SYMMETRIES IN AUTOMATED REASONING

ALEJANDRO EZEQUIEL ORBE

THE CASE OF MODAL LOGICS AND SATISFIABILITY MODULO THEORIES

PhD. Thesis
Facultad de Matemática, Astronomía y Física
Universidad Nacional de Córdoba

Advisor: Carlos Areces

Marzo 2014

A Ceci y Nacho.

A mi vieja y mi viejo.

A mis hermanos Ruben, Fernando y Roberto.

ABSTRACT

The study of symmetries has received much attention during the last years in the automated reasoning community, especially in propositional satisfiability solving (SAT solving), as they can help in solving many hard problems. In automated reasoning, the presence of symmetries in a problem's search space may increase the difficulty of finding a solution by forcing a search algorithm to explore symmetric subspaces that do not contain solutions. Intuitively, if a problem has symmetries and we are able to identify them, we might use them to reduce the difficulty of reasoning by directing a search algorithm to look for solutions only in non-symmetric parts of the search space.

In this thesis we investigate symmetries for modal logics and Satisfiability Modulo Theories (SMT).

For modal logics, we develop the theoretical framework for the study of symmetries in modal logics by generalizing the notion of symmetries of propositional formulas in conjunctive normal form to modal formulas. We prove two key results for the basic modal logic: that symmetries of a basic modal formula partition the model space into equivalence classes such that each equivalence class contains only models or only non-models, and that symmetries can be used as an inference mechanism, and therefore, they can be used to strengthen existing reasoning mechanisms. Then we extend these results to a broad range of modal logics using the framework of coinductive modal models and introduce a more flexible notion of symmetry, called layered symmetry, for those modal logics that have the tree model property.

We then present two graph-based symmetry detection algorithms to detect symmetries in modal formulas and evaluate them empirically in modal benchmarks.

Finally, we present a modal tableau calculus for the basic modal logic with a blocking mechanism that takes advantage of symmetry information about the input formula to restrict the application of the $(\Diamond)$ rule. We prove completeness of the calculus and evaluate it empirically in different modal benchmarks.

For SMT, we focus on the symmetry detection problem. We develop a graph-based symmetry detection algorithm that is able to detect general symmetries involving uninterpreted symbols. We implement the algorithm and evaluate it empirically on several benchmarks from the SMT-LIB.

PUBLICATIONS

Some ideas and figures have appeared previously in the following publications:

- C. Areces, and E. Orbe. *Dealing with Symmetries in Modal Tableaux*. In Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX). Nancy, Francia. 2013.
- C. Areces, D. Deharbe, P. Fontaine, and E. Orbe. *SyMT: finding symmetries in SMT formulas*. 11th International Workshop on Satisfiability Modulo Theories (SMT 2013), Helsinki, Finlandia. 2013
- C. Areces, G. Hoffmann, E. Orbe. *Symmetries in Modal Logics*. Post-Proceedings of the Seventh Workshop on Logical and Semantic Frameworks, with Applications, Rio de Janeiro, Brasil, vol. 113. 2013.
- C. Areces, G. Hoffmann, E. Orbe. *Symmetries in Modal Logics: A Coinductive Approach*. Seventh Workshop on Logical and Semantic Frameworks, with Applications, Rio de Janeiro, Brasil. 2012.

ACKNOWLEDGMENTS

I would like to thank my advisor, Carlos Areces, for guiding me during the last three years and for the thousands of advices that he shared with me. I am pretty sure that guiding me wasn't an easy task, and required a lot of patience and time from Carlos. Thanks a lot Carlos.

I will always be in debt with Gabriel Infante-Lopez. Gabriel was the one who gave me the opportunity to enter the academic world and encouraged me to go for my Phd. Thanks for the talks, advices and your friendship.

Academy also gave me the chance to get to know some of my dearest friends. In particular I would like to thank to Martin, Camper and Diego, for the many "asados" that we have shared and the talks about science and life. I am very confident that we still have a lot of "asados" to eat. I don't want to forget to mention Franco, Chun, Guillaume and Raul, thank you guys for helping me through these five years.

This thesis would have not been possible without the support of my family. Thanks Vieja and Viejo for everything, for being there whenever I needed, for all the love that you gave me. Viejo, I know that you must be really proud, wherever you are. I really miss you. Many thanks to my brothers, Ruben, Fernando y Roberto. You, as usual, have been a constant source of advice and I just don't know what would I do without you.

Finally, I want to thank the loves of my life: Ceci and Nacho. You are the source of my many smiles, and the ones who had to deal with me when things didn't go as expected. All this wouldn't be possible without you. I love you.

Córdoba, 13/02/2014.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Part I

INTRODUCTION

"Symmetry as wide or as narrow as you may define its meaning, is one idea by which man through the ages has tried to comprehend and create order, beauty and perfection."

Hermann Weyl. *Symmetry*. 1952

1 SYMMETRIES IN AUTOMATED REASONING

Symmetry is a familiar notion for us: we recognize it when we see it. The term symmetry is used in a very broad sense not only as a mathematical notion, but as something bridging disciplines, cultures, sciences and arts. Since it first uses by the ancient Greeks the concept of symmetry has been shaped in a long, slow process (see [Brading and Castellani, 2013; Darvas, 2007; Hon and Goldstein, 2008] for an historic review on the evolution of the concept of symmetry).

In its most general setting, we can speak about symmetry if "under any kind of transformation at least one property of the object is left invariant" [Darvas, 2007]. However, it is the group-theoretic definition of symmetry the one that has been proved most useful, and upon which this thesis rely.

Informally, we can state the group-theoretic definition of symmetry as "invariance under a specified group of transformations" [Brading and Castellani, 2013]. This definition unveils a close connection between the notions of symmetry, equivalence and group. A symmetry group induces a partition, of the elements that are exchanged, into equivalence classes such that elements exchanged with one another by the symmetry transformations are related by an equivalence relation, thus forming an equivalence class.

Symmetry has many uses. We can not only study the symmetric properties of an object (geometric, mathematical, etc.) to understand its behavior, but also derive specific consequences regarding the object under study based on its symmetry properties, i. e., using a "symmetry-based argument".

This type of argument is of common usage in mathematical reasoning. Mathematical proofs sometimes state that a certain assumption can be made "without loss of generality". This phrase suggests that although making the assumption at first sight only proves the theorem in a more restricted case, this does nevertheless justify the theorem in full generality [Harrison, 2009].

The idea underlying this type of argument is that structurally similar problems (i. e., symmetrically related problems) must receive correspondingly similar solutions, i. e., that a solution must respect the symmetries of the problem [van Fraassen, 1989].

Therefore, if one can identify the symmetries of a problem, one might use them to reduce the difficulty of reasoning by analyzing in detail only one of the symmetric problems (cases) and then generalizing the result to the others. This is exactly what we try to do when using the symmetries of a problem in automated reasoning.

Many problem classes and, in particular, those arising from real world applications, display a large number of symmetries. In automated reasoning, the presence of symmetries in a problem's search space may increase the difficulty of finding a solution by forcing a search algorithm to explore symmetric subspaces that do not contain solutions. If we are able to recognize that such symmetries exist, we can use them to direct a search algorithm to look for solutions only in non-symmetric parts of the search space, thus reducing the overall difficulty [Sakallah, 2009].

Now, what do we mean when we talk about the "symmetries of a problem" in classical automated reasoning? We can define a symmetry of a problem as a permutation of its variables (or literals) that preserves its structure (its syntactic form) and, hence, its set of solutions (its models).

Depending on which aspect of the problem is kept invariant, we classify symmetries into *semantic* or *syntactic* [Benhamou and Sais, 1992]. *Semantic symmetries* are intrinsic properties of the function that are independent of any particular representation, i. e., a permutation of variables that does not change the value of the function under any variable assignment. *Syntactic symmetries*, on the other hand, correspond to the specific algebraic representation of the function, i. e., a permutation of variables (or literals) that does not change the representation. A syntactic symmetry is also a semantic symmetry, but the converse does not always hold.

Whereas interest in semantic symmetries was primarily motivated by the desire to optimize the design of logic circuits and to speed up their verification [Sakallah, 2009], interest in syntactic symmetries arose in the context of constraint solving, specifically Propositional Satisfiability (SAT).

The earliest reference to the potential benefits of using syntactic symmetries in logical reasoning is attributed to Krishnamurthy [Krishnamurthy, 1985] who introduced the *principle of symmetry* (or *symmetry rule*) to strengthen resolution-based proof systems for propositional logic and showed that, using it, it is possible to obtain short proofs for certain "tricky mathematical arguments" (e. g., checkboard puzzles, Ramsey's theorem and the pigeonhole principle [Urquhart, 1999]).

The intuition underlying the symmetry principle is that in the course of a mathematical proof, one often uses an arbitrary element of a set as a representative of the set, provided the set possesses sufficient symmetry so that the ensuing arguments equally apply to all other elements of the set. In a similar spirit, this principle allows one to recognize that a tautology remains invariant under certain permutations of variable names and uses that information to avoid repeated independent derivations of intermediate formulas that are merely permutational variants of one another.

Actually [Krishnamurthy, 1985] introduces two inference rules: the *global symmetry rule* and the *local symmetry rule*. Given a set of propositional clauses Γ (i. e., a set of sets of propositional literals) and a permutation σ that maps literals to literals, if we derive a clause C from Γ, the global symmetry rule allows us to infer the clause $\sigma(C)$ provided that σ is a symmetry of Γ (i. e., $\sigma(\Gamma) = \Gamma$). On the other hand, the local symmetry rule allows us to infer $\sigma(C)$ provided that for every clause A in Γ used in the derivation of C, $\sigma(A)$ is also in Γ.

Notice that the difference between the two rules lies in the fact that the former requires that the permutation maintains invariant the whole set of clauses Γ, while the latter only requires that the symmetric clauses of clauses used in a derivation belong to the set of clauses Γ. Thus, the local symmetry rule may be applicable even in cases where the global symmetry rule is not and, therefore, it is strictly more powerful than the global symmetry rule [Arai and Urquhart, 2000].

In spite of showing the usefulness of the symmetry rules for some problems [Krishnamurthy, 1985] presents no algorithm to search for the symmetries of a set of clauses. Neither does it discuss in detail how the resolution procedures should use the symmetry rules.

The first reference on how to use symmetries in a backtrack search algorithm is due to [Brown *et al.*, 1989]. In this work, symmetries are used dynamically by the search algorithm to restrict its search to the equivalence classes induced by the symmetries of the formula which are assumed to be given.

In [Benhamou and Sais, 1992; Benhamou and Sais, 1994] a symmetry detection algorithm and search algorithms to exploit symmetries are presented. Symmetry detection is done directly on the input formula by deriving formula-preserving variable permutations incrementally (see Chapter 6.2 for more details). Once the symmetries are detected, the authors show how to use them in two search algorithms: the SLRI [Cubadda, 1988] algorithm (a variant of the SL resolution algorithm [Kowalski and Kuehner, 1972]) and the semantic evaluation [Oxusoff, 1989] algorithm.

For both algorithms, the key to profit from symmetries during deduction is a property that establishes that a literal l has a model in a set of clauses S (i. e., there is a valuation v satisfying S such that $v(l) = true$) if and only if for every symmetric literal l' (i. e., every literal such that $\sigma(l) = l'$ for a symmetry σ) there exists a model v' of S such $v(l') = true$. For example, in the case of SLRI, instead of refuting each literal separately, using symmetries it is possible to refute symmetric literals simultaneously, e. g., let $C = \{l_1, l_2, \ldots, l_{i-1}, l_i, \ldots, l_n\}$ be a clause, and suppose that the literals $l_1, l_2, \ldots, l_{i-1}$ have been refuted and that l_i is the literal we try to refute. If l_i is symmetric to one of the previous literals, then it will be directly refuted by the symmetry property. The more symmetric literals in a clause, the bigger the cut in the resolution tree that we can make. Similarly, in semantic evaluation if the algorithm assigns the value true or false to a literal and it generates the empty clause, then we insert in the model that is being built the opposite of that literal, together with all the other opposites of its symmetric literals.

In [Crawford, 1992] a different approach to symmetry detection is presented. It shows that symmetry detection can be polynomially reduced to the colored graph automorphism detection problem and present a reduction algorithm to create a colored graph from a propositional formula in conjunctive normal form (CNF). Then the symmetry group (in fact, a subgroup of the symmetry group) of the formula is detected by detecting the automorphism group of the resulting graph (see Chapter 6.1 for more details). [Crawford, 1992] also shows how to assess the effects of symmetries on the set of truth assignments by using Polya's Theorem [Pólya and Read, 1987] to count the number of equivalence classes in the set of truth assignments under a set of symmetries of a formula. However no practical algorithm for exploiting symmetries is provided.

Later, in [Crawford *et al.*, 1996], a method for exploiting symmetries is presented. Instead of modifying the search algorithm to use symmetry information, the method modifies the problem being solved by adding a *symmetry-breaking predicate (SBP)* for each symmetry. These predicates are built in such a way that they are true in exactly one valuation in each of the equivalence classes induced by the symmetry group; the resulting formula is equisatisfiable to the original. An ordering on the set of variables is established and used to construct a lexicographic order on the set of assignments. Then, the symmetry-predicates are built in such a way that they are true of only the smallest model (the *lex-leader*) within each equivalence class under this ordering. Intuitively, we can think of the valuations as binary numbers, and the predicates performing a bit-wise comparison between models and selecting the smaller (as binary numbers) of the two.

Given that, in general, breaking every symmetry in the symmetry group, i. e., doing *complete symmetry-breaking*, can result in a formula exponentially larger than the original one (because of the size of the symmetry group and the presence of redundant clauses in the SBPs), the authors propose to break just some of the symmetries, i. e., to do *partial symmetry-breaking* by pruning a *symmetry tree* built from the symmetries in the symmetry group. This however does not always prevent redundant clauses neither does it ensures that the resulting formula is only polynomially larger than the original.

In [Aloul *et al.*, 2003b] improvements on the work in [Crawford, 1992; Crawford *et al.*, 1996] are presented. They present new graph constructions that are able to detect more symmetries (see Chapter 6.1) than the construction in [Crawford, 1992; Crawford *et al.*, 1996]. They also improve on the symmetry-breaking predicate generation. Similarly to [Crawford *et al.*, 1996], this work explores partial symmetry-breaking by building symmetry breaking predicates that do not select only the smallest valuations to satisfy them. Symmetry-breaking predicates are built in a per-symmetry basis, only doing so for the irredundant set of generators of the symmetry group of the formula [Seress, 1997] computed using their symmetry detection technique. By breaking generator symmetries only, they do not break all symmetries, but they achieve significant pruning. Symmetry-breaking predicates are formulated in terms of cycles of a permutation, i. e., the predicates are built for each individual cycle in a symmetry, instead of the entire set of variables. This results in smaller SBPs. Later, in [Aloul *et al.*, 2003a; Aloul *et al.*, 2006] a more systematic and efficient construction of symmetry-breaking predicates is presented. This construction takes into account the cycle structure of symmetry generators, which typically involves very few variables, to drastically reduce the size of the generated SBPs.

A different approach to exploit symmetries is presented in [Sabharwal, 2005]. It presents a modification to the SAT solver zChaff [Moskewicz *et al.*, 2001] that allows it to branch on a set of symmetric variables rather than on single variables. For instance, given a set $\{x_1, x_2, \ldots, x_k\}$ of k symmetric variables, a $k+1$-way branch sets $x_1, \ldots, x_i$ to 0 and $x_{i+1}, \ldots, x_k$ to 1 for each $i \in [0, k]$. This reduces the number of partial assignments that must potentially be explored from 2^k to $k+1$. The identification of symmetric variables is assumed to be available from high-level descriptions of a problem and provided to the solver as an additional input.

In [Benhamou *et al.*, 2010], a technique that combines symmetry reasoning with clause learning [Ryan, 2004] in Conflict-Driven Clause Learning SAT solvers [Een and Sörensson, 2003] is presented. The idea is to augment clause learning by using the symmetries of the problem to learn the symmetric equivalents of conflict-induced clauses. Here the focus is not on guiding the search algorithm directly but instead enriching a complementary process to provide more information to the search algorithm.

So far, we have provided a brief survey of the most important works on symmetries in SAT solving. However much more research has been done on the subject. Despite their differences, we can group all of them into two different categories: *dynamic symmetry breaking* [Brown *et al.*, 1989; Benhamou and Sais, 1992;

Benhamou and Sais, 1994; Sabharwal, 2005] and *static symmetry breaking* [Crawford, 1992; Crawford *et al.*, 1996; Aloul *et al.*, 2003b; Aloul *et al.*, 2003a; Aloul *et al.*, 2006].

In *dynamic symmetry breaking* the idea is to use symmetry information to *dynamically* restrict the search space, while in *static symmetry breaking*, symmetries are eliminated from the problem statement before using a SAT solver, in a preprocessing step. The former is solver dependent but it can take advantage of symmetries that emerge during search, while the latter can be used with any solver. Despite their differences they share the same goal: to identify symmetric branches of the search space and guide the SAT solver away from symmetric branches already explored.

Symmetries have been also extensively investigated and successfully exploited in other domains. In Constraint Satisfaction Problem (CSP) (see [Gent *et al.*, 2006] for a detailed survey) much work has being done, in particular, to provide correct definitions of what exactly are the symmetries of a constraint programming problem [Cohen *et al.*, 2005]. In Integer Programming, algorithms have been adapted to exploit large symmetry groups [Margot, 2002; Margot, 2003]. In Planning, attempts to integrate symmetry reasoning to a state-of-the-art planner have been done in [Fox and Long, 1999; Fox and Long, 2002], and it has been pointed out that symmetry detection during search, and the presence of "almost-symmetries", is very important due to the nature of planning problems. There have also been substantial efforts in Model Checking over some years [Clarke *et al.*, 1996; Ip and Dill, 1996; Sistla *et al.*, 2000; Bošnački *et al.*, 2002]. These works have tended to assume that users recognize the symmetry, and they have also been limited to simple cases of symmetry rather than using the power of computational algebra. In Quantified Boolean Formulas (QBF), [Audemard *et al.*, 2004] presents a hybrid approach that handles QBF and symmetry-breaking predicates. In the context of QBF pure-propositional SBPs cannot be conjunctively added to the original formula, because, as variables are quantified, one can obtain clauses with all its variables universally quantified, and consequently the QBF becomes not valid. To avoid such drawback, the authors proposed to consider the SBP formula as independent from the QBF one. A hybrid solver is then designed to deal simultaneously with the QBF and the SBP formulas. They also present a symmetry detection algorithm for QBF formulas that extends the graph-based detection algorithms used in SAT solving. A more practical SBP generation technique for QBF is presented in [Audemard *et al.*, 2007a; Audemard *et al.*, 2007b].

1.1 SYMMETRIES IN MODAL LOGICS AND SATISFIABILITY MODULO THEORIES

As the keen reader will have noticed, this thesis is about symmetries in modal logics and Satisfiability Modulo Theories. So a natural question would be: what is the situation about symmetry research in these domains? Let us begin to answer it.

In the case of modal logics research has been done on how to exploit symmetries in model checking for the temporal logic LTL [Clarke *et al.*, 1996; Donaldson and Miller, 2005; Miller *et al.*, 2006; Donaldson, 2007], and temporal-epistemic logic [Cohen *et al.*, 2009]. However, to the best of our knowledge, the use of symmetries in satisfiability and automated theorem proving for modal logics remains largely unexplored.

The situation in SMT is somehow better as there have been attempts to exploit symmetries in SMT-based model checking [Audemard *et al.*, 2002b] and SMT solving [Roe, 2006; Déharbe *et al.*, 2011] (see Chapter 4.4 for more details). In particular in [Déharbe *et al.*, 2011] an algorithm for detecting and breaking symmetries is presented and implemented in the state-of-the-art SMT solver veriT [Bouton *et al.*, 2009]. However, the symmetry detection technique is rather heuristic and the symmetry breaking technique works for only certain classes of problems.

In this thesis we have two main objectives. First, we aim at providing a complete presentation on how to exploit symmetries in modal logics by providing the theoretical background to use symmetries, and techniques to detect and use them.

Second, for SMT, we focus on improving the work in [Déharbe *et al.*, 2011], by developing a general detection technique that is able to detect more symmetries than the techniques known so far.

2 THE BASIC MODAL LOGIC

In this chapter we introduce the Basic Modal Logic ($\mathcal{BML}$). We start by presenting its syntax and semantics (Section 2.1). Then we present the concept of bisimulations, that is key to investigate the expressive power of the logic (Section 2.2), and investigate the relation between basic modal logic and first-order logic by presenting the standard translation (Section 2.3). Then we turn our attention to the computational properties of the basic modal logic, by presenting its decidability properties and its computational complexity (Section 2.4). We finish the chapter by presenting a number of extensions (Section 2.5).

For a comprehensive treatment and for references to the extensive literature on the subject we refer the reader to [Blackburn *et al.*, 2001; Blackburn *et al.*, 2006].

2.1 SYNTAX AND SEMANTICS

The basic modal language is a propositional language extended with sentential operators, i. e., operators that take a sentence to deliver another. We call this operators *modalities* or *modal operators*.

Definition 2.1 (Syntax). *Let* $\mathsf{PROP} = \{p_1, p_2, \ldots\}$ *be a countable infinite set of propositional variables and* $\mathsf{MOD} = \{m, m'', \ldots\}$ *a set of modality symbols. We call the pair* $\mathcal{S} = \langle \mathsf{PROP}, \mathsf{MOD} \rangle$ *a* modal signature *(or* similarity type*). The set of basic modal formulas* FORM *over the signature* $\mathcal{S}$ *is defined as*

$$\mathsf{FORM} ::= p \mid \neg\varphi \mid \varphi \vee \psi \mid [m]\varphi,$$

where $p \in \mathsf{PROP}$, $m \in \mathsf{MOD}$, *and* $\varphi, \psi \in \mathsf{FORM}$. $\top$ *and* $\bot$ *stand for an arbitrary tautology and contradiction, respectively. We will also use classical connectives such as* $\wedge, \rightarrow$ *taken to be defined in the usual way. For each* $m \in \mathsf{MOD}$ *we have a* dual operator $\langle m \rangle$ *(diamond) defined as* $\langle m \rangle \varphi = \neg[m]\neg\varphi$. *When* MOD *is a singleton, i. e., in the monomodal case, we simply write* $\Box$ *and* $\Diamond$ *for the box and diamond operators.*

The diamond and box operators can be read in several ways, and different readings suggest different semantics and proof systems. For example, the formula $\Box\varphi$ can be read as "necessarily φ", and, under this reading $\Diamond\varphi$ can be read as "it is not necessarily that not φ", i. e., "it is *possibly* the case that φ". In the context of *epistemic logic*, where the basic modal language is used to reason about knowledge, $\Box\varphi$ (usually written as $K\varphi$) is read as "the agent knows that φ". Finally, in *provability logic*, $\Box\varphi$ is read as "it is *provable* (in some arithmetical theory) that φ". To make this readings clearer let us introduce the semantics of the basic modal logic.

We can think of the basic modal logic as a tool for talking about structures or models. What these structures are depends on the semantic interpretation we are interested in. In this work we focus on the relational semantics, also known as Kripke semantics [Kripke, 1959; Kripke, 1963], of modal logics. Under this view, modal languages let us describe and reason about relational structures (graphs).

Definition 2.2 (Models)**.** *A (Kripke) model $\mathcal{M}$ is a triple $\mathcal{M} = \langle W, \{R^m\}_{m \in \mathsf{MOD}}, V\rangle$ such that:*

i) *W, the* domain*, is a non empty set. Elements of W are called* points, states, worlds, *etc.*

ii) *Each R^m, an* accessibility relation*, is a binary relation on W.*

iii) *V, the* valuation*, is a function that assigns to each element $p \in \mathsf{PROP}$ a subset $V(p) \subseteq W$. Informally, we think of $V(p)$ as the set of states where the propositional symbol p is true.*

The first two components of $\mathcal{M}$ ($\langle W, \{R^m\}_{m \in \mathsf{MOD}}\rangle$) are called the frame *underlying the model. In the mono-modal case we simply write $\mathcal{M} = \langle W, R, V\rangle$.*

Remark. So far, we have defined the syntax and models for the multi-modal case, i. e., for an arbitrary number of modalities. For simplicity sake, in what follows we restrict ourselves to the mono-modal case, i. e., to the case where MOD is a singleton and just one relation exists in the model. Nevertheless, the results we present extend naturally to the multi-modal case.

Let us define the notion of a formula φ being satisfied (or true) in a model $\mathcal{M}$ at point w.

Definition 2.3 (Semantics)**.** *Let $\mathcal{M} = \langle W, R, V\rangle$ be a model and φ a modal formula. We inductively define the notion of φ being* satisfied *(true) in $\mathcal{M}$ at point $w \in W$ as*

$$\begin{array}{lll} \mathcal{M}, w \models p & \textit{iff} & w \in V(p) \textit{ for } p \in \mathsf{PROP}, \\ \mathcal{M}, w \models \neg\varphi & \textit{iff} & \mathcal{M}, w \not\models \varphi, \\ \mathcal{M}, w \models \varphi \vee \psi & \textit{iff} & \mathcal{M}, w \models \varphi \textit{ or } \mathcal{M}, w \models \psi, \\ \mathcal{M}, w \models \Box\varphi & \textit{iff} & \mathcal{M}, v \models \varphi, \textit{for all } v \textit{ s.t. } wRv. \end{array}$$

Definition 2.4 (Semantic Consequence)**.** *A formula φ is a* semantic consequence *of a set of formulas Σ if for all models $\mathcal{M}$ and all points w in $\mathcal{M}$, if $\mathcal{M}, w \models \Sigma$ then $\mathcal{M}, w \models \varphi$, and in such case we write $\Sigma \models \varphi$. Instead of writing $\{\varphi\} \models \psi$ we write $\varphi \models \psi$.*

A formula φ is *globally satisfied* in a model $\mathcal{M}$ if it is satisfied at all points in $\mathcal{M}$. If this is the case we write $\mathcal{M} \models_g \varphi$. A formula φ is *valid* if its globally satisfied in all models. If this is the case we write $\models \varphi$. A formula is *satisfiable in a model* $\mathcal{M}$ if there exists a point in $\mathcal{M}$ in which φ is satisfied and φ is *satisfiable* if there exists a point in a model at which it is satisfied. These definitions are lifted to sets of formulas in the obvious way.

Example 2.1. *Consider the model $\mathcal{M} = \langle W, R, V\rangle$ of Figure 2.1, where $W = \{w_1, w_2, w_3, w_4, w_5\}$, w_iRw_j if and only if $j = i + 1$, $V(p) = \{w_2, w_3\}$ and $V(q) = \{w_1, w_2, w_3, w_4, w_5\}$. In this model we have that $\mathcal{M}, w_1 \models \Diamond\Box p$, $\mathcal{M}, w_1 \not\models \Diamond\Box p \to p$, $\mathcal{M}, w_2 \models \Diamond(p \wedge q)$ and $\mathcal{M}, w_1 \models q \wedge \Diamond(q \wedge \Diamond(q \wedge \Diamond(q \wedge \Diamond q)))$.*

Definition 2.3 highlights some interesting facts. First, notice the internal character of the semantic definition of the basic modal logic: modal formulas are evaluated *inside* models at some particular point. A modal formula is like an automaton placed inside a structure at some point w, and forced to explore the structure by making transitions to accessible points. Second, we can think of diamond and boxes as macros that encode quantification over R-accessible states in a variable-free notation. Third, note that Kripke models can be thought of as directed graphs.

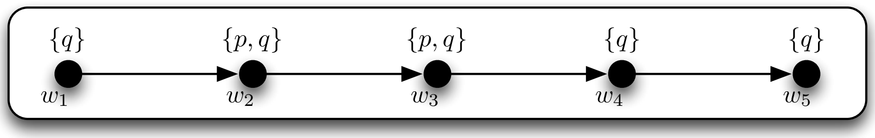


Figure 2.1: Model $\mathcal{M} = \langle W, R, V \rangle$.

2.2 EXPRESSIVE POWER

The expressive power of a language is usually measured in terms of the distinctions it can draw. In modal logics, the basic concept to investigate expressivity is that of a *bisimulation* between two models.

2.2.1 *Bisimulations*

The notion of bisimulation, first formulated for modal languages in [van Benthem, 1977; van Benthem, 1983], states that two models related by a bisimulation are *modally equivalent*, i. e., they make the same modal formulas true.

Definition 2.5 (Bisimulation). *A bisimulation between models $\mathcal{M} = \langle W, R, V \rangle$ and $\mathcal{M}' = \langle W', R', V' \rangle$ is a non-empty relation $Z \subseteq W \times W'$ such that whenever wZw' we have the following properties:*

i) $w \in V(p)$ iff $w' \in V'(p)$, for all $p \in$ PROP. *[Atomic Harmony]*

ii) If wRv then there exists $v' \in W'$ such that vZv' and $w'R'v'$. *[Zig]*

iii) If $w'R'v'$ then there exists $v \in W$ such that vZv' and wRv. *[Zag]*

If there is a bisimulation between two models $\mathcal{M}$ and $\mathcal{M}'$ then we say that $\mathcal{M}$ and $\mathcal{M}'$ are bisimilar *and write $\mathcal{M} \,\underline{\leftrightarrow}\, \mathcal{M}'$. Moreover, we say that two states w and w' are bisimilar and write $w \,\underline{\leftrightarrow}\, w'$ if they are related by some bisimulation.*

Intuitively, a bisimulation between two models relates points such that both carry the same atomic information (atomic harmony) and whenever it is possible to make a transition in one model, it is possible to make a matching transition in the other (zig and zag).

Example 2.2. *Figure 2.2 shows three models: $\mathcal{M}$, $\mathcal{N}$ and $\mathcal{R}$ with distinguished points w_1, v_1 and u_1 respectively.*

We are interested in bisimulations relating the distinguished points in each model. It is easy to verify that the relation

$$Z = \{(w_1, v_1), (w_3, v_1), (w_2, v_2), (w_4, v_2)\}$$

is a bisimulation between models $\mathcal{M}$ and $\mathcal{N}$ such that $w_1 \,\underline{\leftrightarrow}\, v_1$. However, there is no bisimulation between models $\mathcal{N}$ and $\mathcal{R}$ such that $v_1 \,\underline{\leftrightarrow}\, u_1$: a move from u_1 to u_3 has no matching move in $\mathcal{N}$. We cannot move from v_1 to v_2, because v_2 has no successor (is an endpoint) whereas u_3 has it, neither move reflexively from v_1 to itself, as one can move from v_1 to an endpoint, which cannot be done in u_3.

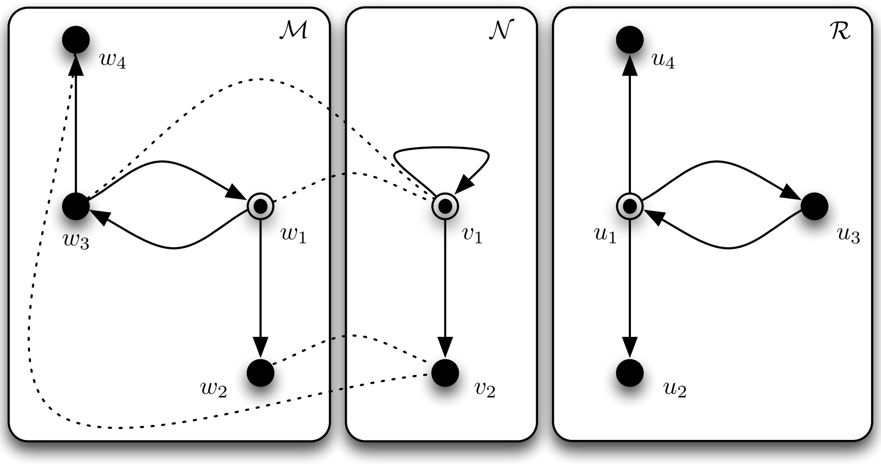


Figure 2.2: Bisimulations between models.

An important consequence of two models being related by a bisimulation is that they make the same formulas true, or more formally, that modal satisfiability is *invariant under bisimulations*. First let us define the notion of two points being *(modally) equivalent*.

Definition 2.6 (Modal Equivalence). *Let $\mathcal{M}$ and $\mathcal{M}'$ be models of the same signature $\mathcal{S}$, and let w and w' be states in $\mathcal{M}$ and $\mathcal{M}'$ respectively. The $\mathcal{S}$-theory of w is the set of all $\mathcal{S}$-formulas satisfied at w: $\{\varphi \mid \mathcal{M}, w \models \varphi\}$. We say that w and w' are* (modally) equivalent *(notation:$w \leftrightsquigarrow w'$) if they have the same $\mathcal{S}$-theories. The $\mathcal{S}$-theory of the model $\mathcal{M}$ is the set of all $\mathcal{S}$-formulas satisfied by all states in $\mathcal{M}$: $\{\varphi \mid \mathcal{M} \models \varphi\}$. Models $\mathcal{M}$ and $\mathcal{M}'$ are called (modally) equivalent (notation:$\mathcal{M} \leftrightsquigarrow \mathcal{M}'$) if their $\mathcal{S}$-theories are identical.*

Proposition 2.1 (Bisimulation Invariance Lemma). *Let $\mathcal{M} = \langle W, R, V\rangle$ and $\mathcal{M}' = \langle W', R', V'\rangle$ be models of the same signature $\mathcal{S}$. Then, for every $w \in W$ and $w' \in W'$, $w \underline{\leftrightarrow} w'$ implies $w \leftrightsquigarrow w'$.*

Proof. By induction on φ. The case for propositional symbols follows directly by atomic harmony. The Boolean cases are straightforward from the inductive hypothesis. For formulas of the form $\Box\varphi$ we have that $\mathcal{M}, w \models \Box\varphi$ iff for all v such that wRv, $\mathcal{M}, v \models \varphi$. Then we have to analyze two cases: a) if there is no v such that wRv, as $w \underline{\leftrightarrow} w'$ there is no v' such that $w'R'v'$ and $\mathcal{M}', w' \models \Box\varphi$ trivially; b) if there is v such that wRv and $\mathcal{M}, v \models \varphi$, then as $w \underline{\leftrightarrow} w'$, there exists v' such that $w'R'v'$ and $v \underline{\leftrightarrow} v'$. By the inductive hypothesis, $\mathcal{M}', v' \models \varphi$ and therefore, $\mathcal{M}', w' \models \Box\varphi$. The converse direction follows a similar argument. $\Box$

Proposition 2.1 states that bisimulation implies modal equivalence. But what about the converse: does modal equivalence imply bisimilarity? In the general case the answer is *no*. However, the result holds for *image-finite* models.

Definition 2.7 (Image-finite Model). *A model $\mathcal{M}$ is* image-finite *if for each state w in $\mathcal{M}$ and each relation R in $\mathcal{M}$ the set $\{v \mid wRv\}$ is finite.*

Proposition 2.2 (Hennessy-Milner Theorem). *Let $\mathcal{M}$ and $\mathcal{M}'$ be two image-finite models. Then, for every $w \in W$ and $w' \in W'$, $w \underline{\leftrightarrow} w'$ if and only if $w \leftrightsquigarrow w'$.*

Proof. The direction from left to right follows from Proposition 2.1. For the right to left direction we will prove that the relation $\leftrightsquigarrow$ of modal equivalence is itself a bisimulation. We define the bisimulation relation Z by wZw' iff $w \leftrightsquigarrow w'$. We now verify that Z is indeed a bisimulation. It is immediate that Z satisfies atomic harmony. Let us prove by contradiction that the zig condition holds. Assume that wZw' and wRv and there is no v' in $\mathcal{M}'$ such that $w'R'v'$ and vZv'. Let $S' = \{u' \mid w'R'u'\}$. Now, as w has an R-successor, we have that $\mathcal{M}, w \models \Diamond\top$. As wZw', we have $\mathcal{M}', w' \models \Diamond\top$ too, hence S' is non-empty. Furthermore, as $\mathcal{M}'$ is image-finite, S' must be finite too, so we can write it as $\{u'_1, \ldots, u'_n\}$. By assumption, for every $u'_i \in S'$ there exists a formula ψ_i such that $\mathcal{M}, v \models \psi_i$ but $\mathcal{M}', u'_i \not\models \psi_i$. It follows that

$$\mathcal{M}, w \models \Diamond(\psi_1 \wedge \ldots \wedge \psi_n) \text{ and } \mathcal{M}', w' \not\models \Diamond(\psi_1 \wedge \ldots \wedge \psi_n),$$

which contradicts our assumption that wZw'. Hence Z satisfies zig. A similar argument holds for the zag condition. Therefore Z is a bisimulation. $\square$

What Proposition 2.2 tells us is that, on finite models, the expressive power of the basic modal language matches up exactly with bisimulation invariance.

2.2.2 *Model Constructions*

Bisimulation is a powerful tool that can be used in a number of ways. In particular we can use it to build models from other models. In this thesis, we are interested in three model constructions based on bisimulations: model unravelling, disjoint unions and generated submodels (see [Blackburn *et al.*, 2001; Blackburn *et al.*, 2006] for more model constructions).

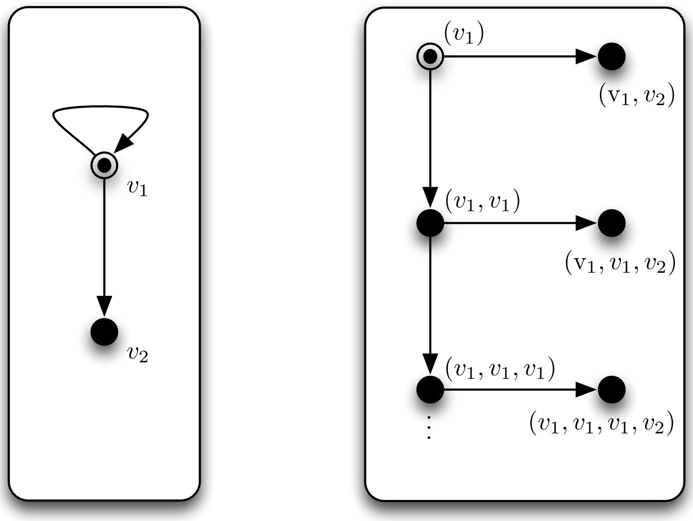
MODEL UNRAVELLING Reasoning about trees is often easier than reasoning about arbitrary graphs, so it would be of considerable utility to find a way to represent arbitrary models as trees without losing information in the process.

Model unravelling [Sahlqvist, 1975] is a model construction based on bisimulations that does exactly that: given a model $\mathcal{M}$, it constructs a tree-like model $\mathcal{M}'$ bisimilar to the original model $\mathcal{M}$. Let us define this construction more formally.

Definition 2.8 (Paths in a model). *Given a model $\mathcal{M} = \langle W, R, V\rangle$ and a point $w_0 \in W$, a (finite)* path rooted at w_0 *is a sequence $\pi = (w_0, v_1, \ldots, v_n)$ such that $0 \leq n$, and there exists a path $w_0Rv_1 \ldots Rv_n$ from w_0 to v_n in $\mathcal{M}$. For a path $\pi = (w_0, v_1, \ldots, v_n)$ we define $first(\pi) = w_0$, $last(\pi) = v_n$, and $length(\pi) = n$. For a path $\pi = (w_0, w_1, \ldots, w_n)$ and a point $v \in W$ such that w_nRv, by $\pi v = (w_0, w_1, \ldots, w_n, v)$ we denote the extension of π by v. We denote the set of all paths rooted at w_0 as $\Pi[w_0]$.*

Definition 2.9 (Model Unravelling). *Given a model $\mathcal{M} = \langle W, R, V\rangle$ and a point $w_0 \in W$, the* unravelling of $\mathcal{M}$ around w_0 *is the model $\mathcal{T}(\mathcal{M}) = \langle \Pi[w_0], R', V'\rangle$ where $R' = \{(\pi, \pi') \mid$ exists $v \in W$ s.t. $\pi v = \pi', \pi \neq \pi'\}$, for $\pi, \pi' \in \Pi[w_0]$, and $\pi \in V'(p)$ if and only if $last(\pi) \in V(p)$.*

Notice that model unravelling constructs a tree model by treating paths trough $\mathcal{M}$ as first-class citizens.



(a) $\mathcal{M}$. (b) $\mathcal{T}(\mathcal{M})$.

Figure 2.3: Unravelling of model $\mathcal{M}$.

Example 2.3. *Figure 2.3a shows an arbitrary model $\mathcal{M}$. Its unravelling around the point v_1 is the infinite comb-like model $\mathcal{T}(\mathcal{M})$ of Figure 2.3b.*

It is easy to verify that for any model $\mathcal{M}$, $\mathcal{M}$ and $\mathcal{T}(\mathcal{M})$ are bisimilar, e. g., the mapping $f : (w_0, v_1, \ldots, v_n) \mapsto v_n$ defines a bisimulation. A direct consequence of this is the so called *tree model property*.

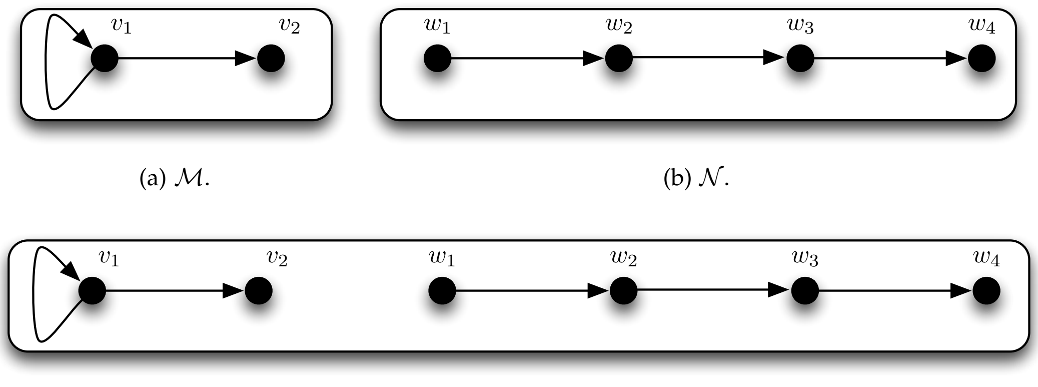
Corollary 2.1 (Tree model property). *Every satisfiable modal formula in the basic modal logic is satisfiable in a tree-like model.*

DISJOINT UNIONS The idea underlying disjoint union, is that given a set of models, we can combine them to construct a bigger model by putting the component models side by side without any relational link between them. A condition to do so is that the models need to be *disjoint*, i. e., their domains must contain no common elements. If the given models are not mutually disjoint then we first take mutually disjoint isomorphic copies and then form the disjoint union of the copies.

Definition 2.10 (Disjoint Unions). *Given mutually disjoint models $\mathcal{M}_i = \langle W_i, R_i, V_i \rangle$ ($i \in I$, for I an indexing set), their* disjoint union *is the structure $\biguplus \mathcal{M}_i = \langle W, R, V \rangle$, where $W = \bigcup_{i \in I} W_i$, $R = \bigcup_{i \in I} R_i$ and for all $p \in \mathsf{PROP}$, $V(p) = \bigcup_{i \in I} V_i(p)$.*

It follows directly from Definition 2.10 that any component model $\mathcal{M}_i$ of the disjoint union $\mathcal{M}$ is bisimilar to $\mathcal{M}$, e. g., just take the identity relation as the bisimulation relation.

Example 2.4. *Figures 2.4a and 2.4b show the disjoint models $\mathcal{M}$ and $\mathcal{N}$. Figure 2.4c shows their disjoint union $\mathcal{M} \uplus \mathcal{N}$. The model $\mathcal{M} \uplus \mathcal{N}$ gathers together all the information in the two smaller models unchanged: no links between models are added nor the valuations modified.*



(a) $\mathcal{M}$. (b) $\mathcal{N}$.

(c) $\mathcal{M} \uplus \mathcal{N}$.

Figure 2.4: Disjoint union of models $\mathcal{M}$ and $\mathcal{N}$.

generated submodels Disjoint unions are useful to make bigger models from smaller ones, generated submodels do the reverse. Generated submodels enable us to throw points away from a satisfying model without compromising satisfiability. The intuition behind this is that, in basic modal logic, satisfiability is only concerned with the set of points reachable from the evaluation point (see Definition 2.3).

Definition 2.11 (Generated Submodels). *Let $\mathcal{M} = \langle W, R, V\rangle$ and $\mathcal{M}' = \langle W', R', V'\rangle$ be two models: we say that $\mathcal{M}'$ is a* submodel *of $\mathcal{M}$ if $W' \subseteq W$, R' is the restriction of R to $\mathcal{M}'$ ($R \restriction W'$) and V' is the restriction of V to $\mathcal{M}'$ ($V \restriction W'$). $\mathcal{M}'$ is a* generated submodel *of $\mathcal{M}$ (notation $\mathcal{M}' \rightarrowtail \mathcal{M}$) if $\mathcal{M}'$ is a submodel of $\mathcal{M}$ and for all the points w the following closure condition holds: if w is in $\mathcal{M}'$ and wRv, then v is in $\mathcal{M}'$.*

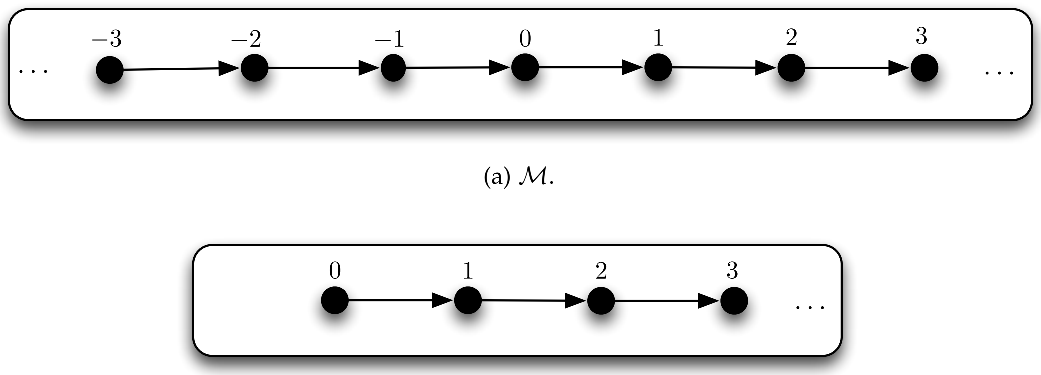
A generated submodel is bisimilar to the model that gave rise to it, e. g., the identity relation relates the two models in the appropriate way.

Example 2.5. *Figure 2.5a shows the model $\mathcal{M} = \langle \mathbb{Z}, <, V\rangle$, i. e., the integers with their usual order and an arbitrary valuation. Figure 2.5b shows the generated submodel $\mathcal{M}^+$ that we obtain by omitting the negative numbers and restricting the valuation to the numbers that remain. It is clear that $\mathcal{M}^+$ is closed under the accessibility relation and that every formula φ that holds in a point m of $\mathcal{M}$ holds at the same point n in $\mathcal{M}^+$ because the only points that are relevant to the satisfiability of φ are those greater than n, and all such points belong to $\mathcal{M}^+$.*

2.3 modal logic and first-order logic

So far we have presented the basic modal logic as an isolated formal system, however, it has a tight relation with first-order logic.

Consider a Kripke model $\mathcal{M} = \langle W, \{R^m\}_{m \in \mathsf{MOD}}, V\rangle$. It is what model theorist call a *relational structure*. It has a domain of quantification (W), a collection of binary relations over this domain ($\{R^m\}_{m \in \mathsf{MOD}}$), and a collection of unary relation as well (V). To talk about a model $\mathcal{M} = \langle W, \{R^m\}_{m \in \mathsf{MOD}}, V\rangle$ we can make use of a first-order language with a binary relation symbol R^m for every $m \in \mathsf{MOD}$, and



(a) $\mathcal{M}$.

(b) $\mathcal{M}^+$.

Figure 2.5: Generated submodel of $\mathcal{M}$.

a unary relation symbol P for every $p \in \mathsf{PROP}$. Modal logicians have a name for this language: the *first-order correspondence language*. It is called the *correspondence language* because every basic modal formula (in the language over PROP and MOD) corresponds to a first-order formula from this language via the *standard translation*.

Definition 2.12 (Standard Translation)**.** *Let x be a first-order variable. The standard translation ST_x taking modal formulas to first-order formulas is defined as*

$$\begin{aligned} ST_x(p) &= P(x) \\ ST_x(\neg\varphi) &= \neg ST_x(\varphi) \\ ST_x(\varphi \vee \psi) &= ST_x(\varphi) \vee ST_x(\psi) \\ ST_x([m]\varphi) &= \forall y(R^m(x,y) \rightarrow ST_y(\varphi)). \end{aligned}$$

The standard translation maps propositional symbols to unary predicates, commutes with Boolean connectives, and handles boxes and diamonds by explicit first-order quantification over R^m-accessible points. The variable y used in the clauses for diamond and boxes is chosen to be any new variable (i. e., one that has not been used so far in the translation). We remarked earlier that diamonds and boxes were essentially a simple macro notation encoding quantification over accessible states; the standard translation expands these macros. Note that $ST_x(\varphi)$ always contains exactly one free variable (namely x). This free variable is what allows for the internal perspective, typical of modal logic: assigning a value to this variable is analogous to evaluating a modal formula inside a model at certain point.

Example 2.6. *Consider the formula $\varphi = p \rightarrow \Diamond p$. The standard translation of φ is*

$$\begin{aligned} ST_x(p \rightarrow \Diamond p) &= ST_x(p) \rightarrow ST_x(\Diamond p) \\ &= P(x) \rightarrow ST_x(\Diamond p) \\ &= P(x) \rightarrow \exists y(R(x,y) \wedge ST_y(p)) \\ &= P(x) \rightarrow \exists y(R(x,y) \wedge P(y)) \end{aligned}$$

An easily verifiable consequence of the standard translation is that a modal formula φ is satisfiable if and only if $ST_x(\varphi)$ is satisfiable.

Proposition 2.3. *For any basic modal formula φ, any model $\mathcal{M}$, and any point w in $\mathcal{M}$ we have that $\mathcal{M}, w \models \varphi$ if and only if $\mathcal{M} \models ST_x(\varphi)[x \leftarrow w]$*[1]*.*

Proof. Trivial by induction on φ. □

The standard translation shows that each modal formula φ corresponds to a first-order formula $ST_x(\varphi)$ containing a free variable x. However, the converse does not hold: some first-order formulas in the correspondence language are not modally definable. A typical example is the formula $\neg R(x, x)$ that defines irreflexivity.

Thus, viewed as a tool for talking about models, modal logics are strictly less expressive than the full first-order correspondence language. Given that a modal language is essentially a fragment of the corresponding first-order language, the question now is exactly which fragment it is. To answer this question we need to introduce a preliminary definition.

Definition 2.13. *A first-order formula $\varphi(x)$ is* invariant for bisimulation *if for all models $\mathcal{M}$ and $\mathcal{M}'$, and all points w in $\mathcal{M}$ and w' in $\mathcal{M}'$, and all bisimulations Z between $\mathcal{M}$ and $\mathcal{M}'$ such that wZw', we have that $\mathcal{M} \models \varphi[x \leftarrow w]$ if and only if $\mathcal{M}' \models \varphi[x \leftarrow w']$.*

Now the main result: basic modal languages correspond to the fragment of their first-order correspondence language invariant for bisimulation.

Theorem 2.1 (Modal Characterization Theorem)**.** *The following are equivalent for all first-order formulas $\varphi(x)$ in one free variable x:*

i) $\varphi(x)$ is invariant for bisimulation.

ii) $\varphi(x)$ is equivalent to the standard translation of a basic modal formula.

Proof. See [van Benthem, 1977; van Benthem, 1983] for a complete proof. □

The standard translation also gives us a bridge between basic modal logic and first-order logic that can be used to transfer meta-theoretic results for first-order logic to basic modal logic.

Proposition 2.4. *The basic modal logic has the compactness property. If Σ is a set of basic modal formulas, and every finite subset of Σ is satisfiable, then Σ itself is satisfiable.*

Moreover, the basic modal logic has the Löwenheim-Skölem property. If a set of basic modal formulas Σ is satisfiable in at least one infinite model, then it is satisfiable in models of every infinite cardinality.

Proof. Let us show that the basic modal logic has the Löwenheim-Skölem property. Suppose that Σ is a set of basic modal formulas that has at least one infinite model. Let $ST_x(\Sigma)$ be the set of (first-order) formulas obtained by translating all the formulas in Σ using the standard translation. Now, as Σ has an infinite model, by Proposition 2.3 so does $ST_x(\Sigma)$. But first-order logic has the Löwenheim-Skölem property, hence $ST_x(\Sigma)$ has a model of every infinite cardinality. But, again by appeal to Proposition 2.3, each of these models satisfies Σ, so basic modal logic has the Löwenheim-Skölem property too. A similar argument holds for the compactness property. □

1 $[x \leftarrow w]$ means assign w to the free variable x.

Another easy consequence of the standard translation is that the set of validities (in basic modal languages) is recursively enumerable. A basic modal formula φ is valid if and only if $ST_x(\varphi)$ is a first-order validity, and the set of first-order validities is recursively enumerable.

These examples show that basic modal logic and first-order logic are analogous in many ways. However, transfer of meta-theoretic properties is not automatic. For example, consider the Craig Interpolation property:

> If $\varphi \models \psi$ then there exists a formula θ whose vocabulary is included in that of both φ and ψ such that $\varphi \models \theta$ and $\theta \models \psi$.

Does the same result hold for basic modal formulas φ and ψ such that $\varphi \models \psi$? Appealing to the result for first-order logic gives us a first-order formula θ such that $ST_x(\varphi) \models \theta$ and $\theta \models ST_x(\psi)$. But nothing guarantees that this interpolant is modally definable. Interpolation does in fact hold for the basic modal logic, but additional work is needed to prove this.

2.4 COMPUTATIONAL PROPERTIES: DECIDABILITY AND COMPLEXITY

One of the reasons that make modal logics so interesting is that they provide a good balance between expressivity and computational costs.

We have already talked about the expressivity of the basic modal logic. We now turn our attention to its computational properties, in particular to the computational properties of the *satisfiability (validity) problem*.

The satisfiability problem for the basic modal logic can be formulated as follows:

> *Given a basic modal formula φ, is φ satisfiable?*

Notice that the satisfiability and the validity problems are dual: a modal formula φ is valid if and only if $\neg\varphi$ is not satisfiable.

The first thing that we can say about the basic modal logic is that it is *decidable*. This can be read as a shorthand for the claim that the *satisfiability problem* for the basic modal logic is decidable, i. e., it is possible, ignoring constraints of time and space, to write a computer program which takes a basic modal formula as input, and halts after a finite number of steps and correctly tells us whether or not it is satisfiable. This follows directly from the fact that basic modal logic has the *bounded finite model property*, i. e., if a modal formula is satisfiable on an arbitrary model, then it is satisfiable on a bounded finite model.

Theorem 2.2 (Bounded Finite Model Property)**.** *Let φ be a basic modal formula. If φ is satisfiable, then it is satisfiable on a finite model containing at most $2^{S(\varphi)}$ points, where $S(\varphi)$ is the number of subformulas of φ.*

If a modal formula φ is satisfiable at all, it is satisfiable on a model containing at most $2^{S(\varphi)}$ points. As there are (up to isomorphism) only finitely many such models, exhaustive search through them all will settle the issue of φ's satisfiability. In other words, the basic modal logic does not have the expressive strength to force the existence of infinite models. Contrast this with what happens in first-order logic where it is quite easy to write a formula that forces the existence of an infinite model, e. g., the formula $\forall x \neg R(x,x) \wedge \forall xyz(R(x,y) \wedge R(y,z) \rightarrow R(x,z)) \wedge \forall x \exists y R(x,y)$ has only infinite models.

Now that we know that we can write an algorithm to decide if a formula is satisfiable or not, we would like to know how complex is the problem, in particular, what resources of time, i. e., computation steps, or space, i. e., memory, are needed to carry out the required computations.

Recall that the basic modal logic is an extension of propositional logic therefore its satisfiability problem is at least as hard as the one of propositional logic, i. e., it is NP-hard. However the added expressivity of basic modal logic (with respect to propositional logic) suggests that its satisfiability problem should be harder than for the propositional case. In fact, the satisfiability problem for the basic modal logic is PSPACE-complete [Ladner, 1977], i. e., given a modal formula φ, it is possible to write an algorithm to determine whether or not φ is satisfiable that uses an amount of computer memory that is only polynomial in the size of φ. How do we design a PSPACE-algorithm for modal satisfiability? A detailed answer is out of the scope of this thesis, but we can point out the underlying theoretic tools needed to build one.

First let us introduce a key syntactic notion: the *modal depth* of a formula.

Definition 2.14 (Modal depth). *The* modal depth *of a formula φ (notation $md(\varphi)$) is a function from formulas to natural numbers defined as:*

$$\begin{array}{lcl} md(p) & = & 0, \text{ for } p \in \mathsf{PROP} \\ md(\neg\varphi) & = & md(\varphi) \\ md(\varphi \vee \psi) & = & \max\{md(\varphi), md(\psi)\} \\ md([m]\varphi) & = & 1 + md(\varphi). \end{array}$$

In words, the *modal depth* of a formula is the the maximum nesting of modal operators, e. g., $md(\langle m\rangle(p \wedge q \wedge p) \wedge [m][m]\neg r) = 2$, and it is often taken as a measure of its complexity. Its appeal lies in that it estimates and summarizes in a single value several aspects of complexity like the expressive power of the formula, the computational cost of evaluating the formula in a model, the minimum size of a model for the formula, among others.

Second, we have the notion of *n-bisimulation*. An n-bisimulation is a way of finitely approximating a bisimulation that builds upon the fact that modal satisfaction is intrinsically local, i. e., modalities only scan those points accessible from the current point. Therefore, there is a relation between the modal depth of a formula and how much from a model it can see from the evaluation point.

Definition 2.15 (n-Bisimulation). *Let $\mathcal{M} = \langle W, R, V\rangle$ and $\mathcal{M}' = \langle W', R', V'\rangle$ be models and w and w' be states of $\mathcal{M}$ and $\mathcal{M}'$ respectively. An* n-bisimulation *between w and w' ($w \underline{\leftrightarrow}_n w'$) is a sequence of binary relations $Z_n \subseteq \ldots \subseteq Z_0$ ($Z_i \subseteq W \times W'$) with the following properties (for $i + 1 \leq n$):*

i) wZ_nw'. *[Root]*

ii) If vZ_0v' then $v \in V(p)$ iff $v' \in V'(p)$, for all $p \in \mathsf{PROP}$. *[Atomic Harmony]*

iii) If $vZ_{i+1}v'$ and vRu, then there exists $u' \in W'$ such that uZ_iu' and $v'R'u'$. *[Zig]* *iv) If $vZ_{i+1}v'$ and $v'R'u'$ then there exists $u \in W$ such that uZ_iu' and vRu.* *[Zag]*

If there is an n-bisimulation linking two states w in $\mathcal{M}$ and w' in $\mathcal{M}'$ we say that w and w' are n-bisimilar and we write $\mathcal{M}, w \underline{\leftrightarrow}_n \mathcal{M}', w'$. If there is some n-bisimulation between two models $\mathcal{M}$ and $\mathcal{M}'$ we say that $\mathcal{M}$ and $\mathcal{M}'$ are n-bisimilar *and write $\mathcal{M} \underline{\leftrightarrow}_n \mathcal{M}'$.*

The intuition is that, if $w \underline{\leftrightarrow}_n w'$, then w and w' *bisimulate up to depth n*, i. e., an n-bisimulation preserves modal formulas of modal depth at most n.

Proposition 2.5 (n-bisimulation Invariance Lemma). *Let $\mathcal{M} = \langle W, R, V\rangle$ and $\mathcal{M}' = \langle W', R', V'\rangle$ be models of the same signature $\mathcal{S}$. Then for every w in $\mathcal{M}$ and w' in $\mathcal{M}'$, the following are equivalent:*

i) $w \underline{\leftrightarrow}_n w'$.

ii) w and w' agree on all modal formulas φ such that $md(\varphi) \leq n$.

Proof. The implication $(i) \to (ii)$ may be proved by induction on n. For the converse implication one can use an argument similar to the one used in the proof of Proposition 2.2. □

Clearly if $w \underline{\leftrightarrow} w'$, then $w \underline{\leftrightarrow}_n w'$ for all n.

Third, the following result, which highlights an expressivity weakness of the basic modal logic, is key to show the existence of a PSPACE-algorithm.

Proposition 2.6. *Let $\mathcal{M} = \langle W, R, V\rangle$ be a model, let $w \in W$, let n be a natural number, let $S_{n,w}$ be a subset of W containing w and all points in W reachable from w by making at most n R-transitions, and let $\mathcal{N}$ be a submodel $\langle S_{n,w}, R \upharpoonright S_{n,w}, V \upharpoonright S_{n,w}\rangle$ where $R \upharpoonright S_{n,w}$ and $V \upharpoonright S_{n,w}$ are the restrictions of R and V respectively to $S_{n,w}$. Then, for all basic modal formulas φ such that $md(\varphi) \leq n$, we have that $\mathcal{M}, w \models \varphi$ if and only if $\mathcal{N}, w \models \varphi$.*

Proof. Follows directly by the fact that $\mathcal{M}, w \underline{\leftrightarrow}_n \mathcal{N}, w$. □

Proposition 2.6 tells us that modal formulas have a shallow vision, i. e., if we take a model $\mathcal{M}$, and extract a submodel $\mathcal{N}$ from it by throwing away all points that are more than n steps away from w, then no formula with modal depth of at most n can distinguish the two models at w.

Combining Proposition 2.6 with what has already been learned about finite models we obtain the following result.

Theorem 2.3. *Every satisfiable formula φ in the basic modal language is satisfiable in a model based on a finite tree of depth at most $md(\varphi)$.*

Proof. As the basic modal logic has the finite model property (see Theorem 2.2), if a modal formula is satisfiable, then it is satisfiable on a finite model $\mathcal{M}$ at some point w. Now consider the model unravelling of $\mathcal{M}$ around w, $\mathcal{T}(\mathcal{M})$. We do not necessarily obtain a finite model, but, as $\mathcal{M}$ is finite, we do obtain a model based on a tree with a finite branch factor, and this model satisfies φ at the root. If we then chop off all points more than $md(\varphi)$ away from the root we obtain a finite model which, by Proposition 2.6, satisfies φ at its root. □

Theorem 2.3 tells us that every satisfiable basic modal formula is satisfiable on a tree, and also puts us in a position to appreciate how PSPACE algorithms for modal satisfiability work. In essence, they construct shallow trees branch by branch. If a branch is successfully constructed (something which takes only space polynomial in the size of the input formula, as the length of the branch is bounded by $md(\varphi)$) the branch is discarded (thus freeing up the memory) and the next branch is then constructed. There may be many branches, so it may take exponential time to construct them all, but as all branches are discarded once they have been constructed, such algorithm uses only polynomial space.

One thing to keep in mind is that the previous PSPACE decidability result concerns satisfiability (validity) of the basic modal language on the class of all models. If we impose restrictions on the class of models or work with richer modal languages (or both) there is no reason to suppose that the satisfiability (validity) problems over such model classes will remain in PSPACE, or even that they will be decidable. Indeed, in many cases they are not.

In some cases, restricting the attention to a certain class of models may lower the computational complexity. For example, if we restrict the attention to the class of models in which R is a partial function, then satisfiability becomes NP-complete, i. e., no worse than the satisfiability problem of propositional logic. But restricting the attention to other classes of models can easily result in undecidable problems. For example, if we consider the class of grid-like models (models that contain regions that look like $\mathbb{N} \times \mathbb{N}$ under two orderings: the horizontal ordering, $(j,k)R^k(j+1,k)$, and the vertical ordering $(j,k)R^k(j,k+1)$), even weak languages can be undecidable, as it is possible to encode the $\mathbb{N} \times \mathbb{N}$ tiling problem that is known to be undecidable.

To finish this section we briefly mention another important reasoning task: the *model checking problem*. The model checking problem can be formulated as follows:

> Given a (finite) model $\mathcal{M}$, a point w in $\mathcal{M}$, and a basic modal formula φ, is φ satisfied in $\mathcal{M}$ at w?

This reasoning task has become of great practical importance, as a wide range of practical tasks can be modeled in a computationally natural manner, and efficiently solved, via model checking, e. g., hardware verification. Also, model checking for basic modal logic is a computationally tractable task, i. e., there exists a polynomial algorithm for model checking. This is in contrast with model checking in first-order logic, which is PSPACE-complete.

2.5 extensions

So far we have been working with the basic modal language. It is time to explore stronger languages. Many modal languages built upon the basic modal language retain its nice properties. In what follows we briefly describe just a few of them, namely those that are relevant to this thesis and refer the reader to [Blackburn *et al.*, 2006] for a more complete list of modal languages. We present the universal modality, the difference modality, the converse modality and hybrid logic.

UNIVERSAL MODALITY The *universal modality* [Goranko and Passy, 1992] incorporates the notion of global satisfiability to the basic modal language. Given the basic modal language, we add a new modality E and its dual A ($A\varphi = \neg E \neg \varphi$), and fix the interpretation of E and A such that in any model $\mathcal{M} = \langle W, R, V \rangle$, both modalities must be interpreted using the universal relation $W \times W$. The satisfaction definition for these modalities is

$$\begin{array}{lll} \mathcal{M}, w \models E\varphi & \text{iff} & \text{there is a } u \in W \text{ such that } \mathcal{M}, u \models \varphi, \\ \mathcal{M}, w \models A\varphi & \text{iff} & \text{for all } u \in W \text{ we have } \mathcal{M}, u \models \varphi. \end{array}$$

Notice that $E\varphi$ scans the entire model for a point that satisfies φ, while $A\varphi$ asserts that φ holds everywhere. The E operator is called the *universal diamond* and the A operator the *universal box*. If it is irrelevant whether we mean E or its dual, we simply talk of the *universal modality*. From a computational point of view, the increase of expressiveness comes at the cost of a much harder satisfiability problem. In fact, the satisfiability problem for the basic modal language enriched with the universal modality is EXPTIME-complete [Spaan, 1993].

DIFFERENCE MODALITY The *difference modality* [Segerberg, 1980] refers to the *existential difference modality D* and its dual, the *universal difference modality B*. The semantics< for these modalities is defined as

$$\begin{array}{lll} \mathcal{M}, w \models D\varphi & \text{iff} & \text{there is } v \neq w \text{ and } \mathcal{M}, v \models \varphi, \\ \mathcal{M}, w \models B\varphi & \text{iff} & \text{for all } v \text{ such that } v \neq w, \mathcal{M}, v \models \varphi. \end{array}$$

Intuitively, $D\varphi$ means *elsewhere, φ holds*, whereas $B\varphi$ means *everywhere else, φ holds*. The difference operator is strong enough to define the universal modality E, since $\varphi \vee D\varphi$ is equivalent to $E\varphi$ and $\varphi \wedge B\varphi$ to $A\varphi$, but D cannot be defined using E, i. e., it is strictly more expressive [Gargov and Goranko, 1993].

CONVERSE MODALITY The *converse modality* is sometimes called the *past modality* and the logic obtained is also sometimes called *tense modal logic*, as a reference to the past tense of natural languages like English. The semantics of the new modal connector $\langle m \rangle^{-}$ (and its dual $[m]^{-}$) is defined as

$$\mathcal{M}, w \models \langle m \rangle^{-} \varphi \text{ iff for some } v \in W, vR^{m}w \text{ and } \mathcal{M}, v \models \varphi.$$

Adding these operators to the basic modal logic causes no shift in complexity, i. e., checking satisfiability remains a PSPACE-complete task. This can be explained by the fact that the converse modality remains local, contrary to E and D. Adding the converse modality to the modal logic that includes the universal modality also leaves the obtained logic EXPTIME-complete.

HYBRID LOGIC Basic modal languages have an obvious expressive weakness: they cannot name points. This implies that we cannot say “this happened then”, or that some particular individual has some property. The *basic hybrid language* overcome this limitation by adding *nominals* to the language.

Nominals are propositional symbols that act as univocal names for states in a model by being forced to be true at exactly one state, i. e., for any valuation V and nominal i, $V(i)$ must be a singleton set. This simple addition results in a more expressive logic. For example, consider the following modal formula:

$$\Diamond(r \wedge p) \wedge \Diamond(r \wedge q) \rightarrow \Diamond(p \wedge q).$$

This formula can be falsified, as the p-witnessing and q-witnessing points given by the antecedent may be distinct. But now consider the following hybrid formula:

$$\Diamond(i \wedge p) \wedge \Diamond(i \wedge q) \rightarrow \Diamond(p \wedge q).$$

It is identical to the preceding formula, except that we have replaced the proposition symbol r by the nominal i. However, the resulting formula is valid. For now we have extra information: the p-witnessing and the q-witnessing successors both make i true, so they are true at the same point, namely the denotation of i. Additionally, hybrid logic also involves new modal operators: the *satisfaction operators* $@_i$. The formula $@_i\varphi$ asserts that φ is satisfied at the unique point named by the nominal i, i. e.,

$$\mathcal{M}, w \models @_i\varphi \text{ iff } \mathcal{M}, u \models \varphi, \text{ where } u \text{ is the denotation of } i.$$

Notice that $@_i$ is a global operator like E and A.

One important point about satisfaction operators is that they give us a modal perspective on the equality relation. To see this consider the formula $@_ij$. This formula says that "at the denotation of i, the nominal j is satisfied", or put it another way, "the point named i is identical to the point named j. Hence the following schemas are valid:

i) $@_ii$. [Reflexivity of equality]

ii) $@_ij \rightarrow @_ji$. [Symmetry of equality]

iii) $@_ij \wedge @_jk \rightarrow @_ik$. [Transitivity of equality]

iv) $@_i\varphi \wedge @_ij \rightarrow @_j\varphi$. [Replacement]

Thus, basic hybridisation is a mechanism for equality reasoning in propositional modal logic. From a complexity point of view, the addition of nominals to the basic modal language (including the satisfaction operators) makes no difference in the complexity of the satisfaction problem, i. e., it remains PSPACE-complete [Areces *et al.*, 2000a]. However, adding nominals to the basic modal language with the converse modality makes the satisfaction problem to be EXPTIME-complete.

A number of stronger hybrid languages have also been explored. One of the most interesting extensions adds the downarrow binder $\downarrow$. This operator binds occurrences of nominals within its scope to the point of evaluation, i. e., to evaluate $\mathcal{M}, w \models \downarrow i.\varphi$, we evaluate $\mathcal{M}, w \models \varphi$ but with all occurrences of the nominal i that were bound by $\downarrow$ now interpreted as naming w. In other words, $\downarrow$ lets us create a name for *here*, and this immediately increases the expressive power at our disposal. This comes at the cost of turning the logic undecidable.

3 SYMMETRIES IN MODAL LOGICS

After a long, but necessary, introduction we finally get to talk about symmetries in modal logics. In Chapter 1 we saw that symmetries have been extensively investigated for propositional satisfiability and other logics. However they remain uninvestigated in the field of automated theorem proving for modal logics. In this chapter we start filling the gap by developing the theoretical foundations that enable the exploitation of symmetries in modal logics. We do so gradually, by presenting first the definitions and results for the basic modal logic, and then extending them to the more powerful *coinductive modal models* framework.

We start by showing how permutations of literals can be used to define symmetries for basic modal formulas in clausal form and how they share many similar properties with symmetries for the propositional case (Section 3.1). Then we extend the results to a broad range of modal languages (Section 3.2). To do so, we present the *coinductive modal models* framework (Section 3.2.1). Then we generalize the notion of symmetries to modal formulas in clausal form for different modal logics including the basic modal language over different model classes (e.g., reflexive, linear or transitive models), and logics with additional modal operators (e.g., universal and hybrid operators) using the framework of coinductive modal models (Section 3.2.2). Finally, in cases where the modal language has the tree model property, we develop a more flexible notion of symmetry that enables us to find symmetries that would not be found otherwise (Section 3.2.3).

The results presented in this chapter were published in [Areces *et al.*, 2012].

3.1 SYMMETRIES IN BASIC MODAL LOGIC

In what follows we work with the language presented in Definition 2.1. As we are interested in the syntactic symmetries of modal formulas, we normalize their syntactic representation.

Definition 3.1 (Literals and Modal CNF). *A* propositional literal *l is either a propositional variable p or its negation $\neg p$. The set of literals over* PROP *is* $\mathsf{PLIT} = \mathsf{PROP} \cup \{\neg p_i \mid p_i \in \mathsf{PROP}\}$. *A modal formula is in* modal conjunctive normal form (modal CNF) *if it is a conjunction of modal CNF clauses. A* modal CNF clause *is a disjunction of propositional and* modal literals. *A* modal literal *is a formula of the form $[m]C$ or $\neg[m]C$ where C is a modal CNF clause.*

We say that two formulas are *equisatisfiable* if the first formula is satisfiable whenever the second is and vice versa; in other words, either both formulas are satisfiable or both are not. Every modal formula can be transformed into an equisatisfiable formula in modal CNF in polynomial time [Patel-Schneider and Sebastiani, 2003b].

From now on, we assume that modal formulas are in modal CNF, and we will refer to them as modal CNF formulas. Also, we often represent a modal CNF formula as a set of modal CNF clauses (interpreted conjunctively), and each modal

CNF clause as a set of propositional and modal literals (interpreted disjunctively). With the set representation we can disregard the order and multiplicity in which clauses and literals appear. This will be important when we define symmetries below.

Example 3.1. *The modal formula $\varphi = \Diamond(p \wedge q \wedge p) \wedge \Box\Box\neg r$ is equisatisfiable to the modal CNF formula $\varphi' = \neg\Box(\neg p \vee \neg q \vee \neg p) \wedge \Box\Box\neg r$. The set representation of φ' is $\{\{\neg\Box\{\neg p, \neg q\}\}, \{\Box\{\Box\{\neg r\}\}\}\}$.*

To ease the definition of symmetry below, from now on we are going to work with a slightly modified definition of models.

Definition 3.2 (Models). *A model (or Kripke model) $\mathcal{M}$ is a triple $\mathcal{M} = \langle W, \{R^m\}_{m \in \mathsf{MOD}}, V\rangle$ such that:*

i) *W, the* domain, *is a non-empty set. Elements of W are called* points, states, worlds, *etc.*

ii) *Each R^m is a binary relation on W.*

iii) *V, the* valuation, *is a function that assigns to each element $w \in W$ a subset $V(w) \subseteq$ PROP. Informally, we think of $V(w)$ as the set of propositional variables that hold at w.*

Notice that Definition 2.2, introduced earlier, differs from that of Definition 3.2 in how it defines the valuation function. In the former, the valuation function is interpreted as the set of states in which a propositional symbol holds, i. e., V is a function $V : \mathsf{PROP} \mapsto \mathcal{P}(W)$. In the latter, the valuation function is interpreted as the set of propositional symbols that hold at a given state, i. e., V is a function $V : W \mapsto \mathcal{P}(\mathsf{PROP})$.

Remark. As in Chapter 2, we are going to restrict ourselves to the mono-modal case. All the results we present extend naturally to the multi-modal case.

Definition 3.3 (Pointed Models). *A pointed model is a model with a distinguished element, e. g., given a model $\mathcal{M} = \langle W, R, V\rangle$ and an element w of W, then, the corresponding pointed model is the tuple $\mathcal{M} = \langle w, W, R, V\rangle$. If a pointed model $\mathcal{M}$ satisfies a formula φ we write $\mathcal{M} \models \varphi$.*

We now define the semantics for modal CNF formulas.

Definition 3.4 (Semantics for modal CNF formulas). *Let $\mathcal{M} = \langle w, W, R, V\rangle$ be a pointed model and φ a modal CNF formula. We inductively define when a formula φ is* satisfied *(true) in $\mathcal{M}$ as*

$$\begin{array}{lll}
\mathcal{M} \models \varphi & \textit{iff} & \textit{for all clauses } C \in \varphi \textit{ we have } \mathcal{M} \models C, \\
\mathcal{M} \models C & \textit{iff} & \textit{exists a literal } l \in C \textit{ such that } \mathcal{M} \models l, \\
\mathcal{M} \models p & \textit{iff} & p \in V(w) \;\textit{ for }\; p \in \mathsf{PROP}, \\
\mathcal{M} \models \neg p & \textit{iff} & p \notin V(w) \;\textit{ for }\; p \in \mathsf{PROP}, \\
\mathcal{M} \models \Box C & \textit{iff} & \langle v, W, R, V\rangle \models C, \textit{for all } v \textit{ such that } wRv, \\
\mathcal{M} \models \neg\Box C & \textit{iff} & \mathcal{M} \not\models \Box C.
\end{array}$$

Given a formula φ, $\mathsf{Mods}(\varphi) = \{\mathcal{M} \mid \mathcal{M} \models \varphi\}$ is the set of all models of φ.

In what follows we work with sets of propositional literals, therefore the following definitions are necessary.

Definition 3.5 (Complete, Consistent and Generated sets of propositional literals). *A set of propositional literals L is* complete *if for each $p \in \mathsf{PROP}$ either $p \in L$ or $\neg p \in L$. It is* consistent *if for each $p \in \mathsf{PROP}$ either $p \notin L$ or $\neg p \notin L$. Any complete and consistent set of propositional literals L defines a unique propositonal valuation $v \subseteq \mathsf{PROP}$ as $p \in v$ if $p \in L$ and $p \notin v$ if $\neg p \in L$. For $S \subseteq \mathsf{PROP}$, the* consistent and complete set of propositonal literals generated by S *(notation L_S) is $S \cup \{\neg p \mid p \in \mathsf{PROP} \backslash S\}$.*

Next, let us introduce the basic notions of group theory that we need to work with symmetries. We refer the reader to Appendix A for more details on the subject.

Definition 3.6 (Permutation). *A* permutation of a set *A is a bijection $\sigma : A \mapsto A$.*

The set of all permutations of any given set A forms a group with function composition as product and the identity as neutral element.

Theorem 3.1 (Permutation Group). *Given a non-empty set A, let S_A be the set of all permutations of A. Then S_A is a group under function composition.*

For $n \in \mathbb{Z}_{\geq 1}$ and σ a permutation, we denote the composition of σ with itself n times by σ^n. σ^0 denote the identity permutation, σ^{-1} the inverse of σ, and σ^{-n}, for $n \in \mathbb{Z}_{\geq 1}$ is the n-times composition of σ^{-1} with itself.

Each permutation σ of a set A determines a natural partition of A into equivalence classes with the property that $a, b \in A$ are in the same equivalence class if and only if $b = \sigma^n(a)$ for some $n \in \mathbb{Z}$.

Definition 3.7 (Orbits of a permutation). *For a permutation σ of a set A, the equivalence classes in A determined by σ are the* orbits of *σ.*

Permutations defined over finite sets can be succinctly written using *cycle notation*.

Definition 3.8 (Cycle Notation). *Let A be a finite set and let $a_1, \ldots, a_n$ be distinct elements of A. The expression $(a_1\ a_2\ \ldots\ a_n)$ is a* cycle *and denotes the action of mapping $a_1 \mapsto a_2, a_2 \mapsto a_3, \ldots, a_{n-1} \mapsto a_n, a_n \mapsto a_1$. An element not appearing in the cycle is understood to be left fixed by the cycle. A cycle with only two elements is called a* transposition.

As cycles define permutations, they can be composed. Therefore, every finite permutation can be expressed as the product of disjoint cycles.

Theorem 3.2 ([Fraleigh and Katz, 2003]). *Every permutation σ of a finite set is a product of disjoint cycles.*

Proof. Let $B_1, B_2, \ldots, B_r$ be the orbits of σ, and let μ_i be the cycle defined by

$$\mu_i(x) \begin{cases} \sigma(x) & \text{for } x \in B_i \\ x & \text{otherwise.} \end{cases}$$

Clearly $\sigma = \mu_1 \mu_2 \ldots \mu_r$. Since the equivalence-class orbits $B_1, B_2, \ldots B_r$ being distinct equivalence classes, are disjoint, the cycles $\mu_1 \mu_2 \ldots \mu_r$ are also disjoint. □

Example 3.2. *$\sigma = (p\ \neg q)(\neg p\ q)$ is the permutation over $\{p, q, \neg p, \neg q\}$ that makes $\sigma(p) = \neg q$, $\sigma(\neg q) = p$, $\sigma(\neg p) = q$ and $\sigma(q) = \neg p$; $\sigma = (p\ q\ r)(\neg p\ \neg q\ \neg r)$ is the permutation $\sigma(p) = q$, $\sigma(q) = r$ and $\sigma(r) = p$ and similarly for the negations.*

In what follows, we deal with permutations of propositional literals.

Definition 3.9 (Permutation of propositional literals)**.** *A* permutation of propositional literals *is a bijective function $\sigma : \mathsf{PLIT} \mapsto \mathsf{PLIT}$. For L a set of propositional literals, $\sigma(L) = \{\sigma(l) \mid l \in L\}$.*

Remark. Notice that Definition 3.9 defines permutations over the infinite set PLIT. However, in practice we only deal with permutations over a finite subset A of PLIT, i. e., the set of propositional literals occurring in the formula at hand. Therefore, we can use all the results on permutations over finite sets introduced previously. Also notice that we can easily extend a permutation over a finite subset A of PLIT to a permutation over PLIT by mapping every element not in A to itself.

In what follows we assume that every permutation is a permutation of propositional literals and just call them a "permutation".

Definition 3.10 (Permutation of a formula)**.** *Let φ be a modal CNF formula and σ a permutation. We define $\sigma(\varphi)$ recursively as*

$$\begin{array}{rcll} \sigma(\varphi) & = & \{\sigma(C) \mid C \in \varphi\} & \textit{for } \varphi \textit{ a modal CNF formula} \\ \sigma(C) & = & \{\sigma(A) \mid A \in C\} & \textit{for } C \textit{ a modal CNF clause} \\ \sigma(\Box C) & = & \Box\sigma(C) & \\ \sigma(\neg\Box C) & = & \neg\Box\sigma(C). & \end{array}$$

Given a set of propositional literals, we only are interested in those permutations that are *consistent*.

Definition 3.11 (Consistent Permutation)**.** *A permutation σ is* consistent *if for every propositional literal l, $\sigma(\sim l) = \sim\sigma(l)$, where $\sim$ is a function that returns the complement of a propositional literal, i. e., $\sim p = \neg p$ and $\sim\neg p = p$.*

Consistency guarantees that a permutation will interact nicely when applied to set of literals, e. g., if we have a consistent set of literals it will remain consistent after applying a consistent permutation to it. Notice that, from a group theoretic perspective, consistent permutations form a subgroup of the group of all permutation over a given set, as the composition of two or more consistent permutations is again a consistent permutation.

Definition 3.12 (Symmetry)**.** *Let φ be a modal CNF formula. A consistent permutation σ is a* (consistent) symmetry *for φ if $\varphi = \sigma(\varphi)$, when conjunctions and disjunctions in φ are represented as sets.*

Example 3.3. *Trivially, the identity permutation $\sigma(l) = l$ is a consistent symmetry of any formula φ. More interestingly, consider $\varphi = \{\{\neg p, r\}, \{q, r\}, \{r, \Box\{\neg p, q\}\}\}$, then the permutation $\sigma = (p\ \neg q)(\neg p\ q)$ is a consistent symmetry of φ.*

We now have in place the necessary tools to start talking about symmetries in basic modal logic. Let us begin with the key concept of the forthcoming theory: σ-simulations.

Definition 3.13 (σ-simulation). *Let σ be a permutation. A* σ-simulation *between models $\mathcal{M} = \langle w, W, R, V\rangle$ and $\mathcal{M}' = \langle w', W', R', V'\rangle$ is a non-empty relation $Z \subseteq W \times W'$ that satisfies the following conditions:*

i) wZw'. *[Root]*

ii) If wZw' then $l \in L_{V(w)}$ if and only if $\sigma(l) \in L_{V'(w')}$. *[Atomic Harmony]*

iii) If wZw' and wRv then exists v' such that $w'R'v'$ and vZv'. *[Zig]*

iv) If wZw' and $w'R'v'$ then exists v such that wRv and vZv'. *[Zag]*

We say that two pointed models $\mathcal{M}$ and $\mathcal{M}'$ are σ-similar (notation $\mathcal{M} \underline{\rightarrow}_\sigma \mathcal{M}'$) if there is a σ-simulation Z between them.

Notice the resemblance with the definition of a bisimulation (Definition 2.5). A σ-simulation is in fact a bisimulation that relaxes the atomic harmony condition to incorporate the effect of permutations. However, in the general case, a σ-simulation is a non-symmetric relation: we can have $\mathcal{M} \underline{\rightarrow}_\sigma \mathcal{M}'$ but not $\mathcal{M}' \underline{\rightarrow}_\sigma \mathcal{M}$.

Example 3.4. *Consider the models $\mathcal{M} = \langle w, \{w\}, \emptyset, V\rangle$ and $\mathcal{M}' = \langle w', \{w'\}, \emptyset, V'\rangle$ where $V(w) = \{p, s\}$ and $V'(w') = \{q, s\}$ respectively. Let $\sigma = (p\ q\ r)(\neg p\ \neg q\ \neg r)$ be a consistent permutation. That $\mathcal{M} \underline{\rightarrow}_\sigma \mathcal{M}'$ is straightforward: just consider the set $L_{V(w)} = \{p, \neg q, \neg r, s\}$, then we have that $\sigma(L_{V(w)}) = L_{V'(w')}$ and the atomic harmony condition holds. However it is not the case that $\mathcal{M}' \underline{\rightarrow}_\sigma \mathcal{M}$. To see this, consider the set $L_{V'(w')} = \{\neg p, q, \neg r, s\}$, then $\sigma(L_{V'(w')}) = \{\neg q, r, \neg p, s\} \neq L_{V(w)}$ and the atomic harmony condition fails.*

Notice that if we restrict ourselves to permutations that can be defined as the product of disjoint transpositions then the σ-simulation relation is indeed symmetric.

From the definition of σ-simulations it intuitively follows that while they do not preserve validity of modal formulas (as is the case with bisimulations) they do preserve validity of *permutations* of formulas.

Proposition 3.1. *Let σ be a consistent permutation, φ a modal CNF formula and $\mathcal{M} = \langle w, W, R, V\rangle$, $\mathcal{M}' = \langle w', W', R', V'\rangle$ models such that $\mathcal{M} \underline{\rightarrow}_\sigma \mathcal{M}'$. Then $\mathcal{M} \models \varphi$ iff $\mathcal{M}' \models \sigma(\varphi)$.*

Proof. The proof is by induction on φ.

<u>Base Case:</u>

- Suppose $\varphi = p$. Then, $\mathcal{M} \models p$ iff $p \in V(w)$ iff $p \in L_{V(w)}$ iff, by definition of σ-simulation, $\sigma(p) \in L_{V'(w')}$ iff $\mathcal{M}' \models \sigma(p)$.
- Suppose $\varphi = \neg p$. Then, $\mathcal{M} \models \neg p$ iff $p \notin V(w)$ iff $\neg p \in L_{V(w)}$ iff, by definition of σ-simulation and consistency of σ, $\sigma(\sim p) = \sim\sigma(p) \in L_{V'(w')}$ iff $\sigma(p) \notin V'(w')$ iff $\mathcal{M}' \models \neg\sigma(p)$.

<u>Inductive Step:</u>

- When $\varphi = C$, with C a clause or a conjunction of clauses, the proof follows by induction directly.

- Suppose $\varphi = \Box\psi$. Then $\mathcal{M} \models \Box\psi$ iff $\langle v, W, R, V\rangle \models \psi$ for all $v \in W$ such that wRv. Given that $\mathcal{M} \underline{\rightarrow}_\sigma \mathcal{M}'$, for all v exists $v' \in W'$ such that $w'R'v'$ and vZv', and, by inductive hypothesis, $\langle v', W', R', V'\rangle \models \sigma(\psi)$. Therefore, $\mathcal{M}' \models \Box\sigma(\psi) = \sigma(\Box\psi)$. The converse follows by using Zag and the inductive hypothesis.

- Suppose $\varphi = \neg\Box\psi$. Then $\mathcal{M} \models \neg\Box\psi$ iff there exists $v \in W$ such that wRv and $\langle v, W, R, V\rangle \models \neg\psi$. Given that $\mathcal{M} \underline{\rightarrow}_\sigma \mathcal{M}'$, exists $v' \in W'$ such that $w'R'v'$ and vZv'. By inductive hypothesis, $\langle v', W', R', V'\rangle \models \sigma(\neg\psi) = \neg\sigma(\psi)$ iff $\mathcal{M}' \models \neg\Box\sigma(\psi)$ iff $\mathcal{M}' \models \sigma(\neg\Box\psi)$. The converse follows by using Zag and the inductive hypothesis.

$\Box$

We also need to consider the effect of applying permutations to models. If φ is true in a model $\mathcal{M}$, we intuitively want $\sigma(\varphi)$ to be true in the model obtained from lifting σ to $\mathcal{M}$.

Definition 3.14 (Permutation of a model)**.** *Let σ be a permutation and $\mathcal{M} = \langle w, W, R, V\rangle$ a model. Then $\sigma(\mathcal{M}) = \langle w, W, R, V'\rangle$, where,*

$$V'(w) = \sigma(L_{V(w)}) \cap \mathsf{PROP} \quad \textit{for all } w \in W.$$

$L_{V(w)}$ is the consistent and complete set of literals generated by $V(w)$. For M a set of models, $\sigma(M) = \{\sigma(\mathcal{M}) \mid \mathcal{M} \in M\}$.

A consequence of the previous definition is that $\mathcal{M}$ and $\sigma(\mathcal{M})$ are always σ-similar.

Proposition 3.2. *Let σ be a consistent permutation and $\mathcal{M} = \langle w, W, R, V\rangle$ a model. Then $\mathcal{M} \underline{\rightarrow}_\sigma \sigma(\mathcal{M})$.*

Proof. We show that the identity relation is a σ-simulation between $\mathcal{M}$ and $\sigma(\mathcal{M})$.

- Atomic Harmony: We have to check that given a literal l, we have that $l \in L_{V(w)}$ iff $\sigma(l) \in L_{V'(w)}$. From the definition of $\sigma(\mathcal{M})$, $L_{V'(w)} = \sigma(L_{V(w)})$, hence if $l \in L_{V(w)}$ then $\sigma(l) \in \sigma(L_{V(w)})$. Moreover, $\sigma(L_{V(w)})$ is a complete set of literals because $L_{V(w)}$ is a complete set of literals and σ is a consistent permutation, and hence the converse also follows.

- Zig and Zag: These conditions are trivial as the relation in both models is the same.

$\Box$

Proposition 3.3. *Let σ be a consistent permutation, φ a modal CNF formula and $\mathcal{M} = \langle w, W, R, V\rangle$ a model. Then $\mathcal{M} \models \varphi$ if and only if $\sigma(\mathcal{M}) \models \sigma(\varphi)$.*

Proof. It follows directly from Proposition 3.1 and Proposition 3.2. $\Box$

Interestingly, if σ is a symmetry of φ then for any model $\mathcal{M}$, $\mathcal{M}$ is a model of φ if and only if $\sigma(\mathcal{M})$ is. This is a direct corollary of the previous propositions in the particular case when σ is a symmetry and hence $\sigma(\varphi) = \varphi$.

Corollary 3.1. *If σ is a symmetry of φ then $\mathcal{M} \in Mods(\varphi)$ if and only if $\sigma(\mathcal{M}) \in Mods(\varphi)$.*

Corollary 3.1 tells us that, in the basic modal logic, symmetries have the same effect on models as they have in propositional logic. The group of symmetries of a formula φ acting on the set of models partitions the set in such a way that equivalence classes (orbits) contain only models or only non-models of φ. As a result of this fact we can avoid looking for a solution in the complete model space and focus just on the representatives from each equivalence class, provided that we can compute them. Remember that the static symmetry breaking approach described in Chapter 1 is based on this fact, as we generate formulas that select one representative model from each equivalence class.

Besides partitioning the model space, symmetries also provide us with a "cheap" inference mechanism.

Theorem 3.3. *Let φ and ψ be modal formulas and let σ be a consistent symmetry of φ. Then $\varphi \models \psi$ if and only if $\varphi \models \sigma(\psi)$.*

Proof. We first show that under the hypothesis of the theorem the following property holds

Claim: $\mathsf{Mods}(\varphi) = \sigma(\mathsf{Mods}(\varphi))$.

[$\supseteq$] Let $\mathcal{X} \in \sigma(\mathsf{Mods}(\varphi))$ and let $\mathcal{M} \in \mathsf{Mods}(\varphi)$ be a model such that $\mathcal{X} = \sigma(\mathcal{M})$. Then $\mathcal{M} \models \varphi$ and by Corollary 3.1 $\sigma(\mathcal{M}) \models \varphi$ and $\sigma(\mathcal{M}) \in \mathsf{Mods}(\varphi)$.

[$\subseteq$] Let $\mathcal{M} \in \mathsf{Mods}(\varphi)$, then $\mathcal{M} \models \varphi$. By Corollary 3.1 $\sigma(\mathcal{M}) \models \varphi$, therefore, $\sigma(\mathcal{M}) \in \mathsf{Mods}(\varphi)$. Since σ is a permutation exists $n \in \mathbb{N}$, $n \geq 1$, such that $\sigma^n(\mathcal{M}) = \mathcal{M}$ and hence $\mathcal{M} \in \sigma(\mathsf{Mods}(\varphi))$.

Now, we have to prove that $\varphi \models \psi$ if and only if $\varphi \models \sigma(\psi)$. By definition, $\varphi \models \psi$ if and only if $\mathsf{Mods}(\varphi) \models \psi$. By Proposition 3.3, this is the case if and only if $\sigma(Mods(\varphi)) \models \sigma(\psi)$.

Given that σ is a symmetry of φ, by the Claim above $\sigma(Mods(\varphi)) \models \sigma(\psi)$ if and only if $Mods(\varphi) \models \sigma(\psi)$, which, by definition, means that $\varphi \models \sigma(\psi)$. ☐

Theorem 3.3 provides an inexpensive inference mechanism that can be used in every situation where entailment is involved during modal automated reasoning. Indeed, applying a permutation on a formula is a calculation that is arguably computationally cheaper than a tableaux expansion or a resolution step. Therefore, new formulas obtained by this means may reduce the total running time of an inference algorithm. In the case of propositional logic, the strengthening of the learning mechanism has already shown its results in [Benhamou *et al.*, 2010]. In the case of modal logic, it remains to be seen when cases of $\varphi \models \psi$ occur during a decision procedure, and how to better take advantage of them. We will discuss how to exploit symmetries from a practical point of view in Chapter 9.

Both Corollary 3.1 and Theorem 3.3 are the core results that enable the exploitation of symmetries in the basic modal logic. As a matter of fact, these results show up in every logic where symmetries have been investigated.

3.2 BEYOND BASIC MODAL LOGIC

So far we have been dealing with symmetries for the basic modal logic. Can we generalize the obtained results to richer modal logics? The short answer is: yes. We could, either, start from scratch for each modal logic for which we want to develop a theory of symmetries and prove the results we discussed above, or we could develop a generic framework applicable to a wide range of modal logics. The latter approach enables us to be more concise at the cost of a higher level of abstraction.

In this thesis we choose the second alternative and to do so we use the framework of *coinductive modal models*. Coinductive modal models were introduced in [Areces and Gorín, 2010] to investigate normal forms for modal logics. Its main characteristic is that it allows the representation of different modal logics in an homogeneous form. Results obtained in the coinductive framework can be easily extended to concrete modal languages by just selecting the appropriate model classes.

In what follows we present the syntax and semantics for the framework of coinductive modal models, and provide several examples to show how we can use it. We then develop a theory of symmetries for coinductive modal models, lifting the results obtained for the basic modal logic.

3.2.1 *Coinductive Modal Models*

Let us start by defining the modal language we will be using throughout this section.

Definition 3.15 (Modal formula). *Let* $\mathsf{ATOM} = \{a_1, a_2, \ldots\}$ *be a countable infinite set of atoms and* $\mathsf{MOD} = \{m, m', \ldots\}$ *be a set of modal symbols. The set of modal formulas over the signature* $\mathcal{S} = \langle \mathsf{ATOM}, \mathsf{MOD} \rangle$ *is defined as*

$$\mathsf{FORM} ::= a \mid \neg\varphi \mid \varphi \vee \psi \mid [m]\varphi,$$

where $a \in \mathsf{ATOM}$*,* $m \in \mathsf{MOD}$*, and* $\varphi, \psi \in \mathsf{FORM}$*.* $\top$ *and* $\bot$ *stand for an arbitrary tautology and contradiction, respectively. We will also use classical connectives such as* $\wedge, \rightarrow$ *and* $\langle m \rangle$*, taken to be defined in the usual way.*

Notice that the language we just defined is the language of the basic multi-modal logic that we introduced in Definition 2.1 but, as we will now see, we will be able to cast other modal logics including, for example, hybrid operators right into this same language in a natural way. How we do this will become clear once we provide our definition of models.

For convenience, we will use *pointed* models (see Definition 3.3).

Definition 3.16 (Models). *Let* $\mathcal{S} = \langle \mathsf{ATOM}, \mathsf{MOD} \rangle$ *be a modal signature and* W *be a fixed, non-empty set.* Mods_W*, the class of all models with domain* W*, for the signature* $\mathcal{S}$*, is the set of all tuples* $\langle w, W, R, V \rangle$ *such that:*

i) $w \in W$.

ii) $R(m, v) \subseteq \mathsf{Mods}_W$ *for* $m \in \mathsf{MOD}$ *and* $v \in W$.

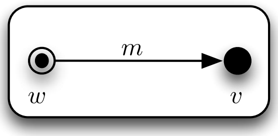
iii) $V(v) \subseteq \mathsf{ATOM}$ *for* $v \in W$.

Mods denotes the class of all models over all domains, i.e., $\mathsf{Mods} = \bigcup_W \mathsf{Mods}_W$.

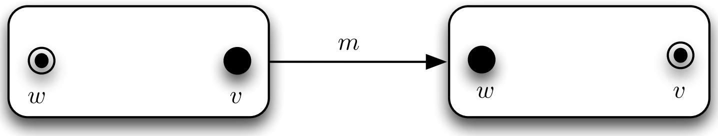
We call w the point of evaluation*, W the* domain*, V the* valuation*, and R the* accessibility relation*. For $\mathcal{M}$ an arbitrary model we write $|\mathcal{M}|$ for its domain, $w^{\mathcal{M}}$ for its point of evaluation, $V^{\mathcal{M}}$ for its valuation and $R^{\mathcal{M}}$ for its accessibility relation. We sometimes write $succs^{\mathcal{M}}(m)$ for the set $R^{\mathcal{M}}(m, w^{\mathcal{M}})$ of immediate m-successors of $w^{\mathcal{M}}$.*

Notice that the main difference between Definition 2.2 (of Kripke models) and Definition 3.16 lies in how we handle m-successors. In the latter, for each modality m and each state w in a model, we define $R(m, w)$, the successors of w through the m modality, as a set of *models* (therefore, this definition of models is coinductive), while in the former we define it as a set of points in the domain.

Example 3.5. *Consider the pointed Kripke model in Figure 3.1a, and its equivalent coinductive modal model in Figure 3.1b. The point of evaluation in each model is circled. The main difference is that the relation of a coinductive model leads to another coinductive model, whereas in a Kripke model the relation leads to another point of the same model.*



(a) Kripke model.



(b) Coinductive model.

Figure 3.1: A Kripke model and its equivalent coinductive model.

Observe that for each W, Mods_W is well-defined (coinductively), and so does Mods, the class of all models. Being the class of all possible models, Mods enjoys some nice closure properties that are useful when considering subclasses of Mods. In particular, we are interested in investigating modal classes that are *closed under accessibility relations*.

Definition 3.17 (Extension of a model)**.** *Given $\mathcal{M} \in \mathsf{Mods}_W$, let* $\mathrm{Ext}(\mathcal{M})$*, the* extension of $\mathcal{M}$*, be the smallest subset of Mods_W that contains $\mathcal{M}$ and is such that if $\mathcal{N} \in \mathrm{Ext}(\mathcal{M})$, then $R^{\mathcal{N}}(m, v) \subseteq \mathrm{Ext}(\mathcal{M})$ for all $m \in \mathsf{MOD}$, $v \in W$.*

Definition 3.18 (Closed class)**.** *A non-empty class of models $\mathcal{C}$ is* closed under accessibility relations *(we will say that $\mathcal{C}$ is a* closed class*, for short) whenever $\mathcal{M} \in \mathcal{C}$ implies $\mathrm{Ext}(\mathcal{M}) \subseteq \mathcal{C}$.*

The extension of a model is simply the class of models reachable via the transitive closure of the union of its accessibility relations; and a class of models $\mathcal{C}$ is closed if for every model $\mathcal{M} \in \mathcal{C}$ the extension of $\mathcal{M}$ is also included in $\mathcal{C}$. Clearly, Mods is a closed class and, as we discuss below, it seems natural to restrict ourselves to investigate only closed classes.

As for the basic modal logic case, we are going to work with formulas in modal CNF form.

Definition 3.19 (Literals and Modal CNF). *An* atom literal *l is either an atom variable a or its negation $\neg a$. The set of literals over* ATOM *is* $\mathsf{ALIT} = \mathsf{ATOM} \cup \{\neg a_i \mid a_i \in \mathsf{ATOM}\}$. *A modal formula is in* modal conjunctive normal form (modal CNF) *if it is a conjunction of modal CNF clauses. A* modal CNF clause *is a disjunction of atom and* modal literals. *A* modal literal *is a formula of the form $[m]C$ or $\neg[m]C$ where C is a modal CNF clause.*

Having properly defined what models are, the definition of the satisfiability relation $\models$ is straightforward.

Definition 3.20 (Semantics). *Let φ be a modal CNF formula and $\mathcal{M} = \langle w, W, R, V\rangle$ a model in* Mods. *We define $\models$ for modal CNF formulas, clauses and literals as*

$$\begin{array}{lll}
\mathcal{M} \models \varphi & \textit{iff} & \textit{for all clauses } C \in \varphi \textit{ we have } \mathcal{M} \models C, \\
\mathcal{M} \models C & \textit{iff} & \textit{exists a literal } l \in C \textit{ such that } \mathcal{M} \models l, \\
\mathcal{M} \models a & \textit{iff} & a \in V(w) \ \textit{ for } \ a \in \mathsf{ATOM}, \\
\mathcal{M} \models \neg a & \textit{iff} & a \notin V(w) \ \textit{ for } \ a \in \mathsf{ATOM}, \\
\mathcal{M} \models [m]C & \textit{iff} & \mathcal{M}' \models C, \textit{for all } \mathcal{M}' \in R(m, w), \\
\mathcal{M} \models \neg[m]C & \textit{iff} & \mathcal{M} \not\models [m]C.
\end{array}$$

If $\mathcal{C}$ is a closed class, we write $\mathcal{C} \models \varphi$ whenever $\mathcal{M} \models \varphi$ for every $\mathcal{M}$ in $\mathcal{C}$, and we say that $\Gamma_{\mathcal{C}} = \{\varphi \mid \mathcal{C} \models \varphi\}$ is the logic *defined by $\mathcal{C}$.*

Inspecting the definition above, we can see that the semantic clause for $[m]$ is the classical condition defining a box operator [Blackburn *et al.*, 2001]. But if we restrict ourselves to the appropriate class of models, we can actually ensure that $[m]$ behaves in different ways. Let us see an example.

Example 3.6 (The universal modality A). *Given a Kripke model $\mathcal{M} = \langle W, R, V\rangle$ the usual semantic clause for the universal modality* A *would be*

$$\mathcal{M}, w \models \mathsf{A}\varphi \textit{ iff } \mathcal{M}, w' \models \varphi, \textit{for all } w' \in W.$$

Instead, let $\mathcal{S} = \langle \mathsf{ATOM}, \mathsf{MOD}\rangle$ with $\mathsf{A} \in \mathsf{MOD}$, and let $\mathcal{C}_{\mathsf{A}}$ be the largest class of models in this signature such that

$$\textit{if } \mathcal{M} \in \mathcal{C}_{\mathsf{A}}, \textit{ then } R^{\mathcal{M}}(\mathsf{A}, w) = \{\langle w', |\mathcal{M}|, R^{\mathcal{M}}, V^{\mathcal{M}}\rangle \mid w' \in |\mathcal{M}|\}.$$

The A*-successors of w are those models identical to $\mathcal{M}$ except in that their point of evaluation is an arbitrary element of the domain. Clearly, the semantic condition of* $[\mathsf{A}]$ *in $\mathcal{C}_{\mathsf{A}}$ (as given in Definition 3.20) coincides exactly with the semantic definition of the universal modality* A *over standard Kripke models.*

By taking suitable classes of models, we can naturally capture many different modal operators. Notice, though, that when defining model classes it seems natural to require the classes to be closed. If a class $\mathcal{C}$ is not closed, the evaluation of some modal formulas on a model in $\mathcal{C}$ might require the inspections of models which are outside the class.

As shown in [Areces and Gorín, 2010], every closed class of models induces a *normal* modal logic [Blackburn *et al.*, 2001]. Moreover, the logic Γ_{Mods} (generated by the class of all possible models) coincides with the basic multi-modal logic (also known as K). In what follows we assume that *every class is closed* and that all of its models conform to some particular, but arbitrary, signature.

Now, how can we restrict the model class? This can be done using *defining conditions*.

Definition 3.21 (Defining conditions). *Predicate P is a* defining condition *for $\mathcal{C}$ whenever $\mathcal{C}$ is the* largest *class of models such that $\mathcal{M} \in \mathcal{C}$ implies that $P(\mathcal{M})$ holds.*

We can use this notation to properly define standard relational modalities.

Definition 3.22 (Relational modalities: the classes $\mathcal{C}^{K}_{m}$ and $\mathcal{C}^{\mathsf{K}}$). *For each $m \in \mathsf{MOD}$, let $\mathcal{C}^{K}_{m}$ be the class defined by the following defining condition:*

$$P^{K}_{m}(\mathcal{M}) \iff \forall v \in |\mathcal{M}|, R^{\mathcal{M}}(m,v) \subseteq \{\langle v', |\mathcal{M}|, R^{\mathcal{M}}, V^{\mathcal{M}}\rangle \mid v' \in |\mathcal{M}|\}$$

Observe that P^{K}_{m} is true in a model $\mathcal{M}$ if every successor of $w^{\mathcal{M}}$ is identical to $\mathcal{M}$ except perhaps on its point of evaluation. We will call m a relational modality *when it is interpreted in $\mathcal{C}^{K}_{m}$. Define the class of models $\mathcal{C}^{K}$ over the signature $\mathcal{S} = \langle \mathsf{ATOM}, \mathsf{MOD}\rangle$ as follows: $\mathcal{M} \in \mathcal{C}^{K}$ if and only if for every modality $m \in \mathsf{MOD}$, $\mathcal{M} \in \mathcal{C}^{K}_{m}$. That is, all modalities are interpreted in $\mathcal{C}^{K}$ as relational modalities.*

3.2.1.1 *Some Modal Logics and their Associated Classes*

We now introduce a number of closed model classes by means of their defining conditions. This model classes capture different modal operators, like the ones from hybrid logics (see Section 2.5).

Consider the signature $\mathcal{S} = \langle \mathsf{ATOM}, \mathsf{MOD}\rangle$ such that

$$\begin{aligned} \mathsf{ATOM} &= \mathsf{PROP} \cup \mathsf{NOM}, \text{ and} \\ \mathsf{MOD} &= \mathsf{REL} \cup \{\mathsf{A}\} \cup \{@_i \mid i \in \mathsf{NOM}\} \cup \{\downarrow i \mid i \in \mathsf{NOM}\}, \end{aligned}$$

where $\mathsf{PROP} = \{p_1, p_2, \ldots\}$, $\mathsf{NOM} = \{n_1, n_2, \ldots\}$ and $\mathsf{REL} = \{r_1, r_2, \ldots\}$ are mutually disjoint, countable infinite sets. In what follows, we are interested in sublanguages of the language defined over $\mathcal{S}$ by Definition 3.15.

We define the following closed classes via their defining conditions. Table 3.1 shows the defining conditions for the universal modality A discussed in Example 3.6 and for different operators from hybrid logics [Areces and ten Cate, 2006].

Class	Defining condition
$\mathcal{C}_{\mathsf{A}}$	$\mathcal{P}_{\mathsf{A}}(\mathcal{M}) := R^{\mathcal{M}}(\mathsf{A}, w) = \{\langle v, \lvert\mathcal{M}\rvert, R^{\mathcal{M}}, V^{\mathcal{M}}\rangle \mid v \in \lvert\mathcal{M}\rvert\}$
$\mathcal{C}_{@_i}$	$\mathcal{P}_{@_i}(\mathcal{M}) := R^{\mathcal{M}}(@_i, w) = \{\langle v, \lvert\mathcal{M}\rvert, R^{\mathcal{M}}, V^{\mathcal{M}}\rangle \mid v \in V(i)\}, i \in \mathsf{NOM}$
$\mathcal{C}_{\downarrow i}$	$\mathcal{P}_{\downarrow i}(\mathcal{M}) := R^{\mathcal{M}}(\downarrow i, w) = \{\langle w, \lvert\mathcal{M}\rvert, R^{\mathcal{M}}, V^{\mathcal{M}}[i \mapsto \{w\}]\rangle\}, i \in \mathsf{NOM}$
$\mathcal{C}_{\mathsf{NOM}}$	$\mathcal{P}_{\mathsf{NOM}}(\mathcal{M}) := V^{\mathcal{M}}(i)$ is a singleton, $\forall i \in \mathsf{NOM}$

Table 3.1: Defining conditions for the universal modality and hybrid operators.

Notice, though, that the defining conditions we introduced above are of different kinds. Predicates $\mathcal{P}_{\mathsf{A}}$ and $\mathcal{P}_{@_i}$, for instance, define the accessibility relation by imposing conditions on the point of evaluation of the accessible models (and hence, the classes defined this way are subclasses of the class of relational modalities). This is just another way to state the fact that the semantics of the universal modality A, and satisfiability operators $@_i$ can all be captured on Kripke models by restricting evaluation to the class of models where the relation is, respectively, the total relation ($\forall xy.R(x,y)$) and the "point to all i" relation ($\forall xy.R(x,y) \leftrightarrow i(y)$). Observe that whenever the atom i is interpreted as a singleton set, the "point to all i" relation becomes the usual "point to i" relation ($\forall xy.R(x,y) \leftrightarrow y = i$) of hybrid logics.

Predicate $\mathcal{P}_{\downarrow i}$, on the other hand, imposes conditions on the valuation. In particular, $\mathcal{C}_{\downarrow i}$ is not a subclass of $\mathcal{C}^{\mathsf{K}}_{\downarrow i}$ (i. e., the class where $[\downarrow i]$ would be interpreted as a relational modality). For example, uniform substitution fails for $\mathcal{C}_{\downarrow i}$: while it is clear that $\mathcal{C}_{\downarrow i} \models [\downarrow i]i$, the uniform substitution of atom i by p yields the formula $[\downarrow i]p$ which is not $\mathcal{C}_{\downarrow i}$-valid.

Finally, predicate $\mathcal{P}_{\mathsf{NOM}}$ turns elements of NOM into nominals, i.e., true at a unique element of the domain of the model. Again, since $\mathcal{P}_{\mathsf{NOM}}$ imposes conditions on the valuation, unrestricted uniform substitution fails.

An interesting feature of this setting is that we can express the combination of modalities as the intersection of their respective classes. For example, $\mathcal{C}_{\mathcal{H}(@,\downarrow)}$, the class of models for the hybrid logic $\mathcal{H}(@,\downarrow)$, can be defined as follows:

$$\mathcal{C}_{\mathcal{H}(@,\downarrow)} = \mathcal{C}_{\mathsf{NOM}} \cap \mathcal{C}_{@} \cap \mathcal{C}_{\downarrow} \cap \mathcal{C}_{\mathsf{REL}}, \text{ where}$$
$$\mathcal{C}_{@} = \textstyle\bigcap_{i \in \mathsf{NOM}} \mathcal{C}_{@_i},\ \mathcal{C}_{\downarrow} = \bigcap_{i \in \mathsf{NOM}} \mathcal{C}_{\downarrow i}, \text{ and } \mathcal{C}_{\mathsf{REL}} = \bigcap_{m \in \mathsf{REL}} \mathcal{C}^{\mathsf{K}}_{m}.$$

Moreover, with this presentation we can define potentially interesting logics which have not been investigated before. For example, consider the logic of $\mathcal{C}_{@}$ (i. e., we take $[@_i]$ to be a jump-to-i operator but we do not restrict i to be interpreted as a singleton). Over this class, the $[@_i]$ operator behaves differently than the hybrid operator $@_i$. For example, $\mathcal{C}_{@} \not\models [@_i]\varphi \leftrightarrow \langle @_i \rangle \varphi$; i. e., $@_i$ is not self dual. But, $\mathcal{C}_{\mathsf{NOM}} \cap \mathcal{C}_{@} \models [@_i]\varphi \leftrightarrow \langle @_i \rangle \varphi$.

The crucial characteristic of the coinductive framework is that all these different modal operators are captured using the same semantic condition introduced in Definition 3.20. All the details defining each particular operator are now introduced as properties of the accessibility relation. As a result, a unique notion of bisimulation is sufficient to cover all of them.

Definition 3.23 (Bisimulations). *Given two models $\mathcal{M}$ and $\mathcal{M}'$ we say that $\mathcal{M}$* and *$\mathcal{M}'$* are bisimilar *(notation $\mathcal{M} \underline{\leftrightarrow} \mathcal{M}'$) if $\mathcal{M}\, Z\, \mathcal{M}'$ for some relation $Z \subseteq \mathrm{Ext}(\mathcal{M}) \times \mathrm{Ext}(\mathcal{M}')$ such that if $\langle w, W, R, V \rangle\, Z \langle w', W', R', V' \rangle$ then:*

i) $w \in V(a)$ iff $w' \in V'(a)$, for all $a \in \mathsf{ATOM}$. *[Atomic Harmony]*

ii) $\mathcal{N} \in R(m, w)$ implies $\mathcal{N}\, Z\, \mathcal{N}'$ for some $\mathcal{N}' \in R'(m, w')$. *[Zig]*

iii) $\mathcal{N}' \in R'(m, w')$ implies $\mathcal{N}\, Z\, \mathcal{N}'$ for some $\mathcal{N} \in R(m, w)$. *[Zag]*

Such Z is called a bisimulation between *$\mathcal{M}$* and *$\mathcal{M}'$.*

The classic result of invariance of modal formulas under bisimulation [Blackburn *et al.*, 2001] can easily be proved.

Theorem 3.4. *If $\mathcal{M} \underline{\leftrightarrow} \mathcal{M}'$, then $\mathcal{M} \models \varphi$ if and only if $\mathcal{M}' \models \varphi$, for all φ.*

It is interesting to observe that this general notion of bisimulation works for every modal logic definable as a closed subclass of Mods. In other words, this definition is capturing a variety of notions of bisimulation. Many well known bisimulations can be seen as specializations of Definition 3.23.

Example 3.7 (Bisimulation for $\mathcal{H}(@)$). *Consider the hybrid logic $\mathcal{H}(@)$. It is well known that if we want formulas of $\mathcal{H}(@)$ to be preserved by a bisimulation between hybrid models $\mathcal{M}$ and $\mathcal{N}$, we need to require (in addition to Atomic Harmony, Zig and Zag) that the bisimulation extends the relation $\{(i^{\mathcal{M}}, i^{\mathcal{N}}) \mid i \in \mathsf{NOM}\}$, where $i^{\mathcal{M}}$ (respectively $i^{\mathcal{N}}$) is the unique state of $\mathcal{M}$ (respectively $\mathcal{N}$) where i is true. Consider now the closed class*

$$\mathcal{C}_{\mathcal{H}(@)} = \mathcal{C}_{\mathsf{NOM}} \cap \mathcal{C}_{@} \cap \mathcal{C}_{Rel}$$

that corresponds to the class of (pointed) hybrid Kripke models.

Now, suppose we have $\mathcal{M} \underline{\leftrightarrow} \mathcal{N}$ for $\mathcal{M}, \mathcal{N} \in \mathcal{C}_{\mathcal{H}(@)}$. That means $\mathcal{M} Z \mathcal{N}$, for some bisimulation Z (cf. Definition 3.23).

We have to show that for all nominals i, $\langle i^{\mathcal{M}}, |\mathcal{M}|, R^{\mathcal{M}}, V^{\mathcal{M}} \rangle \; Z \; \langle i^{\mathcal{N}}, |\mathcal{N}|, R^{\mathcal{N}}, V^{\mathcal{N}} \rangle$ holds (and is well defined).

The defining condition $\mathcal{C}_{\mathsf{NOM}}$ ensures that the interpretation of i in each model is a singleton, and hence $i^{\mathcal{M}}$ and $i^{\mathcal{N}}$ are well defined. Now, the defining condition of the class $\mathcal{C}_{@}$ together with $\mathcal{M} Z \mathcal{N}$ let us infer $\langle i^{\mathcal{M}}, |\mathcal{M}|, R^{\mathcal{M}}, V^{\mathcal{M}} \rangle Z \langle i^{\mathcal{N}}, |\mathcal{N}|, R^{\mathcal{N}}, V^{\mathcal{N}} \rangle$ using either Zig or Zag.

We can even take a finer look, and adapt the notion of n-bisimulations to the present setting.

Definition 3.24 (n-bisimulations). *Given two models $\mathcal{M}$ and $\mathcal{M}'$ we say that $\mathcal{M}$* and *$\mathcal{M}'$* are n-bisimilar *(notation $\mathcal{M} \underline{\leftrightarrow}_n \mathcal{M}'$) if there exists a sequence of binary relations $Z_0 \supseteq Z_1 \supseteq \cdots \supseteq Z_n$ such that $\mathcal{M} Z_n \mathcal{M}'$ and for all $\mathcal{N} = \langle w, W, R, V \rangle \in \mathrm{Ext}(\mathcal{M})$ and $\mathcal{N}' = \langle w', W', R', V' \rangle \in \mathrm{Ext}(\mathcal{M}')$:*

i) $\mathcal{N} Z_0 \mathcal{N}'$ implies $w \in V(a)$ iff $w' \in V'(a)$, for all $a \in \mathsf{ATOM}$. [Atomic Harmony]

ii) $\mathcal{N} Z_{i+1} \mathcal{N}'$ and $\mathcal{N}_2 \in R(m, w)$ implies $\mathcal{N}_2 Z_i \mathcal{N}_2'$ for some $\mathcal{N}_2' \in R'(m, w')$. [Zig]

iii) $\mathcal{N} Z_{i+1} \mathcal{N}'$ and $\mathcal{N}_2' \in R'(m, w')$ implies $\mathcal{N}_2 Z_i \mathcal{N}_2'$ for some $\mathcal{N}_2 \in R(m, w)$. [Zag]

*Such a sequence is called a n-*bisimulation between *$\mathcal{M}$* and *$\mathcal{M}'$.*

As we have seen in Section 2.4, one of the interesting properties of n-bisimulations is that they preserve modal formulas up to a certain *modal depth*. This result can be transfered nicely to the coinductive setting, and, once more, it now applies to any type of modality, on any closed class, and not only to relational modalities.

Theorem 3.5 (Invariance under n-bisimulations). *If $\mathcal{M} \underline{\leftrightarrow}_n \mathcal{M}'$, then $\mathcal{M} \models \varphi$ if and only if $\mathcal{M}' \models \varphi$, for all φ such that $md(\varphi) \leq n$.*

This concludes our introduction to coinductive modal models. We have now in place all the tools we need to tackle the next step: generalize the theory of symmetries to richer modal logics.

3.2.2 *A Generalized Theory of Symmetries*

Let us start by introducing the basic notions. As for the basic modal logic we work with sets of literals.

Definition 3.25 (Complete, Consistent and Generated sets of literals). *A set of literals L is* complete *if for each $a \in \mathsf{ATOM}$ either $a \in L$ or $\neg a \in L$. It is* consistent *if for each $a \in \mathsf{ATOM}$ either $a \notin L$ or $\neg a \notin L$. Any complete and consistent set of literals L defines a unique valuation $v \subseteq \mathsf{ATOM}$ as $a \in v$ if $a \in L$ and $a \notin v$ if $\neg a \in L$. For $S \subseteq \mathsf{ATOM}$, the* consistent and complete set of literals generated by *S (notation L_S) is $S \cup \{\neg a \mid a \in \mathsf{ATOM} \backslash S\}$.*

As expected, we need to adapt our definition of permutation because from now on we work with permutation of atom literals.

Definition 3.26 (Permutation of atom literals). *A* permutation *is a bijective function $\sigma : \mathsf{ALIT} \mapsto \mathsf{ALIT}$. For L a set of atom literals, $\sigma(L) = \{\sigma(l) \mid l \in L\}$.*

In what follows we assume that every permutation is a permutation of atom literals and just call them a "permutation".

Since, in our language, atoms may occur in some modalities (like $@_i$) we should take some care when applying permutations to modal formulas. We say that a modality is *indexed by atoms* if its definition depends on the value of an atom. If m is indexed by an atom a we write $m(a)$.

Definition 3.27 (Permutation of a formula). *Let φ be a modal CNF formula and σ a permutation. We define $\sigma(\varphi)$ recursively as*

$$\begin{array}{rcll} \sigma(\varphi) & = & \{\sigma(C) \mid C \in \varphi\} & \textit{for } \varphi \textit{ a modal CNF formula} \\ \sigma(C) & = & \{\sigma(A) \mid A \in C\} & \textit{for } C \textit{ a modal CNF clause} \\ \sigma([m]C) & = & [\sigma(m)]\sigma(C) & \\ \sigma(\neg[m]C) & = & \neg[\sigma(m)]\sigma(C) & \end{array}$$

where $\sigma(m) = \sigma(m(a)) = m(\sigma(a))$ if m is indexed by a, and $\sigma(m) = m$ otherwise.

As for the basic modal logic, we restrict ourselves to work with consistent symmetries.

Definition 3.28 (Consistent Permutations and Symmetries). *A permutation σ is* consistent *if for every literal l, $\sigma(\sim l) = \sim\sigma(l)$, where $\sim$ is a function that returns the complement of an atom literal, i. e., $\sim a = \neg a$ and $\sim\neg a = a$. A permutation σ is a* symmetry *for φ if $\varphi = \sigma(\varphi)$, when conjunctions and disjunctions in φ are represented as sets.*

Now we are ready to generalize the results obtained for the basic modal logic to the coinductive framework. We begin with the notion of σ-simulation.

Definition 3.29 (σ-simulation). *Let σ be a permutation. A* σ-simulation *between models $\mathcal{M} = \langle w, W, R, V\rangle$ and $\mathcal{M}' = \langle w', W', R', V'\rangle$ is a non-empty relation $Z \subseteq \mathrm{Ext}(\mathcal{M}) \times \mathrm{Ext}(\mathcal{M}')$ that satisfies the following conditions:*

i) $\mathcal{M} Z \mathcal{M}'$. **[Root]** *ii)* $l \in L_{V(w)}$ *iff* $\sigma(l) \in L_{V'(w')}$. *[Atomic Harmony]*

iii) If $\mathcal{M} Z \mathcal{M}'$ *and* $\mathcal{N} \in R(m,w)$ *then* $\mathcal{N} Z \mathcal{N}'$ *for some* $\mathcal{N}' \in R'(\sigma(m), w')$. *[Zig]*

iv) If $\mathcal{M} Z \mathcal{M}'$ *and* $\mathcal{N}' \in R'(m,w')$ *then* $\mathcal{N} Z \mathcal{N}'$ *for some* $\mathcal{N} \in R(\sigma^{-1}(m), w)$. *[Zag]*

We say that two models $\mathcal{M}$ *and* $\mathcal{M}'$ *are* σ*-similar (notation* $\mathcal{M} \underline{\rightarrow}_\sigma \mathcal{M}'$*) if there is a* σ*-simulation Z between them.*

Notice that the definition of σ-simulation takes into account the fact that there exist modalities that are indexed by atoms by considering the permutation when accessing to the successors, e. g., $R'(\sigma(m), w')$.

As for the basic modal logic case, σ-simulations preserve validity of permutations of formulas.

Proposition 3.4. *Let* σ *be a consistent permutation,* φ *a modal CNF formula and* $\mathcal{M} = \langle w, W, R, V\rangle$, $\mathcal{M}' = \langle w', W', R', V'\rangle$ *models such that* $\mathcal{M} \underline{\rightarrow}_\sigma \mathcal{M}'$. *Then* $\mathcal{M} \models \varphi$ *if and only if* $\mathcal{M}' \models \sigma(\varphi)$.

Proof. The proof is by induction on φ.

Base Case:

- Suppose $\varphi = a$ then, $\mathcal{M} \models a$ iff $a \in V(w)$ iff $a \in L_{V(w)}$ iff, by definition of σ-simulation, $\sigma(a) \in L_{V'(w')}$ iff $\mathcal{M}' \models \sigma(a)$.
- Suppose $\varphi = \neg a$ then, $\mathcal{M} \models \neg a$ iff $a \notin V(w)$ iff $\neg a \in L_{V(w)}$ iff, by definition of σ-simulation, $\sigma(\neg a) = \neg\sigma(a) \in L_{V'(w')}$ iff $\sigma(a) \notin V'(w')$ iff $\mathcal{M}' \models \neg\sigma(a)$.

Inductive Step:

- When $\varphi = C$, with C a clause or a conjunction of clauses, the proof follows by induction directly.
- Suppose $\varphi = [m]\psi$. Then $\mathcal{M} \models [m]\psi$ iff $\mathcal{N} \models \psi$ for all $\mathcal{N} \in R(m,w)$. Given that $\mathcal{M} \underline{\rightarrow}_\sigma \mathcal{M}'$, by Zig we know that for all $\mathcal{N}$ exist $\mathcal{N}'$ such that $\mathcal{N} \underline{\rightarrow}_\sigma \mathcal{N}'$ and $\mathcal{N}' \in R'(\sigma(m), w')$. Then, by inductive hypothesis, $\mathcal{N}' \models \sigma(\psi)$ for all $\mathcal{N}' \in R'(\sigma(m), w')$ iff $\mathcal{M}' \models [\sigma(m)]\sigma(\psi)$. Then, by Definition 3.27, $\mathcal{M}' \models \sigma([m]\psi)$. The converse uses Zag and the inductive hypothesis.
- Suppose $\varphi = \neg[m]\psi$. Then $\mathcal{M} \models \neg[m]\psi$ iff there exists $\mathcal{N} \in R(m,w)$ such that, $\mathcal{N} \models \neg\psi$. Given that $\mathcal{M} \underline{\rightarrow}_\sigma \mathcal{M}'$, by Zig, for all $\mathcal{N}$ exist $\mathcal{N}'$ such that $\mathcal{N} \underline{\rightarrow}_\sigma \mathcal{N}'$ and $\mathcal{N}' \in R'(\sigma(m), w')$. Then, by inductive hypothesis, $\mathcal{N}' \models \sigma(\neg\psi) = \neg\sigma(\psi)$ iff $\mathcal{M}' \models \neg[\sigma(m)]\sigma(\psi)$. Then, by Definition 3.27, $\mathcal{M}' \models \sigma(\neg[m]\psi)$. The converse follows using Zag and the inductive hypothesis.

□

The next step is to define the notion of applying permutations to coinductive modal models.

Definition 3.30 (Permutation of a model)**.** *Let σ be a permutation and $\mathcal{M} = \langle w, W, R, V\rangle$ a model. Then $\sigma(\mathcal{M}) = \langle w, W, R', V'\rangle$, where,*

$$\begin{aligned} V'(v) &= \sigma(L_{V(v)}) \cap \mathsf{ATOM} \quad \textit{for all } v \in W\textit{, and,} \\ R'(m, v) &= \{\sigma(\mathcal{N}) \mid \mathcal{N} \in R(\sigma(m), v)\} \quad \textit{for all } m \in \mathsf{MOD} \textit{ and } v \in W. \end{aligned}$$

For M a set of models, $\sigma(M) = \{\sigma(\mathcal{M}) \mid \mathcal{M} \in M\}$.

Besides modifying the valuation, permuting a coinductive modal model involves propagating the permutation to all accessible models.

It follows from the previous definition that $\mathcal{M}$ and $\sigma(\mathcal{M})$ are always σ-similar.

Proposition 3.5. *Let σ be a consistent permutation and $\mathcal{M} = \langle w, W, V, R\rangle$ a model. Then $\mathcal{M} \,\underline{\rightarrow}_\sigma\, \sigma(\mathcal{M})$.*

Proof. Let us define the relation $Z = \{(\mathcal{N}, \sigma(\mathcal{N})) \mid \mathcal{N} \in \text{Ext}(\mathcal{M})\}$ and show that it is a σ-simulation between $\mathcal{M}$ and $\sigma(\mathcal{M})$. The *Zig* and *Zag* conditions are trivial by definition of $\sigma(\mathcal{M})$.

- Atomic Harmony: We have to check that $l \in L_{V(w)}$ iff $\sigma(l) \in L_{V'(w)}$. From the definition of $\sigma(\mathcal{M})$, $L_{V'(w)} = \sigma(L_{V(w)})$, hence if $l \in L_{V(w)}$ then $\sigma(l) \in \sigma(L_{V(w)})$. Moreover, $\sigma(L_{V(w)})$ is a complete set of literals because $L_{V(w)}$ is a complete set of literals and σ is a consistent permutation, and hence the converse also follows.

□

Proposition 3.6. *Let σ be a consistent permutation, $\mathcal{M}$ a model and φ a modal CNF formula. Then $\mathcal{M} \models \varphi$ if and only if $\sigma(\mathcal{M}) \models \sigma(\varphi)$.*

Proof. From Proposition 3.5 ($\mathcal{M} \,\underline{\rightarrow}_\sigma\, \sigma(\mathcal{M})$) and Proposition 3.4. □

The previous results lead us to the desired corollary.

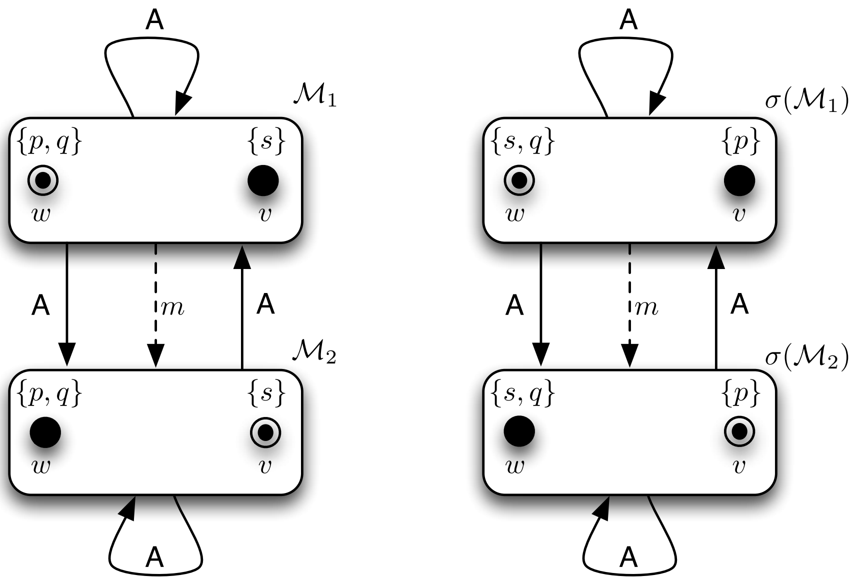
Corollary 3.2. *If σ is a symmetry of φ then $\mathcal{M} \in \mathsf{Mods}(\varphi)$ if and only if $\sigma(\mathcal{M}) \in \mathsf{Mods}(\varphi)$.*

Corollary 3.2 tells us that symmetries have the same effect on coinductive modal models as for the basic modal logic. They partition the space of models into equivalence classes in such a way that every equivalence class contains either models or non-models. Nevertheless, the importance of this corollary lies in the fact that now we can apply it to a number of modal logics, i. e., those modal logics that can be cast to the coinductive framework.

To clarify the implications of the Corollary 3.2, consider the following example.

Example 3.8. *Let $\varphi = (p \vee q \vee r) \wedge (s \vee q \vee r) \wedge (\neg p \vee \neg s) \wedge \langle m\rangle(p \vee s) \wedge [\mathsf{A}](\neg r)$. From Figure 3.2a we can verify that $\mathcal{M}_1 \models \varphi$.*

Now $\sigma = (p\ s)(\neg p\ \neg s)$ is a symmetry of φ. Then, by Corollary 3.2, we have $\sigma(\mathcal{M}_1) \models \varphi$, which can be verified in the model of Figure 3.2b.



(a) $\mathcal{M}_1$. (b) $\sigma(\mathcal{M}_1)$.

Figure 3.2: A model and its symmetric model.

Let us move forward and prove that, in the coinductive framework, symmetries preserve inference. But before a brief remark: The notion of σ-simulation in coinductive modal models is general enough to be applicable to a wide range of modal logics. Notice though, that our definition of σ-simulation makes no assumption about the models being in the same class. Consider, for example, a model $\mathcal{M} \in \mathcal{C}_{\mathcal{H}(@)}$ and an arbitrary permutation $\sigma = (i\ p)(\neg i\ \neg p)$ for $i \in \mathsf{NOM}$, $p \in \mathsf{PROP}$. By the defining condition $\mathcal{C}_{\mathcal{H}(@)}$, nominals in $\mathcal{M}$ are true at a unique element in the domain, but this does not necessary hold for $\sigma(\mathcal{M})$, and hence $\sigma(\mathcal{M})$ might not be in $\mathcal{C}_{\mathcal{H}(@)}$.

However, when working with symmetries of a formula, this is not an issue, and we can be sure that every model $\mathcal{M}$ and its symmetric $\sigma(\mathcal{M})$ are in the same model class. This is so because a symmetry of a formula is, implicitly, a *typed* permutation: it only maps symbols such that the resulting formula is a formula of the language. If this is not the case, either the formula at hand is not in the language of the logic, or the permutation is not a symmetry. Therefore, we can think of the language definition as restricting the possible mappings.

Example 3.9. *Consider the formula $\varphi = [@_i]p$ and the permutation $\sigma = (i\ p)(\neg i\ \neg p)$ for $i \in \mathsf{NOM}$, $p \in \mathsf{PROP}$. Clearly φ is a formula of the language defined in Section 3.2.1.1, but $\sigma(\varphi) = [@_p]i$ is not. Now consider the formula $\varphi = i \vee p$ and the same permutation σ, only that now σ is a symmetry of φ (modulo commutativity of $\vee$). We can see that $\sigma(\varphi) = p \vee i$ is a formula of the language, and clearly, a model $\mathcal{M}$ satisfying φ and its symmetric $\sigma(\mathcal{M})$ must be in the same model class.*

Everything is now in place to show that modal entailment is preserved under symmetries.

Theorem 3.6. *Let φ and ψ be modal formulas, let σ be a consistent symmetry of φ. Then $\varphi \models \psi$ if and only if $\varphi \models \sigma(\psi)$.*

Proof. We first show that under the hypothesis of the theorem the following property holds

Claim: $\mathsf{Mods}(\varphi) = \sigma(\mathsf{Mods}(\varphi))$.

[$\supseteq$] Let $\mathcal{N} \in \sigma(\mathsf{Mods}(\varphi))$ and $\mathcal{M} \in \mathsf{Mods}(\varphi)$ be such that $\mathcal{N} = \sigma(\mathcal{M})$. Then $\mathcal{M} \models \varphi$ and by Corollary 3.2, $\sigma(\mathcal{M}) \models \varphi$, therefore, $\sigma(\mathcal{M}) \in \mathsf{Mods}(\varphi)$.

[$\subseteq$] Let $\mathcal{M} \in \mathsf{Mods}(\varphi)$, then $\mathcal{M} \models \varphi$. By Corollary 3.2, $\sigma(\mathcal{M}) \models \varphi$, therefore, $\sigma(\mathcal{M}) \in \mathsf{Mods}(\varphi)$. Since σ is arbitrary, the results holds also for σ^k, $k \in \mathbb{Z}$.

Since σ is a permutation over a finite set, there exists n such that σ^n is the identity permutation. Now consider $\sigma^{n-1}(\mathcal{M})$, we know $\sigma^{n-1}(\mathcal{M}) \in \mathsf{Mods}(\varphi)$. Hence $\sigma^n(\mathcal{M}) = \mathcal{M} \in \sigma(\mathsf{Mods}(\varphi))$.

Now, we have to prove that $\varphi \models \psi$ iff $\varphi \models \sigma(\psi)$. By definition, $\varphi \models \psi$ iff $\mathsf{Mods}(\varphi) \models \psi$. By Proposition 3.6, this is the case if and only if $\sigma(\mathsf{Mods}(\varphi)) \models \sigma(\psi)$.

Given that σ is a symmetry of φ, by the Claim above, $\sigma(\mathsf{Mods}(\varphi)) \models \sigma(\psi)$ iff $\mathsf{Mods}(\varphi) \models \sigma(\psi)$, which by definition means $\varphi \models \sigma(\psi)$. $\square$

As for the basic modal logic case, Theorem 3.6 provides a "cheap" inference mechanism, but now, it does so for a wide range of modal logics.

3.2.3 *Layered Permutations*

We now present the notion of *layered permutations* that, in cases where the modal language has a tree model property, enables us to develop a more flexible notion of symmetry. The key idea is to use different permutations at each modal depth. Allowing symmetries that would not be found otherwise.

We start by presenting a definition of the tree model property [Blackburn *et al.*, 2001] for coinductive modal models.

Definition 3.31 (Paths in a model). *Given a model $\mathcal{M}$, a (finite)* path rooted at $\mathcal{M}$ *is a sequence $\pi = (\mathcal{M}_0, m_1, \mathcal{M}_1, \ldots, m_k, \mathcal{M}_k)$, for $m_i \in \mathsf{MOD}$ where $\mathcal{M}_0 = \mathcal{M}$, $k \geq 0$, and $\mathcal{M}_i \in R(m_i, w^{\mathcal{M}_{i-1}})$ for $i = 1, \ldots, k$. For a path $\pi = (\mathcal{M}_0, m_1, \mathcal{M}_1, \ldots, m_k, \mathcal{M}_k)$ we define $first(\pi) = \mathcal{M}_0$, $last(\pi) = \mathcal{M}_k$, and $length(\pi) = k$. For a path $\pi = (\mathcal{M}_0, m_1, \mathcal{M}_1, \ldots, m_k, \mathcal{M}_k)$, a model $\mathcal{N}$ and a modality $m \in \mathsf{MOD}$, such that $\mathcal{N} \in R(m, w^{\mathcal{M}_k})$, by $\pi m \mathcal{N} = (\mathcal{M}_0, m_1, \mathcal{M}_1, \ldots, m_k, \mathcal{M}_k, m, \mathcal{N})$ we denote the extension of π by $\mathcal{N}$ through m. We denote the set of all paths rooted at $\mathcal{M}$ as $\Pi[\mathcal{M}]$.*

A *coinductive tree model* is a model that has a unique path to every reachable model (every model in $\mathrm{Ext}(\mathcal{M})$). Formally we can define the class of all coinductive tree models, $\mathcal{C}_{Tree}$, with the following defining condition:

$$\mathcal{C}_{Tree} := P_{Tree}(\mathcal{M}) \iff last : \Pi[\mathcal{M}] \mapsto \mathrm{Ext}(\mathcal{M}) \text{ is bijective.}$$

For example, the *unravelling* construction (in its version for coinductive modal models) shown below always defines a model in $\mathcal{C}_{Tree}$.

Definition 3.32 (Model Unravelling). *Given a model $\mathcal{M} = \langle w, W, R, V \rangle$, the* unravelling *of $\mathcal{M}$, (notation $\mathcal{T}(\mathcal{M})$), is the rooted coinductive modal model $\mathcal{T}(\mathcal{M}) = \langle (\mathcal{M}), \Pi[\mathcal{M}], R', V' \rangle$ where*

$$\begin{aligned} V'(\pi) &= V(w^{last(\pi)}), \textit{ for all } \pi \in \Pi[\mathcal{M}], \\ R'(m, \pi) &= \{\langle \pi', \Pi[\mathcal{M}], R', V' \rangle \mid \textit{ exists } \mathcal{N} \in \mathit{Mods}_W \textit{ s.t. } \pi m \mathcal{N} = \pi', \pi \neq \pi'\}, \end{aligned}$$

for $m \in \mathsf{MOD}, \pi \in \Pi[\mathcal{M}]$.

It is easy to verify that given a model $\mathcal{M}$, its unravelling $\mathcal{T}(\mathcal{M})$ is a tree ($\mathcal{T}(\mathcal{M}) \in \mathcal{C}_{Tree}$) and, as expected, $\mathcal{M}$ and $\mathcal{T}(\mathcal{M})$ are bisimilar.

In what follows, we use trees to define a more flexible type of symmetries that we call *layered symmetries*. The following gives a sufficient condition ensuring that layered symmetries also preserve entailment.

Definition 3.33 (Tree model closure property). *We say that a class $\mathcal{C}$ of models is* closed under trees *if for every model $\mathcal{M} \in \mathcal{C}$ exists a tree model $\mathcal{T} \in \mathcal{C}$ such that $\mathcal{M} \underline{\leftrightarrow} \mathcal{T}$.*

From Definition 3.33 it follows that a class of models $\mathcal{C}$ closed under unravellings ($\mathcal{T}(\mathcal{M}) \in \mathcal{C}$ for all $\mathcal{M} \in \mathcal{C}$) is also closed under trees.

Example 3.10. *Trivially the class* Mods *(i.e., the basic modal logic) is closed under trees, and so does the class $\mathcal{C}_{\mathsf{KAlt}_1}$ of models where the accessibility relation is a partial function. Many classes like $\mathcal{C}_{\mathsf{A}}$, $\mathcal{C}_{@_i}$ and $\mathcal{C}_{\mathsf{NOM}}$ fail to be closed under trees.*

Logics defined over classes closed under trees have an interesting property: there is a direct correlation between the syntactical modal depth of the formula and the depth in a tree model satisfying it. In tree models, a notion of layer is induced by the depth (distance from the root) of the nodes in the model. Similarly, in modal formulas, a notion of layer is induced by the nesting of the modal operators. A consequence of this correspondence is that literals occurring at different layers of the formula are semantically independent of each other [Areces *et al.*, 2000b], i.e., at different layers the same literal can be assigned a different value.

Example 3.11. *Consider the formula $\varphi = (p \vee q) \wedge (r \vee \neg\Box(\neg p \vee q \vee \Box\neg r))$ and a tree model $\mathcal{M}$ of φ. Figure 3.3 shows the layers induced by the modal depth of the formula and the corresponding depth in $\mathcal{M}$.*

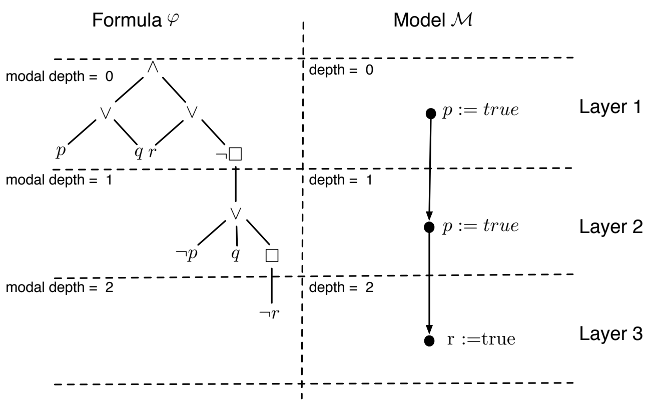


Figure 3.3: Induced layering on a model and a formula.

The independence between literals at different layers enables us to give a more flexible notion of a permutation that we call *layered permutation*. Key to the notion of layered permutation is that of a *permutation sequence*.

Definition 3.34 (Permutation Sequence). *We define a finite permutation sequence $\bar{\sigma}$ as either $\bar{\sigma} = \langle\rangle$ (i.e., $\bar{\sigma}$ is the empty sequence) or $\bar{\sigma} = \sigma : \bar{\sigma}_2$ with σ a permutation and $\bar{\sigma}_2$ a permutation sequence. Alternatively we can use the notation $\bar{\sigma} = \langle\sigma_1, \ldots, \sigma_n\rangle$ instead of $\bar{\sigma} = \sigma_1 : \ldots : \sigma_n : \langle\rangle$.*

Let $|\bar{\sigma}| = n$ be the length of $\bar{\sigma}$ ($\langle\rangle$ has length 0). For $1 \leq i \leq n$, we write $\bar{\sigma}_i$ for the subsequence that starts from the i^{th} element of $\bar{\sigma}$. For $i \geq n$, we define $\bar{\sigma}_i = \langle\rangle$. In particular $\bar{\sigma} = \bar{\sigma}_1$. Given a permutation sequence $\sigma_1 : \bar{\sigma}_2$ we define $head(\sigma_1 : \bar{\sigma}_2) = \sigma_1$ and $head(\langle\rangle) = \sigma_{Id}$, where σ_{Id} is the identity permutation. We say that a permutation sequence is consistent *if all of its permutations are consistent.*

Applying a permutation sequence to a modal CNF formula is defined as follows:

Definition 3.35 (Layered permutation of a formula). *Let φ be a modal CNF formula and $\bar{\sigma}$ a permutation sequence. We define $\bar{\sigma}(\varphi)$ recursively as*

$$\begin{array}{rcll} \langle\rangle(\varphi) & = & \varphi & \\ (\sigma_1 : \bar{\sigma}_2)(l) & = & \sigma_1(l) & \textit{for } l \in \mathsf{ALIT} \\ (\sigma_1 : \bar{\sigma}_2)([m]C) & = & [\sigma_1(m)]\bar{\sigma}_2(C) & \\ \bar{\sigma}(C) & = & \{\bar{\sigma}(A) \mid A \in C\} & \textit{for } C \textit{ a clause or a formula.} \end{array}$$

Notice that layered permutations are well defined even if the modal depth of the formula is greater than the size of the permutation sequence. Layered permutations let us use a different permutation at each modal depth. This enables symmetries (layered symmetries) to be found, that would not be found otherwise.

Example 3.12. *Consider the formula $\varphi = (p \vee [m](p \vee \neg r)) \wedge (\neg q \vee [m](\neg p \vee r))$. If we only consider non-layered symmetries then φ has none. However, the permutation sequence $\langle\sigma_1, \sigma_2\rangle$ generated by $\sigma_1 = (p\ \neg q)(\neg p\ q)$ and $\sigma_2 = (p\ \neg r)(\neg p\ r)$ is a layered symmetry of φ.*

As we can see from the previous example, layered permutations let us map the same literal to different targets at each different modal depth. This additional degree of freedom can result in new symmetries for a given formula.

From now on we can mostly repeat the work we did in the previous section to arrive to a result similar to Theorem 3.6 but involving permutation sequences, with one caveat: the obvious extension of the notion of permuted model $\sigma(\mathcal{M})$ to layered permutations is ill defined if $\mathcal{M}$ is not a tree. Hence, we need the additional requirement that the class $\mathcal{C}$ of models is closed under trees for the result to go through.

Definition 3.36 (Layered Permutation of a model). *Let $\bar{\sigma}$ be a permutation sequence and $\mathcal{M} = \langle w, W, R, V\rangle$ a tree model. Then $\bar{\sigma}(\mathcal{M}) = \langle w, W, R', V'\rangle$, where,*

$$\begin{array}{rcll} V'(v) & = & head(\bar{\sigma})(L_{V(v)}) \cap \mathsf{ATOM} & \textit{for all } v \in W, \textit{ and,} \\ R'(m, v) & = & \{\bar{\sigma}_2(\mathcal{N}) \mid \mathcal{N} \in R(head(\bar{\sigma})(m), v)\} & \textit{for all } m \in \mathsf{MOD} \textit{ and } v \in W. \end{array}$$

For M a set of tree models, $\bar{\sigma}(M) = \{\bar{\sigma}(\mathcal{M}) \mid \mathcal{M} \in M\}$.

We can now extend the notion of σ-simulation to permutation sequences.

Definition 3.37 ($\bar{\sigma}$-simulation)**.** *Let $\bar{\sigma}$ be a permutation sequence. A* $\bar{\sigma}$-simulation *between models $\mathcal{M} = \langle w, W, R, V\rangle$ and $\mathcal{M}' = \langle w', W', R', V'\rangle$ is a family of relations $Z_{\bar{\sigma}_i} \subseteq \mathrm{Ext}(\mathcal{M}) \times \mathrm{Ext}(\mathcal{M}')$, $1 \leq i$, that satisfies the following conditions:*

i) $\mathcal{M} Z_{\bar{\sigma}_1} \mathcal{M}'$. *[Root]*

ii) If $w Z_{\bar{\sigma}_i} w'$ then $l \in L_{V(w)}$ iff $head(\bar{\sigma}_i)(l) \in L_{V'(w')}$. *[Atomic Harmony]*

iii) If $\mathcal{M} Z_{\bar{\sigma}_i} \mathcal{M}'$ and $\mathcal{N} \in R(m, w)$ then $\mathcal{N} Z_{\bar{\sigma}_{i+1}} \mathcal{N}'$ for some $\mathcal{N}' \in R'(head(\bar{\sigma}_i)(m), w')$. *[Zig]*

iv) If $\mathcal{M} Z_{\bar{\sigma}_i} \mathcal{M}'$ and $\mathcal{N}' \in R'(m, w')$ then $\mathcal{N} Z_{\bar{\sigma}_{i+1}} \mathcal{N}'$ for some $\mathcal{N} \in R(head(\bar{\sigma}_i)^{-1}(m), w)$. *[Zag]*

We say that two models $\mathcal{M}$ and $\mathcal{M}'$ are $\bar{\sigma}$-similar (notation $\mathcal{M} \underline{\rightarrow}_{\bar{\sigma}} \mathcal{M}'$), if there is a $\bar{\sigma}$-simulation between them.

An important remark about the previous definition is that it does not make any assumption about the size of the permutation sequence. In fact, it is well defined even if the permutation sequence at hand is the empty sequence. In that case, it just behaves as the identity permutation at each layer, therefore the relation defines a bisimulation between the models.

Now we are ready to prove the main result concerning layered symmetries and entailment.

Theorem 3.7. *Let φ and ψ be modal formulas and let $\bar{\sigma}$ be a consistent permutation sequence, and let $\mathcal{C}$ be a class of models closed under trees. If $\bar{\sigma}$ is a symmetry of φ then for any ψ we have that $\varphi \models_{\mathcal{C}} \psi$ if and only if $\varphi \models_{\mathcal{C}} \bar{\sigma}(\psi)$.*

Proof. We first show that under the hypothesis of the theorem the following two properties hold.

Claim 1: $\mathsf{Mods}_{\mathcal{C}\cap\mathcal{C}_{Tree}}(\varphi) = \bar{\sigma}(\mathsf{Mods}_{\mathcal{C}\cap\mathcal{C}_{Tree}}(\varphi))$.

The argument is the same as for the Claim in Theorem 3.6 but using permutation sequences.

Claim 2: $\mathsf{Mods}_{\mathcal{C}}(\varphi) \models_{\mathcal{C}} \varphi$ iff $\mathsf{Mods}_{\mathcal{C}\cap\mathcal{C}_{Tree}}(\varphi) \models_{\mathcal{C}} \varphi$.

The left-to-right direction is trivial by the fact that $\mathsf{Mods}_{\mathcal{C}\cap\mathcal{C}_{Tree}}(\varphi) \subseteq \mathsf{Mods}_{\mathcal{C}}(\varphi)$. For the other direction, assume $\mathsf{Mods}_{\mathcal{C}\cap\mathcal{C}_{Tree}}(\varphi) \models_{\mathcal{C}} \varphi$ and $\mathsf{Mods}_{\mathcal{C}}(\varphi) \not\models_{\mathcal{C}} \varphi$. Then exists $\mathcal{M} \in \mathsf{Mods}_{\mathcal{C}}(\varphi)$ such that $\mathcal{M} \not\models_{\mathcal{C}} \varphi$. But we know that $\mathcal{M} \underline{\leftrightarrow} \mathcal{T}$, and $\mathcal{T} \in \mathsf{Mods}_{\mathcal{C}\cap\mathcal{C}_{Tree}}(\varphi)$. Hence $\mathcal{T} \models_{\mathcal{C}} \varphi$ which contradicts our assumption.

It remains to prove that $\varphi \models_{\mathcal{C}} \psi$ if and only if $\varphi \models_{\mathcal{C}} \bar{\sigma}(\psi)$.

By definition, $\varphi \models_{\mathcal{C}} \psi$ if and only if $\mathsf{Mods}_{\mathcal{C}}(\varphi) \models_{\mathcal{C}} \psi$. By Claim 2, this is case if and only if $\mathsf{Mods}_{\mathcal{C}\cap\mathcal{C}_{Tree}}(\varphi) \models_{\mathcal{C}} \psi$. By the layered version of Proposition 3.6, this is the case if and only if $\bar{\sigma}(\mathsf{Mods}_{\mathcal{C}\cap\mathcal{C}_{Tree}}(\varphi)) \models_{\mathcal{C}} \bar{\sigma}(\psi)$. Given that $\bar{\sigma}$ is a symmetry of φ, by Claim 1, $\bar{\sigma}(\mathsf{Mods}_{\mathcal{C}\cap\mathcal{C}_{Tree}}(\varphi)) \models_{\mathcal{C}} \bar{\sigma}(\psi)$ if and only if $\mathsf{Mods}_{\mathcal{C}\cap\mathcal{C}_{Tree}}(\varphi) \models_{\mathcal{C}} \bar{\sigma}(\psi)$, which by Claim 2 is the case if and only if $\mathsf{Mods}_{\mathcal{C}}(\varphi) \models_{\mathcal{C}} \bar{\sigma}(\psi)$ which by definition means that $\varphi \models_{\mathcal{C}} \bar{\sigma}(\psi)$. □

3.3 SUMMARY

In this chapter we have presented the theoretical foundations for exploiting symmetries in modal logics.

First we showed two interesting results for the basic modal logic: that symmetries partition the model space into equivalence classes, and that symmetries can be used as an inference mechanism. Both results show that symmetries in the basic modal logic have the same behavior as symmetries in propositional logic. This is expected, as we are dealing with symmetries of propositional literals.

We then extended the results to a broad range of modal logics using the coinductive framework and introduced a more flexible notion of symmetry, i. e., layered symmetries, for modal logics enjoying the tree model property.

It now remains to see how to profit from these theoretical properties in a modal prover.

4 SATISFIABILITY MODULO THEORIES

In this chapter we introduce the basic notions of Satisfiability Modulo Theories (SMT). We begin by presenting the motivations behind SMT (Section 4.1). Then we introduce the necessary definitions about first-order logic and first-order theories (Section 4.2). We continue with a bird-eye-view on the *lazy* approach to SMT, describing its main features (Section 4.3). Finally, we present a survey on symmetries in SMT (Section 4.4). For a comprehensive treatment and for references to the literature on the subject, we refer the reader to [Sebastiani, 2007; Barrett *et al.*, 2009] on which this chapter is based.

4.1 MOTIVATION

Satisfiability, i. e., the problem of determining whether a formula expressing a constraint has a solution, is one of the fundamental problems in theoretical computer science. Satisfiability problems, also known as Constraint Satisfaction Problems (CSP), arise in many diverse areas ranging from hardware and software verification to planning.

Propositional Satisfiability (SAT) (also known as Boolean Satisfiability) is one of the most investigated constraint satisfaction problems. In SAT the goal is to decide whether a propositional formula (or SAT formula) is satisfiable, i. e., whether it can become true, by assigning truth values to its variables. The tools implementing the specialized algorithms to check the satisfiability of a propositional formula are called *SAT solvers*.

During the last decade, SAT have received a lot of attention from the research community. This led to impressive improvements in the efficiency of the available SAT solvers, and many applications in artificial intelligence and formal methods for hardware and software development have greatly benefited from those advances.

However, some problems are more naturally described in more expressive logics such as first-order logic. Intuitively one might think these problems could be handled by general-purpose first-order theorem provers, e. g., provers based on the resolution calculus, but usually this is not the case. The main reason for this is that many applications do not require general first-order satisfiability, but rather satisfiability with respect to some *background theory* that fixes the interpretations of certain predicate and function symbols.

We call *Satisfiability Modulo Theories (SMT)* the problem of deciding the satisfiability of formulas with respect to some background theory $\mathcal{T}$.

Many efficient decision procedures have been developed for many decidable fragments of first-order logic since the pioneering work of Nelson and Oppen [Nelson and Oppen, 1979; Nelson and Oppen, 1980; Oppen, 1980a; Oppen, 1980b] and Shostak [Shostak, 1979; Shostak, 1984]. However most of them can only check the satisfiability of conjunctions of atomic expressions, while many applications require to check the satisfiability of not only atomic expressions in a given theory but also of Boolean combinations of them. This makes it necessary to efficiently combine heavy propositional reasoning with theory-specific reasoning. Therefore, SMT research focuses on how to build decision procedures (or SMT solvers) efficiently combining propositional and theory-specific reasoning.

Currently, two major approaches for implementing SMT solvers exists: the *eager* and the *lazy* approaches.

The *eager* approach [Velev and Bryant, 1999; Bryant *et al.*, 2002; Strichman, 2002; Strichman *et al.*, 2002; Seshia *et al.*, 2003] encodes the problem (an SMT formula) into an equisatisfiable propositional formula using enough relevant consequences of the theory of interest, and feeds the resulting formula to a SAT solver. In principle this approach can be used with any theory with a decidable ground satisfiability problem. As advantages of this approach, we can mention that it can be used with any off-the-shelf SAT solver, and that the formula could be solved quickly since the translation imposes upfront all theory-specific constraints on the search space of the SAT solver. A drawback of this approach is that the cost of the translation can blow-up depending on the theory, and that its viability depends on the SAT solvers ability to quickly process relevant theory-specific information encoded into large SAT formulas.

The *lazy* approach combines theory-specific decision procedures (known as $\mathcal{T}$-solvers) with an efficient SAT solver so that the joint system (i. e., SAT solver + $\mathcal{T}$-solvers) is capable of handling (quantifier-free) first-order formulas with an arbitrary Boolean structure. By using theory-specific solvers, one can use whatever specialized algorithms and data structures are best for the theory in question, leading to a better performance of the system. Currently, the lazy approach underlies most of the state-of-the-art SMT tools.

Recently there has been a great deal of interest on the foundational and practical aspects of SMT due to its applications in many domains (e. g., planning [Wolfman and Weld, 1999], temporal reasoning [Armando *et al.*, 2000], formal verification, including verification of pipelines and of circuits [Burch and Dill, 1994; Parthasarathy *et al.*, 2004; Bozzano *et al.*, 2006a], of proof obligations in software systems [Déharbe and Ranise, 2003; Franzén, 2006], of compiler optimizations [Barrett *et al.*, 2005], of real-time systems [Audemard *et al.*, 2002b; De Moura *et al.*, 2002a; Audemard *et al.*, 2005], etc). Also many SMT solvers have been developed in academia and industry (e. g., Z3 [De Moura and Bjørner, 2008], CVC4 [Barrett *et al.*, 2011], veriT [Bouton *et al.*, 2009], MathSAT [Cimatti *et al.*, 2013], etc.).

4.2 BACKGROUND

In this section we introduce the needed definitions for the rest of the chapter.

4.2.1 *First-order Logic*

In what follows we work in the context of First-Order Logic ($\mathcal{FOL}$) with equality [Ebbinghaus *et al.*, 1994; Enderton, 2001].

Definition 4.1 (Syntax). *Let $\Sigma = \langle \mathcal{V}, \mathcal{F}, \mathcal{P} \rangle$ be a first-order signature such that $\mathcal{V}$ is a countable non-empty set of variables, $\mathcal{F}$ is a countably infinite set of function symbols and*

$\mathcal{P}$ is a countably infinite set of predicate symbols. Each function symbol f and predicate symbol p has associated with it a non-negative number called arity*. We call the 0-arity function symbols* constant *symbols and usually denote them by the letters a, b, possibly with subscripts. Similarly, we call the 0-arity predicate symbols* propositional *symbols, and usually denote them by the letters A, B, possibly with subscripts. The set of predicate symbols is assumed to contain a binary predicate $=$ with arity 2.*
The set of Σ-terms is the smallest set defined by:

$$t ::= x \mid f(t_1, \dots, t_n)$$

where $x \in \mathcal{V}$, $f \in \mathcal{F}$ with arity n and $t_1, \dots, t_n$ are Σ-terms.
The set of Σ-formulas is defined by:

$$\varphi ::= p(t_1, \dots, t_n) \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \forall x.\varphi_1 \mid \exists x.\varphi_1$$

where $x \in \mathcal{V}$, $p \in \mathcal{P}$ of arity n, $t_1, \dots, t_n$ are Σ-terms, and φ_1, φ_2 are Σ-formulas. We will also use classical connectives such as $\wedge$, $\rightarrow$ and $\leftrightarrow$ taken to be defined in the usual way. We say a variable $x \in \mathcal{V}$ is free *in a Σ-formula φ if it is not in the scope of a quantifier ($\forall$ or $\exists$), and* bound *otherwise. A* sentence *is a Σ-formula without free variables. A* quantifier-free Σ-formula *is a Σ-formula not containing $\exists$ or $\forall$. A* ground Σ-formula *is a Σ-formula not containing variables (neither free nor bound).*

A Σ-atom is a formula of the form $p(t_1, \dots, t_n)$ where p is a predicate symbol. If p is a 0-arity predicate symbol we sometimes call it a *Boolean atom*. A Σ-literal is either a Σ-atom (a *positive literal*) or its negation (a *negative literal*). A Σ-clause is a disjunction $l_1 \vee \dots \vee l_n$ of Σ-literals. A CNF formula is a conjunction $c_1 \wedge \dots \wedge c_n$ of zero or more clauses c_i.

Formulas are given a meaning, i. e., a truth value from the set $\{true, false\}$, by means of *(first-order) structures*.

Definition 4.2 (Structures). *A structure (or Σ-structure) $\mathcal{I}$ for a signature $\Sigma = \langle \mathcal{V}, \mathcal{F}, \mathcal{P} \rangle$ is a pair $\mathcal{I} = \langle D, (\_)^{\mathcal{I}} \rangle$ where D is a non-empty set called the* domain *and $(\_)^{\mathcal{I}}$ is a mapping such that, for each variable $x \in V$, $x^{\mathcal{I}} \in D$, for each function symbol f with arity n , $f^{\mathcal{I}}$ is a total function from D^n to D and for each predicate symbol p with arity n, $p^{\mathcal{I}}$ is a subset of D^n. By extension, the interpretation of a term t is given by:*

$$\begin{aligned} \mathcal{I}[x] &= x^{\mathcal{I}} \\ \mathcal{I}[f(t_1, \dots, t_n)] &= f^{\mathcal{I}}(\mathcal{I}[t_1], \dots, \mathcal{I}[t_n]) \end{aligned}$$

We use $\mathcal{I}\{x \mapsto v\}$ to denote a structure identical to $\mathcal{I}$ except in that the variable x is interpreted as $v \in D$.

We now define the notion of a Σ-formula φ being satisfied in a Σ-structure $\mathcal{I}$.

Definition 4.3 (Semantics). *Let φ be a Σ-formula and $\mathcal{I}$ a Σ-structure. The satisfaction relation $\mathcal{I} \models \varphi$ is defined as*

$$\begin{array}{lll} \mathcal{I} \models p(t_1, \dots, t_n) & \textit{iff} & (\mathcal{I}[t_1], \dots, \mathcal{I}[t_n]) \in p^{\mathcal{I}}, \\ \mathcal{I} \models \neg\varphi & \textit{iff} & \mathcal{I} \not\models \varphi, \\ \mathcal{I} \models \varphi \vee \psi & \textit{iff} & \mathcal{I} \models \varphi \text{ or } \mathcal{I} \models \psi, \\ \mathcal{I} \models \exists x.\varphi & \textit{iff} & \mathcal{I}\{x \mapsto v\} \models \varphi \textit{ for some } v \in D, \\ \mathcal{I} \models \forall x.\varphi & \textit{iff} & \mathcal{I}\{x \mapsto v\} \models \varphi \textit{ for all } v \in D. \end{array}$$

A Σ-formula φ is *satisfiable* if there exists a Σ-structure $\mathcal{I}$ such that $\mathcal{I} \models \varphi$, and it is *valid* if for all Σ-structures $\mathcal{I}$, $\mathcal{I} \models \varphi$. If $\mathcal{I}$ satisfies φ we call it a *model of* φ. A Σ-structure $\mathcal{I}$ satisfies a set of Σ-formulas S ($\mathcal{I} \models S$) if $\mathcal{I} \models \varphi$ for every $\varphi \in S$.

4.2.2 *First-order Theories*

In SMT, we are interested in models belonging to a given *theory* $\mathcal{T}$constraining the interpretation of the symbols of Σ.

Definition 4.4 (Σ-theory). *A* Σ-theory *is a collection of sentences over a signature* Σ. *A Σ-structure $\mathcal{I}$ is a model of a Σ-theory $\mathcal{T}$ if $\mathcal{I}$ satisfies every sentence in $\mathcal{T}$. Given a Σ-theory $\mathcal{T}$ and a Σ-formula φ, we say φ is* satisfiable modulo $\mathcal{T}$ *(or* $\mathcal{T}$-satisfiable*) if $\mathcal{T} \cup \{\varphi\}$ is satisfiable. We use $\mathcal{I} \models_{\mathcal{T}} \varphi$ to denote $\mathcal{I} \models \{\varphi\} \cup \mathcal{T}$.*

In the context of SMT, sometimes a Σ-theory $\mathcal{T}$ is defined as a class of structures, those satisfying $\mathcal{T}$, and we say a Σ-formula φ is satisfiable modulo $\mathcal{T}$ if and only if there exists a Σ-structure $\mathcal{I}$ in $\mathcal{T}$ such that $\mathcal{I} \models \varphi$.

Remark. From now on we restrict our attention to *quantifier-free* Σ-formulas. Henceforth, for simplicity and if not specified otherwise, we may omit the "Σ" prefix for term, formula, theory, structure, etc. Moreover, by "formulas", "atoms" and "literals" we implicitly refer to quantifier-free formulas, atoms and literals respectively.

Given a set Γ of formulas and a formula φ, we write $\Gamma \models_{\mathcal{T}} \varphi$ to denote $\Gamma \cup \mathcal{T} \models \varphi$. Two formulas φ and ψ are *$\mathcal{T}$-equisatisfiable* if and only if φ is $\mathcal{T}$-satisfiable if and only if ψ is $\mathcal{T}$-satisfiable.

Remark. From now on we will often use the prefix "$\mathcal{T}$-" to denote "in the theory $\mathcal{T}$": e. g., we call a "$\mathcal{T}$-formula" a formula in the signature of $\mathcal{T}$, "$\mathcal{T}$-model" a model in $\mathcal{T}$, and so on.

All theories we consider are first-order theories *with equality*, which means that the equality symbol $=$ is a predefined predicate and it is always interpreted as the identity in the underlying domain. Consequently, $=$ is interpreted as a relation that is reflexive, symmetric, transitive and it is also a congruence. Since the equality symbol is a predefined predicate, we will not include it explicitly in any signature Σ considered from now on.

Two key concepts concerning first-order theories are those of a *convex* and *stably-infinite* theory.

Definition 4.5 (Convex and Stably-infinite Theories). *A conjunction Γ of $\mathcal{T}$-literals in a theory $\mathcal{T}$ is* convex *if and only if for each disjunction $\bigvee_{i=1}^{n} x_i = y_i$ where x_i, y_i are variables and $i = 1, \ldots, n$ we have that $\Gamma \models_{\mathcal{T}} \bigvee_{i=1}^{n} x_i = y_i$ if and only if $\Gamma \models_{\mathcal{T}} x_i = y_i$ for some $i \in \{1, \ldots, n\}$; a theory $\mathcal{T}$ is* convex *if and only if all the conjunctions of literals are convex in $\mathcal{T}$. A theory $\mathcal{T}$ is* stably-infinite *if and only if for each $\mathcal{T}$-satisfiable formula φ, there exists a model of $\mathcal{T}$ whose domain is infinite and that satisfies φ.*

Notice that any convex theory whose models are non-trivial, i. e., the domains of the models have all cardinalities strictly greater than one, is stably-infinite.

These concepts are key when studying the computational properties of a theory and when combining two or more theories.

Example 4.1 ([Sebastiani, 2007]). *The theory of Linear Arithmetic on the integers ($\mathcal{LA}(\mathbb{Z})$) is non-convex. To see this, consider the set $\Gamma = \{x - z \geq 0, x - z \leq 1, x_0 - z = 0, x_1 - z = 1\}$. Thus, $\Gamma \models_{\mathcal{LA}(\mathbb{Z})} ((x = x_0) \vee (x = x_1))$, but $\Gamma \not\models_{\mathcal{LA}(\mathbb{Z})} (x = x_0)$ and $\Gamma \not\models_{\mathcal{LA}(\mathbb{Z})} (x = x_1)$.*

Other two important notions in SMT are those of *propositional abstraction* and *refinement*.

Definition 4.6 (Propositional Abstraction and Refinement). *A* propositional abstraction *is a a bijective function $\mathcal{T}2\mathcal{B}$ ("Theory-to-Boolean") such that, $\mathcal{T}2\mathcal{B}$ maps propositional symbols (Boolean atoms) into themselves and non-Boolean $\mathcal{T}$-atoms into fresh propositional symbols, so that two atom instances in φ are mapped into the same propositional symbol if and only if they are syntactically identical, and is homomorphic with respect to the logical operators. Its inverse $\mathcal{B}2\mathcal{T}$ ("Boolean-to-Theory"), is called a* refinement.

Example 4.2 ([Sebastiani, 2007]). *Consider the formula φ below. Its propositional abstraction is given by $\mathcal{T}2\mathcal{B}(\varphi)$.*

$$\begin{array}{rcl@{\qquad\qquad}rcl}
\varphi & := & \{\neg(2x_2 - x_3 > 2) \vee A_1\} & \mathcal{T}2\mathcal{B}(\varphi) & := & \{\neg B_1 \vee A_1\} \\
& \wedge & \{\neg A_2 \vee (x_1 - x_5 \leq 1)\} & & \wedge & \{\neg A_2 \vee B_2\} \\
& \wedge & \{(3x_1 - 2x_2 \leq 3) \vee A_2\} & & \wedge & \{B_3 \vee A_2\} \\
& \wedge & \{\neg(2x_3 + x_4 \geq 5) \vee \neg(3x_1 - x3 \leq 6) \vee \neg A_1\} & & \wedge & \{\neg B_4 \vee \neg B_5 \vee \neg A_1\} \\
& \wedge & \{A_1 \vee (3x_1 - 2x_2 \leq 3)\} & & \wedge & \{A_1 \vee B_3\} \\
& \wedge & \{(x_2 - x_4 \leq 6) \vee (x_5 = 5 - 3x_4) \vee \neg A_1\} & & \wedge & \{B_6 \vee B_7 \vee \neg A_1\} \\
& \wedge & \{A_1 \vee (x_3 = 3x_5 + 4) \vee A_2\} & & \wedge & \{A_1 \vee B_8 \vee A_2\}
\end{array}$$

Definition 4.7 (Truth Assignment). *We call a* truth assignment *μ for a $\mathcal{T}$-formula φ a truth value assignment to the $\mathcal{T}$-atoms of φ. A truth assignment is* total *if it assigns a value to all atoms in φ,* partial *otherwise. Syntactically identical instances of the same $\mathcal{T}$-atom are always assigned identical truth values; syntactically different $\mathcal{T}$-atoms, e. g., $(t_1 \geq t_2)$ and $(t_2 \leq t_1)$, are treated differently and may thus be assigned different truth values. Given two truth assignments μ_1 and μ_2, if $\mu_2 \subseteq \mu_1$, then we say μ_1* extends *μ_2 and that μ_2* subsumes *μ_1. We use the Greek letters μ, η to denote truth assignments. We represent a truth assignment μ for φ as a set of $\mathcal{T}$-literals (e. g., $\{\alpha_1, \ldots, \alpha_n\}$) and sometimes we write them as the conjunction of its literals (e. g., $\alpha_1 \wedge \ldots \wedge \alpha_n$).*

A total truth assignment μ for φ *propositionally satisfies φ*, written $\mu \models_p \varphi$, if and only if $\mathcal{T}2\mathcal{B}(\mu) \models \mathcal{T}2\mathcal{B}(\varphi)$.A partial truth assignment μ *propositionally satisfies φ* if and only if every total truth assignment extending μ propositionally satisfies φ.

Intuitively, if we consider a $\mathcal{T}$-formula φ as a propositional formula in its atoms, $\models_p$ is the standard satisfiability in propositional logic. Thus, for every φ_1 and φ_2, we say that $\varphi_1 \models_p \varphi_2$ if and only if $\mu \models_p \varphi_2$ for every μ such that $\mu \models_p \varphi_1$. We also say that $\models_p \varphi$ (φ is *propositionally valid*) if and only if $\mu \models_p \varphi$ for every assignment μ for φ. Thus $\varphi_1 \models_p \varphi_2$ if and only if $\models_p \varphi_1 \to \varphi_2$, and $\models_p \varphi$ if and only if $\neg\varphi$ is propositionally unsatisfiable.

Notice that $\models_p$ is stronger than $\models_\mathcal{T}$, i. e., if $\varphi_1 \models_p \varphi_2$, then $\varphi_1 \models_\mathcal{T} \varphi_2$, but not vice-versa, e. g., $(x_1 \leq x_2) \wedge (x_2 \leq x_3) \models_{\mathcal{LA}} (x_1 \leq x_3)$, but $(x_1 \leq x_2) \wedge (x_2 \leq x_3) \not\models_p (x_1 \leq x_3)$.

Example 4.3. *Consider the formula φ of Example 4.2. The partial truth assignment $\mu = \{\neg B_1, \neg A_2, B_3, \neg B_5, B_6, B_8\}$ propositionally satisfies φ (i. e., $\mu \models_p \varphi$).*

4.2.2.1 *Relevant Theories*

We now briefly introduce some theories of interest. As mentioned previously, all the theories we consider are first-order theories with equality, in which "$=$" is a predefined predicate and it is always interpreted as the identity on the underlying domain. We assume that equality axioms and congruence axioms are implicit in all theories, for every function symbol f and predicate symbol p.

EQUALITY AND UNINTERPRETED FUNCTIONS. The theory of *Equality and Uninterpreted Functions ($\mathcal{EUF}$)* is the quantifier-free first-order theory with equality with no restrictions on Σ. Semantically, there are no axioms other than the equality axioms and the congruence axioms.. If Σ contains no uninterpreted functions or predicates, then congruence axioms are not needed, and we denote the resulting restricted theory by $\mathcal{E}$. $\mathcal{EUF}$ is stably-infinite and convex. $\mathcal{EUF}$-satisfiability of sets of quantifier-free literals is decidable in polynomial time using a procedure known as *congruence closure* [Downey *et al.*, 1980; Bachmair *et al.*, 2003; Nieuwenhuis and Oliveras, 2005b].

LINEAR ARITHMETIC. The theory of *Linear Arithmetic ($\mathcal{LA}$)* on the rationals ($\mathcal{LA}(\mathbb{Q})$) and on the integers ($\mathcal{LA}(\mathbb{Z})$) is the quantifier-free first-order theory with equality whose atoms are written in the form $(a_1 \cdot x_1 + \ldots + a_n \cdot x_n \bowtie a_0)$ such that $\bowtie \in \{\leq, <, \neq, =, \geq, >\}$, the a_i's are (interpreted) constant symbols, each labeling a value in $\mathbb{Q}$ and $\mathbb{Z}$ respectively. The atomic expressions are interpreted according to the standard semantics of linear arithmetic on $\mathbb{Q}$ and $\mathbb{Z}$ respectively.

$\mathcal{LA}(\mathbb{Q})$ is stably-infinite and convex. $\mathcal{LA}(\mathbb{Q})$-satisfiability of sets of quantifier-free literals is decidable in polynomial time. The main algorithms use a variant of the Simplex and Fourier-Motzkin algorithms [Borning *et al.*, 1997; Dutertre and De Moura, 2006].

$\mathcal{LA}(\mathbb{Z})$ is stably-infinite and non-convex. $\mathcal{LA}(\mathbb{Z})$-satisfiability of sets of quantifier-free literals is decidable and NP-complete. Many algorithms exist, involving techniques like Euler's reduction, Gomory-cuts application, Fourier-Motzkin algorithms and branch-and-bound [Land and Doig, 1960].

DIFFERENCE LOGIC. The *theory of differences ($\mathcal{DL}$)* (or *difference logic)* on the rationals ($\mathcal{DL}(\mathbb{Q})$) and the integers ($\mathcal{DL}(\mathbb{Z})$) is the sub-theory of $\mathcal{LA}(\mathbb{Q})$ (resp. $\mathcal{LA}(\mathbb{Z})$) whose atoms are written in the form $(x_1 - x_2 \bowtie a)$ such that $\bowtie \in \{\leq, <, \neq, =, \geq, >\}$, and a is an (interpreted) constant symbol labeling one value in $\mathbb{Q}$ and $\mathbb{Z}$, respectively.

$\mathcal{DL}(\mathbb{Q})$ is stably-infinite and convex. $\mathcal{DL}(\mathbb{Q})$-satisfiability of sets of quantifier-free difference inequalities is decidable and polynomial. Thanks to the convexity of $\mathcal{DL}(\mathbb{Q})$, the $\mathcal{DL}(\mathbb{Q})$-satisfiability of sets of quantifier-free difference inequalities, equalities and disequalities is also polynomial. The main algorithms encode $\mathcal{DL}(\mathbb{Q})$-satisfiability of difference inequalities into the problem of finding negative cycles into a weighted oriented graph called *constraint graph* [Cherkassky and Goldberg, 1999].

$\mathcal{DL}(\mathbb{Z})$ is stably-infinite and non-convex. As with $\mathcal{DL}(\mathbb{Q})$, $\mathcal{DL}(\mathbb{Z})$-satisfiability of sets of quantifier-free difference inequalities is decidable and polynomial, as before, adding equalities does not affect the complexity of the problem. Instead, due to the non-convexity of $\mathcal{DL}(\mathbb{Z})$, $\mathcal{DL}(\mathbb{Z})$-satisfiability of sets of quantifier-free difference inequalities, equalities and disequalities, is NP-complete. Once the problem is rewritten as a set of difference inequalities, the algorithms used for $\mathcal{DL}(\mathbb{Z})$ are the same as for $\mathcal{DL}(\mathbb{Q})$ [Cherkassky and Goldberg, 1999; Cotton and Maler, 2006; Nieuwenhuis and Oliveras, 2005a].

BIT VECTORS. The *theory of fixed-width bit vectors ($\mathcal{BV}$)* is a first-order theory with equality that aims at representing Register Transfer Level (RTL) hardware circuits, so that components such as data paths or arithmetical sub-circuits are considered as entities as a whole, rather than being encoded into purely propositional sub-formulas ("bit-blasting"). $\mathcal{BV}$ can also be used to encode software verification problems [Ganesh and Dill, 2007]. In $\mathcal{BV}$ terms indicate fixed-width bit vectors, and are built from variables (e. g., $\mathbf{x}^{[32]}$ indicates a vector x of 32 bits) and constants (e. g., $\mathbf{0}^{[16]}$ denotes a vector of 16 0's) by means of interpreted functions representing standard RTL operators: word concatenation (e. g., $\mathbf{x}^{[16]} \circ \mathbf{y}^{[16]}$), sub-word selection (e. g., $(\mathbf{x}^{[32]}[20:5])^{[16]}$), modulo-n sum and multiplication (e. g., $\mathbf{x}^{[32]} +_{32} \mathbf{y}^{[32]}$ and $\mathbf{x}^{[32]} \cdot_{32} \mathbf{y}^{[32]}$), bitwise-operators $\mathbf{and}_n$, $\mathbf{or}_n$, $\mathbf{xor}_n$, $\mathbf{not}_n$, left and right shift $\ll_n, \gg_n$ (e. g., $\mathbf{x}^{[32]} \ll_4$). Atomic expressions can be built from terms by applying interpreted predicates like $\leq_n, <_n$ (e. g., $\mathbf{x}^{[32]} \leq_{32} \mathbf{y}^{[32]}$) and equality.

$\mathcal{BV}$ is non-convex and non-stably infinite. $\mathcal{BV}$-satisfiability of sets of quantifier-free literals is decidable and NP-complete [Fallah *et al.*, 1998; Johannsen and Drechsler, 2002].

ARRAYS. The *theory of arrays ($\mathcal{AR}$)* aims at modeling the behavior of arrays/memories. The signature consists of two interpreted functions *write* and *read*, such that *write*(a, i, e) represents (the state of) the array resulting from storing an element e into the location of address i of an array a, and *read*(a, i) represents the element contained in the array a at location i. $\mathcal{AR}$ is formally characterized by the following axioms:

$$\forall a.\forall i.\forall e.(read(write(a,i,e),i) = e), \tag{4.1a}$$
$$\forall a.\forall i.\forall j.\forall e.((i \neq j) \to read(write(a,i,e),j) = read(a,j)), \tag{4.1b}$$
$$\forall a.\forall b.(\forall i.(read(a,i) = read(b,i)) \to (a = b)). \tag{4.1c}$$

(4.1a) and (4.1b), called *McCarthy's axioms*, characterize the intended meaning of *write* and *read*, whilst (4.1c), called the *extensionality axiom*, requires that, if two arrays contain the same values in all locations, then they must be the same array. We call *extensional* to a theory of arrays including the axiom (4.1c), and *non-extensional* otherwise. Although many practical problems do not require extensionality, axiom (4.1c) is explicitly required for some software verification problems in order to represent assignments or comparisons between arrays.

$\mathcal{AR}$-satisfiability of sets of literals is decidable and NP-complete [Stump *et al.*, 2001].

LISTS. The *theory of lists ($\mathcal{LI}$)* aims at modeling the behavior of lists. The signature consists in the three interpreted function symbols *cons*, *car*, *cdr* representing the standard LISP constructor and selectors for lists. $\mathcal{LI}$ is formally characterized by the following axioms:

$$\forall x.(cons(car(x), cdr(x)) = x), \tag{4.2a}$$

$$\forall x.\forall y.(car(cons(x,y)) = x), \forall x.\forall y.(cdr(cons(x,y)) = y), \tag{4.2b}$$

$$\forall x.(car(x) \neq x), \forall x.(cdr(x) \neq x), \forall x.(car(car(x)) \neq x), \ldots \tag{4.2c}$$

(4.2a) and (4.2b), called *construction* and *selection* axioms respectively, characterize the intended meaning of *cons*, *car* and *cdr*, whilst (the infinite sequence of) the *acyclicity axioms* (4.2c) force the list to be acyclic.

$\mathcal{LI}$-satisfiability of sets of literals is decidable in linear time [Oppen, 1980b].

4.3 LAZY SMT

So far we have introduced the basic theoretical aspects of SMT and briefly described some of the most interesting theories in practice. We now present the basic technical aspects of *lazy* SMT.

Lazy SMT is the dominating approach to SMT and underlies most state-of-the-art SMT tools. It is based on the integration of a SAT solver and one or more theory solvers ($\mathcal{T}$-solvers). The former handles the Boolean component of reasoning by enumerating truth assignments satisfying the propositional abstraction of the input formula, while the latter handles the theory-specific one, checking the consistency in the theory $\mathcal{T}$ of the set of literals corresponding to the assignments enumerated by the SAT solver (i. e., that, for a set of literals Γ, $\Gamma \not\models_{\mathcal{T}} \bot$).

The following result assures the soundness of this integration scheme.

Definition 4.8. *We say that a collection $\mathcal{M} := \{\mu_1, \ldots, \mu_n\}$ of (possibly partial) assignments propositionally satisfying φ is* complete *if and only if,*

$$\models_p \varphi \leftrightarrow \bigvee_{\mu_j \in \mathcal{M}} \mu_j.$$

$\mathcal{M} := \{\mu_1, \ldots, \mu_n\}$ is complete in the sense that, for every total assignment η such that $\eta \models_p \varphi$, there exists $\mu_j \in \mathcal{M}$ such that $\mu \subseteq \eta$. Thus $\mathcal{M}$ can be seen as a compact representation of the whole set of total assignments propositionally satisfying φ.

Proposition 4.1. *Let φ be a $\mathcal{T}$-formula and let $\mathcal{M} := \{\mu_1, \ldots, \mu_n\}$ be a complete collection of truth assignments propositionally satisfying φ. Then, φ is $\mathcal{T}$-satisfiable if and only if μ_j is $\mathcal{T}$-satisfiable for some $\mu_j \in \mathcal{M}$.*

Proposition 4.1 provides the theoretical basis for integrating SAT solvers and $\mathcal{T}$-solvers. It tells us that the problem of establishing the $\mathcal{T}$-satisfiability of φ can be decomposed into two orthogonal components: one *Boolean component*, consisting in searching for (up to a complete set of) propositional models μ's propositionally satisfying φ, and one *theory-dependent component*, consisting in checking the $\mathcal{T}$-consistence of μ (i. e., for the set of $\mathcal{T}$-literals in μ). This suggests that an SMT solver can be seen as a combination of two basic components: a *Truth Assignment Enumerator* and a *Theory Solver* for $\mathcal{T}$.

We call a *Truth Assignment Enumerator* (from now on *Enumerator*) a total function that takes as input a $\mathcal{T}$-formula φ and returns a complete collection $\mathcal{M} := \{\mu_1, \ldots, \mu_n\}$ of assignments propositionally satisfying φ. We call a *Theory Solver* for $\mathcal{T}$ ($\mathcal{T}$-solver) a procedure that takes as input a collection of $\mathcal{T}$-literals μ and decides whether μ is $\mathcal{T}$-satisfiable; optionally it can return a $\mathcal{T}$-model satisfying μ, or *Null* if there is none.

4.3.1 *SAT Solvers*

A SAT solver is a tool that implements a decision procedure to decide whether an input propositional formula φ is satisfiable, returning a satisfying assignment if that is the case.

Most state-of-the-art SAT solvers implement decision procedures that are evolutions of the Davis-Putnam-Logemann-Loveland (DPLL) procedure [Davis and Putnam, 1960; Davis *et al.*, 1962] which are non-recursive and based on efficient data structures. Figure 4.1 shows a high-level schema of a modern conflict-driven DPLL procedure (from now on a DPLL engine).

```
DPLL ($\varphi$, $\mu$){
  if (preprocess($\varphi$,$\mu$) == CONFLICT)
      return UNSAT;
  while(1){
    decide_next_branch($\varphi$,$\mu$);
    while(1){
      status = deduce($\varphi$,$\mu$);
      if (status == SAT)
         return SAT;
      else if (status == CONFLICT) {
         blevel = analyze_conflict($\varphi$,$\mu$);
         if (blevel == 0)
             return UNSAT;
         else
             bactrack(blevel,$\varphi$,$\mu$);
      }
      else break;}}}
```

Figure 4.1: Schema of a conflict-driven DPLL procedure [Sebastiani, 2007].

The input formula φ is in CNF; the assignment μ is initially empty and is updated in a stack-based manner. The procedure tries to build a satisfying assignment based on five main operations: `preprocess`, `decide_next_branch`, `deduce`, `analyze_conflict`, and `backtrack`.

- `preprocess` simplifies φ into a simpler and equisatisfiable formula, and updates the assignment μ if it is needed.

- `decide_next_branch` chooses an unassigned literal l from φ according to some heuristic criterion, and adds it to μ. This operation is called a *decision*, l a *decision literal*, and the number of decision literals in μ after this operation is called the *decision level of* l.

- `deduce` iteratively deduces literals l deriving from the current assignments, i. e., $\varphi \wedge \mu \models l$, and updates φ and μ accordingly. This step is repeated until either μ satisfies φ, or μ falsifies φ, or no more literals l can be deduced, returning `Sat`, `Conflict` and `Unknown` respectively.

- If a conflict is detected, `analyze_conflict` detects the subset η of μ that caused the conflict, called the *conflict set* and the decision level `blevel` to backtrack.

- `backtrack` adds $\neg\eta$ to φ (*learning*) and backtracks up to `blevel` (*backjumping*), updating φ and μ accordingly.

4.3.2 *Theory Solvers*

A theory solver (from now on a $\mathcal{T}$-solver) is a procedure that establishes whether a given finite set (or conjunction) of quantifier-free $\mathcal{T}$-literals is $\mathcal{T}$-satisfiable or not.

Many algorithms have been developed for many theories since the pioneering work of Nelson, Oppen and Shostak [Nelson and Oppen, 1979; Shostak, 1979; Nelson and Oppen, 1980; Shostak, 1984].

To achieve its maximum efficiency on an SMT solver, two aspects of a $\mathcal{T}$-solver are of key importance: its *efficiency* in time and memory, and the *effectiveness* of its interaction with the DPLL engine. The former strongly depends on the theory $\mathcal{T}$, while the latter depends on the capability of a $\mathcal{T}$-solver of producing, exchanging and exploiting fruitful information with the DPLL engine.

In particular, the following features are crucial for a $\mathcal{T}$-solver to be effectively used within a lazy SMT solver:

- *Model Generation*: It should be able to produce a $\mathcal{T}$-model if it is invoked in a $\mathcal{T}$-consistent set μ.

- *Conflict set generation*: When invoked on a $\mathcal{T}$-inconsistent set μ, it should be able to produce the (possibly minimal) subset η of μ that has caused its inconsistency. The set η is called a *theory conflict set* of μ.

- *Deduction of unassigned literals*: It should be able to perform deductions of the form $\eta \models_{\mathcal{T}} l$ (when invoked on a $\mathcal{T}$-consistent set μ) where $\eta \subseteq \mu$ and l is a literal on a not-yet-assigned atom in φ.

- *Deduction of interface equalities*: If μ is $\mathcal{T}$-consistent, the $\mathcal{T}$-solver should be able to perform deductions of the form $\mu \models_{\mathcal{T}} e$ (if $\mathcal{T}$ is convex) or $\mu \models_{\mathcal{T}} \bigvee_j e_j$ (if $\mathcal{T}$ is not convex) where $e, e_1, \ldots, e_n$ are equalities between variables or terms occurring in atoms in μ. Because typically $e, e_1, \ldots, e_n$ are interface equalities, we call these forms of deductions *e_{ij}-deductions*, and we say that a $\mathcal{T}$-solver is *e_{ij}-deduction-complete* if it can perform all such possible deductions.

- *Incrementality*: It should be able to "remember" its computation status from one call to the next, so that, whenever it is given as input a set $\mu_1 \cup \mu_2$ such that μ_1 has just been proved $\mathcal{T}$-satisfiable, it avoids restarting the computation from scratch.

- *Backtrackability*: It should be able to undo steps and return to a previous state in an efficient manner.

4.3.3 *Integration of a DPLL Engine and $\mathcal{T}$-solvers*

There are many representations and variants of SMT procedures that integrate a DPLL engine and $\mathcal{T}$-solvers. However, they all share some common aspects.

Figure 4.2 shows a basic architectural schema of a typical lazy DPLL-based SMT procedure, (from now on $\mathcal{T}$-DPLL). The $\mathcal{T}$-DPLL procedure takes as input a $\mathcal{T}$-formula φ, and builds its propositional abstraction $\varphi^p =_{def} \mathcal{T}2\mathcal{B}(\varphi)$, then feeding it as input to the *Enumerator*.

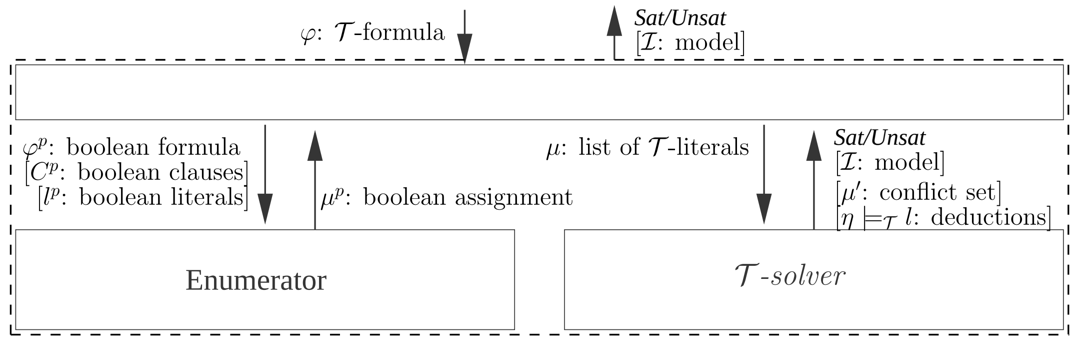

φ: $\mathcal{T}$-formula
Sat/Unsat
[$\mathcal{I}$: model]
φ^p: boolean formula
[C^p: boolean clauses]
[l^p: boolean literals]
μ^p: boolean assignment
μ: list of $\mathcal{T}$-literals
Sat/Unsat
[$\mathcal{I}$: model]
[μ': conflict set]
[$\eta \models_{\mathcal{T}} l$: deductions]
Enumerator
$\mathcal{T}$-solver

Figure 4.2: Basic architectural schema of a lazy SMT procedure [Sebastiani, 2007].

The *Enumerator* enumerates truth assignments in a complete collection $\{\mu_1^p, \ldots, \mu_n^p\}$ for φ^p. For every generated μ^p, $\mathcal{T}$-DPLL feeds its corresponding list of $\mathcal{T}$-literals $\mu =_{def} \mathcal{B}2\mathcal{T}(\mu^p)$ to the $\mathcal{T}$-solver, dropping the Boolean literals as they do not affect the $\mathcal{T}$-satisfiability of μ. If μ is satisfiable, the procedure returns `Sat`, possibly returning a model $\mathcal{I}$. If not, the *Enumerator* generates a new assignment. The process is repeated until either one $\mathcal{T}$-satisfiable assignment is found (i.e., φ is $\mathcal{T}$-satisfiable) or no more assignments are generated by the *Enumerator* (i.e., φ is not $\mathcal{T}$-satisfiable).

If μ is $\mathcal{T}$-unsatisfiable the $\mathcal{T}$-solver can return one or more theory conflict set(s) μ' and Boolean clauses like $C^p =_{def} \mathcal{T}2\mathcal{B}(\neg\mu')$ can be passed back to the *Enumerator*. If μ is $\mathcal{T}$-satisfiable, the $\mathcal{T}$-solver can return one or more deduction(s) $\eta \models_{\mathcal{T}} l$ and return the Boolean clause $C^p =_{def} \mathcal{T}2\mathcal{B}(\neg\eta \vee l)$ and deduced Boolean literals $l^p =_{def} \mathcal{T}2\mathcal{B}(l)$ to the *Enumerator*. In both cases, the returned Boolean clauses help the *Enumerator* to drive its Boolean search.

Depending on how we integrate the DPLL engine, we can classify $\mathcal{T}$-DPLL procedures into two main categories: *offline procedures* and *online procedures*.

OFFLINE PROCEDURES In the offline procedures [Barrett *et al.*, 2002; De Moura *et al.*, 2002b], the DPLL engine is used as a black-box that is invoked from scratch each time an assignment is found $\mathcal{T}$-unsatisfiable, therefore acting as a non-redundant *Enumerator*. Figure 4.3 shows a simplified schema for offline procedures.

For an input formula φ, its propositional abstraction φ^p is computed and given to the DPLL engine. If the DPLL engine decides that φ^p is unsatisfiable, then φ is $\mathcal{T}$-unsatisfiable and the work is done. If the DPLL engine returns a satisfying

```
$\mathcal{T}$–DPLL($\varphi$) {
  $\varphi^p = \mathcal{T}2\mathcal{B}(\varphi)$;
  while (DPLL($\varphi^p$,$\mu^p$) == SAT) {
    if ($\mathcal{T}$−solver($\mathcal{B}2\mathcal{T}(\mu^p)$) == SAT)
      return SAT;
    $\varphi^p = \varphi^p \wedge \neg\mu^p$;
  }
  return UNSAT;}
```

Figure 4.3: Offline integration schema [Sebastiani, 2007].

assignment μ^p then $\mathcal{B}2\mathcal{T}(\mu^p)$ is given as input to the $\mathcal{T}$-solver. If $\mathcal{B}2\mathcal{T}(\mu^p)$ is $\mathcal{T}$-consistent ($\mathcal{T}$-satisfiable), then φ is $\mathcal{T}$-consistent. If not, $\neg\mu^p$ is added as a clause to φ^p, and the SAT solver is restarted from scratch on the resulting formula. A more efficient integration could be achieved if the $\mathcal{T}$-solver is able to return the conflict set η that caused the $\mathcal{T}$-inconsistency of $\mathcal{B}2\mathcal{T}(\mu^p)$, in which case $\mathcal{B}2\mathcal{T}(\neg\eta)$ is added as a clause to φ instead of $\neg\mu^p$. As typically the former is much smaller than the latter, this drastically reduces the search space.

ONLINE PROCEDURES In the online procedures [Giunchiglia and Sebastiani, 1996a; Wolfman and Weld, 1999; Armando *et al.*, 2000; Audemard *et al.*, 2002a; Flanagan *et al.*, 2003; Ganzinger *et al.*, 2004; Bozzano *et al.*, 2006b], the DPLL engine is modified to be used directly as an enumerator of truth assignments, whose $\mathcal{T}$-satisfiability is checked by a $\mathcal{T}$-solver.

Figure 4.4 shows a simplified schema for an online $\mathcal{T}$-DPLL. Similar to regular DPLL engine, a $\mathcal{T}$-DPLL procedure is based on five main operations: $\mathcal{T}$-`preprocess`, $\mathcal{T}$-`decide_next_branch`, $\mathcal{T}$-`deduce`, $\mathcal{T}$-`analyze_conflict` and $\mathcal{T}$-`backtrack`.

- $\mathcal{T}$-`preprocess` combines Boolean preprocessing steps with theory rewriting steps on the $\mathcal{T}$-literals of φ to simplify it into a simpler equisatisfiable formula, and updates μ if it is the case.

- $\mathcal{T}$-`decide_next_branch` plays the same role as `decide_next_branch` in DPLL, but it may take into consideration also the semantics in $\mathcal{T}$ of the literals to select.

- $\mathcal{T}$-`deduce`, similar to the `deduce` operation in DPLL, implements an extended notion of deduction of literals, performing not only Boolean deduction ($\mu^p \wedge \varphi^p \models_p l^p$) but also theory deduction ($\mathcal{B}2\mathcal{T}(\mu^p) \models_\mathcal{T} \mathcal{B}2\mathcal{T}(l^p)$), also known as *$\mathcal{T}$-propagation*.

- $\mathcal{T}$-`analyze_conflict` similar to the `analyze_conflict` operation of DPLL, implements an extended notion of conflict: not only Boolean conflict ($\mu^p \wedge \varphi^p \models_p l^p$), but also theory conflict ($\mathcal{B}2\mathcal{T}(\mu^p) \models_\mathcal{T} \bot$), or even mixed Boolean+theory conflict ($\mathcal{B}2\mathcal{T}(\mu^p \wedge \varphi^p) \models_\mathcal{T} \bot$).

- $\mathcal{T}$-`backtrack` behaves analogously to `backtrack` in DPLL. It adds the clause $\neg\eta^p$ to φ^p and backtracks up to level `blevel`. This features are called *$\mathcal{T}$-learning* and *$\mathcal{T}$-backjumping* respectively.

```
$\mathcal{T}$–DPLL ($\varphi$,$\mu$) {
  if ($\mathcal{T}$–preprocess($\varphi$,$\mu$) == CONFLICT)
     return UNSAT;
  $\varphi^p = \mathcal{T}2\mathcal{B}(\varphi)$;
  $\mu^p = \mathcal{T}2\mathcal{B}(\mu)$;
  while(1){
    $\mathcal{T}$–decide_next_branch($\varphi^p$,$\mu^p$);
    while(1){
      status = $\mathcal{T}$–deduce($\varphi^p$,$\mu^p$);
      if (status == SAT){
         $\mu = \mathcal{B}2\mathcal{T}(\mu^p)$;
         return SAT;
      }
      else if (status == CONFLICT) {
        blevel = $\mathcal{T}$–analyze_conflict($\varphi^p$,$\mu^p$);
        if (blevel == 0)
           return UNSAT;
        else
           $\mathcal{T}$–backtrack(blevel,$\varphi^p$,$\mu^p$);
      }
      else
        break;} } }
```

Figure 4.4: Online integration schema [Sebastiani, 2007].

4.3.4 *Combination of Theories*

We close our review of SMT by mentioning one of its most interesting aspects: the combination of theories.

For many practical applications of SMT, the theory $\mathcal{T}$ is a combination of two or more theories $\mathcal{T}_1, \ldots, \mathcal{T}_n$. For instance, an atom of the form $f(x+4y) = g(2x-y)$, that combines uninterpreted functions symbols from $\mathcal{EUF}$ with arithmetic functions from $\mathcal{LA}(\mathbb{Z})$, could be used to model the abstraction of some functional blocks in an arithmetic circuit [Sebastiani, 2007].

The work on combining decision procedures for distinct theories was pioneered by Nelson and Oppen [Nelson and Oppen, 1979; Oppen, 1980a] and Shostak [Shostak, 1984]. In particular, Nelson and Oppen established the theoretical foundations onto which most current combined procedures are still based on, the *Nelson-Oppen formal framework*.

Given two disjoint signatures Σ_1 and Σ_2 and theories $\mathcal{T}_i$ in Σ_i for $i = 1, 2$, the Nelson-Oppen formal framework reduces the $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiability problem of a set of pure literals $\mu = \mu_{\mathcal{T}_1} \wedge \mu_{\mathcal{T}_2}$ (i. e., literals containing only subterms made of symbols from one signature) to that of finding (the arrangement of) an equivalence relation on the shared variables consistent with both pure parts of μ.

Nelson and Oppen also proposed a general-purpose procedure for integrating $\mathcal{T}_i$-solvers into one combined $\mathcal{T}$-solver, the *Nelson-Oppen procedure*. The combined decision procedure works by performing a structured interchange of interface equalities (disjunctions of interface equalities if $\mathcal{T}_i$ is non-convex) which are inferred by either $\mathcal{T}_i$-solver and then propagated to the other, until convergence is reached.

4.4 SYMMETRIES IN SMT

We now turn our attention to symmetries in SMT. As far as we know, research on symmetries in SMT is recent and symmetries are not fully exploited in SMT solvers yet.

The first attempt to exploit symmetries in SMT can be traced back to [Audemard *et al.*, 2002b], where symmetries are used as a simplification technique for SMT-based model checking.

In [Roe, 2006] an SMT solver introduces symmetry breaking predicates to the input formula in a preprocessing stage. To do so, first all symmetric pairs of variables are detected. Given a formula φ, a pair of variables (a, b), occurring in φ, is said to be a symmetric pair if when replacing all instances of a with b and b with a in φ, we obtain again φ. Once a set of symmetric pairs is computed, symmetric pairs of atoms are identified. Two atoms p_1 and p_2 are said to be symmetric if there is a set of symmetric variable pairs $\{(a_1, b_1), \ldots, (a_n, b_n)\}$ such that p_1 can be transformed in p_2 by simply replacing each a_i with b_i and viceversa. Then a group of symmetric predicates is computed. A group of symmetric predicates is a set of two or more atoms such that any two atoms in the group are symmetric. Finally, symmetry breaking predicates, similar to those introduced in [Aloul *et al.*, 2003b], are added to the original problem. According to what is reported in this work, this approach only works for QF_UF formulas (i. e., quantifier-free formulas built over a signature of uninterpreted sort and function symbols, see Chapter 5.4 for more details), but no experimental results are provide to assess its effectiveness.

More recently, in [Déharbe *et al.*, 2011], an algorithm for detecting and breaking symmetries is presented and implemented in the state-of-the-art SMT solver veriT [Bouton *et al.*, 2009]. The algorithm works by detecting full symmetry groups of uninterpreted constants, and then adding symmetry breaking assumptions.

The work is built upon the observation that in SMT solving a frequent source of symmetries is when some term takes its value from a given finite set of totally symmetric elements. Thus, given a formula φ symmetric (or invariant) with respect to all permutations of some uninterpreted constants $c_0, \ldots, c_n$, for any model $\mathcal{M}$ of φ, if term t does not contain these constants and $\mathcal{M}$ satisfies $t = c_i$ for some $i \in \{1, \ldots, n\}$, then there should be a model in which t equals c_0. While checking for unsatisfiability, it is thus sufficient to look for models assigning t and c_0 to the same value. This is stated more formally by the following result.

Theorem 4.1 ([Déharbe *et al.*, 2011]). *Consider a theory $\mathcal{T}$, uninterpreted constants $c_0, \ldots, c_n$, a formula φ invariant w.r.t permutations of $c_i, \ldots, c_n$, and a term t that is invariant w.r.t permutations of $c_i, \ldots, c_n$. If $\varphi \models_{\mathcal{T}} t = c_0 \vee \ldots \vee t = c_n$, then φ is $\mathcal{T}$-satisfiable if and only if*

$$\varphi' =_{def} \varphi \wedge (t = c_0 \vee \ldots \vee t = c_i)$$

is also $\mathcal{T}$-satisfiable. Clearly, φ' is invariant w.r.t permutations of $c_{i+1}, \ldots, c_n$.

Detection of symmetries is done by first "guessing" a permutation and then checking that it is in fact a symmetry of the formula (see Chapter 6 for more details). Experimental results confirm that this approach is successful, in particular for QF_UF problems. However, this approach is limited in the sense that it only works with full symmetry groups, and in that it does not consider symmetries involving other types of symbols, like predicate symbols, function symbols, interpreted symbols, or quantifiers.

Example 4.4 ([Déharbe *et al.*, 2011]). *A classical problem with symmetries is the pigeonhole problem. Most SMT or SAT solvers require exponential time to solve this problem. We show that the symmetry breaking technique introduced previously greatly improves the performance of a standard SMT solver on this class of problems.*

The problem states that it is impossible to place $n+1$ pigeons in n holes. To model the problem, we introduce n uninterpreted constants $h_1, \dots, h_n$ for the n holes, and $n+1$ uninterpreted constants $p_1, \dots, p_{n+1}$ for the $n+1$ pigeons. Each pigeon is required to occupy one hole: $p_i = h_1 \vee \dots \vee p_i = h_n$. It is also required that distinct pigeons occupy different holes: $p_i \neq p_j$ for $1 \leq i < j \leq n+1$. The generated set of formulas is symmetric by permutations of the constants $p_1, \dots, p_{n+1}$.

From our formulation of the problem, it is direct to notice that $p_i = h_1 \vee \dots \vee p_i = h_n$. Therefore, the set of symmetry breaking clauses could be:

$$\begin{aligned}
&p_1 = h_1\\
&p_2 = h_1 \vee p_2 = h_2\\
&p_3 = h_1 \vee p_3 = h_2 \vee p_3 = h_3\\
&\vdots\\
&p_{n-1} = h_1 \vee \dots \vee p_{n-1} = h_{n-1}
\end{aligned}$$

Without need for any advanced theory propagation techniques, $(n+1) \times n/2$ conflict clauses of the form $p_i = h_i \vee p_j = h_i \vee p_j = p_i$ with $i < j$ suffice to transform the problem into a purely propositional problem. With the symmetry breaking clauses, the underlying SAT solver then concludes (in polynomial time) the unsatisfiability of the problem using only Boolean Constraint Propagation. Figure 4.5 shows the results for different SMT solvers. All solvers (including veriT without symmetry heuristics) timeout on problems of relatively small size, although CVC3 performs significantly better than the other solvers. Using the symmetry heuristics allows veriT to solve much larger problems in insignificant times. In fact, the modified version of veriT solves every instance of the problem with as many as 30 pigeons in less than 0.15 seconds.

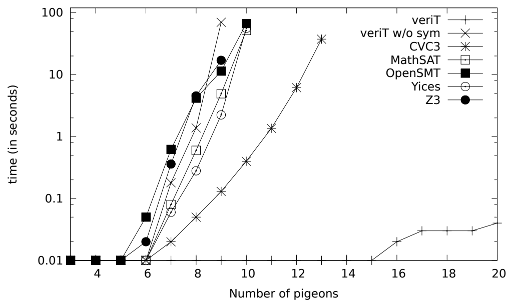


Figure 4.5: Some SMT solvers and the pigeonhole problem [Déharbe *et al.*, 2011].

5 EMPIRICAL TESTING OF DECISION PROCEDURES

This thesis is about using symmetries to improve the efficiency of decision procedures for modal logics and satisfiability modulo theories. In particular, it is about improving the efficiency of algorithms for the satisfiability problem of these logics taking advantage of symmetries. To be able to measure and analyze the effects of using symmetries, we need a set of representative test sets and a methodology to perform testing and to analyze the obtained results. This is where *empirical testing* comes to scene. In this chapter we are going to present the rationale behind empirical testing of decision procedures (Sections 5.1 and 5.2), then we present a survey of empirical testing for modal logics (Section 5.3) and satisfiability modulo theories (Section 5.4).For a comprehensive treatment and for references to the literature on the subject, we refer the reader to [Horrocks *et al.*, 2000] on which this chapter is based.

5.1 WHY EMPIRICAL TESTING?

One of the main tasks when developing decision procedures is to evaluate how well they behave in comparison with other decision procedures. From a theoretical point of view, we can do this by establishing its computational complexity and asymptotic algorithmic complexity. However, when we turn our attention to the implementation of the decision procedures, we not only want to evaluate the algorithm alone, but also the effect of the optimizations and heuristics we introduce. In this case relaying solely on theoretical measures is useless because, in many cases, optimizations and heuristics maintain the worst case complexity of the problem, only improving performance in some particular instances, therefore we have to determine it by *empirical testing*, i. e., by testing the systems with a set of input problems.

5.2 THE QUALITY OF EMPIRICAL TESTING

To avoid results that could lead to wrong conclusions, empirical testing must be carefully planned both on the selection of the input problems (test sets), and on the design of the methodology for analyzing and presenting the obtained results.

Therefore, selecting good test sets and establishing a correct methodology for analyzing and presenting the obtained results is crucial when determining the quality of empirical testing. other

In [Horrocks *et al.*, 2000] a criteria is provided that helps to design *good* tests sets. Although this criteria is provided in the context of empirical testing for modal logics, it is general enough to be applicable in the general case. According to it, a good test set should meet the following requirements:

i) *Reproducibility*: The test set should be reproducible, i. e., the test formulas or their generation function should be available.

ii) *Representativeness*: The test set should represent a significant portion of the potential input space.

iii) *Satisfiable vs. Unsatisfiable balance*: The test set should have a balanced number of satisfiable and unsatisfiable problems.

iv) *Difficulty*: The test set should provide a sufficient level of difficulty for the system(s) being tested.

v) *Parameterization and Control*: The test set should be parameterized with sufficient variables to allow the test set to range over a large portion of the input space.

vi) *Termination*: The test set should terminate and provide information within a reasonable amount of time.

vii) *Data organization*: The data should be summarizable, to make comparison possible with limited effort, and plottable to highlight the qualitative behaviour of the system(s).

Also a good test set should avoid the following problems:

i) *Redundancy*: A test set must avoid redundancy.

ii) *Triviality*: A test set should not contain significant subsets of trivial problems.

iii) *Artificiality*: A test set should correspond closely to inputs from applications.

iv) *Over-size*: Single problems should not be too big with respect to their difficulty.

In general, these criteria aims at "providing a reproducible sample of an interesting portion of the input space with appropiate difficulty" [Horrocks *et al.*, 2000].

However, notice that it can be difficult to comply with all these requirements, therefore, a good test set should aim to comply with as many of the criteria as possible.

5.3 EMPIRICAL TESTING IN MODAL LOGICS

Much work has been done to develop a sound methodology for testing modal decision procedures. However the situation is still not completely satisfactory. The existence of a large number of modal logics makes it difficult to develop a test set and a test methodology that fits all needs.

In this section we present a survey, based on [Horrocks *et al.*, 2000], of test sets and its associated test methodologies for modal decision procedures. This survey is not exhaustive but it takes into account the most common approaches, namely, the Logics Workbench test set, the $3CNF_{\Box_m}$ random test set, the $New\_3CNF_{\Box_m}$ random test set , the modalized test set, and the random QBF test set.

5.3.1 *The Logics Workbench Test Set*

The Logics Workbench (LWB) test set was introduced in [Heuerding and Schwendimann, 1996; Balsiger *et al.*, 2000] as an attempt to overcome the lack of suitable tests sets and methodology for testing modal decision procedures at that time.

The LWB test set is divided into classes of formulas. There are 9 classes of formulas in both valid and invalid versions for the modal logics K (basic (mono)modal logic), KT (basic modal logic with reflexive relations) and S4 (basic modal logic with reflexive and transitive relations).

Each class is generated from a (relatively) simple parameterized logical formula that is either valid or invalid. Some of these formulas are made harder by hiding their structure or adding extra pieces. Formulas in each class are controlled by a single parameter that allows to generate formulas of different size and, therefore, of differing difficulty. Ideally, the difficulty of the formulas should be exponential on the parameter.

The test methodology is to test formulas from each class, starting with the easiest instance, until the validity status of a formula cannot be correctly determined within a timeout of 100 seconds. For each class,the reported result is the parameter of the largest formula solved within the time limit. The parameter ranges from 1 to 21. If a system can solve all 21 instances of a class within the time limit, the result is given as ">".

Tables 5.1a, 5.1b and 5.1c, show example results for the modal prover DLP 3.1 [Patel-Schneider, 1998] on the modal logics K, KT and S4 respectively. For each problem class n_p and n_n are the number of largest formula solved within a timeout of 100 seconds for the satisfiable and unsatisfiable versions respectively.

Class	n_p	n_n
`k_branch`	19	13
`k_d4`	>	>
`k_dum`	>	>
`k_grz`	>	>
`k_lin`	>	>
`k_path`	>	>
`k_ph`	7	9
`k_poly`	>	>
`k_t4p`	>	>

(a) K.

Class	n_p	n_n
`kt_45`	>	>
`kt_branch`	19	12
`kt_dum`	>	>
`kt_grz`	>	>
`kt_md`	3	>
`kt_path`	16	14
`kt_ph`	7	>
`kt_poly`	>	12
`kt_t4p`	>	>

(b) KT.

Class	n_p	n_n
`s4_45`	>	>
`s4_branch`	18	12
`s4_dum`	>	>
`s4_grz`	10	>
`s4_md`	3	>
`s4_path`	15	15
`s4_ph`	7	>
`s4_poly`	>	>
`s4_t4p`	>	>

(c) KT.

Table 5.1: Results for DLP 3.1 on the LWB test set [Horrocks *et al.*, 2000].

Currently, the availability of heavily-optimised provers have rendered many of the formula classes almost trivial, and increasing the parameter does not significantly increase the difficulty of a formula, at least until the formulas become gigantic. Thus the LWB test set, although historically interesting, is nowadays of less importance and should not be used as the only test set for state-of-the-art modal decision procedures.

5.3.2 *The $3CNF_{\Box_m}$ Random Test Set*

The $3CNF_{\Box_m}$ random test set was the first random test set used for testing modal decision procedures. It was proposed in [Giunchiglia and Sebastiani, 1996a; Giunchiglia and Sebastiani, 1996b] as a generalization of the 3SAT random test set (also known as *fixed clause-length model*) that was widely used in propositional satisfiability [Mitchell *et al.*, 1992; Buro and Büning, 1992].

The first version of this test set was heavily criticized in [Hustadt and Schmidt, 1997]. Its current version was proposed in [Giunchiglia *et al.*, 1998] and fixes some of the problems detected in [Hustadt and Schmidt, 1997] and includes some further improvements. In what follows we describe this last version.

Using the $3CNF_{\Box_m}$ random test set we evaluate the performance of a system on test sets containing randomly generated modal formulas in conjunctive normal form (CNF).

A modal CNF formula (from now on a $CNF_{\Box_m}$ formula) is a conjunction of $CNF_{\Box_m}$ clauses, where each clause is a disjunction of either propositional or modal literals. A literal is either an atom or its negation. Modal atoms are formulas of the form $\Box_i C$, where C is a $CNF_{\Box_m}$ clause. A $3CNF_{\Box_m}$ formula is a $CNF_{\Box_m}$ formula where all clauses have exactly 3 literals.

Random formula generation is controlled by the following parameters: i) the *(maximum) modal depth (d)*, ii) the *number of propositional variables (N)*, iii) the *number of distinct box symbols (m)*, iv) the *probability of an atom occurring in a clause at depth* $< d$ *being purely propositional (p)* (to control the "propositional vs. modal" rate for the atoms), and v) the *umber of clauses (L)* (to control the constrainedness of the formula [Williams and Hogg, 1994; Gent *et al.*, 1996]).

The test methodology consist in create test sets for different parameters configurations. To create a $3CNF_{\Box_m}$ test set we fix the parameters N, m, d and p; and vary L in such a way as to empirically cover the "100% satisfiable – 100% unsatisfiable" transition. For each configuration of the five parameters, we generate a certain number of random formulas. These formulas are then given as input to the procedure under test with a timeout of 1000 seconds.

The fraction of satisfiable formulas, median/percentile values of CPU times, and median/percentile values of other parameters, e. g., number of steps, memory, etc., are plotted against the number of clauses L. Figure 5.1 shows example plots for two systems (DLP 4.1 and *SAT 1.3 [Tacchella, 1999]) on a $3CNF_{\Box_m}$ random test set created with parameters $d = 1$, $m = 1$, $p = 0.5$ and N varying from 3 to 9.

The $3CNF_{\Box_m}$ random test set suffers from two major drawbacks: *trivial satisfiability* and *trivial unsatisfiability* [Hustadt and Schmidt, 1997].

A formula is *trivially satisfiable* if its satisfiability can be solved just by purely propositional reasoning. This is, however, not a big problem for random $3CNF_{\Box_m}$ test sets, since a trivially satisfiable formula is not necessarily trivial to solve and its effects are limited to the extreme left part of the satisfiability plots where problems are easy and satisfiable anyway.

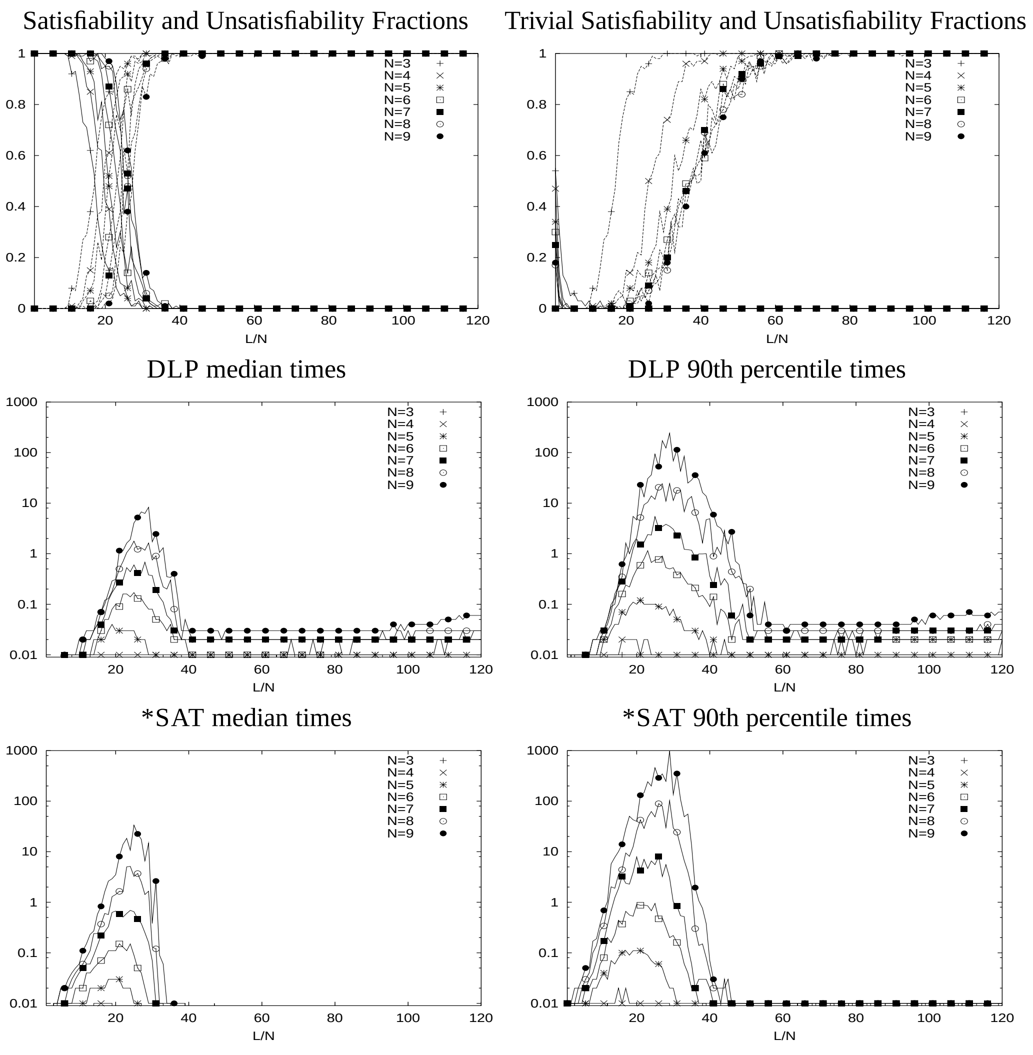


Figure 5.1: Example plots for a test set [Patel-Schneider and Sebastiani, 2003a].

A formula is *trivially unsatisfiable* if its unsatisfiability can be detected by purely propositional reasoning.

The larger the number of clauses (i. e., the value of L), the more likely is the presence of pure propositional clauses and higher the chances of these clauses being jointly unsatisfiable, thus making the formula trivially unsatisfiable. A possible fix for this situation is to set $p = 0$ such that a formula contains no such purely propositional clauses. For $p > 0$, however, trivial unsatisfiability becomes a serious problem in $3CNF_{\Box_m}$ test sets, since a trivially unsatisfiable formula is typically exceedingly easy to solve.

The presence of heavy trivial unsatisfiability is revealed by the presence of a "Half-Dome plot" for the median CPU times and the Qth percentile. For example, Figure 5.2 shows the half-dome shape, whose steep side shows up where the number of trivially unsatisfiable formulas become large before the formulas become otherwise easy to solve.

In [Hustadt and Schmidt, 1999] the authors suggested some guidelines to overcome this problem. However these guidelines introduce new problems like a loss

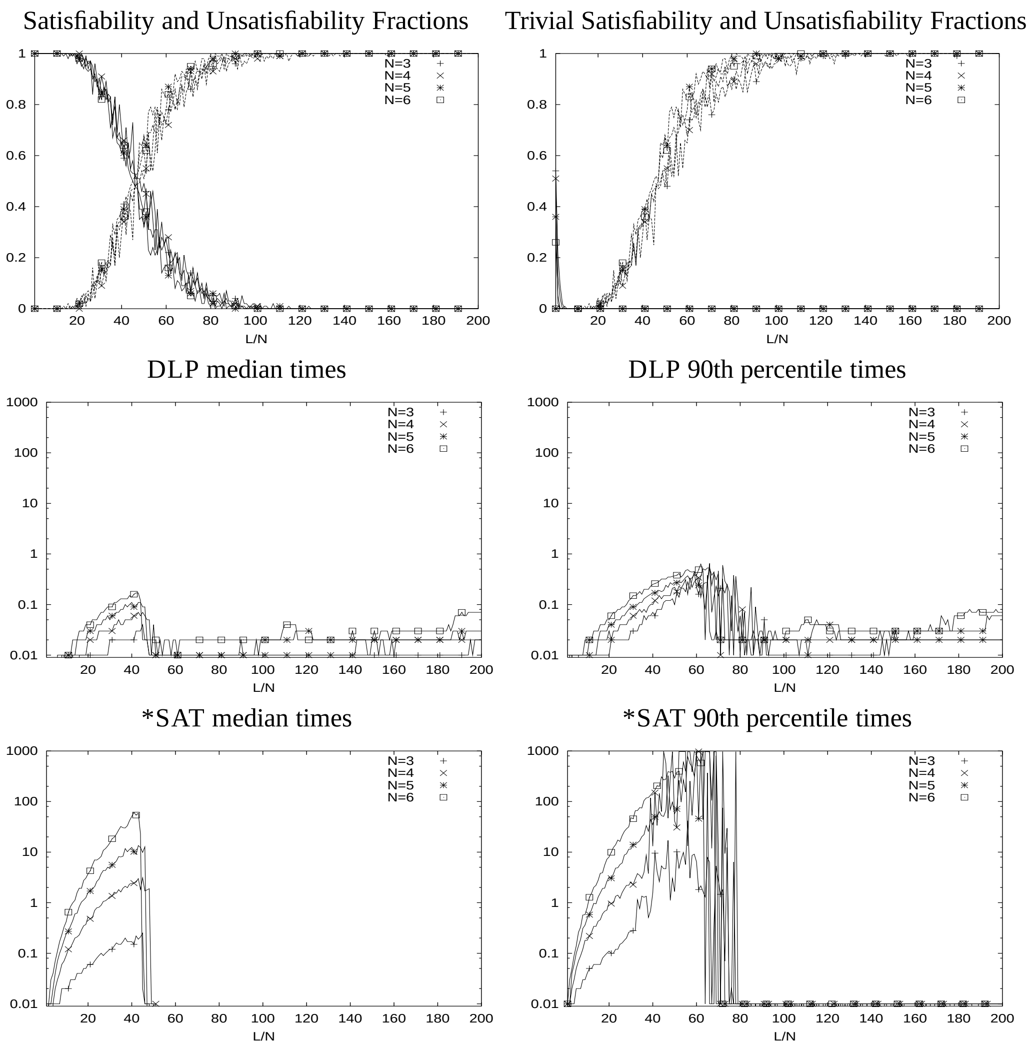


Figure 5.2: Half-dome shape due to trivial unsatisfiability [Patel-Schneider and Sebastiani, 2003a].

of representativeness of the test set, and decay in the complexity of the problem from PSPACE-complete to NP-complete [Halpern, 1995] due to effect of bounding the modal depth of the formulas.

Even with the mentioned problems, the $3CNF_{\Box_m}$ random test set and its associated methodology conforms to most of the criteria of Section 5.2 and therefore it is a good empirical test for many purposes.

5.3.3 *The New_3CNF$_{\Box_m}$ Random Test Set*

The *New_3CNF*$_{\Box_m}$ random test set was presented in [Patel-Schneider and Sebastiani, 2003a] with the aim of fixing and generalizing the $3CNF_{\Box_m}$ random test set.

To avoid generating trivial unsatisfiable formulas, the *New_3CNF*$_{\Box_m}$ test set reinterprets the parameter p (the probability of an atom occurring in a clause at depth $< d$ being purely propositional). Instead of forbidding strictly-propositional clauses except at the maximum modal depth d, by setting $p = 0$, we know require that the ratio between propositional atoms in a clause and the clause size be as close as possible to the propositional probability p for clauses not at the maximum modal depth d. For clauses of size C, if p is k/C for some integral k, this results in all clauses not at modal depth d having k propositional atoms and $C - k$ modal atoms. For other values of p, this results in either $\lfloor pC \rfloor$[1] or $\lceil pC \rceil$[2] propositional atoms in each clause not at modal depth d, with probability $\lceil pC \rceil - pC$ and $pC - \lfloor pC \rfloor$ respectively. If $p \leq (C-1)/C$, this eliminates the possibility of strictly propositional clauses, which are the main cause of trivial unsatisfiability, except at modal depth d. The $New\_3CNF_{\Box_m}$ test set considerably reduces the number of trivially unsatisfiable formulas and almost entirely removes them from the satisfiable/unsatisfiable transition area. Moreover, it is able to generate difficult formulas over a much broader range of L/N than the $3CNF_{\Box_m}$ method.

Another improvement is that a $New\_3CNF_{\Box_m}$ test set also allows the number of literals in a clause C to vary in a manner similar to the number of propositional atoms. If C is an integer then each clause has that many literals. Otherwise, they allow either $\lfloor C \rfloor$ or $\lceil C \rceil$ literals in each clause, with probability $\lceil C \rceil - C$ and $C - \lfloor C \rfloor$, respectively. Then, the number of propositional atoms in each clause is determined based on the number of literals in that clause. This provides another way of controlling difficulty of the test sets. Actually, it allows for a much more fine-grained control, allowing for a direct specification of the probability distribution of the number of propositional atoms in a clause and allowing the distribution to be different for each modal depth from the top level to $d - 1$, and also allowing the direct specification of the probability distribution for the number of literals in a clause at each modal depth.

Thus, the probability distribution for the number of propositional atoms depends on both the modal depth and the number of literals in the clause.

These modifications allows for a fine control of the difficulty of the test sets. To make a test set easier, we can reduce the size of clauses by reducing the value(s) of C, or increase the propositional probability p. This control is missing in the $3CNF_{\Box_m}$ method, as C is restricted to an integral value, and making p much different from 0.0 results in problems with trivial unsatisfiability for maximum modal depths greater than 1.

With respect to the criteria of Section 5.2, the $New\_3CNF_{\Box_m}$ test set inherits all the features of the $3CNF_{\Box_m}$ test set, e. g., scalability, valid vs. not-valid balance, termination, reproducibility, parametrization and data organization, but it also improves in other aspects, like representativeness, difficulty and control.

5.3.4 *The Modalized Test Set*

The modalized test set was proposed by [Massacci, 1999] as a K-modalized variant of the $3CNF_{\Box_m}$ random test set. It borrows an idea from [Halpern, 1995] that consists in, within each $3CNF_{\Box_m}$ formula φ, substitute each occurrence of each propositional variable p_i with the corresponding modal expression $\neg\Box(p_0 \vee \Box^i \neg p_0)$. The enconding preserves satisfiability in K and the resulting formula φ' has only one propositional variable p_0 and depth $d + N + 1$.

1 $\lfloor x \rfloor =_{def} max\{n \in \mathbb{N} \mid n \leq x\}$
2 $\lceil x \rceil =_{def} max\{n \in \mathbb{N} \mid n \geq x\}$

Since a modalized formula is bigger than it non-modalized version, it would be expected for K-modalized $3CNF_{\Box_m}$ formulas to be much harder than their corresponding $3CNF_{\Box_m}$ ones, especially for low d's and high N's. However, experimental results do not confirm this expectation. Despite the increase in depth and size of the formulas, K-modalisation does not seem to produce test sets which are significantly more challenging than standard $3CNF_{\Box_m}$ ones [Horrocks *et al.*, 2000].

5.3.5 *The Random QBF Test Set*

For many modal logics, the class of formulas with bounded depth is in NP [Halpern, 1995], therefore $3CNF_{\Box_m}$ test sets with bounded d have only NP complexity [Massacci, 1999]. To overcome this problem a new kind of random test set was proposed in [Massacci, 1999].

The idea is to generate random QBF formulas [Kleine Büning and Bubeck, 2009; Giunchiglia *et al.*, 2009] according to the method described in [Cadoli *et al.*, 1998], and then convert them into modal formulas by using a variant of the conversion proposed in [Halpern and Moses, 1992]. The resulting modal formulas are satisfiable if and only if the QBF formulas are true.

A typical random QBF test set is defined by fixing the alternation depth D, the number of variables at each alternation V, the number of clauses C and the clause length K, with some constraints on the number of universally and existentially quantified variables within each clause.

The tests are performed on single data points, i.e., on single tuples of parameter's values. For each data point, a certain number of QBF formulas are randomly generated, converted to modal logics and the resulting formulas are given as input to the procedure being tested with a maximum timeout. The number of tests that have been solved within the timeout and the geometrical mean time for successful solutions are then reported. Data are rescaled to abstract away machine and run-dependent characteristics, and then presented in a collection of tables presenting a data pair for each system under test, one data point per row.

Even though it is claimed that modal-encoded QBF formulas can capture the problems in Σ_D^P, as QBF formulas with bounded D and unbounded V are in Σ_D^P, while $3CNF_{\Box_m}$ formulas with bounded d and unbounded N are stuck at NP [Massacci, 1999]. In [Horrocks *et al.*, 2000], it is shown that this statement is misleading, and, in fact, similarly to modal logic $\mathsf{K}_{(m)}$ with bounded depth, the class of random QBF formulas with bounded number of universally quantified variables is in NP. Moreover, as in the case of $3CNF_{\Box_m}$ formulas with bounded d and N, if D and V are bounded, which is the case of every finite-size test set, then the random QBF problems are not only in NP, but even in P.

Another problem with modal-encoded QBF formulas is that they are rather artificial, as their potential models are restricted to those having the regular structure imposed by the QBF and/or binary search trees. Finally, a serious problem with random modal-encoded QBF formulas is their size. Initial versions of the translation method produced test sets in the 1GB range.

5.4 empirical testing in satisfiability modulo theories

We now present the basic aspects of empirical testing for Satisfiability Modulo Theories (SMT). The situation of empirical testing in SMT is much more favorable than for modal logics due to the existence of a well established set of benchmarks, the "SMT Library" (SMT-LIB), that has a great support by the SMT community, mainly because formulas in SMT-LIB format are accepted by the great majority of current SMT solvers and most of the published experimental works on SMT rely significantly on SMT-LIB benchmarks.

5.4.1 *The SMT Library*

The SMT Library (SMT-LIB) [Barrett *et al.*, 2010a; Barrett *et al.*, 2010b] is an international initiative that started in 2003 [Ranise and Tinelli, 2003] with the aims of providing standard rigorous descriptions of background theories used in SMT systems; develop and promote common input and output languages for SMT solvers; and establish and make available a large library of benchmarks (95000+ formulas) for SMT solvers. The current version of the SMT-LIB standard is Version 2.0.

The main motivation of the SMT-LIB is to facilitate the evaluation and the comparison of SMT systems by providing common standards and a library of benchmarks, and thus contributing to advance the state-of-the-art in the field, in the same way as the TPTP library [Sutcliffe, 2009] has done for theorem proving, or the SATLIB library [Hoos and Stützle, 2000] has done for propositional satisfiability.

Informally speaking, from the SMT-LIB perspective, an SMT solver is any software system that implements a procedure for satisfiability modulo some given theory.

Usually, given an SMT solver, one can identify the following main components:

i) The *underlying logic*, e. g., firt-order, modal, temporal, etc.,

ii) The *background theory*, the theory against which satisfiability is checked,

iii) The *input formulas*, the class of formulas the solvers accepts as input, and

iv) The *interface*, the set of functionalities provided by the solver.

For instance, in a solver for linear arithmetic the underlying logic is first-order logic with equality, the background theory is the theory of real numbers, and the input language is limited to conjunctions of inequations between linear polynomials. The interface may be as simple as accepting a system of inequations and returning a binary response indicating whether the system is satisfiable or not. More sophisticated interfaces include the ability to return concrete solutions for satisfiable inputs, return proofs for unsatisfiable ones, allow incremental and backtrackable input, and so on.

The SMT-LIB standard takes into account these aspects by defining a language, the *SMT-LIB language*, and a logic, the *SMT-LIB logic*. In what follows we provide an informal description of the language and the logic, and refer the interested reader to [Barrett *et al.*, 2010b] for more details.

5.4.1.1 *The SMT-LIB Language*

The SMT-LIB language defines: a syntax, similar to that of the LISP programming language (in fact, every expression is a legal S-expression of Common Lisp [Steele, 1990]); how to write terms and formulas, background theories, logics; and a command language for interacting with SMT solvers.

TERMS AND FORMULAS In the SMT-LIB language, terms are constructed out of constant symbols, variables, function symbols, three kinds of binders (`let`, `forall` and `exists`) and an annotation operator (`!`).

In its simplest form, a term is a special constant symbol, a variable, a function symbol, or the application of a function symbol to one or more terms. Function symbols applied to no arguments are used as constant symbols. More complex terms include let, `forall` and `exists` binders. The `forall` and `exists` binders correspond to the usual existential and universal quantifiers of first-order logic, except that the variables they quantify are sorted. A `let` binder introduces and defines one or more local variables in parallel.

All terms of the SMT-LIB language are additionally required to be well-sorted. Well-sorted terms are a subset of the set of all terms. Formulas are well-sorted terms of sort `Bool`. As a consequence, there is no syntactic distinction between function and predicate symbols, the later are simply functions returning `Bool`. Figure 5.3 shows an example formula in the SMT-LIB language.

```
(forall (( x (List Int)) (y (List Int)))
  (= (append x y)
     (ite (= x (as nil (List Int)))
          y
          (let ((h (head x)) (t (tail x)))
            (insert h (append t y))))))
```

Figure 5.3: Formula in the SMT-LIB language.

THEORY DECLARATION A theory defines a vocabulary of sorts and functions, and it also associates a sort with relevant literals. For example, the `Ints` theory defines the `Int` and `Bool` sorts and declares that any `<numeral>` has sort `Int`.

One of the goals of the SMT-LIB is to define a *catalog of background theories*, starting with a small number of popular ones, and adding new ones as solvers for them are developed. Table 5.2 shows the last version of the catalog of theories.

To do so, the SMT-LIB provides syntactic support for defining theories in the SMT-LIB format. Theories are specified independently of any benchmarks or solvers, and each SMT-LIB script may refer, indirectly, to one or more theories in the SMT-LIB catalog.

Theory specifications have mostly documentation purposes. They are meant to be standard references for human readers. For practicality then, the format insists that only the *signature* of a theory, i. e., its set of sorts and sorted function symbols, be specified formally, provided it is finite. By "formally" the standard means written in a machine-readable and processable format, as opposed to written in free text, no matter how rigorously.

Theory	Description
ArraysEx	Functional arrays with extensionality.
FixedSizeBitVectors	Bit vectors with arbitrary size.
Core	Core theory, defining the basic Boolean operators.
Ints	Integer numbers.
Reals	Real numbers.
Reals_Ints	Real and integer numbers.

Table 5.2: SMT-LIB Theories.

Some theories, such as the theory of bit vectors, have an infinite signature. For them, the signature too is specified informally in English. Figure 5.4 shows the definition of the `Core` theory that defines propositional logic. Every theory in the SMT-LIB implicitly contains the `Core` theory.

LOGICS DECLARATION In SMT-LIB, a sublogic (or just a logic) is defined to consist of one or more theories, together with some restrictions on the kinds of expressions that may be used within that logic. For example, the QF_LIA logic includes both the `Core` and `Ints` theories. Those theories define many of the usual operations on `Bool` and `Int` values. However the QF_LIA logic does not allow quantified expressions and allows only linear arithmetic (e.g., multiplication must be by a literal, one cannot multiply two non-literals together). Such restrictions are introduced because there are known decision procedures that can solve satisfiability problems in these cases. Table 5.3 shows the logics supported by the SMT-LIB.

Similar to the definition of theories, in the SMT-LIB, logic definitions are mostly for documentation purposes. Figure 5.5 shows the declaration of the `QF_IDL` logic.

COMMAND LANGUAGE The SMT-LIB includes a scripting language that defines a textual interface for SMT solvers. SMT solvers implementing this interface act as interpreters of the scripting language. The language is command-based, and defines a number of input/output functionalities that go well beyond simply checking the satisfiability of an input formula. In line with the LISP-like syntax, all commands look like LISP-function applications, with a command name applied to zero or more arguments. To facilitate processing, each command takes a constant number of arguments, although some of these arguments can be (parenthesis delimited) lists of variable length.

The intended use of scripts is to communicate with an SMT solver in a read-eval-print loop until a termination condition occurs. The solver reads the next command, acts on it, prints a response, and repeats. Possible responses vary from a single symbol to a list of attributes, to complex expressions like proofs.

Table 5.4 shows the most common commands found in the SMT-LIB benchmarks. Other commands exists, mainly for information retrieval, e.g., `get-assertions`, `get-proof`, `get-unsat-core`, `get-value`, `get-assignment`, etc.

A typical formula in the SMT-LIB is just an script that defines the logic, introduces sorts, function and predicate symbols and makes a number of assertion. Fig-

Logic	Description
AUFLIA	Closed formulas over the theory of linear integer arithmetic and arrays extended with free sort and function symbols but restricted to arrays with integer indices and values.
AUFLIRA	Closed linear formulas with free sort and function symbols over one- and two-dimentional arrays of integer index and real value.
AUFNIRA	Closed formulas with free function and predicate symbols over a theory of arrays of arrays of integer index and real value.
LRA	Closed linear formulas in linear real arithmetic.
QF_ABV	Closed quantifier-free formulas over the theory of bitvectors and bitvector arrays.
QF_AUFBV	Closed quantifier-free formulas over the theory of bitvectors and bitvector arrays extended with free sort and function symbols.
QF_AUFLIA	Closed quantifier-free linear formulas over the theory of integer arrays extended with free sort and function symbols.
QF_AX	Closed quantifier-free formulas over the theory of arrays with extensionality.
QF_BV	Closed quantifier-free formulas over the theory of fixed-size bitvectors.
QF_IDL	Difference Logic over the integers.
QF_LIA	Unquantified linear integer arithmetic. In essence, Boolean combinations of inequations between linear polynomials over integer variables.
QF_LRA	Unquantified linear real arithmetic. In essence, Boolean combinations of inequations between linear polynomials over real variables.
QF_NIA	Quantifier-free integer arithmetic.
QF_NRA	Quantifier-free real arithmetic.
QF_RDL	Difference Logic over the reals. In essence, Boolean combinations of inequations of the form $x - y < b$ where x and y are real variables and b is a rational constant.
QF_UF	Unquantified formulas built over a signature of uninterpreted (i.e., free) sort and function symbols.
QF_UFBV	Unquantified formulas over bitvectors with uninterpreted sort function and symbols.
QF_UFIDL	Difference Logic over the integers (in essence) but with uninterpreted sort and function symbols.
QF_UFLIA	Unquantified linear integer arithmetic with uninterpreted sort and function symbols.
QF_UFLRA	Unquantified linear real arithmetic with uninterpreted sort and function symbols.
QF_UFNRA	Unquantified non-linear real arithmetic with uninterpreted sort and function symbols.
UFLRA	Linear real arithmetic with uninterpreted sort and function symbols.
UFNIA	Non-linear integer arithmetic with uninterpreted sort and function symbols.

Table 5.3: SMT-LIB Logics.

```
(theory Core
  :sorts ((Bool 0))
  :funs ((true Bool) (false Bool) (not Bool Bool)
         (=> Bool Bool Bool :right-assoc) (and Bool Bool Bool :left-assoc)
         (or Bool Bool Bool :left-assoc) (xor Bool Bool Bool :left-assoc)
         (par (A) (= A A Bool :chainable))
         (par (A) (distinct A A Bool :pairwise))
         (par (A) (ite Bool A A A))
        )
  :definition
  "For every expanded signature Sigma, the instance of Core with that signature
   is the theory consisting of all Sigma-models in which:
   - the sort Bool denotes the set {true, false} of Boolean values;
   - for all sorts s in Sigma,
   - (= s s Bool) denotes the function that
     returns true iff its two arguments are identical;
   - (distinct s s Bool) denotes the function that
     returns true iff its two arguments are not identical;
   - (ite Bool s s) denotes the function that
     returns its second argument or its third depending on whether
     its first argument is true or not;
   - the other function symbols of Core denote the standard Boolean operators
     as expected.
  "
  :values "The set of values for the sort Bool is {true, false}."
 )
```

Figure 5.4: The `Core` theory declaration.

ure 5.6 shows a formula from the SMT-LIB, corresponding to the `eq_diamond2` problem.

5.4.1.2 *The SMT-LIB Logic*

SMT-LIB adopts as its *underlying logic* a version of the many-sorted first-order logic with equality [Gallier, 1985; Manzano, 1993; Enderton, 2001] that incorporates some features of higher-order logics like the identification of formulas with terms of a distinguished Boolean sort, and the use of sort symbols of arity greater than 0.

In many-sorted logics, terms are typed, or sorted, and each sort is denoted by a sort symbol from a set of sort symbols. In the SMT-LIB logic, the language of sorts is extended from sort symbols to sort terms built with symbols from the set of sort symbols. For example, if `Int` and `Real` are sort symbols of arity 0, and `List` and `Array` are sort symbols of respective arity 1 and 2, then the expression `List (Array Int (List Real))` and all of its subexpressions are sorts. Unlike traditional many-sorted logic, however, it does not have a syntactic category of formulas distinct from terms. Formulas are just sorted terms of a distinguished Boolean sort, i. e., interpreted as a two-element set in every SMT-LIB theory.

Finally, in addition to the usual existencial and universal quantifiers, the logic includes a *let* binder analogous to the local variable binders found in many programming languages.

```
(logic QF_IDL
 :smt-lib-version 2.0
 :written_by "Cesare Tinelli"
 :date "2010-04-30"
 :theories ( Ints )
 :language
 "Closed quantifier-free formulas with atoms of the form:
  - q
  - (op (- x y) n),
  - (op (- x y) (- n)), or
  - (op x y)
  where
    - q is a variable or free constant symbol of sort Bool,
    - op is <, <=, >, >=, =, or distinct,
    - x, y are free constant symbols of sort Int,
    - n is a numeral.")
```

Figure 5.5: The `QF_IDL` logic declaration.

Command	Description
`set-logic`	Initializes the solver with the specified logic.
`declare-fun`	Declares new uninterpreted symbols. Constants and functions are declared in a uniform way; constants are simply functions with no arguments.
`define-fun`	Declares a new function symbol that is equivalent to a given expression.
`declare-sort`	Declares a new sort.
`define-sort`	Introduces a new symbol that is the abbreviation for a sort expression.
`assert`	Instructs the solver to assume that the stated formula is true.
`check-sat`	Once a series of assert commands have been made, the check-sat command instructs the solver to test for satisfiability.
`set-option`	Sets the value of a specified option.
`set-info`	Sets information about the solver being used.
`exit`	Terminates the solver.

Table 5.4: SMT-LIB Commands.

```
(set-logic QF_UF)
(set-info :source |
Generating minimum transitivity constraints in P-time for deciding Equality
Logic, Ofer Strichman and Mirron Rozanov, SMT Workshop 2005.

Translator: Leonardo de Moura. |)
(set-info :smt-lib-version 2.0)
(set-info :category "crafted")
(set-info :status unsat)
(declare-sort U 0)
(declare-fun x0 () U)
(declare-fun y0 () U)
(declare-fun z0 () U)
(declare-fun x1 () U)
(declare-fun y1 () U)
(declare-fun z1 () U)
(assert (and (or (and (= x0 y0) (= y0 x1))
                      (and (= x0 z0) (= z0 x1)))
             (not (= x0 x1))))
(check-sat)
(exit)
```

Figure 5.6: A typical SMT-LIB script (eq_diamond2.smt2).

Part II

DETECTING AND EXPLOITING SYMMETRIES

"Socrates: *Then, if we are not able to hunt the goose with one idea, with three we may take our prey; Beauty, Symmetry, Truth are the three...*"

Plato. *Philebus*. 65a.

6 SYMMETRY DETECTION

The first step towards exploiting symmetries in any logic, is to detect them. Much research has been devoted to this subject, leading to different symmetry detection techniques. In this chapter we present a survey of the most relevant symmetry detection techniques.

We start by reviewing graph-based techniques for symmetry detection. We briefly describe the basics of the graph automorphism algorithm implemented in available tools and present a number of reduction algorithms from propositional CNF formulas to graphs (Section 6.1). We then present a brief description of formula-based techniques for symmetry detection (Section 6.2).

6.1 GRAPH-BASED SYMMETRY DETECTION

Graph-based symmetry detection is the most common approach for detecting symmetries in formulas, and it has encountered many applications in different domains ranging from SAT solving to finite model generation.

Two key aspects explain its success. First, it is conceptually simple: the idea is to construct a graph from a formula such that the automorphisms of the graph correspond to symmetries of the formula. Second, the availability of efficient graph automorphism tools, that can handle large graphs, makes it easy to implement and integrate with current solvers. Key to this approach is the graph automorphism problem and its related tools.

6.1.1 *The Graph Automorphism Problem*

We start our discussion of the graph automorphism problem by describing a closely related problem which is also one of the most studied subject in computer science: the *graph isomorphism problem*.

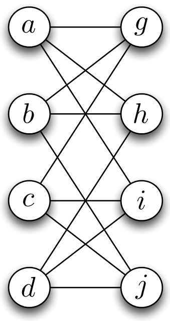
The graph isomorphism problem can be stated as the problem of checking if two graphs that look different are actually the same.

Definition 6.1 (The graph isomorphism problem)**.** *Let a graph be a pair $G = (V, E)$ where V is a set of vertices and $E \subseteq V \times V$ is the set of edges. Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, does there exist a bijection f from the vertex set V_1 to V_2 such that $\forall a, b \in V_1, (a, b) \in E_1 \leftrightarrow (f(a), f(b)) \in E_2$.*

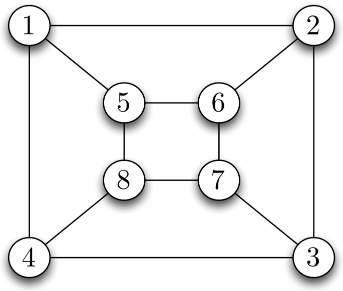
Example 6.1. *Consider the graphs A and B of Figures 6.1a and 6.1b respectively.*

An isomorphism between graphs A and B is given by the following function:

$$\begin{aligned} f(a) &= 1, & f(g) &= 5, \\ f(b) &= 6, & f(h) &= 2, \\ f(c) &= 8, & f(i) &= 4, \\ f(d) &= 3, & f(j) &= 7. \end{aligned}$$



(a) Graph A.



(b) Graph B.

Figure 6.1: Graph isomorphism example.

In the field of complexity theory, the graph isomorphism problem is one of the few problems which is in NP but it is not known to be either in P or NP-complete. It is also not known to be in co-NP [Fortin, 1996]. Since it has resisted efforts to classify it in either P or NP-complete, researchers have taken other approaches to determining its complexity. These approaches range from generalizing the notion of NP to more "esoteric" complexity classes [Goldwasser *et al.*, 1989], to define its own complexity class, GI, of the set of problems with a polynomial-time Turing reduction to the graph isomorphism problem [Booth and Colbourn, 1979; Köbler *et al.*, 1994].

Closely related to the graph isomorphism problem, and of central importance to us, is the *graph automorphism problem*, which is the problem of testing whether a graph has a non-trivial automorphism.

Definition 6.2 (The graph automorphism problem)**.** *Given a graph $G = (V, E)$, does there exist a bijection f, different from the identity function, onto the vertex set V such that $\forall a, b \in V, (a, b) \in E \leftrightarrow (f(a), f(b)) \in E$.*

Example 6.2. *Consider the graph of Figure 6.2. An automorphism of the graph is given by*

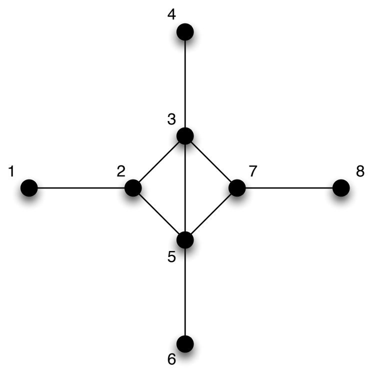


Figure 6.2: Graph automorphism example.

the following function:

$$\begin{array}{rclrcl} f(1) & = & 8, & f(5) & = & 5, \\ f(2) & = & 7, & f(6) & = & 6, \\ f(3) & = & 3, & f(7) & = & 2, \\ f(4) & = & 4, & f(8) & = & 1. \end{array}$$

If such a bijection exists it is called an *automorphism* (or *symmetry*). Notice that an *automorphism* is just an isomorphism that maps a graph G into itself.

Like the graph isomorphism problem, this problem belongs to the class NP, and it is unknown whether it has a polynomial time algorithm or it is NP-complete.

The *colored graph automorphism problem* is a variant of the graph automorphism problem where the automorphisms are constrained by vertex colors, such that each automorphism must map each vertex into a vertex of the same color. Color constraints are specified by means of *ordered partitions* (or *colorings*).

Definition 6.3 (Partition and Coloring)**.** *Let $G = (V, E)$ be a graph. A* partition *of V is a set $\pi = \{V_1, V_2, \ldots, V_n\}$ of non-empty, disjoint subsets V_i (called* cells*) of V such that $\bigcup_i V_i = V$. An* ordered partition *(also known as* coloring*) is a sequence $\pi = (V_1, V_2, \ldots, V_n)$ such that $\{V_1, V_2, \ldots, V_n\}$ is a partition of V. A partition is called* discrete *if all its cells are singleton sets and it is called* unit *if it has only one cell (the set V).*

The set of automorphism of a graph forms a group under composition.

Definition 6.4 (Automorphism Group)**.** *Given a graph $G = (V, E)$ and a coloring $\pi = (V_1, V_2, \ldots, V_k)$ of V, let S_n denote the set of all permutations over the set V. The* automorphism group *of G, $Aut(G, \pi)$, is $\{\sigma \in S_n \mid \sigma(G) = G \text{ and } \sigma(\pi) = \pi\}$, where $\sigma(G) = (\sigma(V), \sigma(E))$ with $\sigma(E) = \{(\sigma(u), \sigma(v)) \mid (u, v) \in E\}$ and $\sigma(\pi) = (\sigma(V_1), \sigma(V_2), \ldots, \sigma(V_k))$.*

In other words, $Aut(G, \pi)$ is the set of permutations of the graph vertices that map edges to edges with the restriction that vertices in any given cell of π can only be mapped to vertices in that same cell.

One of the factors contributing to the large amount of work on the graph automorphism (isomorphism) problem is the many practical applications it has. This has lead to the development of practical algorithms and tools capable of handling large graphs efficiently.

The first of such tools was `Nauty` [McKay, 1981; McKay, 2007]. It was originally conceived to solve the graph isomorphism problem by canonical labeling. A canonical label of a graph, is a function C such that given two graphs G and H, $C(G) = C(H)$ if and only if G and H are isomorphic.

The algorithm implemented by `Nauty` remains at the core of more recent tools like `Saucy` [Darga *et al.*, 2004; Darga *et al.*, 2008; Katebi *et al.*, 2012] and `Bliss` [Junttila and Kaski, 2007].

The central computation in `Nauty`'s algorithm is the *ordered partition refinement*, similar to the one used for state minimization of finite-state automata [Aho and Hopcroft, 1974]. Given an initial coloring π of a graph $G = (V, E)$, ordered partition refinement tries to transform it into a *finer* partition π' that maximally distinguishes unmappable vertices. This is done by splitting cells containing vertices with different *color-relative vertex degrees*.

Definition 6.5 (Color-relative Vertex Degree). *Given a coloring $\pi = (V_1, V_2, \ldots, V_k)$ of the graph $G = (V, E)$ and a vertex $v \in V$, let $d(v, V_i)$ be the number of vertices in V_i that are adjacent to v in G. Note that $d(v, V)$ is simply the degree of v in G.*

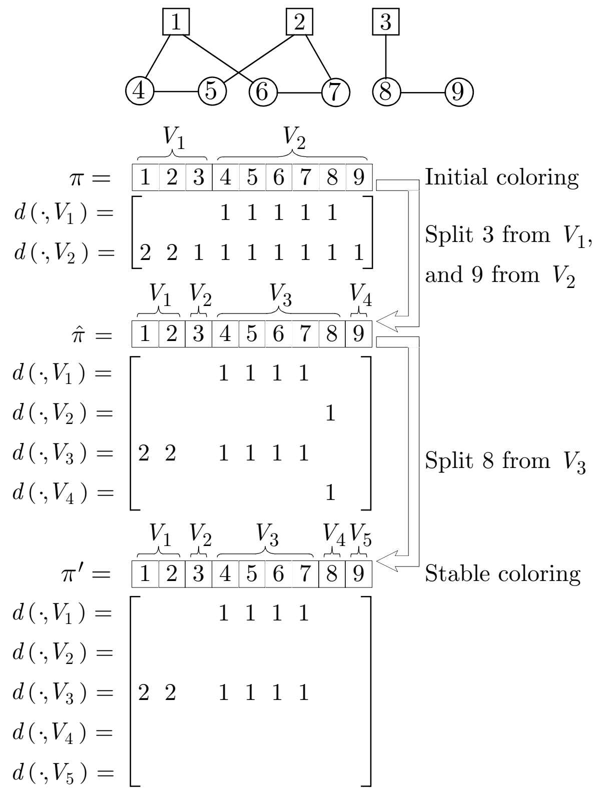
The process of splitting some cells induces further refinement as each vertex in a cell must have the same number of connections to vertices in every cell in the coloring, otherwise they can be distinguished. If no further refinement is possible, then the coloring is *stable* and the refinement process stops.

Definition 6.6 (Stable Coloring). *A coloring π is* stable *if*

$$d(u, V_i) = d(v, V_i), 1 \leq i \leq |\pi|$$

for all pair of vertices $u, v \in V_j, 1 \leq j \leq |\pi|$.

Example 6.3 ([Sakallah, 2009]). *Figure 6.3 illustrates the ordered partition refinement procedure on a 2-colored graph with 9 vertices. The initial coloring π consists of two cells corresponding to the graph's two types of vertices (square and round). In the first refinement iteration, vertex 3 is split off from V_1 because its degree relative to V_2 is different from those of the two other vertices in V_1, i.e., vertex 3 can be distinguished from vertices 1 and 2. Similarly, vertex 9 is split off from V_2 yielding the intermediate coloring $\hat{\pi}$. The second refinement iteration splits vertex 8 from V_3 yielding the final stable coloring π'.*



1
2
3
4
5
6
7
8
9
V_1
V_2
$\pi =$ 1 2 3 4 5 6 7 8 9
Initial coloring
$d(\cdot, V_1) =$ 1 1 1 1 1
$d(\cdot, V_2) =$ 2 2 1 1 1 1 1 1 1
Split 3 from V_1, and 9 from V_2
V_1 V_2 V_3 V_4
$\hat{\pi} =$ 1 2 3 4 5 6 7 8 9
$d(\cdot, V_1) =$ 1 1 1 1
$d(\cdot, V_2) =$ 1
$d(\cdot, V_3) =$ 2 2 1 1 1 1
$d(\cdot, V_4) =$ 1
Split 8 from V_3
V_1 V_2 V_3 V_4 V_5
$\pi' =$ 1 2 3 4 5 6 7 8 9
Stable coloring
$d(\cdot, V_1) =$ 1 1 1 1
$d(\cdot, V_2) =$
$d(\cdot, V_3) =$ 2 2 1 1 1 1
$d(\cdot, V_4) =$
$d(\cdot, V_5) =$

Figure 6.3: Ordered partition refinement [Sakallah, 2009].

If the refinement procedure returns a discrete coloring π', i.e., every cell of the partition is a singleton, then all vertices can be distinguished implying that G has no symmetries besides the identity. However, if π' is not discrete, then there is some non-singleton cell in π' representing vertices that could not be distinguished based on degree and are, thus, member of a potential symmetry. The existence of symmetries is checked by selecting some non-singleton cell T of π', called the *target cell*, and forming $|T|$ descendant colorings from π', each identical to π' except that a distinct $t \in T$ is placed in front of $T - \{t\}$ (remember that a coloring is an ordered partition of the set of vertices). Each of these colorings is subsequently refined, and further descendant colorings are generated if the refined colorings are not discrete; this process is iterated until discrete colorings are reached. The colorings explored in this fashion form a *search tree* with the discrete colorings at the leaves. The leaves of the search tree represent possible symmetries of G.

The next step in the process is to derive from these colorings permutations of the graph vertices that correspond to graph symmetries. This is done by choosing one of these colorings as a reference, and computing the permutations that transform it to the other colorings, i. e., if π_1 and π_2 are discrete colorings and $\sigma(\pi_1) = \pi_2$ then σ is a possible symmetry of G. σ is a symmetry of G if and only if $\sigma(G) = G$.

We can enumerate $Aut(G, \pi)$ by fixing the first leaf encountered in the search, denoted as τ, as a reference coloring, and comparing it to every other discrete coloring, i. e., $Aut(G, \pi) = \{\sigma \mid \pi \text{ discrete }, \tau^{\sigma} = \pi, \text{ and } \sigma(G) = G\}$.

A permutation that does not correspond to a symmetry of the graph triggers backtracking in the search tree so that other branches can be explored.

The performance of this graph automorphism algorithm depends critically on aggressive pruning of the search tree to avoid deriving permutations that can be obtained as the product of other permutations already found. Ideally the output of the algorithm should be a set of irredundant generators. In practice, however the overhead of performing the group theoretic pruning necessary to guarantee this outcome tends to be excessive. Instead, graph automorphism tools are designed to produce at most $n - 1$ generators for an input graph with n vertices, providing an exponentially smaller representation of the complete set of symmetries which is often, but not guaranteed to be, irredundant.

Example 6.4 ([Sakallah, 2009]). *Figure 6.4 shows the resulting search tree for a 6-vertex graph. The initial coloring has a single cell since all vertices are of the same color. Refinement yields a stable coloring with two non-singleton cells. At this point, the first cell is chosen as a target, and the algorithm branches by creating three descendant colorings. The process is now repeated, i. e., each derived coloring is refined and the algorithm branches from it if it has a non-singleton cell, in a depth-first manner until discrete colorings are reached. In this example, the search tree terminates in six leaves corresponding to six different discrete colorings.*

To derive the permutations, we chose the left-most coloring π_0 as the reference; the permutations that convert it to the other colorings, including itself, are labeled γ_0 to γ_5. Each such permutation γ_i is then checked to determine if it is a graph symmetry, i. e., if $\gamma(G) = G$. In this example, all six permutations are indeed symmetries of the graph, yielding $Aut(G, \pi) = \{\gamma_0, \gamma_1, \gamma_2, \gamma_3, \gamma_4, \gamma_5\}$. Notice that in this example, the portion of the search tree enclosed by the dashed outline (nodes F, G, H, and I) can be safely pruned away since the permutations identified at its leaves can be expressed in terms of permutations γ_1 and γ_2.

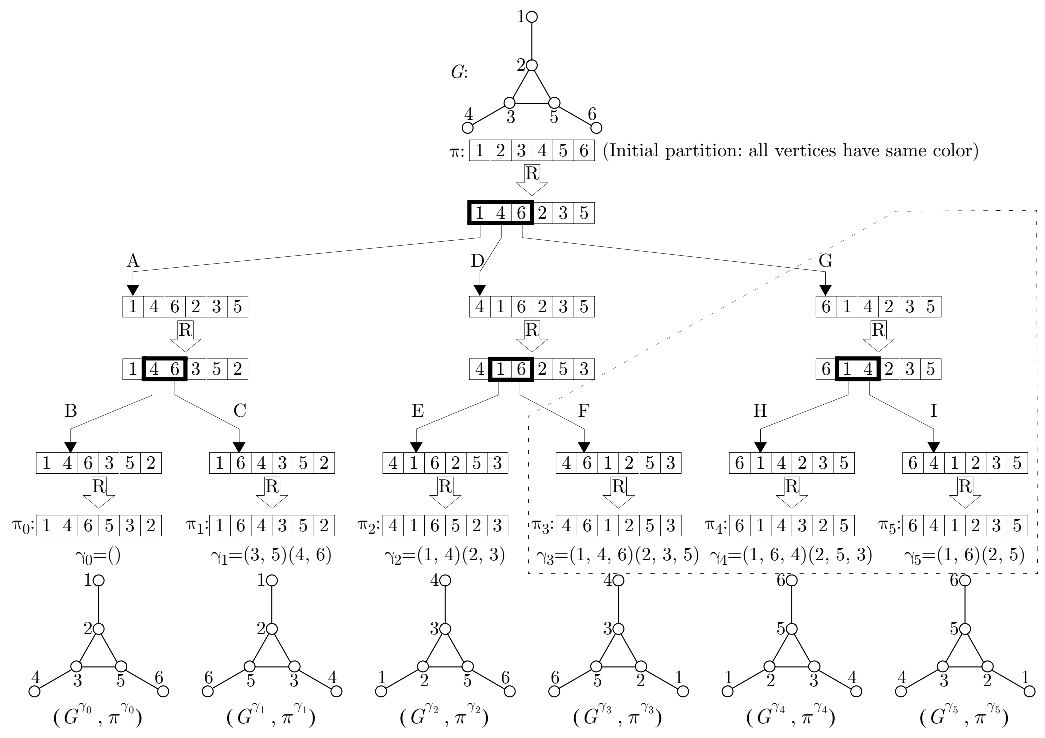


Figure 6.4: Basic flow of the ordered partition refinement procedure [Sakallah, 2009].

Modern implementations of this algorithm, like Saucy and Bliss are capable of handling graphs with millions of vertices efficiently. In particular, both implementations take advantage of the *sparsity* of both the graphs and the generators of their automorphism groups. In this context, sparsity is taken to mean that the average vertex degree in the graph and the average support (the elements not mapped to themselves by a permutation, see Appendix A) of the resulting symmetry generators are both much smaller than the number of graph vertices. Incorporating knowledge of both types of sparsity in the basic automorphism algorithm can result in a substantial pruning of the search tree, essentially yielding a tree whose size is linear, rather than quadratic, in the total support of the symmetry generators [Darga *et al.*, 2008].

As already mentioned, the idea underlying graph-based techniques is to construct a graph from a formula such that the automorphisms of the graph correspond to symmetries of the formula, i. e., to construct a graph such that its automorphism group is isomorphic to the symmetry group of the formula.

Since groups are often described by sets of generators, it is important to know that isomorphisms preserve such descriptions.

Theorem 6.1. *Any group isomorphism maps sets of generators to sets of generators, and maps irredundant sets of generators to irredundant sets of generators.*

Proof. See [Aloul *et al.*, 2003b]. □

Theorem 6.1 tells us that if we build a graph from a formula, such that the automorphism group of the first is isomorphic to the symmetry group of the second, then every generator of the automorphism group corresponds to a generator of the symmetry group.

Theorem 6.1 also provides a criteria against which to compare different reduction algorithms, namely, if they create graphs having an automorphism group that is isomorphic to the symmetry group of the formula from which they were created or not. If the groups are not isomorphic, then we might find *spurious symmetries*, i. e., automorphisms of the graph that do not correspond to a symmetry of the formula.

Another important criteria to compare reduction algorithms is which type of symmetry they can detect. In SAT solving, we are interested in detecting two types of symmetries: *permutational* and *phase-shift*.

Definition 6.7 (Permutational and Phase-shift Permutations). *Let* PROP *be a set of propositional variables and* PLIT *the set of literals over* PROP *(i. e.,* $\mathsf{PLIT} = \mathsf{PROP} \cup \{\neg p_i \mid p_i \in \mathsf{PROP}\}$*). Let* $\sigma : \mathsf{PLIT} \mapsto \mathsf{PLIT}$ *be a permutation of propositional literals. We say* σ *is* permutational *if* σ *maps literals without changing their polarity, e. g.,* $\sigma(p) = q$ *for all* $p, q \in \mathsf{PLIT}$*. We say* σ *is a* phase-shift *if it changes the polarity of at least one pair of literals, e. g.,* $\sigma(p) = \neg q$ *for* $p, \neg q \in \mathsf{PLIT}$.

6.1.2 *Reduction Algorithms*

We now present a survey of different algorithms (called *reduction algorithms*) to transform a propositional formula in CNF into a colored graph.

The first algorithm was introduced in [Crawford, 1992] where the idea of reducing the symmetry detection problem to the graph isomorphism problem was first shown.

The algorithm aims at detecting a particular type of symmetries, called *simple symmetries*. A *simple symmetry* of a CNF formula φ is a permutation σ such that $\sigma(\varphi) = \varphi$ and $\sigma(p) = q$ for a given pair p, q of propositional variables. Given a CNF formula and a pair of *goal* variables p and q, the algorithm constructs two (almost identical) colored graphs $G(\varphi, p)$ and $G(\varphi, q)$ such that only isomorphisms mapping p to q are detected and every detected isomorphism is a symmetry of the input formula.

Definition 6.8 (Crawford I reduction algorithm). *Let* φ *be a CNF formula and let* p *and* q*, two propositional variables occurring in* φ*, be the* goals.

First construct a graph $G(\varphi)$ *as follows:*

i) *For each propositional variable* p *in* $Vars(\varphi)$*:*
   a) *Add three vertices: one positive-literal vertex of color* 0*, one negative-literal vertex of color* 1*, and one inverse vertex of color* 2.
   b) *Add a link between the positive-literal and the inverse vertices.*
   c) *Add a link between the negative-literal and the inverse vertices.*

ii) *For each clause in* φ*:*
   a) *Add a vertex of color* 3.
   b) *Add an edge between the clause vertex and each literal vertex corresponding to a literal occurring in it.*

To construct the graphs $G(\varphi, p)$ and $G(\varphi, q)$:

i) *From the graph $G(\varphi)$ create two graphs, $G(\varphi, p)$ and $G(\varphi, q)$.*

   a) *In $G(\varphi, p)$, type the vertex corresponding to p as a* goal *vertex of color* 4.

   b) *In $G(\varphi, q)$, type the vertex corresponding to q as a* goal *vertex of color* 4.

For a formula with V variables and C clauses, this construction produces two graphs with 5 *colors and $3V + C$ vertices each.*

Example 6.5. *Consider the formula*

$$\varphi = (p \vee \neg q \vee \neg r) \wedge (\neg p \vee q \vee r) \wedge (\neg q \vee r).$$

Figure 6.5 shows the associated colored graphs (colors are represented by shapes and shades in the figure) constructed using the algorithm of Definition 6.8. First, notice that this reduction algorithm results in two graphs, $G(\varphi, p)$ (Figure 6.5a) and $G(\varphi, q)$ (Figure 6.5b), that differ only in which is their goal *node. The goal nodes are used to force any isomorphism between $G(\varphi, p)$ and $G(\varphi, q)$ to map p to q as required. Therefore with this reduction algorithm we cannot detect the only non-trivial symmetry of the formula, namely $\sigma = (p\ \neg p)(q\ \neg r)(\neg q\ r)$, because we look for isomorphisms mapping p to q and viceversa. This implies that in order to find even a single non-trivial symmetry, one may need to traverse all pairs of variables.*
Also notice that we cannot detect phase-shift symmetries (and their composition with permutational symmetries), as positive-literal and negative-literal vertices are colored differently.

Despite its limitations, this reduction algorithm has historical relevance as it was the first to introduce fundamental elements now used by practical reductions. Of particular importance are:

- The modeling of variable by pairs of positive-literal and negative-literal vertices.

- The modeling of clauses by a vertex connected to respective literal vertices by edges.

- Connecting positive-literal and negative-literal vertices to enforce Boolean consistency.

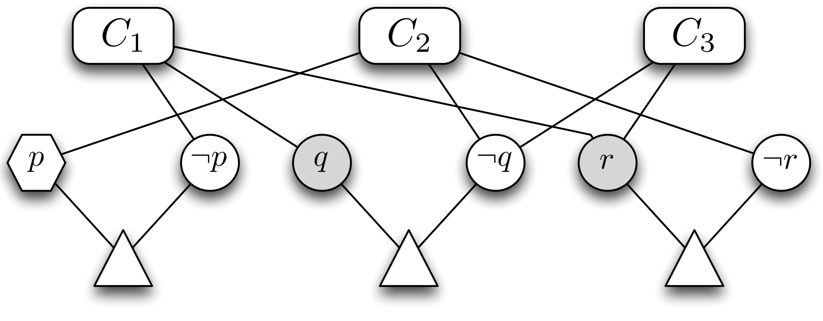
In [Crawford *et al.*, 1996] an alternative reduction algorithm is presented. It is the first one to reduce the problem of detecting symmetries of formulas to the problem of detecting automorphism of a graph.

Definition 6.9 (Crawford II reduction algorithm)**.** *Let φ be a CNF formula. The graph $G(\varphi)$ is constructed as follows:*

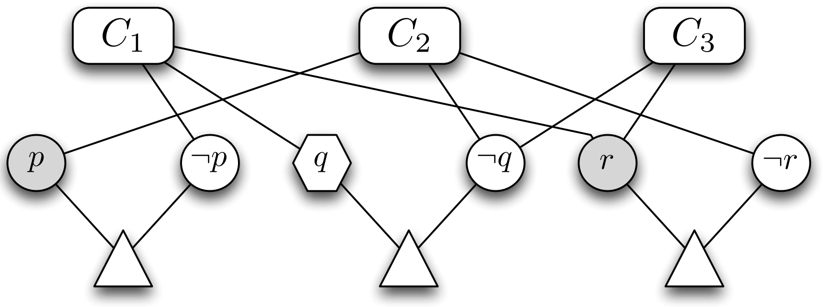
i) *For each propositional variable p in $Vars(\varphi)$:*

   a) *Add two vertices: one positive-literal vertex of color* 0*, one negative-literal vertex of color* 1.

ii) *For each non-binary clause in φ:*



(a) $G(\varphi, p)$



(b) $G(\varphi, q)$

Figure 6.5: Crawford I reduction algorithm example.

a) Add a vertex of color 2.

b) Add an edge between the clause vertex and each literal vertex corresponding to a literal occurring in it.

iii) For each binary clause in φ:

a) Add an edge between the two literals vertices corresponding to the literals occurring in the clause.

For a formula with V variables, $C_{>2}$ non-binary clauses and C_2 binary clauses, this construction produces a graph with 3 *colors and* $2V + C_{>2}$ *vertices.*

Example 6.6. *Consider the formula of Example 6.5. Figure 6.6 shows its associated colored graph $G(\varphi)$ constructed using the algorithm of Definition 6.9. This formula has only one non-trivial symmetry: $\sigma = (p\ \neg p)(q\ \neg r)(\neg q\ r)$. However, we cannot detect it with this reduction algorithm, because phase-shift symmetries (and their composition with permutational symmetries) are not detected.*

An important element introduced by this reduction is the modeling of binary clauses by one edge rather than by one clause vertex and two edges. As the graph automorphism algorithms tend to be more sensitive to the number of vertices of an input graph than to the number of edges, this could lead to great savings when the formula has many binary clauses. However, this reduction does not enforce Boolean consistency, i. e., there is no edge between positive-literal and negative-literal vertices, and therefore, this could lead to the detection of spurious symme-

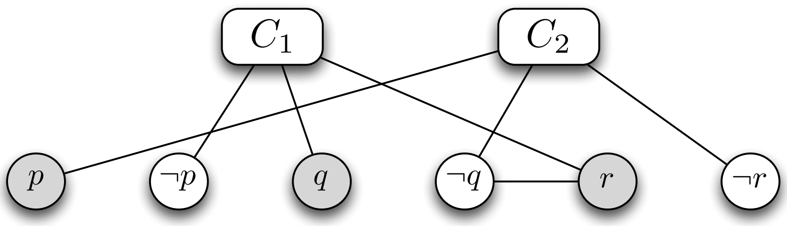


Figure 6.6: Crawford II reduction algorithm example.

tries, i. e., automorphisms of the graph that do not correspond to a symmetry of the formula.

Example 6.7. *Consider the formula $\varphi = (p \vee q)$. Figure 6.7 shows its associated graph $G(\varphi)$ constructed using the algorithm of Definition 6.9. The formula has two symmetries: i) the identity symmetry, and ii) the transposition $(p\ q)$. The graph $G(\varphi)$ has two positive-literal vertices, two negative-literal vertices and binary-clause edge linking the positive-literal vertices. Since no negative literals are used, the respective vertices are disconnected and can be mapped to each other even if the positive-literal vertices are fixed. This graph have four symmetries. One of them is the transposition $(\neg p\ \neg q)$ with p and q fixed which violates Boolean consistency.*

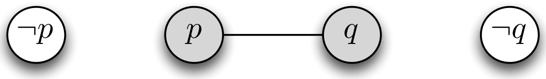


Figure 6.7: Boolean consistency violation using the Crawford II reduction algorithm.

Notice also that this reduction is unable to detect phase-shift symmetries due to the different coloring of the positive and negative literal vertices.

In [Aloul *et al.*, 2003b] three reduction algorithms (2xEDGEs, MIN3C and DAC02) are presented. These reductions improve on the previous ones and constitute the most effective reduction algorithms known so far.

Definition 6.10 (2xEDGES reduction algorithm)**.** *Let φ be a CNF formula. The graph $G(\varphi)$ is constructed as follows:*

i) *For each propositional variable p in $Vars(\varphi)$:*
   a) *Add two vertices of color 0: one for the positive literal and one for the negative literal.*
   b) *Add an edge between the positive-literal and negative-literal vertices.*
ii) *For each non-binary clause in φ:*
   a) *Add a vertex of color 1.*
   b) *Add an edge between the clause vertex and each literal vertex corresponding to a literal occurring in it.*

iii) For each binary clause in φ:

a) Add a double edge between the two literals vertices corresponding to the literals occurring in the clause.

For a formula with V variables, $C_{>2}$ non-binary clauses and C_2 binary clauses, this construction produces a graph with 2 colors and $2V + C_{>2}$ vertices.

Example 6.8. *Consider the formula of Example 6.5. Figure 6.8 shows its associated graph $G(\varphi)$ constructed using the algorithm of Definition 6.10. This construction is able to detect the one non-trivial symmetry, $\sigma = (p\ \neg p)(q\ \neg r)\ (\neg q\ r)$, of the formula.*

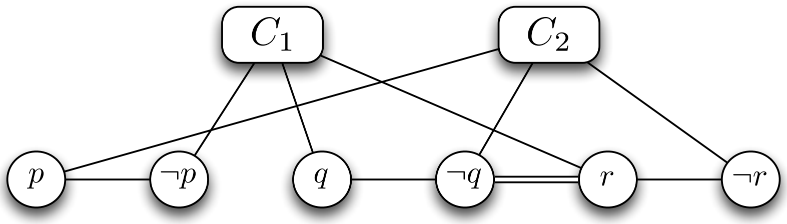


Figure 6.8: 2xEDGES reduction algorithm example.

This reduction improves the one proposed in [Crawford *et al.*, 1996] enforcing Boolean consistency (by adding an edge between the positive and negative literal vertices), and enabling the detection of phase-shift symmetries and their compositions with permutational symmetries (by coloring the positive-literal and negative-literal vertices with the same color). Also, the use of a double edge to model binary clauses prevents symmetries from mapping a Boolean consistency edge to a binary clause edge, which eliminates the detection of spurious symmetries.

However, a major drawback of this reduction algorithm is that current graph automorphism tools (like `Saucy`, `Bliss` and `Nauty`) cannot handle double edges, thus rendering this reduction impractical.

The MIN3C reduction algorithm fixes this issue.

Definition 6.11 (MIN3C reduction algorithm)**.** *Let φ be a CNF formula with V variables and C_2 binary clauses. The graph $G(\varphi)$ is constructed as follows:*

i) For each propositional variable p in $Vars(\varphi)$:

a) Add two vertices of color 0: one for the positive literal, and one for the negative literal.

b) If $V > C_2$: Add an edge between the positive-literal and negative-literal vertices.

c) If $V < C_2$: Add Boolean consistency vertex of color 2 and edges from this vertex to the positive-literal and negative-literal vertices.

ii) For each non-binary clause in φ:

a) Add a vertex of color 1.

b) Add an edge between the clause vertex and each literal vertex corresponding to literals occurring in it.

iii) For each binary clause in φ:

a) If $V > C_2$: Proceed as with a non-binary clause.

b) If $V < C_2$: Add an edge between the two literals vertices corresponding to literals occurring in the clause.

For a formula with V variables, $C_{>2}$ non-binary clauses and C_2 binary clauses, this construction produces a graph with at most 3 colors and $2V + C_{>2} + min\{C_2, V\}$ vertices.

Example 6.9. *Consider the formula of Example 6.5. Figure 6.9 shows its associated graph $G(\varphi)$ constructed using the algorithm of Definition 6.11. This construction is able to detect the non-trivial symmetry, $\sigma = (p\ \neg p)(q\ \neg r)(\neg q\ r)$, of the formula.*

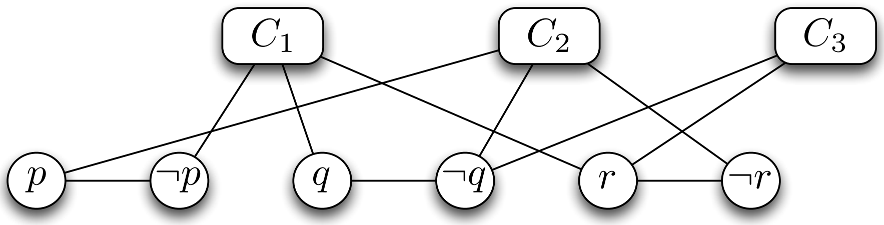


Figure 6.9: MIN3C reduction algorithm example.

The MIN3C algorithm yields a reduction that can be used with current graph automorphism tools and that do not detect spurious symmetries at the cost of generating larger graphs.

Finally, the DAC02 reduction algorithm makes no explicit distinction between the two types of edges (single edges and double edges), but represents the Boolean consistency and binary clause edges by single edges.

Definition 6.12 (DAC02 reduction algorithm)**.** *Let φ be a CNF formula. The graph $G(\varphi)$ is constructed as follows:*

i) For each propositional variable p in $Vars(\varphi)$:

a) Add two vertices of color 0: one for the positive literal, and one for the negative literal.

b) Add an edge between the positive-literal and negative-literal vertices.

ii) For each non-binary clause in φ:

a) Add a vertex of color 1.

b) Add an edge between the clause vertex and each literal vertex corresponding to a literal occurring in it.

iii) For each binary clause in φ:

a) Add an edge between the two literals vertices corresponding to the literals occurring in the clause.

For a formula with V variables and $C_{>2}$ non-binary clauses this construction produces a graph with 2 colors and $2V + C_{>2}$ vertices.

Example 6.10. *Consider the formula of Example 6.5. Figure 6.10 shows its associated graph $G(\varphi)$ constructed using the algorithm of Definition 6.12.*

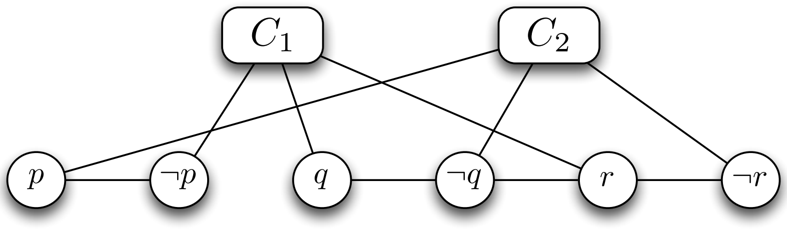


Figure 6.10: DAC02 reduction algorithm example.

Given that the algorithm makes no difference between Boolean consistency edges and binary clause edges, this could result in the detection of spurious symmetries. However, [Aloul *et al.*, 2003b] claims that identifying spurious symmetries is a trivial task and that the superior performance of this reduction algorithm justifies the extra effort of detecting and removing them.

Table 6.1 (adapted from [Aloul *et al.*, 2003b]) summarizes the main properties of the presented reduction algorithms.

Reduction	#Colors	#Nodes	Phase-shifts?	Spurious?	Practical?
Crawford I	5	$6V + C$	No	No	No
Crawford II	3	$2V + C_{>2}$	No	Yes	No
2xEDGES	2	$2V + C_{>2}$	Yes	No	No
MIN3C	3	$2V + C_{>2} + min\{C_2, V\}$	Yes	No	Yes
DAC02	2	$2V + C_{>2}$	Yes	Yes	Yes

Table 6.1: Reduction algorithms comparison.

The applicability of these techniques goes beyond SAT solving, and many extensions have been developed for a number of logics. For example, for QBF formulas, [Audemard *et al.*, 2004] presents a reduction algorithm for QBF formulas, that extends the DAC02 reduction algorithm. The main difference is that the prefix of the QBF formula must be considered because distinct literals can be symmetrical only if they belong to the same quantifier group. Therefore, the algorithm introduces additional colors to make a distinction between literals vertices whose variables belong to different quantifiers groups. For a QBF formula with k quantifiers, the algorithm introduces k different colors.

Example 6.11 ([Audemard *et al.*, 2004]). *Consider the QBF formula*

$$\varphi = \forall x_1, x_2 \exists x_3, x_4 (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \\ \wedge (x_2 \vee \neg x_3 \vee \neg x_4).$$

Figure 6.11 shows the associated graph. It has three colors, one for the non-binary clauses, one for the first universal group (white) and the last for the existential group (dark gray). This instance has two non-trivial symmetries, $(x_1\ x_2)$ and $(x_3\ x_4)$. The permutation $(x_1\ x_3)(x_2\ x_4)$, which is a symmetry of the matrix of φ, it is not a symmetry of φ since x_1 and x_2 are not in the same quantifier group.

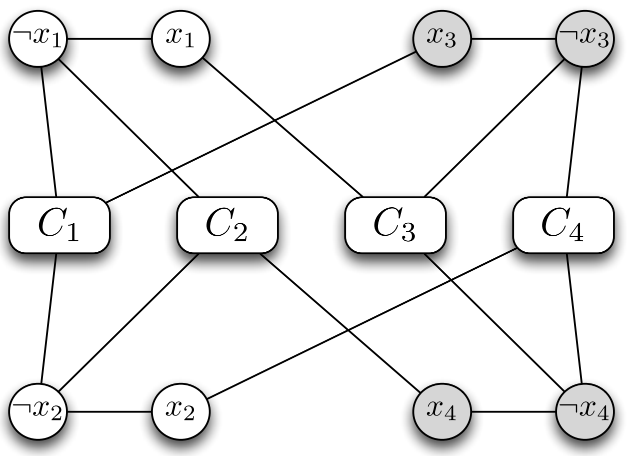


Figure 6.11: Graph representation of a QBF formula.

Graph reduction algorithms are also used in CSP. It has been shown that symmetry detection of variable and value symmetries can be done by detecting automorphisms of the microstructure graph of the problem as well as of the graph related to the intensional representation of each constraint [Puget, 2005].

6.2 FORMULA-BASED SYMMETRY DETECTION

Reducing the symmetry detection problem to the graph automorphism problem is not the only available approach. There exist many symmetry detection algorithms that deal directly with the formula.

We briefly describe two of such algorithms, one for detecting symmetries in propositional CNF formulas and another for detecting symmetries in SMT formulas.

One of the first known algorithms for detecting symmetries in propositional CNF formulas was introduced in [Benhamou and Sais, 1992; Benhamou and Sais, 1994]. The algorithm consists of two stages: the first stage partitions the literals into equivalence classes, while the second stage computes the symmetry using the equivalence classes computed previously.

In the first stage, the algorithm partitions the literals into equivalence classes based on the necessary conditions stated by the following proposition.

Proposition 6.1 (Necessary condition of symmetries). *If two literals (variables) l and l' are symmetrical in a set of clauses S then the number of occurrences of the variable l in S is the same as the number of occurrences of l' and there must be a correspondence between the length of clauses in which l occurs and the length of clauses in which l' occurs.*

Verification of the necessary conditions is an important step of the symmetry detection algorithm, since they reduce drastically the permutation search space by partitioning the set of literals into equivalence classes which are potential candidates for symmetry, i. e., two literals will be candidates to symmetry if they are in the same equivalence class.

In the second stage, the algorithm uses backtrack search to build a symmetry. It works by constructing a permutation, as a product of transpositions (i. e., cycles of size 2), by adding a transposition at each step and checking that the added transposition maintain invariant the set of clauses. For example, suppose the algorithm chooses two literals, x_0 and y_0, in the same equivalence class and builds a permutation $\sigma = (x_0\ y_0)\sigma'$, where σ' is a "sub-permutation" to be defined as the algorithm proceeds. Then, it tries to transform every clause containing x_0 into a clause containing y_0. To do so, it replaces every occurrence of x_0 by y_0 in every such clauses, and process every other variable in such clauses. This results in more transpositions added to the original permutation σ.

Example 6.12. *Consider two clauses $c = x_0 \vee x_1 \vee \ldots \vee x_n$ and $c' = y_0 \vee y_1 \vee \ldots \vee y_n$ of the same length such that x_0 occurs in c and y_0 occurs in c'. And consider the partial permutation $\sigma = (x_0\ y_0)$ To transform c into c', after replacing x_0 by y_0, each variable, $x_i, i \in \{1, \ldots, n\}$, must be linked to a variable $y_j, j \in \{1, \ldots, n\}$, this defines a new transposition $(x_i\ y_i)$ that we have to add to σ. If it is possible to substitute all variables, we say that c is transformed into c' with σ, i. e., $\sigma(c) = c'$.*

If all clauses containing x_0 have been transformed, then we consider the next transposition in σ and apply the same process. If there is no more transposition left to process, then σ is a symmetry of the formula. If given a transposition $(x_i\ y_i)$ we cannot relate a clause containing x_i, if $(x_i\ y_i) = (x_0\ y_0)$, i. e., it is the first transposition of the permutation, then σ is not a symmetry. Otherwise, the algorithm backtracks and tries to link x_i to another variable either on the same clause in which y_i occurs, or else it tries to relate the clause to another clause.

Notice that the algorithm works by finding one symmetry at a time, therefore mechanisms for enumerating just the necessary permutations must be implemented.

Also, the algorithm highly depends on the ordering in which the variables are permuted. Thus a clever strategy for choosing the variables is needed [Benhamou and Sais, 1992].

A similar algorithm is also used in finite model generation to detect symmetries in Skolemized first-order logic formulas with all its variables universally quantified [Audemard *et al.*, 2006].

For SMT formulas, [Déharbe *et al.*, 2011] presents a similar algorithm which is implemented in the SMT solver `VeriT` [Bouton *et al.*, 2009]. Given an SMT formula φ, the algorithm detects full groups of symmetries involving uninterpreted constants. Similarly to the previous algorithm, it works in two stages. In the first stage, the algorithm "guesses" possible full groups of permutations. To do so, it partitions uninterpreted constants in classes $\{c_0, \ldots, c_n\}$ such that all give the same values to some function $f(\varphi, c)$ that computes syntactic information that is unaffected by permutations, e. g., the number of occurrences of a constant c in φ, or the maximal depth of c within an atom of φ. Each identified class represents a set of constants which are possibly fully symmetric, i. e., we can permute any pair of constants in the class. In the second stage, the algorithm verifies that every identified class is fully symmetric. It does so in linear time thanks to the following result.

Proposition 6.2. *A formula φ is symmetric with respect to permutations of constants $c_0, \ldots, c_n$ if both permutations*

- *σ_{circ} such that $\sigma_{circ}(c_i) = c_{i-1}$ for $i \in \{1, \ldots, n\}$ and $\sigma_{circ}(c_0) = c_n$,*
- *σ_{swap} such that $\sigma_{swap}(c_0) = c_1$ and $\sigma_{swap}(c_1) = c_0$,*

are symmetries of φ.

Proposition 6.2 tells us that to verify that a formula is symmetric under the full group of symmetries of a set of constants $c_0, \ldots, c_n$ it suffices to check two permutations: the circular permutation $(c0\ \ldots\ c_n)$ and a permutation that swap to constants in the class, e. g., the permutation $(c_0\ c_1)$. As this permutations are generators of the full group of symmetries involving $c_0, \ldots, c_n$, if φ is symmetric with respect to this generators, then it will be symmetric with respect to any composition of them, and correspondingly, it will be symmetric with respect to the full group generated by the generators.

The algorithm works well in practice. However, this approach is rather limited as it is unable to detect arbitrary symmetries, only detecting full groups of symmetries of uninterpreted constants, and not considering symmetries involving other types of symbols, like predicate symbols, function symbols or interpreted symbols, and quantifiers.

7 SYMMETRY DETECTION FOR MODAL LOGICS

In Chapter 3 we developed the theoretical background for exploiting symmetries in modal logics. In this chapter we focus on how to detect symmetries in modal CNF formulas using graph-based techniques.

First we introduce the needed definitions and notation for the rest of the chapter (Section 7.1). Then we present a reduction algorithm for detecting *global symmetries* in modal CNF formulas for a wide range of modal logics (Section 7.2). Next we extend the algorithm for detecting *layered symmetries* in modal CNF formulas for modal logics having the tree model property (Section 7.3). Finally, we implement the algorithms for the basic modal logic and provide empirical results on several modal benchmarks (Section 7.4).

The results presented in this chapter were published in [Areces *et al.*, 2012; Areces and Orbe, 2013].

7.1 DEFINITIONS

In what follows we are going to work in the context of the coinductive framework defined in Section 3.2.

Unless stated otherwise, we work with modal CNF formulas considering them as sets of sets (see Section 3.1), although we might write them as usual for the sake of clarity. Also by $Sub(\varphi)$ we will denote the set of subformulas of φ as it is usually defined.

Definition 7.1 (Atoms of a formula). *Let φ be a modal CNF formula. By $At(\varphi)$ we denote the set of atoms occurring in φ regardless of the modal depth at which they occur. By $At(\varphi, n)$, $n \in \mathbb{N}$, we denote the set of atoms occurring in φ at modal depth n.*

Definition 7.2 (Top Clauses and Modal Clauses). *Let φ be a modal CNF formula. A* top clause *of φ is a clause occurring at modal depth 0. A* modal clause *of φ is a clause occurring in a modal literal.*

Example 7.1. *Consider the formula $\varphi = (\neg p \vee r) \wedge (q \vee r) \wedge (r \vee \Box(\neg p \vee q))$. Clauses $(\neg p \vee r)$, $(q \vee r)$ and $(r \vee \Box(\neg p \vee q))$ are top clauses, while the clause $(\neg p \vee q)$ is a modal clause.*

A key aspect of building a colored graph is to define how vertices are going to be colored. To do so, we define a *typing function*.

Definition 7.3 (Typing function). *Let $s : \mathsf{MOD} \times \{0,1\} \mapsto \mathbb{N} \backslash \{0,1\}$ be an injective function and let $t : Sub(\varphi) \mapsto \mathbb{N}$ be a partial function defined as:*

$$t(\psi) = \begin{cases} 1 & \text{if } \psi \text{ is a top clause .} \\ s(m,0) & \text{if } \psi = [m]C. \\ s(m,1) & \text{if } \psi = \neg[m]C. \end{cases}$$

The typing *function t assigns a numeric type to every clause (top or modal). For modal clauses, the type is based on the modality and the polarity of the modal literal in which it occurs.*

In what follows we call *global permutation (symmetry)* a permutation (symmetry) as defined by Definition 3.26, and *layered permutation (symmetry)* a permutation sequence as defined by Definition 3.34.

7.2 DETECTING GLOBAL SYMMETRIES

We now introduce a reduction algorithm for detecting *global symmetries* in modal CNF formulas. The algorithm is based on the MIN3C reduction algorithm for propositional CNF formulas presented in Section 6.1.2.

Like the MIN3C algorithm, our reduction algorithm constructs a colored graph from a modal CNF formula, such that the automorphism group of the graph is isomorphic to the symmetry group of the formula.

Unlike the MIN3C algorithm, our reduction algorithm uses two types of edges, and coloring is more complex as it has to deal with different modalities. Also, recall that the MIN3C algorithm adapts its representation of binary clauses and Boolean consistency by modeling them using vertices or edges for one or the other depending on the number of binary clauses and variables in the formula. Our algorithm fixes the representation, using a vertex and two edges to model binary clauses, and an edge between the positive-literal and negative-literal vertices to model Boolean consistency.

Definition 7.4 (Global reduction algorithm). *Let φ be a modal CNF formula and t a typing function. The graph $G(\varphi) = (V, E_1, E_2)$ is constructed as follows:*

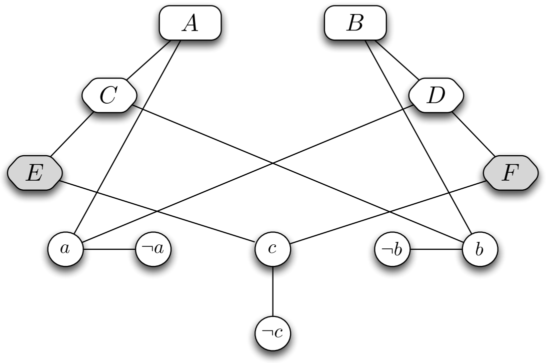
- *i) For each atom $a \in At(\varphi)$:*
  - *a) Add two vertices of color 0: one for the positive literal a and one for the negative literal $\neg a$.*
  - *b) Add an E_1-edge between the positive-literal and the negative-literal vertices to ensure Boolean consistency.*
- *ii) For each top clause C in φ:*
  - *a) Add a* clause *vertex of color $t(C)$.*
  - *b) For each atom literal l occurring in C, add an E_1-edge between the vertex for C and the vertex for l.*
  - *c) For each modal literal $[m]C'$ ($\neg[m]C'$) occurring in C:*
    - *i. Add a clause vertex of color $t([m]C')$ ($t(\neg[m]C')$) to represent the modal clause C'.*
    - *ii. Add an E_1-edge between the vertex of C and the vertex of C'.*
    - *iii. If m is indexed by an atom literal l then add an E_2-edge between the vertex of C' and the vertex of the indexing literal l.*
    - *iv. Repeat the process from point ii)b for each literal (atom or modal) occurring in C'.*

For a formula with A atoms, C top clauses, M modal clauses, and R modalities, this construction produces a graph with $2 + 2R$ colors and $(2|A| + C + M)$ vertices.

Example 7.2. *Consider the formula*

$$\varphi = (a \vee [m](b \vee \neg[m]c)) \wedge (b \vee [m](a \vee \neg[m]c)).$$

This formula has six clauses (2 at modal depth 0, 2 at modal depth 1 and 2 at modal depth 2) and three atoms (six literals). Figure 7.1 shows its associated colored graph $G(\varphi)$ (colors are represented by shapes and shades in the figure) constructed using the algorithm of Definition 7.4.



$$A = (a \vee [m](b \vee \neg[m]c))$$
$$B = (b \vee [m](a \vee \neg[m]c))$$
$$C = [m](b \vee \neg[m]c)$$
$$D = [m](a \vee \neg[m]c)$$
$$E = \neg[m]c$$
$$F = \neg[m]c$$

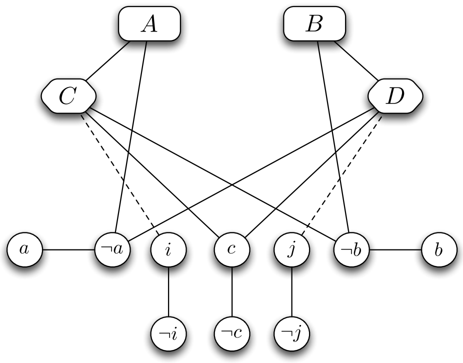
Figure 7.1: Global reduction algorithm example.

The graph has one non-trivial automorphism $\pi = (A\ B)(C\ D)(E\ F)(a\ b)(\neg a\ \neg b)$ which corresponds to the symmetry $\sigma = (a\ b)(\neg a\ \neg b)$ of φ.

Example 7.3. *Consider the formula*

$$\varphi = (\neg a \vee [@_i](\neg b \vee c)) \wedge (\neg b \vee [@_j](\neg a \vee c)).$$

This formulas has 2 modal clauses (C and D) that corresponds to modalities indexed by atoms ($@_i$ and $@_j$). Figure 7.2 shows its associated graph $G(\varphi)$ constructed using the algorithm of Definition 7.4.



$$A = (\neg a \vee [@_i](\neg b \vee c))$$
$$B = (\neg b \vee [@_j](\neg a \vee c))$$
$$C = [@_i](\neg b \vee c)$$
$$D = [@_j](\neg a \vee c)$$

Figure 7.2: Global reduction algorithm example using indexed modalities.

The graph has one non-trivial automorphism $\pi = (A\ B)(C\ D)(a\ b)(\neg a\ \neg b)(i\ j)(\neg i\ \neg j)$ that corresponds to the symmetry $\sigma = (a\ b)(\neg a\ \neg b)(i\ j)(\neg i\ \neg j)$ of φ.

Now let us prove that the reduction algorithm presented in Definition 7.4 is correct. First, let us introduce the needed definitions.

We need a mechanism to identify clauses occurring in a formula, therefore we define a *clause id* as follows.

Definition 7.5 (Clause Id). *Let φ be a modal CNF formula and C a clause occurring in it. The* clause id *of C is defined as $id(C) = \langle m, k, i \rangle$ where m is the modal depth at which the clause occurs, $k = t(\psi)$ is the type of the clause as returned by the* typing *function t and $i \in \mathbb{N}$ is different for each clause at the same modal depth. To simplify notation, in what follows we will assume that each clause C is labeled by its unique identifier $id(C) = \langle m, k, i \rangle$ and write $C_{m,k,i}$.*

Example 7.4. *Consider the formula*

$$\varphi = (a \vee [m_1](b \vee \neg[m_2]c)) \wedge (b \vee \neg[m_1](a \vee \neg[m_2]c)).$$

Assume $t([m_1]C) = 2$, $t(\neg[m_1]C) = 3$, $t([m_2]C) = 4$, and $t(\neg[m_2]C) = 5$. The multiset of clauses of φ is

$$\{(a \vee [m_1](b \vee \neg[m_2]c)), (b \vee \neg[m_1](a \vee \neg[m_2]c)), (b \vee \neg[m_2]c), (a \vee \neg[m_2]c), c, c\}.$$

The clause ids of the clauses are:

$$\begin{array}{rcl} (a \vee [m_1](b \vee \neg[m_2]c)) & \mapsto & \langle 0,1,1 \rangle \\ (b \vee \neg[m_1](a \vee \neg[m_2]c)) & \mapsto & \langle 0,1,2 \rangle \\ (b \vee \neg[m_2]c) & \mapsto & \langle 1,2,1 \rangle \\ (a \vee \neg[m_2]c) & \mapsto & \langle 1,3,2 \rangle \\ c & \mapsto & \langle 2,5,1 \rangle \\ c & \mapsto & \langle 2,5,2 \rangle \end{array}$$

By definition, a symmetry of a formula φ is a bijective function that maps literals to literals. It can naturally be "extended" to a function σ_{ext} that also maps each clause C to $\sigma(C)$.

Definition 7.6 (Extension of σ). *Let $\sigma : \mathsf{ALIT} \mapsto \mathsf{ALIT}$ be a symmetry of φ. Let $Cl(\varphi) = \{id(C) \mid C \text{ is a clause of } \varphi\}$. The* extension of σ *is a bijective function σ_{ext} over $\mathsf{ALIT} \cup Cl(\varphi)$ that maps literals to literals and clause id to clause id.*

The following properties of σ_{ext} are easy to verify.

Proposition 7.1. *Let φ be a modal CNF formula and σ a symmetry of φ. Then for the extension of σ the following holds:*

i) σ_{ext} is a bijective function.

ii) If $\sigma_{ext}(C_{m,k,i}) = C_{m',k',i'}$ then $m = m'$.

iii) If $\sigma_{ext}(C_{m,k,i}) = C_{m',k',i'}$ then $k = k'$.

iv) If $l \in Sub(C_{m,k,i})$ then $\sigma_{ext}(l) \in Sub(\sigma_{ext}(C_{m,k,i}))$.

v) σ_{ext} is a symmetry of φ.

Notice that the reduction algorithm of Definition 7.4 induces a mapping that associates to each literal and clause in the formula the corresponding vertex graph.

We are now ready to prove that the reduction algorithm of Definition 7.4 is correct. The first step is to show that each symmetry of the formula φ correspond to an automorphism of the colored graph $G(\varphi)$.

Proposition 7.2. *Let φ be a modal CNF formula, σ a symmetry of φ, $G(\varphi) = (V, E_1, E_2)$ the colored graph of φ as defined by Definition 7.4, and g the mapping induced by the construction of $G(\varphi)$. Then $\pi = g \circ \sigma_{ext}$ is an automorphism of $G(\varphi)$.*

Proof. To simplify notation let us assume that g is the identity function (i.e., we do not differentiate between a clause (or literal) and its associated vertex in the graph) and as a consequence $\pi = \sigma_{ext}$. Then π is an automorphism of $G(\varphi)$ if the following holds:

i) $(l, \neg l) \in E_1$ iff $(\pi(l), \pi(\neg l)) \in E_1$ for all $l \in V$.

   We have to consider the following cases:

   - σ is a permutational symmetry:
     $\underline{(\rightarrow)}$: Assume $\pi = \sigma_{ext} = (a\ b)(\neg a\ \neg b)$. Then $(\pi(a), \pi(\neg a)) = (b, \neg b) \in E_1$ by construction of $G(\varphi)$.

     $\underline{(\leftarrow)}$: $(a, \neg a) \in E_1$ by construction of $G(\varphi)$.

   - σ is a phase-shift symmetry:
     $\underline{(\rightarrow)}$: Assume $\pi = \sigma_{ext} = (a\ \neg a)$. Then $(\pi(a), \pi(\neg a)) = (\neg a, a)$, but given that $G(\varphi)$ is an undirected graph $(\neg a, a) = (a, \neg a) \in E_1$ by construction.

     $\underline{(\leftarrow)}$: $(a, \neg a) \in E_1$ by construction of $G(\varphi)$.

   - σ is a compositional symmetry: It follows directly from the previous two cases.

ii) $(l, C_{m,k,i}) \in E_1$ iff $(\pi(l), \pi(C_{m,k,i})) \in E_1$ for all $l, C_{m,k,i} \in V$.

   $\underline{(\rightarrow)}$: By construction, $(l, C_{m,k,i}) \in E_1$ only if $l \in C_{m,k,i}$. Then, given that σ_{ext} is a symmetry and by Proposition 7.1iv), we know that $\sigma_{ext}(l)$ and $\sigma_{ext}(C_{m,k,i})$ both occur in φ and $\sigma_{ext}(l) \in \sigma_{ext}(C_{m,k,i})$. Then, by construction, we have $(\pi(l), \pi(C_{m,k,i})) \in E_1$.

   $\underline{(\leftarrow)}$: It follows directly by construction of $G(\varphi)$.

iii) $(C_{m,k,i}, C_{m',k',i'}) \in E_1$ iff $(\pi(C_{m,k,i}), \pi(C_{m',k',i'})) \in E_1$ for all $C_{m,k,i}, C_{m',k',i'} \in V$.

   $\underline{(\rightarrow)}$: If $(C_{m,k,i}, C_{m',k',i'}) \in E_1$ we know that either $m < m'$ or $m > m'$. Assume $m < m'$ then $C_{m',k',i'}$ is a modal clause occurring in $C_{m,k,i}$. By Proposition 7.1iv), we have $\sigma_{ext}(C_{m',k',i'})$ is a modal clause occurring in $\sigma_{ext}(C_{m,k,i})$, and given that σ_{ext} is a symmetry of φ, $\sigma_{ext}(C_{m,k,i})$ and $\sigma_{ext}(C_{m',k',i'})$ both occur in φ, therefore, by construction, $\pi(C_{m,k,i}) \in V$ and, $\pi(C_{m',k',i'}) \in V$, therefore $(\pi(C_{m,k,i}), \pi(C_{m',k',i'})) \in E_1$.

   $\underline{(\leftarrow)}$: It follows directly by construction of $G(\varphi)$.

iv) $(l, C_{m,k,i}) \in E_2$ iff $(\pi(l), \pi(C_{m,k,i})) \in E_2$ for all $l, C_{m,k,i} \in V$.

<u>($\rightarrow$):</u> $(l, C_{m,k,i}) \in E_2$ if the modality of the modal clause $C_{m,k,i}$ is indexed by l. Given that σ_{ext} is a symmetry of φ, we know that $\sigma_{ext}(l) \in \varphi$ and $\sigma_{ext}(C_{m,k,i}) \in \varphi$ and that $\sigma_{ext}(l)$ index the modality of the modal clause $\sigma_{ext}(C_{m,k,i})$, therefore, by construction, $\pi(l) \in V$ and $\pi(C_{m,k,i}) \in V$ and $(\pi(l), \pi(C_{m,k,i})) \in E_2$.

<u>($\leftarrow$):</u> It follows directly by construction of $G(\varphi)$.

v) For every cycle $(x\ y) \in \pi$, x and y have the same color.

Follows from Proposition 7.1iii) and the fact that by construction different types of clauses are assigned different colors in the graph.

□

Next, we show that any automorphism of $G(\varphi)$ induces a symmetry of φ.

Proposition 7.3. *Let φ be a modal CNF formula, $G(\varphi) = (V, E_1, E_2)$ the colored graph of φ as defined by Definition 7.4, π an automorphism of $G(\varphi)$ and g the mapping induced by the construction of $G(\varphi)$. Then $\sigma_{ext} = g^{-1} \circ \pi$ is a symmetry of φ.*

Proof. Once more, assume g is the identity. To prove that σ_{ext} is a symmetry of φ, we have to prove the following properties:

i) σ_{ext} is a consistent permutation, i.e., $\sigma_{ext}(\neg l) = \neg\sigma_{ext}(l)$ for all $l \in \mathsf{ALIT}$.

By construction, Boolean consistency edges only connect literal nodes. Let $l_i \in V$ be a literal node. Then by construction we have $(l_i, \neg l_i) \in E_1$. Now assume $\pi(l_i) = l_j$ for $l_j \in V$. Given that π is an automorphism it must be the case that $(\pi(l_i), \pi(\neg l_i)) \in E_1$, and therefore $\pi(\neg l_i) = \neg l_j = \neg\pi(l_i)$, which implies $\sigma_{ext}(\neg l_i) = \neg\sigma_{ext}(l_i)$.

ii) If $C_{m,k,i} \in Sub(\varphi)$ then $\sigma_{ext}(C_{m,k,i}) \in Sub(\varphi)$.

By construction $C_{m,k,i} \in V$ implies $C_{m,k,i} \in Sub(\varphi)$. As π is an automorphism, $\pi(C_{m,k,i}) \in V$, therefore, $\pi(C_{m,k,i}) \in Sub(\varphi)$ which implies $\sigma_{ext}(C_{m,k,i}) \in Sub(\varphi)$.

iii) If $l \in Sub(\varphi)$ then $\sigma_{ext}(l) \in Sub(\varphi)$.

It follows by the same argument as in the previous case.

iv) If $\sigma_{ext}(C_{m,k,i}) = C_{m',k',i'}$ then $k = k'$.

It follows from the fact that π is a colored automorphism, mapping only nodes of the same color, and by construction, clauses of the same type are assigned the same color in the graph.

v) If $\sigma_{ext}(C_{m,k,i}) = C_{m',k',i'}$ then $m = m'$.

We prove this by induction on m, the modal depth at which a clause occurs in φ.

<u>Base Case: $m = 0$.</u> We have to prove that if $\sigma_{ext}(C_{0,k,i}) = C_{m',k',i'}$ then $m' = 0$. Assume $m' \neq 0$. Then, exists a clause $C_{n,s,j}$, with $n < m'$ such that, $C_{m',k',i'}$ is a modal clause occurring in it. By construction, we then have $(C_{n,s,j}, C_{m',k',i'}) \in E_1$. As π is an automorphism of $G(\varphi)$, we should have $(\pi(C_{n,s,j}), \pi(C_{m',k',i'})) = (\pi(C_{n,s,j}), C_{0,k,i}) \in E_1$, but by construction there is no such edge.

Inductive Step: $n < m \to m$. By construction of $G(\varphi)$ if $(C_{m,k,i}, C_{n,l,j}) \in E_1$ then $|m - n| = 1$. Now, assume $m \neq m'$. We know exists a clause $C_{(m'-1),s,j}$ such that $(C_{(m'-1),s,j}, C_{m',k',i'}) \in E_1$. Then, as π is an automorphism of $G(\varphi)$, it must be the case that $(\pi(C_{(m'-1),s,j}), \pi(C_{m',k',i'})) = (\pi(C_{(m'-1),s,j}), C_{m,k,i}) \in E_1$. By the inductive hypothesis we know $\pi(C_{(m'-1),s,j}) = C_{(m'-1),s,j'}$ and therefore, we have $(C_{(m'-1),s,j'}, C_{m,k,i}) \in E_1$. But then we get $|m - (m' - 1)| \geq 2$, which by construction cannot happen. Therefore $(C_{(m'-1),s,j'}, C_{m,k,i}) \notin E_1$, contradicting the fact that π is an automorphism of $G(\varphi)$.

vi) If l index a clause $C_{m,k,i}$ then $\sigma_{ext}(l)$ index the clause $\sigma_{ext}(C_{m,k,i})$.

If l indexes a clause $C_{m,k,i}$, then by construction $(l, C_{m,k,i}) \in E_2$. Given that π is an automorphism, $(\pi(l), \pi(C_{m,k,i})) \in E_2$, which implies $\sigma_{ext}(l)$ indexes the clause $\sigma_{ext}(C_{m,k,i})$.

We have proved that σ_{ext}, the extension of σ, obtained from an automorphism of the graph, is a symmetry of φ. To obtain the original symmetry σ we just take the restriction of σ_{ext} to atom literals. □

Finally we show that our construction is correct.

Theorem 7.1. *Let φ be a modal CNF formula and $G(\varphi) = (V, E_1, E_2)$ the colored graph constructed following the construction of Definition 7.4. Then every symmetry σ of φ corresponds one-to-one to an automorphism π of $G(\varphi)$.*

Proof. Immediate from Proposition 7.2 and 7.3. □

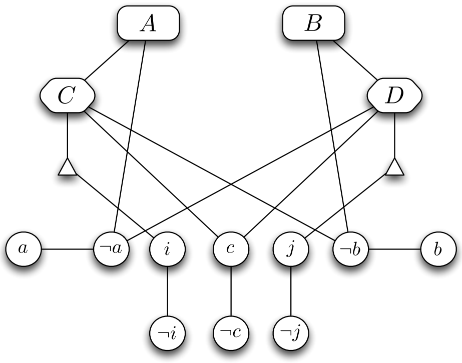
Theorem 7.1 ensures that the reduction algorithm is correct and, therefore, no spurious symmetry is detected.

Given that this construction is developed in the coinductive modal models framework and makes no assumption about the class of models on which we interpret the formulas, we can think of it as a *template* construction from which to derive reduction algorithms for concrete modal logics. For some modal logics, e. g., basic modal logic, the derivation is straightforward and we can use the algorithm as it is defined. However, for other modal logics, this has to be done carefully as they might require to constraint the graph even more. For example, for modal logics with modalities indexed by atoms (e. g., hybrid logics) practical implementations of the algorithm cannot be done using multiple types of edges as current graph automorphism tools can handle just one type of edges. However, this is simple to fix as we can replace every E_2-edge by an additional vertex (an *indexing* vertex) and two edges: one linking the indexing vertex and the modality vertex, and other linking the indexing vertex and the literal vertex.

Example 7.5. *Consider the formula φ of Example 7.3. Figure 7.3 shows the graph $G(\varphi)$ using additional vertices (triangle vertices) to model E_2-edges.*

7.3 DETECTING LAYERED SYMMETRIES

We now present an extension to the reduction algorithm presented in Definition 7.4 that enables the detection of layered symmetries for modal logics having the tree



$$A = (\neg a \lor [@_i](\neg b \lor c))$$
$$B = (\neg b \lor [@_j](\neg a \lor c))$$
$$C = [@_i](\neg b \lor c)$$
$$D = [@_j](\neg a \lor c)$$

Figure 7.3: Graph using additional vertices to represent E_2-edges.

model property. The key observation here is that, in these logics literals occurring at different modal depths are semantically different and, therefore, can be considered independently. Therefore, we can define a different permutation at each modal depth, thus defining a *permutation sequence* (see Definition 3.34).

Definition 7.7 (Layered reduction algorithm). *Let φ be a modal CNF formula of modal depth n and t a typing function. The graph $G(\varphi) = (V, E_1, E_2)$ is constructed as follows:*

i) For each atom $a \in At(\varphi, i)$ with $0 \leq i \leq n$:

  a) Add two vertices of color 0: one for the positive literal and one for the negative literal. Label the vertices a_i and $\neg a_i$ respectively (the i index mark the modal depth of the literal).

  b) Add an E_1-edge between the positive-literal and the negative-literal vertices to ensure Boolean consistency.

ii) For each top clause C in φ:

  a) Add a clause *vertex of color $t(C)$.*

  b) For each atom literal l occurring in C, add an E_1-edge between the vertex for C and the vertex for $l_{md(C)}$, where $md(C)$ is the modal depth at which C occurs.

  c) For each modal literal $[m]C'$ ($\neg[m]C'$) occurring in C:

    i. Add a clause vertex of color $t([m]C')$ ($t(\neg[m]C')$) to represent the modal clause C'.

    ii. Add an E_1-edge between the vertex of C and the vertex of C'.

    iii. If m is indexed by an atom literal l then add an E_2-edge between the vertex of C' and the vertex of the indexing literal labeled $l_{md(C')}$.

    iv. Repeat the process from point ii)b for each literal (atom or modal) occurring in C'.

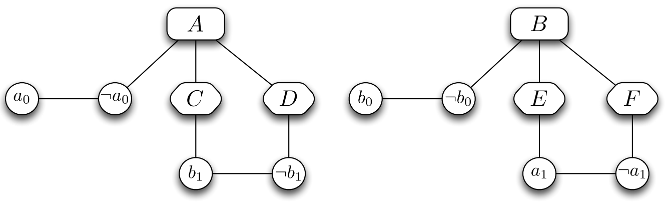
For a formula of modal depth n with A atoms, C top clauses, M modal clauses, and R modalities, this construction produces a graph with $2 + 2R$ colors and at most $(2|A| \times (n+1) + C + M)$ vertices.

This reduction algorithm differs from that of Definition 7.4 in the way literals are handled during the construction of $G(\varphi)$: if a literal occurs at different modal depths, the algorithm treats these occurrences independently adding distinct literal vertices. By doing so it incorporates the notion of layering introduced in Chapter 3.2.3.

Example 7.6. *Consider the following modal CNF formula*

$$\varphi = (\neg a \vee [m]b \vee [m]\neg b) \wedge (\neg b \vee [m]a \vee [m]\neg a).$$

Figure 7.4 shows it associated colored graph $G(\varphi)$ (colors are represented by shapes in the figure) constructed using the algorithm of Definition 7.7.



$$\begin{aligned} A &= (\neg a \vee [m]b \vee [m]\neg b) \\ B &= (\neg b \vee [m]a \vee [m]\neg a) \\ C &= [m]b \\ D &= [m]\neg b \\ E &= [m]a \\ F &= [m]\neg a \end{aligned}$$

Figure 7.4: Graph representation of φ using the layered reduction algorithm.

The automorphism group of the graph is generated by the following three generators (π_{Id} is the identity permutation):

$$\begin{array}{lrl} & \pi_1 = & (C\ D)(b_1\ \neg b_1) \\ Aut(G(\varphi))\ \textit{Generators:} & \pi_2 = & (E\ F)(a_1\ \neg a_1) \\ & \pi_3 = & (A\ B)(C\ E)(D\ F)(a_0\ b_0)(\neg a_0\ \neg b_0)(a_1\ b_1)(\neg a_1\ \neg b_1). \end{array}$$

These automorphism group generators correspond to the following symmetry group generators of φ (σ_{Id} is the identity permutation):

$$\begin{array}{ll} & \bar{\sigma}_1 = \langle \sigma_{Id}, (b\ \neg b) \rangle \\ Sym(\varphi)\ \textit{Generators:} & \bar{\sigma}_2 = \langle \sigma_{Id}, (a\ \neg a) \rangle \\ & \bar{\sigma}_3 = \langle (a\ b)(\neg a\ \neg b), (a\ b)(\neg a\ \neg b) \rangle. \end{array}$$

Proposition 7.4. *Let φ be a modal CNF formula and $G(\varphi) = (V, E_1, E_2)$ the colored graph obtained following the construction of Definition 7.7. Then every symmetry $\bar{\sigma}$ of φ corresponds one-to-one to an automorphism π of $G(\varphi)$.*

Proof. The result follows from a straightforward generalization of Properties 7.2 and 7.3. □

7.4 EXPERIMENTAL EVALUATION

In this section we implement the global and layered reduction algorithms for the basic modal logic, and test them on several modal benchmarks. For simplicity sake we restrict ourselves to the monomodal case, as generalization to the multimodal case is straightforward.

7.4.1 *Reduction Algorithms for the Basic Modal Logic*

In what follows, let $Vars(\varphi)$ denote the set of propositional variables occurring in φ, and $Vars(\varphi, n)$ denote the set of propositional variables occurring in φ at modal depth n. The global reduction algorithm is implemented as follows:

Definition 7.8 (Global reduction algorithm). *Let φ be a modal CNF formula. The graph $G(\varphi) = (V, E)$ is constructed as follows:*

i) *For each propositional variable $p \in Vars(\varphi)$:*
   a) *Add two vertices of color 0: one for the positive literal and one for the negative literal.*
   b) *Add an edge between the positive-literal and the negative-literal vertices to ensure Boolean consistency.*

ii) *For each top clause C in φ:*
   a) *Add a* clause *vertex of color 1.*
   b) *For each propositional literal l occurring in C, add an edge between the vertex for C and the vertex for l.*
   c) *For each modal literal $\Box C'$ ($\neg\Box C'$) occurring in C:*
      i. *Add a clause vertex of color 2 (color 3) to represent the modal clause C'.*
      ii. *Add an edge between the vertex for C and the vertex for C'.*
      iii. *Repeat the process from point ii)b for each literal (propositional or modal) occurring in C'.*

For a formula with V propositional variables, C top clauses, and M modal clauses, this construction produces a graph with 4 colors and $(2|V| + C + M)$ vertices.

The layered reduction algorithm is implemented as follows:

Definition 7.9 (Layered reduction algorithm). *Let φ be a modal CNF formula of modal depth n. The colored graph $G(\varphi) = (V, E)$ is constructed as follows:*

i) *For each propositional variable $p \in Vars(\varphi, i)$ with $0 \leq i \leq n$:*
   a) *Add two vertices of color 0: one for the positive-literal, labeled p_i, and one for the negative literal, labeled $\neg p_i$.*
   b) *Add an edge between the positive-literal and the negative-literal vertices to ensure Boolean consistency.*

ii) *For each top clause C in φ:*
   a) *Add a* clause *vertex of color 1.*
   b) *For each propositional literal l occurring in C, add an edge between the vertex for C and the vertex labeled as $l_{md(C)}$.*
   c) *For each modal literal $\Box C'$ ($\neg\Box C'$) occurring in C:*
      i. *Add a clause vertex of color 2 (color 3) to represent C'.*

ii. Add an edge between the vertex for C and the vertex for C'.

iii. Repeat the process from point ii)b for each literal (propositional or modal) occurring in C'.

For a formula of modal depth n with V propositional variables, C top clauses, and M modal clauses this construction produces a graph with 4 *colors and at most* $2|V| \times (n+1) + C + M$ *vertices.*

Notice that both reduction algorithms are almost identical to the general algorithms of Definitions 7.4 and 7.7, but simpler, as these algorithms do not need a typing function. We now present empirical results about how often symmetries appear in modal benchmarks and how hard it is to actually find them.

7.4.2 *Implementation*

We implemented both reduction algorithms in the tool `sy4ncl`[1]. `sy4ncl` is a command line tool, written in Haskell, that takes a basic modal logic formula in the `intohylo` format, builds the selected graph (for global or layered detection) and outputs it in `Bliss` or `Saucy` format. It also outputs a mapping from vertices to literals and statistics about the graph.

For each formula in our benchmarks we generate the corresponding graph using `sy4ncl`. Then we search for symmetries using the graph automorphism tool Bliss [Junttila and Kaski, 2007]. Bliss takes a graph specification and returns a set of generators for the automorphism group of the graph. If the graphs has non-trivial automorphisms, we reconstruct the symmetries of the formula from them using the mapping generated by `sy4ncl`.

From now on we call *global (layered) detection algorithm* (or just *global (layered) detection*) to the combination of the global (layered) reduction algorithm and the detection of automorphisms using a graph automorphism tool.

Example 7.7. *Consider the formula* $\varphi = (p \vee \neg\Box p) \wedge (q \vee \neg\Box q)$*. Figure 7.5a) shows its representation in the* intohylo *format. Figure 7.5b) shows the content of the mapping file generated by* `sy4ncl`*. This file maps literals in the formula to vertices in the graph. Figure 7.5c) shows the graph generated by* `sy4ncl` *in* `Bliss` *format. Figure 7.5d) shows statistics about the graph. Figure 7.5e) shows the output of* `Bliss` *for the graph corresponding to* φ*. It shows that* `Bliss` *found one non-trivial generator. Using this generator, and the mapping file we can reconstruct the symmetry of the formula* $(p\ q)(\neg p\ \neg q)$*.*

7.4.3 *Benchmarks*

We use two test sets (one structured, one random) for empirical testing. The structured test set is made of 4492 instances: 378 instances were extracted from the Logics Workbench Benchmark for K (LWB_K) [Balsiger *et al.*, 2000] (distributed in 9 problem classes) and 4113 instances from the QBF-LIB benchmarks [Giunchiglia *et al.*, 2001] (distributed in 23 problem classes). Problems from the QBF-LIB benchmarks were translated to the basic modal logic using the `qbf2ml`[2] tool. We use two

1 Download from: `http://cs.famaf.unc.edu.ar/~ezequiel/resource/sy4ncl`
2 Download from:`http://cs.famaf.unc.edu.ar/~ezequiel/resource/qbf2ml`

```
begin
P1 v -[R1]P1;
P2 v -[R1]P2
end
```

a) intohylo Format

```
<md> <node_id> <lit>
0 2 6
0 1 2
0 -1 3
0 -2 7
```

b) Mapping File

```
p edge 8 8
n 1 1
n 2 4
n 3 4
n 4 3
n 5 1
n 6 4
n 7 4
n 8 3
e 2 3
e 1 2
e 1 4
e 4 2
e 6 7
e 5 6
e 5 8
e 8 6
```

c) Bliss Graph

```
Computation time: 0.00000 sec
Color count:[2,0,2,4]
|Nodes|: 8
|Edges|: 8
```

d) Graph Statistics

```
Generator: (1,5)(2,6)(3,7)(4,8)
Nodes:           3
Leaf nodes:      3
Bad nodes:       0
Canrep updates: 1
Generators:      1
Max level:       1
|Aut|:           2
Total time: 0.00 seconds
```

e) Bliss Output

Figure 7.5: Output of the sy4ncl tool.

different translations from QBF to the basic modal logic: *Collapse*1 and *Collapse*2. These translations are both variations of Ladner's translation [Ladner, 1977], that reduce the modal depth of the resulting modal formula yielding smaller formulas than Ladner's translation. The difference between Collapse1 and Collapse2 is that the former uses auxiliary propositional symbols, while the latter does not. We refer the reader to Appendix B for a detailed description of these translations.

The random test set contains 19000 formulas generated using hGen [Areces and Heguiabehere, 2003]. To generate the formulas, we fix the maximum modal depth of the formulas (D) and the clause size to 3. Then, instances are distributed in 10 sets. For each set we fix the number of propositional variables (N) (from 10 to 500) and vary the number of clauses (L) to get different values of the ratio clauses-to-variables (L/N). This ratio is a good indicator of the satisfiability of the formula: formulas with smaller value of L/N are more likely to be satisfiable, whilst formulas with greater values of L/N are often unsatisfiable. Each set contain 100 instances for 19 different values of the ratio L/N (from 0.2 to 35).

All tests were ran on an Intel Core i7 2.93GHz with 16GB of RAM with a timeout of 120 seconds for both graph creation and symmetry detection.

7.4.4 *Results*

Table 7.1 summarizes the results for the LWB_K test set using both detection algorithms (global and layered). Columns *#In*, *#To* and *#Sy* are the number of instances in the test set, the number of instances that timeouted and the number of instances with at least one symmetry, respectively. Columns T_G and T_S are the time in seconds to create the graph and the total time to search for automorphism for all the instances, respectively.

	#In	*#To*	*#Sy*	T_G	T_S
Global	378	0	135	9.83	1.18
Layered	378	0	208	9.80	1.80

Table 7.1: Symmetries in the LWB_K test set.

The table shows that many symmetric instances exists in the LWB_K test set and that the time required to compute the symmetries (graph time + search time) is negligible. It also confirms our claim that by using the layered detection algorithm we could detect more symmetries. Indeed, using layered detection, we find 73 more symmetric instances than with the global detection algorithm.

Table 7.2 shows detailed results for the LWB_K test set. Column *M* is the median number of detected generators in each problem class. This value gives an approximated idea of how symmetric are the instances in a problem class. Problem classes ending in "`_n`" contain satisfiable problems, while those ending in "`_p`" contain unsatisfiable problems. The table shows that the LWB_K test set presents a behavior that matches our expectations: the existence of symmetries is driven by the codification used in each problem class. Many problem classes (`k_branch`, `k_path`, `k_grz`, `k_ph` and `k_poly`) exhibit many symmetric instances, while others exhibit none (`k_d4`, `k_dum`, `k_t4p`) or few symmetric instances (`k_lin`). Also notice the effect of using layered detection. For some problem classes (`k_branch`, `k_ph` and `k_poly`) both detection algorithms yield practically the same results, with the layered version detecting few more symmetries per instances than the global version. However, in the `k_path` and `k_grz` classes, differences are more evident using layered detection because we find symmetries in all instances in those classes, while with global detection we do not.

We now turn our attention to the QBF-LIB test set. Table 7.3 summarizes the results obtained on the QBF-LIB test set using the global and the layered detection algorithms for both translations. Tables 7.4 and 7.5 shows detailed information for each problem class using the Collapse1 and the Collapse2 translations respectively.

These tables highlight some interesting facts. First, Table 7.3 shows that the translated QBF-LIB benchmark is highly symmetric: using the Collapse2 translation and global symmetry detection we get that 65% of the total instances have one or more symmetries and 18 of 23 classes have symmetric instances. These numbers grow up to a 94% of the total instances having one or more symmetries and 23 of 23 classes having symmetric instances when using the Collapse1 translation and the layered symmetry detection.

		Global		Layered	
Class	*#In*	*#Sy*	*M*	*#Sy*	*M*
`k_branch_n`	21	21	12	21	12
`k_branch_p`	21	21	11	21	11
`k_d4_n`	21	0	-	0	-
`k_d4_p`	21	0	-	0	-
`k_dum_n`	21	0	-	0	-
`k_dum_p`	21	0	-	0	-
`k_grz_n`	21	1	1	21	5
`k_grz_p`	21	1	1	21	3
`k_lin_n`	21	0	-	0	-
`k_lin_p`	21	1	1	1	1
`k_path_n`	21	4	1.5	21	36
`k_path_p`	21	5	2	21	33
`k_ph_n`	21	18	1	18	1
`k_ph_p`	21	21	1	21	1
`k_poly_n`	21	21	16	21	19
`k_poly_p`	21	21	16	21	17
`k_t4p_n`	21	0	-	0	-
`k_t4p_p`	21	0	-	0	-

Table 7.2: Symmetries in the LWB_K test set detailed by problem class.

		Collapse1				Collapse2			
	#In	*#To*	*#Sy*	T_G	T_S	*#To*	*#Sy*	T_G	T_S
Global	4113	22	2678	7330.55	57695.41	24	2676	6310.18	44022.18
Layered	4113	20	3874	7596.48	65012.79	20	2680	6342.06	39370.36

Table 7.3: Symmetries in the QBF-LIB test set.

		Global					Layered				
Class	$\#In$	$\#To$	$\#Sy$	M	T_G	T_S	$\#To$	$\#Sy$	M	T_G	T_S
Ansotegui	8	8	0	-	38.90	44.22	8	8	381.50	39.72	34.86
Ayari	16	16	7	17.00	23.59	26.63	16	16	683.50	22.83	79.66
Basler	46	46	46	46.00	59.10	179.67	46	46	2027.50	61.71	664.45
Biere	80	80	0	-	59.06	26.26	80	76	63.00	64.81	27.09
Castellini	169	169	168	10.50	82.53	16.49	169	169	26.00	87.63	21.08
Egly-Seidl	137	137	125	10.00	329.68	2615.57	137	137	4506.00	349.17	4172.66
Faber-Leone-Maratea-Ricca	319	319	94	4.00	611.90	2785.79	319	319	1601.00	638.67	5293.96
Gent-Rowley	39	39	39	102.00	69.19	97.50	39	39	254.00	78.85	47.08
Herbstritt	65	65	65	566.00	32.63	401.95	65	65	566.00	29.37	423.36
Katz	10	10	0	-	14.37	35.22	10	10	529.00	11.55	81.84
Letombe	63	63	0	-	160.47	636.01	63	63	2499.00	172.13	1102.59
Letz	14	14	14	1.00	3.70	1.84	14	14	625.00	3.99	11.04
Ling	8	8	8	6.50	0.91	0.26	8	8	54.50	0.92	0.37
Mangassarian-Veneris	83	83	57	14.00	338.36	56.56	83	79	17.00	351.85	63.78
Messinger	20	20	20	17.00	20.19	22.97	20	20	50.00	18.73	29.17
Miller-Marin	466	464	159	27.00	1749.39	3593.50	466	464	323.00	1830.63	2988.92
Mneimneh-Sakallah	4	4	0	-	2.48	0.91	4	4	911.50	3.40	6.06
Narizzano	1621	1601	1601	900.00	2599.54	38435.65	1601	1601	1124.00	2664.89	39821.14
Pan	72	72	46	2.00	197.30	560.67	72	72	466.50	191.23	421.88
Rintanen	39	39	29	13.00	33.67	43.43	39	39	51.00	34.98	47.43
Scholl-Becker	60	60	38	3.00	53.00	21.75	60	50	123.00	51.90	206.43
Tacchella	693	693	107	4.00	689.91	7959.29	693	493	1599.00	724.75	7879.51
Wintersteiger	81	81	55	62.00	160.68	133.27	81	81	4127.00	162.78	1588.43

Table 7.4: Symmetries in the QBF-LIB test set translated using the Collapse1 translation.

		Global					Layered				
Class	$\#In$	$\#To$	$\#Sy$	M	T_G	T_S	$\#To$	$\#Sy$	M	T_G	T_S
Ansotegui	8	8	0	-	42.80	38.80	8	0	-	38.17	8.91
Ayari	16	16	7	17.00	18.84	15.75	16	7	17.00	20.51	12.74
Basler	46	46	46	46.00	43.49	130.45	46	46	46.00	43.31	30.71
Biere	80	80	0	-	56.16	25.87	80	0	-	62.93	13.56
Castellini	169	169	168	10.50	77.35	16.39	169	168	10.50	78.25	17.99
Egly-Seidl	137	137	125	10.00	228.99	317.20	137	125	10.00	232.64	50.44
Faber-Leone-Maratea-Ricca	319	319	94	4.00	484.39	153.91	319	94	4.00	476.73	87.22
Gent-Rowley	39	38	38	102.00	71.24	219.65	39	39	104.00	69.43	26.49
Herbstritt	65	65	65	566.00	26.77	380.67	65	65	566.00	29.66	420.98
Katz	10	10	0	-	10.96	14.92	10	0	-	11.36	2.93
Letombe	63	63	0	-	128.72	97.59	63	0	-	134.47	26.05
Letz	14	14	14	1.00	2.48	1.21	14	14	1.00	2.78	0.73
Ling	8	8	8	6.50	0.81	0.31	8	8	6.50	0.91	0.38
Mangassarian-Veneris	83	83	57	14.00	283.08	53.94	83	57	14.00	299.60	61.28
Messinger	20	20	20	17.00	17.56	24.39	20	20	17.00	17.55	25.85
Miller-Marin	466	463	158	27.00	1657.67	3303.22	466	161	27.00	1663.61	316.81
Mneimneh-Sakallah	4	4	0	-	2.19	0.52	4	0	-	2.66	0.35
Narizzano	1621	1601	1601	900.00	2320.41	38085.44	1601	1601	900.00	2307.89	38021.60
Pan	72	72	46	2.00	182.40	359.54	72	46	2.00	175.00	51.17
Rintanen	39	39	29	13.00	26.29	44.68	39	29	13.00	26.63	47.66
Scholl-Becker	60	60	38	3.00	45.11	14.55	60	38	3.00	46.26	13.02
Tacchella	693	693	107	4.00	468.61	656.98	693	107	4.00	483.23	86.15
Wintersteiger	81	81	55	62.00	113.88	66.20	81	55	62.00	118.50	47.34

Table 7.5: Symmetries in the QBF-LIB test set translated using the Collapse2 translation.

The tables also highlight how sensitive is symmetry detection to the codification of the formulas. Table 7.3 shows that for formulas translated using the Collapse1 translation we find more symmetries using layered detection than global detection. In fact, Table 7.4 shows that using layered detection we detect symmetries in 5 more classes than with global detection, and that in general layered detection detects more symmetries per instance. On the other hand, for formulas translated using the Collapse2 translation both algorithms detects practically the same number of symmetric instances. This can be explained by the fact that the Collapse1 translation uses auxiliary variables (to "mark" the levels in the resulting tree models) while the Collapse2 translation does not. Therefore, Collapse1 formulas have more propositional literals, at each modal depth, that might be permuted. Table 7.3 also shows that, in terms of efficiency, detecting layered symmetries is harder than detecting global symmetries for Collapse1 formulas, in particular in what respect to the search of automorphisms.

Figures 7.6 and 7.7 show scatter plots of the graph construction time and automorphisms search time for both detection algorithms on instances translated using the Collapse1 translation, respectively. The x axis gives the times of global detection, whereas the y axis gives the times of layered detection. Each point represents an instance and its horizontal and vertical coordinates represent the time necessary to build the graph (or search for automorphisms) in seconds. Points below the diagonal represent instances where layered detection outperforms global detection, while points above the diagonal represent instances where global detection outperforms layered detection. Points on the rightmost and topmost edges represent timeout. Notice that a logscale is used, so that gain or degradation to the far right and far top are exponentially more relevant.

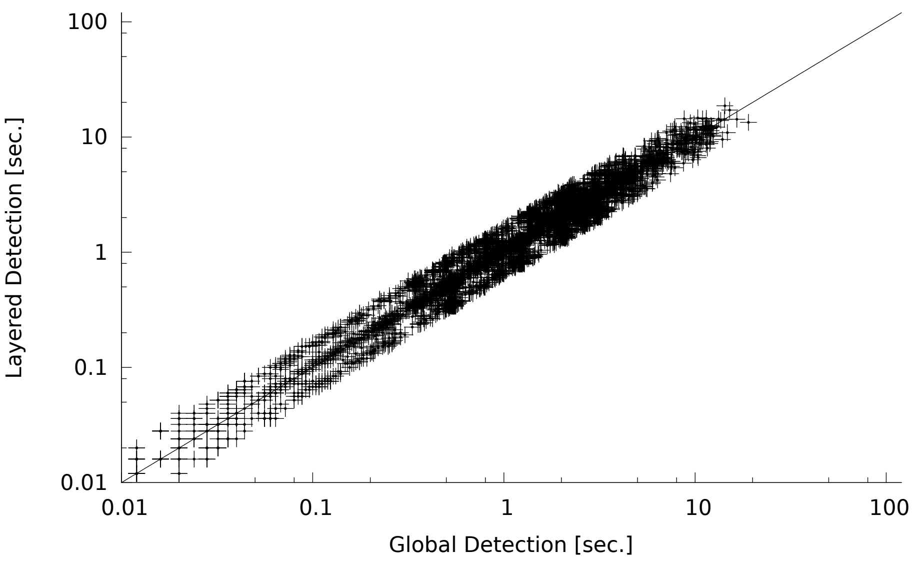


Figure 7.6: Graph construction time on Collapse1-translated instances.

Figure 7.6 shows that in general there are no important differences in the times required to construct the graphs for both algorithms. Figure 7.7, on the other hand, shows that searching for automorphisms is harder for the layered detection algorithm than for the global detection algorithm. This is in line with the fact that

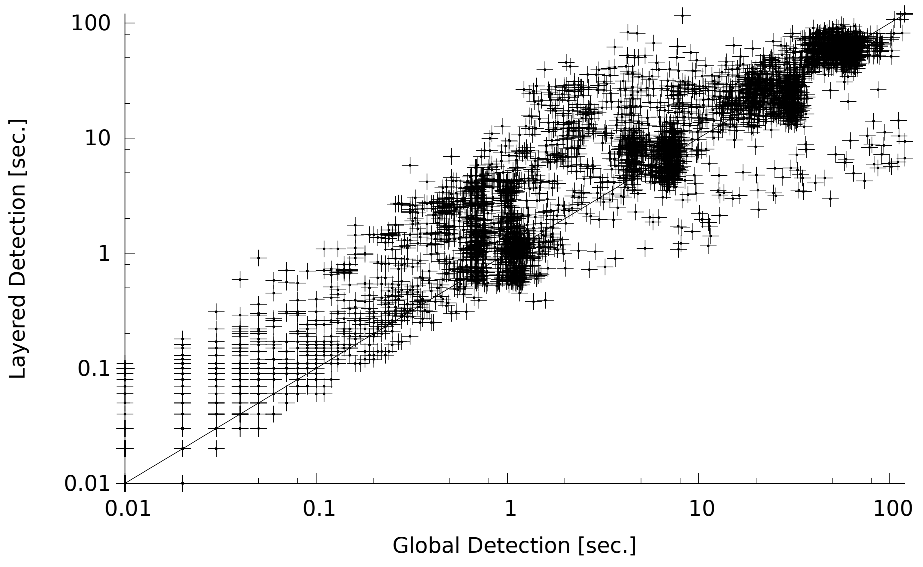


Figure 7.7: Automorphisms search time on Collapse1-translated instances.

detecting layered symmetries involves larger graphs which directly affects the performance of the automorphism graph tools. However, Table 7.3 shows a different behavior for Collapse2 instances. In fact, in this case we get better performance for layered detection than for global detection. Figures 7.8 and 7.9 show scatter plots of the graph construction time and automorphisms search time for both detection algorithms on instances translated using the Collapse2 translation, respectively.

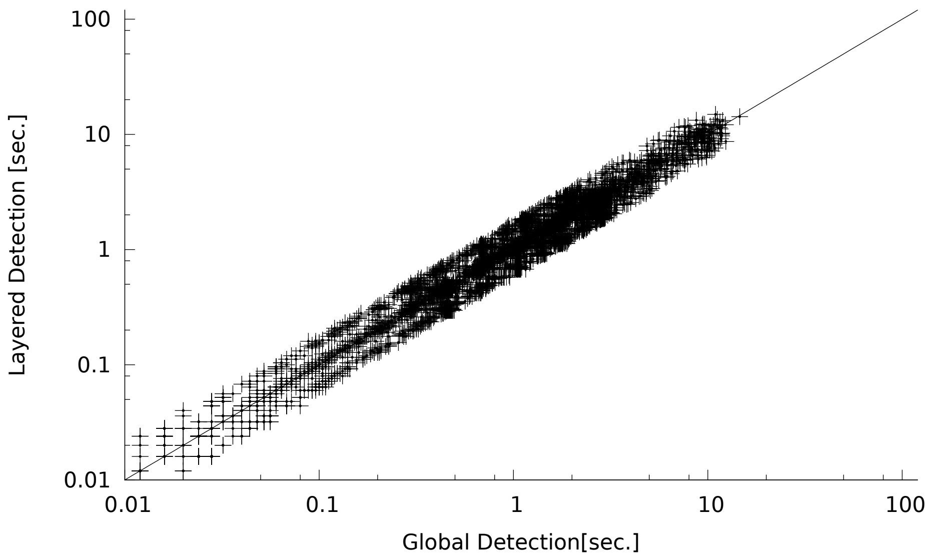


Figure 7.8: Graph construction time on Collapse2-translated instances.

Figure 7.8 shows that, similar to what happens with Collapse1 instances, there are no important differences between the algorithms. However, in Figure 7.9, we observe that for most of the instances, searching layered symmetries is easier than searching global symmetries. At first glance, this seems contradictory, as layered graphs are bigger than global graphs, and we would expect a similar behavior than

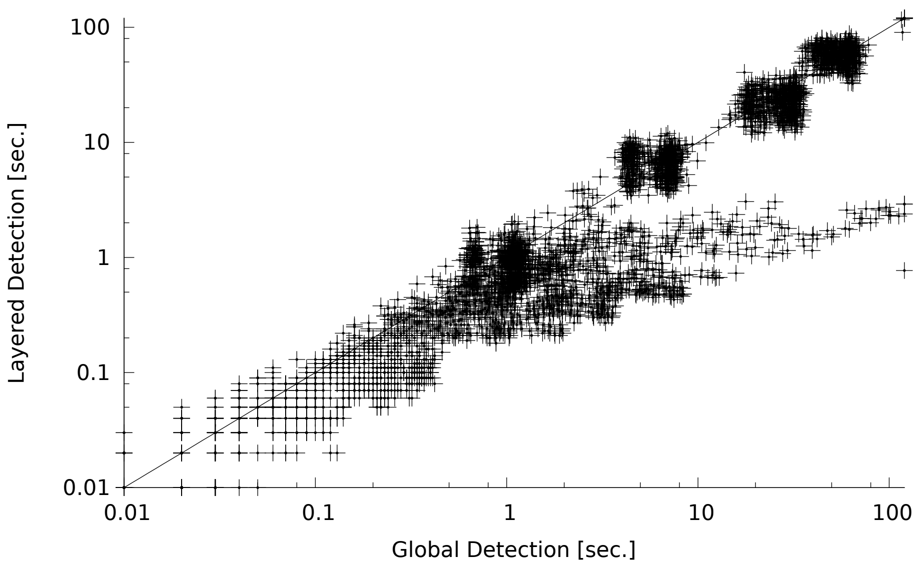


Figure 7.9: Automorphisms search time on Collapse1-translated instances.

for the Collapse1 case. However, this can be explained by the fact that for Collapse2, layered graphs are *sparser* than global graphs. Recall that a graph is sparse if the average degree of the vertices is much smaller than the number of vertices. In our case, for an instance translated with the Collapse2 translation, its layered graph contains more vertices than its global graph, but approximately the same number of edges. This lowers the average degree of the vertices making the layered graph sparser than the global graph. As already mentioned in Chapter 6.1.1, the performance of the graph automorphism tools is directly affected by the sparsity of the graphs.

Finally we test symmetry detection in the random testbed to see how often symmetries occur in random instances. Figures 7.10 and 7.11 show the percentage of symmetric instances for each value of the ratio L/N using global and layered detection respectively. They show that for small values of L/N it is easy to find symmetric instances even in randomly generated formulas. As we increase the value of the ratio, symmetric instances rapidly diminish. Again this coincides with expectations: large values of L/N results from a high number of clauses in the instances, reducing the possibility of symmetries. They also show that using layered detection we find more symmetries than using global detection.

7.5 SUMMARY

In this chapter we have presented two reduction algorithms for detecting symmetries in modal formulas, one for detecting *global* symmetries and another for detecting *layered* symmetries. We proved that the algorithms are correct, and therefore that every detected automorphism of the graph corresponds to a symmetry of the formula from which we built the graph. As the algorithms were presented in the context of coinductive models, they provide us with a template from which to derive concrete implementations for concrete modal logics.

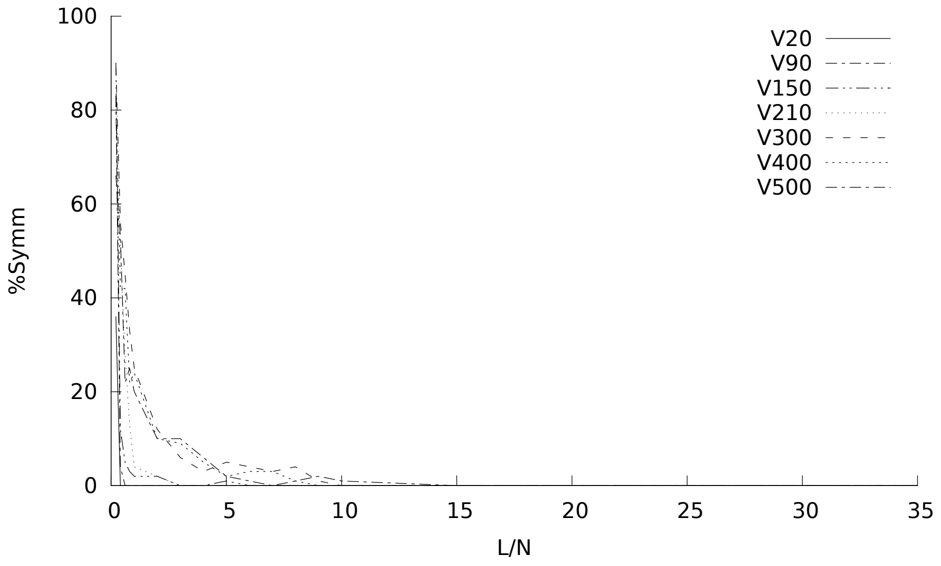


Figure 7.10: Percentage of symmetric instances in random formulas using global detection.

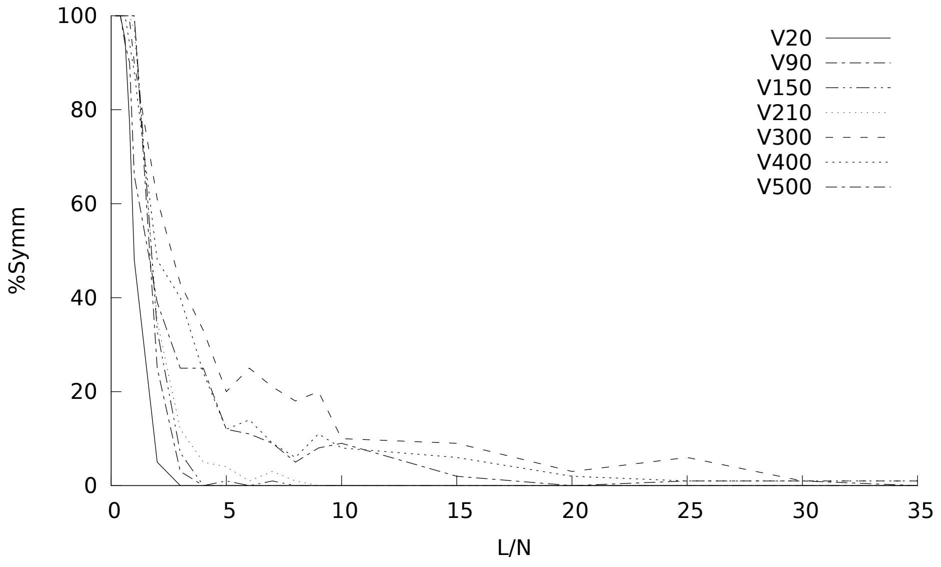


Figure 7.11: Percentage of symmetric instances in random formulas using layered detection.

We then implemented the two reduction algorithms for the basic modal logic and tested them in modal benchmarks. Experimental results showed that symmetries do exists in modal benchmarks. As expected, the presence of symmetries highly depends on the problem codification. Results also showed that detecting symmetries is relatively "cheap" even for large graphs.

It remains to see how to profit from the presence of symmetries in modal formulas.

8 SYMMETRY DETECTION FOR SATISFIABILITY MODULO THEORIES

In the previous chapter we presented a graph-based technique for detecting symmetries in modal formulas. In this chapter we extend this technique to Satisfiability Modulo Theories (SMT) formulas.

First we introduce the needed definitions and notation for the rest of the chapter (Section 8.1). Then we present the reduction algorithm for detecting symmetries in SMT formulas and prove its correctness (Section 8.2). Finally we empirically evaluate the algorithm on several benchmarks from the SMT-LIB (Section 8.3).

The results presented in this chapter were published in [Areces *et al.*, 2013].

8.1 DEFINITIONS

Let us start by defining the language that we are going to use. In Chapter 4 we introduced SMT using a standard first-order language. Nevertheless, problems in SMT (in particular those in the SMT-LIB) are often expressed in many-sorted first-order languages. In contrast to standard first-order languages, in many-sorted languages every term is *typed* (or *sorted*), and each sort is denoted by a sort symbol. Let us start by defining a *many-sorted signature*.

Definition 8.1 (Many-sorted Signature)**.** *A* many-sorted signature *(or* signature*) is a tuple $\Sigma = \langle \mathcal{S}, \mathcal{V}, \mathcal{F}, ar, r \rangle$ where*

- *$\mathcal{S}$ is a countable non-empty set of disjoint sorts (or types) containing the sort `Bool`.*
- *$\mathcal{V}$ is the (countable) union of disjoint countable sets $\mathcal{V}_\tau$ of variables of sort $\tau \in \mathcal{S}$.*
- *$\mathcal{F}$ is a countably infinite set of function symbols, containing the symbols $=, \wedge, \neg, \forall_\tau$, and $\exists_\tau$ for all $\tau \in \mathcal{S}$.*
- *$ar : \mathcal{F} \mapsto \mathbb{N}$ is a total function called the* arity*, and $r : \mathcal{F} \mapsto S^n \times S$ is a function, called the* rank*, such that:*
  - *$r(\wedge) = (\texttt{Bool}, \texttt{Bool}, \texttt{Bool})$ and $ar(\wedge) = 2$.*
  - *$r(\neg) = (\texttt{Bool}, \texttt{Bool})$ and $ar(\neg) = 1$.*
  - *$r(=) = (\tau, \tau, \texttt{Bool})$ and $ar(=) = 2$ for all $\tau \in \mathcal{S}$.*
  - *$r(\forall_\tau) = (\tau, \texttt{Bool}, \texttt{Bool})$ and $ar(\forall_\tau) = 2$ for all $\tau \in \mathcal{S}$.*
  - *$r(\exists_\tau) = (\tau, \texttt{Bool}, \texttt{Bool})$ and $ar(\exists_\tau) = 2$ for all $\tau \in \mathcal{S}$.*
  - *$r(f) \in S^{ar(f)} \times S$ for all $f \in \mathcal{F} \backslash \{=, \wedge, \neg, \forall_\tau, \exists_\tau\}$.*

Notice that the *rank* of a function symbol specifies, in order, the expected sorts of the symbol's arguments and result.

As usual, we call the 0-arity function symbols as *constant* symbols.

In our language terms are built out of variables from $\mathcal{V}$ and function symbols from $\mathcal{F}$.

Definition 8.2 (Σ-terms). *Let $\Sigma = \langle \mathcal{S}, \mathcal{V}, \mathcal{F}, ar, r \rangle$ be a signature. The set of well-sorted Σ-terms is the smallest set defined by:*

$$\begin{array}{rcll} T_\Sigma & ::= & x & \textit{where } x \in \mathcal{V} \\ & | & f(t_1, \dots, t_n) & \textit{where } r(f) = (\tau_1, \dots, \tau_n, \tau) \\ & & & \textit{and for each } i \in \{1, \dots, n\}, t_i \textit{ has sort } \tau_i. \end{array}$$

We speak of bound or free (occurrences of) variables in a term as usual. Terms are closed *if they contain no free variables, and open otherwise. Terms are* ground *if they are variable-free (i. e., they contain no variable, neither free nor bound).*

Definition 8.3 (Σ-formulas). *Let $\Sigma = \langle \mathcal{S}, \mathcal{V}, \mathcal{F}, ar, r \rangle$ be a signature. The set of Σ-formulas is defined as the set of well-sorted terms of sort `Bool`.*

Notice that, for simplicity, the defined language does not contain any logical symbols. Logical connectives for negation, conjunction, the equality symbol, and quantifiers, etc., are just considered as function symbols with a predefined rank and sort. This will prove useful at the moment of defining the reduction algorithm in the following section.

Σ-formulas are given a meaning by means of Σ-structures.

Definition 8.4 (Σ-structures). *A structure $\mathcal{I}$ for a signature $\Sigma = \langle \mathcal{S}, \mathcal{V}, \mathcal{F}, ar, r \rangle$ (or Σ-structure), is a pair $\mathcal{I} = \langle D, (\_)^{\mathcal{I}} \rangle$ where D assigns a non-empty domain D_τ to each sort $\tau \in \mathcal{S}$ and $(\_)^{\mathcal{I}}$ assigns a meaning to each variable and function symbol.*

By extension, a Σ-structure defines a value $\mathcal{I}[t]$ in D_τ for every term of sort τ, and a truth value $\mathcal{I}[\varphi]$ in $\{true, false\}$ for every formula φ.

As usual, we say that a closed Σ-formula φ is *satisfiable* if there is a Σ-structure $\mathcal{I}$ that makes the formula true (notation $\mathcal{I} \models \varphi$), and it is *valid* if for all Σ-structures $\mathcal{I}, \mathcal{I} \models \varphi$. A Σ-structure $\mathcal{I}$ satisfies a set of Σ-formulas S ($\mathcal{I} \models S$) if $\mathcal{I} \models \varphi$ for every $\varphi \in S$.

Similarly to more conventional languages, this language is a family of languages parametrized by the signature Σ. However, when working in the context of a background theory $\mathcal{T}$, the specific signature is jointly defined by the declaration of $\mathcal{T}$ plus any additional sort and function symbol declaration contained in the formula. Function symbols are called *interpreted* if they are defined by $\mathcal{T}$, or *uninterpreted* otherwise.

Notice that, as it follows from our definition of Σ-terms, well-sorted terms have a unique sort. Therefore we can define a *sort* function that tells us the sort of every term and function symbol.

Definition 8.5 (Sort function). *Let $\Sigma = \langle \mathcal{S}, \mathcal{V}, \mathcal{F}, ar, r \rangle$ be a signature. We define a* sort function *$sort : \mathcal{V} \cup \mathcal{F} \cup T_\Sigma \mapsto \mathcal{S}$ that maps every variable, function symbol and Σ-term to its corresponding sort.*

In what follows, for simplicity and if not specified otherwise, we may omit the "Σ" prefix for terms, formulas, structures, etc. Moreover, by *SMT formulas* we mean well-formed formulas in the many-sorted first-order language defined previously.

8.2 DETECTING SYMMETRIES

We now present the reduction algorithm for SMT formulas. The algorithm works by constructing the syntax directed acyclic graph of the formula plus additional vertices and coloring it appropriately to avoid spurious symmetries. The coloring of the graph is defined by a *typing function*.

Definition 8.6 (Typing function)**.** *Let $\Sigma = \langle \mathcal{S}, \mathcal{V}, \mathcal{F}, ar, r \rangle$ be a signature and $\mathcal{T}$ a first-order theory over Σ. By $\mathcal{F}_{\mathcal{T}}$ and F_U we denote the sets of all* interpreted *symbols and of all* uninterpreted *symbols of Σ respectively. Let D be a countably infinite set, and e an element of D. We define the injective functions $a : \mathcal{S} \mapsto A$, $b : \mathcal{S}^n \times \mathcal{S} \mapsto B$, and $c : \mathcal{F}_{\mathcal{T}} \mapsto C$ such that A, B, and C are a partition of $D \backslash \{e\}$ (i. e., A, B and C are non-empty, $A \cap B \cap C = \emptyset$ and $A \cup B \cup C = D \backslash \{e\}$). We define the* typing function *$t : \mathcal{F} \cup T_\Sigma \cup \{\star\} \mapsto D$ as:*

$$t(s) = \begin{cases} e & \text{if } s = \star. \\ a(sort(s)) & \text{if } s \text{ is a term or an uninterpreted constant symbol.} \\ b(r(s)) & \text{if } s \text{ is an uninterpreted function symbol of } ar(s) > 0. \\ c(s) & \text{if } s \text{ is an interpreted function symbol.} \end{cases}$$

Notice how the typing function handles the different elements of a formula. It assigns a color to terms and uninterpreted constant symbols based on their sort, to uninterpreted function symbols based on their rank, and to interpreted symbols a unique color independently of their sort and rank. In what follows we assume colors are represented by natural numbers, i. e., $D = \mathbb{N}$ and $e = 0$.

Definition 8.7 (Reduction algorithm)**.** *Let φ be an SMT formula, and t a typing function. Let $\mathcal{F}(\varphi)$ be the set of function symbols occurring in φ. The colored directed graph $G(\varphi) = (V, E)$ is constructed recursively as follows:*

i) For each symbol $s \in \mathcal{F}(\varphi)$:

a) Add a symbol *vertex of color $t(s)$.*

ii) For each term $f(t_1, \ldots, t_n)$, in φ, of arity $n > 0$:

a) Add an occurrence *vertex of color $t(f(t_1, \ldots, t_n))$.*

b) Add an edge from the occurrence vertex to the symbol vertex of f.

c) If the function is commutative (e. g., $\wedge$, $\vee$, $\leftrightarrow$, $=$, $+$, $*$):

i. Add an edge from the occurrence vertex to the root vertex of the graph $G(t_i)$ for $1 \leq i \leq n$.

d) If the function is not commutative:

i. For each argument t_i, add an argument vertex *of color $t(\star)$ and an edge from this vertex to the root vertex of $G(t_i)$.*

ii. Add an edge from the argument vertex of t_i to the argument vertex of t_{i+1} ($1 \leq i < n$). These edges represent the ordering of the arguments in $f(t_1, \ldots, t_n)$.

iii. Add an edge from the occurrence vertex to the argument vertex of t_1.

iii) For each term $Qx.t$ $Q \in \{\forall, \exists\}$:

a) Proceed as for a term of a commutative function symbol, e.g., the function $Q(x, t)$.

Example 8.1. *Consider the formula $\varphi = f(a, b) \vee f(b, a)$ where f, a, and b are uninterpreted symbols such that $sort(a) = sort(b) = U$ and $r(f) = (U, U, \texttt{Bool})$. Figure 8.1 shows its associated colored graph $G(\varphi)$ constructed using the algorithm of Definition 8.7, assuming that f is commutative (colors are represented by shapes in the figure).*

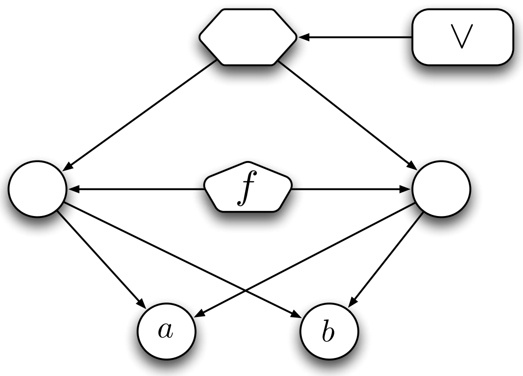


Figure 8.1: Graph representation of $f(a, b) \vee f(b, a)$ (f is commutative).

Example 8.2. *Consider the formula of Example 8.1, but now assume that f is not commutative. Figure 8.2 shows its associated colored graph $G(\varphi)$ (colors are represented by shapes in the figure) constructed using the algorithm of Definition 8.7. Notice how the argument vertices (triangle vertices) are used. For the term $f(a, b)$ the argument vertex of a, the first argument, is directly connected to the occurrence vertex and to the argument vertex corresponding to b. For the term $f(b, a)$ things are inverted. The argument vertex of b is directly connected to the occurrence vertex an to the argument vertex corresponding to a.*

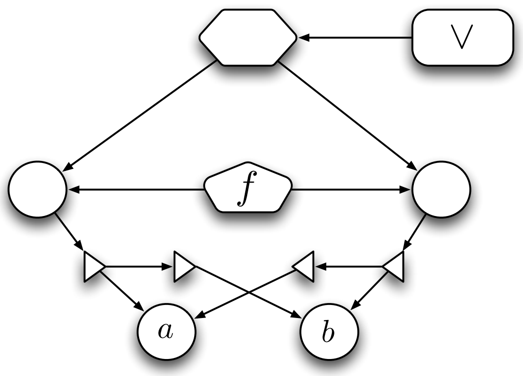


Figure 8.2: Graph representation of $f(a, b) \vee f(b, a)$ (f is not commutative).

Example 8.3. *Consider the formula $\varphi = \forall x.f(x,a)$ where f and a are uninterpreted symbols, sort(a) =`Bool` and $r(f) = (U,$`Bool`$,$`Bool`$)$. Also assume f is commutative. Figure 8.3 shows its associated colored graph $G(\varphi)$ (colors are represented by shapes in the figure). Notice that we handle the universal quantifier as a commutative function, therefore adding no argument vertices.*

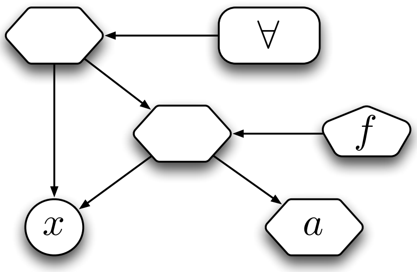


Figure 8.3: Graph representation of $\forall x.f(x,a)$.

Now let us prove that the reduction algorithm is correct.

Theorem 8.1. *Let φ be an SMT formula and $G(\varphi) = (V,E)$ the colored graph constructed from it as defined by Definition 8.7. Then, every automorphism of the graph $G(\varphi)$ is a symmetry of the formula φ.*

Proof. It follows directly by structural induction and from the following observations:

- The graph $G(\varphi)$ is the syntax directed acyclic graph of φ plus additional vertices.
- For terms, $f(t_1,\ldots,t_n)$ of arity > 0, the coloring of vertices, and the combination of root vertices and symbol vertices ensures that only symbols vertices of the same sort (i.e., same arity and same argument sorts) and with the same number of occurrences can be permuted.
- For terms without arguments (constants and predicates), the coloring of symbol vertices and the existence of argument vertices ensures that only symbols of the same sort and occurring the same number of times in the same argument positions can be permuted.

Finally, to reconstruct a formula symmetry from a graph automorphism, we just need to restrict the graph automorphism to symbol vertices. □

Notice that the converse of Theorem 8.1 is false: the proposed graph construction does not find all the symmetries of the input formula. For example, consider the formula $\varphi = f(a,b) \wedge g(c,d)$ where a,d are of some sort and b,c of another sort (with appropriate sorts for f and g). The permutation $\sigma = (f\ g)(a\ c)(b\ d)$ is a symmetry of φ. Nevertheless, we cannot detect it in the graph $G(\varphi)$. This is due to the fact that symbol vertices are colored based on the symbol sort, and this prevent the automorphism component from detecting permutations involving symbols of different sorts. Nevertheless, from a practical point of view, symmetries involving symbols of different sorts are rather unnatural and do not arise often.

8.3 EXPERIMENTAL EVALUATION

We now present empirical results about how often symmetries appear in SMT benchmarks and how hard it is to actually find them.

8.3.1 *Implementation*

For empirical evaluation we implemented the reduction algorithm of Definition 8.7 in the tool SyMT[1].

SyMT is a command line tool implemented in C. It takes into account the commutativity of conjunction, disjunction, addition, multiplication and equality. Given an input SMT formula, SyMT proceeds by creating a colored graph and using a graph automorphism component to compute generators of the automorphism group of the colored graph. In particular, SyMT uses Saucy 3.0 [Katebi *et al.*, 2012] as the graph automorphism component. Integration with Saucy is done via its C API. SyMT also provides simplification capabilities on input formulas, some of which involve theory reasoning (which may fail on large instances). Simplification of input formulas is important because it may uncover hidden symmetries and remove trivial symmetries (e.g., symmetries that do not involve uninterpreted symbols). Simplifications include simple rewriting, simplification of entailed literals, and some normalization of terms and formulas. It is activated with the `-s` option of the tool.

To improve readability of the symmetry group of the input formula, we have developed two post-processing techniques to identify subgroups that are full permutation groups on a subset of the symbols involved in the symmetry. The first one is based on the Schreir-Sims algorithm [Seress, 2003; Rehn and Schürmann, 2010], a polynomial time algorithm that, among other things, makes it possible to check if a given permutation belongs to the group generated by a set of generators. We use this algorithm to check, for each orbit, if the full group of permutations of the elements in the orbit is a subgroup of the symmetry group. Indeed, to check if a group admits as a subgroup the full set of permutations of these elements, it is sufficient to check membership of two permutations, e.g., one permutation between two arbitrary elements, and one cyclic permutation involving all the elements in the orbit.

Although polynomial, the above technique is often unacceptably slow. We designed a second, heuristic-based, efficient technique. In essence, it partitions symbols (using a union-find data-structure) into classes such that two symbols in the same class are symmetric. Generators are used to repeatedly merge classes in the partition, and the partition is used to simplify generators. In practice, this technique works well in most cases, although it can fail to detect full permutation subgroups in some cases. The first and the second techniques are activated with the `-R` and `-r` options, respectively.

Example 8.4. *The command line and output of SyMT on a formula of the QF_UF category of SMT-LIB is as follows:*

1 Download from: `http://www.verit-solver.org/SyMT/`

```
./SyMT -s -r smt-lib2/QF_UF/NEQ/NEQ004_size4.smt2
[c_0 c_1 c_2 c_3]
(p7 p9)(c12 c13)
```

SyMT finds generators for the symmetry group. It detects — using the post-processing techniques described above — that there is a full subgroup of permutation for constants `c_0`*,* `c_1`*,* `c_2`*,* `c_3`*, and a further symmetry that permutes binary predicate symbols* `p7` *and* `p9` *together with* `c12` *and* `c13`*.*

8.3.2 *Results*

We tested SyMT against 19 categories[2] from SMT-LIB [Barrett *et al.*, 2010a] to investigate the existence of symmetries and evaluate the efficiency of the tool. All tests were run on an Intel Xeon X3440 with 16GB. Three different configurations of SyMT were tested. Configuration 1 has no simplification: the formula is parsed and converted to a graph for automorphism detection. Configuration 2 uses trivial syntactic simplifications. Configuration 3 enables stronger simplifications, using the SMT engine, e.g., simplification of atoms implied by unit clauses. Configuration 2 may fail (with no symmetry reported) because the time complexity of the simplification algorithm used is not linear with respect to the size of the input formula. However it often reveals symmetries hidden by syntactic noise easily removed by the simplification procedure. Configuration 3 is likely to fail on large formulas, but again, it may reveal hidden symmetries. Simplification sometimes reduces a formula to false, in which case no symmetry is reported. The timeout (relevant for configuration 2 and, mostly, for configuration 3) is set to 30 seconds.

Among the 19 analyzed categories, three (LRA, QF_UFLRA, QF_UFNRA) do not reveal symmetries with SyMT. Of the only five formulas in UFLRA, one has symmetries. The other 15 categories presented a significant number of symmetries in at least one of the tested configurations. Table 8.1 summarizes the results obtained for these 15 categories. For each category we report the number of instances (#Inst), the number of instances that have symmetries for the various simplification configurations (#Sym[1], #Sym[2] and #Sym[3]), the number of instances that have symmetries in at least one of the configurations (#Sym[P]), the average logarithm in base 2 of the size of the symmetry group (Avg[GS]) for Configuration 1, and the total time in seconds required to analyze all the instances (Time) also for Configuration 1.

Table 8.1 shows clearly that the SMT-LIB has many highly symmetric formulas, in most categories. Moreover, an in-depth study on the detected symmetries reveals that symmetries provide information that highly depends on the category. For example, many symmetries in the quantified formulas (e.g., AUFLIA) are due to repeated (but unused) axiomatisations. While symmetries in the `FISCHER` series of benchmarks (QF_LIA, QF_IDL), which has been automatically generated from bounded model checking of distributed algorithms, reveals that some processes are symmetric.

The cumulative time required to build the graph and detect the symmetries is negligible in all categories. We do not output the times for other configurations

2 Bit vectors are not supported yet by our parser.

Category	#Inst	#Sym[1]	#Sym[2]	#Sym[3]	#Sym[P]	Avg[GS]	Time
AUFLIA	6480	6212	6231	5941	6258	134.00	378.79
AUFLIRA	19917	15779	16475	12500	16476	1.08	9.13
AUFNIRA	989	985	985	923	985	1.00	0.41
QF_AUFLIA	1140	2	71	77	78	1.00	0.72
QF_AX	551	22	22	22	22	1.00	0.37
QF_IDL	1749	348	526	683	756	12745.43	327.95
QF_LIA	5938	728	1172	524	1200	104.55	486.19
QF_LRA	634	73	150	208	210	110.49	29.06
QF_NIA	530	169	169	168	169	5.92	3.92
QF_NRA	166	9	43	43	43	1.00	0.23
QF_RDL	204	0	0	24	24	0.00	10.13
QF_UF	6639	250	3638	375	3638	44.00	34.58
QF_UFIDL	431	19	175	186	189	1.00	2.70
QF_UFLIA	564	0	198	198	198	0.00	0.45
UFNIA	1796	1062	1061	1058	1070	47.08	543.26

Table 8.1: Symmetries in the SMT-LIB.

since there are timeouts and time is dominantly spent in the simplification modules, so these numbers give little insight about symmetry detection itself.

8.4 SUMMARY

In this chapter we have presented a reduction algorithm for detecting symmetries in SMT formulas.

We proved its correctness and then we implemented this algorithm in SyMT, a simple, yet powerful, tool to detect symmetries in SMT formulas. Then we showed that symmetry detection scales on SMT formulas by providing experimental results on executions of the tool on many SMT-LIB categories and showing that many formulas in the SMT-LIB repository exhibit symmetries.

As far as we known, the only alternative technique for detecting symmetries in SMT formulas is the one presented in [Déharbe *et al.*, 2011]. However, this technique is rather heuristic and only detects full group of symmetries missing more general symmetries (e. g., in Example 8.4, using our reduction algorithm we find the symmetry `(p7 p9)(c12 c13)` which is not detected using the techniques of [Déharbe *et al.*, 2011]).

It now remains to see how to use the detected symmetries to improve SMT solving.

9 SYMMETRIES IN MODAL TABLEAUX

In this chapter we present a technique to exploit symmetries in a modal tableau. The technique consists of a blocking mechanism that takes advantage of symmetry information about the input formula to restrict the application of the ($\Diamond$) rule. We start by presenting the basics about labeled tableaux for the basic modal logic (Section 9.1). Then we present the theoretical basis for the blocking mechanism and prove its correcteness (Section 9.2). Finally we present experimental results that show the applicability of this mechanism (Section 9.3).

The results presented in this chapter were published in [Areces and Orbe, 2013].

9.1 LABELED TABLEAUX FOR THE BASIC MODAL LOGIC

In this section we present a labeled tableau calculus for the basic modal logic [Blackburn *et al.*, 2006].

The semantic tableau, or tableau for short, is a proof procedure that decides satisfiability of a formula based on the satisfiability of its subparts. It was introduced in [Beth, 1959], and took its current form independently in [Lis, 1960] and [Smullyan, 1968]. Currently, it is the most popular proof procedure for modal logics because of its many successful computer implementations and its flexibility to adapt to different logics [D'Agostino, 1999].

From a proof theoretic point of view, a tableau calculus can be categorized as a *backward reasoning* method. In backward reasoning methods, we begin with the desired result and work backward from there to create a proof. This is in contrast to, so called, *forward reasoning* methods in which we start with axioms and rules and finish with the desired theorem. Most tableaux calculus satisfy the *subformula property*, i. e., all formulas in a proof are subformulas of the formula being proved. We express this by saying tableaux are *analytic*.

Intuitively, a tableau calculus works by "breaking" complex formulas into smaller ones until complementary pairs of literals are produced or no further expansion is possible. To do so, a tableau calculus consists of a finite collection of rules, one per logical connective, called *tableau expansion rules*. Each rule specifies how to break down one logical connective into its constituent parts and has its condition on when to apply it. These conditions are generally defined on the presence of a formula of certain shape. A tableau calculus may also contain additional global constraints that may prevent the application of certain rules in some cases.

Given a formula φ, a tableau calculus works by building a tree whose nodes are labeled with formulas (set of formulas), with its root node labeled as $\{\varphi\}$, and where edges represent rule applications. We call this tree a *tableau for φ*. At each step, the tree is modified by a rule application that adds a node or creates branches in it. A rule can be applied only if it adds a formula to the current set of formulas in all branches it creates.

$$\frac{\varphi \wedge \psi}{\varphi, \psi}\,(\wedge) \qquad \frac{\varphi \vee \psi}{\varphi \mid \psi}\,(\vee)$$

Figure 9.1: Tableau calculus for propositional logic.

Every tableau can be considered as a graphical representation of a formula, which is equivalent to the set of formulas the tableau is built from. This formula is as follows: each branch of the tableau represents the conjunction of its formulas; the tableau represents the disjunction of its branches. Expansion rules transform a tableau into one having an equivalent represented formula. Since the tableau is initialized as a single branch containing the formulas of the input set, all subsequent tableaux obtained from it represent formulas which are equivalent to that set.

Since the formula represented by a tableau is the disjunction of the formulas represented by its branches, contradiction is obtained when every branch contains a pair of opposite literals. Once a branch contains a literal and its negation, its corresponding formula is unsatisfiable. As a result, this branch can be now "closed", as there is no need to further expand it. If all branches of a tableau are closed, the formula represented by the tableau is unsatisfiable; therefore, the original set is unsatisfiable as well. Obtaining a tableau where all branches are closed is a way of proving the unsatisfiability of the original set. To prove satisfiability it suffices to find a branch that cannot be closed provided that every rule has been applied everywhere it could be applied. A branch were no rule is applicable is said to be *saturated*.

Example 9.1 (Tableau Calculus for Propositional Logic)**.** *Let us consider a tableau calculus for propositional logic. For simplicity sake we assume that all formulas are in negation normal form, and that the set of propositional formulas* FORM *is defined by the following grammar:*

$$\mathsf{FORM} ::= p \mid \neg p \mid \varphi \vee \psi \mid \varphi \wedge \psi,$$

where $p \in \mathsf{PROP}$*, with* $\mathsf{PROP} = \{p_1, p_2, \ldots\}$ *a countable infinite set of propositional variables, and* $\varphi, \psi \in \mathsf{FORM}$*.*

Figure 9.1 shows the tableau rules for the logical connectors $\wedge$ *and* $\vee$*. The rule* $(\wedge)$ *states that whenever a tableau contains a formula* $\varphi \wedge \psi$*, i. e., the conjunction of two formulas, these two formulas are both consequences of that formula, i. e., if* $\varphi \wedge \psi$ *holds, then* φ *and* ψ *hold. The rule* $(\vee)$ *states that whenever a tableau contains a formula* $\varphi \vee \psi$ *we need to explore one of the two disjuncts first, check whether that choice leads to the conclusion that the formula is satisfiable, and if not, try the other disjunct. This rule splits a branch in two, each one representing the possible choices made. The symbol "|" represents that the addition of* φ *and the addition of* ψ *belong to two separate branches.*

Example 9.2. *Consider the formula* $\varphi = ((p \vee q) \wedge \neg p) \wedge \neg q$*. Figure 9.2 shows its corresponding tableau. In this case both branches have a contradiction:* $\neg p$ *and* p *on the left branch, and* $\neg q$ *and* q *on the right branch. Therefore, the tableau is closed and the formula is unsatisfiable.*

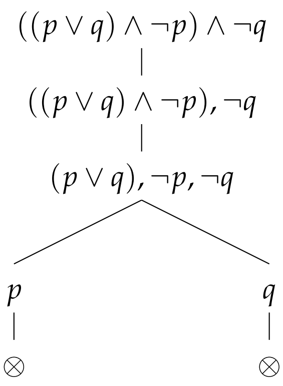

$((p \vee q) \wedge \neg p) \wedge \neg q$
$((p \vee q) \wedge \neg p), \neg q$
$(p \vee q), \neg p, \neg q$
p
q
$\otimes$
$\otimes$

Figure 9.2: Tableau for the formula $\varphi = ((p \vee q) \wedge \neg p) \wedge \neg q$.

Example 9.3. *Consider the formula $\varphi = ((p \vee q) \wedge \neg p) \wedge (\neg q \vee q)$. Figure 9.3 shows a tableau for φ. In this case the right-most branch remains open after applying all possible rules, therefore the formula is satisfiable.*

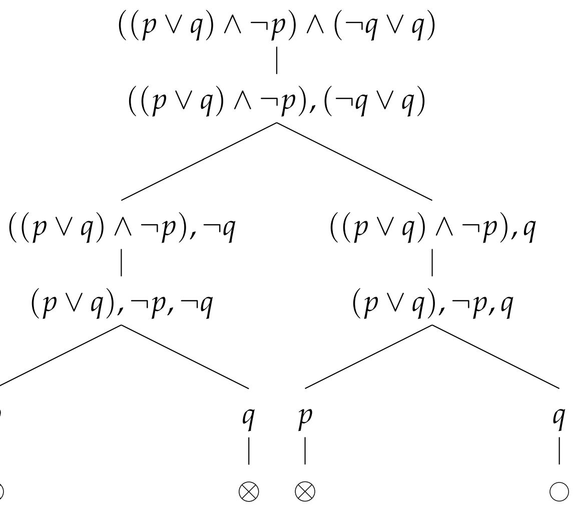

$((p \vee q) \wedge \neg p) \wedge (\neg q \vee q)$
$((p \vee q) \wedge \neg p), (\neg q \vee q)$
$((p \vee q) \wedge \neg p), \neg q$
$((p \vee q) \wedge \neg p), q$
$(p \vee q), \neg p, \neg q$
$(p \vee q), \neg p, q$
p
q
p
q
$\otimes$
$\otimes$
$\otimes$
$\bigcirc$

Figure 9.3: Tableau for the formula $\varphi = ((p \vee q) \wedge \neg p) \wedge (\neg q \vee q)$.

Let us now present the labeled (prefixed) tableau calculus for the basic modal logic that we use in the following sections.

Labeled tableaux systems for the basic modal logic were introduced in [Fitting, 1972; Fitting, 1983], but took on their present form in [Massacci, 1994; Goré, 1999]. Labeled tableaux are closely related to propositional tableaux: they are sets of labeled formulas that partially describes a model. However, instead of only taking into account truth values of propositional symbols in one "world", they do so in an arbitrary number of connected worlds, and each formula is labeled with the world in which it should be true.

Let us define modal tableaux more formally. Let PREF be an infinite, non-empty set of *prefixes*. Here we set PREF $= \mathbb{N}$. We assume that all formulas are modal CNF formulas as defined in Definition 3.1.

$$\frac{\alpha{:}\varphi}{\alpha{:}C_i,}\ (\wedge) \quad \text{for all } C_i \in \varphi \qquad \frac{\alpha{:}C}{\alpha{:}l_1 \mid \ldots \mid \alpha{:}l_n,}\ (\vee) \quad \text{for all } l_i \in C$$

$$\frac{\alpha{:}\neg\Box C}{\alpha R\alpha',\ \alpha'{:}{\sim}C,}\ (\Diamond)^1 \qquad \frac{\alpha{:}\Box C, \alpha R\alpha'}{\alpha' : C}\ (\Box)$$

^{1} ${\sim}C$ is the CNF of the negation of C. The prefix α' is new in the tableau.

Figure 9.4: Labeled tableau calculus for the basic modal logic.

Definition 9.1 (Prefixed Formulas)**.** *Given φ a modal CNF formula, C a modal clause and $\alpha \in$ PREF we call $\alpha{:}\varphi$ and $\alpha{:}C$* prefixed formulas. *The intended interpretation of a prefixed formula $\alpha{:}F$ is that F holds at the state denoted by α.*

Definition 9.2 (Accessibility Statements)**.** *Given $\alpha, \alpha' \in$ PREF we call $\alpha R\alpha'$ an* accessibility statement. *The intended interpretation of $\alpha R\alpha'$ is that the state denoted by α' is accessible via R from α.*

A tableau for a modal CNF formula φ is a tree whose nodes are decorated with prefixed formulas and accessibility statements, such that the root node is $0{:}\varphi$. Moreover, we require that additional nodes in the tree are created according to the rules of Figure 9.4, where φ is a modal CNF formula and C a modal clause.

The rules are interpreted as follows: if the antecedents of a rule appear in nodes in a branch of the tableau, the branch is extended according to the formulas in the consequent. For the case of the $(\vee)$ rule, n immediate successors of the last node of the branch should be created. In all other cases only one successor is created, which is decorated with the indicated formulas. To ensure termination, we require that nodes are created only if they add at least one prefixed formula or accessibility statement that was not already in the branch. Moreover, the $(\Diamond)$ rule can only be applied once to each formula of the form $\alpha{:}\neg\Box C$ in a branch.

Definition 9.3 (Closed, open and saturated branch)**.** *A branch is* closed *if it contains both $\alpha{:}p$ and $\alpha{:}\neg p$, and it is* open *otherwise. We say that a branch is* saturated *if no rule can be further applied in the branch.*

Let $\mathsf{Tab}(\varphi)$ be the set of tableaux for φ whose branches are all saturated. The following classical result establishes that the tableau calculus we just defined is a decision procedure for satisfiability of formulas in the basic modal logic.

Theorem 9.1. *For any formula φ of the basic modal logic, any tableau $\mathsf{T} \in \mathsf{Tab}(\varphi)$ is finite. Moreover T has a saturated open branch if and only if φ is satisfiable.*

Example 9.4. *Consider the formula $\varphi = \Box(p \vee q) \wedge \neg\Box r \wedge \neg\Box s$. Figure 9.5 shows its corresponding tableau. In this case all branches are open after being saturated, i. e., after applying all possible rules, therefore the formula is satisfiable.*

This completes our brief introduction of the standard tableau calculus for the basic modal logic.

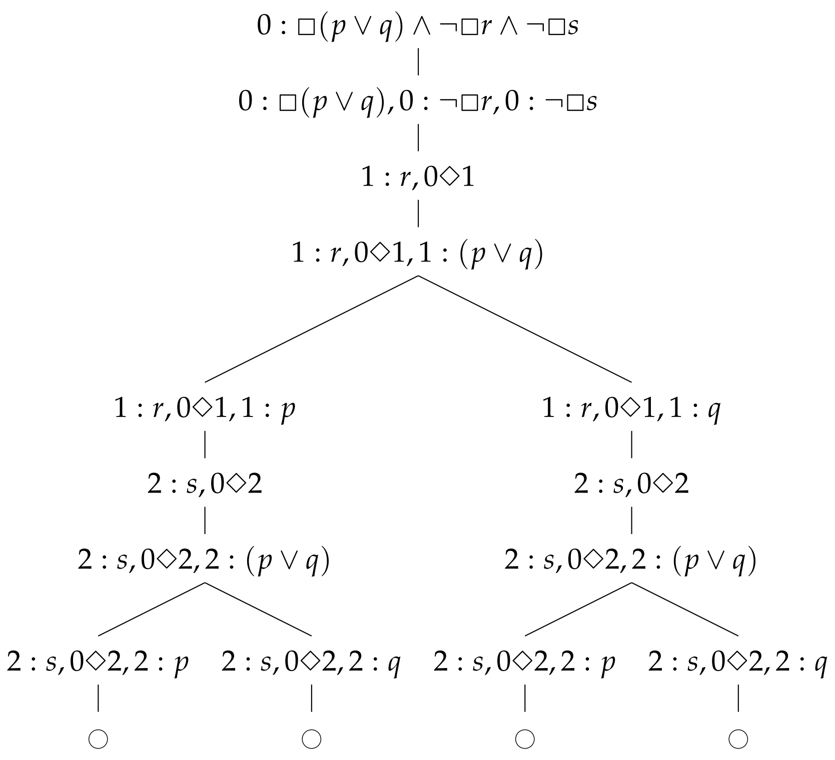


Figure 9.5: Tableau for the formula $\varphi = \Box(p \vee q) \wedge \neg\Box r \wedge \neg\Box s$.

9.2 SYMMETRY BLOCKING

The tableau calculus of Figure 9.4 is terminating because we restrict the application of the $(\Diamond)$ rule such that it can only be applied once to each formula of the form $\alpha{:}\neg\Box C$ in a branch.

We are interested in further restricting the application of the $(\Diamond)$ rule so that it can be applied to a $\alpha{:}\neg\Box C$ formula only if it has not been applied to a symmetric formula before. We call this restriction *symmetry blocking*.

From now on let $\mathsf{T} \in \mathsf{Tab}(\varphi)$, and let Θ be a branch of T. In what follows we work with *permutation sequences* as defined in Definition 3.34. A permutation sequence $\bar{\sigma}$ is either $\bar{\sigma} = \langle\rangle$ (i.e., $\bar{\sigma}$ is the empty sequence) or $\bar{\sigma} = \sigma : \bar{\sigma}_2$ with σ a permutation and $\bar{\sigma}_2$ a permutation sequence and we often write it as $\bar{\sigma} = \langle\sigma_1, \ldots, \sigma_n\rangle$. For a permutation sequence of length n and $1 \leq i \leq n$, we write $\bar{\sigma}_i$ for the subsequence that starts from the i^{th} element of $\bar{\sigma}$. For $i \geq n$, we define $\bar{\sigma}_i = \langle\rangle$ (the empty sequence). For a modal CNF formula φ and a permutation sequence $\bar{\sigma}$, Definition 3.35 defines $\bar{\sigma}(\varphi)$ recursively as follows:

$$\begin{array}{rcll}
\langle\rangle(\varphi) & = & \varphi & \\
(\sigma_1 : \bar{\sigma}_2)(l) & = & \sigma_1(l) & \text{for } l \in \mathsf{PLIT} \\
(\sigma_1 : \bar{\sigma}_2)(\Box C) & = & \Box\bar{\sigma}_2(C) & \\
\bar{\sigma}(C) & = & \{\bar{\sigma}(A) \mid A \in C\} & \text{for } C \text{ a clause or a formula.}
\end{array}$$

Definition 9.4 (Distance from the root prefix). *Let $\alpha \in \mathsf{PREF}$ be a prefix. We define $\mathsf{depth}(\alpha)$ as the distance from the root prefix to the prefix α, measured as the number of* *accessibility statements that have to be traversed to reach the prefix α from the root prefix (in particular, if α is the prefix of φ then $\mathsf{depth}(\alpha) = 0$).*

Given a prefixed formula $\alpha{:}\varphi$ and a permutation sequence $\bar{\sigma}$, we define $\bar{\sigma}(\alpha{:}\varphi) = \alpha{:}\bar{\sigma}_{\mathsf{depth}(\alpha)+1}(\varphi)$. Finally, given a modal formula φ, we define $Vars(\varphi)$ as the set of propositional variables occurring in φ; for S a set of modal formulas, let $Vars(S) = \bigcup_{\varphi \in S} Vars(\varphi)$.

Definition 9.5 (Symmetry Blocking). *Let $\bar{\sigma}$ be a layered symmetry of φ, and let Θ be a branch in a tableau of φ. The rule $(\Diamond)$ cannot be applied to $\alpha{:}\bar{\sigma}(\neg\Box\psi)$ on Θ if it has been applied to $\alpha{:}\neg\Box\psi$ and $Vars(\bar{\sigma}(\neg\Box\psi)) \cap Vars(\Gamma(\alpha)) = \emptyset$, for $\Gamma(\alpha) = \{\psi \mid \alpha : \Box\psi \in \Theta\}$ the set of $\Box$-formulas occurring at prefix α.*

Actually, a more strict symmetry blocking condition is possible, where instead of requiring $Vars(\bar{\sigma}(\neg\Box\psi)) \cap Vars(\Gamma(\alpha)) = \emptyset$ we verify that the variables at each modal depth of $\bar{\sigma}(\neg\Box\psi)$ are disjoint from those in $\Gamma(\alpha)$ (i.e., $\forall n. Vars(\bar{\sigma}(\neg\Box\psi), n) \cap Vars(\Gamma(\alpha), n) = \emptyset$). But this more aggressive blocking condition did not have an impact in our experiments. It is easy to find cases where a $\neg\Box$-formula is only blocked under the more strict condition, but we did not find any such case in the test sets we investigated.

Notice that symmetry blocking is a dynamic condition: after being blocked, a $\neg\Box$-formula can be scheduled for expansion if the blocking condition fails in an expansion of the current branch. This can happen because the set $\Gamma(\alpha)$ increases monotonically as the tableau advances. Also notice that this restriction cannot affect termination or soundness of the calculus, because we are imposing further restrictions on a method that is known to be sound, complete and terminating. We only risk losing completeness.

Example 9.5. *Consider the formula of Example 9.4 and the layered symmetry $\bar{\sigma} = \langle \sigma_{Id}, (r\ s)(\neg r\ \neg s)\rangle$, where σ_{Id} is the identity permutation. Notice that $0 : \neg\Box s = \bar{\sigma}(0 : \neg\Box r)$ and that $Vars(\bar{\sigma}(0 : \neg\Box r)) \cap Vars(\Gamma(0)) = \{s\} \cap \{p, q\} = \emptyset$, therefore, the symmetry blocking condition is met and symmetry blocking is applicable to $0 : \neg\Box s$. Figure 9.6 shows its corresponding tableau with symmetry blocking. Notice that we only have to explore two branches (instead of four) to conclude that the formula is satisfiable.*

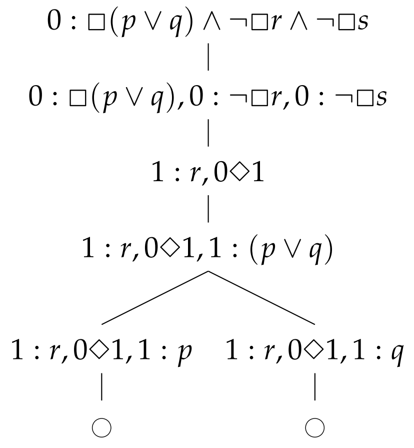

$0 : \Box(p \vee q) \wedge \neg\Box r \wedge \neg\Box s$
$0 : \Box(p \vee q), 0 : \neg\Box r, 0 : \neg\Box s$
$1 : r, 0\Diamond 1$
$1 : r, 0\Diamond 1, 1 : (p \vee q)$
$1 : r, 0\Diamond 1, 1 : p$
$1 : r, 0\Diamond 1, 1 : q$
$\bigcirc$
$\bigcirc$

Figure 9.6: Tableau for the formula $\varphi = \Box(p \vee q) \wedge \neg\Box r \wedge \neg\Box s$.

Example 9.5 shows the potential savings that we can obtain using symmetry blocking. Given that the blocked formula could be arbitrarily complex, symmetry blocking could prevent us from doing a large amount of unnecessary work.

9.2.1 *Completeness*

To prove completeness of our calculus with symmetry blocking we show that we can extend an incomplete model $\mathcal{M}^{\Theta}$, built from a saturated open branch Θ to a complete model even when symmetry blocking was used.

From now on, we consider that branches in $\mathsf{Tab}(\varphi)$ are saturated using symmetry blocking.

Definition 9.6. *Given an open saturated branch Θ of the tableau $\mathsf{T} \in \mathsf{Tab}(\varphi)$, we define a model $\mathcal{M}^{\Theta} = \langle W^{\Theta}, R^{\Theta}, V^{\Theta} \rangle$ as:*

$$\begin{array}{rcl} W^{\Theta} & = & \{\alpha \mid \alpha \textit{ is a prefix on } \Theta\} \\ R^{\Theta} & = & \{(\alpha, \alpha') \mid \alpha, \alpha' \in W^{\Theta} \textit{ and } \alpha R \alpha' \in \Theta\} \\ V^{\Theta}(\alpha) & = & \{p \mid \alpha : p \in \Theta\}. \end{array}$$

Notice that $\mathcal{M}^{\Theta}$ is always a tree. Given a tree and a permutation sequence, we can construct a new model as follows.

Definition 9.7 ($\bar{\sigma}$*-image* of a tree model)**.** *Given a pointed tree model $\mathcal{M} = \langle w, W, R, V \rangle$ and a permutation sequence $\bar{\sigma}$. Let $\mathsf{depth}(v)$ be the distance of v to the root w (in particular $\mathsf{depth}(w) = 0$). We define $\mathcal{M}_{\bar{\sigma}} = \langle w, W_{\bar{\sigma}}, R_{\bar{\sigma}}, V_{\bar{\sigma}} \rangle$ (the $\bar{\sigma}$-*image *of $\mathcal{M}$) as the model identical to $\mathcal{M}$ except that*

$$V_{\bar{\sigma}}(v) = \bar{\sigma}_{\mathsf{depth}(v)+1}(L_{V(v)}) \cap \mathsf{PROP},$$

where $L_{V(v)}$ is the consistent and complete set of literals generated by $V(v)$ (see Definition 3.5).

Given a model $\mathcal{M}$ and an element $w \in W$, let $W[w]$ denote the set of all elements that are reachable from w (by the reflexive and transitive closure of the accessibility relation). Let $\mathcal{M}[w] = \langle w, W[w], R \restriction W[w], V \restriction W[w] \rangle$ denote the sub-model of $\mathcal{M}$ rooted at w.

Given Θ a saturated open branch, let Σ be the set of prefixes added to Θ by the application of the ($\Diamond$) rule to a $\neg\Box$-formula that has a symmetric $\neg\Box$-formula blocked by symmetry blocking. Intuitively, Σ contains the roots of the sub-models that need a symmetric counterpart in the completion of $\mathcal{M}^{\Theta}$.

Let $M[\Sigma] = \{\mathcal{M}^{\Theta}[\alpha] \mid \alpha \in \Sigma\}$. This set contains the sub-models to which we need to construct a symmetric sub-model. By $M[\Sigma]_{\bar{\sigma}} = \{\mathcal{M}^{\Theta}[\alpha]_{\bar{\sigma}} \mid \mathcal{M}^{\Theta}[\alpha] \in M[\Sigma]\}$ we denote the set of $\bar{\sigma}$-images corresponding to the set of sub-models $M[\Sigma]$. Intuitively, $M[\Sigma]_{\bar{\sigma}}$ is the set of models that we need to "glue" to the model $\mathcal{M}^{\Theta}$ to obtain a complete model.

Definition 9.8 (Symmetric Extension). *Given a saturated open branch Θ, a model $\mathcal{M}^{\Theta} = \langle W^{\Theta}, R^{\Theta}, V^{\Theta} \rangle$ and a set of symmetric pointed sub-models $M[\Sigma]_{\bar{\sigma}}$. Define the symmetric extension of $\mathcal{M}^{\Theta}$ as the model $\mathcal{M}^{\Theta}_{\bar{\sigma}} = \langle W^{\Theta}_{\bar{\sigma}}, R^{\Theta}_{\bar{\sigma}}, V^{\Theta}_{\bar{\sigma}} \rangle$ where:*

$$\begin{aligned} W^{\Theta}_{\bar{\sigma}} &= W^{\Theta} \uplus \biguplus W \\ R^{\Theta}_{\bar{\sigma}} &= R^{\Theta} \uplus \biguplus R \cup \{(\alpha, \tau_{\alpha'}) \mid (\alpha, \alpha') \in R^{\Theta}\} \\ V^{\Theta}_{\bar{\sigma}}(\alpha_i) &= V^{\Theta} \uplus \biguplus V \end{aligned}$$

for all $\langle \tau_{\alpha'}, W, R, V \rangle \in M[\Sigma]_{\bar{\sigma}}$, where $\tau_{\alpha'}$ is the element corresponding to α' in the disjoint union.

Notice that we are *gluing* the symmetric sub-models to the original model by adding an edge from the element α to the root, $\tau_{\alpha'}$, of the symmetric sub-model if there is an edge from α to the root, α', of the original sub-model. To be sure that this construction is sound we have to check that after adding these sub-models to the original model, the $\Box$-formulas holding at α, $\Gamma(\alpha)$, still hold.

The following lemma is the key to prove completeness of our calculus. First recall the following well known result.

Proposition 9.1. *Let φ be a modal formula and* PROP *a set of propositional variables such that $Vars(\varphi) \subseteq$ PROP. Let $\mathcal{M}$ be a model such that $V : W \mapsto \mathcal{P}(\mathsf{PROP})$. Then $\mathcal{M}, w \models \varphi$ iff $\mathcal{M} {\restriction} Vars(\varphi), w \models \varphi$.*

Lemma 9.1. *Let φ be a modal CNF formula, $\bar{\sigma}$ a symmetry of φ and Θ be a saturated open branch of* $\mathsf{T} \in \mathsf{Tab}(\varphi)$. *Let α be a prefix such that $\alpha{:}\neg\Box\psi \in \Theta$ and let $\alpha{:}\bar{\sigma}(\neg\Box\psi) \in \Theta$ be a blocked formula. Then $\bar{\sigma}(\psi) \bigwedge \Gamma(\alpha)$ is satisfiable.*

Proof. Given that Θ is a saturated open branch, we know that $\alpha R \alpha' \in \Theta$ and that $\alpha' : \psi \bigwedge \Gamma(\alpha) \in \Theta$. From Θ we can construct a model $\mathcal{M}^{\Theta} = \langle W^{\Theta}, R^{\Theta}, V^{\Theta} \rangle$ such that $\mathcal{M}^{\Theta}, 0 \models \varphi$ and, in particular, $\mathcal{M}^{\Theta}, \alpha' \models \psi \bigwedge \Gamma(\alpha)$ with $(\alpha, \alpha') \in R^{\Theta}$.

Now consider the sub-model rooted at α', $\mathcal{M}^{\Theta}[\alpha'] = \langle \alpha', W', R', V' \rangle$, where $W' = W^{\Theta}[\alpha']$, $R' = R^{\Theta} {\restriction} W^{\Theta}[\alpha']$ and $V' = V^{\Theta} {\restriction} W^{\Theta}[\alpha']$. We know that $\mathcal{M}^{\Theta}[\alpha'] \models \psi \bigwedge \Gamma(\alpha)$. Now consider $\mathcal{N} = \mathcal{M}^{\Theta}[\alpha'] {\restriction} Vars(\psi) = \langle \alpha', W', R', V'_{\mathcal{N}} \rangle$ and $\mathcal{R} = \mathcal{M}^{\Theta}[\alpha'] {\restriction} Vars(\Gamma(\alpha))$ $= \langle \alpha', W', R', V'_{\mathcal{R}} \rangle$. By Proposition 9.1 we know that $\mathcal{N} \models \psi$ and $\mathcal{R} \models \bigwedge \Gamma(\alpha)$.

Let $\mathcal{N}' = \bar{\sigma}(\mathcal{N}) = \langle \alpha', W', R', V''_{\mathcal{N}} \rangle$. By construction, $\mathcal{N} \underrightarrow{\rightarrow}_{\bar{\sigma}} \mathcal{N}'$ and therefore $\mathcal{N}' \models \bar{\sigma}(\varphi)$. Finally, let $\mathcal{U} = \mathcal{N}' \cup \mathcal{R} = \langle \alpha, W', R', V''' \rangle$ where $V'''(w) = V''_{\mathcal{N}}(w) \cup V'_{\mathcal{R}}(w)$ for all $w \in W'$. By the symmetry blocking condition we know that $Vars(\bar{\sigma}(\psi)) \cup Vars(\Gamma(\alpha)) = \emptyset$ and therefore $L_{V''_{\mathcal{N}}(w)} \cap L_{V'_{\mathcal{R}}(w)} = \emptyset$ for all $w \in W'$. It follows that no contradiction will arise when doing $V''_{\mathcal{N}}(w) \cup V'_{\mathcal{R}}(w)$ and hence that the valuation function $V'''(w)$ is well defined.

Now we have to prove that $\mathcal{U} \models \bar{\sigma}(\psi) \bigwedge \Gamma(\alpha)$. First we prove that $\mathcal{U} \models \bar{\sigma}(\psi)$. Take the restriction of $\mathcal{U}$ to $Vars(\bar{\sigma}(\psi))$, $\mathcal{U} {\restriction} Vars(\bar{\sigma}(\psi))$. By construction of $\mathcal{U}$, we know that $\mathcal{U} \restriction Vars(\bar{\sigma}(\psi)) = \mathcal{N}'$ and that $\mathcal{N}' \models \bar{\sigma}(\psi)$. By Proposition 9.1, $\mathcal{U} \models \bar{\sigma}(\psi)$. That $\mathcal{U} \models \bigwedge \Gamma(\alpha)$ holds, follows by the same argument using the model $\mathcal{R}$. $\Box$

We are now ready to prove a correspondence between formulas in a branch Θ and truth in the symmetric extension of model built from it.

Lemma 9.2. *Let Θ be a saturated open branch of a tableau* $\mathsf{T} \in \mathsf{Tab}(\varphi)$ *and $\bar{\sigma}$ a symmetry of φ. For any formula $\alpha : \psi \in \Theta$ we have that $\mathcal{M}^{\Theta}_{\bar{\sigma}}, \alpha \models \psi$.*

Proof. The proof is by induction on ψ.

Base Case:

- Suppose $\psi = p$. By definition, $\alpha \in V^{\Theta}_{\bar{\sigma}}(p)$. This implies $\mathcal{M}^{\Theta}_{\bar{\sigma}}, \alpha \models p$.

- Suppose $\psi = \neg p$. Since Θ is open, $\alpha{:}p \notin \Theta$. Thus $\alpha \notin V^{\Theta}_{\bar{\sigma}}(p)$, which implies $\mathcal{M}^{\Theta}_{\bar{\sigma}}, \alpha \models \neg p$.

Inductive Step:

- Suppose $[\psi = \chi \wedge \theta]$ and $[\psi = \chi \vee \theta]$. Both cases are trivial, by application of the corresponding tableau rules and the induction hypothesis.

- Suppose $\psi = \neg\Box\theta$. We have to consider two cases:

  i) $\neg\Box\theta$ has been expanded by the application of the $(\Diamond)$ rule. By saturation of $(\Diamond)$, $\alpha R \alpha'$, $\alpha'{:}\theta \in \Theta$. By definition of $R^{\Theta}_{\bar{\sigma}}$ and induction hypothesis: $(\alpha, \alpha') \in R^{\Theta}_{\bar{\sigma}}$ and $\mathcal{M}^{\Theta}_{\bar{\sigma}}, \alpha' \models \theta$. Combining this, we obtain $\mathcal{M}^{\Theta}_{\bar{\sigma}}, \alpha \models \neg\Box\theta$, as required.

  ii) $\neg\Box\theta$ has been blocked by the application of symmetry blocking. In this case, $\neg\Box\theta = \bar{\sigma}(\neg\Box\chi) = \neg\Box\bar{\sigma}(\chi)$. By saturation of $(\Diamond)$ we have that $\alpha R \alpha'$, $\alpha'{:}\chi \in \Theta$. Moreover, we have that $(\alpha, \alpha') \in R^{\Theta}$ and that $\mathcal{M}^{\Theta}, \alpha' \models \chi$. By definition of the symmetric extension of $\mathcal{M}^{\Theta}$ we have that $(\alpha, \tau_{\alpha'}) \in R^{\Theta}_{\bar{\sigma}}$ and $\mathcal{M}^{\Theta}_{\bar{\sigma}}, \tau_{\alpha'} \models \bar{\sigma}(\chi)$. Which implies that $\mathcal{M}^{\Theta}_{\bar{\sigma}}, \alpha \models \neg\Box\bar{\sigma}(\chi) = \neg\Box\theta$.

- Suppose $\varphi = \Box\theta$. If there is no state α' such that $(\alpha, \alpha') \in R^{\Theta}_{\bar{\sigma}}$ then this holds trivially. Otherwise, let α' be such that $(\alpha, \alpha') \in R^{\Theta}_{\bar{\sigma}}$. By definition of $R^{\Theta}_{\bar{\sigma}}$ it must be the case that $\alpha{:}\neg\Box\chi \in \Theta$ and $\alpha R \alpha' \in \Theta$. We must consider two cases:

  i) if $\alpha{:}\neg\Box\chi$ has not a symmetric counterpart, i.e., it is not blocking a formula $\alpha : \neg\Box\bar{\sigma}(\chi)$ then, given that $\alpha : \Box\theta \in \Theta$, by saturation of $(\Box)$, we have that $\alpha'{:}\theta \in \Theta$. By inductive hypothesis, we have that $\mathcal{M}^{\Theta}_{\bar{\sigma}}, \alpha' \models \theta$. From this it follows that $\mathcal{M}^{\Theta}_{\bar{\sigma}}, \alpha \models \Box\theta$ as required.

  ii) If it is the case that $\alpha{:}\neg\Box\chi$ is blocking $\alpha{:}\neg\Box\bar{\sigma}(\chi)$, then, by the definition of the symmetric extension $\mathcal{M}^{\Theta}_{\bar{\sigma}}$ and Lemma 9.1, we have that $(\alpha, \tau_{\alpha'}) \in R^{\Theta}_{\bar{\sigma}}$ and $\mathcal{M}^{\Theta}_{\bar{\sigma}}, \tau_{\alpha'} \models \bar{\sigma}(\chi) \wedge \Gamma(\alpha)$. Given that $\theta \in \Gamma(\alpha)$ then, $\mathcal{M}^{\Theta}_{\bar{\sigma}}, \tau_{\alpha'} \models \theta$. From what it follows that $\mathcal{M}^{\Theta}_{\bar{\sigma}}, \alpha \models \Box\theta$ as required.

□

Theorem 9.2. *The tableau calculus with symmetry blocking for the basic modal logic is complete.*

Proof. Let Θ be an open saturated branch of the tableau $\mathsf{T} \in \mathsf{Tab}(\varphi)$. Since $0 : \varphi \in \Theta$, by Lemma 9.2 we get that φ is satisfiable. □

9.3 EXPERIMENTAL EVALUATION

We now present empirical results about how symmetry blocking performs in modal benchmarks.

9.3.1 *Implementation*

To test the effects of symmetry blocking (SB) in modal tableaux we implemented it in the tableau prover HTab [Hoffmann and Areces, 2007]. Implementation is straightforward: whenever there is a $\neg\Box$-formula scheduled for expansion, the solver checks if there is a symmetric formula already expanded. If this is the case, it blocks the $\neg\Box$-formula and continues with the application of the remaining rules. The solver only verifies the blocking condition if it gets a saturated open branch. If the blocking condition holds for all blocked formulas the solver terminates. Otherwise it reschedules formulas for further expansion. Symmetry information is provided as an additional input along with the formula.

9.3.2 *Results*

Our test set includes 954 symmetric instances from the Logics Workbench Benchmark for K(LWB_K) [Balsiger *et al.*, 2000] and QBF-LIB benchmarks [Giunchiglia *et al.*, 2001]. Problems from the QBF-LIB benchmarks were translated to the basic modal logic using the qbf2ml[1] tool using the *Collapse*1 translation, a variant of Ladner's translation [Ladner, 1977], that reduce the modal depth of the resulting modal formula yielding smaller formulas than Ladner's translation (see Appendix B for a detailed description of the translation). All tests were ran on an Intel Core i7 2.93GHz with 16GB of RAM with a timeout of 600 seconds.

Table 9.1 presents the results with and without symmetry blocking (HTab+SB and HTab, respectively). Columns *#Suc* and *#To* are the number of instances for which the solver succeeded to establish their status and the number of instances that timeouted respectively. Columns T_1 and T_2 are total times (including symmetry computation when pertinent), in seconds, on the complete test set, including and excluding timeouts, respectively. The table shows that HTab+SB outperforms HTab: HTab+SB requires less time to solve all the instances and solves 7 instances more than HTab (HTab+SB is able to solve 9 instances that timeout with HTab, but timeouts in other 2 that HTab is able to solve). It also shows that a large number of formulas timeouted. Most of them are formulas coming from the QBF-LIB, that, due to the translation to the basic modal logic, resulted in large modal formulas.

Solver	*#Suc*	*#To*	T_1	T_2
HTab+SB	318	636	9657	391167
HTab	311	643	10634	396434

Table 9.1: Total Times with (HTab+SB) and without (HTab) symmetry blocking.

Figure 9.7 presents a scatter plot of the running times for the 320 formulas that succeed in at least one of the configurations. The x axis gives the running times of HTab without symmetry blocking, whereas the y axis gives the running times of HTab+SB. Each point represents an instance and its horizontal and vertical coordinates represent the time necessary to solve it in seconds. Points below the diagonal

1 Download from:http://cs.famaf.unc.edu.ar/~ezequiel/resource/qbf2ml

Status	*#In*	*#Trig*	B_1	B_2
Satisfiable	157	73	6319	6278
Unsatisfiable	163	79	1038	87

Table 9.2: Symmetry blocking applications.

represent instances where HTab+SB outperforms HTab, while points above the diagonal represent instances where HTab outperforms HTab+SB. Points on the rightmost and topmost edges represent timeout. Notice that a logscale is used, so that gain or degradation to the far right and far top are exponentially more relevant. The figure shows that HTab+SB outperforms HTab. Approximately half of the instances report a performance gain while the other half report a slight performance degradation. However, a deeper look at Figure 9.7 shows us that most of the performance gain is obtained for instances that are not trivial to solve, and that we obtain a gain of several orders of magnitude for many instances. While, performance degradation is reported mostly for trivial instances (i. e., instances that take less than 1 second to solve). In these cases, degradation is due to the extra overhead imposed by the blocking mechanism on instances that never trigger symmetry blocking. Nevertheless, degradation is negligible for most of the instances.

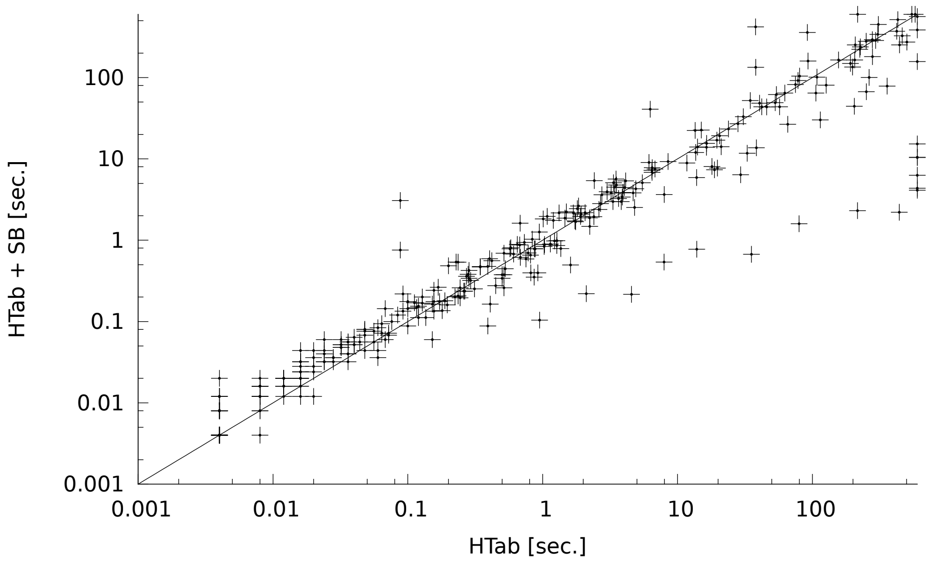


Figure 9.7: Performance of HTab vs. HTab+SB on all formulas.

We classify the 320 instances that succeeded in at least one of the configurations into satisfiable and unsatisfiable instances to better understand the effect of symmetry blocking. Table 9.2 shows information about the application of symmetry blocking on each category of instances. Columns *#In* and *#Trig* are the number of instances in each test set and the number of instances that trigger symmetry blocking at least once respectively. Columns B_1 and B_2 are the number of times that symmetry blocking is triggered and the number of times that a formula was initially blocked but dynamically rescheduled, respectively. For both categories, we observe that symmetry blocking triggers roughly on half of the instances (46% of the satisfiable instances and 48% of the unsatisfiable instances). For satisfiable instances, symmetry blocking triggers many times but in most cases the blocking condition fails later in the branch (remember that the blocking condition is dynamic). For unsatisfiable instances, symmetry blocking triggers less often than for satisfiable instances, but most of the blockings are correct (but remember that symmetry blocking is not validated if the branch closes).

Figures 9.8 and 9.9 show the same data that Figure 9.7 but differentiated into satisfiable and unsatisfiable instances. Figure 9.8 shows that for satisfiable instances HTab+SB outperforms HTab despite that most of the times the blocking condition fails later in the branch. This can be explained by the fact that in many cases where the blocking condition fails, delaying the processing of symmetric formulas is beneficial because the branch has more information available that can avoid branching or close the branch more rapidly. We also can observe that degradation in performance due to this overhead is almost negligible.

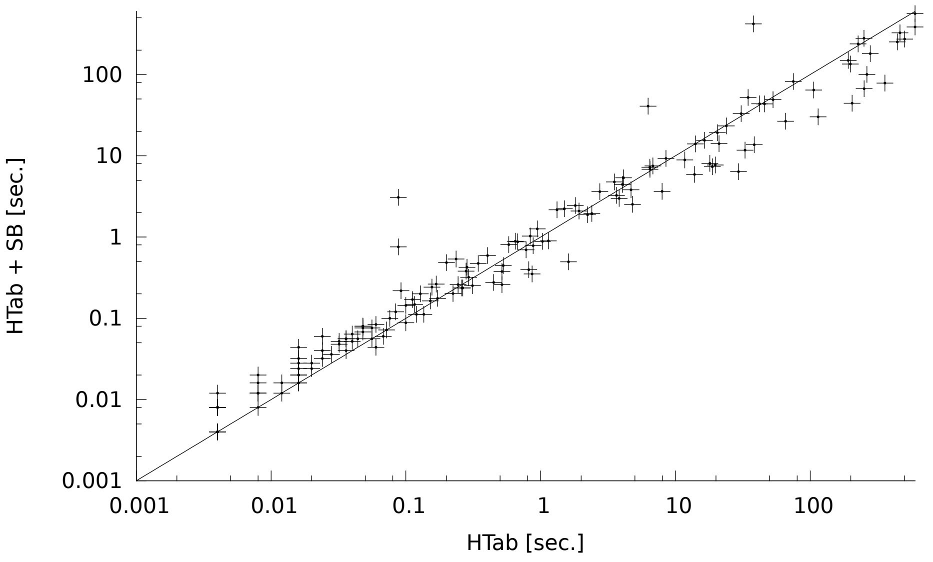


Figure 9.8: Performance of HTab vs. HTab+SB: Satisfiable formulas.

Figure 9.9 shows that for unsatisfiable instances HTab+SB achieves major performance gains for many instances. Also HTab+SB proves 7 more instances than HTab. Degradation is also more noticeable for some instances. A possible explanation is the following: if the blocked formula plays no role in the unsatisfiability of the problem, blocking it avoids unnecessary work resulting in a performance gain. If it plays a role in the unsatisfiability and its processing is delayed by blocking, the solver might be forced to process formulas that would not be processed otherwise.

From the data obtained in our tests, it is clear that the effectiveness of symmetry blocking is highly dependent on the problem class. For some classes symmetry blocking provides an important performance gain (e.g., the k_branch problem class from the LWB_K). On the other hand, for several highly symmetric classes, symmetry blocking does not make a difference as it never gets triggered. In other words, symmetric blocking only addresses a subset of the symmetries usually present in modal formulas.

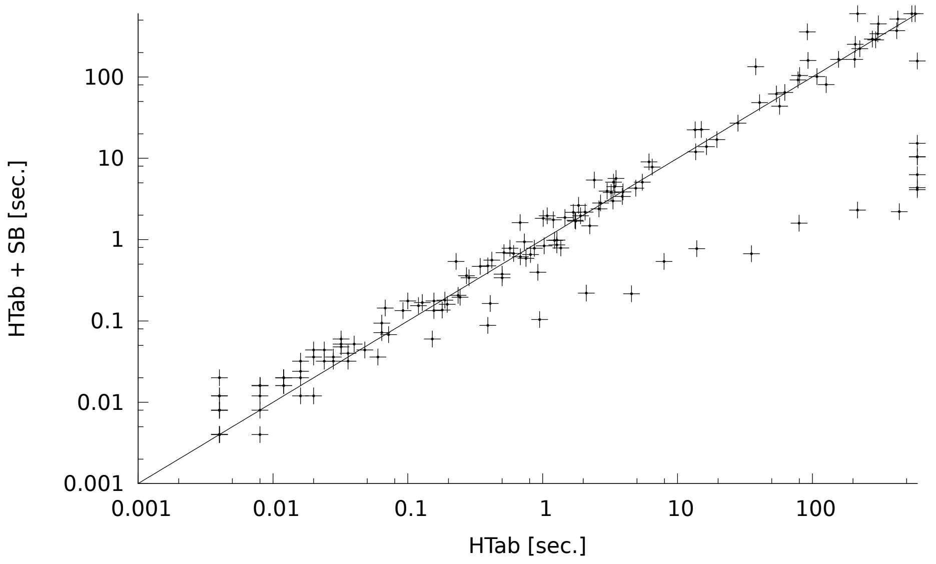


Figure 9.9: Performance of HTab vs. HTab+SB: Unsatisfiable formulas.

9.4 SUMMARY

In this chapter we have presented a tableau calculus that incorporates blocking mechanism, called *symmetry blocking*, that uses symmetry information of the input formula. The idea is to restrict the application of the $(\Diamond)$ rule to a $\neg\Box$-formula if the rule was prior applied to a symmetric $\neg\Box$-formula. We proved that the tableau calculus with symmetry blocking is complete and tested on several modal benchmarks. The experimental evaluation was done by implementing symmetry blocking on the modal tableau prover HTab. The results showed that using symmetry blocking we obtain important performance gains for many instances, showing the applicability of the blocking mechanism.

Part III

CONCLUSIONS

10 FINAL THOUGHTS AND FUTURE WORK

It is time to sum up the work we have done in this thesis. As already mentioned, the goal was to investigate symmetries in the context of modal logics and Satisfiability Modulo Theories (SMT). Let us review our contributions in each of these domains.

10.1 SYMMETRIES IN MODAL LOGICS

As the keen reader has already noticed, the study of symmetries in modal logics is the main topic of this thesis. We began by developing the theoretical foundations for exploiting symmetries in modal logics and proved two key results for the basic modal logic. First, we showed that the symmetries (or *global symmetries*) of a basic modal formula partition the model space into equivalence classes such that each equivalence class contains only models or only non-models. Second, we showed that symmetries can be used as an inference mechanism, and therefore, they can be used to strengthen existing reasoning mechanisms. We then extended these results to a broad range of modal logics using the framework of coinductive modal models and introduced a more flexible notion of symmetry, called *layered symmetries*, for those modal logics that have the tree model property.

Next, we turned our attention to the subject of detecting symmetries in modal formulas. To do so, we extended existing techniques in propositional logic that reduce the problem of detecting symmetries in a formula to the problem of detecting automorphisms in a graph built from it. We presented two reduction algorithms for creating graphs from modal formulas, one for detecting *global* symmetries and another for detecting *layered* symmetries. We also proved that the algorithms are correct, and therefore, that every detected automorphism of the graph corresponds to a symmetry of the formula from which the graph was built. Both algorithms were developed in the context of coinductive models, providing us with a template from which to derive concrete implementations for concrete modal logics. We implemented these reduction algorithms for the basic modal logic and tested them in modal benchmarks to see how often symmetries arise in modal formulas and how hard it is to find them. Experimental results revealed that symmetries do exist, in great number, in modal benchmarks, and that their presence is highly related to the codification of the problem. They also showed that detecting symmetries is fairly cheap even for large formulas.

Finally, we put symmetries to work by implementing a tableau calculus for the basic modal logic, that incorporates a blocking mechanism, called *symmetry blocking*, that uses symmetry information of the input formula to restrict the application of the ($\Diamond$) rule. The idea is to apply the ($\Diamond$) rule to a $\neg\Box$-formula only if the rule was not already applied to a symmetric $\neg\Box$-formula. We proved that the tableau calculus with symmetry blocking is complete, implemented the calculus in the modal tableau prover `HTab`, and tested it on several modal benchmarks. Experimental results showed that using the symmetry blocking mechanism we obtain important performance gains for many instances. Data also showed that the blocking mechanism is not always triggered in many formulas having symmetries. This indicates that other techniques to use symmetries need to be developed to fully profit from them.

The results presented here, both theoretical and experimental, should be considered as a first step towards the development of a more comprehensive treatment of symmetries in modal logics, as there is much room for improvement both in the theoretical and in the experimental side.

Let us now briefly describe what we consider to be interesting paths for future research.

In the theoretical side, it should be possible to develop a theory of symmetries for modal logics that cannot be represented using the coinductive framework and for logics that are notational variants of modal logics, e. g., Description Logics.

In this thesis, we only deal with propositional symmetries (atom symmetries in the coinductive framework) that permute propositional (atom) literals only. It should be possible to define a notion of symmetry that involves modal literals as well. Notice that a permutation of propositional (atom) literals implies a permutation of modal literals. However, it is possible to find symmetric modal literals that are not implied by symmetric propositional literals. A possible path to answer this question could be to consider the *propositional abstraction* of a modal formula, as is done in SMT, and look for symmetries in it.

On the experimental side, there is much work to do when it comes to exploiting symmetries in modal reasoning. One path is to investigate how to develop symmetry breaking predicates (SBP's) for modal formulas as is done for propositional formulas. Currently, we have developed an SBP construction for the basic modal logic similar to the one done in propositional logic [Aloul *et al.*, 2003b]. Our idea is to identify *independent propositional sub-problems* at each modal depth of the formula and generate SBP's if there are symmetric literals present in those sub-problems. Preliminary experimental results shows that for some instances it is possible to identify independent propositional sub-problems and that the SBP's have a positive effect on performance. However, the conditions that drive the detection of independent propositional sub-problems are to restrictive and left out most of the symmetries of a formula. We are now investigating how to relax these conditions and how to construct new types of symmetry breaking predicates.

An interesting path to explore is the use of symmetries in resolution-based modal provers. Key to this is the use of symmetries as a inference mechanisms. In particular, a direct application to modal logics of the symmetry rule introduced in [Krishnamurthy, 1985] for propositional logic should be possible, with minor modifications.

An alternative that deserves some attention is the use of symmetries as a simplification mechanism, i. e., using symmetries to simplify the input formula by removing symmetric subformulas. Notice that the idea is similar in spirit to what is done when we use symmetry blocking, but now, instead of blocking the formula dynamically during search, we do it in a pre-processing stage before feeding the formula to a solver.

Finally, a word of warning: due to the variety of modal logics, it seems hard to find a "one-fit-all" method to exploit symmetries. Instead, it is more likely that tailor-suited methods for each modal logic, and even, for each problem class, lead to better results.

10.2 SYMMETRIES IN SATISFIABILITY MODULO THEORIES

In the context of Satisfiability Modulo Theories (SMT), in this thesis we focused in the development of a new symmetry detection technique. To do so, we extended the graph-based techniques used for propositional logic to SMT and presented a reduction algorithm to create graphs from SMT formulas. We implemented this algorithm in the tool SyMT and tested it on several SMT-LIB benchmarks. Experimental results showed that using this symmetry detection technique we are now able to find symmetries that prior techniques are not able to find.

The detection algorithm is valuable in itself. As a formula inspection tool, SyMT can help users to identify the symmetries in a formula and eliminate them improving the codification of the problems.

There is still room also for improvement on the detection technique and the tool itself. As future work it would be interesting to tailor the Schreir-Sims algorithm to our use (detecting full subgroups) to improve efficiency of this systematic method. In many cases, symmetry groups contain subgroups that are full permutations of tuples of symbols. Improving presentation of those cases would provide better feedback for users. Finally, the tool currently finds only symmetries involving permutations of uninterpreted symbols: this, of course, does not cover all possible symmetries. For instance, the `queens` benchmarks have symmetries that involve arithmetic reasoning.

Similarly to what happens in modal logics, there is still much work to do when it comes to use symmetries in SMT solving. First, the development of symmetry breaking predicates should be investigated. Preliminary results show that, just like for the propositional case, it comes out in the context of SMT that automatic symmetry breaking is not a silver bullet. More testing should be done in this direction. We believe that, for most cases, user expertise is required to design good symmetry breaking formulas, and that the design of this formulas would depend on the problem class and theory at hand.

Another direction of research, is to investigate the effects of using symmetries as an inference mechanism in an SMT solver similarly to what is done in [Benhamou *et al.*, 2010]. In a similar vein, the use of symmetry information in each theory solver should also be considered.

Part IV

APPENDIX

A GROUP THEORY

The study of symmetries in automated reasoning is greatly facilitated by understanding some basic notions from the field of group theory. In this appendix we cover only the basic aspects of group theory that are necessary for the development of the topics in this thesis. For a more comprehensive treatment of this topic we refer the reader to [Fraleigh and Katz, 2003; Seress, 1997].

A.1 GROUPS

We begin with the definition of a group as an abstract algebraic structure with certain properties.

Definition A.1 (Group). *A group is a structure $\langle G, * \rangle$ where G is a (non-empty) set that is closed under a binary operation $*$, such that the following axioms are satisfied:*

- *The operation is* associative*: for all $x, y, z \in G$, $(x * y) * z = x * (y * z)$.*
- *There is an* identity *element $e \in G$ such that for all $x \in G$, $e * x = x * e = x$.*
- *Every element $x \in G$ has an inverse $x^{-1} \in G$ such that: $x * x^{-1} = x^{-1} * x = e$.*

We will tipically refer to the group G, rather than to the structure $\langle G, * \rangle$, with the understanding that there is an associated binary operation on the set G. Furthermore, it is customary to write xy instead of $x * y$ and x^2 instead of $x * x$ muck like we do with the multiplication operation on numbers.

This definition leads to a number of interesting properties that are easy to prove, including:

- **Uniqueness of the identity**: there is only one elemente $e \in G$ such that $e * x = x * e = x$ for all $x \in G$.
- **Uniqueness of the inverse**: for each $x \in G$, there is only one element $x^{-1} \in G$ such that $x * x^{-1} = x^{-1} * x = e$.

The definition does not restrict a group to be finite. However, for our purposes it is sufficient to focus on finite groups.

Definition A.2 (Group Order). *If G is a finite group, its* order $|G|$ *is the number of elements in (i.e., the cardinality of) the set G.*

Definition A.3 (Group Isomorphism). *Let $\langle G, * \rangle$ and $\langle G', *' \rangle$ be two groups. An* isomorphism *of G with G' is a one-to-one function ϕ mapping G onto G' such that:*

$$\phi(x * y) = \phi(x) *' \phi(y) \text{ for all } x, y \in G.$$

If such a map exists, then G and G' are isomorphic groups *which we denote by $G \simeq G'$.*

A.2 SUBGROUPS

Subgroups provide a way to understand the structure of groups.

Definition A.4 (Subgroup). *A non-empty subset H of a group G that is closed under the binary operation of G is itself a group and is referred to as a* subgroup *of G. We indicate that H is a subgroup of G by writing $H \leq G$. Additionally, $H < G$ shall mean that $H \leq G$ but $H \neq G$.*

Definition A.5 (Proper and Trivial subgroups). *If G is a group, then the subgroup consisting of G itself is the* improper subgroup *of G. All other subgroups are* proper subgroups*. The group $\{e\}$ is the* trivial subgroup *of G. All other subgroups are* nontrivial.

A subgroup H of a group G induces a partition of G whose cells are referred to as the *cosets* of H:

Definition A.6 (Cosets). *Let H be a subgroup of a group G. The* left coset *of H containing $x \in G$ is the subset $xH = \{xy \mid y \in H\}$ and the* right coset *of H containing $x \in G$ is the subset $Hx = \{yx \mid y \in H\}$.*

It is easily shown that the number of elements in each coset of a subgroup H of a group G is the same as the number of elements of H. This immediately leads to the following fundamental theorem of group theory:

Theorem A.1 (Theorem of Lagrange). *Let H be a subgroup of a finite group G. Then $|H|$ is divisor of $|G|$.*

A.3 GROUP GENERATORS

Theorem A.2 (Cyclic Subgroup). *Let G be a group and let $x \in G$. Then*

$$H = \{x^n \mid n \in \mathbb{Z}\},$$

i. e., the set of all (not necessarily distinct) powers of x, is a subgroup of G and is the smallest subgroup of G that contains x. This is referred to as the cyclic subgroup of G generated by *x, and is denoted by $\langle x \rangle$.*

Definition A.7 (Generator and Cyclic Group). *An element x of a group G* generates *G and is a* generator *for G if $\langle x \rangle = G$. A group G is* cyclic *if there is some element $x \in G$ that* generates *G.*

Definition A.8 (Group Generators). *Let G be a group and let $H \subset G$ be a subset of the group elements. The smallest subgroup of G containing H is the* subgroup generated by *H. If this subgroup is all of G, then H* generates *G and the elements of H are* generators *of G. A generator is* redundant *if it can be expressed in terms of other generators. A set of generators for a group G is* irredundant *if it does not contain redundant generators.*

An important property of irredundant generators sets of a finite group is that they provide an extremely compact representation of the group. This follows directly from the Theorem of Lagrange (Theorem A.1) and the definition of irredundant generator sets.

Theorem A.3. *Any irredundant set of generators for a finite group G, such that $|G| > 1$, contains at most $log_2|G|$ elements.*

Proof. Let $\{x_i \in G \mid 1 \leq i \leq n\}$ be a set of n irredundant generators for group G and consider the sequence of subgroups $G_1, G_2, \ldots, G_n$ such that group G_i is generated by $\{x_j \in G \mid 1 \leq j \leq i\}$. Clearly $G_1 < G_2 < \ldots < G_n$ because of our assumption that the generators are irredundant. By the Theorem of Lagrange, $|G_n| \geq 2|G_{n-1}| \geq \ldots \geq 2|G_2| \geq 2|G_1|$, i.e., $|G_n| \geq 2^{n-1}|G_1|$. Noting that $G_n = G$ and that $|G_1| \geq 2$, we get $|G| \geq 2^n$. Thus $n \leq log_2|G|$. □

A.4 PERMUTATION GROUPS

We turn now to an important class of groups, namely groups of permutations.

Definition A.9 (Permutation)**.** *A permutation of a set A is a function $\phi : A \mapsto A$ that is both one to one and onto (a bijection).*

Permutations can be "multiplied" using function composition:

$$\sigma\tau(a) \equiv (\sigma \circ \tau)(a) = \sigma(\tau(a))$$

where σ and τ are two permutations of A and $a \in A$. The resulting "product" is also a permutation of A, since it is easily shown to be one to one and onto. This lead us to the following result.

Theorem A.4 (Permutation Group)**.** *Given a non-empty set A, let S_A be the set of all permutations of A. Then S_A is a group under permutation multiplication.*

Definition A.10 (Symmetric Group)**.** *The group of all permutations of the set $\{1, 2, \ldots, n\}$, $n \geq 1$, is the* symmetric group on n letters*, and is denoted by S_n.*

Note that, in general, permutation multiplication is not conmutative, i.e., $\sigma\tau \neq \tau\sigma$.

Finite permutations can be presented in a *tabular format* that explicitly shows how each element of the underlying set is mapped to distinct element of the same set. For instance, $\sigma = \left(\begin{smallmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 8 & 2 & 6 & 4 & 5 & 3 & 7 & 1 \end{smallmatrix}\right)$ is the permutation that maps the element 1 to 8: $\sigma(1) = 8$. This explicit format becomes cumbersome, however, when the cardinality of the set A is large, and especially when a permutation maps many elements of A to themselves. In such cases, the more compact *cycle notation* is preferred. Using cycle notation, the permutation $\sigma = \left(\begin{smallmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 8 & 2 & 6 & 4 & 5 & 3 & 7 & 1 \end{smallmatrix}\right)$ can be expressed succintly as $\sigma = (1\ 8)(3\ 6)$. This permutation is a product of *disjoint* cycles, where a cycle $(a\ b\ c\ \ldots\ z)$ is understood to mean that the permutation maps a to b, b to c, and so on, finally mapping the last element in the cycle z back to a. An element that does not appear in a cycle is understood to be left fixed (mapped to itself) by the permutation. The *length* of a cycle is the number of elements in the cycle, a cycle with k elements will be referred to as a k-cycle.

Definition A.11 (Permutation Support)**.** *Given a permutation σ of a set A, its support consist of those elements of A that are not mapped to themselves by σ:*

$$supp(\sigma) = \{a \in A \mid \sigma(a) \neq a\}.$$

In other words, $supp(\sigma)$ are those elements of A that appear in σ's cycle representation.

B FROM QBF TO MODAL LOGIC

In this appendix we show how to translate QBF formulas in prenex form to modal formulas.

B.1 DEFINITIONS

Definition B.1 (QBF formulas). *The set of quantified boolean formulas is the smallest set S containing all formulas of propositional calculus such that if $\varphi \in S$ and p is a propositional variable, then both $\forall p \varphi$ and $\exists p \varphi \in S$. The quantifiers range over the truth values* 1 *(true) and* 0 *(false), and a quantified boolean formula without free variables is* valid *if and only if it always evaluates to* 1. *A QBF formula is said to be in* prenex normal form *if it is of the form $\varphi = Q_1 p_1 \dots Q_m p_m \theta(p_1, \dots, p_m)$, with $Q_i \in \{\forall, \exists\}$ and $\theta(p_1, \dots, p_m)$ a formula of propositional logic. Moreover, we assume that all p_i's are different. A QBF formula is said to be in* Conjunctive Normal Form *(CNF) if $\theta(p_1, \dots, p_m)$ is a CNF formula. From now on, we assume that all QBF formulas are in prenex normal form and in Conjunctive Normal Form.*

Given $i, j \in \mathbb{N}, i \leq j$ we define

$$\Box^{(i)}\varphi = \varphi \wedge \Box\varphi \wedge \Box^2\varphi \wedge \dots \wedge \Box^{i-1}\varphi \wedge \Box^i\varphi,$$

and,

$$\Box^{(i,j)}\varphi = \Box^i\varphi \wedge \Box^{i+1}\varphi \wedge \dots \wedge \Box^j\varphi.$$

Let us now define the semantics of QBF formulas.

Definition B.2 (QBF semantics). *Let φ and ψ be a QBF formulas with free variables $z_1, \dots, z_n$, and $\mathcal{F}$ a mapping $\mathcal{F} : \{z_1, \dots, z_n\} \mapsto \{0, 1\}$. Then the satisfaction relation $\mathcal{F} \models \varphi$ is defined as:*

$\mathcal{F} \models z_i$	*iff*	$\mathcal{F}(z_i) = 1$ *for* $1 \leq i \leq n$.
$\mathcal{F} \models \neg\varphi$	*iff*	$\mathcal{F}(\varphi) = 0$.
$\mathcal{F} \models \varphi \vee \psi$	*iff*	$\mathcal{F}(\varphi) = 1$ *or* $\mathcal{F}(\psi) = 1$.
$\mathcal{F} \models \varphi \wedge \psi$	*iff*	$\mathcal{F}(\varphi) = 1$ *and* $\mathcal{F}(\psi) = 1$.
$\mathcal{F} \models \exists y \varphi$	*iff*	$\mathcal{F}(\varphi[y/0]) = 1$ *or* $\mathcal{F}(\varphi[y/1]) = 1$.
$\mathcal{F} \models \forall x \varphi$	*iff*	$\mathcal{F}(\varphi[x/0]) = 1$ *and* $\mathcal{F}(\varphi[x/1]) = 1$.

By $\varphi[z_1/a_1, \dots, z_n/a_n]$ we denote the simultaneous substitution of free occurrences of z_i by a_i in φ.

B.2 EVALUATING QBF FORMULAS

Before presenting the translations it is important to understand how to evaluate a QBF formula. Given a prenex QBF formula, we start by peeling off the outermost quantifier. If it is of the form $\exists p$ we choose one of the truth values 1 or 0 and substitute for the newly freed occurrences of p. On the other hand, if it is of the form $\forall p$ we must substitute both 1 and 0 for the newly freed occurrences of p. In this fashion, we work our way succesively through the prefixed list of quantifiers until we reach the matrix, a formula of propositional logic. Abstractly considered we are generating a tree. This tree consists of the root node, and then, working inwards along the quantifier string, each existential quantifier extends it by adding a single branch, and each universal quantifier extends it by adding two branches. Indeed, we are even generating an annotated tree: we can label each node with the substitution it records.

Example B.1. *Consider the QBF formula $\forall p \exists q (p \leftrightarrow \neg q)$. Figure B.1 shows the resulting annotated tree.*

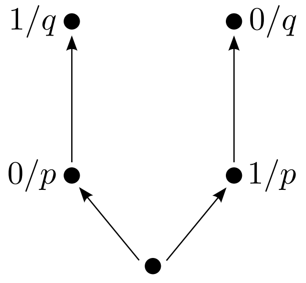


Figure B.1: Quantifier tree for $\forall p \exists q (p \leftrightarrow \neg q)$.

The information in such annotated trees, that we call *quantifier trees* will play a crucial role. For a start, QBF-validity is witnessed by certain quantifier trees: φ is a QBF-validity if and only if there is a quantifier tree for φ such that the substitutions it records ensure that the matrix evaluates to 1. Moreover, quantifier trees give us a bridge between the QBF world and the modal world: we are going to build a modal formula $f_L(\varphi)$ that describes the structure of a quantifier tree evaluating φ. That is, $f_L(\varphi)$ will describe the peel-of-quantifiers-and-substitute evaluation process for φ, i. e., it describes how to generate the quantifier tree for φ.

B.3 LADNER'S TRANSLATION

The first translation from QBF to the basic modal logic was introduced in [Ladner, 1977] and it is known as the Ladner's translation.

Definition B.3 (Ladner's translation). *Given a QBF formula, $\varphi = Q_1 p_1 \dots Q_m p_m$ $\theta(p_1, \dots, p_m)$. We define $f_L(\varphi)$, its translation to basic modal logic, as the conjunction of the following formulas:*

$$\begin{array}{rll}
\textit{(Root)}: & q_0, & \\
\textit{(Level)}: & \bigwedge_{i \neq j} \Box^{(m)}(\neg q_i \vee \neg q_j) & \textit{for } 0 \le i \le m, \\
\textit{(Peel off)}: & \Box^{(m)}(\neg q_i \vee \neg \Box \neg q_{i+1}) & \textit{for } 0 \le i < m, \\
\textit{(Branch)}: & \bigwedge_{i | Q_{i+1} = \forall} B_i, & \\
\textit{(Propagate)}: & S(1, m-1) \wedge S(2, m-1) \wedge \dots \wedge S(m-1, m-1), & \\
\textit{(Matrix)}: & \bigwedge \Box^m(\neg q_m \vee C_j) & \textit{for all clauses } C_j \in \theta.
\end{array}$$

Where $q_0, \ldots, q_m$ are new propositional variables, B_i is defined as

$$B_i = \Box^i(\neg q_i \vee \neg\Box(\neg q_{i+1} \vee \neg p_{i+1})) \wedge \Box^i(\neg q_i \vee \neg\Box(\neg q_{i+1} \vee p_{i+1})), \tag{B.1}$$

and $S(i,j)$ as

$$S(i,j) = \Box^{(i,j)}(\neg p_i \vee \Box p_i) \wedge \Box^{(i,j)}(p_i \vee \Box\neg p_i). \tag{B.2}$$

Ladner's translation works by forcing a tree model such that at every leaf we have a complete valuation of the variables in the prefix and we evaluate the matrix there. The first conjunct, *(Root)*, ensures that every model satisfying the formula has a root node. The q_i variables play a supporting role. They are used by the *(Level)* conjuncts to mark the *level* in the model; that is, they mark the number of upward steps that need to be taken to reach the satisfying node. The *(Peel off)* conjuncts guarantees that if q_i is true and $i < m$ then there is a next level q_{i+1}; which simply amounts to saying that if $i < m$ then we have not yet peeled off all the quantifiers and a new level will be necessary. The *(Branch)* conjuncts assure that branching occurs at level i only if the quantifier Q_{i+1} is a $\forall$. To do so it uses the B_i formulas that force a *branching* to occur at level i by setting the value of p_{i+1} to true at one successor at level $i+1$, and setting p_{i+1} to false at another. The *(Propagate)* conjuncts assure that these newly set truth values are sent further down the tree up to the leaves. This is done by the $S(i,j)$ formulas that propagates the truth values assigned to p_i one level down. Finally, the *(Matrix)* conjunct insists that after m quantifiers have been peeled off, the propositional matrix θ must be true. Note that if the propositional matrix θ is in conjunctive normal form, then the resulting formula is in modal CNF. Clearly $f_L(\varphi)$ is polysize in $|\varphi|$, thus this translation causes no blowup in space requirements.

Despite of being polynomial, the resulting formulas are too big for empirical testing.

To overcome this problem, and produce smaller formulas, a number of translations have been proposed [Schmidt-Schauß and Smolka, 1991; Massacci, 1999; Heguiabehere and de Rijke, 2001], most of them variants of the Ladner's translation. For example, in [Heguiabehere and de Rijke, 2001] a translation is presented that does not use level variables and removes many redundant formulas from the original Ladner's translation.

Definition B.4 (Variant of Ladner's translation [Heguiabehere and de Rijke, 2001]). *Given a QBF formula, $\varphi = Q_1 p_1 \ldots Q_m p_m \theta(p_1, \ldots, p_m)$. We define $f_L(\varphi)$, its translation to basic modal logic, as the conjunction of the following formulas:*

(Peel off) :	$\Box^i\neg\Box\bot$	*for* $0 \leq i < m$,
(Branch) :	$\bigwedge_{i\mid Q_{i+1}=\forall} B_i$,	
(Propagate) :	$S(1,m-1) \wedge S(2,m-1) \wedge \ldots \wedge S(m-1,m-1)$,	
(Matrix) :	$\bigwedge \Box^m C_j$	*for all clauses* $C_j \in \theta$.

Where B_i is defined as

$$B_i = \Box^i\neg\Box\neg p_{i+1} \wedge \Box^i\neg\Box p_{i+1}, \tag{B.3}$$

and $S(i,j)$ as

$$S(i,j) = \Box^{(i,j)}(\neg p_i \vee \Box p_i) \wedge \Box^{(i,j)}(p_i \vee \Box\neg p_i). \tag{B.4}$$

B.4 COLLAPSE TRANSLATIONS

A problem with Ladner's translations is that the modal depth of the resulting formula depends on the number of variables in the prefix, v, and the size of the resulting formula is polynomial in v. For small values of v this is not a problem. However, for QBF formulas coming from real applications, the value of v can easily reach the hundreds of variables, thus yielding excessively large formulas.

To overcome this oversize problem, we want a translation that reduces the modal depth of the resulting formula. One way of doing so, is to modify the "peel off" scheme of Ladner's translation as follows: instead of building a modal formula that forces the creation of a new state for each existentially quantified variable (from now on $\exists$-variables) in a model satisfying it, we could build a formula that collapses all the information corresponding to consecutive $\exists$-variables into the states that the formula forces to be created for the closer universally quantified variable (from now on $\forall$-variable). In other words, we want a formula that only forces the creation of new states only for the $\forall$-quantifiers, forcing a binary tree model, and that forces each created state to have the information of the $\exists$-quantifiers. This is exactly what the following translations (Collapse1 and Collapse2) do.

Before defining them formally, let us define the needed notation.

Definition B.5 (Quantifier Sequences). *Let $\varphi = Q_1 p_1 \dots Q_m p_m \theta(p_1, \dots, p_m)$ be a prenex QBF formula. By $U(\varphi) = \langle Q_i \mid Q_i = \forall \text{ and } 1 \leq i \leq m \rangle$ we denote the sequence of $\forall$-quantifiers in the prefix of φ. By $E(\varphi) = \langle Q_i \mid Q_i = \exists \text{ and } 1 \leq i \leq m \rangle$ we denote the sequence of $\exists$-quantifiers in the prefix of φ. Given a sequence of quantifiers S, by S_i we denote the i-th quantifier in the sequence. S_1 denotes the first element of the sequence. By $|S|$ we denote the size of the sequence.*

Definition B.6 (Collapse Level). *For each existential quantifier $\exists_i$, we define a* collapse level, *$CL(\exists_i)$, which corresponds to the index of the* last *$\forall$-quantifier occurring before $\exists_i$.*

Now we are ready to present the first translation.

Definition B.7 (Collapse1 Translation). *Given a CNF QBF formula $\varphi = Q_1 p_1 \dots Q_m p_m$ $\theta(p_1, \dots, p_m)$, let $m = |U(\varphi)|$. We define $f_L(\varphi)$, its translation to basic modal logic, as the conjunction of the following formulas:*

(Root) :	$q_0,$	
(Level) :	$\bigwedge_{j \neq k} \Box^j(\neg q_j \vee \neg q_k)$	$0 \leq j, k \leq m,$
(Peel&Branch) :	$\bigwedge B(i,j)$	$0 \leq j < m$ *and* $U(\varphi)_{(j+1)} = \forall_i,$
(Propagate-$\forall$) :	$\bigwedge S(i,j,m-1)$	$1 \leq j \leq m-1$ *and* $U(\varphi)_j = \forall_i,$
(Propagate-$\exists$) :	$\bigwedge S(i,j,m-1)$	$1 \leq k \leq \lvert E(\varphi) \rvert$ *and* $E(\varphi)_k = \exists_i$ *and* $j = CL(\exists_k), j \leq m,$
(Matrix) :	$\bigwedge \Box^m(\neg q_m \vee C_j)$	*for all clauses* $C_j \in \theta.$

Where $B(i, j)$ is defined as

$$B(i, j) = \Box^j(\neg q_j \vee \neg\Box(\neg q_{j+1} \vee \neg p_i)) \wedge \Box^j(\neg q_j \vee \neg\Box(\neg q_{j+1} \vee p_i)), \tag{B.5}$$

and $S(i, j, k)$ as

$$S(i, j, k) = \Box^{(j,k)}(\neg p_i \vee \Box p_i) \wedge \Box^{(j,k)}(p_i \vee \Box\neg p_i). \tag{B.6}$$

A variation of the Collapse1 translation is the Collapse2 translation that uses no auxiliary variables.

Definition B.8 (Collapse2 Translation). *Given a CNF QBF formula $\varphi = Q_1 p_1 \ldots Q_m p_m$ $\theta(p_1, \ldots, p_m)$, let $m = |U(\varphi)|$. We define $f_L(\varphi)$, its translation to basic modal logic, as the conjunction of the following formulas:*

(Peel&Branch) :	$\bigwedge B(i, j)$	$0 \leq j < m$ *and* $U(\varphi)_j = \forall_i$,
(Propagate-$\forall$) :	$\bigwedge S(i, j, m-1)$	$1 \leq j \leq m-1$ *and* $U(\varphi)_j = \forall_i$,
(Propagate-$\exists$) :	$\bigwedge S(i, j, m-1)$	$1 \leq k \leq \lvert E(\varphi)\rvert$ *and* $E(\varphi)_k = \exists_i$ *and* $j = CL(\exists_k), j \leq m$,
(Matrix) :	$\bigwedge \Box^m C_j$	*for all clauses* $C_j \in \theta$.

Where $B(i, j)$ is defined as

$$B(i, j) = \Box^j\neg\Box\neg p_i \wedge \Box^j\neg\Box p_i,$$

and $S(i, j, k)$ as

$$S(i, j, k) = \Box^{(j,k)}(\neg p_i \vee \Box p_i) \wedge \Box^{(j,k)}(p_i \vee \Box\neg p_i).$$

BIBLIOGRAPHY

[Aho and Hopcroft, 1974] A. Aho and J. Hopcroft. *The Design & Analysis of Computer Algorithms*. Pearson Education India, 1974. (Cited on page 83.)

[Aloul *et al.*, 2003a] F. Aloul, I. Markov, and K. Sakallah. Shatter: efficient symmetry-breaking for boolean satisfiability. In *Design Automation Conference.*, pages 836–839. IEEE, 2003. (Cited on pages 6 and 7.)

[Aloul *et al.*, 2003b] F. Aloul, A. Ramani, I. Markov, and K. Sakallah. Solving difficult instances of boolean satisfiability in the presence of symmetry. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(9):1117–1137, 2003. (Cited on pages 6, 7, 60, 86, 90, 93, and 142.)

[Aloul *et al.*, 2006] F. Aloul, K. Sakallah, and I. Markov. Efficient symmetry breaking for boolean satisfiability. *Computers, IEEE Transactions on*, 55(5):549–558, 2006. (Cited on pages 6 and 7.)

[Arai and Urquhart, 2000] N. Arai and A. Urquhart. Local symmetries in propositional logic. In *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 40–51, 2000. (Cited on page 4.)

[Areces and Gorín, 2010] C. Areces and D. Gorín. Coinductive models and normal forms for modal logics (or how we learned to stop worrying and love coinduction). *Journal of Applied Logic*, 8(4):305–318, 2010. (Cited on pages 32 and 35.)

[Areces and Heguiabehere, 2003] C. Areces and J. Heguiabehere. hGen: A Random CNF Formula Generator for Hybrid Languages. In *Methods for Modalities 3*, Nancy, France, 2003. (Cited on page 108.)

[Areces and Orbe, 2013] C. Areces and E. Orbe. Dealing with symmetries in modal tableaux. In D. Galmiche and D. Larchey-Wendling, editors, *TABLEAUX*, volume 8123 of *Lecture Notes in Computer Science*, pages 13–27. Springer, 2013. (Cited on pages 97 and 125.)

[Areces and ten Cate, 2006] C. Areces and B. ten Cate. Hybrid logics. In P. Blackburn, F. Wolter, and J. van Benthem, editors, *Handbook of Modal Logics*, pages 821–868. Elsevier, 2006. (Cited on page 35.)

[Areces *et al.*, 2000a] C. Areces, P. Blackburn, and M. Marx. The computational complexity of hybrid temporal logics. *Logic Journal of IGPL*, 8(5):653–679, 2000. (Cited on page 23.)

[Areces *et al.*, 2000b] C. Areces, R. Gennari, J. Heguiabehere, and M. de Rijke. Tree-based heuristics in modal theorem proving. In *Proceedings of ECAI'2000*, pages 199–203, Berlin, Germany, 2000. (Cited on page 43.)

[Areces *et al.*, 2012] C. Areces, G. Hoffmann, and E. Orbe. Symmetries in modal logics. In D. Kesner and P. Viana, editors, *LSFA*, volume 113 of *EPTCS*, pages 27–44, 2012. (Cited on pages 25 and 97.)

[Areces *et al.*, 2013] C. Areces, D. Deharbe, P. Fontaine, and E. Orbe. Symt: finding symmetries in smt formulas. In *11th International Workshop on Satisfiability Modulo Theories (SMT 2013)*, Helsinki, Finland, 07/2013 2013. (Cited on page 117.)

[Armando *et al.*, 2000] A. Armando, C. Castellini, and E. Giunchiglia. Sat-based procedures for temporal reasoning. In *Recent Advances in AI Planning*, pages 97–108. Springer, 2000. (Cited on pages 48 and 58.)

[Audemard *et al.*, 2002a] G. Audemard, P. Bertoli, A. Cimatti, A. Korniłowicz, and R. Sebastiani. A sat based approach for solving formulas over boolean and linear mathematical propositions. In *Automated Deduction (CADE-18)*, pages 195–210. Springer, 2002. (Cited on page 58.)

[Audemard *et al.*, 2002b] G. Audemard, A. Cimatti, A. Kornilowicz, and R. Sebastiani. Bounded model checking for timed systems. In *Formal Techniques for Networked and Distributed Sytems (FORTE 2002)*, pages 243–259. Springer, 2002. (Cited on pages 8, 48, and 60.)

[Audemard *et al.*, 2004] G. Audemard, B. Mazure, and L. Sais. Dealing with symmetries in quantified boolean formulas. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, pages 257–262, 2004. (Cited on pages 7 and 93.)

[Audemard *et al.*, 2005] G. Audemard, M. Bozzano, A. Cimatti, and R. Sebastiani. Verifying industrial hybrid systems with mathsat. *Electronic Notes in Theoretical Computer Science*, 119(2):17–32, 2005. (Cited on page 48.)

[Audemard *et al.*, 2006] G. Audemard, B. Benhamou, and L. Henocque. Predicting and detecting symmetries in fol finite model search. *Journal of Automated Reasoning*, 36(3):177–212, 2006. (Cited on page 95.)

[Audemard *et al.*, 2007a] G. Audemard, S. Jabbour, and L. Sais. Efficient symmetry breaking predicates for quantified boolean formulae. In *Proceedings of SymCon-Symmetry in Constraitns-CP workshop*. Citeseer, 2007. (Cited on page 7.)

[Audemard *et al.*, 2007b] G. Audemard, S. Jabbour, and L. Sais. Symmetry breaking in quantified boolean formulae. In *IJCAI*, pages 2262–2267, 2007. (Cited on page 7.)

[Bachmair *et al.*, 2003] L. Bachmair, A. Tiwari, and L. Vigneron. Abstract congruence closure. *Journal of Automated Reasoning*, 31(2):129–168, 2003. (Cited on page 52.)

[Balsiger *et al.*, 2000] P. Balsiger, A. Heuerding, and S. Schwendimann. A benchmark method for the propositional modal logics K, KT, S4. *Journal of Automated Reasoning*, 24(3):297–317, 2000. (Cited on pages 65, 107, and 134.)

[Barrett *et al.*, 2002] C. Barrett, D. Dill, and A. Stump. Checking satisfiability of first-order formulas by incremental translation to sat. In *Computer Aided Verification*, pages 236–249. Springer, 2002. (Cited on page 57.)

[Barrett *et al.*, 2005] C. Barrett, Y. Fang, B. Goldberg, Y. Hu, A. Pnueli, and L. Zuck. Tvoc: A translation validator for optimizing compilers. In *Computer Aided Verification*, pages 291–295. Springer, 2005. (Cited on page 48.)

[Barrett *et al.*, 2009] C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. *Satisfiability Modulo Theories*, chapter 26, pages 825–885. Volume 185 of Biere et al. [2009], February 2009. (Cited on page 47.)

[Barrett *et al.*, 2010a] C. Barrett, A. Stump, and C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2010. (Cited on pages 71 and 123.)

[Barrett *et al.*, 2010b] C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB Standard: Version 2.0. In A. Gupta and D. Kroening, editors, *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK)*, 2010. (Cited on page 71.)

[Barrett *et al.*, 2011] C. Barrett, C. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli. Cvc4. In *Computer Aided Verification*, pages 171–177. Springer, 2011. (Cited on page 48.)

[Benhamou and Sais, 1992] B. Benhamou and L. Sais. Theoretical study of symmetries in propositional calculus and applications. In *Automated Deduction (CADE-11)*, pages 281–294, 1992. (Cited on pages 4, 5, 7, 94, and 95.)

[Benhamou and Sais, 1994] B. Benhamou and L. Sais. Tractability through symmetries in propositional calculus. *Journal of Automated Reasoning*, 12(1):89–102, 1994. (Cited on pages 5, 7, and 94.)

[Benhamou *et al.*, 2010] B. Benhamou, T. Nabhani, R. Ostrowski, and M. Saidi. Enhancing clause learning by symmetry in SAT solvers. In *Proceedings of the 22nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 329–335, 2010. (Cited on pages 6, 31, and 143.)

[Beth, 1959] E. Beth. *The foundations of mathematics: a study in the philosophy of sciences*. North-Holland Amsterdam, 1959. (Cited on page 125.)

[Biere *et al.*, 2009] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009. (Cited on pages 159, 164, 166, and 168.)

[Blackburn *et al.*, 2001] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001. (Cited on pages 9, 13, 34, 35, 36, and 42.)

[Blackburn *et al.*, 2006] P. Blackburn, J. van Benthem, and F. Wolter. *Handbook of Modal Logic*, volume 3 of *Studies in Logic and Practical Reasoning*. Elsevier Science Inc., New York, NY, USA, 2006. (Cited on pages 9, 13, 21, and 125.)

[Booth and Colbourn, 1979] K. Booth and C. Colbourn. *Problems polynomially equivalent to graph isomorphism*. Computer Science Department, Univ., 1979. (Cited on page 82.)

[Borning *et al.*, 1997] A. Borning, K. Marriott, P. Stuckey, and Y. Xiao. Solving linear arithmetic constraints for user interface applications. In *Proceedings of the 10th annual ACM symposium on User interface software and technology*, pages 87–96. ACM, 1997. (Cited on page 52.)

[Bošnački *et al.*, 2002] D. Bošnački, D. Dams, and L. Holenderski. Symmetric spin. *International Journal on Software Tools for Technology Transfer*, 4(1):92–106, 2002. (Cited on page 7.)

[Bouton *et al.*, 2009] T. Bouton, D. De Oliveira, D. Déharbe, and P. Fontaine. verit: an open, trustable and efficient smt-solver. In *Automated Deduction (CADE-22)*, pages 151–156. Springer, 2009. (Cited on pages 8, 48, 60, and 95.)

[Bozzano *et al.*, 2006a] M. Bozzano, R. Bruttomesso, A. Cimatti, A. Franzén, Z. Hanna, Z. Khasidashvili, A. Palti, and R. Sebastiani. Encoding rtl constructs for mathsat: a preliminary report. *Electronic Notes in Theoretical Computer Science*, 144(2):3–14, 2006. (Cited on page 48.)

[Bozzano *et al.*, 2006b] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, S. Ranise, P. Van Rossum, and R. Sebastiani. Efficient theory combination via boolean search. *Information and Computation*, 204(10):1493–1525, 2006. (Cited on page 58.)

[Brading and Castellani, 2013] K. Brading and E. Castellani. Symmetry and symmetry breaking. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. The Stanford Encyclopedia of Philosophy, spring 2013 edition, 2013. (Cited on page 3.)

[Brown *et al.*, 1989] C. Brown, L. Finkelstein, and P. Purdom Jr. Backtrack searching in the presence of symmetry. In *Applied algebra, algebraic algorithms and error-correcting codes*, pages 99–110. Springer, 1989. (Cited on pages 5 and 7.)

[Bryant *et al.*, 2002] R. Bryant, S. Lahiri, and S. Seshia. Modeling and verifying systems using a logic of counter arithmetic with lambda expressions and uninterpreted functions. In *Computer Aided Verification*, pages 78–92. Springer, 2002. (Cited on page 48.)

[Burch and Dill, 1994] J. Burch and D. Dill. Automatic verification of pipelined microprocessor control. In *Computer Aided Verification*, pages 68–80. Springer, 1994. (Cited on page 48.)

[Buro and Büning, 1992] M. Buro and H. Büning. Report on a sat competition. Technical report, Fachbereich Math.-Informatik, Univ. Gesamthochschule, 1992. (Cited on page 66.)

[Cadoli *et al.*, 1998] M. Cadoli, A. Giovanardi, and M. Schaerf. An algorithm to evaluate quantified boolean formulae. *AAAI/IAAI*, 98:262–267, 1998. (Cited on page 70.)

[Cherkassky and Goldberg, 1999] B. Cherkassky and A. Goldberg. Negative-cycle detection algorithms. *Mathematical Programming*, 85(2):277–311, 1999. (Cited on pages 52 and 53.)

[Cimatti *et al.*, 2013] A. Cimatti, A. Griggio, B. Schaafsma, and R. Sebastiani. The mathsat5 smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 93–107. Springer, 2013. (Cited on page 48.)

[Clarke *et al.*, 1996] E. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9(1-2):77–104, 1996. (Cited on page 7.)

[Cohen *et al.*, 2005] D. Cohen, P. Jeavons, C.r Jefferson, K. Petrie, and B. Smith. Symmetry definitions for constraint satisfaction problems. In *Principles and Practice of Constraint Programming*, pages 17–31. Springer, 2005. (Cited on page 7.)

[Cohen *et al.*, 2009] M. Cohen, M. Dam, A. Lomuscio, and H. Qu. A symmetry reduction technique for model checking temporal-epistemic logic. In *IJCAI*, volume 9, pages 721–726, 2009. (Cited on page 7.)

[Cotton and Maler, 2006] S. Cotton and O. Maler. Fast and flexible difference constraint propagation for dpll (t). In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT'06)*, pages 170–183. Springer, 2006. (Cited on page 53.)

[Crawford *et al.*, 1996] J. Crawford, M. Ginsberg, E. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *Proceedings of KR 1996*, pages 148–159, 1996. (Cited on pages 5, 6, 7, 88, and 91.)

[Crawford, 1992] J. Crawford. A theoretical analysis of reasoning by symmetry in first-order logic. In *Proceedings of AAAI Workshop on Tractable Reasoning*, pages 17–22, San Jose, CA, 1992. (Cited on pages 5, 6, 7, and 87.)

[Cubadda, 1988] C. Cubadda. *Variantes de l'algorithmes de sl-résolution avec retenue d'informations*. PhD thesis, GIA Luminy (Marseille), 1988. (Cited on page 5.)

[D'Agostino, 1999] M. D'Agostino. *Handbook of tableau methods*. Springer, 1999. (Cited on page 125.)

[Darga *et al.*, 2004] P. Darga, M. Liffiton, K. Sakallah, and I. Markov. Exploiting structure in symmetry detection for CNF. In *Design Automation Conference.*, pages 530–534, 2004. (Cited on page 83.)

[Darga *et al.*, 2008] P. Darga, K. Sakallah, and I. Markov. Faster symmetry discovery using sparsity of symmetries. In *Proceedings of the 45th annual Design Automation Conference*, pages 149–154. ACM, 2008. (Cited on pages 83 and 86.)

[Darvas, 2007] G. Darvas. *Symmetry. Cultural-historical and ontological aspects of science-arts relations. The natural and man-made world in an interdisciplinary approach.* Basel: Birkhäuser, 2007. (Cited on page 3.)

[Davis and Putnam, 1960] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM (JACM)*, 7(3):201–215, 1960. (Cited on page 55.)

[Davis *et al.*, 1962] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962. (Cited on page 55.)

[De Moura and Bjørner, 2008] L. De Moura and N. Bjørner. Z3: An efficient smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008. (Cited on page 48.)

[De Moura *et al.*, 2002a] L. De Moura, H. Rueß, and M. Sorea. Lazy theorem proving for bounded model checking over infinite domains. In *Automated Deduction (CADE-18)*, pages 438–455. Springer, 2002. (Cited on page 48.)

[De Moura *et al.*, 2002b] L. De Moura, H. Rueß, and M. Sorea. Lemmas on demand for satisfiability solvers. In *Proceedings of the 5th International Symposium on the Theory and Applications of Satisfiability Testing (SAT'02)*. Citeseer, 2002. (Cited on page 57.)

[Déharbe and Ranise, 2003] D. Déharbe and S. Ranise. Light-weight theorem proving for debugging and verifying units of code. In *Proceedings of the First International Conference on Software Engineering and Formal Methods,*, pages 220–228. IEEE, 2003. (Cited on page 48.)

[Déharbe *et al.*, 2011] D. Déharbe, P. Fontaine, S. Merz, and B. Woltzenlogel Paleo. Exploiting symmetry in smt problems. In *Automated Deduction (CADE-23)*, volume 6803 of *Lecture Notes in Computer Science*, pages 222–236. Springer Berlin Heidelberg, 2011. (Cited on pages xiv, 8, 60, 61, 95, and 124.)

[Donaldson and Miller, 2005] A. Donaldson and A. Miller. Automatic symmetry detection for model checking using computational group theory. In *Formal Methods*, pages 481–496. Springer, 2005. (Cited on page 7.)

[Donaldson, 2007] A. Donaldson. *Automatic techniques for detecting and exploiting symmetry in model checking*. PhD thesis, University of Glasgow, 2007. (Cited on page 7.)

[Downey *et al.*, 1980] P. Downey, R. Sethi, and R. Tarjan. Variations on the common subexpression problem. *Journal of the ACM (JACM)*, 27(4):758–771, 1980. (Cited on page 52.)

[Dutertre and De Moura, 2006] B. Dutertre and L. De Moura. A fast linear-arithmetic solver for dpll (t). In *Computer Aided Verification*, pages 81–94. Springer, 2006. (Cited on page 52.)

[Ebbinghaus *et al.*, 1994] H. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical logic*. Springer, New York, 1994. (Cited on page 48.)

[Een and Sörensson, 2003] N. Een and N. Sörensson. An extensible sat-solver. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, pages 502–518, 2003. (Cited on page 6.)

[Enderton, 2001] H. Enderton. *A mathematical introduction to logic*. Academic Press, 2nd edition, 2001. (Cited on pages 48 and 75.)

[Fallah *et al.*, 1998] F. Fallah, S. Devadas, and K. Keutzer. Functional vector generation for hdl models using linear programming and 3-satisfiability. In *Design Automation Conference.*, pages 528–533. IEEE, 1998. (Cited on page 53.)

[Fitting, 1972] M. Fitting. Tableau methods of proof for modal logics. *Notre Dame Journal of Formal Logic*, 13(2):237–247, 1972. (Cited on page 127.)

[Fitting, 1983] M. Fitting. *Proof methods for modal and intuitionistic logics.* D. Reidel (Dordrecht, Holland and Boston, USA and Hingham, MA), 1983. (Cited on page 127.)

[Flanagan *et al.*, 2003] C. Flanagan, R. Joshi, X. Ou, and J. Saxe. Theorem proving using lazy proof explication. In *Computer Aided Verification*, pages 355–367. Springer, 2003. (Cited on page 58.)

[Fortin, 1996] S. Fortin. The graph isomorphism problem. Technical report, Technical Report 96-20, University of Alberta, Edomonton, Alberta, Canada, 1996. (Cited on page 82.)

[Fox and Long, 1999] M. Fox and D. Long. The detection and exploitation of symmetry in planning problems. In *IJCAI*, volume 99, pages 956–961, 1999. (Cited on page 7.)

[Fox and Long, 2002] M. Fox and D. Long. Extending the exploitation of symmetries in planning. In *AIPS*, pages 83–91, 2002. (Cited on page 7.)

[Fraleigh and Katz, 2003] J. Fraleigh and V. Katz. *A first course in abstract algebra.* Addison-Wesley world student series. Addison-Wesley, 2003. (Cited on pages 27 and 147.)

[Franzén, 2006] A. Franzén. Using satisfiability modulo theories for inductive verification of lustre programs. *Electronic Notes in Theoretical Computer Science*, 144(1):19–33, 2006. (Cited on page 48.)

[Gallier, 1985] J. Gallier. *Logic for computer science: foundations of automatic theorem proving.* Harper & Row Publishers, Inc., 1985. (Cited on page 75.)

[Ganesh and Dill, 2007] V. Ganesh and D. Dill. A decision procedure for bit-vectors and arrays. In *Computer Aided Verification*, pages 519–531. Springer, 2007. (Cited on page 53.)

[Ganzinger *et al.*, 2004] H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Dpll (t): Fast decision procedures. In *Computer aided verification*, pages 175–188. Springer, 2004. (Cited on page 58.)

[Gargov and Goranko, 1993] G. Gargov and V. Goranko. Modal logic with names. *Journal of Philosophical Logic*, 22(6):607–636, 1993. (Cited on page 22.)

[Gent *et al.*, 1996] I. Gent, E. MacIntyre, P. Prosser, and T. Walsh. The constrainedness of search. In W. Clancey and D. Weld, editors, *AAAI/IAAI, Vol. 1*, pages 246–252. AAAI Press / The MIT Press, 1996. (Cited on page 66.)

[Gent *et al.*, 2006] I. P Gent, K. Petrie, and J. Puget. Symmetry in constraint programming. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 10. Elsevier, 2006. (Cited on page 7.)

[Giunchiglia and Sebastiani, 1996a] F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures – the case study of modal k. In M.A. McRobbie and J.K. Slaney, editors, *Automated Deduction (CADE-13)*, volume 1104 of *Lecture Notes in Computer Science*, pages 583–597. Springer Berlin Heidelberg, 1996. (Cited on pages 58 and 66.)

[Giunchiglia and Sebastiani, 1996b] F. Giunchiglia and R. Sebastiani. A sat-based decision procedure for alc. In L. Aiello, J. Doyle, and S. Shapiro, editors, *KR*, pages 304–314. Morgan Kaufmann, 1996. (Cited on page 66.)

[Giunchiglia *et al.*, 1998] E. Giunchiglia, F. Giunchiglia, R. Sebastiani, and A. Tacchella. More evaluation of decision procedures for modal logics. In *Principles of Knowledge Representation and Reasoning*, pages 626–635. MORGAN KAUFMANN PUBLISHERS, 1998. (Cited on page 66.)

[Giunchiglia *et al.*, 2001] E. Giunchiglia, M. Narizzano, and A. Tacchella. Quantified Boolean Formulas satisfiability library (QBFLIB), 2001. `http://www.qbflib.org`. (Cited on pages 107 and 134.)

[Giunchiglia *et al.*, 2009] E. Giunchiglia, P. Marin, and M. Narizzano. *Reasoning with Quantified Boolean Formulas*, chapter 24, pages 761–780. Volume 185 of Biere et al. [2009], February 2009. (Cited on page 70.)

[Goldwasser *et al.*, 1989] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989. (Cited on page 82.)

[Goranko and Passy, 1992] V. Goranko and S. Passy. Using the universal modality: gains and questions. *Journal of Logic and Computation*, 2(1):5–30, 1992. (Cited on page 22.)

[Goré, 1999] R. Goré. Tableau methods for modal and temporal logics. In Marcello D'Agostino, DovM. Gabbay, Reiner HÃ€hnle, and Joachim Posegga, editors, *Handbook of Tableau Methods*, pages 297–396. Springer Netherlands, 1999. (Cited on page 127.)

[Halpern and Moses, 1992] J. Halpern and Y. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial intelligence*, 54(3):319–379, 1992. (Cited on page 70.)

[Halpern, 1995] J. Halpern. The effect of bounding the number of primitive propositions and the depth of nesting on the complexity of modal logic. *Artificial Intelligence*, 75(2):361–372, 1995. (Cited on pages 68, 69, and 70.)

[Harrison, 2009] J. Harrison. Without loss of generality. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics*, volume 5674 of

Lecture Notes in Computer Science, pages 43–59, Munich, Germany, 2009. Springer-Verlag. (Cited on page 3.)

[Heguiabehere and de Rijke, 2001] J. Heguiabehere and M. de Rijke. The random modal qbf test set. In *Workshop on Issues in the Design and Experimental Evaluation of System for Modal and Temporal Logics*, pages 58–67, 2001. (Cited on page 153.)

[Heuerding and Schwendimann, 1996] A. Heuerding and S. Schwendimann. A benchmark method for the propositional modal logics k, kt, s4. Technical Report IAM-96-015, University of Bern, Switzerland, 1996. (Cited on page 65.)

[Hoffmann and Areces, 2007] G. Hoffmann and C. Areces. Htab: A terminating tableaux system for hybrid logic. In *Proceedings of Methods for Modalities 5*, November 2007. (Cited on page 134.)

[Hon and Goldstein, 2008] G. Hon and B. Goldstein. *From Summetria to Symmetry: The Making of a Revolutionary Scientific Concept*, volume 20. Springer Netherlands, 2008. (Cited on page 3.)

[Hoos and Stützle, 2000] H. Hoos and T. Stützle. *SATLIB: An Online Resource for Research on SAT*, pages 283–292. Kluwer Academic, 2000. (Cited on page 71.)

[Horrocks *et al.*, 2000] I. Horrocks, P. Patel-Schneider, and R. Sebastiani. An analysis of empirical testing for modal decision procedures. *Logic Journal of IGPL*, 8(3):293–323, 2000. (Cited on pages xvi, 63, 64, 65, and 70.)

[Hustadt and Schmidt, 1997] U. Hustadt and R. Schmidt. On evaluating decision procedures for modal logic. In *IJCAI*, volume 1, pages 202 – 207. Morgan Kaufmann, 1997. (Cited on page 66.)

[Hustadt and Schmidt, 1999] U. Hustadt and R. Schmidt. An empirical analysis of modal theorem provers. *Journal of Applied Non-Classical Logics*, 9(4):479–522, 1999. (Cited on page 67.)

[Ip and Dill, 1996] C. Ip and D. Dill. Better verification through symmetry. *Formal methods in system design*, 9(1-2):41–75, 1996. (Cited on page 7.)

[Johannsen and Drechsler, 2002] P. Johannsen and R. Drechsler. Speeding up verification of rtl designs by computing one-to-one abstractions with reduced signal widths. In *SOC Design Methodologies*, pages 361–374. Springer, 2002. (Cited on page 53.)

[Junttila and Kaski, 2007] T. Junttila and P. Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In *Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2007. (Cited on pages 83 and 107.)

[Katebi *et al.*, 2012] H. Katebi, K. Sakallah, and I. Markov. Conflict anticipation in the search for graph automorphisms. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 243–257. Springer, 2012. (Cited on pages 83 and 122.)

[Kleine Büning and Bubeck, 2009] H. Kleine Büning and U. Bubeck. *Theory of Quantified Boolean Formulas*, chapter 23, pages 735–760. Volume 185 of Biere et al. [2009], February 2009. (Cited on page 70.)

[Köbler *et al.*, 1994] J. Köbler, U. Schöning, and J. Torán. *The graph isomorphism problem: its structural complexity*. Birkhauser Verlag, 1994. (Cited on page 82.)

[Kowalski and Kuehner, 1972] R. Kowalski and D. Kuehner. Linear resolution with selection function. *Artificial Intelligence*, 2(3):227–260, 1972. (Cited on page 5.)

[Kripke, 1959] S. Kripke. A Completeness Theorem in Modal Logic. *Journal of Symbolic Logic*, 24(1):1–14, 1959. (Cited on page 9.)

[Kripke, 1963] S. Kripke. Semantical considerations on modal logic. *Acta Philos. Fennica*, 16:83–94, 1963. (Cited on page 9.)

[Krishnamurthy, 1985] B. Krishnamurthy. Short proofs for tricky formulas. *Acta Informatica*, 22(3):253–275, 1985. (Cited on pages 4 and 142.)

[Ladner, 1977] R. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM journal on computing*, 6(3):467–480, 1977. (Cited on pages 19, 108, 134, and 152.)

[Land and Doig, 1960] A. H Land and A. Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, pages 497–520, 1960. (Cited on page 52.)

[Lis, 1960] Z. Lis. Wynikanie semantyczne a wynikanie formalne. *Studia Logica*, 10(1):39–54, 1960. (Cited on page 125.)

[Manzano, 1993] M. Manzano. Introduction to many-sorted logic. In *Many-sorted Logic and its Applications*, pages 3–86. John Wiley & Sons, Inc., 1993. (Cited on page 75.)

[Margot, 2002] F. Margot. Pruning by isomorphism in branch-and-cut. *Mathematical Programming*, 94(1):71–90, 2002. (Cited on page 7.)

[Margot, 2003] F. Margot. Exploiting orbits in symmetric ilp. *Mathematical Programming*, 98(1-3):3–21, 2003. (Cited on page 7.)

[Massacci, 1994] F. Massacci. Strongly analytic tableaux for normal modal logics. In *Automated Deduction (CADE-12)*, pages 723–737. Springer, 1994. (Cited on page 127.)

[Massacci, 1999] F. Massacci. Design and results of the tableaux-99 non-classical (modal) systems comparison. In *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 14–18. Springer, 1999. (Cited on pages 69, 70, and 153.)

[McKay, 1981] B. McKay. Practical Graph Isomorphism. *Congressus Numerantium*, 30:45–87, 1981. (Cited on page 83.)

[McKay, 2007] B. McKay. Nauty userś guide (version 2.4). *Computer Science Dept., Australian National University*, 2007. (Cited on page 83.)

[Miller *et al.*, 2006] A. Miller, A. Donaldson, and M. Calder. Symmetry in temporal logic model checking. *ACM Comput. Surv.*, 38(3), September 2006. (Cited on page 7.)

[Mitchell *et al.*, 1992] D. Mitchell, B. Selman, and H. Levesque. Problem solving: Hardness and easiness-hard and easy distributions of sat problems. In *Proceeding of the 10th National Conference on Artificial Intelligence (AAAI-92), San Jose, California*, pages 459–465, 1992. (Cited on page 66.)

[Moskewicz *et al.*, 2001] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535. ACM, 2001. (Cited on page 6.)

[Nelson and Oppen, 1979] G. Nelson and D. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 1(2):245–257, 1979. (Cited on pages 47, 56, and 59.)

[Nelson and Oppen, 1980] G. Nelson and D. Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM (JACM)*, 27(2):356–364, 1980. (Cited on pages 47 and 56.)

[Nieuwenhuis and Oliveras, 2005a] R. Nieuwenhuis and A. Oliveras. Dpll (t) with exhaustive theory propagation and its application to difference logic. In *Computer Aided Verification*, pages 321–334. Springer, 2005. (Cited on page 53.)

[Nieuwenhuis and Oliveras, 2005b] R. Nieuwenhuis and A. Oliveras. Proof-producing congruence closure. In *Term Rewriting and Applications*, pages 453–468. Springer, 2005. (Cited on page 52.)

[Oppen, 1980a] D. Oppen. Complexity, convexity and combinations of theories. *Theoretical computer science*, 12(3):291–302, 1980. (Cited on pages 47 and 59.)

[Oppen, 1980b] D. Oppen. Reasoning about recursively defined data structures. *Journal of the ACM (JACM)*, 27(3):403–411, 1980. (Cited on pages 47 and 54.)

[Oxusoff, 1989] L. Oxusoff. *L'évaluation sémantique en calcul propositionnel*. PhD thesis, Université Aix-Marseille 2, 1989. (Cited on page 5.)

[Parthasarathy *et al.*, 2004] G. Parthasarathy, M. Iyer, K. Cheng, and L. Wang. An efficient finite-domain constraint solver for circuits. In *Proceedings of the 41st annual Design Automation Conference*, pages 212–217. ACM, 2004. (Cited on page 48.)

[Patel-Schneider and Sebastiani, 2003a] P. Patel-Schneider and R. Sebastiani. A new general method to generate random modal formulae for testing decision procedures. *Journal of Artificial Intelligence Research*, 18:351–389, 2003. (Cited on pages xiv, 67, and 68.)

[Patel-Schneider and Sebastiani, 2003b] P. Patel-Schneider and R. Sebastiani. A new general method to generate random modal formulae for testing decision procedures. *Journal of Artificial Intelligence Research*, 18:351–389, 2003. (Cited on page 25.)

[Patel-Schneider, 1998] P. Patel-Schneider. Dlp system description. In E Franconi, G. De Giacomo, R. MacGregor, W. Nutt, and C. Welty, editors, *Description Logics*, volume 11 of *CEUR Workshop Proceedings*. CEUR-WS.org, 1998. (Cited on page 65.)

[Pólya and Read, 1987] G. Pólya and R. Read. *Combinatorial enumeration of groups, graphs, and chemical compounds*. Springer-Verlag New York, Inc., 1987. (Cited on page 5.)

[Puget, 2005] J. Puget. Automatic detection of variable and value symmetries. In *Principles and practice of constraint programming*, pages 475–489. Springer, 2005. (Cited on page 94.)

[Ranise and Tinelli, 2003] S. Ranise and C. Tinelli. The smt-lib format: An initial proposal. In *Proceedings of the 1st Workshop on Pragmatics of Decision Procedures in Automated Reasoning, Miami*, 2003. (Cited on page 71.)

[Rehn and Schürmann, 2010] T. Rehn and A. Schürmann. C++ tools for exploiting polyhedral symmetries. In Komei Fukuda, Jorisvander Hoeven, Michael Joswig, and Nobuki Takayama, editors, *Mathematical Software*, volume 6327 of *LNCS*, pages 295–298. Springer, 2010. (Cited on page 122.)

[Roe, 2006] K. Roe. The heuristic theorem prover: Yet another smt modulo theorem prover. In *Computer Aided Verification*, pages 467–470. Springer, 2006. (Cited on pages 8 and 60.)

[Ryan, 2004] L. Ryan. Efficient algorithms for clause-learning SAT solvers. Master's thesis, Simon Fraser University, 2004. (Cited on page 6.)

[Sabharwal, 2005] A. Sabharwal. Symchaff: A structure-aware satisfiability solver. In *AAAI*, volume 5, pages 467–474, 2005. (Cited on pages 6 and 7.)

[Sahlqvist, 1975] H. Sahlqvist. Completeness and correspondence in the first and second order semantics for modal logic. *Studies in Logic and the Foundations of Mathematics*, 82:110–143, 1975. (Cited on page 13.)

[Sakallah, 2009] K. Sakallah. *Symmetry and Satisfiability*, chapter 10, pages 289–338. Volume 185 of Biere et al. [2009], February 2009. (Cited on pages xiv, 3, 4, 84, 85, and 86.)

[Schmidt-Schauß and Smolka, 1991] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial intelligence*, 48(1):1–26, 1991. (Cited on page 153.)

[Sebastiani, 2007] R. Sebastiani. Lazy satisfiability modulo theories. *Journal on Satisfiability, Boolean Modeling and Computation*, 3:141–224, 2007. (Cited on pages xiv, 47, 51, 55, 57, 58, and 59.)

[Segerberg, 1980] K. Segerberg. A note on the logic of elsewhere. *Theoria*, 46(2-3):183–187, 1980. (Cited on page 22.)

[Seress, 1997] A. Seress. An introduction to computational group theory. *Notices of the AMS*, 44:671–679, 1997. (Cited on pages 6 and 147.)

[Seress, 2003] A. Seress. *Permutation Group Algorithms*. Cambridge Tracts in Mathematics. Cambridge University Press, 2003. (Cited on page 122.)

[Seshia *et al.*, 2003] S. Seshia, S. Lahiri, and R. Bryant. A hybrid sat-based decision procedure for separation logic with uninterpreted functions. In *Proceedings of the 40th annual Design Automation Conference*, pages 425–430. ACM, 2003. (Cited on page 48.)

[Shostak, 1984] R. Shostak. Deciding combinations of theories. *Journal of the ACM (JACM)*, 31(1):1–12, 1984. (Cited on pages 47, 56, and 59.)

[Shostak, 1979] R. Shostak. A practical decision procedure for arithmetic with function symbols. *Journal of the ACM (JACM)*, 26(2):351–360, 1979. (Cited on pages 47 and 56.)

[Sistla *et al.*, 2000] A. Sistla, V. Gyuris, and E. Emerson. Smc: a symmetry-based model checker for verification of safety and liveness properties. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 9(2):133–166, 2000. (Cited on page 7.)

[Smullyan, 1968] R. Smullyan. *First-order logic*. Springer-Verlag (Berlin and New York etc), 1968. (Cited on page 125.)

[Spaan, 1993] E. Spaan. *Complexity of modal logics*. PhD thesis, Universiteit van Amsterdam, 1993. (Cited on page 22.)

[Steele, 1990] G. Steele. *Common LISP: the language*. Digital press, 1990. (Cited on page 72.)

[Strichman *et al.*, 2002] O. Strichman, S. Seshia, and R. Bryant. Deciding separation formulas with sat. In *Computer Aided Verification*, pages 209–222. Springer, 2002. (Cited on page 48.)

[Strichman, 2002] O. Strichman. On solving presburger and linear arithmetic with sat. In *Formal Methods in Computer-Aided Design*, pages 160–170. Springer, 2002. (Cited on page 48.)

[Stump *et al.*, 2001] A. Stump, C. Barrett, D. Dill, and J. Levitt. A decision procedure for an extensional theory of arrays. In *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science*, pages 29–37. IEEE, 2001. (Cited on page 53.)

[Sutcliffe, 2009] G. Sutcliffe. The tptp problem library and associated infrastructure. *Journal of Automated Reasoning*, 43(4):337–362, 2009. (Cited on page 71.)

[Tacchella, 1999] A. Tacchella. *sat system description. In P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, and P. Patel-Schneider, editors, *Description Logics*, volume 22 of *CEUR Workshop Proceedings*. CEUR-WS.org, 1999. (Cited on page 66.)

[Urquhart, 1999] A. Urquhart. The symmetry rule in propositional logic. *Discrete Applied Mathematics*, 96–97(0):177 – 193, 1999. (Cited on page 4.)

[van Benthem, 1977] J. van Benthem. *Modal Correspondence Theory*. PhD thesis, Universiteit van Amsterdam, 1977. (Cited on pages 11 and 17.)

[van Benthem, 1983] J. van Benthem. *Modal Logic and Classical Logic*. Bibliopolis, 1983. (Cited on pages 11 and 17.)

[van Fraassen, 1989] B. van Fraassen. *Laws and Symmetry*. Oxford Scholarship Online Monographs, 1989. (Cited on page 3.)

[Velev and Bryant, 1999] M. N Velev and R. Bryant. Exploiting positive equality and partial non-consistency in the formal verification of pipelined microprocessors. In *Design Automation Conference.*, pages 397–401. IEEE, 1999. (Cited on page 48.)

[Williams and Hogg, 1994] C. Williams and T. Hogg. Exploiting the deep structure of constraint problems. *Artificial Intelligence*, 70(1):73–117, 1994. (Cited on page 66.)

[Wolfman and Weld, 1999] S. Wolfman and D. Weld. The lpsat engine & its application to resource planning. In *IJCAI*, pages 310–317, 1999. (Cited on pages 48 and 58.)