which is distinct from the currently chosen branch. The "nogood monitor" in *repair-branch* provides for repairs of all such nogood violations.

- Previously harmless, non-realizable nogood violations may become realizable due to new labels introduced by b. In this case, b forms one of the realizability witnesses. Depending on whether b is larger or smaller than the maximal branch index in the dependency set of the nogood, some branch with index larger than b or b itself may be undone—but the latter occurs only after a new realizability-propagated nogood blocks b from being chosen again. These nogood violations are also recognized by the "nogood monitor" in *repair-branch*.

- Standing inconsistencies introduced by the branch will also be repaired by the function *repair-branch*, as we have already seen.

Proposition 6.9.4 : *The procedure satisfy-unbranched-label terminates finitely.*

Proof: We have already seen that the procedures *merge-nogood* and *repair-branch* terminate finitely. Each is called only once from within *satisfy-unbranched-label* and *branching*, respectively, and apart from these, the procedure *satisfy-unbranched-label* may call *undo-realizability* up to m times, each of which also terminates finitely. □

Proposition 6.9.5 : *After satisfy-unbranched-label terminates, B does not contain any standing inconsistencies or realizably violate any nogoods.*

Proof: With one exception, standing inconsistencies and nogood violations are introduced as aresult of branching over a new label. But the *branching* procedure always calls *repair-branch*. We just discussed the types of standing inconsistencies and realizable nogood violations which may arise due to branching, and showed that they are either averted by introducing extra exceptions to λ, or repaired through *repair-branch*. Outside the *branching* procedure, the only other source of realizable nogood violations is inside *merge-nogood*, but we showed that merged nogoods are repaired immediately therein. immediately afterwards. □

6.10 Soundness and Correctness

If the main **while** loop condition in Algorithm 6.9.1 no longer applies, the set $M(B)$ is returned as a model for S. However, *repair-nogood-violation* may encounter an empty nogood which cannot be repaired, in which case the model finder exits and returns *unsatisfiable*. These are the two ways in which the algorithm may terminate. We will now show that whenever it terminates, the answer is correct. That is, if the algorithm returns a set B, then $M(B)$ is indeed a model for S, and if it returns *unsatisfiable*, then S is indeed unsatisfiable. The latter is referred to as *soundness*.

Our first goal is to show that $M(B)$ verifies every clause in S at all times. (This may sound surprising, but we remind the reader that $M(B)$ usually contains clashes, e.g. due to the presence of realizable unbranched labels.) We prove this for any arbitrary fixed clause C, by successively showing a number of invariants which are preserved through each iteration of the **while** loop. But in order to do this, we need to somehow trace the order in which labels have been branched on, unbranched on, and branched on again. For this purpose, we define a counter k, initialize it to 0, let σ_0 be the outer label of C, and increment k. During the algorithm, new main labels on branches in C come into existence in the following contexts:

- on $U(C)$, in *undo-branch*;
- on $U(C)$, in *branching*;
- on b_0, in *merge-nogood*.

We index each new main label as σ_k and increment k. Note that newly introduced labels

do not have exceptions, but they may acquire some later. If another simple label[12] $\sigma_{k'}$ identical to the new label already exists on the branch, re-index it as σ_k. It is evident that only finitely many main labels exist in the branches on a clause at any given time.

Proposition 6.10.1 *For any clause $C = \sigma_0\ F$, the following invariants hold before and after each iteration of the* **while** *loop (understand "branch on C" as including the default branch $U(C)$):*

(1) *σ_0 is the main label of a branch on C.*

(2) *Every exception ξ to a branch label on C with main label $\sigma_{k'}$ instantiates a label $\lambda = \sigma_k - \Sigma$ of another branch on C so that $k > k'$.*

(3) *$M(B) \models_s \sigma\ F$ for every label $\sigma - \Sigma$ on every branch in C.*

(4) *$M(B) \models_s C$.*

Proof: The first invariant holds initially, as $\sigma_0 \in U(C)$. If at some point the algorithm branches on clause C, it takes the entire label σ_0 and makes it the main label of some branch on a term in C, so the invariant remains. In a call of *undo-branch*, it is possible that the branch gets completely undone over σ_0 on some "real" branch as well, but in that case σ_0 returns to the default branch $U(C)$, and the cycle repeats.
The second invariant holds initially because no branch on C has any exceptions. Now we systematically consider all possible "attacks" on the invariant due to changes in main labels and exceptions anywhere in the algorithm:

- If main labels merely get swapped between $U(C)$ and a "real" branch, for instance in *branching*, their index does not change.
- If simple labels get reintroduced and hence re-indexed as explained above, their index only increases. So if ξ instantiates $\sigma_{k''}$, it continues to instantiate the re-indexed label σ_k, and $k > k'' > k'$, so the invariant holds unchanged.

[12]This also works if $\sigma_{k'}$ is the main label of a complex label on the same branch, in which case we can remove all exceptions from $\sigma_{k'}$ and re-index it as σ_k. This would constitute a viable but undue modification of our algorithm we will refrain from undertaking at this point in our discussion.

- Calling *undo-branch* on a "real" branch b over some label σ may introduce new exceptions $(\sigma_{k'}, \sigma)$ on a branch label $\sigma_{k'} - \Sigma$. But immediately afterwards, σ will be introduced into $U(C)$ as a new main label σ_k, and $(\sigma_{k'}, \sigma)$ instantiates σ_k, whereas $k > k'$, and the invariant holds for these new exceptions as well.

- In *undo-branch*, the entire label $\sigma_{k'} - \Sigma$ may get undone because $\sigma \sqsubseteq \sigma_{k'}$. But in that case, ξ too instantiates σ by transitivity of $\sqsubseteq$, and as before, σ is assigned a brand-new index $k > k'' > k'$, which shows that the invariant is defended.

- In *undo-branch*, a new exception $(\sigma_{k'}, \sigma)$ may result in ξ no longer being an instance of λ. But this can only happen because $(\sigma_{k'}, \sigma) \sqsubseteq \xi$. Again by transitivity, we show that ξ instantiates the new label σ_k.

- New exceptions are also introduced in *branching*. But since immediately afterwards the same labels are introduced into $U(C)$ where they get indexed σ_k, they too satisfy the invariant.

For the third invariant, consider a label $\sigma - \Sigma$ on some branch in C. As an induction hypothesis, we assume the invariant shown for all labels $\sigma = \sigma_{k''}$, $k'' > k'$. Notice that the newest main label σ_k cannot have any exceptions, which would violate (2). By Theorem 5.3.9, part (4), the induction hypothesis implies $(M(B))_{\langle\sigma'\rangle} \models_s F$ for all instances σ' of $\sigma_{k''}$. Now let $\sigma = \sigma_{k'}$, and take an arbitrary instance σ' of σ. If σ' instantiates an exception $\xi \in \Sigma$, then because of (2) it also instantiates some main label $\sigma_{k''}$, $k'' > k'$. By the induction hypothesis, $(M(B))_{\langle\sigma'\rangle} \models_s F$. Otherwise we distinguish two cases: first, if $\sigma - \Sigma \in U(C)$, then $M(B)$ contains $(\sigma - \Sigma)\,\bot$. With $M' = \{(\sigma - \Sigma)\,\bot\}$, $M'_{\sigma'}$ is utopian and trivially verifies F. Secondly, if $\sigma - \Sigma$ is a label of a "regular" branch b on C, then Proposition 6.4.2 shows that $(M_0(B))_{\sigma'} \models_s \beta(b)$, whence we get $(M_0(B))_{\sigma'} \models_s F$. In either case, we call upon the monotonicity principle to conclude $(M(B))_{\langle\sigma'\rangle} \models_s F$. We have shown this for all instances σ' of σ, so Theorem 5.3.9, part (4) in reverse entails that $M(B) \models_s [\sigma]\, F$. Finally, σ was required to be the main label of some branch label, so it is a prefix of some main label in $M(B)$, which guarantees it to be scwr in $M(B)$. This shows $M(B) \models_s \sigma\, F$ for $\sigma = \sigma_{k'}$. The same follows for *any* branch label $\sigma - \Sigma$ by induction on k'.

The fourth invariant follows directly from the first and third: The outer label σ_0 of C occurs as the main label on some branch on C, and the third invariant shows $M(B) \models_s \sigma_0\, F$ as desired. □

Since Proposition 6.10.1 holds for every clause in S, we get:

Corollary 6.10.2 *Before and after every iteration of the* **while** *loop in model-finder, wh invariantly have* $M(B) \models\!\!\!\sim_s S$ *for the current set of branches* B.

This is rather intriguing: We can invariably ensure that $M(B)$ verifies S! We accomplished this by "bridging the gaps": for every unbranched label σ in every clause, we introduced $\sigma \perp$ into $M(B)$, forcing the respective clause to be verified over this instance, until we get around to finding a proper branch for σ. Usually these σ are realizable, which constitutes a clash; this is why the **while** loop iterates and branches on realizable unbranched labels. In fact, their non-existence is characteristic for clash-freeness of $M(B)$:

Corollary 6.10.3 *If the algorithm terminates and returns a set* B, *then* $M(B) \models_s S$.

Proof: After each iteration of the **while** loop, we have $M(B) \models\!\!\!\sim_s S$ according to Corollary 6.10.2. Another invariant at that point, shown in Proposition 6.9.5, is that B contains no realizable standing inconsistencies. Finally, the condition for normal termination of the **while** loop is that B not contain realizable unbranched labels. But as we found in Proposition 6.6.1, in the absence of realizable standing inconsistencies and unbranched labels the set $M(B)$ is clash-free. So when the **while** loop terminates and returns B, then $M(B)$ is a model for S. □

Conversely, abnormal termination in *undo-branch* is characteristic for unsatisfiability, as we will now show in analogy to Corollary 6.5.20:

Proposition 6.10.4 *If an empty nogood exists and its witness is level-0-realizable, then* S *is unsatisfiable.*

Proof: We apply Corollary 6.5.14 with $b = 0$, since the maximum branch index in an empty nogood is 0. Thus every set B' which agrees with the current set B on branch 0 contains a clash. But all sets of branches agree on branch 0, so none of them can be a model. □

Corollary 6.10.5 *If the algorithm returns unsatisfiable, then* S *is unsatisfiable.*

Proof: The algorithm exits iff *undo-branch*(see Algorithm 6.4.10) is called with parameter $b = 0$. In the calling function *repair-nogood-violation*—the only function which can call

sl undo-branch with parameter $b = 0$—we see that this can happen only if $b = 0$ is the maximal branch in the nogood NG violated (which means, NG is the empty nogood) and the nogood witness is level-0-realizable. And *repair-nogood-violation* was called in response to the new nogood NG on B. So Proposition 6.10.4 applies, showing that S is unsatisfiable. □

Corollary 6.10.5 shows that the algorithm is sound. Let us now verify that it always terminates.

6.11 Termination and Completeness

In the previous section, we showed how our two termination criteria indicate the satisfiability and unsatisfiability of S, respectively. Now we need to address an important question. We have committed ourselves to deriving models of the form $M(B)$, where B is a set of branches. But the models of a satisfiable set S need not all be of this form. Can we be certain that at least *one* set B exists so that $M(B)$ is a model for S?

Theorem 6.11.1 *If a set S of formulas in LCNF has a strong model M, then there exists a set of branches B so that $M(B)$ is also a strong model for S.*

Proof: Since M is a model, it cannot be utopian, so it must satisfy all clauses in S. We now proceed clause by clause, iteratively building the set B, and with it the model $M(B)$, using the following indirect technique:

- Guided by the existing model M, continue to choose suitable branches and add their assertions to M, *without introducing clashes*, until we arrive at a set B of branches and a set of assertions $M(B) \cup M$. Being a superset of M, this set also satisfies S.
- Since $M(B) \cup M$ is clash-free, so is the subset $M(B)$. Corollary 6.10.2 shows that $M(B)$ satisfies S, provided B has been constructed in accordance with the invariants of Proposition 6.10.1; once we have verified this, $M(B) \models_s S$ follows.

So we will first construct the set B from the model M, then prove the invariants for $M(B)$, and finally show that no clashes were introduced. For the first task, we employ a proof technique resembling that of Proposition 6.2.10. Let $C = \sigma_0\,(\beta_1 \vee \ldots \vee \beta_m)$ be any clause in S, and $b_1, \ldots, b_m$ the corresponding branch indices. Since $M \not\models_s C$, σ_0 must be scwr in

M; we remember this fact for later. Furthermore, using part (3) of Theorem 5.3.9, we have $M_{\langle\hat{\sigma}\rangle} \models_s \beta_1 \vee \ldots \vee \beta_m$ for all egi $\hat{\sigma}$ of σ_0 in M. We wish to use these $\hat{\sigma}$ as branch labels in some b_j so that $M_{\langle\hat{\sigma}\rangle} \models_s \beta_j$. However, a complication arises in that we cannot guarantee all egi in Σ_j^+ to be strongly constant-wise realizable, so we may not freely use them as main labels in branches. (We presented one such case in Example 5.4.4, where an egi 13 featured a constant which was never used as a main label. We said that there is no practical use for sets of this form, but they could not easily be excluded from our theory.) But we can easily fix this problem, by reverting to $[\hat{\sigma}]$ (which is always scwr) whenever $\hat{\sigma}$ is not scwr.
So for each $j = 1, \ldots, m$, we classify each egi $\hat{\sigma}$ (or $[\hat{\sigma}]$, if the former is not scwr) into the set Σ_j^+ if $M_{\langle\hat{\sigma}\rangle} \models_s \beta_j$, and into Σ_j^- otherwise. It follows that every egi is classified into at least one of the Σ_j^+. Notice that whenever $M_{\langle\hat{\sigma}\rangle}$ is utopian, it trivially satisfies *all* disjuncts, and $\hat{\sigma}$ cannot be realizable in M. Now we define labels for the branches b_j as follows: for each egi $\hat{\sigma}$ in Σ_j^+, add the label $\hat{\sigma} - \{\xi \in \Sigma_j^- : \hat{\sigma} \sqsubseteq \xi\}$ to $\Lambda(b_j)$. Repeat this for all terms in all clauses, and let B be the set of branches thus obtained. We claim that $M \cup M(B)$ is clash-free and B obeys the invariants of Proposition 6.10.1 and Definition 6.4.1, just as a set of branches returned by the algorithm would do. Let us consider these first, for any clause C in S:

- All exceptions on the branch labels are *strict* instances of the respective main label.

  This follows because no label can be in Σ_j^+ and Σ_j^- at the same time. Therefore, none of the exceptions $\xi \in \Sigma_j^-$ above can be identical to $\hat{\sigma}$.

- σ_0 is the main label of a branch on some term in F.

  The label σ_0 is an egi of itself, and it is scwr, so just like any other egi, it exists in some Σ_j^+, where it functions as the main label of some branch label on β_j.

- Every exception ξ to a branch label $\sigma_{k'}$ on C instantiates a label $\lambda = \sigma_k - \Sigma$ of another branch on C, so that $k > k'$. Here we are free to choose any suitable indexing scheme for the labels on B, so long as we can prove this invariant. We decide to index any given label σ with the number k of constants (existential or conditional) in it. Therefore, the condition $k > k'$ holds if a label σ_k can be found which strictly instantiates $\sigma_{k'}$.

  All exceptions ξ on every branch are egi of the outer label σ_0 of C, the clause to which the branch belongs, and we know that every egi satisfies $M_{\langle\xi\rangle} \models_s \beta_{j'}$ for *some* disjunct $\beta_{j'}$, which gives us $\xi \in \Sigma_{j'}^+$. Since ξ *strictly* instantiates its main label $\sigma_{k'}$, as we said

above, we can choose $\sigma_k = \sigma_{k'}$, and σ_k has been introduced as a main label to the branch on $\beta_{j'}$.

- For any label $\sigma - \Sigma$ on the branch over term β_j, we have $(M(B))_{\langle\sigma\rangle} \models_s \beta_j$.

  This was Proposition 6.4.2, and it is a property on branches, independent of the method by which they are produced. (We did require though that all exceptions in Σ are strict instances of σ, which we showed.)

- $M(B) \models_s \sigma\, F$ for every label $\sigma - \Sigma$ on every branch in C.

  We carefully chose only scwr labels σ as branch labels. Furthermore, we showed $(M(B))_{\langle\sigma\rangle} \models_s F$ for all egi.

- $M(B) \models_s C$.

  This is just a special case of the previous, namely $\sigma = \sigma_0$, which we know to be the main label on some branch label in b_j.

We conclude that $M(B) \models_s S$ as required. Now we need to ensure that none of the assertions added to M introduces a clash. So consider the set M' at any stage during the construction, where a branch label $\hat{\sigma} - \Sigma$ on β_j, leads us to successively add each of the assertions $(\hat{\sigma}\sigma_{j,i} - \Sigma)\, a_{j,i}$ to M', $i = 1, \ldots, n_j$. We know that all exceptions in Σ are egis of σ_0 and hence of the same length as $\hat{\sigma}$. (Note that all exceptions in the labels of M' were chosen to be egi in M; hence all egis in M' are also egis in M, and there is no ambiguity.) We also know that $\hat{\sigma}$ is scwr in M and hence in the larger set M'. Now consider an instance σ of $\hat{\sigma}$ so that $M_{\langle\sigma\rangle} \not\models_s \beta_j$, and find the least general egi σ' instantiating it. If $M_{\langle\sigma'\rangle} \models_s \beta_j$, then by Corollary 5.3.6, part (4), $M_{\langle\sigma\rangle}$ also verifies β_j, which contradicts our assumption. So $M_{\langle\sigma'\rangle} \not\models_s \beta_j$, and σ' is found in Σ_j^- and must instantiate some exception in Σ; so does σ. We have shown that every instance σ of $\hat{\sigma} - \Sigma$ verifies β_j; this further entails $M_{\langle\sigma\rangle} \models_s \sigma_{j,i}\, a_{j,i}$, and hence $M'_{\langle\sigma\rangle} \models_s \sigma_{j,i}\, a_{j,i}$ by monotonicity.

Now all prerequisites for the complex version of Lemma 6.3.1 are established, showing that $M' \cup (\hat{\sigma}\sigma_{j,i} - \Sigma)\, a_{j,i} \models_s S$. We repeat the same argument over all conjuncts $\sigma_{j,i}\, a_{j,i}$, then over all branch labels, all branches on C, and finally over all clauses C in S.

We have thus shown that the final result $M \cup M(B)$ is a model for S. (We only need clash-freeness here.) Being a subset of $M \cup M(B)$, $M(B)$ is also clash-free. We also showed $M(B) \models_s S$, so we conclude $M(B) \models_s S$. □

Similarly as in Proposition 6.2.10, it usually suffices to introduce much fewer labels and exceptions to each branch than the theorem states. Beyond the cases mentioned on Proposition 6.2.10, we may additionally have exception-generated instances $\hat{\sigma}$ which are not realizable. Instead of adding these as main labels to *all* branches on C (as done here, since $\hat{\sigma} \in \Sigma_j^+$ for all j), we can just add them to the default branch $U(C)$, thus introducing $\hat{\sigma}\ \bot$ into $M(B)$. Theorem 6.11.1 shows that every satisfiable set of formulas S has a strong model generated by a set of branches. This guarantees that our search space, namely the set of all sets of branches, is nonempty, so if we perform an exhaustive search, we must find a model. *Termination* of our search, on the other hand, is warranted by our strategy of branching and blocking. To prove it formally, we consider the current *search state* as the current set B of brances, including the nogoods stored on the branches of B as term nogoods. We introduce a strict partial order on search states, which we call *constrainedness*. If we can show that each step (iteration of the main **while** loop) leads to an increase in constrainedness, while infinitely increasing chains are impossible, then the algorithm must terminate.

Definition 6.11.2 *For a search state B, we define the following sets:*

- $\Delta^+(B)$*: the set of all* σ*:* b[13] *so that* σ *instantiates a branch label* $\lambda \in \Lambda(b)$*,*
- $\Delta^-(B)$*: the set of all* σ*:* b *so that a term nogood on* b *realizably blocks branching on* σ*,*
- $\Delta(B) = \Delta^+(B) \cup \Delta^-(B)$.

Due to our nogood invariants, σ: b can only be temporarily in both $\Delta^+(B)$ and $\Delta^-(B)$: If σ: b is blocked by a term nogood, we may not branch on it, or we must immediately undo branching on this instance (of a "larger" branch label), and if a new nogood is introduced and realizably violated, we always undo branch b on σ, which is exactly the instance on which the nogood, becoming a term nogood of b, will block branching over σ.

Definition 6.11.3 *We define a strict order $\prec$ between search states B and B', determined by the smallest* σ*:* b *(wrt the branch order $\preceq$) for which one of the following is true:*

- σ*:* $b \in \Delta(B)$ *but* σ*:* $b \notin \Delta(B')$*: then* $B \prec B'$

[13] *We allow only labels σ made from $*$ and label constants used in the formulas of S, to ensure that $\Delta^+(B)$, $\Delta^-(B)$, and $\Delta(B)$ are finite.*

- σ: $b \in \Delta(B')$ *but* σ: $b \notin \Delta(B)$*: then* $B' \prec B$
- σ: $b \in \Delta^+(B)$ *and* σ: $b \in \Delta^-(B')$*: then* $B' \prec B$
- σ: $b \in \Delta^+(B')$ *and* σ: $b \in \Delta^-(B)$*: then* $B \prec B'$

Let us analyze how different operations and procedures in the algorithm affect the constrainedness of B. As a convention for our discussion, we assume that B' is the set of branches obtained from B after the respective operation.

- Branching increases constrainedness, since $\Delta^+(B')$ contains some σ: b which has not been in $\Delta^+(B)$ or $\Delta^-(B)$ before.
- Introducing a nogood increases constrainedness, since this nogood becomes a term nogood on some branch b blocking a label σ. Nogoods are always violated at the time they are introduced, that means b was branched upon, over the same instance σ. Hence σ: b was in $\Delta^+(B)$. This also shows that b could not have been blocked over the same instance σ by some other nogood before, so the combination of undoing the branch and introducing the nogood amounts to moving σ: b over from $\Delta^+(B)$ to $\Delta^-(B')$, which indeed increases constrainedness.
- Undoing a branch by itself decreases constrainedness. But other than in response to a newly introduced nogoods, undoing of branches occurs only in the *branching* procedure, where we undo strict instances of a label on which we have just branched. We showed previously that $\sigma \sqsubseteq \xi$ implies σ: $b \preceq \xi$: b, so over the scope of the entire *branching* procedure, branching on σ outweighs undoing branches on ξ.
- Within *repair-branch*, the algorithm checks for existing nogoods which may become realizably violated as a result of branching on σ: b. These cannot be term nogoods of b, as term nogood violations have already been taken care of in *branching*, resulting in undoing b over instances of σ. We need to distinguish several other cases:
  - If σ: b instantiates one of the term nogood premises, it cannot be the maximal nogood premise. So the branch undone must be some σ': b' with σ: $b \preceq \sigma'$: b', and the decrease in constrainedness is still outweighed by the increase due to branching on σ: b.

- If σ: b serves as one of the realizability witnesses for the realizable nogood violation and σ: $b \preceq \sigma'$: b' for the maximal nogood premise, then no matter how the nogood violation is repaired, only branches of higher order than σ: b get undone, which is outweighed by the increase due to branching.
- If σ: b itself is undone as one of the realizability witnesses (which may occur when it is of higher order than σ': b'), then a realizability-propagated nogood has been introduced; therefore, σ: b gets removed from $\Delta^-(B)$ but also added to $\Delta^+(B')$, which results in an increase in constrainedness.

- In *merge-nogood*, no label gets branched upon, but a new nogood NG_0 gets introduced. As part of repairing NG_0, some other branch, say σ': b', gets undone and becomes a blocking instance, that is, σ': b' is moved from $\Delta^+(B)$ to $\Delta^-(B')$. As a result, one of the original nogoods NG_j will no longer be violated; more precisely, some instance σ: b_j of the maximal nogood premise σ_j: b_j will be removed from $\Delta^-(B)$. If we had σ: $b_j \preceq \sigma'$: b' for all branches σ': b' undone in repairing NG_0, then σ_j: $b_j\sigma'$: b', so NG_j was not level-b_j-realizably violated and could be repaired, contradicting the assumption made by calling *merge-nogoods*. Therefore, we must have σ': $b' \preceq \sigma$: b_j for at least one σ': b' moved from $\Delta^+(B)$ to $\Delta^-(B')$, the effect of which outweighs all other changes, so B' is more constrained than B.

By considering all operations (branching, nogood repair, and nogood merging) performed in the course of the algorithm, we found that no matter what operations are performed in one iteration of the main **while** loop, the new search state is more constrained than the previous. We already proved in Proposition 6.9.4 that each iteration terminates finitely. Now we observe that the total number of simple labels is finite, and hence the number of branch labels σ: b is finite, and there are only finitely many arrangements of these in the sets $\Delta^+(B)$ and $\Delta^-(B)$. This shows that infinite $\prec$-chains are impossible. We conclude:

Theorem 6.11.4 *After a finite number of iterations of the* **while** *loop, reaches a search state B of maximal constrainedness, at which point the algorithm must terminate.*

We still need to clarify that the algorithm terminates in one of the expected ways—by returning *unsatisfiable* or by returning a model:

Theorem 6.11.5 *Let B be a maximally constrained set of branches. Then no clause has a realizable unbranched label (and $M(B)$ is clash-free), or branch 0 contains an empty nogood with level-0-realizable witness.*

Proof: Suppose B is maximally constrained. If no clause has a realizable unbranched label in B (the termination criterion of the main **while** loop), then Corollary 6.10.3 shows that $M(B)$ is a model for S. So suppose we still have some realizable unbranched label $\lambda = \sigma - \Sigma$ in some clause C, upon which we enter the procedure *satisfy-unbranched-label*. If we could still pick one of the branches b on C over λ, thus adding σ: b to $\Delta^+(B)$, then B would not be maximally constrained. Similarly, if branching on either branch violates a term nogood on some branch b but not level-b-realizably so, then we could undo all branches witnessing the realizability of the nogood violation, and subsequently branch on b, which also increases the constrainedness of B, again contradicting our assumption. So C must fail on λ, and all participating term nogoods have level-$b_{\max}$-realizable witnesses. Therefore, we can merge these nogoods into a propagated nogood which, as we have shown, is also realizably violated. If this nogood did not exist before, then introducing it would increase the constrainedness of B, which is a contradiction to our maximality assumption. So the nogood must have existed before, which now contradicts Proposition 6.9.5 saying that after the previous **while** loop iteration, B did not realizably violate a nogood. The only escape is that the previous iteration produced a nogood violation which *undo-branch* could not repair; as we showed, this is the case only for the empty nogood with level-0-realizable witness; but this condition terminates the algorithm from inside *undo-branch*, returning *unsatisfiable*. □

We put our results of this and the preceding section together into one final theorem, stating the soundness, completeness, and finite termination of our decision procedure:

Theorem 6.11.6 *The algorithm model-finder terminates finitely for every finite set S of formulas, correctly deciding the satisfiability of S. If S is satisfiable, the algorithm returns a set B so that $M(B)$ is a model for S.*

6.12 Remarks on Implementability

As we see, the specification of our model-finding algorithm has grown fairly complex, compared to classical tableau or DPLL-style algorithms for **K**. This should not pass entirely

uncriticized. Anecdotal evidence suggests that automated reasoners are very difficult to implement correctly, even when given a detailed sound and complete algorithm specification. Many reasoners which have been tested on small problems and entered into competitions in good faith were discovered to be unsound upon running them on difficult problems. This may be due in part to memory leaks or other platform-related problems, rather than the implementation itself. But more often than that, unforeseen situations arise, such as the presence of a trivial formula $\bot \vee \bot$ which the implementation may not recognize as unsatisfiable, thus returning an incorrect answer. Another common error source is the implementation of branching and backtracking. Saving a copy of the complete state of the reasoner before branching and returning to it when the branch fails is memory-inefficient, so this is usually not done. But then one must undo all steps performed during branching. Here careless implementations may erroneously retain some data found inside the branch, referenced by pointers which were not reset. Given that we propose a much more sophisticated algorithm to start with and that it will likely be difficult to implement, can we justify such an undertaking?

To address this concern, we claim that the comparison with classical algorithms is too simplistic. Yes, our algorithm is more complex than a classical tableau algorithm without optimizations. But if implemented, it can be expected to outperform such simple reasoners easily. Instead, we should compare our approach to highly optimized reasoners such as those existing in the field [48, 84, 43][14]. Accounting for all optimizations, their algorithmic specifications are of considerable size too [47]; yet this did not stop their authors from implementing them. The comparison is justified because our algorithm already incorporates many of these optimizations: namely, dependency-directed backtracking and caching of unsatisfiable formulas as nogoods. Secondly, our approach makes some other optimizations redundant or less interesting. We will discuss this in our comparison in Section 7.2. Finally, some other optimizations are easily added with little extra overhead (see next section, where we will demonstrate this for Boolean Constraint Propagation). So we can say with confidence that our algorithmic specification gives rise to an already optimized reasoner.

One remarkable feature is that our algorithm has absolutely no recursive function calls (see the chart in Figure 6.1). This is desirable, because it need not allocate a lot of space on the program stack nor handle large dynamically allocated data structures to be thrown

[14] We do admit that this is not entirely fair either, since all these reasoners handle a much larger variety of logics and constructs.

away when undoing branches. This not only saves time but also makes the algorithm's performance less reliant on the platform's memory management and garbage collection strategies at the least, and reduces the potential for said memory leaks and unsoundness at best.

On the downside, our algorithm will accumulate a whole host of nogoods during its search, which we expect to be the main implementation bottleneck. Nogood caches tend to increase linearly with the running time of the algorithm [31]. Starting from a certain problem size or complexity, it will be imperative to manage and delete nogoods without losing completeness. The first measure to be taken is to get rid of redundant nogoods. Among these we classify:

- nogoods of the form $\sigma_1\colon b_1, \sigma_2\colon b_2 \rightarrow \sigma\ \bot$ which just describe a clash between two literals p and $\neg p$ in branches b_1 and b_2. These can be read off the branches directly.
- subsumed nogoods. For instance, if $\sigma_1\colon b_1, \ldots, \sigma_n\colon b_n \rightarrow \sigma\ \bot$ exists, any nogood which specifies more branches on the same labels, or the same branches on instances of these labels, is subsumed and can be deleted.
- nogoods which refer to a part of the search tree which has been exhausted. They will never be referred to again.

When this does not keep the nogood cache from overflowing, more sophisticated techniques may be required to delete nogoods while retaining completeness. An adaptation of dynamic backtracking [31] may be very desirable here.

Finally, we briefly reflect on one major source of complexity in the algorithm specification, namely the need to verify level-b-realizability in its various forms. In non-deontic modal logics such as **K**, we cannot assume that a world has a successor, but as a result of branching on a disjunct with a $\Diamond$-operator (or a constant in $\mathcal{LBL}$), one may come into existence at any time, which makes previously harmless nogood violations realizable. This problem is well-known to frustrate many reasoning algorithms [25, 9], particularly those using free variables or translations to FOL. In the latter, a *dead-end predicate* is used to capture this case [82], but this introduces disjunctions in the scope of each $\Box$-operator, which makes it expensive to handle. Our algorithm does not clutter the set formulas with additional terms. Instead, we provide a dedicated procedure for testing realizability, which gives us control over when and how often to call it.

On the upside, many interesting problems, even in **K**, can be stated without this source of complexity. For instance, all common translation methods for QBF problems into **K**

produce formulas with no $\Diamond$-operators inside disjunctions. In these problems, we know at the outset which instances are realized. While realizability checks will still be NP-complete, we could now cache the results of these checks reliably, so we will not have to redo them each time a new branch is introduced. An algorithm which can handle problems of this restricted subclass will be much easier to implement[15], while remaining useful for a large class of interesting problems.

6.13 Refinements

Among the many possible ways of improving the algorithm, we would like to point out a few we deem most important. They are all concerned with maximizing the amount of deterministic reasoning before and between branching steps.
Consider the set $S = \{** (p \vee q), c* \neg p, *c \neg q\}$. We easily see that S is unsatisfiable, but can we prove this deterministically, without tedious branching, branch repairs, and nogood propagation? We can, if we perform the following preprocessing steps:

- For each original formula in S, create term nogoods in all branches whose terms feature conjuncts complementary with original assertions. In our example above, branch b_1 (on the disjunct p) would be marked with a term nogood $c*$: $b_1 \rightarrow c* \bot$, and branch b_2 (on q) would be marked with $*c$: $b_2 \rightarrow *c \bot$.

- Thus having propagated all original assertions into the branches, search for labels on which a clause fails. This is the case in our example, as the labels $c*$ and $*c$ unify into cc (which is obviously realizable). Now propagate the term nogoods into $\rightarrow cc \bot$, an empty nogood with level-0-realizable witness, showing that S is unsatisfiable.

The general case is as follows: If $\sigma\, l$ is an atom on branch b in clause $C(b)$ with outer label σ_0, and an original assertion $(\sigma' - \Sigma)\, \bar{l}$ exists in S, then derive a term nogood $\lfloor \sigma_0, \sigma' \rfloor\, b \rightarrow (\sigma_0\sigma, \sigma')\, \bot$.

[15] An implementation for a suitable translation of **QBF** has been developed by the author, consisting of 1,400 lines of LISP code. This is a reasonable size, and a version for full **K** should be within reach. At the time of writing, the existing version has not been sufficiently tested to convincingly warrant correctness, and no systematic evaluations have been performed yet. Contact the author for information on further development.

The prospect of deterministically finding inconsistencies usually justifies the cost associated with propagating assertions into branches and searching for failed clauses. Even "near-failures"—empty nogoods with non-realizable witness—impose strong restrictions on potential models for a problem,since they prohibit some instances from ever *becoming* realized, and significantly restrict the search space in which models may exist.

In the realm of propositional satisfiability, the benefits of maximizing deterministic reasoning are well-testified[16]. Particularly in problems with short average clause length (e.g. less than 3 terms), which is typical for real-world problems, it has been observed that deterministic reasoning tends to account for the majority of processing time [77]. This is certainly not because deterministic reasoning is slow, but rather because it has such a far-reaching effect. In propositional problems, deterministic reasoning is performed (mostly) by means of *Boolean constraint propagation* (BCP), also known as *unit propagation*[17].

In obtaining a suitable version of BCP for $\mathcal{LBL}$, term nogoods play an integral role, as we have just seen: Some term nogoods can be derived from branches and complementary original assertions. Similarly, term nogoods can be derived from assertions introduced in a branching step. Note that in this case, the branch index and branch label must be included in the dependency set of the term nogood.

Next, failed clauses should be actively searched for, rather than chancing upon them while branching on an unbranched label. But we can go even further: If but one term in a clause C can be branched upon over a label—or equivalently, if all but one branch are blocked by term nogoods whose labels unify—we can infer the remaining branch deterministically over the mgu of these labels. We can express this by the following nogood-like *forced branch rule*:

$$\sigma_1\colon b_1, \ldots, \sigma_n\colon b_n \rightarrow \sigma\colon b,$$

which expresses a propagation of branch assignments: If we have branched on $b_1, \ldots, b_n$

[16]In the class of QBF, however, classes of problems are known where full BCP creates an exponential number of assertions, compared to the original size of the problem, whereas the problems can be shown (un)satisfiable in a polynomial amount of time by means of DPLL and like procedures [88]. Nonetheless, even in QBF, BCP is more frequently beneficial than it is harmful, and an obvious way to avoid getting trapped in such extreme cases is to limit the number of BCP steps per branching step.

[17]We should formally keep a distinction between BCP and other optimization techniques [17, 66], such as tautology elimination and detection of inconsistencies. But we argue that these are appropriately discussed together, as the same mechanisms can be used in $\mathcal{LBL}$ to handle any of these. And we skip the Pure Literal rule entirely, as we see not much use for it.

over $\sigma_1, \ldots, \sigma_n$, respectively, then we are forced to choose branch b on the label σ (because $b_1, \ldots, b_n$ cause all other branches in C to be blocked). This establishes all assertions in $\beta(b)$ deterministically; these assertions in turn can be propagated into term nogoods for other branches, and so on. Forced branch rules with an empty premise play a special role, since they cannot be undone: indeed, we have "learned" a new "original" assertion.

Forced branch rules behave like nogoods in many aspects: they themselves can be propagated into other forced branch rules or nogoods, and in fact they may incorporate nogoods, namely when the term $\beta(b)$ contains a conjunct $\sigma_i \bot$. This similarity to nogoods suggests that forced branch rules and BCP can be easily integrated into our algorithm.

Finally, we will briefly discuss the potential for a technique often used in tableau-based reasoners for **K**, called *semantic branching* [47, 52][18]. Consider a disjunction $p_1 \vee \ldots \vee p_n$. In a classical tableau algorithm, branching would be performed successively on $p_1, \ldots, p_n$, until one choice leads to a satisfying assignment. Using semantic branching, in turn, we would branch on some p_i and then on $\neg p_i$. One advantage of semantic branching is that it splits the search space into two disjoint spaces, whereas in syntactic branching, parts of the search space (e.g. when two variables p_i and p_j are *true*) occur in several branches, leading to redundant search. Another advantage is that all disjunctions containing p_i are simultaneously satisfied and can be ignored for the rest of the first branch, whereas all disjunctions containing $\neg p_i$ are satisfied and can be ignored in the second branch. A disadvantage surfaces when instead of p_i, we have a more complex (e.g. modal) formula φ_i, whose negation may not exist in the original problem; in fact, $\neg\varphi_i$ may introduce new $\Diamond$-operators, and hence new accessible worlds. In some classes of problems, semantic branching has been shown inferior to syntactic branching, while in the majority of cases the converse is believed to be true.

In $\mathcal{LBL}$, we need to keep in mind that problems usually cannot be converted into CNF, but only into the more general LCNF, where each term is a conjunction of labelled atoms. Keeping this in mind, semantic branching could be performed in one of two ways:

- Branch on a propositional variable p over a substantially general label, such as some $*^k$, or even over all $*^k$, $k = 0, \ldots, d(S)$, where $d(S)$ is the depth of the problem. Branch on $\neg p$ over all instances where branching over p leads to a clash. This approach is

[18] It is an ongoing debate whether such a reasoning algorithm can be rightfully classified as a tableau algorithm, or rather a DPLL-type algorithm.

very much in the spirit of the DPLL procedure. As a disadvantage, branching over p by itself may not lead to any of the clauses in the problem being satisfied, because it may always occur in a conjunction with other atoms in all branch terms.

- Branch (simultaneously) on (all conjuncts in) a term β_j over a substantially general label. (In our standard algorithm, we only branch over the outer label σ_0 of the clause in which β_j occurs.) Consider all clauses of the problem satisfied, in which the same term β_j occurs (and whose outer label instantiates the branch label). This can be done quite elegantly, if we define as the branch index of each term β in the problem a Gödelization of β. Then identical terms are given identical branch indices. This approach is not much different from the standard approach if the terms in the problem are mostly distinct. Furthermore, in response to a clash, it is not likely efficient to introduce the negation of β_j which is another disjunction, read: clause, in itself. Therefore, this approach only offers only the benefit of simultaneously satisfying multiple clauses, but it is not truly semantic branching.

If all terms in S consist of only one (labelled) atom, then both approaches coincide, and also, many of the cited disadvantages do not apply, making this the most promising class of problems for semantic branching. Problems of this class include, for instance, translations of QBF problems into $\mathcal{LBL}$.

A general note of caution on semantic branching in $\mathcal{LBL}$ is that the search space is not as clearly divided into two disjoint parts. For instance, branching on σp and then on $\sigma\neg p$ is unsound, unless σ is ground. Therefore, any algorithm which branches over more than just ground labels must address the problem of correctly responding to clashes, including finding suitable labels σ' for which $\sigma' \neg p$ may be safely inferred.

Chapter 7

Discussion and Conclusion

Now we see but a poor reflection as through a mirror; but then we shall see face to face. Now I know in parts; but then I shall know fully, as I have also been known.

The Bible, 1 Corinthians 13:12

7.1 Contributions of this Work

In this work we described a new logic $\mathcal{LBL}$ which can be used for representing formulas of the modal logic **K**. We introduced a model-based semantics, as well as a translation scheme for converting $\mathbf{K_{NNF}}$-formulas into $\mathcal{LBL}$, and showed that this translation preserves satisfiability.

We provided a careful analysis of the strengths and limitations of our approach. While there are classes of formulas for which minimal $\mathcal{LBL}$-models are larger than minimal Kripke models, the ability to specify variable assignments over sets of ground instances typically results in a more efficient representation of models. The notion of labels with exceptions, allowing us to specify default variable assignments with exceptional instances, results in even more concise models. We motivated why a decrease in model size will lead to a decrease in running time for reasoning tasks as well. We supported our claims by describing a large class of problems—translations of $\mathbf{QBF}_2$—for which polynomial-size $\mathcal{LBL}$-models but no polynomial-size Kripke models exist. Advances in reducing the model size are interesting in the real world where automated reasoners have space constraints: they enlarge the class of

satisfiable problems for which a reasoner can return a model, instead of just reporting that the formula is satisfiable.

Finally, we presented a model-finding algorithm for sets of $\mathcal{LBL}$-formulas. This algorithm constitutes a sound and complete decision procedure for $\mathcal{LBL}$, and thus for **K**. The algorithm utilizes the representational power of labels with variables and exceptions, trying to verify universally scoped variables by universal variable assignments wherever possible, and iteratively repairing contradictions which may occur, by creating exceptional instances. One favourable characteristic of the algorithm is that it accommodates the addition of formulas to the knowledge base: a previously found model can be extended and repaired locally, so as to satisfy the additional formulas as well. The algorithm can be furnished with a variety of search strategies and heuristics.

An implementation has not been provided, and experimental evidence is ultimately needed to support our claims of having found an efficient formalism. However, we motivated why our algorithm is well-suited for solving hard translations of **QBF**-problems [61], such as those used in the 1999 and 2000 modal logic theorem prover comparisons [72, 74]. These problems are hard for tableau-based provers, because minimal Kripke models for them, if they exist, are exponential in the modal depth of the problem. By contrast, $\mathcal{LBL}$-models for these formulas need not be exponential.

We must point out that these benchmark problems did not arise from real-world applications, and an algorithm's efficiency on hard theoretical problems may be a poor measure of its performance on real-world problems; these are often of lower complexity in theory but hard in practice because of their much larger size. Given the clear and present need for reasoners suitable for solving real-world problems, the ability to perform well on these is much more important. But we did not fail to motivate why our algorithm is expected to solve real-world problems efficiently as well. This is because it has the ability, frequently needed in practice, to specify default assignments and revise them on exceptional instances, possibly recursively

The research led to a few interesting byproducts which we will mention briefly:

- We gained some insight into the structure of $\mathcal{LBL}$-formulas (which reflect the structure of **K**-formulas) and how the interplay of labels and propositional connectives leads to various degrees of complexity:
  - **K**-formulas can be converted into negation normal form in linear time and without size increase. This is a well-known fact.

- The modal operators in a $\mathbf{K_{NNF}}$-formula can be converted to constants and wildcards over a domain D. This constitutes a "reification of possibilities": instead of saying that a formula is *true* in *some* world, we now fix a *specific* world in which it is true. Assigning a fixed label to this possible world does not compromise satisfiability, as long as distinct $\Diamond$-operators are assigned distinct constants; but it may lead to an increase in the size of ground models.
- A set of labelled formulas can be converted into our so-called expanded And/Or normal form. This reflects the fact that labels distribute over conjunctions. This may increase the size of each formula by a worst-case factor $d(F)$, the formula depth, because its labels may be duplicated; the conversion can be performed in linear time in the size of the *new* formula. Note that this factor $d(F)$ is usually small compared to the size of the problem.
- We see that reasoning with labelled formulas consisting purely of labels and conjunctions does not pose any difficulties—after conversion to ENF they form a set of assertions which verifies itself.
- Next, we are faced with the problem of eliminating disjunctions preceded by labels. This is the hard part; we see that the PSPACE-completeness of reasoning in **K** arises solely from this task. Our observation in the introduction is confirmed by the theory: disjunctions scoped over necessities (read: non-ground labels) *are* the main bottleneck for automated reasoning in **K**.
- The occurrence of labels (with existential constants) *inside* disjunctions does not increase the theoretical complexity of eliminating disjunctions, but it poses additional implementation challenges, as the set of rgl changes dynamically throughout the search.
- The result of eliminating all disjunctions is a set of assertions verifying the $\mathcal{LBL}$-formula. We must check (or ensure by our algorithm) that this set is clash-free in order for it to qualify as a model. This task is still intractable, but much easier than the previous, namely co-NP complete.

- We presented a normal form for $\mathcal{LBL}$-formulas which we named *labelled clause normal form*, in analogy to the propositional clause normal form. To the best of our knowledge, no such normal form has been proposed for **K** or any similar logic before. The LCNF is useful as a specification format, because it eases the task of specifying

a reasoning algorithm, while its adverse effects on model size and running time are tolerable.

- We identified a class of Kripke models we termed *strict models*, arising from the classical tableau algorithm without optimizations; they are equivalent to *ground* weak $\mathcal{LBL}$-models.

Our approach is readily adaptable to logics equivalent to **K**, such as **QBF**. Multi-modal logics such as $\mathbf{K}_m$ and its description logic twin $\mathcal{ALC}$ (without terminological reasoning) can be handled by a straightforward extension: Instead of one domain D of constants, we introduce m sorts of constants with domains D_j, $j = 1, \ldots, m$. Universal statements over the entire domain can be made using a sorted wildcard $*_j$. For an early draft of our approach in the language of $\mathbf{K}_m$, see [46].

Probably the greatest hindrance to the applicability of our approach is that most real-world knowledge bases are modelled using additional constructs, such as global axioms, number restrictions, transitive and inverse modalities. Our formalism in its current form does not handle these. However, we claim that the ideas presented in this work are not restricted to **K**; in Section 7.3 below, we will outline how they might be adapted to handle the above extensions.

7.2 Comparison with Existing Approaches and Techniques

In the field of modal logic reasoning, approaches based on translation into FOL and using resolution continue to be actively pursued and improved [58][1]. Resolution is a rather powerful reasoning technique [77]; for instance, the reasoning algorithms underlying all the powerful SAT solvers in existence today can be viewed as derivations of propositional resolution. Although we have not formally shown it, we conjecture this to be true for (a first-order logic version of) our algorithm as well. A proof would follow along the lines of its propositional logic equivalent: all nogoods introduced during the algorithm and propagated in order to derive the empty nogood correspond to first-order formulas which could just as well be generated by resolution. However, we already mentioned the main shortcoming of

[1] We mention in passing that "direct" resolution calculi for modal logics without translation into FOL are possible and have been proposed [76, 1] but received less attention.

resolution: it provides poor guidance on how to find a solution or compute meaningful nogoods. Correspondingly, systematically propagating all nogoods in all possible ways would quickly lead to combinatorial explosion. An exacerbating factor is that our problems are only translated into LCNF; through conversion into CNF, which is common practice in FOL reasoning, the problem size will further increase, and their structure further obscured.

Recently, two promising alternatives to the resolution method in first-order logic have been proposed: the Model Evolution calculus [2, 5], and the disconnection tableau calculus [12, 64, 65]. The latter is based on tableaux, as its name suggests, while the former implements DPLL-style reasoning. Both calculi have been implemented [4, 63, 65], and their running times on some classes of problems are reported to be comparable to those of resolution-based provers [94, 93]. Remarkably, both approaches resemble ours in that clauses are satisfied by default literal assignments, and ensuing clashes repaired by introducing exceptions. (Note that exceptions are not explicitly represented as such; instead, whenever two "assertions" have common instances, the most specific one applies.) However, we have tested the reported algorithms on translations of labelled formulas into FOL, and found that they do not reach the reasoning power of our algorithm: Whenever one of these algorithms can find a truth assignment using defaults with exceptions, so can ours; conversely, our algorithm could find a concise representation for some instances where the other two algorithms would resort to an enumeration of all ground instances as exceptions. We conjecture that our algorithm achieves greater conciseness because it is more powerful in handling exceptions. Of course, the final verdict on runtime efficiency must be deferred until an implementation of our own algorithm becomes available. We are not aware of any efforts to develop a modal logic version of the Disconnection Tableau calculus or the Model Evolution calculus.

The ultimate standard our algorithm must measure up to is set by the highly optimized, industrial-strength tableau and DPLL-style algorithms for modal and description logics. We will collectively refer to them as "ground algorithms", since from our point of view, they reason over ground instances (worlds). Let us briefly go over some of the optimization techniques which made them so successful, and find out which of them are matched by equivalent or superior features of our algorithm, and which ones we may reap additional benefits from. Our selection is mostly based on the relative utility of the technique in conventional approaches. For instance, heuristics have been reported to contribute little efficiency gain, so we will omit them here. For more detailed treatments on all these techniques see [47, 52, 51].

Intelligent backtracking We discussed dependency-directed backtracking and backjumping in Section 6.5. When reasoning on ground instances, dependency-directed backtracking becomes much simpler: there are no branch labels and nogood witnesses to remember, only branch indices. *Backjumping* is even simpler but weaker: When returning to a branch with index b, then all branches with index greater than b are completely discarded. This is done because the intermediate branches are considered not worth keeping, or because the cost of bookkeeping involved in repairing an earlier branch and resuming with a higher branch index outweighs the advantage of keeping information from intermediate branch points. Indeed, while implementing backjumping results in considerable efficiency gain over unoptimized implementations [52], an implementation of dynamic backtracking in the DLP prover has only resulted in a speedup by another factor of 2 over backjumping in experiments [86][2]. Of course, our algorithm already incorporates dependency-directed backtracking in the form of nogood reasoning [19], on which it relies for completeness. So we should reap the same or perhaps even more advantages, since one branch point in our algorithm (over a label with wildcards) may correspond to several branch points in a "ground" algorithm.

Caching This technique is well-known to prune the search tree, but in fact it does so by reducing the size of models for formulas. For each world w visited during the algorithm, the set Φ of subformulas which need to be satisfied in this world is stored in a cache, possibly along with a partial model for these formulas, if one has been found. If the same set Φ is encountered on another world w', the partial model with root w can be reused as a partial model with root w'. Likewise, if Φ has been shown unsatisfiable in w, then it is also unsatisfiable in w'. This saves us from repeating previous reasoning steps. Some implementations distinguish an S-cache and a U-cache for satisfiable and unsatisfiable sets of formulas, respectively [35]. Furthermore, the criterion of identical matches for Φ in cache hits can be weakened to *subsumption* [45]. Caching can also be used to implement loop checking in modal logics with global axioms [45]. When viewed from the perspective of Kripke structures, satisfiability caching is a special case

[2]The picture appears different in QBF: dependency-directed backtracking with learning has resulted in considerable gain over backjumping in many problem instances [62, 33], but not in all [34]. Note that, strictly speaking, backjumping is a special case of dependency-directed backtracking where all nogoods from intermediate branches are discarded. However, we often use the term *dependency-directed backtracking* in the sense that at least some learning, i.e. caching and re-use of intermediate nogoods, takes place.

of *node merging*, a technique we will describe in the Appendix. The important lesson we will learn there is that even with caching, the number of instances (worlds) visited by a "ground" algorithm is at least as large as the number of worlds in a minimal Kripke model for a problem.

Arguably the greatest hindrance to the effectiveness of caching is the occurrence of mutually incompatible sets of formulas. We cited the Ladner translation of QBF formulas [61] several times, mentioning that minimal Kripke models for these problems are exponentially larger than their modal depth. According to what we said above, caching cannot change this fact. More so, it is ineffective, because any two worlds feature mutually incompatible variable assignments. A "ground" algorithm necessarily takes an exponential amount of time traversing all instances. Indeed, the poor performance of all three major provers on problems of this class displays this behaviour [42, 49, 85].

The set of nogoods in our algorithm can be regarded as a generalized U-cache. But again, thanks to our nogoods, we attain the same or better conciseness than a "ground" algorithm with caching. This is because nogoods are more flexible than the per-world sets of formulas in a conventional U-cache: they pinpoint the source of the inconsistency[3], and since they can be specified over labels with wildcardss, they can capture inconsistencies of variable assignments across different and perhaps even disjoint sets of instances (worlds). On the other hand, most of the cache hits in "ground" algorithms occur when Φ is a set of subformulas in the scope of $\Box$-modalities, where the two worlds w and w' are merely instances of these same modalities. In Example 3.4.4 we needed to construct a world satisfying the variable q, as necessitated by the original formula $\Box \Diamond q$. Having done this once (and cached the model for q), we can reuse this world for each subsequent instance of the $\Box \Diamond$ modality, as shown in Figure 3.2. In $\mathcal{CBL}$, the assertion $*c\, q$ offers the same concise representation without caching. Note also that mutually incompatible variable assignments are no longer a hindrance to concise models *per se*: even the slightly modified example $\{c_1 * p, c_2 * \neg p, *c\, q\}$ offers the same concise representation of q over all instances as before. We studied this in more generality in Section 4.7; see Propositions 4.7.9 and 4.8.2 in particular. To summarize,

[3] At least the DLP prover [86] tries to trace the source of a clash in an unsatisfiable world, and caches only the subset of formulas participating in the clash. But even so, these "nogoods" are tied to formulas over one specific world (ground label).

much of the need for reusing partial models is eliminated by the concise representation of models already achieved. While some form of S-cache is imaginable—we could store combinations of branches verifying certain subsets of clauses, for instance—we wonder whether our algorithm would benefit much from it.

Model Merging This is another technique for improving the efficiency of reasoning by reducing the size of models. When a set of formulas $\{\Diamond\ \varphi, \Diamond\ \psi\}$ is encountered, a model can be obtained by merging two compatible models for φ and ψ, effectively creating a model for $\Diamond\ (\varphi \wedge \psi)$. (See Example 3.4.3 and Figure 3.1, where $\varphi = p$ and $\psi = q$.) This is another instance of our more general *node merging* technique we describe in the Appendix. In the setting of $\mathcal{LBL}$, model merging corresponds to non-strict labelling of $\Diamond$-modalities in order to reduce the number of constants in labels. We discussed in Section 4.8 why this optimization technique is beneficial, and that great care must be taken to preserve soundness. An implementation of this approach may follow along the lines of [47], adapted to labels with wildcards.

BCP and Simplification We already explained how BCP can be quite naturally integrated into our algorithm, and it promises to improve efficiency considerably. Simplification is a generalization of Boolean constraint propagation: If a set of formulas contains a disjunction $\varphi \vee \psi$ and a formula φ, then we need not branch on $\varphi \vee \psi$; rather, this disjunction is redundant and can be deleted, in accordance with the law of absorption in natural deduction. *Modus ponens* is another instance of simplification: If $\neg\varphi \vee \psi$ and φ are given, we can replace the first formula by ψ, thus removing a disjunction which might otherwise lead to branching. A detailed treatment of simplification has been given in [69]. While most implementations apply a few simplification rules, we know of only one reasoner which performs simplification extensively [44]. Simplification can be performed as a single preprocessing step on the initial set of formulas, or also on all new formulas encountered during the search. It has also been shown that simplification can improve the efficiency of caching [35, 45].

The idea of reasoning with labelled formulas was originally born out of the desire to enhance the effect of simplification. Consider the two formulas $\Box\ (\neg\varphi \vee \psi)$ and $\Diamond\ \varphi$. Inside the modal operators, we could apply *modus ponens* to obtain ψ just as above, which would save us a branching step in one instance. However, the universal scope in the first formula prevents us from performing the simplification: we cannot express

the fact that $\neg\varphi \vee \psi$ must be branched on over *all but one* of the accessible worlds. Upon translating the problem into $\mathcal{LBL}$, we can utilize labels with exceptions and preprocess these formulas into $(* - \{1\})\ (\neg F \vee G)$, $1\,F$, $1\,G$[4]. We only need to extend the syntax of $\mathcal{LBL}$ slightly, in that labels with exceptions must be allowed in formulas as well as assertions. Adjustments to the definition of $\models_s$ can be easily made.

We consider simplification a very useful step in the initial translation stage of our algorithm. It can alleviate the negative effects of strict labellings (or pre-empt the use of node merging). For instance, the two formulas $\Diamond\ \varphi$ and $\Diamond\,(\varphi \vee \psi)$ can be simplified into one formula $\Diamond\ \varphi$, so in the translation to $\mathcal{LBL}$ only one constant instead of two will be introduced. Once we have converted the $\mathcal{LBL}$-formula into LCNF, we do not really introduce new formulas and thus have no further need for simplification beyond what can be attained by BCP alone. Lastly, simplification will be a very useful technique for controlling the size of the nogood cache, as it identifies subsumed nogoods which can be erased.

Absorption This technique seems unrelated at first glance. It originates from Description Logic reasoning. Real-world knowledge bases tend to have a large number of so-called *concept inclusion axioms*, which are just a special case of disjunctions over a large set of formulas. As we said, these types of formulas constitute a major bottleneck for tableau-based provers. The technique of absorption addresses their said devastating effect on performance [54]. It specifies a method of integrating these axioms into the definition of concepts, so they will no longer be branched upon in individual instances, unless that respective concept is specifically mentioned.

Our approach is not related to absorption, but it offers an alternative technique for dealing with disjunctions over large sets of instances. We branch on them universally, which produces little overhead and will lead to clashes only in instances which mention literals complementary to those in disjunctions; those clashes are then repaired locally by branching (on other terms). The fact that absorption so successfully alleviates this bottleneck of reasoning on real-world problems is one of the reasons why we have confidence that our own algorithm will perform well on those problems too.

[4] If F and G are atoms, the same effect can be gotten by BCP alone.

7.3 Directions for Future Research

As we said, the ultimate confirmation of the usefulness of our algorithm must come from its performance in practical applications. Therefore, our most immediate goal is to develop and test a prototypical implementation and run it on existing theoretical hard benchmarks and real-world problems. Once this has shown fruitful, we ought to study branching strategies, heuristics, and other optimization methods we mentioned, which may improve efficiency in the context of labelled formulas. However, one would be hard-pressed to justify the development of yet another full-fledged, general-purpose reasoning system for description logics. Instead, we envision that our ideas be integrated into existing systems, which already embody years of research and development into heuristics, machine-level optimizations, additional logical and algebraic constructs, and user interfaces.
The new formalism and model-finding algorithm give rise to a number of interesting questions and possible areas of further research, which we will briefly discuss:

- We mentioned the relationship between **K** and **QBF**, and hence between $\mathcal{LBL}$ and **QBF**, in various places throughout this work. Reasoning with QBF itself constitutes a very active research area, which includes work on efficiency techniques [32, 62], as well as an annual QBF solver evaluation[10]. In Section 4.7, we studied a translation method from **QBF** into $\mathcal{LBL}$, which we could use in order to solve QBF using our algorithm. We do not expect this to be a very efficient method *per se*. However, the structure of translated QBF is much more regular than that of translated **K**-formulas: we know from the outset which ground instances are realized. All the complications due to realizability checks, introduction of new constants in labels, and nogoods with multiple witnesses, do not occur in QBF problems. Therefore, we can streamline our algorithm code and eliminate or simplify many of the procedures. A number of further optimizations are possible due to the fact that a world may have at most two successors; but we cannot go into details here. One novel feature our research may contribute, which so far has been considered difficult [34], is that branching need no longer follow the order of propositional variables prescribed by the problem. On the other hand, there are a number of specialized techniques for QBF we would have to compete with. So without experimental evidence, we cannot tell for certain whether an adaptation of our approach will prove competitive in this field.

- Our algorithm is strongly related to local search techniques. Let us revisit our general strategy of finding a model for a set S of formulas, as described in Section 6.4: Create a "starter set" of branches, scoped globally over the clause labels, so that S is entirely covered; then repair any clashes between the assertions in the branches, until no more clashes occur. This is strongly reminiscent of the method of iterative repair, a local search technique in Constraint Satisfaction Problems [75]. In fact, we may argue that our algorithm really is a local search procedure, with (a rather expensive machinery of) constructs—nogoods and branch indices with a prescribed order for undoing branches—to ensure its completeness. Now this high and laudable standard becomes meaningless when faced with problems prohibitively large for a deterministic solver[5]. If only we give it up in favour of an incomplete search, we will have a large variety of methods, techniques and heuristics for guiding the search at hand, easily adaptable to the context of $\mathcal{LBL}$. This field is much too wide to give an overview here. After all, our research was motivated by finding models for satisfiable formulas; so we would gladly trade the ability of proving a formula unsatisfiable for the considerable additional mileage a local search algorithm may afford us.

- As for the ability to handle additional logical constructs, global axioms top the list of most desirable constructs. Global axioms are formulas which hold in every ground instance, and they constitute the terminological part (TBox) of a description logic knowledge base. It is very easy to *represent* global axioms in $\mathcal{LBL}$: we propose to use the notation $*^*$ as a prefix in each axiom. This new type of placeholder can be instantiated by any label of any length; it thus represents the infinite union of all labels $*^k$, $k \in I\!N_0$. *Reasoning* with these labels gives rise to new challenges, though. It is well-known that finite Kripke trees are not an adequate representation of models for **K** with global axioms in general. We either need to abandon finiteness or allow loops in models. In our formalism, sets of assertions cannot represent models with loops, and reasoning with infinite models is certainly not a feasible option either; so we must add new constructs in models as well, which we envision to be of the form $\sigma \mapsto \sigma'$: this expresses that all worlds matching label σ are to be identified with worlds in label σ'. The idea is essentially the same as that of loop checking in tableaux for

[5]The phenomenon of astronomically large but provably finite running times is jokingly referred to as Ctrl-C-completeness.

modal logics with axioms [22, 39, 42, 45]—see also our earlier discussion of caching: Whenever a world γ is encountered which must satisfy the same set of formulas as another world γ' we have seen before, these formulas can be satisfied by identifying γ with γ'. In the spirit of this work, we would not content ourselves with the ground case, but demand to perform loop checking over labels with variables and exceptions. Devising a complete and finitely terminating algorithm is not only a non-trivial task; it is interesting also from an algebraic point of view, since the $\mapsto$ operator induces an equivalence $\cong$ on the set of strings over $D \cup \{*\}$, and proving the finiteness of the partition defined by $\cong$ (which is what we must do to ensure the finiteness of our model) is equivalent to the well-known *word problem.*

- Many other constructs, such as transitivity, reflexivity, symmetry, and to some extent number restrictions as well, can also be handled using models involving the new $\mapsto$ operator. While this approach may have the appeal of working universally over a large variety of modal logics, it may not be very efficient. As a point in case, consider a number restriction stating—or implying—that a world γ can have at most n successors. If we have constructed a model in which $\gamma*$ has more than n rgi, then we can use $\mapsto$ constructs to identify some of these rgi with one another. But the difficulty lies in doing this over labels implicitly defined by assertions with non-ground labels. Simply put: How can we determine the most general label over which a number restriction is violated, and which of the possible pairings of (non-ground!)labels should we identify via $\sigma \mapsto \sigma'$, in order to fix it? Instead it may be easier to modify the algorithm for labelling a formula, ensuring that no more than n constants are introduced in places which can instantiate a number-restricted modality. Since this constitutes a non-strict labelling, we are then faced with the nontrivial task of preserving soundness; we may have to backtrack over several attempts at labelling the modal operators, an additional source of complexity.

  Other alternatives to handle modal logics with transitivity, symmetry etc. may be found in approaches based on first-order translations of these logics. See [82] for an overview of techniques.

- The formal properties of $\mathcal{LBL}$ itself may be an interesting object of further study. We can show that $\mathcal{LBL}$ can be expressed within the description logic $\mathcal{ALC}(\sqcup)$ with role union. Thinking of each constant $c \in D$ as a role, the formulas $c\,F$, $[c]\,F$ and $*\,F$

can be expressed as $\exists c\, \top \wedge \forall c\, F$, $\forall c\, F$, and $\forall D\, F$, respectively, whereas D represents the union of all roles represented by its constants. However, the well-formed $\mathcal{LBL}$-formulas form a substantially restricted subclass of this logic. On the other hand, $\mathcal{LBL}$ contains formulas which cannot be equivalently expressed in **K**. One such instance is the formula $F = 12\, p \wedge *2\, q$. It entails $12\, (p \wedge q)$. By contrast, the **K**-formula $\Diamond\Diamond\, p \wedge \Box\Diamond\, q$ does not entail $\Diamond\Diamond\, (p \wedge q)$. In fact, there is no **K**-formula which would have a set of Kripke models equivalent to the set of labelled models for F. It would be interesting to explore whether the expressivity of $\mathcal{LBL}$ can be utilized to represent an interesting type of knowledge for which **K** is too weak and $\mathcal{ALC}(\sqcup)$ too strong. Finding a suitable axiomatization of this logic would be an asset here.

- We compared our approach with recent first-order calculi and found many similarities. Now we can obviously take the opposite route and ask whether $\mathcal{LBL}$ can be extended to handle at least a subclass of FOL. In a way, this question is already answered affirmatively, if we translate $\mathcal{LBL}$ into FOL. The real challenge, though, is to find an extension to a "natural" subclass of FOL such as the two-variable guarded fragment. We are confident that some interesting results can be obtained, so the research presented here may well provide a contribution to the popular area of first-order logic.

Appendix A

Tableaux and Optimizations

Ever since the days of Kripke's original paper [60], the tableau method as a proof tool for modal logics has been accompanying the semantical construct of Kripke models. Apart from being very intuitive and thus amenable to reasoning by hand, it has also proven very successful in Automated Theorem Proving with modal and description logics. The three arguably most widely used theorem provers in the field, FaCT [48], DLP [84], and RACER [41, 43], implement optimized variants of the tableau algorithm. This stands in sharp contrasts to the related fields of propositional and first-order Logic. Tableau calculi for these logics exist (see e.g. [91, 23, 12]), but in terms of practical efficiency, they have so far been inferior to algorithms based on DPLL (for SAT) and resolution (for FOL in clausal form).

Apart from their importance in the field, there are two other reasons why we introduce the tableau calculus here. First, the terminology and proof techniques in Chapters 2 and 3 were developed with the tableau calculus in mind. We will demonstrate in this section how our techniques can be used to prove some known results about tableaux, and contrast them with "classical" techniques. Secondly, we wish to compare the expressivity of Kripke models with that of tableaux. We will derive two results which are novel to the best of our knowledge: First, we will show that any Kripke model for a satisfiable formula can be "derived" by a tableau of comparable size, provided we modify one of the rules slightly. This modification subsumes a commonly and successfully used optimization techniques but has an even more general scope. We will also show that the "standard" unoptimized tableau calculus can derive any *strict* Kripke model, providing another justification for defining and characterizing strict models. Conversely, any satisfiable complete node in a tableau gives

rise to a strict Kripke model of comparable size, and the same is true for the optimized tableau calculus and general Kripke models. We will thus have shown that tableaux are equally concise to a class of Kripke models. So our results of Chapter 4 on comparing the conciseness of $\mathcal{LBL}$ with that of Kripke semantics extend analogously to tableaux.

The tableau calculus originally stems from Gentzen's much older sequent calculus [29]: A tableau for a formula $\neg\varphi$ can be viewed as a Gentzen sequent for φ, written upside-down. Branch closures in the tableau correspond to the initial tautologies in the Gentzen sequent, from which φ is derived by rules which are the inverse of tableau rules. It is not surprising that the tableau calculus is generally considered a refutation method [23]: If the above tableau for $\neg\varphi$ closes, then φ is shown *valid*. However, with at least equal justification, tableaux can be seen as a tool for checking *satisfiability*, and even for finding satisfying models, the primary goal of this work. (The information found on a complete open node is sufficient for constructing a model.) It is intriguing that both soundness and completeness of tableau calculi are usually stated in the negative: "a formula is unsatisfiable iff it has a closed tableau", but shown using the contrapositive: "a formula is satisfiable iff all its tableaux are open". Yet except for these proofs, the literature on the aspect of model finding using tableaux is rather sparse. We hope that this chapter will provide a useful contribution to the field.

Our treatment of the tableau calculus is self-contained, but it differs from the "standard" approach in many ways. For instance, we will develop a few general, logic-independent results, before narrowing down on propositional logics and finally on $\mathbf{K_{NNF}}$. For a standard treatment of tableaux on various normal modal logics including **K**, we refer the reader to [56, 22, 39].

A.1 General Definitions

Treatments of tableau calculi vary greatly not only in the logics covered, but also in the way they are represented, which is a common source of confusion. We will focus our attention on one specific style which best suits our needs, called *prefixed (or labelled) tableaux*. Given a logic **L**, they are specified over the following entities:

Definition A.1.1 *A* prefixed formula *is of the form $w\,\varphi$, where w is some object (label), and φ is an* **L***-formula.*

An r-expression *is of the form $w\,R\,w'$, where w and w' are both objects.*

These labels have a slightly different connotation from those used in $\mathcal{LBL}$. They are not part of the formula, but they rather specify a context in which the formula applies. The r-expressions define a relation between labels. In **K**, we can think of them as worlds and accessibility arcs in a (real or hypothetical) Kripke frame. But they could be used in a more general sense. Labelled tableaux are an instance of the even more general labelled deduction systems [26].

In keeping with our established practice, we use greek letters for **L**-formulas and sets of formulas, and roman letters for labelled formulas and sets of them. As usual, we use a shorthand notation $w\,\Phi = \{w\,\varphi : \varphi \in \Phi\}$ for a set of formulas prefixed by the same label w. Conversely, we write $S(w) = \{\varphi : w\,\varphi \in S\}$ for the set of all formulas whose label in S is w.

Definition A.1.2 *A* tableau calculus *for* **L** *provides a set of* tableau rules *and a* closure condition. Tableau rules *are of the form*

$$\frac{P}{C_1|\ldots|C_n}\quad (R),$$

where R is the name of the rule, and P (the premise *of R) and $C_1, \ldots, C_n$ (the* conclusions *of R) are sets of* schemata *of prefixed formulas: that is, expressions containing metavariables which can be instantiated by formulas and world labels, so as to give prefixed formulas. Any such instantiation must be uniform across the rule: that is, the same variables in P and $C_1, \ldots, C_n$ must be replaced by the same expressions. Set brackets are omitted for singleton sets. Some rules will have side conditions; we will specify these informally. A rule R is* applicable *to a set of prefixed formulas S, if (using notation from above) it has an instance $R\sigma$ so that $P\sigma$ is a subset of S but none of the $(C_i)\sigma$ is. The sets obtained from S by* applying *(this instance of) rule R are $S \cup (C_1)\sigma, \ldots, S \cup (C_n)\sigma$.*

A tableau *for a set of formulas Φ is a tree whose nodes are sets of formulas and whose root is the set $w_0\,\Phi$, whereas the successors of any non-leaf node S in the tree are exactly the sets obtained by applying a fixed applicable rule to S. Usual terminology for trees applies. In particular, a* branch *to (or: of) a node S is the unique directed path from the root to S as its* final node. *As usual, we speak of a tableau for $\{\varphi\}$ simply as a tableau for φ. A tableau T' is an* extension *of a tableau T, if T' is a supergraph of T and both tableaux share the same root. (Obviously this relation is a partial order on tableaux.)*

A tableau node is closed *if it satisfies the closure condition, and* open *otherwise. A tableau is* closed *if all its leaf nodes are closed. A tableau node S is* complete *or* saturated *wrt a set of tableau rules, if none of the rules are applicable to S; S is* complete *if it is complete wrt all tableau rules. A branch in a tableau is called* complete *if it has a complete final node, and a tableau is* complete *if all its leaf nodes are complete.*

According to this definition, a tableau node accumulates all formulas which have been derived anywhere along its branch. While in practice one would avoid duplicating all formulas in each node and instead consider all formulas on the branch, this view helps simplify our description greatly. We would also like to point out that there are three conceptually different graph-like structures: first, the labels and r-expressions on a single node define a relation, which may or may not give rise to a tree. Secondly, one single tableau constitutes a tree of nodes. And finally, the set of all tableaux due to the nondeterminism in choosing formulas and rules for each rule application constitutes a tree. This nondeterminism is relevant in some mostly older tableau and Gentzen-type calculi such as [21, 22] for **K**: If one tableau fails to show a formula unsatisfiable, then alternate tableaux must be considered, until all possible tableaux have been exhausted. One reason we use labelled tableaux is that rule applications are "don't care" nondeterministic [73]: their order does not matter in finding a formula unsatisfiable[1]; for satisfiable formulas, one can show that one tableau produces the same models modulo renaming of labels as another. This property is also called *strong proof confluence* [55]; of course, we will need to prove it. For this reason, we will define the property of strong completeness below, along with other more common properties:

Definition A.1.3 *A tableau calculus is* sound *(or* correct*) if whenever a tableau for a set Φ is closed then Φ is unsatisfiable. A calculus is* complete *if every unsatisfiable set of formulas has a closed tableau and* strongly complete[2] *if every tableau for an unsatisfiable set of formulas has a closed extension. A calculus is* finitely terminating *if every tableau for a finite set of formulas is finite.*

---

[1] But the order of rule applications does matter greatly for finding a proof or satisfying model *efficiently.*

[2] The "classical" definition of strong completeness reads: "Every fair derivation from an unsatisfiable set of formulas is a refutation" [3]. In the language of tableaux, this means: Every complete tableau for an unsatisfiable set is closed. This statement is easily shown equivalent to our definition, justifying the term "strong completeness". The main advantage of strong completeness is expressed in Proposition A.1.4: If a tableau does not close, no alternate tableaux need to be considered; instead, the formula has been shown satisfiable. A related property of tableaux called *confluence* holds for our calculus, but we will not prove it, since we do not need it to demonstrate strong completeness.

Following the standard, we have defined soundness and completeness "in the negative". But we prefer the contrapositive of these definitions; as we said in the introduction to this chapter, we will use it to derive results related to satisfiable formulas and the existence of models. Among these are the following:

Proposition A.1.4 *In a strongly complete tableau calculus, if Φ has a tableau $\mathcal{T}$ with a complete open branch, then Φ is satisfiable.*

Proof: Let S be the complete open final node on the said branch. Since no rule is applicable to S, S will always be a leaf node in any extension of $\mathcal{T}$. Therefore, no extension of $\mathcal{T}$ can be closed. Since the calculus is strongly complete, Φ cannot be unsatisfiable. □

For the next few results, we assume that the calculus is finitely terminating.

Proposition A.1.5 *In a finitely terminating tableau calculus, any tableau $\mathcal{T}_0$ for any set of formulas has a finite complete extension $\mathcal{T}$.*

Proof: We construct a sequence of tableau extensions $(\mathcal{T}_i)$, starting with $\mathcal{T}_0$, as follows: If every leaf node in $\mathcal{T}_i$ is complete, we are done. Otherwise, we take a non-complete leaf node S and apply an applicable rule to it; the sets thus obtained are appended to node S, yielding a new tableau $\mathcal{T}_{i+1}$. Observe that $\mathcal{T}_{i+1}$ is an extension of $\mathcal{T}_i$. If the sequence $(\mathcal{T}_i)$ is infinite, then the graph union of all $\mathcal{T}_i$ is an infinite tableau for Φ, contradicting the fact that the calculus is finitely terminating. Hence, the sequence must end in some complete tableau $\mathcal{T}_i$, according to our construction. □

Corollary A.1.6 *In a finitely terminating tableau calculus, any set of formulas has a complete tableau.*

Proof: The structure $\mathcal{T}_0$ with one node w_0 Φ is a tableau for Φ which can be extended to a complete tableau, as shown in Proposition A.1.5. □

A similar argument as in Proposition A.1.5 shows that after a finite number of rule applications along any branch a complete node must be reached.

Proposition A.1.7 *If the calculus is sound and finitely terminating, then any tableau T_0 for a satisfiable set Φ of formulas can be extended to a tableau with a complete open node.*

Proof: By Proposition A.1.5, $\mathcal{T}_0$ can be extended to a complete tableau $\mathcal{T}$. Since Φ is satisfiable and the calculus is sound, $\mathcal{T}$ cannot be closed, so one of the branches (all of which are complete) must be open. □

In a calculus which is sound, strongly complete and finitely terminating, we observe that the existence of a complete open node in some (extension of a) tableau for Φ is necessary (see Proposition A.1.7) and sufficient (see Proposition A.1.4) for Φ to be satisfiable. That node alone serves as a witness for Φ. So we can be oblivious about the tableau's tree structure and think of the calculus as a (nondeterministic) expansion of the set $w_0\ \Phi$. The tableau rules thus act as saturation rules for this set. In upcoming theorems, we may refer to a tableau node S as a *node expansion* of $w_0\ \Phi$ (or of Φ). And we say that a set of formulas in node S is *fully expanded* if no rule can be applied to any of these formulas—usually because the conclusions of any such rule are already in S.

In order to *characterize* the property of finite termination, König's Lemma is a useful tool. In the context of labelled tableaux (in which we recall that each rule application leads to a finite number of successors), it states:

Lemma A.1.8 *The tableau calculus is finitely terminating iff the node expansion along any single branch in every tableau for every set of formulas terminates after a finite number of expansions.*

Sadly, Propositions A.1.4 and A.1.7 cannot serve as a *characterization* of completeness and soundness, which we are interested in. To get such characterizations, following the standard pattern of soundness and completeness proofs, we need to show that tableau rules preserve satisfiability. Now notice that we have not yet defined when a tableau node is satisfiable. This must obviously be specific to the logic and tableau rules under consideration. However, we would expect certain properties from this notion of satisfiability, which we will discuss now:

(1) The root $w_0\ \Phi$ is satisfiable iff Φ is satisfiable.

It is reasonable to expect the notions of satisfiability in **L** and in our tableaux to coincide in this way.

(2a) Every closed node is unsatisfiable.

This reflects the meaning of closed nodes as unsatisfiability witnesses. One ought not to expect the converse to be true. An open node can be found unsatisfiable in the course of expanding the tableau and finding that all descendant nodes are closed. But we should have this weaker property for nodes which cannot be further expanded:

(2b) Every complete open node is satisfiable.

Finally, we expect tableau expansions to be satisfiability-preserving. We split this requirement into two directions:

(3a) Every satisfiable non-leaf node has at least one satisfiable successor.
(3b) If a node has a satisfiable successor, then it is itself satisfiable.

Two alternate characterizations of (3a) and (3b) will prove useful later:

(3a') Every satisfiable non-leaf node has at least one satisfiable descendant.
(3b') If a node has a satisfiable descendant, then it is itself satisfiable.

It is easy to see that (3a') is weaker than (3a), and (3b') is stronger than (3b), and that (3a) and (3b) together are equivalent to (3a') and (3b'), either side stating that tableau expansions preserve satisfiability. As it turns out, these properties equip us with everything we need to prove soundness and completeness:

Proposition A.1.9 *If the notion of satisfiability obeys properties (1), (3a') and (2a), then the tableau calculus is sound.*

Proof: We prove the contrapositive of soundness in Definition A.1.3. Assume that Φ is satisfiable, then so is $w_0\,\Phi$ according to (1). Suppose there existed a closed tableau $\mathcal{T}$ for $w_0\,\Phi$. Then all leaf nodes in $\mathcal{T}$ must be closed and by (2a) unsatisfiable. Now (3a) allows us to find a chain of satisfiable descendants, beginning at the root of $\mathcal{T}$. Since any satisfiable non-leaf node must have at least one satisfiable descendant, the chain can only end in a leaf node which thus is shown satisfiable, contradicting our earlier finding that all leaves are unsatisfiable. □

Proposition A.1.10 *If the notion of satisfiability obeys properties (1), (3b) and (2b), and the calculus is finitely terminating, then it is strongly complete.*

Proof: As before, we prove the contrapositive of strong completeness in Definition A.1.3. Assume all extensions of a given tableau for Φ are open. Since the calculus is finitely terminating, a finite complete extension can be found. Choose an open leaf node S (whose existence is warranted by our assumption). By Property (2b), S is satisfiable. Now follow the chain of nodes from S backwards to the root $w_0\ \Phi$. Property (3b) guarantees that each predecessor node is satisfiable, so ultimately $w_0\ \Phi$ is satisfiable, which is equivalent to Φ being satisfiable. This shows strong completeness. □

We remark that finite termination is not a requirement in some completeness proofs in the literature. In our setting we can do away with it by generalizing property (2b) to arbitrary sets S; on an infinite open branch, we then take the union of all nodes on the branch as S. Then we must require that S is complete on any infinite branch; this requires some fair order of evaluation of rules, ensuring each rule is applied is applied to any formula on which it is applicable. Since our tableaux for $\mathbf{K_{NNF}}$ turn out to be finitely terminating, we skip these cumbersome details.

A.2 Tableau Calculi for $\mathbf{K_{NNF}}$

In this section we will introduce two prefixed tableau calculi for the modal logic $\mathbf{K_{NNF}}$. Both are variants of the calculus in [22], which in the case of **K** coincides with the single-step tableaux of [71, 73, 9]. The main difference with any of these is that we regard labels as atomic, whereas all cited approaches use strings to represent labels; instead of through r-expressions, the strings implicitly define a relation between themselves, very much like ground labels in $\mathcal{LBL}$. One of our calculi is sound, strongly complete and finitely terminating, whereas the other gives rise to more concise models while giving up soundness. Having proved this, the results of the previous section show that existence of a complete open branch in a tableau for Φ is characteristic for Φ being satisfiable. We will see that the formulas on such a branch provide enough information to construct a Kripke model for φ.

Let us first state the closure condition:

Definition A.2.1 *A tableau node S is closed if it has a label w so that $S(w)$ contains a clash.*

In other words: S is closed iff $w \perp \in S$ or both $w\, p \in S$ and $w\, \neg p \in S$ for some w and p. Now we present two rules for dealing with propositional operators. (In many traditional treatments, several types of rules are needed, in addition to a double negation rule. Thanks to considering formulas in NNF only, we only need one each.)

$$\frac{w\, \alpha_1 \wedge \ldots \wedge \alpha_n}{\{w\, \alpha_1, \ldots, w\, \alpha_n\}} \quad (\alpha) \qquad \frac{w\, \beta_1 \vee \ldots \vee \beta_n}{w\, \beta_1 | \ldots | w\, \beta_n} \quad (\beta)$$

We have the following two rules for the modal operators:

$$\frac{\{w\, \Box\, \nu_0, w\, R\, w'\}}{w'\, \nu_0} \quad (\nu) \qquad \frac{w\, \Diamond\, \pi_0}{\{w'\, \pi_0, w\, R\, w'\}} \quad (\pi)$$

The rule (π) has the following two side conditions: It is only applicable to a set S if there exists no w' so that the formulas in the conclusion are already in S. Secondly, if applied, we must choose a label w' which does not (yet) exist in S.
We distinguish two tableau calculi for $\mathbf{K_{NNF}}$: The *standard* calculus includes rules (α), (β), (ν), and (π), whereas the *calculus with node merging* is derived from the standard calculus by dropping the second side condition from the (π) rule. That is, we are free to choose w' among all labels, including those already existing in S. It is easy to see that this leads to an unsound calculus:

Example A.2.2 The set $\Phi = \{p, \Diamond\, \neg p\}$ is satisfiable. The calculus with node merging lets us apply the (π) rule to $w_0\, \Phi$ so as to obtain $S = \{w_0\, p, w_0\, \Diamond\, \neg p, w_0 R w_0, w_0\, \neg p\}$. This node is closed, and since it is the only leaf node in the tableau, the entire tableau is closed.

As our first result, we would like to show that both calculi are finitely terminating. The proof for this is quite analogous to the standard proof given in [22], Chapter 8, so we highlight the important principles, referring the reader to the above source for details. The first crucial element of the proof is the so-called *subformula principle* [91]; let us, state it recursively first:

Proposition A.2.3 *In any of the tableau rules above, every formula φ occurring in the conclusion is a subformula of some formula φ' occurring in the premise. In particular, we always have $d(\varphi) \leq d(\varphi')$.*

This can be easily verified on the definitions of the rules above. From this recursive form the non-recursive version follows easily:

Corollary A.2.4 *Every formula φ in every node of any tableau for a set Φ is a subformula of some formula in Φ, and $d(\varphi) \leq d(\Phi)$.*

Since Φ has only $O(|\Phi|)$ different subformulas, any node S in a tableau can only have this many formulas carrying the same world label w. Our second task is to show that the number of different labels in a node is finite and bounded. One may think of using König's Lemma in conjunction with the depth function. The problem is that r-expressions need not define a tree on the labels in S—they do so only in the standard calculus. We rather propose a method which works for both calculi. Our approach also differs from that in [22] to accommodate for the different way in which we label formulas. Rather than considering the length of a label (which we have not defined), we consider the set W_k of labels preceding a formula of depth at least k in a given tableau node S. We just observed that the depth of any formula on a tableau node is no larger than $d(\Phi)$; now we will show that for each such $k = 0, \ldots, d(\Phi)$ the set W_k is bounded. It is easy to see that $W_{d(\Phi)} = \{w_0\}$, because the only rule through which a label w can be introduced is the (π)-rule, and any new formula introduced into w through the (π)- or (ν)-rule is of the form π_0 or ν_0, for which $d(\pi_0) = d(\Diamond\ \pi_0) - 1 < d(\Phi)$, similar for ν_0. We can use the same idea recursively to show that W_k for any depth $k < d(\Phi)$ is also bounded:

Lemma A.2.5 *For a given node S in any tableau for Φ, and a given $k = 1, \ldots, d(\Phi)$, the inequality $|W_{k-1}| \leq (b+1)|W_k|$ holds, where b is the number of π-subformulas in Φ.*

Proof: We obviously have $W_k \subseteq W_{k-1}$, so the only interesting labels are those in $W_{k-1} - W_k$; we will show that there are no more than $b \times |W_k|$ of them.
Given a particular label $w' \in W_{k-1}$, find the first node on the branch leading to S on which w' precedes a formula of depth $k-1$, and call it S'. If S' is the root $w_0\ \Phi$, then $w' = w_0$, which is a label already occurring in W_k. Otherwise, the formula has just been introduced through application of a tableau rule. If this was an (α)- or (β)-rule, then the premise is a formula of depth at least $k-1$ preceded by the same label w', occurring on the previous node. But if its depth is exactly $k-1$, we have a contradiction to our assumption that S' is the first node on which w' precedes a formula of depth $k-1$, and if the depth is at least k, then we have $w' \in W_k$ which is uninteresting. Next, if the formula is of the form $w'\ \nu_0$

introduced through the $(\nu) - rule$, then we must have two premises $w\ R\ w'$ and $w\ \Box\ \nu_0$ in S'. Since $d(\Box\ \nu_0) = k$, we have $w \in W_k$. Furthermore, $w\ R\ w'$ must have been introduced previously by application of a (π)-rule. Finally, if the formula is of the form $w'\ \pi_0$ introduced through the (π)-rule, then we must have a premise $w\ \Diamond\ \pi_0$ in S', so $w \in W_k$ (and we also introduced $w\ R\ w'$ into S'). In all cases, new labels $w' \in W_{k-1}$ are created by applying the (π)-rule to a π-formula with a label $w \in W_k$. By the subformula principle, every such π-formula must be a subformula in Φ; furthermore, after the (π)-rule has been applied once to a formula $w\ \Diamond\ \pi_0$, it becomes unapplicable to the same world label w and formula $\Diamond\ \pi_0$. So the total number of (π)-rule applications producing labels in W_{k-1} is limited by $b \times |W_k|$, where b is the number of π-formulas in Φ. □

We remark that with a little more effort in the proof, we can get a smaller bound on the "branching factor" b. For instance, our upcoming Proposition A.3.5 will show that the π-formulas used in our proof above are part of an atomic cover for S, and we can infer that their number is bounded by the maximal number of π-formulas occurring (as subformulas) in (the formulas of) any atomic cover of Φ. But this refinement, though important in practice, does not improve on our theory.

By applying Lemma A.2.5 recursively until $k = 0$, we get an overall bound on the number of labels occurring in any node S. Since the number of formulas preceded by any label is also bounded, any extension along one branch of the tableau must reach a node at which no more rule application is possible, which is a complete node. Using Proposition A.1.8, we arrive at our conclusion that the calculus is finitely terminating:

Theorem A.2.6 *Any tableau for Φ can be extended in a finite number of steps to a complete tableau.*

A.3 Soundness and Completeness

Now we will work towards defining a Kripke model from a node in the tableau. To motivate our approach, let us make a simple observation on nodes in the standard calculus:

Proposition A.3.1 *For any tableau for a set $w_0\ \Phi$ in the standard calculus, and any node S in the tableau, every label w' in S occurs exactly once on the right-hand side of an expression $w\ R\ w'$ in S, with the exception of w_0 which does not occur on the right-hand side at all.*

Proof: Each r-expression $w\ R\ w'$ arises from applying the (π) rule, which is also the only way of introducing a new label w'. Furthermore, once introduced, the side condition stipulates that w' not be used again on the right-hand side in another application of the (π) rule. □

We define a graph $G(S) = (W, R)$ as follows: Let W be the set of labels occurring in S, and R the set of pairs (w, w') occurring in the r-expressions $w\ R\ w'$ in S. We can see that $G(S)$ is a Kripke frame, rooted in w_0. (The latter follows from an easy induction argument: Assuming that there exists a path from w_0 to w in $G(S)$, and S is expanded to S' by applying a (π) rule, adding $w\ R\ w'$, then there exists a path from w_0 to w' (and to any other label) in $G(S')$.) For the standard calculus, Proposition A.3.1 shows that $G(S)$ is a tree.

Definition A.3.2 *Let S be an* open *node, and $G(S)$ be the graph defined above. Define a valuation function V by selecting*

$$\{w \in W : w\,p \in S\} \subseteq V(p) \subseteq \{w \in W : w\,\neg p \notin S\} \tag{A.1}$$

for each $p \in P$. Then $K(S) = (W, R, V)$ is a Kripke model generated by S.

A valuation satisfying (A.1) can always be found when S is open, because $w\,p \in S$ implies $w\,\neg p \notin S$. But usually there is some slack in the way V is specified, so the definition of $K(S)$ is nondeterministic.

Proposition A.3.3 *If $S \subseteq S'$, then $G(S)$ is a subgraph of $G(S')$, and for any $K(S')$ there exists a corresponding $K(S)$ which is a submodel of $K(S')$ (see Definition 3.2.5).*

Proof: With $G(S) = (W, R)$ and $G(S') = (W', R')$, we immediately see that $W \subseteq W'$ and $R \subseteq R'$, so $G(S)$ is a subgraph of $G(S')$. Now given the valuation V' on any model $K(S')$, we define $V(p) = V'(p) \cap W$, which makes (W, R, V) a Kripke model. We only need to show that this model is generated by S, so we verify (A.1). For $K(S')$, we have:

$$\{w \in W' : w\,p \in S'\} \subseteq V'(p) \subseteq \{w \in W' : w\,\neg p \notin S'\}$$

Upon intersecting all sets with W, we get:

$$\{w \in W : w\,p \in S'\} \subseteq V(p) \subseteq \{w \in W : w\,\neg p \notin S'\}$$

And since $S \subseteq S'$, we conclude:

$$\{w \in W : w\,p \in S\} \subseteq V(p) \subseteq \{w \in W : w\,\neg p \notin S\}$$

This completes the proof that (W, R, V) is a Kripke model generated by S. □

We will see that every Kripke model generated by a complete open set in a tableau for Φ is indeed a model for Φ, and conversely, every Kripke model for Φ is, in a sense, generated by a complete open set in some tableau for Φ. Our main goal of this section is to prove these two theorems, which entail the soundness and completeness of the standard tableau calculus. The proof follows the familiar pattern of this work: We will initially establish some results which hold in *any* tableau calculus employing the propositional rules (α) and (β), and then extend these results to the calculi we introduced above. Thus, let $\mathcal{T}$ be a tableau (for any formula in a propositional logic) in a calculus with (at least) rules (α) and (β) and the closure condition, Definition A.2.1.

Proposition A.3.4 *Let S be a node in $\mathcal{T}$, w a label in S, and Ψ the set of formulas $\varphi \in S(w)$ so that φ is an atom, or one of the rules (α) and (β) is applicable to $w\,\varphi$. Then Ψ is a cover for Φ.*

Proof: First, let us show that Ψ contains a cover for every formula $\varphi \in \Phi$. We show this by induction on the structure of φ: If φ is an atom, then $\varphi \in \Psi$, and we can simply choose $\{\varphi\}$ as a cover for itself. Secondly, we show that the claimed property is preserved under conjunction. Let $\varphi = \alpha_1 \wedge \ldots \wedge \alpha_n \in \Phi$. If $\alpha_1, \ldots, \alpha_n$ are not all in Φ (which means, $w\,\alpha_1, \ldots, w\,\alpha_n$ are not all in S), then rule (α) is applicable to $w\,\varphi$, so φ is in Ψ, and again $\{\varphi\}$ can be chosen as a cover for itself. Otherwise the induction hypothesis applies to all α_i, so Ψ contains a cover for every α_i. Therefore, Ψ also contains the union of these covers, which is a cover of φ. Finally, we show that the property is preserved under disjunction. Let $\varphi = \beta_1 \vee \ldots \vee \beta_n \in \Phi$. If none of the β_i are in Φ, then rule (β) is applicable to $w\,\varphi$, so φ is in Ψ and can be chosen as its own cover. Otherwise, some β_i is in Φ, and by applying the induction hypothesis to β_i, Ψ contains a cover for β_i, which is also a cover for φ. By structural induction, we have shown the property for all $\varphi \in \Phi$.

Now take the union of all covers found in the first half of the proof. We said that Ψ contains a cover for each $\varphi \in \Phi$, so it contains their union. Conversely, each element in Ψ was chosen as a cover for itself, so the union of all covers contains Ψ. Put together, we have shown that Ψ is the union of covers of all elements in Φ and thus a cover of Φ. □

Corollary A.3.5 *Let S be a* complete *node in $\mathcal{T}$, w a label in S, and Φ the set of formulas φ so that $w\,\varphi \in S$. Then the atoms in Φ form an atomic cover for Φ.*

Proof: Consider the cover Ψ found in Proposition A.3.4. It contains all the atoms in Φ and all the formulas to which a tableau rule is applicable. But since S is complete, none of the rules (α) and (β) is applicable to any of the formulas in S, so Ψ contains only the atoms in Φ, hence it is an atomic cover. □

Proposition A.3.6 *Let $\mathcal{T}$ be a complete tableau, w a label in S, Φ any set so that $w\,\Phi$ is a subset of some node S in $\mathcal{T}$, and Ψ any minimal cover for Φ. Then S has a complete successor node S' which contains $w\,\Psi$.*

Proof: We recall Corollary 2.1.31 which says that every cover for Φ is obtained from Φ through a finite number of successive refinements. So we can show this proposition by induction over the refinements. The base case $\Psi = \Phi$ is obvious, as S' is a superset of S and as such contains $w\,\Phi$. □

Next, let us define a notion of satisfiability for nodes:

Definition A.3.7 *Let S be a node in a tableau for Φ. For a Kripke model K which has $K(S)$ as a submodel, we write $K \models_k S$ if $K, w \models_k \varphi$ for every $w\,\varphi \in S$. If such a model K exists, we say that S is satisfiable, otherwise S is unsatisfiable.*

Some of the desired properties of Section A.1 are quite obvious:

Proposition A.3.8 *(Property 1) For any Kripke model K so that $K, w_0' \models_k \Phi$, there exists an equivalent model K' containing w_0 so that $K' \models_k w_0\,\Phi$. Conversely, for any such K', we have $K', w_0 \models_k \Phi$. Hence, $w_0\,\Phi$ is satisfiable iff Φ is.*

Proof: Convert K into K' simply by renaming the worlds so that the root w_0' becomes the label w_0 used in the tableau node. The conditions $K' \models_k w_0\,\Phi$ and $K', w_0 \models_k \Phi$ are equivalent because they are equivalent for each $\varphi \in \Phi$ according to Definition A.3.7. □

Proposition A.3.9 *(Property 2a) A closed node is unsatisfiable.*

Proof: For a closed node S, $K(S)$ was not defined. Therefore, no Kripke model can be found in Definition A.3.7. □

Proposition A.3.10 *Any subset S of a satisfiable set S' is satisfiable.*

Proof: A Kripke model K witnessing the satisfiability of S' in Definition A.3.7 must have $K(S')$ as a submodel. But Proposition A.3.3 shows that $K(S')$ in turn has $K(S)$ as a submodel. Furthermore, since $S \subseteq S'$, $K, w \models_k \varphi$ for every $w\,\varphi$ in the smaller set S. So K obeys the requirements of Definition A.3.7 with respect to S too, which shows that S is satisfiable. □

Corollary A.3.11 *(Property 3b) If S has a satisfiable successor, then S is satisfiable.*

Proof: This follows from Proposition A.3.10 and the fact that S is strictly growing through successive rule applications. □

The two missing pieces, namely Properties (3a) and (2b), are the hardest and also the most interesting parts of the soundness and completeness proofs, respectively. We put Property (3a) in the following wording:

Proposition A.3.12 *(Property 3a) If $K \models_k S$ and a rule of the* standard *tableau calculus is applicable, then $K \models_k S \cup C_i$ for one of the conclusions C_i of the rule.*

Proof: We consider each rule separately. If (α) is applicable to a formula $w\,\alpha_1 \wedge \ldots \wedge \alpha_n \in S$, then $K, w \models_k \alpha_1 \wedge \ldots \wedge \alpha_n$. But then $K, w \models_k \alpha_i$ for every $i = 1, \ldots, n$, so $K \models_k S \cup \{w\,\alpha_i : i = 1, \ldots, n\}$. For rule (β), applicable to $w\,\beta_1 \vee \ldots \vee \beta_m \in S$, we show analogously that $K \models_k S$ implies $K \models_k S \cup \{w\,\beta_j\}$ for some $j = 1, \ldots, m$. The modal rules are more interesting: Suppose rule (ν) is applicable to $w\,\Box\,\nu_0, w\,R\,w' \in S$. Then $K, w \models_k \Box\,\nu_0$. Now we recall that $K(S)$ is a submodel of K, so $w\,R\,w'$ implies that w' is a successor of w in K. Therefore $K, w' \models_k \nu_0$ which shows $K \models_k S \cup \{w'\,\nu_0\}$, as required. Finally, if rule (π) is applicable to $w\,\Diamond\,\pi_0$, then $K, w \models_k \Diamond\,\pi_0$, so w must have a successor w'' in K so that $K, w'' \models_k \pi_0$. The side condition of the (π)-rule requires us to introduce a *new* label w', but w'' may already be in use in S. To overcome this problem, we duplicate the subtree $K_{w''}$ and rename its root to w'. The result is a model K' in which w' is a distinct successor of w, and $K', w' \models_k \pi_0$. This proves $K' \models_k S \cup \{w\,R\,w', w'\,\pi_0\}$, and it is easy to see that $K(S \cup \{w\,R\,w', w'\,\pi_0\})$ is still a submodel of K'.

It remains to be shown that the new tableau node is open. Assume the node contained a formula $w\,\bot$ or two formulas $w\,p$, $w\,\neg p$. Since as shown K witnesses the satisfiability of the new node, we would have $K, w \models_k \bot$, or $K, w \models_k p$ and $K, w \models_k \neg p$, which is impossible. (Replace K by K' in case of the (π)-rule.) □

Theorem A.3.13 *The standard calculus for* $\mathbf{K_{NNF}}$ *is sound.*

Proof: We showed properties (1) (Proposition A.3.8), (2a) (Corollary A.3.9) and (3a) (Proposition A.3.12), and Proposition A.1.9 implies soundness. □

What can go wring in the calculus with node merging? If we re-use an already existing world w' for the consequent of a (π)-rule, we have no control over whether w' is really a successor of w in K, or at least whether there exists a model K in which w' could be *made* a successor of w by introducing an additional arc. In general this is impossible because the calculus is unsound, as we have already seen.

We will now establish Property (2b) for both calculi. That is, we would like to show that any complete open node is satisfiable. As before, we need a Kripke model to witness this. It turns out that $K(S)$ itself contains all the information we need:

Proposition A.3.14 *(Property 2b) If* S *is a complete open node, then* $K(S) \models_k S$. *Hence,* S *is satisfiable.*

Proof: Since S is open, $K(S)$ exists and is well-defined. In our proof we will make no assumption about whether $K(S)$ is rooted or not, or for that matter, what tableau it occurs in, and for what set of formulas. We only stipulate that none of our tableau rules be applicable to S, and that the maximal depth of all formulas in S is a finite integer k_0. We classify the formulas in S according to their depth:

$$S_k = \{w\,\varphi \in S : d(\varphi) \leq k\} \qquad (k = 0, \ldots, k_0)$$

Just as S itself, these sets split further along the labels of their elements into sets $S_k(w)$, for any w occurring in S. As an immediate consequence of Proposition A.3.10, we see that S_k is complete. (And being a subset of S, S_k is also open.) Using induction on k, we will show that $K(S) \models_k S_k$ for all k, which proves the proposition.

So take any w occurring in S, and any $k = 0, \ldots, k_0$. Corollary A.3.5 showed that the atoms in $S_k(w)$ form an atomic cover for $S_k(w)$. As usual, these atoms split into sets Ψ_A, Ψ_N, and Ψ_P. We first observe that Ψ_A is clash-free, since $S_k(w)$ is open. So Ψ_A does not contain $\bot$, nor does it contain both p and $\neg p$ for the same $p \in P$. If $p \in \Psi_A$, then $w\,p \in S$, so $w \in V(p)$ by (A.1), and hence $K(S), w \models_k p$. Likewise, if $\neg p \in \Psi_A$, then $w\,\neg p \in S$, so $w \notin V(p)$ by (A.1), and therefore $K(S), w \models_k \neg p$. An occurrence of $\top$ is trivially satisfied, so altogether we get $K(S), w \models_k \Psi_A$. If $k = 0$, then Ψ_N and Ψ_P are empty. Since this

result holds for all w, we have shown $K(S) \models_k S_0$, establishing the base case. For $k > 0$, assume $K(S) \models_k S_{k-1}$, so $K(S), w' \models_k S_{k-1,w'}$ for all w'. Now consider any $\Diamond\ \pi_0 \in \Psi_P$. Since S is complete, the (π)-rule must be unapplicable, so we must have two formulas of the form $w\ R\ w'$ and $w'\ \pi_0$ in S. Since $d(\pi_0)$ is at most $k-1$, we find π_0 in $S_{k-1,w'}$. Since our induction hypothesis stated that $K(S), w' \models_k S_{k-1,w'}$, and since w' must be a successor of w in $K(S)$, we showed $K(S), w \models_k \Diamond\ \pi_0$. Thirdly, consider any $\Box\ \nu_0 \in \Psi_N$, and any successor w' of w in $K(S)$. The accessibility relation of $K(S)$ contains exactly the pairs corresponding to the r-expressions in S, so $w\ R\ w' \in S$. But then the (ν)-rule would be applicable to $w\ \Box\ \nu_0$ and $w\ R\ w'$, unless $w'\ \nu_0 \in S$, This in turn implies $K(S), w' \models_k \nu_0$ by the induction hypothesis (as ν_0 is of maximal depth $k-1$), and since we showed this for any w-successor w', we obtain $K(S), w \models_k \Box\ \nu_0$. The conclusion of our induction step, $K(S), w \models_k S_k(w)$ for all w occurring in S, follows from Theorem 3.3.1. And by the principle of induction, we conclude $K(S) \models_k S_k$ for any k. Now choose $k = k_0$, in which case $S_k = S$, which completes the proof. □

We wrap up the completeness result as follows:

Theorem A.3.15 *Both calculi for* $\mathbf{K_{NNF}}$*, with or without node merging, are strongly complete.*

Proof: We showed properties (1) (Proposition A.3.8), (2b) (Corollary A.3.14) and (3b) (Proposition A.3.11). Proposition A.1.10 implies completeness. □

A.4 Tableaux and Kripke Models

Let us re-evaluate Proposition A.3.14 in light of how we constructed $K(S)$. We introduced one world corresponding to each label in S, an accessibility relation corresponding to the r-expressions $w\ R\ w' \in S$, and a valuation according to the propositional literals in S. Then we showed that this sufficiently describes a Kripke model for S and thus for the original set Φ, provided S is complete. We can thus give an upper bound on the size of a Kripke model in terms of complete tableau nodes:

Corollary A.4.1 *If a tableau for* Φ *has a complete open node* S*, then* Φ *has a Kripke model with at most* $|S|$ *worlds.*

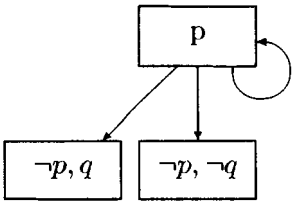

Figure A.1: *A minimal model for* $\Phi = \{p, \Diamond(\neg p \wedge q), \Diamond\Diamond(\neg p \wedge \neg q)\}$.

We have thus shown that Kripke models are at least as concise as tableaux. In practice, $K(S)$ would be smaller than S by a factor of up to $|\Phi|$, due to the presence of nonatomic formulas in S, and once again, the actual size of $K(S)$ depends on details of how the accessibility relation and valuation function are represented.

A bit more interestingly, we will now show that the converse is true: Given a Kripke model for a set Φ, we can construct a tableau (or, if you will, a node expansion) with no more than $\#K$ labels. For we have:

Theorem A.4.2 *Given any set* Φ *of* $\mathbf{K_{NNF}}$*-formulas, if* $K = (W, R, V)$ *is a Kripke model for* Φ*, then* Φ *has a complete open node expansion* S *in the calculus* with node merging *so that* $K' \cong K(S)$ *for some submodel* K' *of* K *and a suitable completion* $K(S)$.

Before we set out to prove this, we remark that this theorem does not follow from Proposition A.3.12. In our proof, we had to "expand" K by duplicating existing submodels, which gives us no control over the eventual size of K. Nor does the result follow from soundness proofs given elsewhere in the literature. (These proofs use a mapping from worlds in K to labels on the tableau, and again it is not guaranteed that the mapping is one-to-one.) In fact, to the best of our knowledge, this result is novel.

The idea of the proof sounds easy: Drop from K the worlds and arcs which are not relevant in satisfying Φ, and show that the remaining model is equivalent to $K(S)$. However, it is not obvious to decide which worlds and arcs are irrelevant. The following example shows what can happen if K contains cycles:

Example A.4.3 Consider the set $\Phi = \{p, \Diamond(\neg p \wedge q), \Diamond\Diamond(\neg p \wedge \neg q)\}$. At first glance, we should expect that the root world need only have two successors, and any additional successors must be redundant. Now consider the Kripke model given in Figure A.1. (In

fact, this model is minimal, as at least three worlds are needed to accommodate the three mutually inconsistent variable assignments.) Here w_0 has three successors including itself, and the world w_2 is not needed to satisfy any of the two π-formulas in Φ. But w_2 is still needed in order to satisfy the *inner* π-formula $\Diamond(\neg p \wedge \neg q)$ which also gets evaluated on w_0. This happens because we must allow w_0 to be "visited" again. In order to find all the relevant parts of K, we will keep track of all atoms (including ν- and π-formulas) which have to be satisfied in each world, accumulating atoms as we revisit worlds.

Proof: (of Theorem A.4.2) To reduce the overhead in notation, we will assume that the labels used in the proposed node expansion S match the worlds in K, and show that a suitable $K(S)$ is *equal* to a submodel of K. In the general case, we would construct a partial one-to-one mapping μ from the worlds in K to labels in S; its restriction to $\mathrm{Dom}(\mu)$ then defines the equivalence relation between the submodel K' and $K(S)$.
We will describe an algorithm for finding a complete open set S satisfying the requirements. At any stage in the construction of S, we suppose that $S(w)$ "magically" updates with any update of S.) For any set Ψ_w of atoms, we define $\Psi_{w,a}$, $\Psi_{w,n}$, and $\Psi_{w,p}$ as the set of propositional atoms, ν-formulas, and π-formulas in Ψ_w, respectively. Let us state the overall algorithm first, filling in details as we go:

Algorithm A.4.4 *expand-node*

Parameters:

Φ: a set of $\mathbf{K_{NNF}}$-formulas

K: a Kripke model $K = (W, R, V)$ so that $K, w_0 \models_k \Phi$

Returns:

a complete open node expansion S

begin

set $S := w_0\,\Phi$

while there exists $w\,\varphi \in S$ to which some rule can be applied **do**

prop-expand (S, w, K)

set $\Psi_w :=$ atoms$(S(w))$

foreach $\Diamond\,\pi_0$ **in** $\Psi_{w,p}$ **do**

pi-expand (S, w, K, π_0)

done

foreach $\Box\ \nu_0$ **in** $\Psi_{w,n}$ **do**
nu-expand (S, w, ν_0)
done
done
return S
end

The three subroutines mentioned in this algorithm all update S based on some tableau rule applications: Assuming that $K, w \models_k S(w)$ (which will be one of our invariants stated below), *prop-expand*(S, w, K) constructs a tableau rooted in S, using only (α)- and (β)-rule applications on formulas labelled with w, until all branches are saturated wrt the (α)- and (β)-rules. Then Proposition A.3.6 shows that every minimal atomic cover of $S(w)$ occurs on some saturated node. (This node is currently a leaf node, as we have not applied any other rules yet.) One of these minimal covers must be a subset of the cover Ψ (for which $K, w \models_k \Psi$) warranted by Theorem 3.3.1. The node corresponding to this cover is taken to be the new S.

Again assuming that $K, w \models_k \Diamond\ \pi_0$, *pi-expand*($S$, w, K,π_0) finds a w-successor w' so that $K, w' \models_k \pi_0$, and applies the (π)-rule if applicable, using w' as the new label. (That is, the two expressions $w\ R\ w'$ and $w'\ \pi_0$ are added to S.) Finally, *nu-expand*(S, w, ν_0) finds all w' so that $w\ R\ w' \in S$, and applies the (ν)-rule using w', provided it is applicable. That is, $w'\ \nu_0$ is added to S. Notice that no information from K is needed for this.

In the algorithm, the **while** loop continues as long as S is not complete, and in each iteration some rule is applied, at least to the formula $w\ \varphi$. Since the tableau calculus itself terminates after a finite number of rule applications, the algorithm itself must terminate. We will now state and prove a few invariants of the algorithm. Most of the proofs are induction-style, that is, we show that the invariants hold in the initial set S, and that they are preserved as S gets updated through either of the subroutines *prop-expand*, *pi-expand*, and *nu-expand*:

1. Each w occurring in S also occurs in W, and for each $w\ R\ w' \in S$, we also have $(w, w') \in R$.

   The existence of w_0 in W was given; no other world, and no relation pair, occurs in the initial set S. The only subroutine through which a new label w' and an r-expression $w\ R\ w'$ are introduced is *pi-expand*, where we explicitly stated that w' exists in W and is an R-successor of w.

2. For each $w\,\varphi \in S$, we have $K, w \models_k \varphi$.

   Initially, all elements in S are of the form $w_0\,\varphi$, $\varphi \in \Phi$. For these the statement is true since $K, w_0 \models_k \Phi$. In *prop-expand* (which we treat as one monolithic step, although it may consist of several rule applications) on label w, we chose the new node S so that the cover (which the algorithm later identifies as Ψ_w) satisfies Theorem 3.3.1. This guarantees that $K, w \models_k \varphi$ for any $\varphi \in \Psi_w$, and since Ψ_w is now a cover for $S(w)$, $K, w \models_k \varphi$ for any other formula in Ψ_w. Only formulas labelled w were added to S. But Theorem 3.3.1 also shows that $K, w' \models_k \nu_0$ for all $\Box\ \nu_0 \in \Psi_{w,n}$ and all w-successors w'. By Property (1), these w' include all labels w' on which the (ν)-rule will ever be applied to $w\ \Box\ \nu_0$ in the entire course of deriving the node expansion, which guarantees that the property is preserved through application of *nu-expand*. Finally for the (π)-rule, $K, w' \models_k \pi_0$ was a prerequisite for choosing w' in *pi-expand*, so there is nothing more to show.

3. There is no w such that $w\,\bot \in S$.

   This follows immediately from (2). If $w\,\bot \in S$, then $K, w \models_k \mathit{false}$, which is impossible.

4. The valuation V, restricted to the worlds occurring in S, satisfies the inclusion relation (A.1).

   In other words, we must show that whenever $w\,p \in S$, then $w \in V(p)$, and whenever $w\,\neg p \in S$, then $w \notin V(p)$. But this is straightforward since $w\,p \in S$ implies $K, w \models_k p$ by (2), which by definition of $\models_k$ implies $w \in V(p)$. Similarly for $\neg p$.

5. S is open.

   This follows from (3) and the fact that the inclusion relation (A.1) verified in (4) prohibits any $w\,p$ and $w\,\neg p$ from both being in S for the same w.

At the end of the algorithm, we thus have a complete (by the termination condition on the **while** loop) open (by (5)) node, all whose labels and relation pairs occur in K (by (1)), and the valuation V, restricted to the worlds occurring in S, satisfies (A.1) (by (4)). This shows that the restriction of K to worlds and arcs in S is a suitable $K(S)$, as we needed to show. □

The important consequence of this theorem is that it gives us an upper bound on the size of a complete node expansion in terms of the smallest Kripke model.

Corollary A.4.5 *Given a Kripke model for a set Φ with n worlds, we can find a complete open node expansion of $w_0\,\Phi$ which mentions at most n different labels.*

Again the actual size of the complete open node depends on its representation. The additional clutter due to intermediate propositional subformulas created by propositional rule applications in the course of the *prop-expand* subroutine contributes a factor of at most $|\Phi|$. In fact, these intermediate formulas need not even be stored, so if we only store atoms, the factor reduces to $d(\Phi)$. (We may still need to store k subformulas of $\Box^k\ p$, for instance.) Since the number of worlds in K is by far the dominating factor in the size of the model, we can make a statement analogous to that in Section 4.5, namely that the minimal size of a tableau for Φ with node merging is essentially bounded by the size of a minimal Kripke model for Φ.

While this expressive power of the calculus with node merging is remarkable and desirable, there remains the problem of unsoundness (which is the likely reason why this calculus has never been proposed elsewhere). We will talk about some known and possible remedies in the next section. We would like a similar result for our safe (and sound) fallback, the standard tableau calculus. It turns out that strict models provide the right level of expressivity to compare with:

Theorem A.4.6 *Given any set Φ of $\mathbf{K_{NNF}}$-formulas, if $K = (W, R, V)$ is a* strict *Kripke model for Φ, then Φ has a complete open node expansion S in the standard calculus so that $K' \cong K(S)$ for some submodel K' of K and a suitable completion $K(S)$.*

Proof: Apart from a slight clarification in the *prop-expand* and *pi-expand* subroutines, to be introduced later, we use the same algorithm as given in the proof of Theorem A.4.2 to obtain the complete open node S. However, this proof hinges on a subtle but important difference in how the (*pi*)-rule is applied. Remember that we must always use a world label w' which does not already exist in S; on the other hand, we wish to guarantee that every world label introduced into S already exists in K. The—still valid—fact $K, w \models_k S(w)$ does not suffice to guarantee this; we need to derive a stronger invariant, namely that $K, w \models_c S(w)$. We do this through a couple of preparatory steps:

1. After the **while** loop for a label w is finished, all formulas in $w\,S(w)$ are fully expanded in S.

After the *prop-expand* subroutine is executed, all propositional formulas in $S(w)$ are fully expanded. After all calls of *pi-expand*, all π-formulas are also expanded, and for the remainder of the loop no more r-expressions will be introduced. Hence, after *nu-expand* has been called on all ν-formulas, these too are fully expanded. Notice that neither *pi-expand* nor *nu-expand* introduce any new formulas of the form $w\,\varphi$ either, so at the end of the loop, all formulas in $S(w)$ are fully expanded.

2. The **while** loop is executed at most once for each w occuring in S, and each $S(w)$ gets updated at most twice: first when it gets introduced, and the second time when *prop-expand* is executed on it.

   We already showed in Proposition A.3.1 that $G(S)$ is a tree. Consider the root $w = w_0$ first. $S(w_0)$ gets introduced right at the beginning of the program. Since w_0 cannot occur on the right-hand side of an r-expression, no new formulas labelled w_0 can be introduced into S by application of (ν)- or (π)-rules. Moreover, as the **while** loop is run on w_0 (early on in the algorithm), *prop-expand* is executed on $S(w_0)$, with the result of $S(w_0)$ being propositionally expanded too.

   Now let $w \neq w_0$, and assume the claimed invariant is true for all ancestors of w. Since $K(S)$ is a tree, w has only one predecessor; call it $\hat{w}$. While the **while** loop is run on $\hat{w}$ (which by the induction hypothesis happens only once), one call of *pi-expand* will introduce the label w into S, and the subsequent calls of *nu-expand* will introduce further formulas labelled w. After this, *pi-expand* and *nu-expand* are never called again on $\hat{w}$, so no new formulas labelled w will be introduced through (ν)- or (π)-rules. So $S(w)$ must remain unchanged until the **while** loop is run on w itself, as part of which $S(w)$ gets propositionally expanded. Thereafter, since $S(w)$ is fully expanded, the **while** loop will not be called on w again; more generally, no more formula can be introduced into $S(w)$. By induction on the structure of $K(S)$, the claimed invariant is true for all w in S.

3. For every w occurring in S, we have $K_w, w \models_c S(w)$.

   We show that this invariantly holds both times $S(w)$ gets updated according to (2). Let us assume that $K_w, w \models_c S(w)$ at the beginning of the **while** loop. This is certainly true for $S(w_0) = \Phi$, since $K, w \models_c \Phi$ is a premise of this theorem. We modify *prop-expand* slightly, in that we pick an atomic cover Ψ_w in accordance with

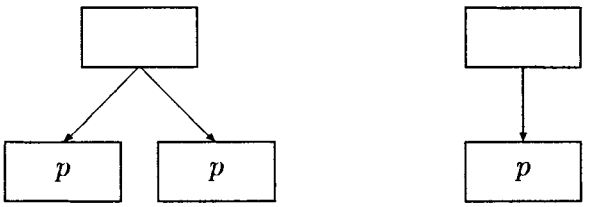

Figure A.2: *A strict model (left) and the model generated by S (right) for the set Φ in Example A.4.7.*

Definition 3.4.1, and select the new node S accordingly. Note that Ψ_w is just as well an atomic cover of the expanded $S(w)$ after execution of *prop-expand*, so Definition 3.4.1 in reverse shows that $K_w, w \models_c S(w)$ holds unchanged. Continuing execution of the **while** loop, we find that *pi-expand* and *nu-expand* produce new sets $S(w')$ of the form $\{\pi_0\} \cup \{\nu_0 : \Box\ \nu_0 \in \Psi_{w,n}\}$ for each $\Diamond\ \pi_0 \in \Psi_{w,p}$. But these are exactly the sets for which Definition 3.4.1 states $K_{w'}, w' \models_c S(w')$; moreover, these w' can be chosen *distinct* from each other (and distinct from any other world in S, since K is a tree model). We thus modify *pi-expand* slightly, stating that w' is chosen as the distinct R-successor of w in K_w as guaranteed above. Since hitherto the label w' does not occur in S, this is a valid application of the (π)-rule obeying the side conditions. (We remark as an aside that all $\Diamond\ \pi_0 \in \Psi_{w,p}$ are distinct, and this implies easily that the (π)-rule is indeed applicable to every $\Diamond\ \pi_0 \in \Psi_{w,p}$. But this observation is not essential. Contrast this with Example A.4.7 below.) Moreover, we have shown $K_{w'}, w' \models_c S(w')$ for all w' which got newly introduced for the duration of the **while** loop, which proves this invariant.

This theorem now follows analogously to Theorem A.4.2. We upheld the invariant (1) of Theorem A.4.2 that every label and every r-pair in S also occurs in K. Since every strict model is a model, invariants (4) and (5) of Theorem A.4.2 hold unchanged. Therefore, once the algorithm terminates, S is a complete open node, and the restriction of K to worlds and arcs in S is a suitable $K(S)$, which was to be shown. □

It may appear that the standard calculus and strict Kripke models have the *same* degree of expressivity. However, this is not true:

Example A.4.7 Given the set $\Phi = \{\Box\ p, \Diamond\ \top, \Diamond\ p\}$, a minimal strict model consists

of the root world with two successor worlds, both of which are in $V(p)$. However, by applying the (π)-rule to $\Diamond\ \top$ first, followed by the (ν)-rule on $\Box\ p$, we obtain the set $S = \{w_0\ \Box\ p, w_0\ \Diamond\ \top, w_0\ \Diamond\ p, w_1\ \top, w_0\ R\ w_1, w_1\ p\}$, in which w_0 has only one successor world. (See Figure A.2.) Notice that by using the (ν)-rule before the second (π)-rule, we introduced a formula $w_1\ p$ which makes the (π)-rule inapplicable to $w_0\ \Diamond\ p$. At this stage S is complete, since all formulas are expanded.

This example shows that complete open nodes may have strictly fewer labels than any strict models. So the standard calculus is more expressive than strict models, albeit only slightly.

Bibliography

[1] Carlos Areces, Hans de Nivelle, and Maarten de Rijke. Prefixed resolution: A resolution method for modal and description logics. In Harald Ganzinger, editor, *Automated Deduction—CADE-16 (Trento, Italy, 1999)*, number 1632 in LNAI, pages 187–201. Springer-Verlag, 1999.

[2] Peter Baumgartner. FDPLL – A First-Order Davis-Putnam-Logeman-Loveland Procedure. In David McAllester, editor, *Automated Deduction—CADE-17*, number 1831 in LNAI, pages 200–219. Springer-Verlag, 2000.

[3] Peter Baumgartner, Norbert Eisinger, and Ulrich Furbach. A confluent connection calculus. In Harald Ganzinger, editor, *Automated Deduction—CADE-16 (Trento, Italy, 1999)*, number 1632 in LNAI, pages 329–343. Springer-Verlag, 1999.

[4] Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. Darwin: A theorem prover for the model evolution calculus. In Stephan Schulz, Geoff Sutcliffe, and Tanel Tammet, editors, *IJCAR Workshop on Empirically Successful First Order Reasoning (ESFOR (aka S4))*, Electronic Notes in Theoretical Computer Science, 2004. `http://www.mpi-sb.mpg.de/~baumgart/publications/darwin.pdf`.

[5] Peter Baumgartner and Cesare Tinelli. The model evolution calculus. In F. Baader, editor, *Proceedings of the 19th International Conference on Automated Deduction, CADE-19 (Miami, Florida, USA)*, number 2741 in LNAI, pages 350–364. Springer-Verlag, 2003.

[6] Paul Beame, Henry Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, 2004.

[7] Bernhard Beckert and Rajeev Goré. Free variable tableaux for propositional modal logics. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 1997)*, number 1227 in LNAI, pages 91–106. Springer-Verlag, 1997.

[8] Bernhard Beckert and Rajeev Goré. System description: leanK 2.0. In *Automated Deduction—CADE-15*, number 1421 in LNAI, pages 51–55. Springer-Verlag, 1998.

[9] Bernhard Beckert and Rajeev Goré. Free variable tableaux for propositional modal logics. *Studia Logica*, 69:59–96, 2001.

[10] Daniel Le Berre, Laurent Simon, and Armando Tacchella. Challenges in the QBF arena: the SAT'03 evaluation of QBF solvers, 2003. `http://satlive.org/QBFEvaluation/eval03.pdf`.

[11] Wolfgang Bibel. *Automated Theorem Proving.* Vieweg, Braunschweig/Wiesbaden, Germany, 1982.

[12] Jean-Paul Billon. The disconnection method. a confluent integration of unification in the analytic framework. In *Proceedings of the 5th International Workshop TABLEAUX 1996 (Terrasini, Italy)*, number 1071 in LNAI, pages 110–126. Springer-Verlag, 1996.

[13] Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified Boolean formulas. *Information and Computation*, 117:12–18, 1995.

[14] R. Carnap. Modalities and quantification. *Journal of Symbolic Logic*, 11:33–64, 1946.

[15] Brian F. Chellas. *Modal Logic—an Introduction.* Cambridge University Press, 1980.

[16] Marcello d'Agostino. Tableau methods for classical propositional logic. In M. d'Agostino, D. M. Gabbay, R. Hähnle, and J. Posegga, editors, *Handbook of Tableau Methods*, pages 45–123. Kluwer Academic Publishers, Dordrecht/Boston/London, 1999.

[17] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.

[18] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[19] J. de Kleer. An assumption-based truth maintenance system. *Artificial Intelligence*, 28:127–162, 1986.

[20] Francesco M. Donini, Bernhard Hollunder, Mario Lenzerini, A. M. Spaccamela, Daniele Nardi, and Werner Nutt. The complexity of existential quantification in concept languages. *Artificial Intelligence*, 53:309–327, 1992.

[21] F. Fitch. Tree proofs in modal logics. *Journal of Symbolic Logic*, 31:152, 1966.

[22] Melvin C. Fitting. *Proof Methods for Modal and Intuitionistic Logics.* D. Reidel Publishing, Dordrecht, Holland, 1983.

[23] Melvin C. Fitting. *First-order Logic and Automated Theorem Proving.* Springer-Verlag, New York/Berlin/Heidelberg, 1996.

[24] Jon W. Freeman. *Improvements to propositional satisfiability search algorithms.* Ph. D. thesis, University of Pennsylvania, Philadelphia, PA, USA, 1995.

[25] Alan Frisch and Richard Scherl. A general framework for modal deduction. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings, 2nd Conference on Principles of Knowledge Representation and Reasoning*, pages 196–207. Morgan-Kaufmann, 1991.

[26] Dov M. Gabbay. *Labelled Deductive Systems*, volume 33 of *Oxford Logic Guides*. Clarendon Press, Oxford, 1996.

[27] M.R. Garey and D.S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., San Francisco, CA, 1979.

[28] J. Gaschnig. Performance measurement and analysis of certain search algorithms. Technical Report CMU-CS-79-124, Carnegie-Mellon University, 1979.

[29] Gerhard Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210 and 405–431, 1935.

[30] Lilia Georgieva, Ulrich Hustadt, and Renate A. Schmidt. A new clausal class decidable by hyperresolution. In *Automated Deduction—CADE-18*, number 2392 in LNAI, pages 260–274. Springer-Verlag, 2002.

[31] Matthew L. Ginsberg. Dynamic backtracking. *Journal of AI Research*, 1:25–46, 1993.

[32] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. Backjumping for quantified Boolean logic satisfiability. In *Proceedings of the Seventeenth International Joint Conferences on Artificial Intelligence (IJCAI'01), Seattle, Washington, USA*, pages 275–281, 2001.

[33] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. Learning for quantified Boolean logic satisfiability. In *Proceedings of the 18th Nternational Conferences on Artificial Intelligence (AAAI 2002), Edmonton, Canada*, pages 649–654, 2002.

[34] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. QBF reasoning on real world instances, 2004. `http://www.satisfiability.org/SAT04/programme/94.pdf`.

[35] Enrico Giunchiglia and Armando Tacchella. A subset-matching size-bounded cache for satisfiability in modal logics. In Roy Dyckhoff, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference(TABLEAUX 2000)*, number 1847 in LNAI, pages 237–251. Springer-Verlag, 2000.

[36] Ernesto Giunchiglia and Armando Tacchella. *SAT: a system for the development of modal decision procedures. In *Automated Deduction—CADE-17*, number 1831 in LNAI, pages 291–296. Springer-Verlag, 2000.

[37] Fausto Giunchiglia and Roberto Sebastiani. Building decision procedures for modal logics from propositional decision procedures—the case study of modal **k**. In *Automated Deduction—CADE-13*, number 1104 in LNAI, pages 583–597. Springer-Verlag, 1996.

[38] Fausto Giunchiglia and Roberto Sebastiani. Building decision procedures for modal logics from propositional decision procedures - the case study of modal $\mathbf{K}(m)$. *Information and Computation*, 162(1/2):158–178, 2000.

[39] Rajeev Goré. Tableau methods for modal and temporal logics. In M. d'Agostino, D. M. Gabbay, R. Hähnle, and J. Posegga, editors, *Handbook of Tableau Methods*, pages 297–396. Kluwer Academic Publishers, Dordrecht/Boston/London, 1999.

[40] Erich Grädel. On the restraining power of guards. *Symbolic Logic*, 64:1719–1742, 1999.

[41] Volker Haarslev and Ralf Möller. RACE system description. In *Proceedings of the International Workshop on Description Logics (DL 1999), Linköping, Sweden*, pages 130–132, 1999.

[42] Volker Haarslev and Ralf Möller. Consistency testing: the RACE experience. In Roy Dyckhoff, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference(TABLEAUX 2000)*, number 1847 in LNAI, pages 57-61. Springer-Verlag, 2000.

[43] Volker Haarslev and Ralf Möller. RACER system description. In *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR 2001)*, number 2083 in LNAI, pages 701–706. Springer-Verlag, 2001.

[44] Jens Happe. The ModProf theorem prover. In *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR 2001)*, number 2083 in LNAI, pages 459–463. Springer-Verlag, 2001.

[45] Jens Happe. A subsumption-aided caching technique. In *Issues in the Design and Experimental Evaluation of Systems for Modal and Temporal Logics (IJCAR 2001 Workshop)*, Technical Report DII 14/01, pages 49–57. Dipartimento di Ingegneria dell'Informazione, Unversitá degli Studi di Siena, Siena, Italy, 2001.

[46] Jens Happe. Efficient reasoning with labelled formula translations of $\mathbf{K}_m$. Technical report, AAAI Press, to appear.

[47] Ian Horrocks. *Optimising Tableaux Decision Procedures for Description Logics.* Ph. D. thesis, University of Manchester, 1997.

[48] Ian Horrocks. Using an expressive description logic: FaCT or fiction? In *Principles of Knowledge Representation and Reasoning: Proceedings of the 6th International Conference (KR 1998)*, pages 636–647. Morgan Kaufmann Publishers, 1998.

[49] Ian Horrocks. Benchmark analysis with FaCT. In Roy Dyckhoff, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference(TABLEAUX 2000)*, number 1847 in LNAI, pages 62–66. Springer-Verlag, 2000.

[50] Ian Horrocks. DAML+OIL: a description logic for the semantic web. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 25(1):4–9, 2002.

[51] Ian Horrocks. Implementation and optimization techniques. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 306–346. Cambridge University Press, 2003.

[52] Ian Horrocks and Peter F. Patel-Schneider. Optimising description logic subsumption. *Journal of Logic and Computation*, 9(3):267–293, 1999.

[53] Ian Horrocks, Ute Sattler, and Stefan Tobies. Reasoning with individuals for the description logic $\mathcal{SHIQ}$. In David McAllester, editor, *Automated Deduction—CADE-17*, number 1831 in LNAI, pages 482–496. Springer-Verlag, 2000.

[54] Ian Horrocks and Stefan Tobies. Reasoning with axioms: Theory and practice. In *Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 285–296, 2000.

[55] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.

[56] G.E. Hughes and M.J. Creswell. *An Introduction to Modal Logic*. Methuen & Co., London, 1968.

[57] Ulrich Hustadt and Renate A. Schmidt. MSPASS: Modal reasoning by translation and first-order resolution. In *Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2000)*, number 1847 in LNAI, pages 67–71. Springer-Verlag, 2000.

[58] Ulrich Hustadt and Renate A. Schmidt. A principle for incorporating axioms into the first-order translation of modal formulae. In *Automated Deduction—CADE-19*, number 2741 in LNAI, pages 412–426. Springer-Verlag, 2003.

[59] Michel Klein, Jeen Broekstra, Dieter Fensel, Frank van Harmelen, and Ian Horrocks. Ontologies and schema languages on the web. In Dieter Fensel, James Hendler, Henry Lieberman, and Wolfgang Wahlster, editors, *Spinning the Semantic Web: Bringing the World Wide Web to its full potential*, pages 95–140. MIT Press, 2003.

[60] Saul A. Kripke. Semantical analysis of modal logic I: Normal propositional calculi. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 9:67–96, 1963.

[61] Richard E. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM Journal of Computing*, 6(3):467–480, 1977.

[62] Reinhold Letz. Lemma and model caching in decision procedures for quantified boolean formulas. In *Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2002)*, number 2381 in LNAI, pages 160–175. Springer-Verlag, 2002.

[63] Reinhold Letz and Gernot Stenz. DCTP: A disconnection calculus theorem prover. In *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR 2001)*, number 2083 in LNAI, pages 381–385. Springer-Verlag, 2001.

[64] Reinhold Letz and Gernot Stenz. Proof and model generation with disconnection tableaux. In R. Nieuwenhuis and Andrei Voronkov, editors, *Proceedings of the International Conference on Logic, Programming, and Automated Reasoning (LPAR 2001)*, number 2250 in LNAI, pages 142–156. Springer-Verlag, 2001.

[65] Reinhold Letz and Gernot Stenz. Generalised handling of variables in disconnection tableaux. In *Automated Reasoning: Second International Joint Conference (IJCAR 2004), Cork, Ireland*, number 3097 in LNAI, pages 289–306. Springer-Verlag, 2004.

[66] Donald W. Loveland. *Automated Theorem Proving: a Logical Basis.* North-Holland, 1968.

[67] Carsten Lutz, Ute Sattler, and Stefan Tobies. A suggestion of an n-ary description logic. In *Proceedings of the International Workshop on Description Logics (DL 1999), Linköping, Sweden*, pages 81–85, 1999.

[68] João P. Marques-Silva and Karem A. Sakallah. Conflict analysis in search algorithms for propositional satisfiability. In *Proceedings of the IEEE Conference on Tools with Artificial Intelligence*, pages 467–469, 1996.

[69] Fabio Massacci. Simplification with renaming: A general proof technique for tableau and sequent based provers. Technical Report 424, Computer Laboratory, Univ. of Cambridge, UK, 1997.

[70] Fabio Massacci. Simplification: A general constraint propagation technique for propositional and modal tableaux. In *Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 1998)*, number 1397 in LNAI, pages 217–231. Springer-Verlag, 1998.

[71] Fabio Masssacci. Strongly analytic tableaux for normal modal logics. In *Automated Deduction—CADE-12*, number 814 in LNAI, pages 723–737. Springer-Verlag, 1994.

[72] Fabio Masssacci. Design and results of the TABLEAUX-99 non-classical (modal) systems comparison. In *Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 1999)*, number 1617 in LNAI, pages 14–18. Springer-Verlag, 1999.

[73] Fabio Masssacci. Single step tableaux for modal logics: Methodology, computations, algorithms. *Journal of Automated Reasoning*, 24(3):319–364, 2000.

[74] Fabio Masssacci and Francesco M. Donini. Design and results of TANCS-2000 non-classical (modal) systems comparison. In Roy Dyckhoff, editor, *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2000)*, number 1847 in LNAI, pages 52–56. Springer-Verlag, 2000.

[75] Steve Minton, M. D. Sohnston, A. B. Philips, and P. Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 17–24, 1990.

[76] G. Mints. Resolution calculi for modal logics. *American Mathematical Society Translations*, 143:1–14, 1989.

[77] David G. Mitchell. A SAT solver primer. In Yuri Gurevich, editor, *The Logic In Computer Science Column*. Microsoft Research, to appear.

[78] M. Moskewicz, C. Madigan, Y. Zhao, and L. Zhang. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC2001)*, pages 530–535, 2001.

[79] Ike Nassi and Ben Shneiderman. Flowchart techniques for structured programming. *Sigplan Notices*, 8(8), 1973.

[80] Hans-Jürgen Ohlbach. Semantics based translation methods for modal logics. *Journal of Logic and Computation*, 1(5):691–746, 1991.

[81] Hans-Jürgen Ohlbach and Renate A. Schmidt. Functional translation and second-order frame properties of modal logics. *Journal of Logic and Computation*, 7(5):581–603, 1997.

[82] H.J. Ohlbach, A. Nonnengart, M. de Rijke, and D. Gabbay. Encoding two-valued nonclassical logics into classical logic. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 21, pages 1403–1486. Elsevier Science, 2001.

[83] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, 1994.

[84] Peter F. Patel-Schneider. DLP system description. In E. Franconi, G. de Giacomo, R.M. MacGregor, W. Nutt, C.A. Welty, and F. Sebastiani, editors, *Collected Papers from the International Description Logics Workshop (DL 1998)*, pages 87–89, 1998.

[85] Peter F. Patel-Schneider. TANCS-2000 results for DLP. In Roy Dyckhoff, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference(TABLEAUX 2000)*, number 1847 in LNAI, pages 67–71. Springer-Verlag, 2000.

[86] Peter F. Patel-Schneider. What's new in DLP. In *Collected Papers from the International Description Logics Workshop (DL 2000), Aachen, Germany*, pages 227–235, 2000.

[87] A.L. Rector, W.A. Nowlan, and A.J. Glowinski. Goals for concept representation in the GALEN project. In *17th annual Symposium on Computer Applications in Medical Care, Washington, USA, SCAMC 93*, pages 414–418, 1993.

[88] Jussi Rintanen. Partial implicit unfolding in the davis-putnam procedure for quantified Boolean formulae. In R. Nieuwenhuis and Andrei Voronkov, editors, *Proceedings of the International Conference on Logic, Programming, and Automated Reasoning (LPAR 2001)*, number 2250 in LNAI, pages 362–376. Springer-Verlag, 2001.

[89] Lawrence Ryan. Efficient algorithms for clause learning SAT solvers. M. Sc. thesis, Simon Fraser University, 2004.

[90] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept description with complements. *Artificial Intelligence*, 48:1–26, 1991.

[91] R. Smullyan. *First-order Logic*. Springer-Verlag, New York, 1968.

[92] L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[93] Geoff Sutcliffe and Christian Suttner. The IJCAR ATP system competition (CASC-J2). *AI Communications*, to appear.

[94] Geoff Sutcliffe, Christian Suttner, and Jeff Pelletier. The IJCAR ATP system competition (CASC-JC). *Journal of Automated Reasoning*, 28(3):307–320, 2002.

[95] Stefan Tobies. PSPACE reasoning for graded modal logics. *Journal of Logic And Computation*, 11(1):85–106, 2001.

[96] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedings of the International Conference on Computer Aided Design (ICCAD 2001), San Jose, California*, pages 279–285, 2001.

Glossary of Notation

Notation	Description	
$\#K$	the number of worlds in Kripke structure K	42
$\|K\|$	the absolute size of a Kripke structure K	42
$\|\|\lambda\|\|$	the normalization of complex label λ	131
$\|\|\Sigma\|\|_\sigma$	normalization of Σ: ensures $\sigma\ {}_p\sqsubseteq\ \xi$ for all $\xi \in \Sigma$	131
$\|\Phi\|$	the number of atoms in Φ	37
$\#S$	the number of formulas in S	60
$\|S\|$	the absolute size of the set of formulas S	60
$\uplus$	the disjoint union of sets	44
$\cong$	isomorphism relation between two Kripke structures	44
$\equiv_g$	equivalence of formulas in any prop. logic	14
$\equiv_k$	equivalence of formulas in $\mathbf{K_{NNF}}$	43
$\equiv_l$	equivalence of formulas in $\mathcal{LBL}$	62
$\prec$	is less constrained than (of search states)	253
$\preceq$	an order on nogood premises (for undoing branches)	212
$\preceq$	is finer than (of prop. covers)	23
$\models$	a general satisfiability relation	12
$\approx$	a general semantic relation on structures and formulas	12
$\models_c$	strict satisfiability relation for modal logic $\mathbf{K}$(in NNF)	46
$\approx_g$, $\models_g$	semantic/satisfiability relations for any prop. logic	13
$\models_k$	satisfiability relation for modal logic $\mathbf{K}$(in NNF)	43
$\approx_l$,$\models_l$	weak semantic/satisfiability relations for $\mathcal{LBL}$ (flavours $\mathcal{LBL}_w$ and $\mathcal{LBL}_x$)	60, 67, 157
$\models_p$	satisfiability relation for propositional logic (in NNF)	32
$\approx_{pp}$, $\models_{pp}$	alternative semantic relation for propositional logic (in NNF), using sets of atoms as models	34
$\approx_s$, $\models_s$	strong semantic/satisfiability relations for $\mathcal{LBL}_s$	159

Notation	Description	
$\models_s$,$\models_s$	strong semantic/satisfiability relation for $\mathcal{LBL}_s$	166
$\lfloor\sigma_1, \sigma_2\rfloor$	the lower mgu of two simple labels, if it exists	128
(σ_1, σ_2)	the mgu of two simple labels, if it exists	58
(λ_1, λ_2)	the mgu of two complex labels, if it exists	132
$\lceil\sigma_1, \sigma_2\rceil$	the upper mgu of two simple labels, if it exists	128
$\sqsubseteq$	the instance relation: $\sigma \sqsubseteq \sigma'$ iff σ' instantiates σ	58
$\sqsubset$	the strict instance relation: $\sigma \sqsubset \sigma'$ iff $\sigma \sqsubseteq \sigma'$ and $\sigma \neq \sigma'$	58
$\sqsubseteq$	instance relation: $\lambda \sqsubseteq \sigma'$ iff $\sigma \sqsubseteq \sigma'$ but $\xi \not\sqsubseteq_p \sigma'$, all $\xi \in \Sigma$	129
${}_p\sqsubseteq$	instance relation: $\lambda\ {}_p\sqsubseteq \sigma'$ iff $\sigma\ {}_p\sqsubseteq \sigma'$ but $\xi \not\sqsubseteq_p \sigma'$, all $\xi \in \Sigma$	129
${}_p\sqsubseteq$	instance relation: $\sigma\ {}_p\sqsubseteq \sigma'$ iff σ' instantiates a prefix of σ	128
$\sqsubseteq_p$	instance relation: $\lambda \sqsubseteq_p \sigma'$ iff $\lambda \sqsubseteq \sigma''$ for some prefix σ'' of σ'	129
$\sqsubseteq_p$	instance relation: $\sigma \sqsubseteq_p \sigma'$ iff a prefix of σ' instantiates σ	128
$*$	an anonymous placeholder, represents an arbitrary constant	57
$\emptyset$	the inconsistent label (with no instances)	57
(α)	a tableau rule for conjunctions	284
a	a propositional atom	60
α	a conjunctive formula of the form $\alpha_1 \wedge \ldots \wedge \alpha_n$	14
B	a set of branches, esp. those taken in the model finder algorithm	203
(β)	a tableau rule for disjunctions	284
b	a branch index (nonneg. integer), also u. f. the branch itself	202
$\beta(b)$	the term over which b branches	202
β	a disjunctive formula of the form $\beta_1 \vee \ldots \vee \beta_m$	14
$[c]$	a conditional constant in a label	128
c	an (existential) constant from a domain D, used in labels	57, 128
$C(b)$	the clause of branch b	202
χ	a mapping from $\mathcal{LBL}$-formulas to constants, aka Gödelization	77
CNF	clause (or conjunctive) normal form	10
cwr	constant-wise realizable (of labels)	71, 153

Notation	Description	
D	a domain of constants to be used in labels	57
Δ	a dependency set, esp. the premises of a nogood	212, 227
$d(\cdot)$	the depth of a formula or set of formulas	17, 43, 60
D^*	the set of all strings over domain D	58
DAG	directed acyclic graph	48, 114
DPLL	The Davis-Putnam-Logemann-Loveland procedure [18, 17]	256
$\Delta(B)$	the union of $Delta^+(B)$ and $Delta^-(B)$	253
$\Delta^+(B)$	the σ: b branched upon in a search state)	253
$\Delta^-(B)$	the σ: b blocked in a search state)	253
ε	the empty label (of length 0)	57
egi	exception-generated instance (of a label)	139
ENF	expanded And/Or normal form	189
FOL	First-order logic	1
Γ	a universe of ground labels, closed under prefixes	58, 129
$\gamma, \gamma', \ldots$	ground labels, consisting of (existential) constants only	57
$\mathcal{G}(\lambda, \Gamma)$	the set of ground instances of λ in Γ	129
$\mathcal{G}_p(\lambda, \Gamma)$	the set of ground prefix instances of λ in Γ	129
$\mathcal{G}(\sigma, \Gamma)$	the set of ground instances of σ in Γ	58
$G(S)$	a Kripke frame spanned by a set of prefixed formulas and r-expressions S	287
$I(M)$	the set of rgls of a set of assertions	147, 164
$I(\lambda, M)$	the set of rgis of a complex label λ	147
$I(S)$	the set of rgls of a set of $\mathcal{LBL}$-formulas	63
$I(\sigma, S)$	the set of rgis of a simple label σ	63
K	a Kripke structure (or model) (W, R, V)	42
K	the smallest basic normal modal logic	41
κ	a mapping from labels to modalities, u. f. defining labellings	76
$K(M)$	the Kripke model generated by a set of assertions	78
$\mathbf{K_{NNF}}$	the restriction of **K** to formulas in NNF	41

Notation	Description	
$K(S)$	a Kripke model generated by a set of prefixed formulas and r-expressions S	287
$K\|_{W'}$	the submodel of K induced by set W' of worlds	44
K_w	the submodel of K induced by world w and its successors	44
L	identifier for a generic, usu. propositional logic	277
L	identifier for a generic, usu. propositional, logic	12
$\bar{l}$	the complement of a literal	32
λ	identifier for labels with exceptions	129
l	a propositional literal	32
$\lambda\, a$	standard notation for an assertion with complex label, a is a prop. atom	133
$\Lambda(b)$	the set of labels (or scope) associated with a branch	202
λ: b	branch b applied over label λ	201
$\mathcal{LBL}$	the logic of labelled formulas, object of our study	60
$\mathcal{LBL}_s$	the strong logic $\mathcal{LBL}$: $\models_s$ relation and complex labels in assertions	127, 159
$\mathcal{LBL}_w$	the weak logic $\mathcal{LBL}$: $\models_l$ relation and simple labels in assertions	127
$\mathcal{LBL}_x$	the extended logic $\mathcal{LBL}$: $\models_l$ relation and complex labels in assertions	127, 157
LCNF	labelled clause normal form	190
$\mathcal{L}(\mathbf{L})$	The language of a logic (set of well-formed formulas)	12
$\lambda(\varphi_0)$	the standard translation of a QBF-formula φ_0 into $\mathcal{LBL}$	102
M,M'	sets of assertions, u. f. semantic structures in $\mathcal{LBL}$	60
M^-	a counterinstance against $M \models_l F$ for some formula F	92
$\mathcal{M}$	a semantic structure; sometimes u. f. a model	12
$M(B)$	the assertions introduced by B, including default branches	204
$M_0(B)$	the assertions introduced by the set of branches B	203
mgu	most general unifier, esp. of simple or complex labels	58
$M_{\langle\sigma\rangle}$	the set obtained from M by iteratively subscripting the positions of σ	133, 163
μ	bijection between worlds of two Kripke structures, defining an isomorphism	44

Notation	Description	
$M_{\langle x \rangle}$	the subscripted set $\{(\sigma - \Sigma_{\langle x \rangle})\, a : (x\sigma - \Sigma)\, a \in M$ or $(*\sigma - \Sigma)\, a \in M\}, \varepsilon \notin \Sigma_{\langle x \rangle}$	133
$M_{\langle x \rangle}$	the subscripted set $\{(\sigma - \Sigma_{\langle x \rangle})\, a : \varepsilon \notin \Sigma_{\langle x \rangle}, (x\sigma - \Sigma)\, a \in M$ or $(*\sigma - \Sigma)\, a \in M\}$	163
(ν)	a tableau rule for formulas of the form $\Box\ \nu_0$	284
$\mathbb{N}_0$	the set of nonnegative integers	17
NG	generic name for a nogood	212
NNF	negation normal form	11
ν	a modal formula of the form $\Box\ \nu_0$	43
Ω	a set of conclusions of a nogood-like expression, esp. a nogood	212
OWL	Web Ontology Language	2
P	The set of propositional variables	12
(π)	a tableau rule for formulas of the form $\Diamond\ \pi_0$	284
p, q	propositional variables	31
$P_c(\sigma', M)$	a property: for any $\gamma \in I(\sigma', M)$, $\gamma c \in I(M)$	153
Φ	a set of formulas in any prop. logic, esp. in **K**	13
$\varphi(p_1, \ldots, p_n)$	a propositional formula inside a QBF (its matrix)	100
φ, ψ	formulas in any prop. logic, esp. in **K**	13
π	a modal formula of the form $\Diamond\ \pi_0$	43
$\mathbf{PL_{NNF}}$	the logic of propositional formulas in negation normal form	32
$\mathbf{\Pi}_k^p$	a complexity class, corresponding to $\overline{\mathbf{QBF}}_k$	102
pref(S)	the set of all prefixes of labels occurring in S	60
pref(Σ)	the set of all prefixes of labels in Σ	57
Ψ	propositional covers of a set of formulas, also: sets of atoms	23
Ψ_A	a set of propositional atoms, also u. f. a semantic structure in $\mathbf{PL_{NNF}}$	34
Ψ_A, Ψ_N, Ψ_P	the propositional atoms, ν-formulas, and π-formulas in an atomic cover of a $\mathbf{K_{NNF}}$-formula	44
PSPACE	the complexity class of problems solvable in polynomial space	48, 101
QBF	quantified Boolean formula	100
$\mathbf{QBF}_k$	the class of QBF with k quantifier alterations	100

Notation	Description	
$\overline{\mathbf{QBF}}_k$	the class of dual problems to $\mathbf{QBF}_k$	102
$Q_c(\sigma', M)$	a property: for any $\sigma' \sqsubseteq \gamma$, $\gamma \in I(M \cup \{\gamma\, \top\})$	154
$Q(F)$	a formal property on $\mathcal{LBL}$-formulas	62
Q_j	the quantifiers of a QBF: either $\exists$ or $\forall$	100
$Q(\mathcal{M})$	a formal property on semantic structures	16
$Q(\varphi)$	a formal property on formulas	16
$Q(\sigma)$	a formal property on simple labels	60, 140
$Q(\sigma, M)$	a formal property on simple labels and sets of assertions	142
(R)	a generic tableau rule	278
R	an accessibility relation between worlds in a Kripke structure	42
ρ	a (nogood) substitution	216
rgi	realized ground instance	63, 147
rgl	realized ground label	63, 147, 164
$\rho(\sigma_i \mapsto \sigma_i')$	the substitution by uniform instantiation of σ_i into σ_i'	217
$\rho(* \mapsto *_t, \sigma_i^p)$	the substitution induced by re-indexing of wildcards	217
$\rho(*)$	the index-eliminating substitution	217
$R(W)$	the R-successors of a world W	42
S	identifier for sets of $\mathcal{LBL}$-formulas or prefixed formulas	60, 278
T	a (usu. atomic) prop. cover of a set of $\mathcal{LBL}$-formulas S	69
Σ	a set of simple labels, esp. exceptions to a main label	57
Σ	a set of simple labels, particularly exceptions	129
$[\sigma]$	σ, with all constants replaced by conditional constants	128
σ	a simple label, consisting of constants and $*$	128
$\sigma, \sigma', \ldots$	simple labels, consisting of constants and $*$	57
T_A, T_N	the unlabelled and labelled formulas in an atomic cover of an $\mathcal{LBL}$-formula	69
$\sigma\, a$	standard notation for an assertion, a is a prop. atom	60
σ: b	a nogood premise, see also λ: b	212
$S_{\langle c \rangle}$	the subscripted set $\{\sigma\, F : c\sigma\, F \in S$ or $*\sigma\, F \in S\}$	60
scwr	strongly constant-wise realizable (of labels)	154
$\sigma\, F$	standard notation for a labelled formula	60
$S_{\langle \gamma \rangle}$	the set obtained from S by iteratively subscripting the positions of γ	60

Notation	Description	
$s(M)$	the saturation of M (conversion from $\mathcal{LBL}_w$- to $\mathcal{LBL}_s$-model)	168
Σ_k^p	a complexity class, corresponding to $\mathbf{QBF}_k$	101
$\Sigma_{\langle\sigma\rangle}$	the set of labels obtained from Σ by iteratively subscripting the positions of σ	59
$\sigma - \Sigma$	a complex label: main label σ, exceptions Σ	129
σS	the set of augmented formulas $\{\sigma\sigma' F : \sigma' F \in S\}$	60
$S(w)$	a shorthand for $\{\varphi : w\,\varphi \in S\}$	278
$\Sigma_{\langle x\rangle}$	the subscripted set $\{\sigma : x\sigma \in \Sigma \text{ or } *\sigma \in \Sigma\}$	59
T, T'	identifiers for a tableau	278
τ	a mapping between two logics, esp. a homomorphism	20
$U(C)$	the unbranched labels (or default branch) of a clause C	204
V	a propositional or modal valuation function	31, 42
V_w	a propositional valuation function, defined by the valuation V on world w	42
W	the worlds in a Kripke structure	42
$w(M)$	the expansion of M (conversion from $\mathcal{LBL}_x$- to $\mathcal{LBL}_w$-model)	174
$w\,\Phi$	a shorthand for $\{w\,\varphi : \varphi \in \Phi\}$	278
$w\,\varphi$	a prefixed formula on a tableau node	277
$w\,R\,w'$	an r-expression on a tableau node	277
ξ	a simple label, used as an exception in another label	129
x	a position in a label, either a constant or $*$	57
$x\Sigma$	the set of labels augmented by x: $\{x\sigma : \sigma \in \Sigma\}$	59