UNIVERSIDAD NACIONAL DE CÓRDOBA

FACULTAD DE MATEMÁTICA, ASTRONOMÍA Y FÍSICA

SERIE “A”

TRABAJOS DE INFORMÁTICA

Nº 6/2012

A Note about Modal Symmetries

Ezequiel Orbe - Carlos Areces - Gabriel Infante Lopez

Editores: Luciana Benotti – Laura Brandan Briones

CIUDAD UNIVERSITARIA – 5000 CÓRDOBA

REPÚBLICA ARGENTINA

A Note about Modal Symmetries

Ezequiel Orbe, Carlos Areces, and Gabriel Infante-López

FaMAF, Universidad Nacional de Córdoba, Argentina

Abstract. In this paper we show how permutation of literals can be used to define symmetries for modal formulas in clausal form. We show that the symmetries of a modal formula φ preserve inference: if σ is a symmetry of φ then $\varphi \models \psi$ if and only if $\varphi \models \sigma(\psi)$. Hence, a modal theorem prover that has access to the symmetries of the input formula, can use them during search to cheaply derive symmetric inferences (e.g., as is done during clause learning in propositional SAT). We also present in a mechanism to efficiently compute symmetries using graphs automorphisms, and preliminary empirical results showing that symmetries appear in many cases in both randomly generated and hand-tailored modal formulas.

1 Introduction

Many concrete, real life problems present symmetries. For instance, if we want to known whether trying to place three pigeons in two pigeonholes results in two occupying the same nest, it does not really matter which of all pigeons gets in each pigeonhole. Starting by putting the first pigeon to the first pigeonhole is the same as if we put the second one in it. In mathematical and commonsense reasoning these kinds of symmetries are often used to reduce the difficulty of reasoning – one can analyze in detail only one of the symmetric cases and generalize the result to the others.

The exact same is done in propositional theorem proving. Many problem classes and, in particular, those arising from real world applications, displays a large number of symmetries; and current SAT solvers take into account these symmetries to avoid exploring duplicate branches of the search space. In the last years there has been extensive research in this area, focusing on how to define symmetries, how to detect them efficiently, and how can SAT solvers better profit from them [21].

Informally, we can define a symmetry of a discrete object as the permutation of its components that leaves the object (or some aspect of it) intact. Think in the rotations of a spatial solid. In the context of SAT solving we can formally define a symmetry as a permutations of the variables (or literals) of the problem that preserves its structure and, in particular, its set of solutions. Depending on which aspect of the problem is kept invariant, symmetries are classified in the literature into semantic or syntactic [8]. Semantic symmetries are intrinsic properties of the Boolean 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 representations of the function, i.e., a permutation of variables that does not change the representation. A syntactic symmetry is also a semantic symmetry, but the other implication does not always hold.

The first work to deal with syntactic symmetries in the context of SAT solving was [18]. In this article, Krishnamurthy introduces the notions of *global* and *local* symmetries as inference rules to strengthen resolution-based proof systems for propositional logic, and showed that these rules can be used to shorten the proofs of certain difficult propositional problems such as the pigeon-hole principle. Since then, many articles investigating how to detect and exploit symmetries have appeared. Most of them can be grouped into two different approaches: static symmetry breaking and dynamic symmetry breaking.

In the static symmetry breaking approach [12, 13, 1], symmetries are detected and eliminated from the problem statement before the SAT solver is used. They work as a preprocessing step. In contrast, the dynamic symmetry breaking approach [11, 8, 9] detects and breaks symmetries during the search space exploration. The first approach has the advantage that the theorem provers do not suffer any modifications while the second approach can take advantage of symmetries that emerge during the process of SAT solving.

The first articles discussing the static approach are those of Crawford et al. [12, 13], which laid the theoretical foundation for reasoning by symmetry. [12] shows that the symmetry group of a formula induces an equivalence relation on the set of truth assignments, i.e., it partitions the space of possible assignments into equivalence classes. In principle, only one truth assignment of each equivalence class should be considered by a SAT solver while working on a problem with symmetries. [12] also proves that symmetry detection can be polynomially reduced to the colored graph automorphism problem and explicitly describes how to create a colored graph from a propositional formula in conjunctive normal form whose automorphism group coincides with the symmetry group of the original formula. Some years later, in [13], Crawford et al. introduce a method for symmetry breaking. The method uses symmetry-breaking predicates which are added on a per-symmetry basis and are chosen such that they are true of exactly one element in each of the equivalence classes induced by the symmetry group. The main drawback of this approach is that the resulting formula can be exponentially larger than the original one. Later, Aloul [1] developes these ideas presenting new graph constructions together with the notion of *partial* symmetry breaking, in which symmetry-breaking predicates are introduced only for the irredundant set of generators of the symmetry group of the formula [23].

Despite their differences both the static and the dynamic approach, share the same goal: to identify symmetric branches of the search space and guide the SAT solver away from symmetric branches already explored. A third alternative was introduced in [7], which combines symmetry reasoning with clause learning [22] in Conflict-Driven Clause Learning (CDCL) SAT solvers [15]. The idea is to augment the clause learning process by using the symmetries of the problem as an inference rule to learn the symmetric equivalents of conflict-induced clauses. This

approach is particularly appealing because it does not imply major modifications to the search procedure and the required modification to the clause learning process is minor.

Summing up then, symmetries have been extensively investigated for propositional SAT solving as the discussion about shows. And some of these ideas have been explored also for other logics [4, 5]. But to the best of our knowledge they have not been investigated in the field of automated theorem proving for modal logics. In this paper, after a brief introduction to the basic notions about basic modal logics and symmetries in Section 2, we show how permutation of literals can also be used to define symmetries for modal formulas in clausal form. In Section 3 and 4 we provide the theoretical foundations that enables the exploitation of symmetries. In particular, we show that the symmetries of a modal formula φ preserve inference: if σ is a symmetry of φ then $\varphi \models \psi$ if and only if $\varphi \models \sigma(\psi)$. Hence, a modal theorem prover that has access to the symmetries of the input formula, can use them during search to cheaply derive symmetric inferences (e.g., as is done during clause learning in propositional SAT). We also present in Section 5 a mechanism to efficiently compute symmetries using graphs automorphisms, and preliminary empirical results showing that symmetries appear in many cases in both randomly generated and hand-tailored modal formulas. Finally, in Section 6 we sum up the contributions of the paper, and briefly discuss how the results could be used by a tableaux based theorem prover for modal logics. At the moment of writing this article, the ideas discussed here have not been implemented into a theorem prover, but work is on the way to include symmetry learning into the hTab prover [16].

2 Modal logic and modal symmetry

For completeness, we will start with a short review to modal logic that will introduce the basic notions (see [10] for a complete introduction).

Definition 1 (Syntax). *Let* $\mathsf{PROP} = \{p_1, p_2, \ldots\}$ *be a countable infinite set of* propositional variables*. The set of (basic) modal formulas* FORM *is defined as*

$$\mathsf{FORM} := p \mid \neg p \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \neg\Box\varphi \mid \Box\varphi,$$

where $p \in \mathsf{PROP}$, $\varphi, \psi \in \mathsf{FORM}$*. For simplicity, we will only consider the basic mono-modal logic* K*. Given a formula* φ *we represent as* $Var(\varphi)$ *the set of propositional variables appearing in* φ*.*

The set of propositional literals *over* PROP *is defined as* $\mathsf{PLIT} = \mathsf{PROP} \cup \{\neg p \mid p \in \mathsf{PROP}\}$*. If* l *is a literal, then we will represent as* $|l|$ *the propositional symbol appearing in* l*. If* l *is a literal, then* $\neg l$ *is defined as follows*

$$\neg l = \begin{cases} p & \text{if } l = \neg p, p \in \mathsf{PLIT} \\ \neg p & \text{if } l = p, p \in \mathsf{PLIT} \end{cases}$$

Given a formula φ, we write $\mathsf{PLIT}(\varphi)$ as the set of literals appearing in φ defined recursively as:

$$\begin{aligned}\mathsf{PLIT}(p) &= \{p\} \text{ for } p \in \mathsf{PROP}\\ \mathsf{PLIT}(\neg p) &= \{\neg p\} \text{ for } p \in \mathsf{PROP}\\ \mathsf{PLIT}(\varphi \wedge \psi) = \mathsf{PLIT}(\varphi \vee \psi) &= \mathsf{PLIT}(\varphi) \cup \mathsf{PLIT}(\psi)\\ \mathsf{PLIT}(\neg\Box\varphi) = \mathsf{PLIT}(\Box\varphi) &= \mathsf{PLIT}(\varphi).\end{aligned}$$

We say that a set of propositional literals L is complete *if for each $p \in \mathsf{PROP}$ either $p \in L$ or $\neg p \in L$. We say that L is* consistent *if for each $p \in \mathsf{PROP}$ either $p \notin L$ or $\neg p \notin L$. Any complete and consistent set of literals L defines a unique propositional valuation $v : \mathsf{PROP} \mapsto \{\top, \bot\}$ as $v(p) = \top$ if $p \in L$ and $v(p) = \bot$ if $\neg p \in L$.*

Let $S \subseteq \mathsf{PROP}$ be a set of propositional variables. The consistent and complete set of literals generated by *S (notation L_S) is defined as $L_S = S \cup \{\neg p \mid p \in \mathsf{PROP}\backslash S\}$. Let L be a set of literals and φ a modal formula, the* restriction of *L to φ is defined as: $L_{|\varphi} = \{l \in L \mid |l| \in Var(\varphi)\}$.*

We say that a formula is in KCNF (or that it is a KCNF form) if it is a conjunction of KCNF clauses. A KCNF clause *is a disjunction of propositional and modal literals. A* modal literal *is a formula of the form $\Box C$ or $\neg\Box C$ where C is a KCNF clause. Every modal formula can be transformed into an equivalent KCNF formula (see [20] for details).*

A formula in KCNF can be represented as a set of KNCF clauses (which are to be interpreted conjunctively), and each clause can be represented as a set of propositional and modal literals (wich are to be interpreted disjunctively). The advantage of using sets is that we can disregard the order in which clauses and litterals appear and their multiplicity. This will be important when we define symmetries below. From now on we will always assume that modal formulas are in KCNF. Modal formulas are interpreted on labeled transition graphs:

Definition 2 (Semantics). *A modal model $\mathcal{M}$ is a structure $\langle W, R, V\rangle$, where W, the* domain*, is a non-empty set; R is the* accessibility relation*, a binary relation on W; and $V : W \mapsto \mathcal{P}(\mathsf{PROP})$ is the* valuation function *that assigns to each element w of the domain, a subset $V(w)$ of PROP, that correspond to the propositional variables that are true in that state. Given a model $\mathcal{M}$ and a state w in $\mathcal{M}$ the pair $(\mathcal{M}, w)$ is called a* pointed model*. We will usually drop the parenthesis and call $\mathcal{M}, w$ a pointed model.*

Given a formula φ, an a pointed model $\mathcal{M}, w$ the satisfiability relation *$\mathcal{M}, w \models \varphi$ is defined inductively as:*

$$\begin{aligned}\mathcal{M}, w \models p &\text{ iff } p \in V(w) \text{ for } p \in \mathsf{PROP}\\ \mathcal{M}, w \models \neg p &\text{ iff } p \notin V(w) \text{ for } p \in \mathsf{PROP}\\ \mathcal{M}, w \models \Box C &\text{ iff } \forall v \text{ such that } wRv \text{ we have } \mathcal{M}, v \models C\\ \mathcal{M}, w \models \neg\Box C &\text{ iff } \mathcal{M}, w \not\models \Box C\\ \mathcal{M}, w \models C &\text{ iff } \exists \varphi \in C \text{ such that } \mathcal{M}, v \models \varphi \text{ for } C \text{ a KCNF clause}\\ \mathcal{M}, w \models S &\text{ iff } \forall \varphi \in S \text{ we have } \mathcal{M}, v \models \varphi \text{ for } S \text{ a KCNF formula.}\end{aligned}$$

As is clear from the definition, KCNF clauses are interpreted disjunctively (one of the members should be satisfied) while KCNF formulas are interpreted conjunctively (all members should be satisfied). We will lift $\models$ to set of pointed models in the usual way: for M a set of pointed models, $M \models \varphi$, if and only if $\mathcal{M}, w \models \varphi$ for every $\mathcal{M}, w \in M$. We say that a formula φ is satisfiable if there is a pointed model $\mathcal{M}, w$ such that $\mathcal{M}, w \models \varphi$. Given a formula φ, $Mod(\varphi)$ is the set of pointed models *of φ, i.e., $Mod(\varphi) = \{\mathcal{M}, w \mid \mathcal{M}, w \models \varphi\}$.*

Given formulas φ and ψ we say that ψ can be inferred from *φ and write $\varphi \models \psi$ if for any pointed model $\mathcal{M}, w$ we have that $\mathcal{M}, w \models \varphi$ implies $\mathcal{M}, w \models \psi$. In other words, $\varphi \models \psi$ if and only if $Mod(\varphi) \subseteq Mod(\psi)$.*

We extend the use of $\models$ to propositional formulas. For L a consistent and complete set of literals and φ a propositional formula we write $L \models \varphi$ when φ is true under the propositional valuation defined by L.

A crucial tool in modal model theory is the notion of *bisimulation*: if two models are bisimilar, they satisfy the same modal formulas.

Definition 3 (Bisimulation). *Given two models $\mathcal{M} = \langle W, R, V\rangle$ and $\mathcal{M}' = \langle W', R', V'\rangle$, a* bisimulation between *$\mathcal{M}$ and $\mathcal{M}'$ is a non-empty relation $S \subseteq W \times W'$ that satisfies*

- ***Atomic Harmony:*** *if wSw' then $p \in V(w)$ if and only if $p \in V(w')$.*
- ***Zig:*** *if wSw' and wRv then there is v' such that $wR'v'$ and vSv'.*
- ***Zag:*** *if wSw' and $w'R'v'$ then there is v such that wRv and vSv'.*

We say that two pointed models $\mathcal{M}, w$ and $\mathcal{M}', w'$ are bisimilar (notation $\mathcal{M}, w \leftrightarrows \mathcal{M}', w$) if there is a bisimulation S between $\mathcal{M}$ and $\mathcal{M}'$ such that wSw'.

The following is a classical property of bisimulations. The proof is by induction on formulas (details can be found in [10]).

Proposition 1. *If $\mathcal{M}, w \leftrightarrows \mathcal{M}', w'$ then $\mathcal{M}, w \models \varphi$ if and only if $\mathcal{M}', w' \models \varphi$.*

With this we complete the basic classical notions about modal logic we need in the rest of the paper. We need now to introduce the basic notions concerning symmetries:

Definition 4. *[Permutation and symmetry] A* permutation *is bijective function $\sigma : \mathsf{PLIT} \mapsto \mathsf{PLIT}$. Permutations are lifted to sets of literals in the usual way: if L is a set of literals then $\sigma(L) = \{\sigma(l) \mid l \in L\}$. Given a modal formula φ and a permutation σ we define $\sigma(\varphi)$ as the formula obtained by simultaneously replacing all literals l in $\mathsf{PLIT}(\varphi)$ by $\sigma(l)$.*

When used in applications we will be mostly interested in permutations that only involve only a finite number of elements (i.e., $\sigma(l) = l$ except for a finite number of literals). In these cases we can succinctly define a permutation by listing the cycles that generate it.

We say that a permutation σ is consistent *if for every literal l, $\sigma(\neg l) = \neg\sigma(l)$. We say that a permutation σ is a* symmetry *for φ if $\varphi = \sigma(\varphi)$, when conjunctions and disjunctions in φ are represented as sets.*

Example 1. Trivially, the identity permutation $\sigma(l) = l$ is a consistent symmetry of any formula φ. More interestingly, consider the formula $\varphi = \{\{\neg p, r\}, \{q, r\}, \{\Box\{\neg p, q\}\}\}$, then the permutation $\sigma = (p, \neg q)$ is a consistent symmetry of φ.

In the next section we will show that consistent symmetries preserve inferences: i.e., if σ is a consistent symmetry of φ and $\varphi \models \psi$ then $\varphi \models \sigma(\psi)$.

3 Symmetry preserves modal inference

In this section we show that consistent symmetries for modal formulas behave similarly as in the propositional case and, hence, could assist in modal theorem proving if they can be computed efficiently. In particular, we will prove the following theorem.

Theorem 1. *Let φ and ψ be modal formulas and let σ be a consistent symmetry of φ. Then $\varphi \models \psi$ if and only if $\varphi \models \sigma(\psi)$.*

The main ingredient in the proof of Theorem 1 will be the notion of σ-simulations.

Definition 5 (σ-simulation). *Let σ be a permutation. A* σ-simulation *between models $\mathcal{M} = \langle W, R, V\rangle$ and $\mathcal{M}' = \langle W', R', V'\rangle$ is a non-empty relation $S \subseteq W \times W'$ that satisfies:*

- ***Atomic Harmony:*** *if wSw' then $l \in L_{V(w)}$ if and only if $\sigma(l) \in L_{V'(w')}$.*
- ***Zig:*** *if wSw' and wRv then there is v' such that $w'R'v'$ and vSv'.*
- ***Zag:*** *if wSw' and $w'R'v'$ then there is v such that wRv and vSv'.*

We say that two pointed models $\mathcal{M}, w$ and $\mathcal{M}', w'$ are σ-similar (notation $\mathcal{M}, w \underline{\rightarrow}_\sigma \mathcal{M}', w'$ if there is a σ-simulation S such that wSw'.

Notice that $\underline{\rightarrow}_\sigma$ is not a symmetric relation: we can have $\mathcal{M}, w \underline{\rightarrow}_\sigma \mathcal{M}', w'$ but not $\mathcal{M}', w' \underline{\rightarrow}_\sigma \mathcal{M}', w'$. From the definition of σ-simulations intuitively follows that while they don't preserve validity of modal formulas (as is the case with bisimulation) they do preserve validity of *permutations* of formulas.

Proposition 2. *Let σ be a consistent permutation and $\mathcal{M}$, $\mathcal{M}'$ models such that $\mathcal{M}, w \underline{\rightarrow}_\sigma \mathcal{M}', w'$. Then $\mathcal{M}, w \models \varphi$ iff $\mathcal{M}', w' \models \sigma(\varphi)$.*

Proof. The proof is by induction on φ. The only interesting case is the base case, as all the other cases are taken care by they inductive hypothesis and the Zig and Zag conditions as with bisimulations (see [10]).

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

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

Given a model $\mathcal{M}$ and a permutation σ we can apply σ to the model and obtain $\sigma(\mathcal{M})$. As we will show next, $\mathcal{M}$ and $\sigma(\mathcal{M})$ are always σ-similar.

Definition 6. *Let σ be a permutation and $\mathcal{M} = \langle W, R, V\rangle$ a modal model. Then $\sigma(\mathcal{M}) = \langle W, R, V'\rangle$, where,$V'(w) = \sigma(L_{V(w)})\backslash\{\neg l : \neg l \in \sigma(L_{V(w)})\}$. We will lift this construction to sets of models in the usual way: for M a set of models, $\sigma(M) = \{\sigma(\mathcal{M}) \mid \mathcal{M} \in M\}$.*

Proposition 3. *Let σ be a consistent permutation and $\mathcal{M} = \langle W, R, V\rangle$ a model. Then $\mathcal{M} \underline{\rightarrow}_\sigma \sigma(\mathcal{M})$.*

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

Atomic Harmony: We have to check that $l \in L_{V(w)}$ iff $\sigma(l) \in L_{V'(w)}$. But 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.

The **Zig** and **Zag** conditions are trivial as the relation in both models is the same.

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 will be a direct corollary of the following proposition in the particular case when σ is a symmetry and hence $\sigma(\varphi) = \varphi$.

Proposition 4. *Let σ be a consistent permutation, φ a modal formula and $\mathcal{M}$ a modal model. Then $\mathcal{M}, w \models \varphi$ iff $\sigma(\mathcal{M}), w \models \sigma(\varphi)$.*

Proof. This lemma follows directly from the fact that $\mathcal{M} \underline{\rightarrow}_\sigma \sigma(\mathcal{M})$ (Proposition 3) and Proposition 2.

Corollary 1. *If σ is a symmetry of φ then $\mathcal{M} \in Mod(\varphi)$ iff $\sigma(\mathcal{M}) \in Mod(\varphi)$.*

We are now ready for the proof of Theorem 1.

Proof (of Theorem 1). We first show that under the hypothesis of the theorem the following property holds

Claim: $Mod(\varphi) = \sigma(Mod(\varphi))$.

[$\supseteq$] Let $\mathcal{X} \in \sigma(Mod(\varphi))$ and let $\mathcal{M} \in Mod(\varphi)$ be such that $\mathcal{X} = \sigma(\mathcal{M})$. Then $\mathcal{M}, w \models \varphi$ and by Corollary 1 $\sigma(\mathcal{M}), w \models \varphi$ and $\sigma(\mathcal{M}) \in Mod(\varphi)$.

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

Now, we have to prove that $\varphi \models \psi$ if and only if $\varphi \models \sigma(\psi)$. By definition, $\varphi \models \psi$ iff and only if $Mod(\varphi) \models \psi$. By Proposition 4, this is the case if and only if $\sigma(Mod(\varphi)) \models \sigma(\psi)$.

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

4 Permutations and layering.

Propositional variables appearing at different modal depths in modal formulas of the basic modal logic (where no particular property is assumed of the accessibility relation in models) are independent of each other. This property actually relies on the tree model property that many modal logics share: every pointed model $\mathcal{M}, w$ in the logic is bisimilar to a tree model $\mathcal{T}, w$ (i.e., a model where each not has at most one predecesor and w has zero predecesors). Given M a class of pointed models, let us write *Tree*(M) for the subclass of tree models of M. Then the tree model propery ensures that $\varphi \models \psi$ if and only if *Tree*(*Mod*(φ)) $\models \psi$.

This was used in [2], for example, to introduce the idea of *layering* and define a new translation of modal formulas into first order logic. It is easy to see that the same ideas can be applied to symmetries: a different permutation could be used at each modal depth without changing the results we have proved in the previous Section.

Definition 7 (Permutation sequences). *Let $\bar{\sigma} = \langle \sigma_1, \ldots, \sigma_n \rangle$ be a finite (possibly empty) sequence of permutations. $|\bar{\sigma}| = n$ will denote the lenght of the sequence, and for $\bar{\sigma}$ such that $|\bar{\sigma}| \geq 1$, head($\bar{\sigma}$) $= \sigma_1$ will denote the first element of the sequence, and tail($\bar{\sigma}$) $= \langle \sigma_2, \ldots, \sigma_n \rangle$ will denote the sequence of permutations obtained after removing the first.*

Example 2. Consider the modal formula $\varphi = (p \vee \Box(p \vee \neg r)) \wedge (\neg q \vee \Box(\neg p \vee r))$. Without layering, this formula has no symmetries. But the sequence of permutations $\langle \sigma_1, \sigma_2 \rangle$ generated by $\sigma_1 = (p \neg q)$ and $\sigma_2 = (p \neg r)$ is a symmetry.

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

Definition 8 ($\bar{\sigma}$-simulation). *Let $\bar{\sigma}$ be a permutation sequence such that $|\bar{\sigma}| = n$. A* $\bar{\sigma}$-simulation *between tree pointed models $\mathcal{T}, w = \langle W, R, V \rangle, w$ and $\mathcal{T}', w' = \langle W', R', V' \rangle, w'$ is a family of relations $S_{\bar{\sigma}_i} \subseteq W \times W'$ that satisfies the conditions for $|\bar{\sigma}_i| \geq 1$:*

- ***Root:*** $w S_{\bar{\sigma}} w'$.
- ***Atomic Harmony:*** *if $w S_{\bar{\sigma}_i} w'$ then $l \in L_{V(w)}$ iff head($\bar{\sigma}_i$)(l) $\in L_{V'(w')}$.*
- ***Zig:*** *if $w S_{\bar{\sigma}_i} w'$ and wRv then there is v' such that $w'R'v'$ and $v S_{tail(\bar{\sigma}_i)} v'$.*
- ***Zag:*** *if $w S_{\bar{\sigma}_i} w'$ and $w'R'v'$ then there is v such that wRv and $v S_{tail(\bar{\sigma}_i)} v'$.*

We say that two tree pointed models $\mathcal{T}, w$ and $\mathcal{T}', w'$ are $\bar{\sigma}$-similar (notation $\mathcal{T}, w \underline{\rightarrow}_{\bar{\sigma}} \mathcal{T}'$), w' if there is a $\bar{\sigma}$-simulation between them.

We can now repeat the work we did in the previous section to arrive to a result similar to Theorem 1 but involving sequences of permutations. Formally, for a modal formula of modal depth n we consider a sequence of permutation $\bar{\sigma}$

of length $n+1$ and define $\bar{\sigma}(\varphi)$ as

$$\begin{aligned}
\bar{\sigma}(p) &= head(\bar{\sigma})(p) \\
\bar{\sigma}(\neg p) &= head(\bar{\sigma})(\neg p) \\
\bar{\sigma}(\psi \wedge \theta) &= \bar{\sigma}(\psi) \wedge \bar{\sigma}(\theta) \\
\bar{\sigma}(\psi \vee \theta) &= \bar{\sigma}(\psi) \vee \bar{\sigma}(\theta) \\
\bar{\sigma}(\Box\psi) &= \Box tail(\bar{\sigma})(\psi) \\
\bar{\sigma}(\neg\Box\psi) &= \neg\Box tail(\bar{\sigma})(\psi)
\end{aligned}$$

Once more, we will say that a sequence of permutations $\bar{\sigma}$ of size $md(\varphi)+1$ is a symmetry of φ if $\bar{\sigma}(\varphi) = \varphi$ when conjunctions and disjunctions in φ are considered as sets. A sequence of permutations is consistent if each of the permutations in the sequence is consistent. We can then prove:

Theorem 2. *Let φ and ψ be modal formulas and let $\bar{\sigma}$ be a consistent sequence of permutations of size $md(\varphi)+1$. If $\bar{\sigma}$ is symmetry of φ then for any ψ with $md(\psi) \leq md(\varphi)$ we have that $\varphi \models \psi$ if and only if $\varphi \models \sigma(\psi)$.*

5 Detecting modal symmetries.

Different techniques have been proposed for detecting symmetries of propositional formulas in clausal form. Some of them, deal directly with the formula (e.g., [9]), while others, reduce the problem to the problem of finding automorphism in colored graph [12, 13, 1].

The availability of efficient tools to detect graph automorphisms (e.g., [19, 14, 17]) has made the later approach the most successful one because it is fast and easy to integrate . The main idea behind it is to use the formula to construct a colored graph whose symmetry group is isomorphic to the symmetry group of the original formula. [1] defines different ways to map formulas to graphs, and shows the existence of an isomorphism between the automorphism group of the graph and the symmetry group of the original propositional formula.

As an example, we explain how one of these constructions, denominated MIN3C, transforms a formula φ into a colored graph G_φ:

1. Each clause of φ is represented in G_φ by a node of color 1.
2. Each propositional variable is represented by two nodes of color 2. One represents the positive literal and the other the negative literal.
3. An edge is created between a literal l and $\neg l$ to ensure Boolean consistency (see [1]).
4. An edge is created between a clause C and a literal l if and only if $l \in C$.

We will now extend this construction to modal formulas. When dealing with modal formulas we must take into account modal operators and propositional variables appearing at different modal depths.

In the following, a clause preceded by $\Box$ will be called a $\Box$-*clause* and, similarly, one preceded by $\neg\Box$ is called a $\neg\Box$-*clause*.

Definition 9. *Let φ be a KCNF formula. The colored graph G_φ corresponding to φ is constructed as follows:*

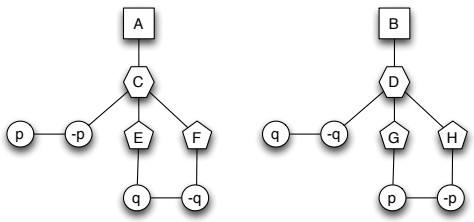
1. *Each clause C of φ is represented in G_φ by a* clause *node of color 1.*
2. *For each propositional variable p occurring in C:*
   (a) *Add two* literal *nodes of color 2: one labelled p and one labelled $\neg p$.*
   (b) *Add an edge between these two nodes to ensure Boolean consistency.*
   (c) *Add an edge from C to nodes representing literals occurring in C.*
3. *For each modal literal $\Box C'$ and each literal $\neg\Box C''$ occurring in C:*
   (a) *Add a* $\Box$-clause *node of color 3 to represent C', or an* $\neg\Box$-clause *node of color 4 to represent C'' as the case may be.*
   (b) *Add an edge from C to this node.*
   (c) *Repeat the process from point 2 for each literal ocurring in C' and C''.*

This construction creates a graph with 4 colours and at most $2|V| \times (md(\varphi)+1) + \#Clauses + \#\Box Clauses + \#\neg\Box Clauses$ nodes.

Example 3. Let us consider the following KCNF formula $\varphi = \neg\Box(\neg p \vee \Box q \vee \Box\neg q) \wedge \neg\Box(\neg q \vee \Box p \vee \Box\neg p)$. Using sets for conjunctions and disjunctions we can represent φ as:

$$\varphi = \{\ \{\neg\Box\{\neg p, \Box\{q\}, \Box\{\neg q\}\}\}, \{\neg\Box\{\neg q, \Box\{p\}, \Box\{\neg p\}\}\}\ \}$$

This formula has eight clauses (2 at modal depth 0, 2 at modal depth 1 and 4 at modal depth 2) and six literals (2 at modal depth 1, and 4 at modal depth 2). The associated G_φ 4 -colored graph is shown in Figure 1 (colors are represented by shapes in the figure).



$A = \{\neg\Box\{\neg p, \Box\{q\}, \Box\{\neg q\}\}\}$
$B = \{\neg\Box\{\neg q, \Box\{p\}, \Box\{\neg p\}\}\}$
$C = \neg\Box\{\neg p, \Box\{q\}, \Box\{\neg q\}\}$
$D = \neg\Box\{\neg q, \Box\{p\}, \Box\{\neg p\}\}$
$E = \Box\{q\}$
$F = \Box\{\neg q\}$
$G = \Box\{p\}$
$H = \Box\{\neg p\}$

$\bar{\sigma}$-generators :

$\bar{\sigma}_1 = \langle (p)(\neg p)(q)(\neg q), (p\ \neg p)(q)(\neg q)\rangle$
$\bar{\sigma}_2 = \langle (p)(\neg p)(q)(\neg q), (p)(\neg p)(q\ \neg q)\rangle$
$\bar{\sigma}_3 = \langle (p\ q)(\neg p\ \neg q), (p\ \neg q)(\neg p\ q)\rangle$

Fig. 1. Graph representation of φ.

Notice the way literals are handled during the construction of G_φ: the construction duplicates literals nodes occurring at different modal depth. By doing this we incorporate the notion of layering introduced in Section 4. Also, $\Box$-clauses and $\neg\Box$-clauses are colored different. This is to avoid spurious permutations that maps $\Box$-clauses to $\neg\Box$-clauses and the other way around.

5.1 Experimental results

In the previous seccions we have defined modal symmetries and show how to compute them efficiently, but the question remains: *do modal symmetries really appear sufficiently often to care about them?* There seems to be no easy way to answer this question in a definitive way. The answer will surely depend on the clase of problems we are targetting, and even then, on the particular way we are encoding these problems as modal formulas.

With the aim of getting at least an empirical handle on the answer and, as a byproduct, to test how hard it is to actually find modal symetries using the construction described above, we carried out the following preliminary experiment[1].

We will work with a testbed that includes both random and hand-tailored formulas:

- **Random formulas:** We generated 300000 formulas using hGen [3]. Formulas were divided into 600 classes of 500 instance formulas each according to the parameters used to generate them. We used a wide spectrum of parameters within those available in hGen, with modal depth ranging from 1 to 5, number of propositional variables ranging from 3 to 15, and number of clauses ranging from 3 to 9[2].
- **Hand-tailored formulas:** A set of formulas coming from the Logics Workbench Benchmark (LWB) [6]. For lack of time, we focused on four classes of problem formulas for modal logic K, namely, `k_branch`, `k_dumm`, `k_lin` and `k_path`. All the problems were first translated to the KCNF format.

All tests were run on a Intel Core i7 2.93GHz with 16GB of RAM using Bliss [17] as the graph automorphism engine. This tool receives as input the graph specification and returns the set of generators of the formula's symmetry group. Figure 2 summarizes the results for both classes of problems. For each formula category we report the number of instances analyzed (#Inst), the percentage of formulas with at least one symmetry (%wSymm), and the average number of generators (AvgGens).

Category	# Inst	% wSymm	AvgGens
Random	300000	55.4	1.567
Hand-tailored	168	50.6	22.718

Fig. 2. Results on Random and Hand-tailored formulas

The empirical results shows then, than both in random and hand-tailores formulas the chances of finding formulas with symmetries are good (intuitively,

[1] The results shown are those obtained at the time of submission, we plan to continue the experiment and have more complete coverage for the final paper.

[2] The probability of finding at least one symmetry should increase with the number of clauses and for that reason this parameter was kept low.

you will run into them more than once every two formulas you consider). Of course these results are far from conclusive and further testing is needed.

The case of hand-tailored formulas presents a behaviour that coincide with our expectations. While the total percentage of formulas having at least one symmetry is below the corresponding percentage for random formulas, their distribution is clearly driven by the codification used in each problem class, as can be seen by looking in detail the results obtained for each problem class in this category. Figure 3 summarizes the results for the four classes of problems in the hand-tailored test set. In this table we can observe that from the four problem classes, two (`k_branch` and `k_path`) of them exhibit a great amount of symmetries in its instances, while the others two (`k_dum` and `k_lin`), exhibit none. Table 3 also includes the average generator search time (AvgTime), expressed in seconds, for each problem class. Note that in all the cases it is negligible.

Problem Class	# Inst	% wSymm	AvgGens	AvgTime
K_BRANCH_N	21	100	12	0.22
K_BRANCH_P	21	100	11	0.19
K_DUM_N	21	0	0	0
K_DUM_P	21	0	0	0
K_LIN_N	21	0	0	0.01
K_LIN_P	21	0.050	1	0
K_PATH_N	21	100	35.952	0.07
K_PATH_P	21	100	32.952	0.06

Fig. 3. Hand-tailored formulas: Results by problem class

For the random formulas, we also estimated the probability that a random formula will contain at least one symmetry.

To do so, we consider X, as a boolean random variable, that is equal to one if a random formula has at least one symmetry and 0 otherwise. We estimate the probability of $X = 1$ using a random formula generator which is controlled by a vector Θ of parameters that we suppose uniformly distributed. We decompose $p(X = 1)$ as follows,

$$p(X = 1) = \sum_{\Theta} p(X = 1, \Theta) = \sum_{\Theta} p(X = 1|\Theta)p(\Theta) = q \sum_{\Theta} p(X = 1|\Theta)$$

where q is the probability of a particular value for Θ. In our experiments q is equal to $\frac{1}{600}$ given that Θ can take one of 600 different values. In order to estimate $p(X|\Theta)$, we randomly generated 500 formulas for each possible value of Θ. Figure 4 summarizes all different values we got. The figure is an histogram that shows the number of configurations that turn out to have a particular $p(X|\Theta)$. For example, it shows that there are around 50 configurations that produce formula without symmetries and that there are around 45 that will for

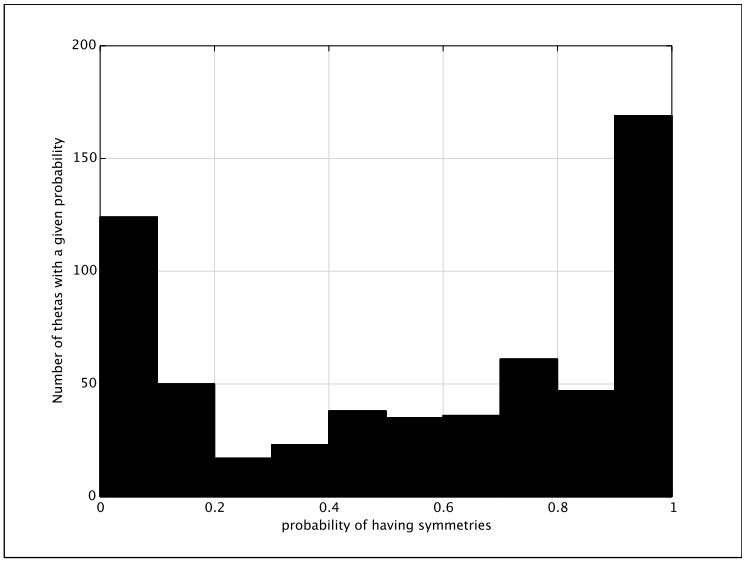


Fig. 4. Histogram counting the number of parameters that produce each value of probability,

sure produce formulae with symmetries. Using these empirical estimations, it turns out that $p(X = 1)$ is equal to 0.554.

6 Conclusions and further work

In this work, we have shown that the notion of symmetry can be extended to modal formulas in clausal form. We provided the theoretical foundations, defining the concept of σ-simulation and showed that this is the key notion to establish that the symmetries of a modal formula preserves inference. This property is the key needed to incorporate symmetry learning in a theorem prover. We also presented an extension of σ-simulation to layering, denominated $\bar{\sigma}$-simulation which provides more flexibility at the moment of defining a symmetry of the formula, and thus capture more symmetries than a σ-simulation. Finally we provided a detection algorithm that captures the notion of layering and presented experimental results that shows the existence of symmetries in modal formulas.

Our ongoing research now focus on incorporating symmetry learning in the hTab prover. To do so, we plan to enrich the *semantic branching* rule with symmetry knowledge.

The semantic branching rule, an efficient alternative to the standard $\vee$-rule of modal tableau is:

$$\frac{\psi \vee \theta}{\begin{array}{c|c} \psi & \theta \\ & \neg\psi \end{array}}$$

This branching rule ensures that the search space associated to each of the branches is disjoint. Moreover, during the exploration of the θ branch we can assume that the ψ branch has closed (otherwise, the search for a satisfying model would have stopped). Hence $\neg\psi$ is a consequence of the root formula in the tableau. A tableaux based prover having access to symmetries could implement the following, further enriched, branching formula:

$$\frac{\psi \vee \theta}{\begin{array}{c|c} \psi & \theta \\ & \sigma_i(\neg\psi) \end{array}} \text{ for } \sigma_i \text{ any symmetry of the root formula.}$$

It remains to explore other forms of using symetries in automated theorem proving for modal logics. One promising theme that we will investigate in the future is the definition of permutation that can map modal literals modal literals, in addition to propositional literals.

References

1. 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.
2. C. Areces, R. Gennari, J. Heguiabehere, and M. de Rijke. Tree-based heuristics in modal theorem proving. In W. Horn, editor, *Proceedings of ECAI'2000*, pages 199–203, Berlin, Germany, 2000.
3. C. Areces and J. Heguiabehere. hGen: A Random CNF Formula Generator for Hybrid Languages. In *Methods for Modalities 3*, Nancy, France, 2003.
4. G. Audemard. Reasoning by symmetry and function ordering in finite model generation. In *Proceedings of CADE-18*, pages 226–240, 2002.
5. G. Audemard, B. Mazure, and L. Sais. Dealing with symmetries in quantified boolean formulas. In *Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, pages 257–262, 2004.
6. 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.
7. 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.
8. B. Benhamou and L. Sais. Theoretical study of symmetries in propositional calculus and applications. In *Proceedings of CADE*, pages 281–294, 1992.
9. B. Benhamou and L. Sais. Tractability through symmetries in propositional calculus. *Journal of Automated Reasoning*, 12(1):89–102, 1994.
10. P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2000.
11. C. Brown, L. Finkelstein, and P. Walton Purdom. Backtrack searching in the presence of symmetry. In *AAECC*, pages 99–110, 1988.
12. J. Crawford. A theoretical analysis of reasoning by symmetry in first-order logic. *In AAAI Workshop on Tractable Reasoning*, 1992.
13. J. Crawford, M. Ginsberg, E. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *KR*, pages 148–159, 1996.

14. P. Darga, M. Liffiton, K. Sakallah, and I. Markov. Exploiting structure in symmetry detection for CNF. In *Design Automation Conference, 2004. Proceedings. 41st*, pages 530–534, 2004.
15. N. Een and N. Sörensson. An extensible sat-solver. In *SAT*, pages 502–518, 2003.
16. G. Hoffmann and C. Areces. Htab: A terminating tableaux system for hybrid logic. In *Proceedings of Methods for Modalities 5*, November 2007.
17. T. Junttila and P. Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In *ALENEX*, 2007.
18. B. Krishnamurthy. Short proofs for tricky formulas. *Acta Informatica*, 22(3):253–275, 1985.
19. B. McKay. Nauty user's guide. Technical report, Australian National University,Computer Science Department, 1990.
20. 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.
21. M. Prasad, A. Biere, and A. Gupta. A survey of recent advances in SAT-based formal verification. *International Journal on Software Tools for Technology Transfer*, (7):156–173, 2005.
22. L. Ryan. Efficient algorithms for clause-learning sat solvers. Master's thesis, Simon Fraser University, 2004.
23. A. Seress. An introduction to computational group theory. *Notices of the AMS*, 44:671–679, 1997.