Mapping Inferences: Constraint Propagation and Diamond Satisfaction
Gennari, R.

Citation for published version (APA):
Chapter 8

The Layered Translation

8.1 Introduction

8.1.1 Motivations

The need for efficient automated reasoning methods for modal logics is increasingly being felt in areas such as knowledge representation, reasoning about programs, and reasoning systems for autonomous agents [Are00, Was00]. We can identify at least four general strategies for modal theorem proving:

1. develop purpose-built calculi and tools, like tableaux systems;

2. translate modal problems into automata-theoretic problems, and then adopt automata-theoretic methods to obtain answers;

3. translate modal problems into first-order problems, and use general first-order tools;

4. build dedicated solvers for modally quantified formulas on top of solvers for propositional formulas; for instance, in [Seb97, GS00], a tableux-based procedure for modal logic is built on top of the Davis-Logemann-Loveland procedure for the propositional component, known as DPLL or DP in the SAT community — where the letter P stays for Putnam.

The advantage of indirect methods such as (2), (3) and (4) is that they allow us to re-use well-developed and well-supported tools instead of having to develop new ones from scratch.

In this chapter, we focus on the third option: translation-based theorem proving for modal logics, where modal formulas are translated into first-order formulas and reasoning problems are to be fed to first-order provers. Our starting
point is the standard relational translation introduced in Section 7.3. First-order theorem provers perform poorly on the standard outputs of this translation [ONdRG01, HS97]. To overcome this, very sophisticated decision procedures have been developed [dNdR02] together with alternative translations [ONdRG01].

Our proposal in this chapter consists in a simple refinement of the standard relational translation that allows us to encode additional modal information. In fact, this new translation centres around a strong form of the tree modal property, which is often identified, nowadays, as one of the main reasons for the good computational behaviour of those modal logics that enjoy it (see [Grä99, Var97] and also Section 8.3 below): a modal formula is satisfiable (or more precisely: $\mathbf{K(\text{Index})}$-satisfiable) if and only if it is satisfiable at the root of a model based on a tree.

### 8.1.2 Outline

We divide the material of this chapter in two main parts. First, we propose our new translation of modal formulas. Our translation results from the composition of the standard relational translation and a translation that maps modal formulas into an intermediate multimodal language. It is in this intermediate multimodal language that we first encode the semantic property known as the tree model property — this fact is also used in Chapter 9. This semantic information is then partially encoded by the relational translation into the layered fragment; i.e., the first-order fragment that is carved out by the new translation. This fragment is contained in the one identified by the standard translation; more precisely, the former fragment is strictly contained in the latter in the case of multimodal languages.

In the second part of the present chapter, we show how to use a first-order theorem prover on the first-order fragments identified by the standard translation and the new one, respectively. The theorem prover SPASS is used to perform the experimental comparison between the outcome of the two translations: hence the analysis of the comparison results, that we illustrate in Section 8.5, highlights that SPASS performs up to several orders of magnitude better on the outcome of the new translation than on the standard one, in terms of both memory space and execution times.

The encoding of formula layers, carried over by our translation, is the key factor behind the improvement in performance: it is this encoding that allows us to partially exploit the tree-model property of $\mathbf{K(\text{Index})}$ at the purely syntactic level of the theorem proving process.

### 8.1.3 Structure

The chapter is organised as follows. In Section 8.2, we provide the base inference rules behind first-order theorem proving, as this is used then in Section 8.5:
8.2. Modal Theorem Proving via the Standard Translation and Resolution

i.e., propositional and first-order resolution. Then Section 8.3 is devoted to the so-called tree model property of basic modal logics. That property is used in Section 8.4 to define our refinement of the standard translation from modal into first-order logics. In that section, we also introduce the layered fragment of first-order logic, as carved out by the new translation. We then show, in Section 8.5, how the theorem prover SPASS performs better on that fragment than on the full first-order fragment carved out by the standard translation. We conclude this chapter with Section 8.6.

8.2 Modal Theorem Proving via the Standard Translation and Resolution

Resolution is at the core of most automated theorem provers for first-order logic. It is a refutation procedure, whose goal is to derive a logical contradiction from a given formula. The basic rule applies to conjunctions of disjunctions, and is essentially based on the following propositional tautology:

\[ M \land (\lor L \lor \neg M) \rightarrow \lor L. \]

In other words: the occurrence of both \( M \) and \( \neg M \) as in the antecedent of the above formula is irrelevant with respect to the truth value of the overall formula. Thus \( M \) and \( \neg M \) can be safely removed.

In what follows, we give a precise content to this brief introduction, as much as space allows. We only assume the reader to have an idea of what substitutions and variable renamings are. For a complete introduction to the topic, there are a number of good texts in the literature: we refer the interested reader to [RV01] for a comprehensive overview of automated reasoning methods, to [Lov78] for an introduction to them, based on logics; to [Doe94] for a more logic-programming approach to resolution, and to [Apt97] as its natural companion for the logic programming language Prolog.

8.2.1 Propositional Resolution

Before passing a propositional formula to a theorem prover based on resolution, the formula has to be in "conjunctive normal form". This is essentially a conjunction of disjunctions, where negations are pushed inwards. We provide a bit of terminology below, as it will be used over and over in the remainder of the thesis.

A propositional literal is either an atom, like \( p \), or a negation of an atom, like \( \neg p \). We shall mainly consider disjunctions of literals in the remainder of this chapter: for instance, formulas like \( p \lor \neg q \). Literals of the form \( L \) and \( \neg L \) are complementary.
The following definition is used over and over in the remainder of this thesis, thus we highlight it as follows.

**Definition 8.2.1.** A propositional formula \( \phi \) is in **conjunctive normal form** (CNF) if it is a conjunction of disjunctions of literals.

For instance: \( \neg p \land (\neg q \lor p) \) is in conjunctive normal form, whereas \( \neg(p \lor q) \land p \) is not. There is a standard procedure, based on the famous De Morgan tautologies of classical logic

\[
\neg\phi \land \neg\psi \leftrightarrow \neg(\phi \lor \psi), \quad \neg\phi \lor \neg\psi \leftrightarrow \neg(\phi \land \psi),
\]

that reduces any given formula to its conjunctive normal form. In the above example, the formula \( \neg(p \lor q) \land p \) is reduced to the equivalent formula \( \neg p \land \neg q \land p \).

Literal disjunctions are transformed into the set of their literals, called *clauses* (this is the *clausification* process); for instance, \( p \lor \neg q \) is reduced to \( \{p, \neg q\} \).

Here and in the following, we adopt the standard convention of representing a clause without parentheses; for instance, \( \{p, \neg q\} \) will be rewritten as \( p, \neg q \). Then a conjunction of literal disjunctions is represented as a set of clauses. These are usually represented as list. For instance, the formula \( \neg p \land (p \lor q) \) corresponds to the clause set represented as the following clause list:

1. \( \neg p \),
2. \( p, q \).

Thus the binary ground resolution rule can be applied to a clause set and return a clause set as displayed in the following.

\[
\frac{L, L_1, \ldots, L_n, \neg L, L'_1, \ldots, L'_m}{L_1, \ldots, L'_n, L'_1, \ldots, L'_m} \quad \text{(Res)}
\]

A derivation via (Res) of the empty clause from a given clause set \( C \) is a sequence of clause sets, each of which is either \( C \) or obtained from antecedent clause sets in the sequence via (Res).

This simple rule is sufficient for determining whether a formula is a classical tautology, due to the following result.

**Theorem 8.2.2.** A propositional formula is unsatisfiable iff the empty clause can be derived from it by means of (Res).

A proof of the above statement can be found, for instance, in [Lov78].
8.2.2 First-order Resolution

The first-order case is more complicated than the propositional one due to the presence in the language of variables, function symbols and quantifiers.

Let us assume that the only logical symbols that occur in the formula $\phi$ are conjunction, disjunction and negation; this is not a restricted assumption, since the other logical symbols can be defined in terms of this.

A first-order formula like $\phi$ is first reduced by pushing all negation symbols inwards, and so obtaining its negated normal form. This is done by using the following classical equivalences:

$$\forall x \phi \leftrightarrow \neg \exists \neg \phi, \quad \exists x \phi \leftrightarrow \forall \neg \phi, \quad \neg \phi \land \neg \psi \leftrightarrow \neg (\phi \lor \psi), \quad \neg \phi \lor \neg \psi \leftrightarrow \neg (\phi \land \psi).$$

Then the resulting formula is reduced to its Skolem form (there are different and equivalent versions of this, see [dN94]). This amounts to substituting variables, bound by only existential quantifiers, by new different constant symbols (i.e., not occurring elsewhere in the formula). Moreover, each occurrence of an existential quantifier, within the scope of $n + 1$ occurrences of universal quantifiers of the form $\forall x_0 \ldots \forall x_n$, results in the removal of the variables bound by the existential quantifier, and their substitutions with new different function symbols, applied to the variables bound by the universal quantifiers; e.g., $f(x_1, \ldots, x_n)$. The resulting formula is satisfiable iff the original formula is.

As soon as a formula is in Skolem form, all universal quantifiers are moved leftwards, renaming variables if needed; then all these universal quantifiers are removed. For instance, $\forall x (Rx \land \forall z Sx)$ is equivalently transformed into $\forall x \forall y (Rx \land Sy)$, and then into $Rx \land Ry$, where we implicitly read all variables as being universally quantified over.

Finally, the resulting formula is reduced to its conjunctive normal form (see Subsection 8.2.1), and this into a clause set as in Subsection 8.2.1. In the first-order case, literals are atomic formulas or their negations. For instance, $Rx$ and $\neg Ry$ and $Rx$ are first-order clauses; the set that contains both of them is a clause set.

The resulting clause sets can be passed to the resolution rule for first-order logic. However, the presence of universally quantified variables forces us to "unify" variables in the resolution inference. For instance, $Rx$ and $\neg Rc$ do not contradict each others propositionally. But remind that $Rx$ is implicitly universally quantified over; thus $Rx$ stands also for $Rc$, so to speak; therefore the two clauses $Rc$ and $\neg Rx$ constitute a contradiction in first-order logic. Roughly speaking, the way we can put forward this contradiction is by interleaving propositional resolution steps and substitutions. In our example, first $Rx$ would be instantiated to $Rc$, and afterwards a propositional resolution step would infer a contradiction, by generating the empty clauses. Put more precisely, we need to incorporate unification, as defined below, in the first-order resolution inferences.

The unification procedure by Martelli and Montanari, as quoted in [Apt97], returns the most general unifier $\theta$ of a set of terms. Initially, $\theta$ is instantiated to
the set containing \( t = s \), where \( t \) and \( s \) are the given two terms to unify. Non-deterministically, an equation \( t = s \) is chosen from \( \theta \), and the associated action is performed:

1. in case \( t = s \) is of the form \( ft_1 \cdots t_n = gs_1 \cdots s_m \), then the procedure halts and returns failure if \( f \) and \( g \) are two different function symbols; else \( m = n \) and the procedure restarts with \( \theta \) equal to the union of \( (\theta - \{ t = s \}) \) and the set \( \{ x_1 = y_1, \ldots, x_n = y_n \} \);

2. in case \( t = s \) is \( x = s \) and \( x \) occurs elsewhere in \( \theta \), then each occurrence of \( x \) in \( \theta' := \theta - \{ s = t \} \) is simultaneously substituted by \( s \), and \( \theta \) is set equal to \( \theta' \cup \{ x = s \} \);

3. in case \( t = s \) is \( x = s \) and \( x \) occurs in \( s \), then the procedure halts and returns failure;

4. in case \( t = s \) is \( t = x \), then \( E \) is set to \( (E - \{ s = t \}) \cup \{ x = t \} \);

5. in case \( t = s \) is \( x = x \), then remove it from \( \theta \).

The same procedure is applicable with atoms in place of terms. In both cases, it terminates (see [Apt97]) by producing either failure, or a most general unifier only if it exists; this is unique modulo variable renamings.

Resolution calculus, in the first-order case, can be cast in terms of two rules: resolution and factorisation. These embed unification and, if applied to a clause set, each returns a clause set as displayed in the following:

\[
\frac{M, L_1, \ldots, L_n}{\nu, \mu} \quad \frac{M', L'_1, \ldots, L'_m}{\nu', \mu'} \quad (\text{Res'}) \quad \frac{N, L_1, \ldots, N'}{\nu, \mu, \mu'} \quad \frac{N', L'_1, \ldots, L'_m}{\nu', \mu'} \quad (\text{Fact})
\]

where \( \mu \) is the most general unifier of the literals, respectively, in \( \{ M, \neg M \} \) and \( \{ N, N' \} \). The (Res') rule is applied to two clauses that have no variables in common. This requirement of variable disjointness can be easily met by renaming variables, if necessary.

A derivation via (Res') and (Fact) of the empty clause from a given clause set \( C \) is a sequence of clause sets, each of which is either \( C \) or obtained from antecedent clause sets in the sequence via (Res').

Again, we have a result similar to the one in Theorem 8.2.2; see [Lov78] for a proof.

**Theorem 8.2.3.** A first-order formula is unsatisfiable iff the empty clause can be derived from it by means of (Res') and (Fact).
8.2.3 Challenging Cases

Consider the formula $\Box(\neg p \lor \Diamond p)$ of Example 7.3.3 again; that formula is clearly satisfiable, for instance, on a model with only one world and no relations. Proving this in first-order logic, by means of resolution, amounts to showing that the set with the following clauses is satisfiable:

1. $\neg R(a, y), \neg P(y), R(y, f(y))$,
2. $\neg R(a, z), \neg P(z), P(f(z))$.

Observe now that the above clauses have two resolvents:

3. $\neg R(a, a), \neg P(a), \neg P(f(a)), P(f(f(a)))$
4. $\neg R(a, f(z)), R(f(z), f(f(z))), \neg R(a, z), \neg P(z)$.

Clauses 2 and 4 resolve to produce the following new clause:

5. $\neg R(a, f(f(z))), R(f(f(z)), f(f(f(z)))), \neg R(a, f(z)), \neg R(a, z), \neg P(z)$.

Clauses 2 and 5 resolve again to produce an analogue of 5, with even higher term-complexity etc. None of the clauses is redundant and can be deleted; in the limit our input set has infinitely many resolvents. This shows that standard resolution may not terminate in the case of clauses that result from the standard translation of satisfiable modal formulas, even though the satisfiability problem for basic modal logics is decidable — in non-deterministic space.

An obvious question suggests itself: What went wrong in the above example? More precisely: Which features of the original modal formula get lost when clauses are generated from the first-order formulas returned by the standard translation, that is instead needed by the above resolution based method to terminate? How can we recover that information?

Observe that, to obtain the resolvent in line 4, the unary $P$ literals were resolved upon; these literals (or rather the modal operators in which scope the literals are) occur at different modal depths in the original formula $\Box(p \rightarrow \Diamond p)$. Thus this resolution step is pointless, from the perspective of modal logics like $K$: the negative $P$ literal derives from the $\Box$-operator, so this literal occurs at modal depth 1; whereas the positive $P$ literal is also bound by the $\Diamond$-operator, hence this literal occurs at modal depth 2. Unless we stipulate so, by means of additional axioms, distinct modal depths are independent. A similar comment pertains to the resolvent obtained in line 3, where again we resolved upon binary $R$ literals that correspond to modal operators occurring in the formula at different modal depths.

Similar examples as the above one, and the questions they pose triggered our refinement of the standard relational translation. In [AGHdR00], the latter was refined by marking literals, with distinct modal depths, by means of syntactically distinct indices. The mathematical justification is provided by a strong form of the tree model property, as we explain below, in Section 8.3.
8.3 The Importance of Having Layers

The example in Section 8.2.3 is interesting in many ways. Above all, it highlights how the structure of the original modal formula gets lost in the standard translation process from modal to first-order formulas in clausal form, and naturally suggests how this information could help to avoid the flaws of first-order resolution in deciding the satisfiability of modal formulas. In fact, in our remark following the quoted example, we propose to consider "layers" of modal formulas as key information to be retrieved and passed to the theorem prover. We explain precisely what we mean by layers and their use with respect to first-order theorem proving in the remainder of the present section.

8.3.1 Trees and Layers

In what follows, $S^+$ and $S^*$ denote the transitive and reflexive, transitive closure of the relation $S$, respectively.

**Definition 8.3.1.** A rooted tree, or simply a tree is a relational structure of the form $\mathcal{T} := (T, S)$ that enjoys the following properties:

1. $T$, the set of nodes, contains a special node $r \in T$, called the root;
2. the root $r$ is the only node in $T$ such that $\forall t \in T (S^*rt)$;
3. every element of $T$, distinct from $r$, has a single $S$ predecessor: that is, $\forall t \in T (\exists s \in T \land Sst \land \forall s' \in T (Ss't \rightarrow s' = t))$; the root has no $S$ predecessors;
4. $S^+$ is acyclic: i.e., $\forall t \in T (\neg S^+tt)$.

A path in a tree $\mathcal{T}$ is a finite sequence of $T$ nodes of the form $s := \langle t_i : i \leq n \rangle$ such that $St_it_{i+1}$ holds for every two adjacent nodes $t_i$ and $t_{i+1}$ in the sequence and $t_0$ is the root $r$ of $\mathcal{T}$. The length of the path $s$ is the number $n$ of nodes in $s$ minus 1.

The above properties are quite intuitive if we keep in mind the image of a tree. The first property states that a tree cannot be empty, at least its root must belong to it. The second property qualifies the root as the only node from which all the other nodes can be reached, via a finite number of $S$ transitions. Then the third property requires that any node, different from the root, should have precisely one predecessor via an $S$ transition; moreover, the root cannot be reached via any $S$ transition. The last property imposes a tree to be free of loops: i.e., there cannot be a finite number of $S$ transitions starting and finishing at the same node.
The Importance of Having Layers

DEFINITION 8.3.2.

- A tree model for the unimodal language $\mathcal{ML}(P)$ is a model $\mathcal{M} = (W, R, V)$ such that the relational structure $\langle W, R \rangle$ is a tree.
- A tree-like model for the multimodal language $\mathcal{MMML}(\text{Index}, P)$ is a model $\langle W, \{R_i : i \in \text{Index}\}, V \rangle$ such that $\langle W, \bigcup_i R_i \rangle$ is a tree.
- A logic $L$ has the tree model property if every $L$-satisfiable formula is satisfiable at the root of a tree or tree-like model for $L$.

The advantage of dealing with a tree-like structure $T$ is that every world of $T$ can be reached through a unique path of $T$’s relations starting from $w$. We state this well-known property of trees as a fact (for instance, see [Wil96]), and it is immediate to prove given our Definition 8.3.1.

FACT 8.3.3. There is precisely one path terminating at each node of a tree-like structure. $\square$

The above fact is used over and over in the remaining proofs of this chapter, to well define valuations in models via paths of trees or tree-like structures.

8.3.2 Modal Depth and Layers

The notion of layering for basic modal logics emerges at both the semantic level, via tree models, and the syntactic level of modal formulas. In fact, tree or tree-like models as introduced above come with a layering induced by paths. Likewise, the parse tree of a modal formula induces a natural formula layering, where new layers begin at nodes labelled by modal operators. For instance, in $\Box(\neg p \vee \Diamond p)$, the operator $\Box$ occurs in layer 1, while the operator $\Diamond$ and its argument occur in layer 2. The following definition captures precisely this sort of syntactical layering.

DEFINITION 8.3.4. Let $\phi$ be a modal formula. The modal depth $\text{mdepth}(\phi)$ of $\phi$ is defined as:

$$\begin{align*}
\text{mdepth}(p) &= \text{mdepth}(\neg p) = 0 \\
\text{mdepth}(\psi \land \chi) &= \text{mdepth}(\psi \lor \chi) = \max\{\text{mdepth}(\psi), \text{mdepth}(\chi)\} \\
\text{mdepth}(\Diamond \psi) &= 1 + \text{mdepth}(\psi).
\end{align*}$$

There is a direct correlation between formula layers and layers in a tree or tree-like models; we state and prove it in the following section. As a consequence of the results below, literals occurring in distinct formula layers will not be resolved upon, and not be combined; in this manner, we avoid the problems encountered in the example discussed in Subsection 8.2.3.
8.3.3 The Tree Model Property: Layers at Work

Since we are only concerned with satisfiability in a world, the theorem below allows us to restrict our attention to a tree model and its root. Furthermore, the result below highlights the link between the modal depth of a formula, on which our translation is based (see Definitions 8.4.1 and 8.4.7 below), and the layering that comes with a tree-like model, as remarked after Definition 8.3.4.

The proof of the following statement can be found in books of modal logic like [dR93, BdRV01].

**Theorem 8.3.5 (Tree Model Property).** Let \( \mathcal{L} \) be any multimodal language, \( \phi \) an \( \mathcal{L} \) formula, and \( \mathcal{M} \) an \( \mathcal{L} \) model. Then there exists an \( \mathcal{L} \) tree-like model \( \mathcal{T} \) that enjoys the following properties:

- \( \phi \) is satisfiable at the root of \( \mathcal{T} \) iff it is satisfiable in \( \mathcal{M} \);
- consider a natural number \( i \) such that \( 0 \leq i \leq \text{mdepth}(\phi) \), and assume that \( \psi \) is a subformula at modal depth \( i \) in \( \phi \); then the satisfiability of \( \psi \) can be tested in a \( \mathcal{T} \) world \( t \) such that, if \( k \) is the length of the \( \mathcal{T} \) path to \( t \), then \( i \leq k \leq \text{mdepth}(\phi) \).

Figure 8.1 below illustrates Theorem 8.3.5 for the simple case of \( \phi := \Diamond \Diamond p \lor \Diamond q \): to test the satisfiability of \( \phi \) and \( \Diamond \Diamond p \) we need to walk, from the root, along paths of length at most 2; the satisfiability of both \( \Diamond p \) and \( \Diamond q \) can be tested starting from layer 1, and reaching at most layer 2; finally, the satisfiability of \( p \) and \( q \) can be tested in the layer 2.

![Diagram of Theorem 8.3.5](image)

**Figure 8.1:** The Tree Model Property.

Observe that the tree model property and the finite model property are independent: in fact, there are modal logics for which the former fails but the latter holds, and vice versa. We refer the reader to any introduction to modal logic for these; see [BdRV01], for instance.
8.4 Layer by Layer

In this section we exploit the tree model property to devise our refinement of the standard translation of modal formulas into first-order formulas.

The new translation proceeds in two steps. First, modal formulas are translated into formulas of an intermediate modal language. It is in this intermediate step that the layering induced by trees (see Theorem 8.3.2) is made explicit at the syntactical level: the modal depths of formulas (see Definition 8.3.4), which are related to the layers of tree-like structures as in Theorem 8.3.5, are encoded as indices. In turn, these intermediate formulas with layers as indices are passed to the standard translation (see Definition 7.3.1), and thus transformed into formulas of a first-order language.

All in all, the new translation will mark relations and propositions according to the number of modal operators in whose scope a given modal subformula occurs; i.e., the modal depth at which the subformula occur. For instance, the modal formula

$$
\Box \Diamond p
$$

is translated into a multimodal formula with diamonds and proposition letters labelled according to the modal depth at which these occur in the above formula:

$$
\Diamond_1 \Diamond_2 p_2.
$$

The standard relational translation into first-order logic transforms the latter into the following formula:

$$
\exists y (R_1 xy \land \exists z (R_2 yz \land P_2 z)).
$$

Similarly, $$\Box(p \rightarrow \Diamond p)$$ becomes first $$\Box_1 (p_1 \rightarrow \Diamond_2 p_2)$$; then the standard translation generates the first-order formula

$$
\forall y (R_1 xy \rightarrow (P_1(y) \rightarrow \exists x (R_2 yx \rightarrow P_2 x))).
$$

In the remainder of the present section, we focus on the unimodal case (see Subsection 8.4.1 below), and briefly touch on the multimodal one (see Subsection 8.4.2 below), since this constitutes a trivial extension of the former.

8.4.1 The Unimodal Case

As the above example illustrates, our final relational translation is reached via an intermediate step through an intermediate multimodal language. This collects the modal operator and the proposition letters of the given unimodal language, and mark them with natural numbers, as formalised in the following definition.

**Definition 8.4.1.** Consider a modal language $$\mathcal{ML} := \mathcal{ML}(Index, P)$$, and a multimodal language $$\mathcal{IML} := \mathcal{IML}(P)$$ with set of propositions equal to $$\{p_n : p \in P\}$$, and modal operators in $$\{\Box_i, \Diamond_i : i \geq 0\}.$$
Suppose that $\phi$ is a modal formula of $\mathcal{ML}$. Let $n$ be a natural number. The translation $Tr(\phi, n)$ of $\phi$ into the intermediate multimodal language $T\mathcal{ML}$ is defined as follows:

$$
\begin{align*}
Tr(p, n) & := p_n, \\
Tr(\neg \psi, n) & := \neg Tr(\psi, n), \\
Tr(\psi \land \chi, n) & := Tr(\psi, n) \land Tr(\chi, n), \\
Tr(\psi \lor \chi, n) & := Tr(\psi, n) \lor Tr(\chi, n), \\
Tr(\square \psi, n) & := \Box_{n+1} Tr(\psi, n+1), \\
Tr(\Diamond \psi, n) & := \Diamond_{n+1} Tr(\psi, n+1).
\end{align*}
$$

Denote by $LT_x$ the composition of $Tr(\_, 0)$ and $ST_x$: i.e., for every modal formula $\phi$,

$$
LT_x(\phi) := ST_x \circ Tr(\phi, 0).
$$

The layered relational translation $LT$ is $LT_x$, for a generic first-order variable $x$.

In the two lemmas below, we adopt the following notational convention.

**Convention 8.4.2.** If $T$ is a tree or tree-like structure with root $r$, then let $path(t)$ denote the length of the $T$ path to $t$.

Notice that this path is unique in virtue of Fact 8.3.3.

**Lemma 8.4.3.** Let $\phi$ be a unimodal formula and $T$ a tree-like model in the language of $\phi$. If the intermediate multimodal formula $Tr(\phi, n)$ is satisfiable at a world $t$ in the model $T$ such that $path(t) = n$, then the unimodal formula $\phi$ is satisfiable as well.

**Proof.** Let $T$ be a tree model as in the above statement, with universe $T$, a finite number of labelled relations $R_i$ and valuation $V$.

Now, let us construct a unimodal model $\mathcal{N}$ on $T$ whose relation $R$ is defined as follows:

$$
R := \{(t, u) \in T \times T : R_j tu \text{ for some } R_j \text{ of } T\}.
$$

(The Rel)

The valuation $V'$ of the model $\mathcal{N}$ is defined as follows: for every proposition letter $p$ and for every $t$ such that $path(t) = n$, $t \in V'(p)$ iff $t \in V(Tr(\phi, n))$.

Given this model $\mathcal{N}$ and our tree-like model $T$, we can prove the following stronger claim, from which follows our lemma:

$$
T, t \models Tr(\phi, n) \text{ iff } \mathcal{N}, t \models \phi,
$$

for every $t \in T$ such that $n$ is equal to $path(t)$.

We prove the above claim by structural induction on $\phi$. The atomic and Boolean cases are easy to spell out, since both immediately follow from the above
choice of $V'$ and the fact that the intermediate translation $Tr$ is a homomorphism on Boolean formulas, see Definition 8.4.1. Next, assume that $\phi$ is a formula of the form $\Diamond \psi$. In this case,

$$Tr(\phi, n) = \Diamond_{n+1} Tr(\psi, n + 1).$$

Assume that $t$ is a node of $T$, and the length of the $T$ path to $t$ is $n$; i.e.,

$$\text{path}(t) = n.$$ We have that $T, t \models Tr(\phi, n)$ iff there exist $u$ and $R_j$ in $T$ such that $R_jtu$, and $T, u \models Tr(\psi, n)$. Since $\text{path}(u)$ is equal to the length of the path to $t$ plus 1 (i.e., the length of the $R_j$ transition from $t$ to $u$), by induction hypothesis we know that $T, u \models Tr(\psi, n + 1)$ is equivalent to $\mathcal{N}, u \models \psi$. Therefore, this and (Rel) yield that $T, t \models Tr(\phi, n)$ iff $\mathcal{N}, t \models \phi$. \hfill \Box

In the following lemma, we prove the reverse implication of the above lemma.

**Lemma 8.4.4.** Let $\phi$ be a unimodal formula and $T$ a tree model in the language of $\phi$. If $\phi$ is satisfiable at the world $t$ in the model $T$ such that $\text{path}(t) = n$, then the intermediate multimodal translation $Tr(\phi, n)$ of $\phi$ is satisfiable as well.

**Proof.** We define a model $\mathcal{N} := \langle T, \{R_{n+1} : n \geq 0\}, V'\rangle$ that has the same universe $T$ as $T$. The relations of $\mathcal{N}$ are defined by stipulating the following:

$$R_{n+1}uv \text{ holds iff both } \text{path}(u) = n \text{ and } Ruv \text{ hold.}$$

We complete the characterisation of $\mathcal{N}$ by defining its valuation $V'$ as follows: for every proposition letter $p$ and every world $t \in T$ such that $\text{path}(t) = n$, we stipulate that $t \in V'(Tr(p, n))$ holds iff $t \in V(p)$.

The following intermediate claim follows easily now by structural induction on $\phi$, like in Lemma 8.4.3: for every unimodal formula $\phi$, every world $t$ and $n$ such that $\text{path}(t) = n$, we have

$$T, t \models \phi \text{ iff } \mathcal{N}, t \models Tr(\phi, n) \text{ holds.}$$

Our lemma is clearly an immediate consequence of the above claim. \hfill \Box

We now combine the above lemmas and prove that the intermediate translation $Tr$ preserves satisfiability.

**Theorem 8.4.5.** Assume a modal formula $\phi$. Thus the following holds true:

- $\phi$ is satisfiable iff its intermediate modal translation $Tr(\phi, n)$ is satisfiable;
- $\phi$ is satisfiable iff its intermediate modal translation $Tr(\phi, 0)$ is satisfiable.
PROOF. The first item is an immediate consequence of Lemmas 8.4.4 and 8.4.3, via Theorem 8.3.5, and it yields the second item. □

Finally, the combination of Theorems 8.4.5 and 7.3.4 yields that the layered translation, being the composition of \( \text{Tr} \) and the standard translation, preserves satisfiability too.

**Theorem 8.4.6.** Let \( \phi \) be a modal formula. Then \( \phi \) is satisfiable iff \( \text{LT}_x(\phi) \) is so, for any first-order variable \( x \).

**Proof.** The statement follows from Theorems 8.4.5 and 7.3.4, since \( \text{LT}_x \) results from the composition of the standard translation \( \text{ST}_x \) and \( \text{Tr}(\cdot, 0) \), see Definition 8.4.1. □

In the subsection below, we reformulate some of the above definitions and statements for the case of the multimodal logics \( K(\text{Index}) \).

### 8.4.2 The Multimodal Case

The layered relational translation is easily extended to the multimodal language \( \mathcal{MML}(\text{Index}, P) \) by means of a slightly more complex encoding. We need strings of labels instead of natural numbers to capture the different relations involved. The result is an analogous of Definition 8.4.1.

The set of operators of the intermediate language is now labelled by sequences, whose values are the indices of the modal operators of the original language; so is its set of proposition letters, as we specify below.

**Definition 8.4.7.** Consider a multimodal language \( \mathcal{MML} := \mathcal{MML}(\text{Index}, P) \), with modal operators in \( \{\Box_a, \Diamond_a : a \in \text{Index}\} \). Then the multimodal language \( \mathcal{IMML} := \mathcal{IMML}(\text{Index}, P) \) is the multimodal language with set of propositions equal to \( \{p_s : p \in P \text{ and } s \in \text{Index}^*\} \), and modal operators in \( \{\Box_s, \Diamond_s : s \in \text{Index}^*\} \).

- Suppose that \( \phi \) is a multimodal formula in \( \mathcal{MML} \). Let \( s \in \text{Index}^* \). The intermediated translation \( \text{Tr}_m(\phi, s) \) of \( \phi \) into the intermediate multimodal language \( \mathcal{IMML} \), for a string \( s \) in \( \text{Index}^* \), is defined as follows:

\[
\begin{align*}
\text{Tr}_m(p, n) & := p_n, \\
\text{Tr}_m(\neg \psi, n) & := \neg \text{Tr}_m(\psi, n), \\
\text{Tr}_m(\psi \land \chi, n) & := \text{Tr}_m(\psi, n) \land \text{Tr}_m(\chi, n), \\
\text{Tr}_m(\psi \lor \chi, n) & := \text{Tr}_m(\psi, n) \lor \text{Tr}_m(\chi, n), \\
\text{Tr}_m(\Diamond_a \psi, s) & := \Diamond_{s*(a)} \text{Tr}_m(\psi, s \ast \langle a \rangle), \\
\text{Tr}_m(\Box_a \psi, s) & := \Box_{s*(a)} \text{Tr}_m(\psi, s \ast \langle a \rangle).
\end{align*}
\]
8.4. Layer by Layer

- $\text{MLT}_x$ denotes the composition of the translation $\text{Tr}_m(\cdot, \epsilon)$ above and $\text{ST}_x$: i.e., for every multimodal formula $\phi$ of $\text{MML}$,

$$\text{MLT}_x(\phi) := \text{ST}_x \circ \text{Tr}_m(\phi, \epsilon),$$

where $\epsilon$ is the empty sequence. The \textit{multimodal layered relational translation $\text{MLT}$} is $\text{MLT}_x$ for some first-order variable $x$.

- The \textit{layered fragment} of first-order logic is the range of the multimodal layered translation.

The following result is proved as in the unimodal case. We let the reader spell out the details or check the proof in [AGHdR00].

**Theorem 8.4.8.** Let $\phi$ be a multimodal formula. Then $\phi$ is satisfiable iff $\text{MLT}_x(\phi)$ is so, for any first-order variable $x$. \hfill $\square$

8.4.3 Finale

The layered translation constitutes a new way of turning modal problems into first-order problems. The new translation, and the intermediate translation into multimodal languages are both \textit{conservative}, in the sense that they can work \textit{on top} of existing strategies for first-order and modal logics, respectively. We discuss the latter fact in Chapter 9, and put at work the former in Section 8.5 below.

In particular, the layered translation is a refinement of the standard translation; hence the layered fragment is contained in the fragment identified by the standard translation and in its generalisation, i.e., the guarded fragment. Thus we can use \textit{any decision procedure} and strategy tuned for the latter, see [dNdR02]. We state this precisely as follows.

**Theorem 8.4.9.** Let $\mathcal{R}_{\text{ST}}(\phi)$ and $\mathcal{R}_{\text{LT}}(\phi)$ denote the sets of clauses derivable by means of resolution and factoring from $\text{ST}(\phi)$ and $\text{LT}(\phi)$ respectively. Then $|\mathcal{R}_{\text{ST}}(\phi)| \leq |\mathcal{R}_{\text{LT}}(\phi)|$. The same result holds with $\text{LT}$ replaced by $\text{MLT}$. \hfill $\square$

The above result yields that first-order theorem provers will perform at least as well on the layered translations as on the standard one. In the section below, we report our experimental comparison between the two translations. This witnesses the improvements — up to orders of magnitude better — that the theorem prover \textit{SPASS} gains by means of the new translation $\text{LT}$. 
8.5 Experimental Comparisons

In this section, we compare the two translations, the standard versus the layered one, by running some experimental tests. First we briefly introduce and comment on the problem set and prover used in our experiments, then we display and explain the results.

8.5.1 The Problem Set

Our tree-based heuristics was evaluated by running a series of tests on a number of problem sets. Our main focus was on the modal QBF benchmark. This benchmark is the basic yardstick for the Tableaux Non-Classical Systems Comparisons (TANCS) competition on theorem proving and satisfiability testing for non-classical logics, see [TAN00]. It is a random problem generator that has been designed to evaluate solvers of either satisfiable or unsatisfiable problems of the modal logic K.

The modal formulas of this benchmark are generated by means of quantified Boolean formulas. For the generation, first a quantified Boolean formula is generated with $C$ clauses, quantifier alternation depth equal to $D$, and maximum number of variables $V$ for each alternation. Then the resulting quantified Boolean formula is translated into modal logic via an encoding that was originally proposed by Halpern, see [Hal95]. See [HdR01] for a detailed analysis of the QBF test set.

The output of the QBF generator is a file named p-qbf-cnf-K4-Cn-Vm-Dl, in which the numerical parameters are explained as follows: $n$ is the number of clauses; $m$ the number of variables; $D$ the quantifier depth.

8.5.2 The Theorem Prover

Tests were performed on a Sun ULTRA II (300 MHz) with 1Gb RAM, under Solaris 5.2.5, with the automated theorem prover SPASS version 1.0.3. This is an automated theorem prover for full sorted first-order logic with equality that extends superposition by sorts and a splitting rule for case analysis; it has been in development at the Max-Planck-Institut für Informatik for a number of years, see [SPA00]. SPASS was invoked with the automode switched on; no sort constraints were built, and both optimized and strong Skolemization were disabled.

8.5.3 Experimental Comparisons

The modal QBF benchmark

To explore the behaviour of our heuristics in a large portion of the landscape of the K-satisfiability problem, we randomly generated sets of 10 problems by means
8.5. Experimental Comparisons

<table>
<thead>
<tr>
<th>C/V/D</th>
<th>ST Average Time</th>
<th>LT Average Time</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>5/2/1</td>
<td>9.6222</td>
<td>0.5346</td>
<td>1</td>
</tr>
<tr>
<td>10/2/1</td>
<td>3.9909</td>
<td>0.41734</td>
<td>1</td>
</tr>
<tr>
<td>15/2/1</td>
<td>0.13172</td>
<td>0.10839</td>
<td>0</td>
</tr>
<tr>
<td>5/2/2</td>
<td>450.44</td>
<td>0.66141</td>
<td>3</td>
</tr>
<tr>
<td>10/2/2</td>
<td>370.09</td>
<td>0.78297</td>
<td>3</td>
</tr>
<tr>
<td>15/2/2</td>
<td>147.38</td>
<td>0.75656</td>
<td>2</td>
</tr>
<tr>
<td>5/2/3</td>
<td>N/A</td>
<td>36.048</td>
<td>N/A</td>
</tr>
<tr>
<td>10/2/3</td>
<td>N/A</td>
<td>58.886</td>
<td>N/A</td>
</tr>
<tr>
<td>15/2/3</td>
<td>2094.4</td>
<td>94.192</td>
<td>1</td>
</tr>
<tr>
<td>5/2/4</td>
<td>N/A</td>
<td>20.362</td>
<td>N/A</td>
</tr>
<tr>
<td>10/2/4</td>
<td>N/A</td>
<td>33.084</td>
<td>N/A</td>
</tr>
<tr>
<td>15/2/4</td>
<td>2094.4</td>
<td>35.068</td>
<td>1</td>
</tr>
<tr>
<td>5/2/5</td>
<td>N/A</td>
<td>1136.1</td>
<td>N/A</td>
</tr>
<tr>
<td>10/2/5</td>
<td>N/A</td>
<td>2896</td>
<td>N/A</td>
</tr>
<tr>
<td>15/2/5</td>
<td>N/A</td>
<td>3758.2</td>
<td>N/A</td>
</tr>
<tr>
<td>5/3/1</td>
<td>7.1862</td>
<td>2047.9</td>
<td>2</td>
</tr>
<tr>
<td>10/3/1</td>
<td>9.752</td>
<td>2324.2</td>
<td>2</td>
</tr>
<tr>
<td>15/3/1</td>
<td>14.066</td>
<td>1506.8</td>
<td>2</td>
</tr>
<tr>
<td>5/3/2</td>
<td>N/A</td>
<td>7.0931</td>
<td>N/A</td>
</tr>
<tr>
<td>10/3/2</td>
<td>N/A</td>
<td>8.3192</td>
<td>N/A</td>
</tr>
<tr>
<td>15/3/2</td>
<td>N/A</td>
<td>9.3902</td>
<td>N/A</td>
</tr>
<tr>
<td>5/3/3</td>
<td>N/A</td>
<td>1445.2</td>
<td>N/A</td>
</tr>
<tr>
<td>10/3/3</td>
<td>N/A</td>
<td>4045.1</td>
<td>N/A</td>
</tr>
<tr>
<td>15/3/3</td>
<td>N/A</td>
<td>4865.4</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 8.1: Comparison by average time.

of the modal QBF generator for different sets of parameters. Table 8.1 compares the average time in CPU seconds, while Table 8.2 compares the average number of clauses for two methods: layered (our improved translation, see Definition 8.4.1) and standard (the relational method, see Definition 7.3.1). The shorthand C/V/D in the first column denotes the number of clauses, the number of variables, and the depth used in the generation. Columns labelled by M show the magnitude of the difference between the preceding two columns, i.e., round(log(N/N')). We used a time out of 3 hours on a shared machine; N/A indicates that a value is not available due to a time out.

As can easily be seen from Tables 8.1 and 8.2, our improved translation method outperformed the standard translation in every case, both in computing time (CPU time) and number of clauses generated. This is not only an average behaviour, but it was observed in each instance. For some configurations the drop
in computing time is as much as three orders of magnitude or two. This is always the case when the depth of the formula increases, as our translation cleverly exploits the modal depth information. The average number of clauses generated was nearly always smaller by one order of magnitude.

In Figure 8.2 we display a sample from our experimental results: 64 instances of the 10/3/1 configuration. The top curve indicates the CPU time needed by the standard relational translation, and the bottom one the CPU time needed by the layered translation. Note that the standard translation can be very sensitive to certain hard problems, which results in significant differences between easy and hard instances; the layered method responds in a much more controlled way to hard problems. Interestingly, the curves follow each other, even at many orders of magnitude of difference. This shows that our heuristics does not change the

<table>
<thead>
<tr>
<th>C/V/D</th>
<th>ST Average Clause Number</th>
<th>LT Average Clause Number</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>5/2/1</td>
<td>5695</td>
<td>726</td>
<td>1</td>
</tr>
<tr>
<td>10/2/1</td>
<td>2367</td>
<td>546</td>
<td>1</td>
</tr>
<tr>
<td>15/2/1</td>
<td>10</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>5/2/2</td>
<td>27209</td>
<td>437</td>
<td>2</td>
</tr>
<tr>
<td>10/2/2</td>
<td>22306</td>
<td>500</td>
<td>2</td>
</tr>
<tr>
<td>15/2/2</td>
<td>11368</td>
<td>473</td>
<td>1</td>
</tr>
<tr>
<td>5/2/3</td>
<td>N/A</td>
<td>10714</td>
<td>N/A</td>
</tr>
<tr>
<td>10/2/3</td>
<td>N/A</td>
<td>15395</td>
<td>N/A</td>
</tr>
<tr>
<td>15/2/3</td>
<td>45789</td>
<td>20786</td>
<td>1</td>
</tr>
<tr>
<td>5/2/4</td>
<td>N/A</td>
<td>3121</td>
<td>N/A</td>
</tr>
<tr>
<td>10/2/4</td>
<td>N/A</td>
<td>4971</td>
<td>N/A</td>
</tr>
<tr>
<td>15/2/4</td>
<td>N/A</td>
<td>5358</td>
<td>N/A</td>
</tr>
<tr>
<td>5/2/5</td>
<td>N/A</td>
<td>48546</td>
<td>N/A</td>
</tr>
<tr>
<td>10/2/5</td>
<td>N/A</td>
<td>91767</td>
<td>N/A</td>
</tr>
<tr>
<td>15/2/5</td>
<td>N/A</td>
<td>106870</td>
<td>N/A</td>
</tr>
<tr>
<td>5/3/1</td>
<td>105960</td>
<td>4372</td>
<td>1</td>
</tr>
<tr>
<td>10/3/1</td>
<td>108110</td>
<td>5390</td>
<td>1</td>
</tr>
<tr>
<td>15/3/1</td>
<td>72605</td>
<td>6687</td>
<td>1</td>
</tr>
<tr>
<td>5/3/2</td>
<td>N/A</td>
<td>1804</td>
<td>N/A</td>
</tr>
<tr>
<td>10/3/2</td>
<td>N/A</td>
<td>2221</td>
<td>N/A</td>
</tr>
<tr>
<td>15/3/2</td>
<td>N/A</td>
<td>2687</td>
<td>N/A</td>
</tr>
<tr>
<td>5/3/3</td>
<td>N/A</td>
<td>52153</td>
<td>N/A</td>
</tr>
<tr>
<td>10/3/3</td>
<td>N/A</td>
<td>107800</td>
<td>N/A</td>
</tr>
<tr>
<td>15/3/3</td>
<td>N/A</td>
<td>119150</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 8.2: Comparison by average number of generated clauses.
nature of the problem: it simply makes it much easier for the resolution prover.

The latter phenomenon can also be observed more globally. The plots in Figure 8.3 were obtained with the following settings: $V = D = 2$, while $C$ ranged from 2 to 40. Figures 8.3 (a) and (b) show the number of clauses generated and the CPU time needed, respectively, for the standard and layered method, while 8.3 (c) plots the proportion of satisfiable instances as $C$ increases. The curves for the standard and layered methods are very similar, with the layered method lacking the sharp lows and highs that seem to be characteristic for the relational method. Both display a clear easy-hard-easy behaviour, but the layered translation is better by several orders of magnitude.

Note that the biggest improvements are achieved in the satisfiable region, i.e., for $C < 26$. Once we were confident that the layered method consistently displayed a good behaviour and a significant improvement over the standard translation, we ran the standardized tests provided by TANCS (64 instances randomly generated with the 20-clauses/2-variables/2-depth parameters); see Figure 8.4 for the outcomes.

Finally, to obtain the results in Figure 8.5 we generated 64 instances of problems for 2 and 3 variables with depths ranging from 1 to 6, again with a time out of 3 hours. The figure shows the average values we obtained. We ran the
Figure 8.3: Easy-hard-easy.
8.5. **Experimental Comparisons**

---

**Figure 8.4:** Standard TANCS Test 20/2/2.

---

**Figure 8.5:** LT Tests on 64 Problem Instances.
same tests with the standard instead of the layered translation, but even for moderate depths the computing time and number of clauses exceeded the available resources.

Additional Tests
Given that the problems returned by the QBF generator were generally too hard for the prover using the standard translation, we also performed tests with a number of easier problem sets — so to speak — that include the one proposed by Heuerding and Schwendimann in [HS96], which were used in, for example, Tableaux'98. Invariably, the layered translation outperformed the standard one; it was able to solve substantially harder instances in all categories.

8.6 Conclusions

8.6.1 Synopsis
In this chapter, a new relational translation of modal formulas into first-order formulas is described. The key idea underlying this refinement is to encode a very strong form of the tree model property in an intermediate translation into multi-modal languages, and hence in a translation into first-order languages — see Definition 8.4.1.

Using our tree-based heuristics, we have consistently observed improvements, both in terms of the number of clauses generated and in terms of CPU time used.

8.6.2 Discussion

So the methodology used pays off: instead of modifying theorem provers, or develop new ones from scratch, we reuse existing ones and optimise their behaviour by refining the encoding of the modal problems. In the future, it could be interesting to explore the behaviour of our heuristics in larger parts of the problem space.

It could also be interesting to see how to encode weaker forms of the tree model property to boost the performance of resolution provers on input from different modal logics, such as K4, S4, and temporal logic.

In Chapter 9, we appeal to the same semantic intuitions underpinning the results of this chapter. As noticed above, the intermediate translation is already based on these: it makes explicit, at the syntactic level, that the satisfiability of a modal formula can be tested propositionally, proceeding layer by layer (i.e., index by index) in the modal formula. It is precisely this semantic property which is used in procedures for modal logics that are based on propositional solvers, as explained in the following chapter.