Handbook of Modal Logic – P. Blackburn et al. (Editors)

4

COMPUTATIONAL MODAL LOGIC

Ian Horrocks, Ullrich Hustadt, Ulrike Sattler, and Renate Schmidt

1 Introduction . . . 181
2 Syntax, semantics, and reasoning problems of modal logics . . . 183
3 Translation-based methods . . . 186
3.1 Local satisfiability in multi modal $\mathbf{K}_n$ . . . 186
3.2 Global satisfiability, non-logical axioms, transitive modalities, and $\mathbf{K4}_n$ . . . 200
3.3 Converse modalities and the modal logics $\mathbf{KB}_n$ and $\mathbf{KB4}_n$ . . . 202
3.4 Implementation and optimisation . . . 203
3.5 Other extensions (counting, nominals) . . . 207
4 Tableau-based algorithms . . . 207
4.1 Tableau algorithms in general . . . 208
4.2 Local satisfiability for multi modal $\mathbf{K}_n$ . . . 209
4.3 Transitive modalities and $\mathbf{K4}_n$ . . . 217
4.4 Non-logical axioms and background theories . . . 219
4.5 Converse modalities . . . 222
4.6 Converse modalities and background theories . . . 223
4.7 Other extensions (counting, nominals, transitive closure, and fixpoints) . . . 224
5 Other computational approaches . . . 226
5.1 Automata-based algorithms . . . 226
5.2 Modal resolution . . . 227
5.3 Sequent-based approaches . . . 229
5.4 Inverse method . . . 230
6 Other reasoning problems . . . 231
6.1 Model checking . . . 231
6.2 Proof checking . . . 231
6.3 Computing correspondences . . . 232
6.4 Model generation . . . 233
6.5 Bisimulation . . . 234
6.6 Modal logic programming . . . 236
7 Review and Discussion . . . 237

1 INTRODUCTION

As we have seen in preceding chapters, the worst case complexity of basic reasoning tasks, such as deciding the satisfiability of a modal formula, is at least NP-complete for almost all modal logics. Moreover, for logics extended with features that are useful in practice, the worst case complexity can be much higher, e.g., ExpTime-complete for $\mathbf{K}_n$ extended with non-logical

axioms (background theories), and NExpTime-complete for $\mathbf{K}_n$ extended with converse modalities, graded modalities and nominals.

Some may regard these results as discouraging and the question arises whether automated computation with such logics can be feasible in practice. Fortunately, the kinds of pathological formulae/theories that give rise to these worst case results seem to be rarely encountered in realistic applications, and this has allowed for the successful development and deployment of automated reasoning systems for modal logics and their notational variant, description logics; see Chapter 13 of this handbook. Applications of such systems include, e.g., multi-agent systems [53, 60, 196], configuration [137], conceptual modelling [73], information integration [32], and ontology tools and applications [125, 131, 167, 189, 138, 197].

Even for application derived formulae/theories, however, naive implementations of theoretical proof systems, such as the tableau calculi presented in Chapter 2 of this handbook, are unlikely to be of practical utility. As pointed out in [40], without the use of an analytic cut rule, the minimal length of proofs using these calculi can exceed that of proofs using the truth table method for certain propositional (and modal) formulae. Further, not only is it important that short proofs exist, but also how we go about finding a proof or a counter-model. Much of the work presented in this chapter deals with techniques that reduce the size of the search space or help to traverse the search space more efficiently. Successful modern reasoning systems crucially employ specialised reasoning techniques along with optimisations to dramatically improve typical case performance; cf. for example [85, 90, 97, 115, 116, 158, 159]. In this chapter, we focus on reasoning and optimisation techniques used in tableau-based algorithms and translation-based methods.

Translation-based methods make use of the fact that a wide variety of modal logics can be translated into first-order logic; in fact, they can be considered as characterising certain fragments of first-order logic as explained in Section 2 of Chapter 1 of this handbook. To the translated modal formulae, we can apply first-order reasoning methods, in particular, refinements of resolution [16]. Using this combination of a translation method and resolution has some obvious advantages. Any modal logic which can be embedded into first-order logic can be treated. The translations are straightforward, and can be performed in time $O(n \log n)$, so the engineering effort is minimal. For the resolution part, standard resolution provers can be used, or otherwise they can be used with small adaptations. Modern resolution provers [169, 183, 194] are among the most sophisticated and fastest first-order logic theorem provers currently available. The translation method is generic, it can handle first-order modal logics, undecidable modal logics, and combinations of modal and non-modal logics. In all cases, soundness and completeness of the method is immediate from results showing that the translation is satisfiability equivalence preserving and the soundness and completeness of the resolution calculus for first-order logic. The semi-decidability of first-order logic and the behaviour of first-order resolution on first-order formulae does not give us, however, any immediate insight into the modal fragment of first-order logic, which certainly is decidable, or the behaviour of first-order resolution on translated modal formulae. While termination of a resolution derivation from a translated modal formula is not always guaranteed, there are various ways, using different translations and different refinements of resolution, of obtaining translation-based decision procedures. In Section 3, we discuss some of these approaches and illustrate them using the modal logics $\mathbf{K}_n$, $\mathbf{K4}_n$, $\mathbf{KB}_n$, $\mathbf{K}_n^{\smile}$ ($\mathbf{K}_n$ with converse modalities), and $\mathbf{KB4}_n$. Also, using the modal logic $\mathbf{K}_n$, we want to provide some fundamental understanding of how modern resolution provers work in general, what kind of optimisations are available, and how they can be used to provide effective and practical decision procedures for modal logics.

Tableau-based algorithms are closely related to the prefixed tableau systems presented in Section 6 of Chapter 2 of this handbook. In Section 4, we first explain the exact relationship between the two before describing a tableau algorithm which decides the satisfiability of formulae in the basic multi-modal logic $\mathbf{K}_n$. We then discuss implementation and optimisation techniques which can be used to turn this tableau algorithm into an effective and practical decision procedure for $\mathbf{K}_n$. Following the same structure, we also describe tableau-based algorithms for the modal logics $\mathbf{K4}_n$, $\mathbf{K}_n$ with non-logical axioms, $\mathbf{K}_n^{\smile}$, and their combinations and discuss implementation issues of those algorithms. Whereas the $\mathbf{K}_n$ tableau algorithm terminates "automatically", we use certain cycle detection mechanisms to ensure termination for other modal logics. It can be easily seen that these mechanisms must be chosen carefully to preserve correctness of the algorithm and, at the same time, to enable termination as soon as possible so as to avoid an unnecessarily long search. Interestingly, it has been shown by state of the art description logic reasoners [159, 90, 160] that such tableau algorithms are amenable to optimisation, and that they behave better than their worst-case complexity or that of the corresponding reasoning problem suggest: they implement non-deterministic double exponential decision procedures for logics that are ExpTime-complete.

In Section 5, we give an overview of alternative computational approaches to the satisfiability problem in modal logics. These include automata-based algorithms, direct resolution, the inverse method, and sequent-based approaches. In Section 6, we survey reasoning problems other than satisfiability and provability which are relevant for applications of modal logics, namely, model checking, proof checking, and computing correspondence properties for modal axiom schemata. Finally, we conclude the chapter with a brief review and discussion of current and future research.

2 SYNTAX, SEMANTICS, AND REASONING PROBLEMS OF MODAL LOGICS

Throughout this chapter, we use a notation that is compatible with the one presented in Chapter 1 of this handbook. We will use the symbols $p, q, p_i, q_i, \ldots$ for propositional variables. Here, we will be concerned with extensions and variants of the multi-modal logic $\mathbf{K}_n$. The set of $\mathbf{K}_n$ *formulae* is the smallest set that contains all propositional variables, is closed under Boolean operators, and contains $[i]\psi$ and $\langle i\rangle\psi$ for each $1 \leq i \leq n$ and each $\mathbf{K}_n$ formula ψ. Formulae of the form $[i]\psi$ and $\langle i\rangle\psi$ are called *box formulae* and *diamond formulae*, respectively. In different sections, we will consider different normal forms of $\mathbf{K}_n$ formulae, and thus we are generous here and allow all kinds of Boolean operators and abbreviations, e.g. $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\top$ (for any tautology), $\bot$ (for $\neg\top$), etc.

As usual, the semantics of $\mathbf{K}_n$ is defined in terms of relational, Kripke structures or frames. A *frame* is a tuple $\langle W, R\rangle$ of a non-empty set W (of worlds) and a mapping R from natural numbers i, $1 \leq i \leq n$ to binary relations over W, thus $R(i) \subseteq W \times W$. Here and in the rest of the chapter, we use R_i as an abbreviation of $R(i)$, and we say that w is *i-accessible from v* if $R_i(v, w)$. A *model* is given by a triple $\mathfrak{M} = \langle W, R, V\rangle$, where $\langle W, R\rangle$ is a frame and V is a mapping from propositional variables to subsets of W. The notion of a formula ψ *being true* in a model $\mathfrak{M}$ at a world $w \in W$ is inductively defined as follows (we omit the definition for most Boolean operators).

$$
\begin{array}{lcll}
\mathfrak{M}, w & \models & p & \text{iff } w \in V(p) \\
\mathfrak{M}, w & \models & \neg\psi & \text{iff not } \mathfrak{M}, w \models \psi \\
\mathfrak{M}, w & \models & \psi \wedge \phi & \text{iff } \mathfrak{M}, w \models \psi \text{ and } \mathfrak{M}, w \models \phi \\
\mathfrak{M}, w & \models & \psi \vee \phi & \text{iff } \mathfrak{M}, w \models \psi \text{ or } \mathfrak{M}, w \models \phi \\
\mathfrak{M}, w & \models & [i]\psi & \text{iff } \mathfrak{M}, v \models \psi, \text{for all } v \text{ with } R_i(w, v) \\
\mathfrak{M}, w & \models & \langle i\rangle\psi & \text{iff } \mathfrak{M}, v \models \psi, \text{for some } v \text{ with } R_i(w, v)
\end{array}
$$

A modal formula ϕ is *satisfiable* in $\mathbf{K}_n$ if there exists some $\mathfrak{M} = \langle W, R, V\rangle$ such that, for some $w \in W$, $\mathfrak{M}, w \models \phi$. In this case, we say that ϕ is *satisfied in* $\mathfrak{M}$. ϕ is *valid* in $\mathbf{K}_n$ if, for every $\mathfrak{M} = \langle W, V\rangle$ and every $w \in W$, $\mathfrak{M}, w \models \phi$; ϕ and ψ are *equivalent* if $\phi \leftrightarrow \psi$ is valid.

As usual, satisfiability and validity are inter-reducible, i.e., ϕ is satisfiable iff $\neg\phi$ is *not* valid, and ϕ is valid iff $\neg\phi$ is *un*satisfiable. Thus, in what follows, we will mostly concentrate on a single inference problem, namely satisfiability testing. It is well-known that the satisfiability problem (and thus validity) in $\mathbf{K}_n$ is PSpace-complete [93, 129] (see also Chapter 3 of this handbook), and there are various decision procedures for this problem [49, 93, 129] and implementations thereof [87, 90, 120, 159]. Many of these procedures exploit the fact that any satisfiable $\mathbf{K}_n$ formula is satisfied in a finite tree model (i.e., one where the relational structure of the frame forms a finite tree) of depth linear in the size of the input formula. In this chapter, we will discuss in depth a resolution-based algorithm (in Section 3) and a tableau-based algorithm (in Section 4) for the satisfiability of $\mathbf{K}_n$, and then explain how these two basic algorithms can be modified to also decide more expressive modal logics.

We will often restrict our attention to formulae in *negation normal form (NNF)*. In formulae in NNF, $\wedge$, $\vee$, and $\neg$ are the only Boolean connectives used, and negation occurs only in front of propositional variables. Each formula of $\mathbf{K}_n$ and all extensions of $\mathbf{K}_n$ discussed in this chapter can be easily transformed into an equivalent formula in NNF in linear time, by pushing negation inwards, using a combination of de Morgan's laws and the duality between box and diamond formulae.

In this chapter we refer to a number of extensions of $\mathbf{K}_n$ which we define in the following.

$\mathbf{K4}_n$, $\mathbf{KB}_n$, and $\mathbf{KB4}_n$. We will discuss decision procedures for $\mathbf{K4}_n$, the multi-modal logic of *transitive frames*, $\mathbf{KB}_n$, the multi-modal logic of *symmetric frames*, and $\mathbf{KB4}_n$, the multi-modal logic of *symmetric and transitive frames*. All these logics share the same language with $\mathbf{K}_n$, but their semantics is based on different classes of frames. As $\mathbf{K4}_n$ models, we only consider those models that are based on frames (W, R) in which each R_i is *transitive*, i.e., where for any $u, v, w \in W$, $R_i(u, v)$ and $R_i(v, w)$ imply $R_i(u, w)$. For example, $\langle i\rangle\langle i\rangle p \wedge [i]\neg p$ is $\mathbf{K}_n$ satisfiable, whereas the same formula is $\mathbf{K4}_n$ unsatisfiable. As $\mathbf{KB}_n$ models, we only consider models that are based on frames (W, R) in which each R_i is *symmetric*, i.e., where for any $u, v \in W$, $R_i(u, v)$ implies $R_i(v, u)$. An example of a formula which is $\mathbf{K}_n$ and $\mathbf{K4}_n$ satisfiable, but $\mathbf{KB}_n$ unsatisfiable, is $\neg p \wedge \langle i\rangle[i]p$. Finally, $\mathbf{KB4}_n$ models are based on frames (W, R) where each R_i is symmetric and transitive. The formula $\langle i\rangle\neg p \wedge \langle i\rangle[i]p$, for example, is $\mathbf{KB4}_n$ unsatisfiable, but satisfiable in $\mathbf{K}_n$, $\mathbf{K4}_n$, and $\mathbf{KB}_n$.

The modal logic $\mathbf{K4}_n$ is axiomatised by the axioms of the modal logic $\mathbf{K}_n$ (see, for example, Chapter 2 of this handbook), plus the axiom schema **4** below. Similarly, $\mathbf{KB}_n$ is axiomatised by adding the axiom schema $\mathbf{B}$ to the axiomatisation of $\mathbf{K}_n$. Finally, for $\mathbf{KB4}_n$, we add both axiom schemas $\mathbf{B}$ and **4**.

Axiom **4** $\quad [i]\phi \rightarrow [i][i]\phi$

Axiom $\mathbf{B}$ $\quad \phi \rightarrow [i]\langle i\rangle\phi$

The modal logic $\mathbf{K4}_n$ is of interest since both tableau-based algorithms and translation-based methods require additional techniques to ensure termination. The modal logic $\mathbf{KB}_n$ only requires a rather straightforward modification of the reasoning procedures we present for $\mathbf{K}_n$, but raises some implementation issues for tableau systems. It is also worthwhile to remember that, in classical tableau systems, the treatment of $\mathbf{KB}_n$ requires some form of *cut rule*. Finally, the procedures we present for $\mathbf{KB4}_n$ combine the techniques we introduce for $\mathbf{K4}_n$ and $\mathbf{KB}_n$.

Non-logical axioms. We consider *background theories*, i.e., finite sets $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ of *non-logical axioms* γ_i. A model $\mathfrak{M}$ *satisfies* a background theory Γ if, for each $w \in W$ and each $\gamma \in \Gamma$, $\mathfrak{M}, w \models \gamma$. As a reasoning problem, we are interested in the satisfiability of a formula ϕ w.r.t. a background theory Γ, i.e., whether there exists a model $\mathfrak{M}$ that satisfies both ϕ and a background theory Γ. Note that, in such a model, ϕ has to be true in at least one world, whereas all formulae in Γ have to be true in all worlds.

Next, we explain why we discuss algorithms that reason w.r.t. background theories. Firstly, considering background theories makes reasoning more difficult, i.e., satisfiability w.r.t. background theories is ExpTime-complete [173], and thus they present a considerable challenge for automated reasoning tools. As with $\mathbf{K4}_n$, tableau algorithms for reasoning w.r.t. background theories no longer terminate on all inputs. However, in contrast to $\mathbf{K4}_n$, background theories allow us to enforce models with paths of exponential length using the standard encoding of incrementation modulo 2^n on n propositional variables representing a binary counter (see, for example, page 14 of [133] for such a formula). The latter can be viewed as a symptom of the increased complexity since we might have to consider and search models with paths of exponential length. Secondly, background theories can be viewed as a weak form of the *universal modality*, i.e., ϕ is satisfiable w.r.t. Γ iff

$$\phi \wedge \bigwedge_{\gamma \in \Gamma} [0]\gamma,$$

is satisfied in a model based on a frame $\langle W, R\rangle$ with $R_0 = W \times W$. In such a model, $[0]$ is called the *universal modality* because it can be used to access all worlds. In [4], it was shown how to reduce satisfiability in $\mathbf{K}_n$ with the universal modality (i.e., where the universal modality might also occur at a deeper modal level) to satisfiability w.r.t. background theories. Thirdly, background theories can be used to "internalise" axioms or "circumscribe" frame conditions. To do this, we first restrict our attention to formulae in negation formal form. As an example, let us consider $\mathbf{K4}_n$ as discussed above and a $\mathbf{K4}_n$ formula ϕ in NNF. It can be shown that ϕ is satisfiable iff ϕ is $\mathbf{K}_n$ satisfiable w.r.t. the background theory

$$\{[i]\psi \rightarrow [i][i]\psi \mid [i]\psi \text{ is a subformula of } \phi \text{ and } 1 \leq i \leq n\}.$$

Finally, background theories are notational variants of description logic *TBoxes* or *terminologies*, which are used in applications to hold the intensional domain knowledge [44, 173]; see also Chapter 13 of this handbook. A TBox is (the description logic variant of) a background theory of the form $\{\phi_i \rightarrow \psi_i \mid 1 \leq i \leq m\}$. Restricting non-logical axioms to implications is (i) not a real restriction: we can transform each background theory Γ into a single implication of the form $\top \rightarrow \bigwedge_{\psi \in \Gamma} \psi$; (ii) quite natural in various application, and (iii) enables the use of efficient optimisation techniques in tableau systems [109], which we will discuss in Section 4.4.

Converse modalities. We discuss modifications of our algorithms to decide satisfiability of $\mathbf{K}_n^{\smile}$ formulae w.r.t. background theories, where $\mathbf{K}_n^{\smile}$ is the extension of $\mathbf{K}_n$ that allows the use of *converse* modal parameters $i^{\smile}$ in modalities. That is, $[i^{\smile}]\phi$ and $\langle i^{\smile}\rangle\phi$ are also well-formed formulae. The mapping R is extended to converse modal parameters as follows: $R_{i^{\smile}} = \{(w, v) \mid$

$R_i(v, w)\}$. Converse modalities are of interest because they occur naturally in applications, e.g., in description logics [105, 171] and temporal logics [163, 187], and because they require reasoning techniques that are able to reason in both directions over relations. For example, to detect the unsatisfiability of the formula $q \wedge \langle i\rangle(p \wedge [i^{\smile}]\neg q)$, one has to reason both ways over R_i. This is similar to the kind of reasoning required for $\mathbf{KB}_n$ but slightly more tricky since, in $\mathbf{K}_n^{\smile}$, reasoning in "both ways" over a relation depends also on the worlds related by R_i. For example, the $\mathbf{KB}_n$ formula $[i]\psi$ is equivalent to the $\mathbf{K}_n^{\smile}$ formula $[i]\psi \wedge [i^{\smile}]\psi$.

Graded/deterministic modalities. We discuss modifications of the tableau algorithm to handle deterministic and graded modalities. The former have (atomic) modal parameters i whose interpretation R_i has to be a functional relation. To understand graded modalities, note that a diamond formula $\langle i\rangle\phi$ can be read as "in at least one i-related world, ϕ is true", and a box formula $[i]\phi$ can be read as "in at most zero i-related worlds, ϕ is not true". *Graded* modalities generalise these formulae: $\mathbf{K}_n^c$ (resp. $\mathbf{K}_n^{\smile,c}$) is the extension of $\mathbf{K}_n$ (resp. $\mathbf{K}_n^{\smile}$) where we also allow for formulae of the form $\langle i\rangle_m\phi$ and $[i]_m\phi$. We read $\langle i\rangle_m\phi$ as "in at least $(m+1)$ i-related worlds, ϕ is true" and $[i]_m\phi$ as "in at most m i-related worlds, ϕ is true". The semantics is extended in the obvious way.

$\mathfrak{M}, w \models \langle i\rangle_m\phi$ iff there are at least $m+1$ worlds $v \in W$ with $R_i(w, v)$ and $\mathfrak{M}, v \models \psi$

$\mathfrak{M}, w \models [i]_m\phi$ iff there are at most m worlds $v \in W$ with $R_i(w, v)$ and $\mathfrak{M}, v \models \psi$

Please note that $\neg[i]_m\phi$ is equivalent to $\langle i\rangle_m\neg\phi$ and that, by adding $\top \to [i]_1\top$ to our background theory, we restrict models to those in which R_i is a functional relation. Thus algorithms that can handle both graded modalities and background theories can also handle deterministic modalities. From a complexity point of view, adding graded/deterministic modalities rarely effects the worst case complexity, e.g., $\mathbf{K}_n^c$ and $\mathbf{K}_n^{\smile,c}$ are both PSpace-complete without background theories [93, 186] and ExpTime-complete w.r.t. background theories [173, 184, 186]. From a practical reasoning perspective, graded modalities add quite some difficulty: consider, e.g., the formulae $[i]_3\top \wedge \langle i\rangle_1(p \vee q) \wedge \langle i\rangle_1(\neg p \vee q)$ and $[i]_1 p \wedge [i]_1\neg p \wedge \langle i\rangle_2 q$. The former is satisfiable, but we have to find that the two diamond formulae can be satisfied via a "common" i-related world. The latter is unsatisfiable, but we have to find that a third i-related world in which neither p nor $\neg p$ holds cannot exist.

Nominals. In their simplest form, nominals [5, 163] are propositional variables that are true in exactly one world; we use $o_1, o_2, \ldots$ for these variables and indicate the availability of nominals in a logic by the superscript $\cdot^o$ as, e.g., in $\mathbf{K}_n^o$. Nominals are of interest for automated reasoning since they destroy the *tree model property (TMP)* (see Chapter 1 of this handbook) of a logic. For example, the formula $o_2 \wedge \langle i\rangle(o_1 \wedge \langle i\rangle o_2)$ has only models with a cycle of length two. We mentioned above that $\mathbf{K}_n$ enjoys this property (and so do its extensions with converse and graded modalities and background theories), and that this property is exploited by tableau- and some resolution-based algorithms. Interestingly, adding nominals to $\mathbf{K}_n^{\smile}$ takes the complexity from PSpace- to ExpTime-completeness [5], and adding nominals to $\mathbf{K}_n^{\smile,c}$ takes the complexity to NExpTime-completeness [186]. Here, we will consider $\mathbf{K}_n^{\smile,o}$ with background theories.

3 TRANSLATION-BASED METHODS

3.1 Local satisfiability in multi modal $\mathbf{K}_n$

As outlined in Chapter 1 of this handbook, using the *standard translation* formulae of the basic modal logic $\mathbf{K}$ can be embedded into first-order logic. This translation is also called the *relational translation*, since it is based on the relational Kripke semantics for modal logic. In the following

we denote this translation by π_r and present here its straightforward generalisation to multi-modal $\mathbf{K}_n$.

$$\begin{aligned}
\pi_r(\top, x) &= \top & \pi_r(\bot, x) &= \bot \\
\pi_r(p, x) &= P_p(x) & \pi_r(\neg\varphi, x) &= \neg\pi_r(\varphi, x) \\
\pi_r(\varphi \star \psi, x) &= \pi_r(\varphi, x) \star \pi_r(\psi, x) \quad \text{for } \star \in \{\wedge, \vee, \rightarrow, \leftrightarrow\} \\
\pi_r([i]\varphi, x) &= \forall y\,(R_i(x, y) \rightarrow \pi_r(\varphi, y)) & \pi_r(\langle i\rangle\varphi, x) &= \exists y\,(R_i(x, y) \wedge \pi_r(\varphi, y))
\end{aligned}$$

In the translation, each propositional variable p is uniquely associated with a unary predicate symbol P_p, while each modal parameter i, $1 \leq i \leq n$, is uniquely associated with a binary predicate symbol R_i. In addition, x is an arbitrary first-order variable while y is an arbitrary first-order variable distinct from x.

This translation is *satisfiability equivalence preserving*, that is, for every modal formula φ, φ is $\mathbf{K}_n$ satisfiable iff $\pi_r(\varphi, x)$ is first-order satisfiable. The free variable x is assumed to be existentially quantified.

The currently predominant method for reasoning about first-order formulae is *resolution* [170]. The method requires that a first-order formula or set of first-order formulae φ is first transformed into a satisfiability equivalent set of clauses N_0. This set of clauses is then *saturated* using the *resolution* rule and *factoring* rule shown in Figure 1. That is, given a clause set N_i, $i \geq 0$, these inference rules are applied (top-down) to clauses already in the set and the conclusion C of such an application is added to N_i to give us the clause set N_{i+1}. This process continues until either (i) the current clause set N_i contains the empty clause (i.e. $\bot$) or (ii) no new clauses can be derived, that is, any conclusion of an application of the resolution and factoring rules to clauses in N_i is already contained in N_i. Any clause set containing the empty clause is unsatisfiable. Thus, in the case (i), N_i is unsatisfiable and by the soundness of the resolution calculus, so is N_0. This implies that φ is unsatisfiable, since N_0 is satisfiable iff φ is satisfiable. In the case (ii), if N_i does not contain the empty clause, N_i is satisfiable and it is possible to construct a model for N_i. By the completeness of the resolution calculus, N_0 is satisfiable. It follows that φ is satisfiable. Due to the undecidability of first-order logic, there is in general no guarantee that, after a finite number of steps, we either always encounter case (i) or (ii). If we apply the inference rules of the resolution calculus in a *fair* way, then the completeness of the resolution calculus ensures that, eventually, the empty clause is derived. However, if the formula φ and the clause set N_0 are satisfiable, then the saturation process may continue indefinitely (unless suitable resolution refinements and/or translation methods are used, see below).

The last observation is also true if the formula φ we are considering belongs to a decidable fragment of first-order logic or is the result of translating a formula belonging to a decidable modal logic like $\mathbf{K}_n$. However, starting with [123], a large number of fragments of first-order logic have been shown to be decidable by resolution or refinements of resolution [48, 65, 76, 82, 117, 179].

Resolution: $\dfrac{C \vee A_1 \quad \neg A_2 \vee D}{(C \vee D)\sigma}$	**Factoring:** $\dfrac{C \vee L_1 \vee L_2}{(C \vee L_1)\sigma}$
where σ is the most general unifier of atoms A_1 and A_2	where σ is the most general unifier of literals L_1 and L_2

Figure 1. The resolution calculus R

Let us start by considering what happens if we apply the basic unrefined resolution calculus to the relational translation of modal formulae. Consider, for example, the modal formula $\varphi_1 = [2](p \rightarrow \langle 1 \rangle p)$. Its translation $\pi_r(\varphi_1, x)$ is the first-order formula $\psi_1^r = \forall y\,(R_2(x,y) \rightarrow (P_p(y) \rightarrow \exists z\,(R_1(y,z) \wedge P_p(z))))$. The corresponding set of clauses N_0^r consists of the two clauses

$$\begin{array}{ll} (1) & \neg R_2(a,x) \vee \neg P_p(x) \vee R_1(x, f(x)) \\ (2) & \neg R_2(a,y) \vee \neg P_p(y) \vee P_p(f(y)) \end{array}$$

where a is a constant introduced during the clause form transformation (for the free variable x in ψ_1^r). We assume that the variables in two clauses to which we want to apply the resolution rule are renamed so that they are variable disjoint, and we consider such *variant clauses* to be equal. There are several possibilities to apply the resolution rule to clauses (1) and (2). For example, we can resolve clause (1) on its second literal, $\neg P_p(x)$, with clause (2) on its third literal, $P_p(f(z))$. The conclusion is

$$[(1)2,\mathrm{R},(2)3] \quad (3) \quad \neg R_2(a, f(z)) \vee R_1(f(z), f(f(z))) \vee \neg R_2(a,z) \vee \neg P_p(z)$$

Clause (3) resolves with clause (2) and yields

$$[(2)3,\mathrm{R},(3)4] \quad (4) \quad \begin{array}{l} \neg R_2(a, f(y)) \vee \neg R_2(a, f(f(y))) \vee R_1(f(f(y)), f(f(f(y)))) \\ \qquad \vee \neg R_2(a,y) \vee \neg P_p(y) \end{array}$$

This clause also resolves with clause (2), and again, the conclusion resolves with (2), and so forth. Repeatedly resolving the newly derived clauses with (2) yields clauses with increasingly more literals and increasingly more complex terms. All these clauses are new, that is, none is the same as an input clause or a clause derived earlier. As the formula φ_1 and its translation ψ_1^r are satisfiable, we are not able to derive the empty clause. So, the saturation process will continue indefinitely.

There are three approaches that we can take to solve this termination problem:

1. We can develop and use *alternative translations* of modal formulae to first-order (clause) logic, and try to find a translation for which resolution is a decision procedure.

2. We can develop and use *refinements of resolution* which restrict the application of the inference rules of the resolution calculus and use powerful redundancy elimination methods.

3. We can develop and use *alternative inference methods* for first-order logic.

These three approaches are not mutually exclusive, in particular, alternative translations can be combined with both refinements of resolution and alternative calculi.

Investigations following the first approach have resulted in the introduction of the *optimised functional translation* of $\mathbf{K}_n$ to sorted first-order logic, more precisely, to a monadic fragment of sorted first-order logic called *basic path logic* [149, 177]. Basic path logic has a sort S_W for the set of worlds W and a sort S_i for each modal parameter i, $1 \leq i \leq n$, in a modal logic. It has n binary functions $[\_\_]_i$ of sort $S_W \times S_i \rightarrow S_W$. Also there are special unary predicates def_i of sort S_W representing subsets of W. Each propositional variable p is uniquely associated with a unary predicate symbols P_p of sort S_W. Commonly, the optimised functional translation π_{of} is defined as a two step process: (i) the application of the functional translation to a modal formula which translates it to basic path logic, followed by (ii) the application of a quantifier exchange operation which converts the first-order formula obtained from the functional translation into

prenex normal form and moves all existential quantifiers inwards as far as possible. Since we focus here only on the satisfiability problem, we can give a simplified presentation of the optimised functional translation obtained in just one step.

$$\begin{aligned}
\pi_{of}(\top, s) &= \top & \pi_{of}(\bot, s) &= \bot \\
\pi_{of}(p, s) &= P_p(s) & \pi_{of}(\neg\varphi, s) &= \neg\pi_{of}(\varphi, s) \\
\pi_{of}(\varphi \star \psi, s) &= \pi_{of}(\varphi, s) \star \pi_{of}(\psi, s) \quad \text{for } \star \in \{\wedge, \vee, \rightarrow, \leftrightarrow\} \\
\pi_{of}([i]\varphi, s) &= \forall y{:}S_i(\mathit{def}_i(s) \rightarrow \pi_{of}(\varphi, [s\, y{:}S_i]_i)) \\
\pi_{of}(\langle i\rangle\varphi, s) &= \mathit{def}_i(s) \wedge \pi_{of}(\varphi, [s\, y{:}S_i]_i)
\end{aligned}$$

where s denotes a *(world) path* and $y{:}S_i$ denotes a variable of sort S_i. The omission of the quantifiers in the definition for $\langle i\rangle\varphi$ is intensional. The optimised functional translation of a modal formula φ in negation normal form is now given by $\pi_{of}(\varphi, x{:}S_W)$, where $x{:}S_W$ is an arbitrary variable of sort S_W, and $x{:}S_W$ as well as the $y{:}S_i$ from $\pi_{of}(\langle i\rangle\varphi, s)$ are free variables which are implicitly existentially quantified.

As an example, consider again the modal formula $\varphi_1 = [2](p \rightarrow \langle 1\rangle p)$. Its optimised functional translation is $\varphi_1^{of} = \pi_{of}(\varphi_1, x{:}S_W) = \forall y{:}S_2\,(\mathit{def}_2(x{:}S_W) \rightarrow (P_p([x{:}S_W\, y{:}S_2]_2) \rightarrow (\mathit{def}_1([x{:}S_W\, y{:}S_2]_2) \wedge P_p([[x{:}S_W\, y{:}S_2]_2\, z{:}S_1]_1))))$.

In the representation of paths we often remove all occurrences of the binary functions $[\_\,\_]_i$ except for the outermost occurrence and also leave out the index of that remaining occurrence, e.g. $[[x{:}S_W\, y{:}S_2]_2\, z{:}S_1]_1$ is written as $[x{:}S_W\, y{:}S_2\, z{:}S_1]$. It is straightforward to restore the original path based on the remaining information. Intuitively, a path term like $[x{:}S_W\, y{:}S_2\, z{:}S_1]$ represents a path from a world x to another, possible identical, world in a Kripke frame via a series of 'steps' along the accessibility relations of the frame. Here an R_2-step is followed by an R_1-step, which is indicated by the sorts S_2 and S_1 associated with the variables y and z, respectively. The def_i predicates express 'definability' for a world in the sense that $\mathit{def}_i(s)$ is true iff the world s has an i-successor.

The standard translation π_r accommodates axiom schemas like $\mathbf{4}$ and $\mathbf{B}$ by adding first-order formulae

$$(R_{\mathbf{B}}) \qquad \forall x\, y\, (R_i(x, y) \rightarrow R_i(y, x))$$

$$(R_{\mathbf{4}}) \qquad \forall x\, y\, z\, ((R_i(x, y) \wedge R_i(y, z)) \rightarrow R_i(x, z))$$

representing the *relational frame properties* corresponding to these axiom schemas. By contrast, in the case of the optimised functional translation, we add so-called *functional frame properties* in the form of (conditional) equations between path terms, for example

$$(F_{\mathbf{B}}) \qquad \begin{aligned} \forall x{:}S_W \forall y{:}S_i \exists z{:}S_i\, (&\mathit{def}_i(x{:}S_W) \rightarrow \mathit{def}_i([x{:}S_W\, y{:}S_i]) \wedge \\ &\mathit{def}_i(x{:}S_W) \rightarrow x{:}S_W = [[x{:}S_W\, y{:}S_i]\, z{:}S_i]) \end{aligned}$$

$$(F_{\mathbf{4}}) \qquad \begin{aligned} \forall x{:}S_W \forall y{:}S_i \forall z{:}S_i \exists u : S_i\, ((&\mathit{def}_i(x{:}S_W) \wedge \mathit{def}_i([x{:}S_W\, y{:}S_i])) \rightarrow \\ &[[x{:}S_W\, y{:}S_i]\, z{:}S_i] = [x{:}S_W\, u : S_i]) \end{aligned}$$

Functional frame properties corresponding to axiom schemas $\mathbf{5}$, $\mathbf{D}$, $\mathbf{T}$, $\mathbf{G}$, as well as functional frame properties corresponding to weak density, irreflexivity, and McKinsey's axiom can be found in [177].

THEOREM 1 ([149]). *Let $\mathbf{K}_n\Sigma$ be a complete modal logic such that the functional frame properties corresponding to the axiom schemas in Σ are a set of first-order formulae F_Σ. Then φ is satisfiable in $\mathbf{K}_n\Sigma$ iff $F_\Sigma \wedge \pi_{of}(\varphi, x{:}S_W)$ is first-order satisfiable.*

If we are only interested in establishing the satisfiability of formulae in the basic modal logic $\mathbf{K}_n$ or extensions of $\mathbf{K}_n$ by the axiom schema $\mathbf{D}$ for some or all modalities, then the use of sorted first-order logic and binary function symbols can be avoided by using k-ary predicates where the sort information is coded into the predicate names [95, 116]. The k-ary predicate symbols are $P_{p,\sigma}$ and $def_{i,\sigma}$ where p denotes a propositional symbol, σ is a k-sequence of natural numbers and $1 \leq i \leq n$, $n \geq 0$. We use $\overline{x}$ to denote a sequence of variables $x_1, \ldots, x_k$, and we use 'ϵ' and '.' for the empty sequence and the concatenation operation on sequences, respectively.

$$\pi'_{of}(\top, \overline{x}, k, \sigma) = \top \qquad \pi'_{of}(\bot, \overline{x}, k, \sigma) = \bot$$

$$\pi'_{of}(p, \overline{x}, k, \sigma) = \begin{cases} P_{p,\epsilon} & \text{if } \sigma = \epsilon \text{ and } k = 0 \\ P_{p,\sigma}(x_1, \ldots, x_k) & \text{otherwise} \end{cases}$$

$$\pi'_{of}(\neg\varphi, \overline{x}, k, \sigma) = \neg\pi'_{of}(\varphi, \overline{x}, k, \sigma)$$

$$\pi'_{of}(\varphi \star \psi, \overline{x}, k, \sigma) = \pi'_{of}(\varphi, \overline{x}, k, \sigma) \star \pi'_{of}(\psi, \overline{x}, k, \sigma) \quad \text{for } \star \in \{\wedge, \vee\}$$

$$\pi'_{of}([i]\varphi, \overline{x}, k, \sigma) = \forall x_{n+1}\,(def_{i,\sigma}(\overline{x}) \rightarrow \pi'_{of}(\varphi, \overline{x}.x_{k+1}, k{+}1, \sigma.i))$$

$$\pi'_{of}(\langle i\rangle\varphi, \overline{x}, k, \sigma) = def_{i,\sigma}(\overline{x}) \wedge \pi'_{of}(\varphi, \overline{x}.x_{k+1}, k{+}1, \sigma.i)$$

The translation of a modal formula φ in negation normal form is given by $\pi'_{of}(\varphi, \epsilon, 0, \epsilon)$. In the case of the modal logic $\mathbf{KD}_n$, and in fact for any modal logic where an accessibility relation R_i is serial, all occurrences of $def_{i,\sigma}$ can be replaced with the logical constant $\top$.

The translation π'_{of}, called the polyadic optimised functional translation, takes advantage of two observations:

1. All paths in $\pi_{of}(\varphi, x{:}S_W)$ start with $x{:}S_W$ and since this variable is free, it is implicitly existentially quantified, and is interpreted as a constant. Therefore, removing $x{:}S_W$ from all paths is a satisfiability equivalence preserving transformation.

2. The variables in the paths occurring in $\pi_{of}(\varphi, x{:}S_W)$ are *prefix stable*, that is, for any variable $x_{i+1}{:}S_{j_i}$ there exists a unique prefix $[x{:}S_W\, x_0{:}S_{j_0} \ldots x_i{:}S_{j_i}]$ such that every path containing $x_{i+1}{:}S_{j_i}$ has the form $[x{:}S_W x_0{:}S_{j_0} \ldots x_i{:}S_{j_i} x_{i+1}{:}S_{j_{i+1}} \ldots x_k{:}S_{j_k}]$. Thus, if a variable occurs at position i in one path, then it occurs at position i in all paths. This property is due to a characteristic ordering of variables in the path terms determined by the structure of modal formulae and is a reflection of the tree model property. Also, since variables do not 'move', the sort information can be associated with the position at which a variable occurs instead of with the variable itself. Thus, we can code the sort information into predicate names. Replacing $P_p([x{:}S_W\, x_0{:}S_{j_0} \ldots x_k{:}S_{j_k}])$ by $P_{p,S_W S_{j_0} \ldots S_{j_k}}(x, x_0, \ldots, x_k)$ is therefore a satisfiability equivalence preserving transformation.

Taking both observations together, and also taking advantage of the assumption that each sort S_i is uniquely identified by its index i, we see that we can replace $P_p([x{:}S_W\, x_0{:}S_{j_0} \ldots x_k{:}S_{j_k}])$ with $P_{p,j_0 \ldots j_k}(x_0, \ldots, x_k)$.

For the modal formula $\varphi_1 = [2](p \rightarrow \langle 1\rangle p)$ the translation using π'_{of} is $\psi_1^{of'} = \forall y\,(def_{2,\epsilon} \rightarrow (P_{p,2}(y) \rightarrow (def_{1,2}(y) \wedge P_{p,21}(y, z))))$. The corresponding set $N_0^{of'}$ of clauses again consists of two clauses.

$$(5) \quad \neg def_{2,\epsilon} \vee \neg P_{p,2}(y) \vee def_{1,2}(y)$$
$$(6) \quad \neg def_{2,\epsilon} \vee \neg P_{p,2}(y) \vee P_{p,21}(y, z)$$

Unlike for the clausal form of the relational translation of φ_1, for these two clauses there is no possibility to apply either the resolution rule or the factoring rule. For, we see that $P_{p,21}(y,z)$ in clause (6) is not unifiable with $P_{p,2}(y)$ in clause (5), nor is it unifiable with $P_{p,2}(y)$ in clause (6). So, for this particular example we are able to conclude that $N_0^{of'}$, $\psi_1^{of'}$, and φ_1 are satisfiable without the need to perform a single inference step.

With this translation, is termination of the saturation process guaranteed? Consider the modal formula $\varphi_2 = [2](\neg p \vee [1]q) \vee [2]p$. Its optimised functional translation is

$$\psi_2^{of'} = \forall y(def_{2,\epsilon} \rightarrow (\neg P_{p,2}(y) \vee \forall z(def_{1,2}(y) \rightarrow P_{q,21}(y,z)))) \vee \forall u(def_{2,\epsilon} \rightarrow P_{p,2}(u)).$$

The corresponding set N_0^{of} of clauses consists of just one clause

$$(7) \quad \neg def_{2,\epsilon} \vee \neg P_{p,2}(y) \vee \neg def_{1,2}(y) \vee P_{q,21}(y,z) \vee \neg def_{2,\epsilon} \vee P_{p,2}(u).$$

We consider clauses to be *multisets* of literals, that is, a literal can occur more than once in a clause, as is the case with the literal $\neg def_{2,\epsilon}$ in clause (7). It is sometimes convenient to consider clauses as *sets* of literals. However, this complicates the completeness proof for resolution calculi which commonly proceeds by lifting ground level derivations to the non-ground level. This lifting is easier if clauses are considered to be multisets on both the ground and the non-ground level. Furthermore, multisets make the computational effort explicit which has to go into removing duplicate literals from clauses.

We can resolve clause (7) with itself on the second literal and the last literal. Remember that this means that we first have to generate a variable-disjoint copy of clause (7) to serve as second premise of a resolution step. The conclusion is

$$[(7)2,R,(7)6] \quad (8) \quad \begin{aligned} &\neg def_{2,\epsilon} \vee \neg P_{p,2}(y_1) \vee \neg def_{1,2}(y_1) \vee P_{q,21}(y_1,z_1) \vee \neg def_{1,2}(y_2) \\ &\vee P_{q,21}(y_2,z_2) \vee \neg def_{2,\epsilon} \vee P_{p,2}(u_2) \vee \neg def_{2,\epsilon} \vee \neg def_{2,\epsilon}. \end{aligned}$$

We observe that the number of occurrences of $\neg def_{2,\epsilon}$ has doubled, to four, and also that we now have two subclauses $\neg def_{1,2}(y_1) \vee P_{q,21}(y_1,z_1)$ and $\neg def_{1,2}(y_2) \vee P_{q,21}(y_2,z_2)$ which are variants of each other.[1] Note that these two subclauses are not simply duplicates, so in a clauses-as-sets setting they would still remain, while all the duplicates of $\neg def_{2,\epsilon}$ would not occur. Clause (8) can again be resolved with itself or with clause (7). A possible resolvent is:

$$[(7)2,R,(8)8] \quad (9) \quad \begin{aligned} &\neg def_{2,\epsilon} \vee \neg P_{p,2}(y_3) \vee \neg def_{1,2}(y_3) \vee P_{q,21}(y_3,z_3) \\ &\vee \neg def_{1,2}(y_1) \vee P_{q,21}(y_1,z_1) \vee \neg def_{1,2}(y_2) \vee P_{q,21}(y_2,z_2) \\ &\vee \neg def_{2,\epsilon} \vee P_{p,2}(u_2) \vee \neg def_{2,\epsilon} \vee \neg def_{2,\epsilon} \vee \neg def_{2,\epsilon} \vee \neg def_{2,\epsilon} \end{aligned}$$

We can continue this process indefinitely, producing bigger and bigger clauses. This shows that the saturation process does not terminate.

We observe, however, that resolution is not the only inference rule that can be applied to clause (8): we can also apply the factoring rule. Indeed, there are several possibilities to do so, for example, we can apply the factoring rule to $P_{q,21}(y_1,z_1)$ and $P_{q,21}(y_2,z_2)$. The resulting factor is:

$$[(8)4,F,(8)6] \quad (10) \quad \begin{aligned} &\neg def_{2,\epsilon} \vee \neg P_{p,2}(y_1) \vee \neg def_{1,2}(y_1) \vee \neg def_{1,2}(y_1) \vee P_{q,21}(y_1,z_1) \\ &\vee \neg def_{2,\epsilon} \vee P_{p,2}(u_2) \vee \neg def_{2,\epsilon} \vee \neg def_{2,\epsilon}. \end{aligned}$$

[1] We consider two formulae or clauses to be equal iff they are *variants* of each other, that is, they are syntactically equal modulo variable renaming.

We see that the clause (10) is a subclause of the clause (8) from which it was derived. Clause (10) thus subsumes clause (8). In general, a clause C *subsumes* a clause D iff there is a substitution σ such that $C\sigma$ is a subclause of D. Subsumed clauses are *redundant* and can be removed from a clause set without losing completeness.

Further factoring steps are possible on clause (10), for example, on the two occurrences of $\neg def_{1,2}(y_1)$. It turns out that the final clause that we can derive by a series of factoring steps is a condensation of clause (8). By definition, a *condensation* $\mathrm{Cond}(C)$ of a clause C is a minimal subclause of C which is also an instance of C. A clause C is *condensed* iff there exists no condensation of C which is a strict subclause of C. For any clause C, $\mathrm{Cond}(C)$ subsumes C, and hence C is redundant in the presence of $\mathrm{Cond}(C)$ and can be removed. This justifies that we can systematically replace clauses with their condensation.

The condensation of clauses (7) and (8) are the clauses

$$\begin{array}{ll} (7') & \neg def_{2,\epsilon} \vee \neg P_{p,2}(y) \vee \neg def_{1,2}(y) \vee P_{q,21}(y,z) \vee P_{p,2}(u) \\ (8') & \neg def_{2,\epsilon} \vee \neg P_{p,2}(y_1) \vee \neg def_{1,2}(y_1) \vee P_{q,21}(y_1,z_1) \vee P_{p,2}(u_2). \end{array}$$

These two clauses are variants of each other, that is, we regard them as equal. So, the only clause derivable from (7′) is identical to (7′), which means the saturation process terminates.

It turns out that systematically replacing clauses by their condensation is sufficient to guarantee termination not only for this particular example formula, but for any modal formula in $\mathbf{K}_n$ or $\mathbf{KD}_n$.

THEOREM 2 ([175]). *Let φ be a modal formula and $N_0 = \pi'_{of}(\varphi, \epsilon, 0, \epsilon)$. Then the saturation process from N_0 by the resolution calculus R defined in Figure 1 in which clauses are systematically and eagerly replaced with their condensation always terminates with a clause set N_n, and φ is $\mathbf{K}(\mathbf{D})_n$ unsatisfiable iff N_n contains the empty clause.*

It is important to understand how Theorem 2 is to be interpreted. The theorem says that R plus condensing is a decision procedure for this translation of modal satisfiability problems. It does not stipulate that we must use R plus condensing. Rather, the theorem sets out the minimal requirement or weakest condition we have to impose on a saturation process by R to ensure that it terminates. It states that, as long as we keep clauses condensed, we can use any refinement of the calculus R, we can perform inference steps on any literals in a clause and can perform inference steps in any order, and the saturation process is still guaranteed to terminate. Condensing can be simulated by factoring and subsumption deletion. Consequently, any first-order theorem prover which implements some refinement of R (and subsumption deletion, which is standardly available) can serve as a decision procedure for $\mathbf{K}_n$ and $\mathbf{KD}_n$.

The question as to which particular refinement of resolution to use (determining which inference steps are required for completeness) and which particular strategies and heuristics to use (determining the order in which inference steps are performed) is then subject to both theoretical and empirical investigation. We leave empirical aspects aside for the moment and instead focus on refinements of resolution.

A wide range of refinements of resolution can be formulated in the general resolution calculus of Bachmair and Ganzinger; full details can be found in [16]. In the general resolution calculus, here denoted by $\mathsf{R}^{\succ}_S$, inference rules are parameterised by an admissible ordering $\succ$ on literals and a selection function S. Essentially, an admissible ordering is a total (well-founded) strict ordering on the ground level such that for literals: $\ldots \succ \neg A_n \succ A_n \succ \ldots \succ \neg A_1 \succ A_1$. This is extended to the non-ground level in a canonical manner. A *selection* function S assigns to each clause a possibly empty set of occurrences of negative literals. If C is a clause, then the

Deduction: $\dfrac{N}{N \cup \{\mathrm{Cond}(C)\}}$
if C is either a resolvent or a factor of clauses in N.

Deletion: $\dfrac{N \cup \{C\}}{N}$
if C is redundant in N.

Splitting: $\dfrac{N \cup \{C \vee D\}}{N \cup \{C\} \mid N \cup \{D\}}$
if C and D are variable-disjoint.

Resolution: $\dfrac{C \vee A_1 \quad \neg A_2 \vee D}{(C \vee D)\sigma}$
where (i) σ is the most general unifier of atoms A_1 and A_2, (ii) no literal is selected in C, and $A_1\sigma$ is strictly $\succ$-maximal with respect to $C\sigma$, and (iii) $\neg A_2$ is either selected, or $\neg A_2\sigma$ is maximal with respect to $D\sigma$ and no literal is selected in D.

Positive Factoring: $\dfrac{C \vee A_1 \vee A_2}{(C \vee A_1)\sigma}$
where (i) σ is the most general unifier of atoms A_1 and A_2, and (ii) no literal is selected in C and $A_1\sigma$ is $\succ$-maximal with respect to $C\sigma$.

Figure 2. Expansion and inference rules of $\mathsf{R}_S^{\succ}$

literal occurrences in $S(C)$ are *selected*. No restrictions are imposed on the selection function. The calculus comprises *expansion rules* of the general form

$$\frac{N}{N_1 \mid \cdots \mid N_n}$$

where both the numerator N and the denominators $N_1, \ldots, N_n$ ($n \geq 1$) are finite sets of clauses. Expansion rules are applied top-down. There are three kinds of expansion rules: Deduction, Deletion and Splitting which are defined in Figure 2. The inferences rules consist of the resolution and the factoring rule also defined in Figure 2. The left premise of the resolution rule is called the *positive premise* and the right premise is called the *negative premise*. The implicit assumption is that the premises have no common variables. *Resolvents* are conclusions of resolution steps, while *factors* are conclusions of factoring steps.

A *derivation* in $\mathsf{R}_S^{\succ}$ from a set of clauses N is a finitely branching, ordered tree T with root N and nodes which are sets of clauses. The tree is constructed by applications of the expansion rules to the leaves. We assume that no resolution or factoring inference (on the same premises) is performed twice on the same branch of the derivation. A branch $N(= N_0), N_1, \ldots$ in a derivation T is called a *closed branch* in T iff the clause set $\bigcup_{j \geq 0} N_j$ contains the empty clause, otherwise it is called an *open branch*. We call a branch B in a derivation tree *complete* (with respect to $\mathsf{R}_S^{\succ}$) iff no new successor nodes can be added with $\mathsf{R}_S^{\succ}$ to the endpoint of B, otherwise it is called an *incomplete branch*. A derivation T is a *refutation* iff every path $N(= N_0), N_1, \ldots$ in it is a closed branch, otherwise it is called an *open derivation*.

In general, the calculus $\mathsf{R}_S^{\succ}$ can be enhanced with standard simplification rules such as tautology deletion and subsumption deletion. In fact, it can be enhanced by all simplification rules which are compatible with a general notion of redundancy [16, 18]. For example, C is redundant in $N \cup \{\mathrm{Cond}(C)\}$. A set N of clauses is *saturated up to redundancy* with respect to a particular refinement of resolution if the conclusion of every inference from non-redundant premises in N is either contained in N, or else is redundant in N. A derivation T from N is called *fair* if, for any path $N(= N_0), N_1, \ldots$ in T with *limit* $N_\infty = \bigcup_{j \geq 0} \bigcap_{k \geq j} N_k$, it is the case that each clause C which can be deduced from non-redundant premises in N_∞ is contained in some N_j. Intuitively, fairness means that no non-redundant inferences are delayed indefinitely. For a finite

complete branch $N(= N_0), N_1, \ldots N_n$, the limit N_∞ is equal to N_n.

THEOREM 3 ([18]). *Let T be a fair $\mathsf{R}_S^\succ$ derivation from a set N of clauses. Then*

1. *if $N(= N_0), N_1, \ldots$ is a path with limit N_∞, then N_∞ is saturated (up to redundancy),*
2. *N is satisfiable iff there exists a path in T with limit N_∞ such that N_∞ is satisfiable, and*
3. *N is unsatisfiable iff for every path $N(= N_0), N_1, \ldots$ the clause set $\bigcup_{j\geq 0} N_j$ contains the empty clause.*

As an aside, we note that it follows from the decidability result for the optimised functional translation (Theorem 2) that we can use any instance of $\mathsf{R}_S^\succ$ for the clause sets obtained by applying $\pi_{of'}$ to modal formulae in $\mathbf{K}_n$ or $\mathbf{KD}_n$. In particular, this gives us full flexibility with respect to orderings and selection functions. Furthermore, by Theorem 2, even instances of $\mathsf{R}_S^\succ$ without splitting will terminate.

The purpose of the ordering $\succ$ and the selection function S is to restrict the set of literals in a clause to which resolution and factoring can be applied. This limits the number of inferences performed and consequently reduces the search space. For example, reconsider the clause set N_0^{of} consisting of just the clause

$$(7) \quad \neg def_{2,\epsilon} \vee \neg P_{p,2}(y) \vee \neg def_{1,2}(y) \vee P_{q,21}(y,z) \vee \neg def_{2,\epsilon} \vee P_{p,2}(u).$$

Using an ordering we could restrict inference steps to the literals $P_{q,21}(y,z)$ and $P_{p,2}(u)$. Now $\neg P_{p,2}(y)$ can no longer be resolved with $P_{p,2}(u)$, since $\neg P_{p,2}(y)$ is neither maximal nor selected. Alternatively, using a selection function we could restrict inference steps to the negative literals $\neg def_{2,\epsilon}$ or $\neg P_{p,2}(y)$. Again, no inference steps are possible.

Let us reconsider our very first example, the set of clauses N_0^r obtained via the relational translation of $\varphi_1 = [2](p \rightarrow \langle 1\rangle p)$:

$$(1) \quad \neg R_2(a,x) \vee \neg P_p(x) \vee R_1(x, f(x))$$
$$(2) \quad \neg R_2(a,y) \vee \neg P_p(y) \vee P_p(f(y))$$

If we use an ordering $\succ$ such that $R_1(x, f(x))$ is maximal in clause (1) and $P_p(f(y))$ is maximal in clause (2), then no inference steps are possible on N_0^r. Likewise, if we select the literals $\neg R_2(a,x)$ and $\neg R_2(a,y)$ in their respective clauses, then again no inference steps are possible.

This raises the question whether it is possible to obtain decision procedures for $\mathbf{K}_n$ satisfiability based on the relational translation π_r and the calculus $\mathsf{R}_S^\succ$ by using particular ordering or selection functions. To simplify matters, we use a technique called structural transformation. The purpose of the structural transformation is to convert the first-order translation into a more manageable form. Before we describe it formally, we need to define some basic notions.

The polarity of (occurrences of) modal or first-order subformulae is defined as usual. Any occurrence of a proper subformula of an equivalence has *zero polarity*. For occurrences of subformulae not below a '$\leftrightarrow$' symbol, an occurrence of a subformula has *positive polarity* if it is inside the scope of an even number of (explicit or implicit) negations, and it has *negative polarity* if it is inside the scope of an odd number of negations. For any first-order formula φ, if λ is the position of a subformula in φ, then $\varphi|_\lambda$ denotes the subformula of φ at position λ and $\varphi[\psi \mapsto \lambda]$ is the result of replacing $\varphi|_\lambda$ at position λ by ψ. The set of all the positions of subformulae of φ is denoted by $\Lambda(\varphi)$.

Structural transformation, also referred to as *renaming*, associates a predicate symbol Q_λ and a literal $Q_\lambda(\overline{x})$ with each element λ of $\Lambda \subseteq \Lambda(\varphi)$, where $\overline{x} = x_1, \ldots, x_n$ are the free variables

of $\varphi|_\lambda$, the symbol Q_λ does not occur in φ and two symbols Q_λ and $Q_{\lambda'}$ are equal only if $\varphi|_\lambda$ and $\varphi|_{\lambda'}$ are equivalent formulae. In practice, one may want to use the same symbols for variant subformulae, or subformulae which are obviously equivalent, for example, $\varphi \vee \psi$ and $\psi \vee \varphi$. Let $\mathrm{Def}^+_\lambda(\varphi) = \forall \overline{x}\,(Q_\lambda(\overline{x}) \to \varphi|_\lambda)$ and $\mathrm{Def}^-_\lambda(\varphi) = \forall \overline{x}\,(\varphi|_\lambda \to Q_\lambda(\overline{x}))$. The *definition* of Q_λ is the formula

$$\mathrm{Def}_\lambda(\varphi) = \begin{cases} \mathrm{Def}^+_\lambda(\varphi) & \text{if } \varphi|_\lambda \text{ has positive polarity,} \\ \mathrm{Def}^-_\lambda(\varphi) & \text{if } \varphi|_\lambda \text{ has negative polarity,} \\ \mathrm{Def}^+_\lambda(\varphi) \wedge \mathrm{Def}^-_\lambda(\varphi) & \text{otherwise.} \end{cases}$$

The corresponding clauses are called *definitional clauses*. Now, assume that Λ is a set of positions in a formula φ and that we want to systematically replace subformulae at positions in Λ while adding definitions for the newly introduced predicate symbols. A convenient way to do so, is to start by the renaming innermost subformulae, and then to proceed up towards the root of φ. Formally, define $\mathrm{Def}_\Lambda(\varphi)$ inductively by:

$$\mathrm{Def}_\emptyset(\varphi) = \varphi \quad \text{and} \quad \mathrm{Def}_{\Lambda\cup\{\lambda\}}(\varphi) = \mathrm{Def}_\Lambda(\varphi[Q_\lambda(\overline{x}) \mapsto \lambda]) \wedge \mathrm{Def}_\lambda(\varphi),$$

where λ is maximal in $\Lambda \cup \{\lambda\}$ with respect to the prefix ordering on positions. A *definitional form* of φ is $\mathrm{Def}_\Lambda(\varphi)$, where Λ is a subset of all positions of subformulae of φ (usually, non-atomic or non-literal subformulae).

THEOREM 4 (e.g. [29, 161]). *Let φ be a first-order formula. Then*

1. *φ is satisfiable iff $\mathrm{Def}_\Lambda(\varphi)$ is satisfiable, for any $\Lambda \subseteq \Lambda(\varphi)$, and*
2. *$\mathrm{Def}_\Lambda(\varphi)$ can be computed in polynomial time (or linear time if new symbols are introduced for all formulae occurring in Λ).*

By $\Lambda_m(\varphi)$ we denote the set of positions in $\pi_r(\varphi, x)$ corresponding to non-atomic subexpressions of the modal formula φ.

Structural transformation allows us to keep the structure of the clauses we have to deal with very simple. This in turn simplifies the characterisation of classes of clause sets that can be derived from some initial clause set using $\boldsymbol{R}^{\succ}_S$. For example, assume that, in the relational translation of the modal formula $\varphi_3 = [2]\langle 1\rangle p$, we apply structural transformation to all positions that correspond to non-atomic subexpressions of the original modal formula φ_3. The result is the set of formulae on the left of Figure 3, while the clausal form is given on the right. In general, the formulae we obtain in this way from the relational translation of modal formulae (as well as the corresponding sets of clauses) belong to quite a number of decidable fragments of first-order logic, for example, the two-variable fragment, the guarded fragment [3], Maslov's class K [135], and fluted logic [165, 166]. Resolution decision procedures have been developed for the guarded

$Q_{[2]\langle 1\rangle p}(x)$	$Q_{[2]\langle 1\rangle p}(a)^*$
$\wedge\forall x\,(Q_{[2]\langle 1\rangle p}(x) \to \forall y\,(R_1(x,y) \to Q_{\langle 1\rangle p}(y)))$	$\neg Q_{[2]\langle 1\rangle p}(x) \vee \neg R_2(x,y)^* \vee Q_{\langle 1\rangle p}(y)$
$\wedge\forall x\,(Q_{\langle 1\rangle p}(x) \to \exists y\,(R_1(x,y) \wedge P_p(y)))$	$\neg Q_{\langle 1\rangle p}(x) \vee R_1(x, f(x))^*$
	$\neg Q_{\langle 1\rangle p}(x) \vee P_p(f(x))^*$

Figure 3. The structural transformation and the clausal form of $[2]\langle 1\rangle p$

fragment [48, 76], for Maslov's class K [111, 117], for fluted logic [179] and various other classes related to modal logics, see e.g. [65, 82, 83, 111]. Here we use the results for the clausal class $\mathbf{DL}^*$ defined in [49]. $\mathbf{DL}^*$ is a variation of the class of DL-clauses, that was introduced in [119] for the purpose of deciding expressive description logics.

In order to simplify the definition of $\mathbf{DL}^*$, all clauses are assumed to be maximally split. The components in the variable partition of a clause are called *variable-disjoint* or *split components*, that is, split components do not share variables. If $C_1, \ldots, C_n$ are the split components of C, then we say C can be *decomposed* into $C_1, \ldots, C_n$. A clause which cannot be split further is called a *maximally split clause* or an *indecomposable clause*. Now, a maximally split clause C is a $\mathbf{DL}^*$*-clause* iff the following conditions are satisfied: (i) all literals are unary, or binary; (ii) there is no nesting of function symbols; (iii) every functional term in C contains all the variables of C (this condition implies that, if C contains a functional ground term, then C is ground); (iv) every binary literal (even if it has no functional terms) contains all the variables of C. It can be shown that all clauses in structural form obtained from $\mathrm{Def}_\Lambda(\pi_r(\varphi, x))$ for a modal formula φ belong to $\mathbf{DL}^*$ [49].

In order to decide the class $\mathbf{DL}^*$, we use the following ordering. First we define an order $>_d$ on terms: $s >_d t$ if s is deeper than t, and every variable that occurs in t, occurs deeper in s. Then we define $P(s_1, \ldots, s_n) \succ Q(t_1, \ldots, t_m)$ as $\{s_1, \ldots, s_n\} >_d^{\mathrm{mul}} \{t_1, \ldots, t_m\}$. Here $>_d^{\mathrm{mul}}$ is the multiset extension of $>_d$ [16]. So we have $P(f(x)) \succ P(a), P(x)$ and $P(x, y) \succ Q(x)$, but not $P(f(x)) \succ P(f(a))$. The selection function S is empty. We denote this particular instance of the resolution calculus $\mathsf{R}_S^{\succ}$ by $\mathsf{R}^{\mathsf{ord}}$.

In the example in Figure 3, the maximal literals (with respect to $\succ$) are marked with *. These are the literals that we may apply resolution or factoring to.

In order to prove that the procedure $\mathsf{R}^{\mathsf{ord}}$ is indeed a decision procedure we have to show that it is complete and terminating. Completeness follows immediately from the completeness of $\mathsf{R}_S^{\succ}$. Termination follows from the fact, that over a finite signature, there are only finitely many maximally split $\mathbf{DL}^*$-clauses (module variable renaming), and the fact that, from $\mathbf{DL}^*$-clauses, $\mathsf{R}^{\mathsf{ord}}$ produces only clauses that are again in $\mathbf{DL}^*$, or are splittable into components in $\mathbf{DL}^*$ (cf. [111, 119]).

THEOREM 5 ([49, 180]). *Let Σ be an arbitrary set of axiom schemas such that $\mathbf{K}_n\Sigma$ is complete and the clausal form of the relational frame properties F_Σ corresponding to the axiom schemas in Σ are expressible in $\mathbf{DL}^*$. Let φ be a $\mathbf{K}_n$ formula and let N be the clausal form of $F_{\Sigma,\varphi} = F_\Sigma \wedge \mathrm{Def}_{\Lambda_m(\varphi)}(\pi_r(\varphi, x))$. Then*

1. *φ is unsatisfiable in $\mathbf{K}_n\Sigma$ iff $F_{\Sigma,\varphi}$ is first-order unsatisfiable iff there is a refutation of N by $\mathsf{R}^{\mathsf{ord}}$,*
2. *N is a set of $\mathbf{DL}^*$ clauses, and*
3. *any derivation from N in $\mathsf{R}^{\mathsf{ord}}$ (up to redundancy) terminates in double exponential time; if Σ is empty, then any derivation from N in $\mathsf{R}^{\mathsf{ord}}$ (up to redundancy) terminates in exponential time, and*

Here, and in subsequent theorems, we assume that the complexity of redundancy elimination is at most exponential in the size of a clause set. The theorem remains true for $\mathsf{R}^{\mathsf{ord}}$ without the splitting rule, but condensing is key for decidability.

It is usually the case that, when studying modal decidability problems by analysing the decidability of related clausal classes, one comes to realise that stronger results are possible than

initially anticipated. In [49], extensions of $\mathbf{K}_n$ with $\mathbf{PDL}$-like relational operations have been studied. Relational operations expressible in $\mathbf{DL}^*$ include intersection, union, complementation, and converse, as are non-logical axioms.

In $\mathsf{R}^{\mathsf{ord}}$, the inferences performed are determined by a refinement based on an ordering and the empty selection function. We now consider results from [49, 119, 121, 180] for a different refinement which is based solely on a selection function and an optional ordering. More precisely, the calculus is based on maximal selection of negative literals. This means the selection function S selects exactly the set of all negative literals in any non-positive clause. When no ordering refinement $\succ$ is used, the resolution rule of $\mathsf{R}^{\succ}_{S}$ can be replaced with the following rule.

Resolution with maximal selection:

$$\frac{C_1 \vee A_1 \quad \cdots \quad C_n \vee A_n \quad \neg A_{n+1} \vee \ldots \vee \neg A_{2n} \vee D}{(C_1 \vee \ldots \vee C_n \vee D)\sigma}$$

provided that for every i, $1 \leq i \leq n$, (i) σ is the most general unifier of A_i and A_{n+i}, (ii) $C_i \vee A_i$ and D are positive clauses, (iii) no A_i occurs in C_i, and (iv) the $\neg A_{n+i}$ are selected. The *negative premise* is $\neg A_{n+1} \vee \ldots \vee \neg A_{2n} \vee D$ and the other premises are the *positive premises*. The literals A_i and A_{n+i} are the *eligible literals*.

Let $\mathsf{R}^{\mathsf{hyp}}$ be the instance of $\mathsf{R}^{\succ}_{S}$ based on maximal selection and no ordering. This means that the rules are the above resolution rule, positive unordered factoring and splitting. This refinement of resolution is also referred to as *hyperresolution plus splitting*. Condensation is not needed, but could of course be added without losing completeness and will improve the performance of the procedure. Tautology deletion is used as a simplification rule. All derivations in $\mathsf{R}^{\mathsf{hyp}}$ are generated by strategies in which no application of the resolution or factoring with identical premises and identical consequence may occur twice on the same path in any derivation. In addition, deletion rules, splitting, and the deduction rules are applied in this order, except that splitting is not applied to clauses which contain a selected literal.

All clauses occurring in the clausal form of $\mathrm{Def}_{\Lambda_m(\varphi)}(\pi_r(\varphi, x))$ for a modal formula in $\mathbf{K}_n$ have one of the forms described in Figure 4 [49, 119]. The literals marked with $^+$ are selected in the clauses by the maximal selection function S. The notation $\mathcal{P}(s)$ in the figure represents some literal with a unary predicate symbol and argument term s, and $\mathcal{R}(s,t)$ represents some literal with a binary predicate symbol and argument terms s and t (not necessarily in this order). Two occurrences of $\mathcal{P}(s)$ or $\mathcal{R}(s,t)$ need not be identical, for example, $\neg Q_\psi(x) \vee P_p(x) \vee Q_\chi(x)$ is an instance of $\neg Q_\psi(x) \vee \mathcal{P}(x) \vee \mathcal{P}(x)$, while $\neg Q_\psi(x) \vee \neg R_i(y,x) \vee Q_\chi(y)$ is an instance of $\neg Q_\psi(x) \vee \neg \mathcal{R}(x,y) \vee \mathcal{P}(y)$.

As all non-unit clauses of a typical input set contain a selected literal, all definitional clauses

$\mathcal{P}(a)$	
$\neg Q_\psi(x)^+ \vee \neg P_p(x)^+$	if $\psi = \neg p$
$\neg Q_\psi(x)^+ \vee \mathcal{P}(x)$ $[\vee \mathcal{P}(x)]$	if $\psi = \phi_1 \wedge \phi_2$ $[\psi = \phi_1 \vee \phi_2]$
$\neg Q_\psi(x)^+ \vee \neg \mathcal{R}(x,y)^+$ $[\vee \mathcal{P}(y)]$	if $\psi = [i]\bot$ $[\psi = [i]\phi]$
$\neg Q_\psi(x)^+$ $[\vee \mathcal{P}(f(x))]$ $\neg Q_\psi(x)^+ \vee \mathcal{R}(x, f(x))$	if $\psi = \langle i \rangle \top$ $[\psi = \langle i \rangle \phi]$

Figure 4. Schematic clausal forms for $\mathbf{K}_n$

can only be used as negative premises of resolution steps. To begin with, there is only one candidate for a positive premise, namely, the ground unit clause $Q_\varphi(a)$ (which representing the input formula φ). Inferences with such ground unary unit clauses produce ground clauses consisting of positive literals only, which are split into ground unit clauses. It can be shown that maximally split (non-empty) inferred clauses have one of two forms: $\mathcal{P}(s)$, or $\mathcal{R}(s, f(s))$, where s is a ground term [119]. In general, s is a nested non-constant functional ground term, which is typically avoided in resolution decision procedures based on an ordering refinement because, in most situations, nesting causes unbounded computations. For the class of clauses under consideration, however, any derived clause is smaller than its positive parent clauses with respect to a well-founded ordering which reflects the structure of the formula.

THEOREM 6 ([119, 121]). *Let φ be a $\mathbf{K}_n$ formula and let N be the clausal form of the formula* $\mathrm{Def}_{\Lambda_m(\varphi)}(\pi_r(\varphi, x))$. *Then*

1. *φ is unsatisfiable in $\mathbf{K}_n$ iff there is a refutation of N by R^{hyp}, and*
2. *any R^{hyp} derivation from N terminates.*

THEOREM 7 ([49]). *Let φ be a $\mathbf{K}_n$ formula. The space complexity for testing the satisfiability of a modal formula φ with R^{hyp} is bounded by $O(nd^m)$, where n is the number of symbols in φ, d is the number of different diamond subformulae in φ, and m is the modal depth of φ.*[2]

Formulae in $\mathbf{K}_n$ translate by the relational translation into the guarded fragment, in particular, into the two-variable guarded fragment $\mathbf{GF}^2$. It is not difficult to see that formulae in $\mathbf{K}_n$ are in fact translated into the subfragment $\mathbf{GF}^-$, introduced in [134]. Under the assumption that either (i) there is a bound on the arity of predicate symbols in $\mathbf{GF}^-$ formulae, or (ii) that each subformula of a $\mathbf{GF}^-$ formula has a bounded number of free variables, the satisfiability problem of $\mathbf{GF}^-$ is PSpace-complete, the same as for the satisfiability problem of $\mathbf{K}_n$. Obviously, there is a bound of two on the arity on predicate symbols occurring in the relational translation of modal formulae in $\mathbf{K}_n$. From these observations a well-known result follows.

THEOREM 8. *The computational complexity of the satisfiability problem of $\mathbf{K}_n$ is PSpace-complete.*

In [81] it is shown that R^{hyp} can be implemented as a modification of the main procedure of a standard (saturation based) first-order theorem prover with splitting (e.g. (M)SPASS [120, 174, 192, 194]) to provide a space optimal decision procedure for $\mathbf{GF}^-$. A direct consequence is the following.

THEOREM 9 ([81, 180]). *R^{hyp} can be turned into a polynomial space resolution decision procedure for $\mathbf{K}_n$.*

A more detailed description of how this can be done is given in Section 3.4.

Another interesting aspect of R^{hyp} is that it can polynomially simulate tableau algorithms [118, 119, 121]. In general, a proof system $\mathcal{A}$ *polynomially simulates* (*p-simulates*) a proof system $\mathcal{B}$ iff there is a function g, computable in polynomial time, mapping proofs of any given formula φ in $\mathcal{B}$ to proofs of φ in $\mathcal{A}$ [38]. To establish a correspondence between tableau proofs and derivations in R^{hyp}, we make use of the fact that each subformula ψ of a given modal formula φ corresponds to a predicate symbol Q_ψ in $\mathrm{Def}_{\Lambda_m(\varphi)}(\pi_r(\varphi, x))$. Every node w occurring in a tableau completion tree corresponds to a term t_w occurring in a R^{hyp} derivation. A formula ψ

[2] The *modal depth* of a formula φ is the maximal nesting of modal operators $\langle i \rangle$ or $[i]$ in φ.

occurring in a set labelling a node w corresponds to a unit clause $Q_\psi(t_w)$ and any edge between nodes w and v with label i in a completion tree corresponds to a unit clause $R_i(t_w, f(t_w))$ in a $\mathsf{R}^{\mathsf{hyp}}$ derivation, where t_w is the term corresponding to node w and $f(t_w)$ is the term corresponding to node v, for some function symbol f. Given these correspondences, each application of a tableau expansion rule to a completion tree can be simulated by at most two applications of expansion rules in a $\mathsf{R}^{\mathsf{hyp}}$ derivation.

This p-simulation result extends to tableau algorithms for many extension of $\mathbf{K}_n$, for example extensions by the modal axiom schemas $\mathbf{T}$, $\mathbf{D}$, $\mathbf{B}$, $\mathbf{4}$, and $\mathbf{5}$ [121]. It also extends to other forms of tableau and sequent-style calculi.

The notion of p-simulation leaves open the possibility that an algorithm based on the proof system $\mathcal{A}$ which p-simulates a proof system $\mathcal{B}$ would have to search a much larger search space to find a proof for a given formula than an algorithm based on $\mathcal{B}$. For $\mathsf{R}^{\mathsf{hyp}}$, however, it is possible to show that the search space corresponds to that of the tableau algorithm for $\mathbf{K}_n$ presented in Section 4.2 [121]. Related simulation results of tableau procedures for description logics can be found in [118, 119], see also [65]. All these simulation results provide valuable insights into the similarities and difference between tableau methods and resolution. On the one hand, the view presented is that many tableau algorithms are essentially hyperresolution with lazy translation to first-order logic. On the other hand, because of the generality of the setting (first-order logic) it is even possible to exploit the close link with hyperresolution and use it as a basis for systematically developing new tableau procedures. Using this approach, a new tableau decision procedure was essentially 'read off' in [49] from a translation-based hyperresolution decision procedure for an expressive PDL-style modal logic.

For the modal logic $\mathbf{K}_n$, an improved version of the relational translation is presented in [9]. In the original presentation, this translation consists of two steps, first mapping a formula from one multi-modal logic into another, and then applying the relational translation to it. Our presentation merges both steps into one. We uniquely associate a unary predicate symbol $P_{p,\sigma}$ with every propositional variable p and sequence σ of modalities. Similarly, we uniquely associate a binary predicate symbol R_σ with every sequence σ of modalities. Then the *tree(-based) relational translation* π_{tr} is defined as follows.

$$\begin{aligned}
\pi_{tr}(\top, x, \sigma) &= \top & \pi_{tr}(\bot, x, \sigma) &= \bot \\
\pi_{tr}(p, x, \sigma) &= P_{p,\sigma}(x) & \pi_{tr}(\neg\varphi, x, \sigma) &= \neg\pi_{tr}(\varphi, x, \sigma) \\
\pi_{tr}(\varphi \star \psi, x, \sigma) &= \pi_{tr}(\varphi, x, \sigma) \star \pi_{tr}(\psi, x, \sigma) & \text{for } \star &\in \{\wedge, \vee, \rightarrow, \leftrightarrow\} \\
\pi_{tr}([i]\varphi, x, \sigma) &= \forall y(R_{\sigma.i}(x, y) \rightarrow \pi_{tr}(\varphi, y, \sigma.i)) \\
\pi_{tr}(\langle i\rangle\varphi, x, \sigma) &= \exists y(R_{\sigma.i}(x, y) \wedge \pi_{tr}(\varphi, y, \sigma.i))
\end{aligned}$$

The translation of a modal formula is given by $\pi_{tr}(\varphi, x, \epsilon)$. The tree relational translation can be viewed as incorporating a feature of the (optimised) functional translation into the relational translation. Whereas the relational translation uses a family of binary predicate symbols R_i, where i is a modal parameter, the tree relational translation uses a larger family of binary predicate symbols R_σ, where σ is a sequence of modal parameters representing a path from the initial world, to encode transitions between worlds. Another difference is that in the tree-based translation the σ are also encoded into the unary predicates.

THEOREM 10. *A modal formula φ is satisfiable in $\mathbf{K}_n$ iff $\pi_{tr}(\varphi, x, \epsilon)$ is first-order satisfiable.*

If we restrict ourselves to $\mathbf{K}$, that is, our logic has only one modality, then the sequence σ in the definition of π_{tr} only serves as a unary coding of the natural numbers. Thus, we can further

simplify the translation by using $\pi_{tr}(\varphi, x, 0)$ as translation of a modal formula and modifying the translation π_{tr} as follows.

$$\pi_{tr}([i]\varphi, x, \sigma) = \forall y(R_{\sigma+1}(x, y) \rightarrow \pi_{tr}(\varphi, y, \sigma{+}1))$$
$$\pi_{tr}(\langle i\rangle\varphi, x, \sigma) = \exists y(R_{\sigma+1}(x, y) \wedge \pi_{tr}(\varphi, y, \sigma{+}1))$$

All other cases in the definition of π_{tr} remain unchanged. In [8] it is shown that we can use the following ordering to ensure that derivations in $\mathsf{R}_S^{\succ}$ from the clausal form of $\pi_{tr}(\varphi, x, 0)$ of a $\mathbf{K}$ formula φ terminates: $P_\sigma(s_1, \ldots, s_n) \succ Q_\delta(t_1, \ldots, t_m)$ if either $\sigma < \delta$, or $\sigma = \delta$ and $n > m$. This result can easily be extended to $\mathbf{K}_n$ by defining the ordering $\succ$ as $P_\sigma(s_1, \ldots, s_n) \succ Q_\delta(t_1, \ldots, t_m)$ if either $\mathit{length}(\sigma) < \mathit{length}(\delta)$ or $\mathit{length}(\sigma) = \mathit{length}(\delta)$ and $n > m$. This ordering restriction can be seen to force a kind of top-down approach.

THEOREM 11. *Let φ be a modal formula in $\mathbf{K}_n$ and let N be the clausal form of $\pi_{tr}(\varphi, x, \epsilon)$. Then any derivation from N in $\mathsf{R}^{\mathsf{ord}}$ (up to redundancy) without splitting terminates.*

One of the interesting aspects of this result is that it does not require the use of structural transformation (nor does it require the use of the splitting rule, but condensing is crucial).

3.2 Global satisfiability, non-logical axioms, transitive modalities, and $\mathbf{K4}_n$

So far we have focused on *local satisfiability*, that is, the problem whether for a given modal formula φ, there exists a model $\mathfrak{M} = \langle W, R, V\rangle$ and a world $w \in W$ such that $\mathfrak{M}, w \models \varphi$. Now we turn to the problem of determining whether there is a model $\mathfrak{M}$ such that for all worlds $w \in W$ $\mathfrak{M}, w \models \varphi$, i.e. is φ globally true in some model. The modifications necessary to allows us to determine the *global satisfiability* of a modal formula in $\mathbf{K}_n$ based on the relational translation are minimal: φ is globally satisfiable in $\mathbf{K}_n$ iff $\forall x\, \pi_r(\varphi, x)$ is first-order satisfiable. Is it straightforward to see that the clausal form N of $\mathrm{Def}_{\Lambda_m(\varphi)}(\forall x\, \pi_r(\varphi, x))$ still consists only of $\mathbf{DL}^*$ clauses. Consequently, $\mathsf{R}^{\mathsf{ord}}$ can decide the satisfiability of the clause set N.

THEOREM 12. *Let Σ be an arbitrary set of axiom schemas such that $\mathbf{K}_n\Sigma$ is complete and the clausal form of the relational frame properties F_Σ corresponding to the axiom schemas in Σ is in $\mathbf{DL}^*$. Let φ be a modal formula in $\mathbf{K}_n$ and let N be the clausal form of $F_{\Sigma,\varphi} = F_\Sigma \wedge \mathrm{Def}_{\Lambda_m(\varphi)}(\forall x\, \pi_r(\varphi, x))$. Then*

1. *φ is not globally satisfiable in $\mathbf{K}_n\Sigma$ iff $F_{\Sigma,\varphi}$ is not first-order satisfiable iff there is a refutation of N by $\mathsf{R}^{\mathsf{ord}}$,*
2. *N is a set of $\mathbf{DL}^*$ clauses, and*
3. *any derivation from N in $\mathsf{R}^{\mathsf{ord}}$ (up to redundancy) terminates in double exponential time and in exponential time, if Σ is empty.*

For local and global $\mathbf{K}_n$-satisfiability w.r.t. to a background theory of non-logical axioms the same result is true. Furthermore the complexity of ordered resolution is optimal.

THEOREM 13. *Let $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ be a finite set of of modal formulae and let φ be a modal formula. Let $F_{\Gamma,\varphi}$ be the first-order formula $\exists x\, \pi_r(\varphi, x) \wedge \bigwedge_{i=1,\ldots,n} \forall x\, \pi_r(\gamma_i, x)$ and let N be the clausal form of $\mathrm{Def}_{\Lambda_{\Gamma,\varphi}}(F_{\Gamma,\varphi})$, where $\Lambda_{\Gamma,\varphi}$ contains all non-atomic positions of $F_{\Gamma,\varphi}$. Then*

1. *φ is unsatisfiable in $\mathbf{K}_n$ w.r.t. Γ iff $F_{\Gamma,\varphi}$ is first-order unsatisfiable iff there is a refutation of N by $\mathsf{R}^{\mathsf{ord}}$,*

2. *N is a set of $\mathbf{DL}^*$ clauses, and*

3. *any derivation from N in $\mathsf{R}^{\mathsf{ord}}$ (up to redundancy) terminates in exponential time.*

In contrast to $\mathsf{R}^{\mathsf{ord}}$, derivations in $\mathsf{R}^{\mathsf{hyp}}$ from the clausal form of $\mathrm{Def}_{\Lambda_{\Gamma,\varphi}}(F_{\Gamma,\varphi})$, as defined in Theorem 13, are not guaranteed to terminate. Tableau algorithms face the same problem, and the solution typically used is a technique called *blocking*. See Sections 4.3 and 4.4 for details. This technique can be transferred to the context of first-order clausal logic and $\mathsf{R}^{\mathsf{hyp}}$ derivations as described in [118]. It involves the addition of a *blocking rule* which at certain points during a derivation adds equations $t_1 \approx t_2$ between ground terms t_1 and t_2 to the clause set, rendering inferences on literals involving the greater of the two terms redundant. One of the interesting properties of this approach is that completeness follows immediately from the general completeness result for $\mathsf{R}^{\mathsf{hyp}}$ [18], only soundness needs to be established. Another way of combining blocking with $\mathsf{R}^{\mathsf{hyp}}$ is presented in [27]. In addition, optimisations techniques like *lazy unfolding* and *absorption*, which will be discussed in detail in Section 4.4, are in-built and therefore free in $\mathsf{R}^{\mathsf{hyp}}$.

However, for $\mathbf{K}_n$ extended with axiom schemas sometimes quite different approaches are required. For example the formula $\forall xyz\,((R_i(x,y) \wedge R_i(y,z)) \rightarrow R_i(x,z))$ stating the transitivity of R_i is not a formula in any of the relevant decidable first-order fragments. The corresponding clause does not belong to $\mathbf{DL}^*$ either. To handle transitive modal logics one possibility is to use the *ordered chaining calculus* introduced in [15] for binary relations satisfying the general schema $R_i \circ R_j \subseteq R_k$. A decision procedure for a first-order fragment covering the modal logics $\mathbf{K4}$, $\mathbf{KD4}$, and $\mathbf{KT4}$, and their multi-modal variants, which is based on ordered chaining, is presented in [77]. Recent work in [124] presents an extension of $\mathsf{R}^{\succ}_{S}$ which can decide the guarded fragment with transitive guards. This provides a decision procedure for all modal logics translatable into this fragment.

In the following we present another approach, the *axiomatic translation* approach [181], which allows a variety of modal logics with transitive modalities to be embedded in $\mathbf{DL}^*$. Consequently this allows the use of $\mathsf{R}^{\mathsf{ord}}$ to decide these logics. This method is not restricted to transitive modal logics and applies to a large class of modal logics.

Remember that structural transformation introduces for each modal subformula $[i]\psi$ of a modal formula φ a predicate symbol $Q_{[i]\varphi}$ in $\pi_r(\varphi, x)$. The general principle of the axiomatic translation approach for $\mathbf{K4}_n$ is the following. For every transitive modality $[i]$ and every subformula $[i]\psi$ of the formula φ, add the first-order formula

$$(A_{\mathbf{4}}) \qquad \forall xy\,((Q_{[i]\psi}(x) \wedge R_i(x,y)) \rightarrow Q_{[i]\psi}(y)).$$

to the translation. The main technical question with the axiomatic translation principle is to know how many instances of such a 'schema formula' need to be added to the translation. In the Hilbert axiomatisation, axioms such as $\mathbf{4}$ are valid for all substitution instances. Since we do not have access to a substitution rule, we need to make sure from the outset that enough instances of the schema formulae are present in the translation of φ. (Of course, this does not preclude a lazy implementation which delays the translation of subformulae and the inclusion of instances of schema formulae until absolutely necessary.) The clausal form of $A_{\mathbf{4}}$ is $\neg Q_{[i]\psi}(x) \vee \neg R_i(x,y) \vee Q_{[i]\psi}(y)$, which is a $\mathbf{DL}^*$ clause (and a guarded clause).

THEOREM 14. *Let φ be a modal formula and Ξ the set of all subformulae of the form $[i]\psi$ of φ. Let $F_{\mathbf{4}}$ be the first-order formula $\bigwedge_{[i]\psi\in\Xi} \forall xy\,((Q_{[i]\psi}(x) \wedge R_i(x,y)) \rightarrow Q_{[i]\psi}(y))$. Let N be the clausal form of $F_{\mathbf{4},\varphi} = F_{\mathbf{4}} \wedge \mathrm{Def}_{\Lambda_m(\varphi)}(\pi_r(\varphi, x))$. Then*

1. *φ is unsatisfiable in $\mathbf{K4}_n$ iff $F_{\mathbf{4},\varphi}$ is first-order unsatisfiable iff there is a refutation of N by R^{ord},*
2. *N is a set of $\mathbf{DL}^*$ clauses, and*
3. *any derivation from N in R^{ord} (up to redundancy) terminates in exponential time.*

The same result is true for global satisfiability in $\mathbf{K4}_n$ and also reasoning with respect to non-logical axioms. Theorem 14 reduces reasoning in $\mathbf{K4}_n$ to reasoning in $\mathbf{K}_n$ with background theories. Consequently, $\mathsf{R}^{\mathsf{hyp}}$ combined with a blocking rule provides an alternative decision procedure for $\mathbf{K4}_n$.

3.3 Converse modalities and the modal logics $\mathbf{KB}_n$ and $\mathbf{KB4}_n$

Extending the results of Section 3.1 to modal logics with *converse modalities* or to the modal logics $\mathbf{KB}_n$ and $\mathbf{KB4}_n$ is straightforward. For converse modalities we have to extend our definition of the relational translation π_r as follows:

$$\pi_r([i^\smile]\varphi, x) = \forall y\,(R_i(y,x) \to \pi_r(\varphi, y)) \qquad \pi_r(\langle i^\smile\rangle\varphi, x) = \exists y\,(R_i(y,x) \wedge \pi_r(\varphi, y))$$

Then, Theorem 13 extends to the following.

THEOREM 15. *Let $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ be a finite set of of $\mathbf{K}_n^\smile$ formulae and let φ be a $\mathbf{K}_n^\smile$ formula. Let $F_{\Gamma,\varphi}$ be the first-order formula $\exists x\,\pi_r(\varphi, x) \wedge \bigwedge_{i=1,\ldots,n} \forall x\,\pi_r(\gamma_i, x))$. Let N be the clausal form of $\mathrm{Def}_{\Lambda_{\Gamma,\varphi}}(F_{\Gamma,\varphi})$ where $\Lambda_{\Gamma,\varphi}$ contain all non-atomic positions of $F_{\Gamma,\varphi}$. Then*

1. *φ is unsatisfiable in $\mathbf{K}_n^\smile$ w.r.t. Γ iff $F_{\Gamma,\varphi}$ is first-order unsatisfiable iff there is a refutation of N by R^{ord},*
2. *N is a set of $\mathbf{DL}^*$ clauses, and*
3. *any derivation from N in R^{ord} (up to redundancy) terminates in exponential time.*

In the case of the modal logic $\mathbf{KB}_n$ we extend the relational translation (or the axiomatic translation) by adding the relational frame property $R_{\mathbf{B}}$ corresponding to $\mathbf{B}$, namely $\forall xy\,(R_i(x,y) \to R_i(y,x))$, to the translation of φ. Finally, in the case of $\mathbf{KB4}_n$ we restrict ourselves to the axiomatic translation and again add the relational frame property $R_{\mathbf{B}}$ to the translation of φ.

In all these cases, the clausal form N of the translated modal formulae as well as that of $R_{\mathbf{B}}$ consists only of $\mathbf{DL}^*$ clauses. Consequently, $\mathsf{R}^{\mathsf{ord}}$ provides us with a decision procedure for the satisfiability of N.

THEOREM 16. *Let φ be a $\mathbf{KB}_n$ formula and let N be the clausal form of the first-order formula $F_{\mathbf{B},\varphi} = \forall xy\,(R_i(x,y) \to R_i(y,x)) \wedge \mathrm{Def}_{\Lambda_m(\varphi)}(\pi_r(\varphi, x))$. Then*

1. *φ is unsatisfiable in $\mathbf{KB}_n$ iff $F_{\mathbf{B},\varphi}$ is first-order unsatisfiable iff there is a refutation of N by R^{ord},*
2. *N is a set of $\mathbf{DL}^*$ clauses, and*
3. *any derivation from N in R^{ord} (up to redundancy) terminates in exponential time.*

The result extends easily to global satisfiability and non-logical axioms. So, the axiomatic translation for $\mathbf{KB}$ is another reduction into $\mathbf{DL}^*$, but also $\mathbf{GF}^2$, and $\mathsf{R}^{\mathsf{ord}}$ is an exponential time decision procedure [181]. Besides $\mathsf{R}^{\mathsf{ord}}$ we can also use $\mathsf{R}^{\mathsf{hyp}}$ to decide the satisfiability of N (this is a consequence of the main results in [82, 83]).

THEOREM 17. *Let φ be a $\mathbf{KB4}_n$ formula and let Ξ be the set of all subformulae of the form $[i]\psi$ of φ. Let $F_{\mathbf{B4}}$ be the first-order formula*

$$\forall xy\,(R_i(x,y) \rightarrow R_i(y,x)) \wedge \bigwedge\nolimits_{[i]\psi\in\Xi} \forall xy\,((Q_{[i]\psi}(x) \wedge R_i(x,y)) \rightarrow Q_{[i]\psi}(y)).$$

Let N be the clausal form of $F_{\mathbf{B4},\varphi} = F_{\mathbf{B4}} \wedge \mathrm{Def}_{\Lambda_m(\varphi)}(\pi_r(\varphi, x))$. Then

1. *φ is unsatisfiable in $\mathbf{KB4}_n$ iff $F_{\mathbf{B4},\varphi}$ is first-order unsatisfiable iff there is a refutation of N by $\mathsf{R}^{\mathsf{ord}}$,*
2. *N is a set of $\mathbf{DL}^*$ clauses, and*
3. *any derivation from N in $\mathsf{R}^{\mathsf{ord}}$ (up to redundancy) terminates in exponential time.*

As described in Section 3.1, in the case of the optimised functional translation, we add so-called *functional frame properties* in the form of (conditional) equations between path terms to accommodate additional axiom schemas like $\mathbf{4}$ and $\mathbf{5}$. Alternatively, one can replace syntactic unification in the inference rules of R with *theory unification* [175, 176]. The resulting calculus is called *theory resolution*. So far, the only decision procedures for modal logics like $\mathbf{K4}_n$ or $\mathbf{KB}_n$ based on theory resolution make use of a term depth bound, that is, any derived clause involving terms of depth greater than a pre-computed bound dependent on the modal formula whose satisfiability is tested will be removed [175, 178].

This section is an incomplete discussion of the different uses of first-order resolution. Due to space restrictions we have only been able to present a few of the translations that are available and have omitted a lot of details. Other translation methods are surveyed in [64, 148]. See also the surveys [65, 49, 122, 180, 182].

3.4 Implementation and optimisation

In this section, we give a brief overview of the implementation of the resolution calculus presented in Section 3.1 and discuss some of the issues involved in using such an implementation for theorem proving in modal logic. For further details on the implementation of first-order theorem provers see e.g. [193, 169, 183].

The procedure ResolutionProver presented in Figure 5 is the main procedure implementing the calculus $\mathsf{R}^{\succ}_{S}$. The input is a set N of clauses. The output on termination is a proof of unsatisfiability or a saturated clause set. The procedure operates on two sets of clauses, $\mathcal{US}$ and $\mathcal{WO}$ (the set of *usable clauses* and the set of *worked-off clauses*). The set $\mathcal{WO}$ contains all the clauses that have already been used as premises in inference steps (or can never be used as premises) and the set $\mathcal{US}$ contains all the clauses that still need to be considered as premises. In our particular case, the input set N is the clausal form of the translation of some modal formula.

The procedure proceeds as follows. First, the input set N is simplified by the function ired, that is, all tautologies and strictly subsumed clauses are deleted from N (this is achieved by the two argument ired function). The set N is then divided into two sets: the *usable clauses* $\mathcal{US}$ and the *worked-off clauses* $\mathcal{WO}$. The set $\mathcal{US}$ contains all the clauses which are candidates for

```
Procedure ResolutionProver(N)
local WO, US, NEW, Given;
begin
  WO := ∅;
  US := ired(N,N);
  Stack := emptystack();
  while (US ≠ ∅) and (⊥ ∉ US or not stackempty(Stack))
  do
    if (⊥ ∈ US) then
      (Stack, US, WO) := backtrack(Stack, US,WO);
    else
      begin
        (Given, US) := choose(US);
        if (splittable (Given)) then
          begin
            NEW := firstsplitcase(Given);
            Stack := push(Stack, secondsplitcase(Given))
          end
        else
          begin
            WO := WO ∪ {Given};
            NEW := inf(Given, WO);
          end
      end
      (NEW, WO, US) := ired(NEW, WO, US);
  return(US)
end
```

Figure 5. Standard inference loop in a saturation theorem prover

inferences and the set $\mathcal{WO}$ contains all the clauses that have already been selected for inferences. Initially, the set $\mathcal{WO}$ is the empty set, while $\mathcal{US}$ contains all clauses of N remaining after application of ired. Next the procedure enters the main inference loop in which it remains while the set $\mathcal{US}$ is not empty and the empty clause $\bot$ has not been derived or there are still alternative branches of the derivation tree that need to be considered. Within the main loop it is first checked whether the set $\mathcal{US}$ contains the empty clause. If so, the current branch of the derivation is a closed branch and backtracking takes the computation to a different branch of the derivation. Otherwise the function choose selects a clause from $\mathcal{US}$. This clause is called the *given clause*. If the splitting rule can be applied to the given clause, one of its two split components is taken to be the newly derived clause, which is stored in $\mathcal{NEW}$, and the other split component is pushed onto a stack. Basically, this creates a new branch in the derivation tree that is explored later, if it turns out that the current branch can be closed. This corresponds to a depth-first construction of the derivation tree. If the splitting rule cannot be applied, we add the given clause to the set $\mathcal{WO}$ and compute all conclusions of inferences by resolution and factoring between the given clause and clauses in $\mathcal{WO}$ using the function inf. After removing redundant clauses from the sets $\mathcal{US}$, $\mathcal{WO}$, as well as the newly derived clauses (this is achieved by the three argument ired function), the remaining new clauses are added to the set $\mathcal{US}$, and a new iteration of the main

loop is entered.

Important points to note about the ResolutionProver procedure are the following. First, in the main inference loop, the function inf computes all conclusions derivable from the given clause and clauses in $\mathcal{WO}$. For example, suppose we use the $\mathsf{R}^{\mathsf{hyp}}$ instance of $\mathsf{R}^{\succ}_{S}$. Let the set $\mathcal{US}$ contain the clauses $Q_{\langle 1\rangle\varphi}(t)$, $D_0 = \neg Q_{\langle 1\rangle\varphi}(x) \vee R_1(x, f(x))$, and $D_1 = \neg Q_{\langle 1\rangle\varphi}(x) \vee Q_{\varphi}(f(x))$, as well as n clauses of the form $C_{i+1} = Q_{[1]\psi_i}(t)$ and $D_{i+1} = \neg Q_{[1]\psi_i}(x) \vee \neg R_i(x, y) \vee Q_{\psi_i}(y)$, for $1 \leq i \leq n$. If we first choose each D_i, $0 \leq i \leq n{+}1$ they are simply moved to $\mathcal{WO}$, without any new clause being inferred from them. The same is true if we continue by choosing each C_{i+1} in turn. Finally, when we choose $Q_{\langle 1\rangle\varphi}(t)$, the clauses $R_i(t, f(t))$ and $Q_{\varphi}(f(t))$ is computed by inf and moved to $\mathcal{US}$. When $R_1(t, f(t))$ becomes the given clause, inf computes in one step the clauses $Q_{\psi_i}(f(t))$, for $1 \leq i \leq n$. This corresponds to the application of the tableau inference rule

$$\frac{w : \langle i\rangle\varphi \qquad w : [i]\psi_1 \quad \cdots \quad w : [i]\psi_n}{v : \varphi \qquad v : \psi_1 \quad \cdots \quad v : \psi_n}$$

where v is an i-successor of w. However, if we choose clauses starting with D_0, followed by D_1, and then $Q_{\langle 1\rangle\varphi}(t)$, inf infers $R_1(t, f(t))$ and $Q_{\varphi}(f(t))$, corresponding to an application of the $\Diamond$-rule in the tableau algorithm defined in Figure 8. If we proceed by choosing $R_1(t, f(t))$, then each C_{i+1} directly followed by D_{i+1}, inf infers $Q_{\psi_i}(f(t))$, corresponding to a series of applications of the $\Box$-rule in Figure 8. This shows that the way in which clauses are selected by choose gives us added flexibility in how the search for a refutation is directed.

Second, the ordering $\succ$ and the selection function S only influence the function inf without changing what has just been said. Concerning the selection function S the user is able to select among a fixed set of pre-defined selection functions. The selection function which selects every negative literal in any clause is usually included in that set. Concerning the ordering $\succ$, state-of-the-art first-order theorem provers standardly contain implementations of recursive path orderings, lexicographic path orderings or Knuth Bendix orderings, which are parameterised by an ordering on the signature of the input clause set N, which the user can specify. Refinements of the particular ordering $\succ$ defined in Section 3.1 can be obtained by either recursive path orderings or Knuth Bendix orderings (definitions of orderings and ordering extensions can be found in [52]).

Third, the remaining functions in ResolutionProver are firstsplitcase and secondslitcase which basically determine the order in which branches of a derivation tree are investigated. Again, it is possible to exercise control on this order by using some heuristic.

Fourth, the implementation of backtrack has significant influence on the performance of the prover. On the stack we only store the second split component that may need to be considered at a later point, but not the current state of $\mathcal{WO}$ and $\mathcal{US}$. The information required to return $\mathcal{WO}$ and $\mathcal{US}$ to the correct state on backtracking is stored in each clause, allowing us to remove clauses which are no longer derivable and restoring clauses which are no longer redundant after backtracking. When we derive a contradiction it is not necessary to backtrack to the state associated with the split component currently on top of the stack. Instead more intelligent forms of backtracking are possible. For example, the theorem prover SPASS [194] implements *branch condensing*. Here, on backtracking, all first components not used to derive a contradiction are removed from the set $\mathcal{US}$ as well as all the corresponding second split components on the stack. The prover then backtracks to the second split component which is now on top of the stack, removing clauses which are no longer derivable and restoring clauses which are no longer redundant. For further details see [193]. This form of intelligent backtracking is closely related but not identical to *conflict-directed backjumping* [78, 164]. See also Section 4.2.

An alternative to explicit splitting is *splitting through new propositional variables* [168] implemented in the theorem prover VAMPIRE [169] or the generalisation called *separation* in [179]. In the splitting through propositional variables approach, a clause $C \vee D$ with variable-disjoint components C and D is replaced with two clauses $C \vee p$ and $D \vee \neg p$, where p is a new propositional variable called a *split propositional variable*. The ordering $\succ$ and the selection function S are extended to ensure that p is minimal in $D \vee \neg p$ and $\neg p$ is selected in D. This makes it impossible for the clause $C \vee D$ to be derived from the two new clauses and also blocks $D \vee \neg p$ for inferences until we derive a clause in which p is maximal. The derivation of a contradiction from the split component C in explicit splitting then corresponds to the derivation of a clause $E \vee p$ where E consists solely of split propositional variables. If p is maximal in $E \vee p$ we can derive $E \vee D$ which corresponds to backtracking to the branch of the derivation in which D is true. Note that this again is a form of intelligent backtracking since $E \vee p$ is a representation of all the split components involved in deriving a 'contradiction'. Thus, in backtracking we ignore all other splits not represented by a split propositional variable in $E \vee p$. Unlike branch condensing and (conflict-driven) backjumping, however, those splits are still present. A disadvantage of splitting through new propositional symbols is that subsumption and reductions such as unit propagation are not as effective as for explicit splitting. De Nivelle [47] has suggested modifications of the standard inference and redundancy elimination rules which take account of split propositional variables.

Both explicit splitting and splitting through new propositional variables split a clause $C \vee D$ into split components C and D. The two branches of the derivation do not necessarily investigate disjoint sets of Kripke/first-order models. For variants of splitting we have the option to add the negation of C, $\neg C$, to the branch on which D is true. However, in contrast to propositional logic, the benefit is less obvious. For example, assume that C is $Q_{[i]p}(a)$ and that the clause set to which we add $\neg C = \neg Q_{[i]p}(a)$ contains already the unit clause $Q_{[i](p \wedge q)}(a)$. We can propagate the unit clause $\neg C$ to all clauses in the clause set which removes all occurrences of $Q_{[i]p}(a)$ from those clauses. However, the contradiction between $\neg Q_{[i]p}(a)$ and $Q_{[i](p \wedge q)}(a)$ is not detected. This is true even if the clause set contains the definitional clauses $Q_{[i]p}(x) \vee R(x, f(x))$, $Q_{[i]p}(x) \vee \neg P_p(f(x))$, which we can use to derive $R(a, f(a))$ and $\neg P_p(f(a))$. Only when $R(a, f(a))$ and $Q_{[i](p \wedge q)}(a)$ together with the definition clauses $\neg Q_{[i](p \wedge q)}(x) \vee \neg R(x, y) \vee P_p(y)$ (and $\neg Q_{[i](p \wedge q)}(x) \vee \neg R(x, y) \vee P_q(y)$) are used to derive $P_p(f(a))$, is a contradiction detected. Note also that the clause $R(a, f(a))$ might trigger the derivation of a large number of additional clauses which would not be derived in the absence of $\neg Q_{[i]p}(a)$ or its definitional clauses. In general, the computational effort expended to this point might be great without a guarantee that there is a payoff. Termination is however not compromised.

Used as a procedure to test the satisfiability of a $\mathbf{K}_n$ formula φ with any refinement of $R_S^{\succ}$ and any of the translations presented in this section, ResolutionProver requires exponential space in the size of φ. In [81] we have shown how ResolutionProver can be turned into a space optimal decision procedure for the class $\mathbf{GF}^-$. This modified procedure provides also a polynomial space decision procedure for the relational translation of $\mathbf{K}_n$ and $\mathbf{KB}_n$ formulae. If we focus on $\mathbf{K}_n$, then a simple modification of ResolutionProver as described in Figure 6 is sufficient. The procedure uses an additional local variable t which stores the term we currently focus on. Initially it is the only ground term in N, and is returned by groundTerm. The procedure choose selects the given clauses in a particular order. It starts by choosing non-ground clauses. This transfers all definitional clauses from $\mathcal{US}$ to $\mathcal{WO}$ without any inference steps being performed. Then it selects ground clauses in an order which ensures that the derivation corresponds to a depth-first exploration of the completion tree in a tableau derivation. Finally, ired is modified so

Procedure ResolutionProver(N)
local $\mathcal{WO}, \mathcal{US}, \mathcal{NEW}$, Given, t;
begin
  t := groundTerm(N);
  ...
  (Given, $\mathcal{US}$, t) := choose($\mathcal{US}$, t);
  ...
  ($\mathcal{NEW}$, $\mathcal{WO}$, $\mathcal{US}$) :=
    ired($\mathcal{NEW}$, $\mathcal{WO}$, $\mathcal{US}$, t);
  ...
end

Procedure choose($\mathcal{US}$, t)
begin
  if ($C \in \mathcal{US}$ where C is non-ground) **then**
    return($C, \mathcal{US} - \{C\}, t$)
  else if ($Q(t) \vee C \in \mathcal{US}$) **then**
    return($Q(t) \vee C, \mathcal{US} - \{Q(t) \vee C\}, t$)
  else if ($R_i(t, s) \in \mathcal{US}$) **then**
    return($R_i(t, s), \mathcal{US} - \{R_i(t, s)\}, s$)
  else if ($R_i(u, v) \in \mathcal{US}$ with v having greatest depth in $\mathcal{US}$) **then**
    return($R_i(u, v), \mathcal{US} - \{R_i(u, v)\}, v$)
end

Figure 6. Modified procedures for a polynomial space decision procedure for $\mathbf{K}_n$

that it removes from $\mathcal{WO}$ all clauses containing argument terms which are not subterms of the term t. This modification ensures that the information on terms which have been fully explored and does not contribute to a refutation is removed, bringing the space requirements down to polynomial space.

3.5 Other extensions (counting, nominals)

The modal logics $\mathbf{K}_n^c$ and $\mathbf{K}_n^{\smile,c}$ with graded modalities and the modal logic $\mathbf{K}_n^o$ with nominals can be translated to first-order logic using a number of different embeddings. The simplest one is an extension of the relational translation as follows (the symbol o_i denotes a nominal).

$$\pi_r(\langle i \rangle_m \varphi, x) = \exists y_1 \ldots y_m \, (R_i(x, y_1) \wedge \ldots \wedge R_i(x, y_m) \wedge y_1 \not\approx y_2 \wedge \ldots \wedge y_{m-1} \not\approx y_m)$$
$$\pi_r([i]_m \varphi, x) = \forall y_1 \ldots y_{m+1} \, ((R_i(x, y_1) \wedge \ldots \wedge R_i(x, y_{m+1})) \rightarrow (y_1 \approx y_2 \vee \ldots \vee y_n \approx y_{m+1}))$$
$$\pi_r(o_i, x) = (x \approx o_i)$$

The superposition calculus [14] and the basic superposition calculus [17] are extensions of $\mathsf{R}_S^{\succ}$ with rules for equality reasoning. In [113, 112] it is shown that the basic superposition calculus can be used to decide the satisfiability of knowledge bases in the $\mathcal{SHIQ}$ description logic. It follows that it can also be used to decide the satisfiability of formulae in $\mathbf{K}_n^c$ and $\mathbf{K}_n^{\smile,c}$.

An extension of the optimised functional translation to $\mathbf{K}_n^c$ is presented and shown to be sound and complete in [150].

4 TABLEAU-BASED ALGORITHMS

In this section, we describe tableau-based decision procedures for modal logics and discuss their complexity and implementation issues. First, we discuss various choices for presenting tableau algorithms in general, and then present the basic tableau algorithm for $\mathbf{K}_n$ together with a detailed discussion of implementation and optimisation issues. Next, we modify this algorithm to handle $\mathbf{K4}_n$, background theories, converse modalities, and their combinations, and point out relevant modifications concerning the implementation and optimisation.

Intuitively, a tableau algorithm tries to construct, for an input formula φ, a model of φ; i.e., to decide the validity of a formula ψ, the tableau algorithm is started with $\neg\psi$. Depending on the modal logic, it is often convenient to consider an abstraction of models rather than models, namely a so-called *tableau*.

4.1 Tableau algorithms in general

We start with a description of a tableau algorithm for multi modal $\mathbf{K}_n$. Roughly speaking, this algorithm takes the input formula φ and deduces constraints on the model it is going to build by breaking it down into its sub-formulae. We will first describe different styles in which this attempt at a model construction has been described and relate them to each other.

The "breaking down" is realized through *tableau expansion rules*; quite often, we find one such expansion rule per logical constructor. For example, if we know that $\psi_1 \wedge \psi_2$ should be true in world w of the model we are constructing, then we break the conjunction down and explicitly add the constraints that each ψ_i has to be true in w. Next, we discuss the rules that handle box- and diamond formulae. Intuitively, if we know that $\langle i\rangle\psi$ should be true in world w, then we "generate" a witness world w', which is i-accessible from w and in which ψ is true. This can be formalised in different ways:

- for certain modal logics such as $\mathbf{K}_n$, one can first handle all formulae talking about a single world, then collect *all* constraints concerning another world and process these, and so on [129]. This approach is sometimes called "distructive" [92] (see also Chapter 2) because we can forget the constraints concerning an "old" world once we procede to the next one.

- *labelled* tableaux are closely related to propositional tableaux: they are sets of *labelled* formulae that partially describe a model: each formula is labelled with the world it should be true in (see Chapter 2 of this handbook). For example, the case where $\langle 1\rangle\psi$ is true in world w would translate to finding the labelled formula $w : \langle 1\rangle\psi$ in our tableau, and the $\Diamond$-rule adds labelled formulae $w' : \psi$ and $w1w'$, where the latter encodes that w' is 1-accessible from w. This information is required if we find, additionally, a formula of the form $w : [1]\psi'$: in this case, the $\Box$-rule adds $w' : \psi'$.

  Alternatively, one can store the information that w' is i-accessible from w in the labels by using appropriate sequences instead of atomic "names" w, w'. We start with the empty sequence labelling the input concept, and then append these labels, e.g., as follows: if a world is generated for a labelled formula $s : \langle i\rangle\psi$, we name this world $s(i, \psi)$ and simply introduce the new labelled formula $s(i, \psi) : \psi$. Please note that $si : \psi$ does not suffice because $\langle 1\rangle\psi \wedge \langle 1\rangle\neg\psi$ is satisfiable, but only in a world that has two *distinguished* 1-acccessible worlds.

- other tableau algorithms explicitly store the relational structure of the model (or tableau) they are building. More precisely, they work on labelled graphs (often trees) where nodes represent worlds and labelled edges represent i-accessibility. Moreover, nodes are labelled with the *set of formulae* that should be true in the corresponding world. Thus, instead of finding two labelled formulae $w' : \psi'$ and $w' : \psi''$ in a tableau, we would find both formulae in the label of the node w', written $\{\psi', \psi''\} \subseteq \mathcal{L}(w')$.

  An advantage of this approach is that all information concerning a single world is kept in the same place. For example, it allows for the detection of obvious inconsistencies such as $w : p$ and $w : \neg p$ by a test that is local to $\mathcal{L}(w)$. When considering logics with converse

or graded modalities, the advantages of this "one node per world" approach become even more pronounced. State-of-the-art implementations of modal tableau algorithms adopt this approach [159, 90], which is why we have chosen it for this section.

Similarly, the $\vee$-rule is often formulated using either branching or non-determinism in the model construction. For example, if we know that $\psi_1 \vee \psi_2$ should be true in w, then the $\vee$-rule can be formalised in the following two ways:

- we *branch* our model construction into two, one in which ψ_1 is true in w and one in which ψ_2 is true in w, and then continue with the construction of each branch independently.

  This is how non-deterministic constructors are handled in standard first order and modal logic tableau: the tableau rules expand a tree where each branching represents a non-deterministic choice, and thus where each path stands for a possible model.

- we non-deterministically choose one ψ_i to be true in w. This yields a non-deterministic algorithm which, when implemented, requires back-tracking to be complete.

  From a computational perspective, this approach is preferable since, in contrast to the above "branching" alternative, it preserves the useful "one node per world" property. Additionally, it can easily be adapted to exploit techniques developed for solving SAT problems, such as David-Putnam and related heuristics [41, 87]. State-of-the-art implementations of modal tableau algorithms handle disjunctions (and possibly other non-deterministic operators) in this way, and are combined with intelligent back-tracking (or back-jumping) and heuristics to make the "good" choice first, see Section 4.2.

The algorithms described in this chapter will use the latter, non-deterministic formulation, and will work on a single model/tableau at any world in time, where all information concerning each world is stored in a single node.

Figure 7 shows two example applications of different tableau algorithms to decide the satisfiability of the $\mathbf{K}$ formula $\psi = \langle 1\rangle p \wedge \langle 1\rangle q \wedge [1](\neg p \vee \neg q)$. On the left hand side, we show the result of a standard labelled tableau, where we use sequences as labels. First, we have broken down the conjunctions, then generated two new labels $(1, p)$ and $(1, q)$ for the two diamond formulae. Next, we have expanded the box formula for both new worlds, and finally branched for the disjunctions. The resulting tree stands for four different attempts to construct a model, one for each path from a leaf node to the root. Only the one ending in the filled node corresponds to a model since all other branches contain obvious inconsistencies: e.g., the first one contains both $(1, p) : p$ and $(1, p) : \neg p$.

On the right hand side, we show a (successfull) application of the non-deterministic version of a tableau algorithm working on trees. It has generated three nodes w_i with labels that are completely expanded sets of formulae. Here, the edges stand for the accessibility relations, i.e., w_2 and w_3 are 1-accessible from w_1. In contrast, on the left hand side, (the formulae along) one *path* in the tree represents a model, i.e., edges relate formulae that are true in the same model.

4.2 Local satisfiability for multi modal $\mathbf{K}_n$

Before we describe the algorithm, we introduce an appropriate data structure in which to represent models (and later tableaux). Firstly, it will be convenient to assume that all formulae descriptions are in *negation normal form* (see Section 2). The tableau algorithms presented in this section work on *completion trees*: a *completion tree* is a finite tree where each node x is

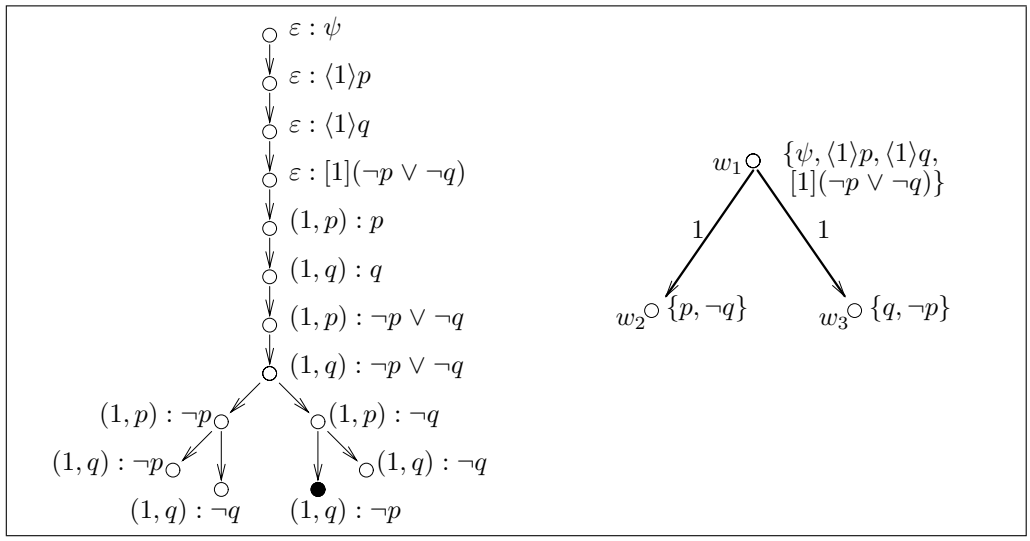

Figure 7. Two application of tableau algorithms to the same formula.

labelled with a *set* $\mathcal{L}(x)$ of formulae, and edges are labelled with modal parameters. A node y is called an *i-successor of* a node x if y is a successor of x and the edge from x to y is labelled with i. A completion tree is said to be *closed* if it contains a node x with $\{p, \neg p\} \subseteq \mathcal{L}(x)$; a completion tree that is not closed is *open*, and it is *complete* if no *expansion rule* applies—the expansion rules are given in Figure 8. Please note that they are formulated in such a way that, if a rule is applicable (i.e., the corresponding condition is satisfied by the current completion tree), then its application indeed changes the tree.

To decide the satisfiability of ϕ (in NNF), the tableau algorithm is started with a completion tree consisting of the root node x_0 only, with $\mathcal{L}(x_0) = \{\phi\}$. It applies the expansion rules until the completion tree becomes closed or complete, and returns "ϕ is satisfiable" if the expansion rules can be applied such that they yield a complete and open tableau, and "ϕ is unsatisfiable" otherwise. The "*can* be applied" formulation is due to the non-deterministic $\vee$-rule, as discussed in Section 4.1. Also, the algorithm does not fix any order in which the rules are to be applied, which means that an implementation has to/can chose a "good" one.

$\wedge$**-rule:**	*If*	there is a node x with $\psi_1 \wedge \psi_2 \in \mathcal{L}(x)$ and $\{\psi_1, \psi_2\} \not\subseteq \mathcal{L}(x)$,
	then	$\mathcal{L}(x) := \mathcal{L}(x) \cup \{\psi_1, \psi_2\}$.
$\vee$**-rule:**	*If*	there is a node x with $\psi_1 \vee \psi_2 \in \mathcal{L}(x)$ and $\{\psi_1, \psi_2\} \cap \mathcal{L}(x) = \emptyset$,
	then	$\mathcal{L}(x) := \mathcal{L}(x) \cup \{\psi_i\}$ for some $i \in \{1, 2\}$.
$\Diamond$**-rule:**	*If*	there is a node x with $\langle i\rangle\psi \in \mathcal{L}(x)$ and x has no i-successor y with $\psi \in \mathcal{L}(y)$,
	then	create a new i-successor y of x with $\mathcal{L}(y) := \{\psi\}$.
$\Box$**-rule:**	*If*	there is a node x with $[i]\psi \in \mathcal{L}(x)$ and x has an i-successor y with $\psi \notin \mathcal{L}(y)$,
	then	$\mathcal{L}(y) := \mathcal{L}(y) \cup \{\psi\}$.

Figure 8. The expansion rules for $\mathbf{K}_n$.

Before discussing the properties of this algorithm, we would like to point out that the tableau

rule

$$\frac{w : \langle i\rangle\varphi \qquad w : [i]\psi_1 \quad \cdots \quad w : [i]\psi_n}{v : \varphi \qquad v : \psi_1 \quad \cdots \quad v : \psi_n}$$

mentioned in Section 3.4 of this chapter corresponds, in our notation, to

If there is a node x with $\{\langle i\rangle\varphi, [i]\psi_1, \ldots, [i]\psi_n\} \subseteq \mathcal{L}(x)$,
then create a new i-successor y of x with $\mathcal{L}(y) := \{\varphi, \psi_1, \ldots, \psi_n\}$.

The fact that our algorithm decides satisfiability of $\mathbf{K}_n$ formulae is an immediate consequence of the following lemma, for which we first need to define the semantics of completion trees. Let $\mathbf{T}$ be a completion tree, $\mathfrak{M} = \langle W, R, V\rangle$ a model, and π a (total) mapping from the nodes of $\mathbf{T}$ to W. Then $\mathfrak{M}$ is said to *satisfy* $\mathbf{T}$ *via* π if, for each node x in $\mathbf{T}$,

1. for each $\psi \in \mathcal{L}(x)$, we have $\mathfrak{M}, \pi(x) \models \psi$ and
2. for each i successor y of x, we have $R_i(\pi(x), \pi(y))$.

LEMMA 18. *Let ϕ be a $\mathbf{K}_n$ formula and $\mathbf{T}$ a completion tree generated by the tableau algorithm for ϕ.*

1 *When applied to ϕ, the tableau algorithm terminates.*

2 *If $\mathfrak{M}$ satisfies $\mathbf{T}$ via π and one of the expansion rules is applicable to $\mathbf{T}$, then this rule can be applied in such a way that it yields a $\mathbf{T}'$ satisfied by $\mathfrak{M}$ via π or an extension of π.*

3 *If $\mathbf{T}$ is complete, then there exists a model $\mathfrak{M}$ and a mapping π such that $\mathfrak{M}$ satisfies $\mathbf{T}$ via π iff $\mathbf{T}$ is open.*

Lemma 18.1 is due to the fact that (i) the breadth and depth of the completion tree are bounded linearly by the length of ϕ, (ii) node labels are sets of subformulae of ϕ, and (iii) the completion tree is built in a monotonic way, i.e., each rule strictly increases node labels or adds new nodes. Property (i) is due to the fact that there are at most $|\phi|$ diamond modalities in ϕ and that the maximal modal depth of formulae in node labels strictly decreases from a node to its (i-)successors. Lemma 18.2 is an immediate consequence of the semantics of $\mathbf{K}_n$ and completion trees. For example, let the $\Diamond$-rule be applicable to some $\mathbf{T}$ with $\langle i\rangle\psi \in \mathcal{L}(x)$, and let $\mathfrak{M}$ satisfy $\mathbf{T}$ via π. Hence $\mathfrak{M}, \pi(x) \models \langle i\rangle\psi$, and thus there exists some $w \in W$ with $R_i(\pi(x), w)$ and $\mathfrak{M}, w \models \psi$. As a consequence, we can extend π to $\pi(y) = w$ for y the newly introduced node, and $\mathfrak{M}$ satisfies the result of this rule application via (the extended) π. The "if" direction of Lemma 18.3 is easy since each open completion tree can be viewed as a model with W the set of nodes, $x \in V(p)$ if $p \in \mathcal{L}(x)$, and $R_i(x, y)$ if y is an i-successor of x. The only-if direction of Lemma 18.3 is trivial.

THEOREM 19. *The $\mathbf{K}_n$ tableau algorithm decides $\mathbf{K}_n$ satisfiability and can be implemented in polynomial space.*

As an immediate consequence of Lemma 18 and the fact that each model $\mathfrak{M}$ satisfying ψ is one that satisfies the initial completion tree (and vice versa), we thus have the first point of Theorem 19. The second part follows from the following observations. As a consequence of (i) and (ii) in the proof sketch of Lemma 18.1, we can store each branch of a completion tree in space bounded polynomially in the length of ψ. Next, we observe that we can consider each branch independently, and thus we can build the completion tree in a depth first manner, keeping only a single branch in memory at each point in time. Finally, our $\vee$-rule is non-deterministic, but it is known how to transform a non-deterministic polynomial space algorithm into a deterministic one that also runs in polynomial space [172].

Implementation Issues

Even this "simplest" modal logic $\mathbf{K}_n$ extends propositional logic, and thus the complexity is rather discouraging from an implementational perspective: we may have to consider a number of models (or completion trees) that is exponential in the size of the input formula. Moreover, because the completion tree is usually built in a depth first manner, with the $\wedge$- and $\vee$-rules being exhaustively applied to a given node before creating any modal successors with the $\Diamond$-rule, it is easy to find formulae with unsatisfiability "hidden" in the leaves of the tree for which a naive implementation will always exhibit pathological worst case behaviour. Consider, for example, the formula:

$$\phi = (p_1 \vee q_1) \wedge \ldots \wedge (p_n \vee q_n) \wedge \langle i \rangle \psi \wedge [i] \neg \psi. \tag{11}$$

There are 2^n different ways in which the combination of the $\wedge$-and $\vee$-rules can be applied to a node whose label is initialised with $\{\phi\}$, but in each case subsequent applications of the $\Diamond$- and the $\Box$-rules will eventually lead to a closed completion tree. A naive implementation of the trace technique with "chronological" backtracking search would consider all 2^n possible expansions before concluding that the input formula is unsatisfiable; this kind of unproductive backtracking search in often referred to as *thrashing*.

Fortunately, a wide range of optimisation techniques has been developed in order to improve the efficiency with which the algorithm explores the space of possible models [104, 103]. Although these optimisations may lead to a situation in which the worst case behaviour would actually be much worse than the theoretical worst case, empirical studies have shown that such optimised algorithms are very effective with *typical* formulae, i.e., formulae derived from applications. These techniques include normalisation and simplification, dependency directed backtracking, SAT based search techniques, simplification of node labels, heuristics and caching.

Normalisation and Simplification As usual, our description of the $\mathbf{K}_n$ tableau algorithm assumes that the input formula is in negation normal form (NNF); this simplifies the (description of the) algorithm, but it means that a completion tree will only be closed when a propositional variable and its negation occur in the same node label. For example, when testing the satisfiability of the formula $(p \wedge q) \wedge \neg(p \wedge q)$, the transformation into NNF would give $(p \wedge q) \wedge (\neg p \vee \neg q)$; in practise this means that, in spite of the "obvious" contradiction, backtracking search will be performed in order to determine that the formula is unsatisfiable.

For this reason, practical algorithms do not transform the input concept into NNF, but include a $\neg$-rule that performs a single (negation) normalisation step (e.g., applying the $\neg$-rule to $\neg(p \wedge q) \in \mathcal{L}(x)$ would cause $\neg p \vee \neg q$ to be added to $\mathcal{L}(x)$), and a completion tree is closed if it contains a node x with $\{\psi, \neg\psi\} \subseteq \mathcal{L}(x)$ for an arbitrary formula ψ. Moreover, in order to facilitate the detection of such closure conditions, the input formula is *normalised* and *simplified* so that logically equivalent formulae are more often syntactically equivalent. This is achieved by (recursively) applying a set of rewrite rules to the input formula, and by ordering conjuncts w.r.t. some total ordering. For example, we re-write $\vee$ and $\Diamond$ formulae as negated $\wedge$ and $\Box$ formulae, respectively; we remove redundant parentheses between conjunctions; we order conjuncts; and we simplify formulae using the following equivalences: $(\psi \wedge \psi) \leftrightarrow \psi$, $\neg\neg\psi \leftrightarrow \psi$, $(\psi \wedge \neg\psi \wedge \rho) \leftrightarrow \neg\top$, $(\psi \wedge \top) \leftrightarrow \psi$, $(\psi \wedge \neg\top) \leftrightarrow \neg\top$, and $[i]\top \leftrightarrow \top$.

If the above transformations are applied to the formula ϕ from the above example (11), then $\langle i \rangle \psi$ would be rewritten as $\neg[i]\neg\psi$, $\neg[i]\neg\psi \wedge [i]\neg\psi$ would be rewritten as $\neg\top$ and $(p_1 \vee q_1) \wedge \ldots \wedge (p_n \vee q_n) \wedge \neg\top$ would be rewritten as $\neg\top$, a formula that is trivially unsatisfiable.

If ψ_1 was added to $\mathcal{L}(x)$ by the
- $\wedge$-rule for $\psi_1 \wedge \psi_2 \in \mathcal{L}(x)$, then $\mathsf{dep}(\psi_j, x) := \mathsf{dep}(\psi_1 \wedge \psi_2, x)$ for each $j \in \{1, 2\}$
- $\vee$-rule for $\psi_1 \vee \psi_2 \in \mathcal{L}(x)$, then $\mathsf{dep}(\psi_j, x) := \mathsf{dep}(\psi_1 \vee \psi_2, x) \cup \{b\}$ for each $j \in \{1, 2\}$
- $\Diamond$-rule for $\langle i \rangle \psi_1 \in \mathcal{L}(x')$, then $\mathsf{dep}(\psi_1, x) := \mathsf{dep}(\langle i \rangle \psi_1, x')$
- $\Box$-rule for $[i]\psi_1 \in \mathcal{L}(x')$, then $\mathsf{dep}(\psi_1, x) := \mathsf{dep}([i]\psi_1, x') \cup \mathsf{dep}(\langle i \rangle \psi_2, x')$ where x was generated by the $\Diamond$-rule for $\langle i \rangle \psi_2 \in \mathcal{L}(x')$

Figure 9. Inductive definition of $\mathsf{dep}(\psi, x)$

Dependency Directed Backtracking As we saw in the above example (11), inherent unsatisfiability concealed in sub-formulae can lead to large amounts of unproductive backtracking search known as thrashing. Although the normalisation and simplification technique described above solved the problem for this example, this might not have been the case if the unsatisfiability caused by the modal sub-formulae had been slightly less trivial. Consider, e.g., the following, only slightly modified formula ϕ':

$$\phi' = (p_1 \vee q_1) \wedge \ldots \wedge (p_n \vee q_n) \wedge \langle i \rangle(\psi \wedge \rho) \wedge [i]\neg\psi. \tag{12}$$

To avoid an exponential search in the case of ϕ', a more sophisticated solution is required, and can be found by adapting a form of dependency directed backtracking called *backjumping*, which has also been used, e.g., in solving constraint satisfiability problems [19] and (in a slightly different form) in the HARP theorem prover [153].

Intuitively, backjumping works by labelling each formula ψ in the label of a node x with a dependency set $\mathsf{dep}(\psi, x)$ indicating the *branching points* (i.e., applications of the $\vee$-rule) on which it depends. In case the completion tree is closed because it contains some node x with $\{\psi, \neg\psi\} \in \mathcal{L}(x)$, we use $\mathsf{dep}(\psi, x)$ and $\mathsf{dep}(\neg\psi, x)$ to identify the most recent branching point b on which ψ or $\neg\psi$ depends. The algorithm can then *jump* back to b over intervening branching points *without* exploring any alternative branches (non-deterministic choices), and make a different non-deterministic choice which might not lead to the same closure condition being encountered. In case no such b exists, the closure did not depend on any non-deterministic choice, and the algorithm stops.

To be more precise, a *branching point* is simply a non-negative integer b indicating the b-th $\vee$-rule application in the run of the tableau algorithm. Initially, for x_0 the root node and ϕ the input formula, $\mathsf{dep}(\phi, x_0) := \emptyset$. The sets $\mathsf{dep}(\psi, x)$ are then defined inductively as shown in Figure 9. In this way, each formula in each node label is associated with a dependency set. If the completion tree is closed because it contains some node x with $\{\psi, \neg\psi\} \in \mathcal{L}(x)$, the *closure dependency set* $S := \mathsf{dep}(\psi, x) \cup \mathsf{dep}(\neg\psi, x)$, and the algorithm backtracks to the b-th $\vee$-rule application (or exits if $b = 0$).

The procedure for expanding a completion tree $\mathbf{T}$ is given in Figure 10. For an input formula ϕ, $\mathbf{T}$ is initialised to contain a single node x_0 with $\mathcal{L}(x_0) = \{\phi\}$ and $\mathsf{dep}(\phi, x_0) := \emptyset$; ϕ is satisfiable if Satisfiable($\mathbf{T}$,0) returns $\{-1\}$ and unsatisfiable otherwise. For example, when expanding the formula ϕ' from 12 above, the $\wedge$-rule might first be applied exhaustively via recursive calls to Satisfiable, resulting in $\{p_1 \vee q_1, \ldots, p_n \vee q_n, \langle i \rangle(\psi \wedge \rho), [i]\neg\psi\} \subseteq \mathcal{L}(x_0)$ and $\mathsf{dep}(\psi_j, x_0) = \emptyset$ for each formula $\psi_j \in \mathcal{L}(x_0)$. These dependencies reflect the fact that, so far, no non-deterministic choices have been made. A top-down and "left to right" strategy might then cause Branch to be called n times, with, for the j-th call, $b = j$, $f_1 = p_j$, $f_2 = q_j$ and

```
Procedure Satisfiable(T,b)
local f;
begin
  if for some node x in T, {ψ, ¬ψ} ∈ L(x)
  then
    return(dep(ψ, x) ∪ dep(¬ψ, x))
  else if T is complete then
    return({−1})
  else
    begin
      f := some unexpanded formula in node
      x in T
      if f is of the form ψ1 ∨ ψ2 then
        return(Branch(T,b                    +
        1,x,ψ1,ψ2,dep(f, x)))
      else
        begin
          expand f (as per Fig. 8 and 9)
          return(Satisfiable(T,b))
        end
    end
end
```

```
Procedure Branch(T,b,x,f1,f2,D)
local S, T-saved;
begin
  T-saved := T
  add f1 to L(x) with dep(f1, x) = {b}∪D
  S := Satisfiable(T,b)
  if b ∉ S then
    return(S)
  else
    begin
      T := T-saved
      add f2 to L(x) with dep(f2, x) =
      b ∪ d
      return(S ∪ Satisfiable(T,b))
    end
end
```

Figure 10. Procedure for tableau expansion with backjumping

$D = \emptyset$, so that $p_1, \ldots, p_n$ are added to $\mathcal{L}(x_0)$ with $\mathsf{dep}(p_j, x_0) = j$. Next, recursive calls to Satisfiable would expand: $\langle i \rangle(\psi \wedge \rho) \in \mathcal{L}(x_0)$, causing the generation of an i-successor x_1 of x_0 with $\mathcal{L}(x_1) = \{\psi \wedge \rho\}$ and $\mathsf{dep}(\psi \wedge \rho, x_1) = \emptyset$; $[i]\neg\psi \in \mathcal{L}(x_0)$, causing $\neg\psi$ to be added to $\mathcal{L}(x_1)$, with $\mathsf{dep}(\neg\psi, x_1) = \emptyset$; and $\psi \wedge \rho \in \mathcal{L}(x_1)$, causing ψ and ρ to be added to $\mathcal{L}(x_1)$, with $\mathsf{dep}(\psi, x_1) = \mathsf{dep}(\rho, x_1) = \emptyset$. The completion tree would then be closed, as $\{\psi, \neg\psi\} \subseteq \mathcal{L}(x_1)$, and Satisfiable would return $\mathsf{dep}(\psi, x_1) \cup \mathsf{dep}(\neg\psi, x_1) = \emptyset$.

If we were using chronological backtracking, the recursion would return to the n-th branching point, i.e., the one where Branch was called with $b = n$, $f_1 = p_n$ and $f_2 = p_n$. $\mathbf{T}$ would be restored to its state prior to adding p_n to $\mathcal{L}(x_0)$, and the rule would be applied again such that q_n was added to $\mathcal{L}(x_0)$. Using backjumping, however, we return from Branch immediately because $b \notin S$. This is obviously true for all of the preceding branching points, so all calls to Branch will return without expanding the completion trees obtained by adding the various q_j to $\mathcal{L}(x_0)$, and Satisfiable will eventually return $\emptyset$, allowing us to conclude that ϕ' is unsatisfiable.

SAT Based Search Techniques Even with the addition of dependency directed backtracking, a naive implementation of the $\Diamond$-rule is inherently inefficient as it can lead to the repetition of parts of the expansion. For example, given an input formula

$$\phi'' = (\rho \vee \psi_1) \wedge \ldots \wedge (\rho \vee \psi_n), \tag{13}$$

where $\psi_1 \wedge \ldots \wedge \psi_n$ is satisfiable but ρ is not, the procedure described above would lead to the construction of n (possibly large) closed completion trees, each with $\rho \in \mathcal{L}(x_0)$, before a complete and open completion tree is constructed.

This problem can be avoided by using more sophisticated search techniques. One of best

known of these is the Davis-Putnam algorithm, originally designed for solving propositional satisfiability (SAT) problems [42]. The basic idea behind Davis-Putnam is that, instead of branching on unexpanded disjunctions, we branch on a formula ψ such that ψ occurs in an unexpanded disjunction in a node x of the completion tree and $\{\psi, \neg\psi\} \cap \mathcal{L}(x) = \emptyset$; the algorithm then searches the two possible trees obtained by adding ψ or $\neg\psi$ to $\mathcal{L}(x)$. This basic technique is usually enhanced with heuristics and simplification rules (which we will discuss in more detail below); in particular, we usually branch first on formulae that occur in many unexpanded disjunctions and, if $\{\psi \vee \rho, \neg\psi\} \subseteq \mathcal{L}(x)$, then $\psi \vee \rho$ is deterministically expanded by adding ρ to $\mathcal{L}(x)$. It is easy to see that if this strategy is applied to ϕ'' above, we would branch first on ρ (as it occurs in n unexpanded disjunctions), and at most one closed completion tree (if ρ is tried first) would be constructed before finding a complete and open one.

This technique has been shown to be very effective with formulae generated at random using generators adapted from those used to generate SAT problems [87, 114]. Such problems typically include a relatively small number of propositional variables (so there is likely to be significant repetition of the sub-formulae occurring in disjunctions), and have a very low modal depth (so the importance of propositional reasoning is emphasised); this is because large numbers of propositional variables and/or a high modal depth would result in almost all problems of reasonable size being trivially satisfiable. Formulae from applications, however, typically do not exhibit these characteristics, and Davis-Putnam is much less effective—in fact it can even be counter-productive if the negated formulae that Davis-Putnam introduces are large and/or complex [103].

An alternative technique used in [56] is to enhance the standard chronological backtracking method with a *no-good list* for each node, i.e., a set of formulae, each of which has already been shown to lead to a closed completion tree when it is added to the node label by an application of $\vee$-rule. Formulae in the no-good list are not considered when applying the $\vee$-rule. Using this technique with ϕ'' above, ρ would be added to the no-good list after the first application of the $\vee$-rule leads to a closed completion tree. In subsequent applications of the $\vee$-rule, ρ would not be considered, and ψ_j would always be selected. This technique has the advantage that wasted search is avoided without adding negated formulae that could themselves lead to additional (possibly non-deterministic) expansion.

Note that, when using these (and other) optimisations in addition to backjumping, care must be taken to ensure that *all* dependencies are being taken into consideration. For example, when using a no-good list to restrict the possible choices made by the $\vee$-rule, it is important to also consider the dependencies associated with the relevant formulae in the no-good list.

Simplification of Node Labels As well as the standard tableau expansion rules described in Figure 8, additional inference rules can be applied to the formulae occurring in a node label, usually with the objective of simplifying them and reducing the number of $\vee$-rule applications. The most commonly used simplification, often called *Boolean Constraint Propagation* (BCP) [74], is again derived from SAT solvers, where it is usually used in conjunction with the Davis-Putnam procedure. The basic idea is to identify a disjunction $\psi_1 \vee \ldots \vee \psi_n \in \mathcal{L}(x)$ such that the negations of all but one of the ψ_j are already elements of $\mathcal{L}(x)$; when this is the case, the formula can be deterministically expanded by adding the relevant ψ_j to $\mathcal{L}(x)$. This amounts to applying the following inference rule

$$\frac{\neg\psi_1, \ldots, \neg\psi_n, \psi_1 \vee \ldots \vee \psi_n \vee \psi}{\psi}$$

to the formulae in a node label, which is a restricted variant of hyper resolution, see 3.1.

As we have already seen, when $\neg\rho$ is added to $\mathcal{L}(x_0)$ during the expansion of ϕ'' above, the BCP rule can be applied to all the remaining $\phi \vee \psi_j$ formulae, leading to a complete and open completion tree without any further applications of the $\vee$-rule. Note that, as with the more sophisticated search techniques described above, careful consideration needs to be given to the dependencies of formulae added by such inference rules if they are to be used together with backjumping.

Heuristics As mentioned in Section 4.1, one advantage of the non-deterministic formulation of the $\vee$-rule is that an algorithm can try to choose a "good" order in which to try the different possible expansions. In practise, this usually means using heuristics to select the way in which the $\vee$-rule is applied to the disjunctions in a node label, and the order in which the successor nodes created by $\Diamond$-rule applications are expanded; in either case, a heuristic function is used to compute the relative "goodness" of candidate formulae/nodes.

When using the Davis-Putnam technique, the well known MOMS heuristic [74] is often used to select the formulae on which to branch; it tries to select formulae that will maximise the effect of BCP and so minimise the number of non-deterministic choices needed in order to complete the completion tree [103]. There is little evidence, however, that (a suitably adapted form of) this heuristic is effective with modal formulae, and even some evidence to suggest that interference with the backjumping optimisation makes it counter productive [103].

An alternative heuristic, whose design was prompted by this observation, tries to maximise the effect of backjumping by preferentially selecting formulae with low valued dependencies [103, 99]. This heuristic has the added advantage that it can also be used to select the order in which successor nodes are expanded.

Caching When using the top-down construction strategy, all information from predecessors is added to a node label before it is processed. This means that, when a given node has been fully expanded (i.e., the expansion rules have been exhaustively applied to it), a successor node y with $\mathcal{L}(y) = \{\psi_1, \ldots, \psi_n\}$ can be treated as an independent problem, equivalent to testing the satisfiability of $\psi_1 \wedge \ldots \wedge \psi_n$.

A completion tree may contain many such nodes, and the labels of nodes tend to be quite similar, particularly as the labels of i-successors of a node x each contain the same formulae resulting from $\Box$-rule applications to $[i]\psi$-formulae in $\mathcal{L}(x)$. For some formulae, this may result in the same sub-problem being solved again and again. In order to avoid this, it is possible to cache and re-use the results of such sub-problems. The usual technique is to use a hash table to store the satisfiability status of node labels (i.e., sets of formulae treated as a conjunction). Before applying any expansion rules to a new node x, the cache is interrogated to determine if the satisfiability status of $\mathcal{L}(x)$ is already known. If it is known, then the result can be used without further expansion, i.e., $\mathcal{L}(x)$ can be treated as though it were either $\{\bot\}$ (for unsatisfiable) or $\{\top\}$ (for satisfiable). If the satisfiability status of $\mathcal{L}(x)$ is not known, then $\mathcal{L}(x)$ is added to the cache, and its status set to satisfiable if a complete and open completion tree rooted in x can be constructed, and to unsatisfiable otherwise.

Since the satisfiability of a set of formulae L implies the satisfiability of each subset of L, and the unsatisfiability of a set of formulae L implies the unsatisfiability of each superset of L, this basic idea can be extended to check for satisfiable supersets of $\mathcal{L}(x)$ and unsatisfiable subsets of $\mathcal{L}(x)$. However, this requires a considerably more sophisticated data structure if cache operations are to be efficient [100, 86].

Apart from the problem of the storage required for the cache, another more subtle disadvantage of caching is that, in the case where the cache returns "unsatisfiable" for $\mathcal{L}(x)$, there is no information about the cause of the unsatisfiability that can be used to derive the dependency

information required for backjumping. Backjumping can still be performed by combining the dependency sets of all of the formulae in $\mathcal{L}(x)$, but this is likely to overestimate the set of branching points on which the unsatisfiability depends.

Another useful form of caching is a technique known as *model merging* [103, 91]. The idea here is to prove the satisfiability of a node label $\mathcal{L}(x)$ by showing that open and complete completion trees for $L_1, \ldots, L_k$ with $L_1 \cup \ldots \cup L_k = \mathcal{L}(x)$ can be combined into an open and complete completion tree for $\mathcal{L}(x)$ by simply "gluing" their root nodes together. This is possible if there are no "interactions" between the various completion trees, e.g., if there are no j, k such that either $\psi \in L_j$ and $\neg\psi \in L_k$ or $\langle i\rangle\psi \in L_j$ and $[i]\rho \in L_k$ for some i, ψ and ρ. Thus model merging involves (a) cashing satisfiable sets of formulae that occur as root labels of open and complete completion trees, and (b) trying to prove the satisfiability of some $\mathcal{L}(x)$ by finding cached sets L_j that do not interact in the above sense.

4.3 Transitive modalities and $\mathbf{K4}_n$

The main problem one has to overcome when modifying the $\mathbf{K}_n$ tableau algorithm presented in Section 4.2 to $\mathbf{K4}_n$ is *termination*. Please recall that the $\mathbf{K}_n$ tableau algorithm terminates "automatically" since it builds a tree of bounded depth and breadth in a monotonic way. As we will see, this is not the case for $\mathbf{K4}_n$. Consider, e.g., the $\mathbf{K4}_n$ formula $\langle i\rangle\psi \wedge [i]\langle i\rangle\psi$. A $\mathbf{K4}_n$ tableau algorithm would start with a root node x_0 labelled with this formula, then apply the $\wedge$-rule, and then generate an i-successor x_1. Next, the $\Diamond$-rule would be applicable and it would add $\langle i\rangle\psi$ to $\mathcal{L}(x_1)$. Thus the $\Diamond$-rule would generate an i-successor x_2 of x_1. At his point, the difference between $\mathbf{K}_n$ and $\mathbf{K4}_n$ becomes apparent: in $\mathbf{K4}_n$ models, R_i has to be transive, and thus x_2 should be i-accessible from x_0, i.e., $\langle i\rangle\psi$ would also need to be true in (the world represented by) x_2. Hence, we would need a (new) rule that adds $\langle i\rangle\psi$ to $\mathcal{L}(x_2)$. However, this would trigger the applicability of the $\Diamond$-rule, which would generate an i-successor x_3 of x_2. Now we can use R_i's transitivity again to argue that $\langle i\rangle\psi$ needs to be added to $\mathcal{L}(x_3)$, and continue the whole pattern to construct an infinite i-chain. Thus the tableau algorithm would not terminate: in contrast to the $\mathbf{K}_n$ tableau algorithm, the maximal modal depth of formulae in node labels no longer decreases from a node to its successors.

To regain termination, we observe that, creating this infinite path, we keep repeating the same actions. More precisely, the node labels of the nodes $x_1, x_2, \ldots$ are all identical. In the following, we show how we can prevent this "looping" using a cycle detection mechanism called "blocking". Intuitively, after the creation of x_2 and the application of the $\Box$-rule, we could have noticed that $\mathcal{L}(x_2) = \{\psi, \langle i\rangle\psi\} = \mathcal{L}(x_1)$, and decided to *not* apply the $\Diamond$-rule to x_2 because (i) we would continue repeating ourselves and (ii) it is not necessary since $\mathcal{L}(x_1) = \mathcal{L}(x_2)$ implies that we can use (the world represented by) x_1 for the world represented by x_2. The latter means that we can build a model $\mathfrak{M}$ with $(x_1, x_1) \in R_i$ in which $\mathfrak{M}, x_1 \models \psi$ and, from the semantics, $\mathfrak{M}, x_1 \models \langle i\rangle\psi$.

The other problem we have to overcome is how we are going to take care of R_i's transitivity. Consider an i-successor y of a node x. Now if y has in turn an i-successor z, then this situation represents a model in which $(x, z) \in R_i$, i.e., z "should" also be an i-successor of x. That is, if $[i]\psi \in \mathcal{L}(x)$, then ψ should be in $\mathcal{L}(z)$. One possible way to achieve this would be to give up on working on completion *trees*, and instead work on graphs where we would add the additional i-edge between x and z. For implementation purposes, however, trees are clearly advantagous, and thus we choose to employ an alternative technique: the same effect as adding the additional i-edge between x and z can be obtained by adding $[i]\psi$ to $\mathcal{L}(y)$ for each $[i]\psi \in \mathcal{L}(x)$. This will

be realized in a modified $\Box$-rule.

Now we formalise this in our tableau algorithm. First, we define the notion of a *blocked* node. We use ancestors and offsprings in the usual way; a node x is *directly blocked* if it has an ancestor x' with $\mathcal{L}(x) \subseteq \mathcal{L}(x')$; a node is *blocked* if it is directly blocked or if it has an ancestor that is directly blocked. Next, we take this notion into account in the $\mathbf{K4}_n$ expansion rules, which are given in Figure 11. Compared to the $\mathbf{K}_n$ expansion rules, these expansion rules only apply to nodes that are not blocked (however, $\Box$-rule can add formulae to the label of a directly blocked node), and the $\Box-$rule "pushes" box formulae in the way discussed above.

$\wedge$**-rule:**	*If*	there is a node x that is not blocked with $\psi_1 \wedge \psi_2 \in \mathcal{L}(x)$ and $\{\psi_1, \psi_2\} \not\subseteq \mathcal{L}(x)$,
	then	$\mathcal{L}(x) := \mathcal{L}(x) \cup \{\psi_1, \psi_2\}$.
$\vee$**-rule:**	*If*	there is a node x that is not blocked with $\psi_1 \vee \psi_2 \in \mathcal{L}(x)$ and $\{\psi_1, \psi_2\} \cap \mathcal{L}(x) = \emptyset$,
	then	$\mathcal{L}(x) := \mathcal{L}(x) \cup \{\psi_i\}$ for some $i \in \{1, 2\}$.
$\Diamond$**-rule:**	*If*	there is a node x that is not blocked with $\langle i \rangle \psi \in \mathcal{L}(x)$ and x has no i-successor y with $\psi \in \mathcal{L}(y)$,
	then	create a new i-successor y of x with $\mathcal{L}(y) := \{\psi\}$.
$\Box$**-rule:**	*If*	there is a node x that is not blocked with $[i]\psi \in \mathcal{L}(x)$ and x has an i-successor y with $\psi \notin \mathcal{L}(y)$,
	then	$\mathcal{L}(y) := \mathcal{L}(y) \cup \{\psi, [i]\psi\}$.

Figure 11. The expansion rules for $\mathbf{K4}_n$.

To convince ourselves that this algorithm indeed decides satisfiability of $\mathbf{K4}_n$ formulae, we sketch the same technical lemma as for $\mathbf{K}_n$.

LEMMA 20. *Let ψ be a $\mathbf{K4}_n$ formula and $\mathbf{T}$ a completion tree generated by the tableau algorithm for ψ.*

1. *When applied to ψ, the tableau algorithm terminates.*

2. *If $\mathfrak{M}$ satisfies $\mathbf{T}$ via π and one of the expansion rules is applicable to $\mathbf{T}$, then this rule can be applied in such a way that it yields a $\mathbf{T}'$ satisfied by $\mathfrak{M}$ via (possibly an extension of) π.*

3. *If $\mathbf{T}$ is complete, then there exists a model $\mathfrak{M}$ and a mapping π such that $\mathfrak{M}$ satisfies $\mathbf{T}$ via π iff $\mathbf{T}$ is open.*

Again, we only sketch the proof. Termination is due to the same three observations as in the sketch of Lemma 18, but the reason for the bound of the depth of the tree is more involved (and the bound is now quadratic). Consider three nodes x, y, and z where y is an i-successor of x and z a j-successor of y. If $i \neq j$, then the maximal modal depth of formulae in the label of z is strictly smaller than the one in the label of y. If $i = j$, then either $\mathcal{L}(z) \subseteq \mathcal{L}(y)$ or y was generated for a different different diamond formula in $\mathcal{L}(x)$ than z in $\mathcal{L}(y)$. In the former case, z is blocked. The latter case can only occur linearly often in the length of the input formula. As a consequence, paths in the completion tree are of length at most quadratic in the length of the input formula. Lemma 20 (ii) is similar to the $\mathbf{K}_n$ case, but we have to exploit the transitivity of R_i to explain why pushing $[i]\psi$ from a node to its i-successor preserves $\mathfrak{M}$ being a model via π. Finally, the construction of a model from an open, complete completion tree in Lemma 20 (iii) is slightly modified: firstly, only un-blocked nodes represent worlds in the model. Secondly, we also add (x, y') to R_i if x has an i-successor y which is blocked and y' is an ancestor of y with

$\mathcal{L}(y) \subseteq \mathcal{L}(y')$. Thirdly, we extend R_i so that it is transitively closed, i.e., if $\{(x,y),(y,z)\} \subseteq R_i$, then we also have $(x,z) \in R_i$.

The same reasons as for $\mathbf{K}_n$ then yield the following theorem.

THEOREM 21. *The $\mathbf{K4}_n$ tableau algorithm decides $\mathbf{K4}_n$ satisfiability and can be implemented in polynomial space.*

Implementation Issues

As we have seen, the main difference between the tableau algorithms for $\mathbf{K}_n$ and $\mathbf{K4}_n$ is the introduction of blocking. In fact the blocking condition described above, which specifies a subset relationship between the labels of blocked and blocking nodes, is already optimised w.r.t. the one originally described in [93], which specified label equality. The subset condition means that blocking can occur sooner, thus avoiding possibly costly expansion.

Consider, for example, a node x labelled as follows:

$$\mathcal{L}(x) = \{\rho, \psi, \langle i\rangle\psi, [i]\langle i\rangle\psi\}, \tag{14}$$

With subset blocking, an i-successor y of x with $\mathcal{L}(y) = \{\psi, \langle i\rangle\psi, [i]\langle i\rangle\psi\}$ would be blocked by x; with equality blocking, a block would not be established until an i-successor z of y is constructed, with $\mathcal{L}(z) = \mathcal{L}(y)$. This may lead to significant additional work if ψ is itself a large and/or complex formula.

Apart from blocking, the algorithm is very similar to the $\mathbf{K}_n$ case, and most of the optimisation techniques described in Section 4.2 can be applied without modification. Blocking does, however, mean that additional care is required when caching and re-using the satisfiability of a set of formulae, because the satisfiability of the set of formulae in the label of a blocked node is contingent on the satisfiability of the set of formulae in the label of the blocking node [103]. This dependency also extends to the satisfiability of the sets of formulae in the labels of any nodes on the path between the blocking node and the blocked node.

Consider, for example, a node x labelled as in 14 above, where ψ is unsatisfiable. As we have seen, an application of the $\Diamond$-rule to $\langle i\rangle\psi \in \mathcal{L}(x)$, followed by applications of the $\Box$-rule to $[i]\langle i\rangle\psi \in \mathcal{L}(x)$, would lead to the creation of an i-successor y of x with $\mathcal{L}(y) = \{\psi, \langle i\rangle\psi, [i]\langle i\rangle\psi\}$, and no expansion rule would be applicable to y as it would be blocked by x. Updating the cache to indicate that the set of formulae $\mathcal{L}(y)$ is satisfiable would, however, clearly be an error, as ψ is unsatisfiable.

4.4 Non-logical axioms and background theories

Now that we have understood how to handle transitivity in $\mathbf{K4}_n$, understanding how to handle background theories is easy. Consider the satisfiability of a formula ϕ w.r.t. the background theory $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$, and remember that the nodes of our completion tree represent worlds of the model we are trying to build, which has to be a common model of ϕ and Γ. Moreover, $\psi \in \mathcal{L}(x)$ stands for the fact that ψ is true in the world (represented by) x. As before, at least one node (the root node) will carry ϕ in its label. Additionally, we will make sure that all nodes will carry each γ_i in their label. As a consequence, we will have a similar problem with termination as we have seen for $\mathbf{K4}_n$, i.e., the maximal modal depth of formulae in node labels does no longer decrease from a node to its successor. Fortunately, we can use the same blocking technique as for $\mathbf{K4}_n$: a node x is *directly blocked* if it has an ancestor x' with $\mathcal{L}(x) \subseteq \mathcal{L}(x')$, and it is *blocked* if it is directly blocked or if it has an ancestor that is directly blocked. The expansion rules for

$\mathbf{K}_n$ w.r.t. background theories are given in Figure 12: they contain the $\mathbf{K}_n$ $\Box-$rule, an additional Γ-rule that adds Γ to each node label, and the $\mathbf{K4}_n$ restriction to blocked nodes. We call the resulting algorithm the *extended* $\mathbf{K}_n$ tableau algorithm.

$\wedge$**-rule:**	*If*	there is a node x that is not blocked with $\psi_1 \wedge \psi_2 \in \mathcal{L}(x)$ and $\{\psi_1, \psi_2\} \not\subseteq \mathcal{L}(x)$,
	then	$\mathcal{L}(x) := \mathcal{L}(x) \cup \{\psi_1, \psi_2\}$.
$\vee$**-rule:**	*If*	there is a node x that is not blocked with $\psi_1 \vee \psi_2 \in \mathcal{L}(x)$ and $\{\psi_1, \psi_2\} \cap \mathcal{L}(x) = \emptyset$,
	then	$\mathcal{L}(x) := \mathcal{L}(x) \cup \{\psi_i\}$ for some $i \in \{1, 2\}$.
$\Diamond$**-rule:**	*If*	there is a node x that is not blocked with $\langle i \rangle \psi \in \mathcal{L}(x)$ and x has no i-successor y with $\psi \in \mathcal{L}(y)$,
	then	create a new i-successor y of x with $\mathcal{L}(y) := \{\psi\}$.
$\Box$**-rule:**	*If*	there is a node x that is not blocked with $[i]\psi \in \mathcal{L}(x)$ and x has an i-successor y with $\psi \notin \mathcal{L}(y)$,
	then	$\mathcal{L}(y) := \mathcal{L}(y) \cup \{\psi\}$.
Γ**-rule:**	*If*	there is a node x that is not blocked with $\Gamma \not\subseteq \mathcal{L}(x)$,
	then	$\mathcal{L}(x) := \mathcal{L}(x) \cup \Gamma$.

Figure 12. The expansion rules for $\mathbf{K}_n$ with background theories.

We can state and prove an analogous technical lemma as for $\mathbf{K}_n$ and $\mathbf{K4}_n$, and then use the same reasons to conclude the following theorem.

THEOREM 22. *The extended $\mathbf{K}_n$ tableau algorithm decides satisfiability w.r.t background theories.*

However, in contrast, we no longer can implement our tableau algorithm in polynomial space: firstly, it is known that satisfiability of $\mathbf{K}_n$ formulae w.r.t. background theories is ExpTime-complete (we can adapt the proofs in [68, 162]). Secondly, we can easily construct a formula and a background theory such that each of their model contains a path of length exponential in the input formulae: we can use propositional variables $p_1, \ldots, p_\ell$ as a "binary counter" for numbers between 0 and $2^\ell - 1$, and non-logical axioms to enforce that, if the p_i at a world w represent a number k, then the p_i at a world w' with $(w, w') \in R_i$ represent the number $k + 1 \mod \ell$. Thirdly, in the worst case, our algorithm indeed constructs completion trees that are of depth exponential in the length of the input formulae: for $\mathbf{K4}_n$, we could argue that the maximal modal depth decreases from a node to a j-successor of its i-successor (if $i \neq j$). For $\mathbf{K}_n$ with background theories, this is no longer true. As a consequence of this exponential length and the non-deterministic $\vee$-rule, our tableau algorithm runs, in the worst case, in non-deterministic double exponential time—which is clearly sub-optimal. In [56], an optimal tableau algorithm for (the description logic) variant of $\mathbf{K}_n$ with background theories was presented; however, to the best of our knowledge, this algorithm has never been implemented, whereas the sub-optimal one described here has proven to work surprisingly well in practice [159, 90].

Implementation Issues

One obvious consequence of the above algorithm is that expansion of the formulae in Γ occurs in every node in the completion tree, and this can easily lead to an explosion in the size of the completion tree or in the number of different possible completion trees that can be (non-deterministically) constructed for a given input formula. For example, if $\Gamma = \{\langle i \rangle \psi_1, \ldots, \langle i \rangle \psi_n\}$, a completion tree containing $nn! + 1$ nodes will be constructed. Similarly, if $\gamma \in \Gamma$, with

$\gamma = ((p_1 \vee q_1) \wedge \ldots \wedge (p_n \vee q_n))$, and the input formula leads to the construction of a completion tree containing k nodes, then there are 2^{kn} different ways to apply the $\wedge$- and $\vee$-rules to the resulting k copies of γ. This explosion in the size of the search space can easily lead to a catastrophic degradation in performance, even when optimisations such as backjumping and caching are employed [102].

Fortunately, optimisations known as *lazy unfolding* and *absorption* have proved to be very effective in reducing the size of the search space, particularly for background theories derived, e.g., from class based knowledge representation formalisms.

Lazy Unfolding In background theories, formulae are often (restricted to be) of the form $p \rightarrow \psi$ or $p \leftrightarrow \psi$ for some propositional variable p. A theory

$$\Gamma = \{p_1 \leftrightarrow \psi_1, \ldots, p_\ell \leftrightarrow \psi_\ell, p_{\ell+1} \rightarrow \psi_{\ell+1}, \ldots, p_{\ell+m} \rightarrow \psi_{\ell+m}\}$$

is said to be *unfoldable*, if it satisfies the following conditions.

- Formulae in Γ are *unique*. I.e., for each propositional variable p, Γ contains at most one formula of the form $p \leftrightarrow \psi$ (i.e., $p_i \neq p_j$ for $1 \leqslant i < j \leqslant \ell$), and if it contains a formula of the form $p \leftrightarrow \psi$, then it does not contain any formulae of the form $p \rightarrow \psi$. (Note that an arbitrary set of formulae $\{p \rightarrow \psi_1, \ldots, p \rightarrow \psi_n\}$ can be combined into a single formula $p \rightarrow (\psi_1 \wedge \ldots \wedge \psi_n)$.)
- Γ is *acyclic*. I.e., there is no formula $p_i \leftrightarrow \psi_i \in \Gamma$ such that p_i occurs either directly or indirectly in ψ_i.[3] A propositional variable p occurs indirectly in a formula ψ if there is a propositional variable formula p' such that p' occurs directly in ψ, and there is a formula $p' \leftrightarrow \psi' \in \Gamma$ such that p occurs either directly or indirectly in ψ'.

Instead of being dealt with using the Γ-rule, such a set of formulae can be lazily *unfolded* during the tableau expansion. I.e., for a formula $p_1 \rightarrow \psi_1 \in \Gamma$, if p_i is added to $\mathcal{L}(x)$ for some node x, then ψ_i is also added to $\mathcal{L}(x)$, and for a formula $p_j \leftrightarrow \psi_j \in \Gamma$, if p_j ($\neg p_j$) is added to $\mathcal{L}(x)$ for some node x, then ψ_j (resp. $\neg\psi_j$) is also added to $\mathcal{L}(x)$.

It is obvious that an arbitrary background theory Γ can be divided into an unfoldable part Γ_u and a general part Γ_g such that $\Gamma_u \cup \Gamma_g = \Gamma$ and $\Gamma_u \cap \Gamma_g = \emptyset$. The unfoldable part Γ_u can then be dealt with using lazy unfolding while the general part Γ_g is dealt with using the Γ-rule.

In fact it has been shown that the definition of an unfoldable theory can be extended somewhat while still allowing the use of the above lazy unfolding technique. In particular, the formulae occurring on the left hand side of (bi-) implications can also be negated propositional variables, and the acyclicity condition can be relaxed by distinguishing positive and negative occurrences of propositional variables in a stratified theory [109, 132].

Absorption Given the effectiveness of lazy unfolding in dealing with the unfoldable part of a background theory Γ, it makes sense to try to rewrite the formulae in Γ so that the size of Γ_g can be reduced. Absorption is just such a rewriting optimisation.

The idea behind absorption derives from the observation that (apparently non-unfoldable) formulae in Γ_g are often of the form $p \wedge \rho \rightarrow \psi$. This formula can be rewritten as $p \rightarrow (\psi \vee \neg\rho)$, which allows it to be moved from Γ_g to Γ_u, provided that Γ_u does not already contain a formula of the form $p \leftrightarrow \psi'$. In case Γ_u does contain such a formula, then the technique can be extended by using the formulae in Γ_u to perform further rewriting. E.g., if $p \leftrightarrow \psi_i \in \Gamma_u$ and $p \rightarrow \psi_j \in \Gamma_g$, then the second formula can be rewritten as $\psi_i \rightarrow \psi_j$ and, if ψ_i is of the form $q \wedge \psi_i'$, the formula

[3] For the purposes of lazy unfolding, only cycles consisting entirely of $\leftrightarrow$ axioms are problematical.

can be further rewritten as $q \rightarrow \psi_j \vee \neg\psi'_i$. A more detailed description of the various re-writings used in absorption can be found in [109].

4.5 Converse modalities

So far, our tableau algorithms only use expansions rules that are either local to a single node, create new successors, or push formulae from a node label into the label of a *successor*. The objective of this section is to discuss a tableau algorithm for $\mathbf{K}_n^{\smile}$, i.e., $\mathbf{K}_n$ with converse modalities. It is well-known that satisfiability in $\mathbf{K}_n^{\smile}$ can be polynomially reduced to the satisfiability of $\mathbf{K}_n$ w.r.t. background theories [43]. However, from an implementation perspective, this approach is not feasible since it leads to a dramatic performance degradation, and we thus present a direct algorithm.

As mentioned in Section 2, $\mathbf{K}_n^{\smile}$ requires reasoning in both ways over relations R_i. For our tableau algorithm, this will simply mean that we push formulae up and down in a completion tree. To realize this, we define the notion of an i-neighbour, which requires a few other concepts: firstly, to avoid numerous case distinction, we introduce a function $\mathsf{Cv}(\cdot)$ on modal parameters as follows:[4] $\mathsf{Cv}(i) = i^{\smile}$ and $\mathsf{Cv}(i^{\smile}) = i$. Next, we consider completion trees where each edge is labelled with a possibly converse modal parameter i or $i^{\smile}$. Finally, for α a (possibly converse) modal parameter, we call a node y an *α-neighbour* of a node x if y is an α-successor of x or if x is a $\mathsf{Cv}(\alpha)$-successor of y. The expansion rules for $\mathbf{K}_n^{\smile}$ are identical to those for $\mathbf{K}_n$, with the only difference being that the $\Box$- and the $\Diamond$-rules now consider α-neighbours instead of α-successors (but the $\Diamond$-rule still generates an α-successor if no appropriate α-neighbour is available); they can be found in Figure 13.

$\wedge$**-rule:**	*If*	there is a node x with $\psi_1 \wedge \psi_2 \in \mathcal{L}(x)$ and $\{\psi_1, \psi_2\} \not\subseteq \mathcal{L}(x)$,
	then	$\mathcal{L}(x) := \mathcal{L}(x) \cup \{\psi_1, \psi_2\}$.
$\vee$**-rule:**	*If*	there is a node x with $\psi_1 \vee \psi_2 \in \mathcal{L}(x)$ and $\{\psi_1, \psi_2\} \cap \mathcal{L}(x) = \emptyset$,
	then	$\mathcal{L}(x) := \mathcal{L}(x) \cup \{\psi_i\}$ for some $i \in \{1, 2\}$.
$\Diamond$**-rule:**	*If*	there is a node x with $\langle\alpha\rangle\psi \in \mathcal{L}(x)$ and x has no α-neighbour y with $\psi \in \mathcal{L}(y)$,
	then	create a new α-successor y of x with $\mathcal{L}(y) := \{\psi\}$.
$\Box$**-rule:**	*If*	there is a node x with $[\alpha]\psi \in \mathcal{L}(x)$ and x has an α-neighbour y with $\psi \notin \mathcal{L}(y)$,
	then	$\mathcal{L}(y) := \mathcal{L}(y) \cup \{\psi\}$.

Figure 13. The expansion rules for $\mathbf{K}_n^{\smile}$.

We can state and prove an analogous technical lemma as for $\mathbf{K}_n$, and then use similar reasons to conclude the first part of following theorem.

THEOREM 23. *The $\mathbf{K}_n^{\smile}$ tableau algorithm decides $\mathbf{K}_n$ satisfiability and can be implemented in polynomial space.*

To implement the $\mathbf{K}_n^{\smile}$ tableau algorithm in polynomial space, we can use the following "restart" technique: for each node x, we first apply the $\wedge$- and the $\vee$-rule exhaustively.[5] Next, if a formula is added to $\mathcal{L}(x)$ by the $\Box$-rule for some $[\alpha]\psi$ in a $\mathsf{Cv}(\alpha)$-successor of x, then we disregard the whole sub-tree below x and re-start its construction from scratch. As a consequence of this "strategy", all branches of a completion tree are independent, and we can still construct a completion tree depth first.

[4] Remember that $\mathbf{K}_n$ with converse modalities provides modal parameters i and $i^{\smile}$ for $1 \leq i \leq m$.

[5] Please note that we never made any assumptions or restrictions on the order in which the rules are to be applied.

Implementation Issues

Although the restart technique can be used to enable $\mathbf{K}_n^{\smile}$ completion trees to be constructed using a depth first strategy, the technique is not used in practice as rebuilding discarded parts of the completion tree can be very costly (and space usage is rarely a problem in practice). Without this technique, however, extra care is required when using some of the optimisation techniques described above.

Without the depth first strategy, the satisfiability of (the formula represented by) the label of a node x can no longer be treated as an independent problem, because the results of expanding x might affect its predecessor (unless x is the root node). This means that, although we can re-use cached unsatisfiability results from the cache as before, we must either disregard satisfiable results, or use more sophisticated caching techniques (e.g., storing additional information that would allow us to check for possible interactions with the predecessor node) [103].

Computation of the dependencies used in backjumping is also made more difficult by the loss of the depth first strategy. In particular we need to consider the dependency set of the $\langle i \rangle$ formula in x that led to the generation of an i-successor y in order to compute $\mathsf{dep}(\psi, y)$ when ψ is added to $\mathcal{L}(y)$ as a result of a $\Box$-rule application to a formula $[i]\psi \in \mathcal{L}(x)$. With depth first expansion, this is usually accomplished by combining $\Diamond$-rule applications with all relevant $\Box$-rule applications. Without depth first expansion, this is usually achieved by extending the labelling of either nodes or edges with the dependency set of the $\Diamond$-formula that caused them to be added to the completion tree.

Finally, without the depth first strategy it is necessary, in general, to save the state of the whole completion tree at each $\vee$-rule application (as mentioned above, the depth first strategy allows state saving and restoring to be restricted to a single node label). This problem can be ameliorated by using a lazy state saving strategy, where node labels are only saved when they are about to be extended by some rule application.

4.6 Converse modalities and background theories

In the last sections, we have seen how to extend the basic $\mathbf{K}_n$ tableau algorithm to a decision procedure for $\mathbf{K4}_n$, for $\mathbf{K}_n$ with background theories, and for $\mathbf{K}_n^{\smile}$. For the first two extensions, we discussed a technique to "artificially" ensure termination while preserving soundness and completeness. For the third extension, we introduced the concept of *neighbours* and modified the expansion rules as to work up and down the completion tree. In this section, we will put these techniques and concepts together—and show that their combination requires a further adjustment.

To be more precise, in this section, we discuss a tableau algorithm for $\mathbf{K}_n^{\smile}$ with background theories, i.e., converse modal parameters can occur both in the input formula and in the formulae of the background theory. Next, we discuss the expansion rules, which are given in Figure 14. Clearly, in the presence of converse modal parameters, we use the notion of α-neighbours. Similarly, in the presence of background theories, we use the Γ-rule, and we use blocking to ensure termination. However, the combination of background theories with converse modal parameters requires two modifications. Consider an i-successor y of x with $[i^{\smile}]\psi \in \mathcal{L}(y)$, and assume that y is blocked and x is not blocked. Hence there is some node y' with $\mathcal{L}(y) \subseteq \mathcal{L}(y')$. In case the tableau algorithm stops with an open, complete completion tree, we will try to construct a model $\mathfrak{M}$ from this tree, and we will have $(x, y') \in R_i$. Now $\mathcal{L}(y) \subseteq \mathcal{L}(y')$ implies that $[i^{\smile}]\psi \in \mathcal{L}(y')$, and we thus have to show that $\mathfrak{M}, x \models \psi$. However, if we would not apply the $\Box$-rule to y

because y is blocked, we might not find $\psi \in \mathcal{L}(x)$, and thus our construction might fail. This observations leads to the first modification:

1. we call a node *indirectly blocked* if it is blocked, and if its predecessor is blocked as well. Then we apply all but the $\Diamond$-rule to nodes that are not indirectly blocked.

In our example case, y was indirectly blocked, and thus the $\Box$-rule would add ψ into $\mathcal{L}(x)$. Next, consider some $[i^\smile]\psi' \in \mathcal{L}(y') \setminus \mathcal{L}(y)$. The same reasons as for $[i^\smile]\psi$ imply that we should find $\psi' \in \mathcal{L}(x)$—which we would not since our blocking condition only requires $\mathcal{L}(y) \subseteq \mathcal{L}(y')$. This observation leads to the second modification:

2. a node x is *directly blocked* if it has an ancestor x' with $\mathcal{L}(x') = \mathcal{L}(x)$.

For obvious reasons, we refer to the former blocking condition as *subset blocking*, and to this new condition as *equality blocking*. Please note that, in this setting, it is unavoidable that blocking is "dynamic", that is, a blocked node can later become not blocked. In contrast, with a certain strategy for the order of rule applications, this can be avoided in the $\mathbf{K}_n$ case.

$\wedge$**-rule:**	*If*	there is a node x that is not indirectly blocked with $\psi_1 \wedge \psi_2 \in \mathcal{L}(x)$ and $\{\psi_1, \psi_2\} \not\subseteq \mathcal{L}(x)$,
	then	$\mathcal{L}(x) := \mathcal{L}(x) \cup \{\psi_1, \psi_2\}$.
$\vee$**-rule:**	*If*	there is a node x that is not indirectly blocked with $\psi_1 \vee \psi_2 \in \mathcal{L}(x)$ and $\{\psi_1, \psi_2\} \cap \mathcal{L}(x) = \emptyset$,
	then	$\mathcal{L}(x) := \mathcal{L}(x) \cup \{\psi_i\}$ for some $i \in \{1, 2\}$.
$\Diamond$**-rule:**	*If*	there is a node x that is not blocked with $\langle\alpha\rangle\psi \in \mathcal{L}(x)$ and x has no α-neighbour y with $\psi \in \mathcal{L}(y)$,
	then	create a new α-successor y of x with $\mathcal{L}(y) := \{\psi\}$.
$\Box$**-rule:**	*If*	there is a node x that is not indirectly blocked with $[\alpha]\psi \in \mathcal{L}(x)$ and x has an α-neighbour y with $\psi \notin \mathcal{L}(y)$,
	then	$\mathcal{L}(y) := \mathcal{L}(y) \cup \{\psi\}$.
Γ**-rule:**	*If*	there is a node x that is not indirectly blocked with $\Gamma \not\subseteq \mathcal{L}(x)$,
	then	$\mathcal{L}(x) := \mathcal{L}(x) \cup \Gamma$.

Figure 14. The expansion rules for $\mathbf{K}_n^\smile$ with background theories.

4.7 Other extensions (counting, nominals, transitive closure, and fixpoints)

In this section, we discuss two other extensions of our tableau algorithms. Firstly, we discuss $\mathbf{K}_n^{\smile,o}$, the extension of $\mathbf{K}_n^\smile$ with background theories and *nominals*. Secondly, we discuss $\mathbf{K}_n^c$, the extension of $\mathbf{K}_n$ with graded modalities, and also how to ensure termination in the additional presence of background theories. Finally, we discuss modal logics with a transitive closure operator and fixpoints.

Further add nominals to $\mathbf{K}_n^\smile$ with background theories

$\mathbf{K}_n^{\smile,o}$ with background theories is of interest because it lacks the tree models property and because it requires another form of non-local reasoning. The former point was already discussed in Section 2. To see the latter point, consider the formula $\langle i\rangle(p \wedge \langle\ell\rangle o) \wedge \langle j\rangle o \wedge [j][\ell^\smile]\langle i\rangle(p \wedge \langle\ell\rangle o) \wedge \langle j\rangle(o \wedge [\ell^\smile]q)$. The first three conjuncts imply the existence of an infinite

(possibly cyclic) R_i-path $w_1, w_2, \ldots$ such that *the* world in which o is true is ℓ-accessible from each w_k. The fourth conjunct implies that, in all w_k, q is true—however, this is only "detected" when the $\Diamond$-rule is applied to the fourth conjunct.

To handle, additionally, nominals, we can further modify our extended $\mathbf{K}_n^{\smile}$ tableau algorithm as follows. Firstly, we give up completion trees. More precisely, if $o_1, \ldots, o_\ell$ are all nominals occurring in ϕ and Γ, we start our tableau with $\ell + 1$ root nodes x_i, where $\mathcal{L}(x_0) = \{\phi\}$ and $\mathcal{L}(x_i) = \{o_i\}$, for each $1 \leq i \leq \ell$. Then, whenever we find a nominal o_i in a node $x \neq x_i$, we *merge* x into x_i; that is, we merge x and x_i's labels and incoming and outgoing edges. As a consequence of this merging, we will possibly find several edges going into a nominal node x_i; however, removing these edges clearly yields a forest structure. Correctness is then straightforward, and termination is due to the fact that (a) each path starting at some x_j is of bounded length because of blocking, and (b) if a successor node was created for some $\Diamond i\psi \in \mathcal{L}(x)$, then we will not create it "again", even if x was merged into another node. For details, see [5, 106].

Further add graded modalities to $\mathbf{K}_n$

In this section, we will discuss, on a rather abstract level, what modifications are necessary to handle graded modalities $\langle i \rangle_n \phi$ and $[i]_n \phi$; for a more detailed description, see [108, 101].

Firstly, following our previous approach, it is quite obvious that, when we find $\langle i \rangle_n \psi \in \mathcal{L}(x)$, we should make sure that we find $n + 1$ i-successors y_j of x with $\psi \in \mathcal{L}(y_j)$. Usually, when we do not find them, we create them all in a single step. Similarly, if we find $[i]_n \psi \in \mathcal{L}(x)$, we must make sure that we do not find more than n i-successors y_j of x with $\psi \in \mathcal{L}(y_j)$. Thus, if there are more such i-successors, we *merge* two of them, say y_j and y_k, i.e., we merge y_k's node label and outgoing edges into y_j's and remove y_k, thus reducing the number of such i-successors by one.

Secondly, in the presence of contradicting graded modalities $\langle i \rangle_n \psi$ and $[i]_{n'} \psi'$ with $\psi \rightarrow \psi'$ and $n' \leq n$ in the label of a node x, the above naive approach would lead to the repeated generation and merging of i-successors of x, and thus to non-termination. To prevent this "yoyo"-effect, when introducing $n+1$ i-successors for some $\langle i \rangle_n \psi \in \mathcal{L}(x)$, we use an explicit inequality relation between these i-successors, do not merge "explicitly unequal" nodes, and extend the notion of a clash to also cover the case where $[i]_n \psi \in \mathcal{L}(x)$ but x has more than n "explicitly unequal" i-successors with ψ in their label.

Thirdly, these modification yield a terminating yet unsound decision procedure: consider, for example, the formula $[i]_1 p \wedge [i]_1 \neg p \wedge \langle i \rangle_2 q$. With the modifications made so far, our tableau algorithm would generate three (explicitly unequal) i-successors y_j of a root node x_0 with $q \in \mathcal{L}(y_j)$, stop, and return "satisfiable", which is clearly the wrong answer. The reason for this incorrect answer is that we only merge surplus i-successors for some $[i]_n \psi$ if we already know that they must satisfy ψ, i.e., if ψ is found in their label. However, as the previous example shows, this is not enough: if $[i]_n \psi \in \mathcal{L}(x)$, we must determine, for each i-successor y_j, whether it does or does not satisfy ψ. We can do this using an additional, non-deterministic *choose*-rule that adds, to each such i-successor, either ψ or the negation normal form of $\neg\psi$.

For $\mathbf{K}_n$, these modifications lead to a decision procedure for satisfiability, even in the presence of either background theories or converse modalities (where we only have to take care to count and merge *i-neighbours* correctly). However, for $\mathbf{K}_n^{\smile}$ with background theories, we need a further modification, namely one to the blocking condition: otherwise, the algorithm is not correct (see, e.g., the example in [105]). Since this logic lacks the finite model property, a construction

of a model from a completion tree uses standard unravelling where, instead of a path going to a blocked node, it goes to the node blocking it. Now, in the presence of graded modalities, we must make sure that this does not lead to additional i-accessible worlds which thus would violate some graded modal formulae. Roughly speaking, we ensure this using *double blocking*, i.e., instead of a node being blocked by an ancestor, a node and its predecessor is blocked by an ancestor and its respective predecessor. For details, see [105, 108].

Implementation Issues

As we have seen, the tableau algorithm $\mathbf{K}_n^{\smile}$ requires a more complex blocking condition in order to ensure that a completion tree can be unravelled into an infinite tableau. This can adversely affect performance, because blocks can take (much) longer to establish, and the completion tree can thus grow (much) larger. The problem can be ameliorated by using a more precise (weaker) blocking condition that identifies the cases where double blocking is really needed (i.e., where a cyclical model cannot be built from a branch of the completion tree blocked using the original single blocking condition), and compares only those parts of the node label pairs that are relevant to determining if the completion tree could be unravelled to give an infinite tableau [107].

Transitive Closure and Fixpoints

There are various extension of modal logics with transitive closure operators and general fixpoints, see Chapter 12 of this handbook. However, there are only few "practicable" satisfiability algorithms in the sense that one could dare to implement them and expect a reasonable behaviour in any non-trivial case.[6] To the best of our knowledge, there are only two such algorithms based on tableau, namely the ones described in [11, 45] for extensions of $\mathbf{K}_n$ with transitive closure, and there has only been a single attempt at an implementation, namely in the system DLP [159]. For this kind of extensions, automata-based techniques (see 5.1) seem to be suited best: for example, the only known decision procedure for the μ-calculus is based on automata, see Chapter 12 of this handbook.

5 OTHER COMPUTATIONAL APPROACHES

5.1 Automata-based algorithms

Roughly speaking, automata-based algorithms work as follows. To decide the satisfiability of a logic $\mathcal{L}$, we first show an appropriate tree-model property for $\mathcal{L}$, i.e., prove that each satisfiable $\mathcal{L}$ formula is satisfiable in a model (or an abstraction of a model) whose relational structure forms a tree. For example, it is well-known that each satisfiable $\mathbf{K}_n$ formula is satisfiable in a tree model which is, additionally, finite [93]. For other logics, e.g., $\mathbf{K4}_n$, we can easily show that each satisfiable formula has a model with an infinite tree *abstraction*, where we can obtain a model from such an abstraction by transitively closing the accessibility relations [93]. Secondly, for an $\mathcal{L}$ formula ϕ, we define an automaton $\mathcal{A}_\phi$ such that $\mathcal{A}_\phi$ accepts all tree models of ϕ (or abstractions thereof). Depending on the logic and its model properties, we use automata on finite or on infinite trees. Thus we have reduced the satisfiability of formulae in $\mathcal{L}$ to the emptiness

[6] For other reasoning problems such as model checking, these algorithms exist and have been implemented successfully, see Chapter 17 of this handbook.

problem of a certain class of automata, and we can use well-known algorithms to decide these emptiness problems.

For a variety of logics, this approach has several of advantages. Consider, for example, $\mathbf{K}_n$ with background theories. It can easily be seen that this logic enjoys the tree model property, and thus we only need to devise the construction of an automaton $\mathcal{A}_\phi$. Using *alternating automata*, this construction is quite straightforward and yields, surprisingly, a (worst-case) optimal decision procedure (for a similar construction for a more powerful logic see, e.g., [188]): the automaton $\mathcal{A}_\phi$ is of size polynomial in the input tree, and testing its emptiness can be done in deterministic exponential time [128]. Thus, in contrast to the tableau algorithm described in Section 4.4, we effortlessly obtain a *deterministic* algorithm, and do not even need to take care of termination or finite models: using automata on infinite trees makes this unnecessary.

Concerning the implementability of automata-based approaches, we observe that their worst-case complexity often coincides with their best-case complexity: to decide the emptiness of alternating automata, we first translate them into non-deterministic ones that are then tested for emptiness, i.e., we first build a structure of exponential size, for which we then decide emptiness in polynomial time [128]. In case we directly use non-deterministic automata, they tend to be of size exponential in the size of the input formula, and we are thus confronted with the same problem. Thus, any naive implementation is doomed to failure. However, there are at least two ways out: in [158], it was shown how BDDs can be used to efficiently represent and handle large automata, thus proving that (variations of) automata-based algorithms can be implemented efficiently using appropriate data structures. In [12], it was shown how an automata-based approach can be transformed mechanically into a tableau-based decision procedure: as a consequence, we only need to "hand-craft" the automata-based algorithm, and then get both a (possibly optimal) worst-case upper bound and a (possibly practicable) tableau-based algorithm for free.

5.2 Modal resolution

In the late 1980s and early 1990s various direct resolution methods for modal logics have been investigated [1, 10, 33, 46, 59, 61, 72, 79, 126, 139, 140]. According to [139] a *resolution method* for a logic L is determined by specifying (i) a class of formulae called clauses, (ii) a reduction method which allows us to transform any formula of L into a finite set of clauses, (iii) a calculus consisting of a set of resolution rules for deriving clauses (and possibly redundancy elimination and simplification rules), and (iv) a derivation process which starts from an initial set of clauses and constructs a sequence of derivable clauses. One can then define a *modal resolution method* to be a resolution method in which clauses are formulae of the modal logic L under consideration. This definition excludes methods which do not use a clausal form from the outset, e.g. destructive modal resolution [72], or methods which use auxiliary labels, e.g. prefixed resolution [6] and labelled modal resolution [7]. Methods which use additional modal operators like the resolution calculus for temporal logics of knowledge presented in [54] can be considered to be borderline cases.

In the following we focus on the modal resolution method of [59] but follow the presentation in the survey paper [64], where a more complete overview of various direct resolution methods and other methods can be found.

A modal formula of $\mathbf{K}$ is in *disjunctive normal form* iff it is a (possibly empty) disjunction of the form $\bigvee L_i \vee \bigvee \Box D_j \vee \bigvee \Diamond A_k$ where each L_i is a propositional literal, each D_j is a modal formula in disjunctive normal form, and each A_k is a modal formula in conjunctive normal form. A modal formula is in *conjunctive normal form* iff it is a conjunction $\bigwedge D_l$ where each

Axioms			
axiom1:	$p, \neg p \Rightarrow \bot$	axiom2:	$\bot, A \Rightarrow \bot$

Resolution rules		
$\vee$-rule1:	$A \vee D, B \vee D' \Rightarrow C \vee D \vee D'$	if $A, B \Rightarrow C$
$\vee$-rule2:	$A \vee C \Rightarrow B \vee C$	if $A \Rightarrow B$
$\Diamond$-rule1:	$\Diamond(A, B, N) \Rightarrow \Diamond(A, B, C, N)$	if $A, B \Rightarrow C$
$\Diamond$-rule2:	$\Diamond(A, N) \Rightarrow \Diamond(B, A, N)$	if $A \Rightarrow B$
K-rule1:	$\Box A, \Diamond(B, N) \Rightarrow \Diamond(B, C, N)$	if $A, B \Rightarrow C$
K-rule2:	$\Box A, \Box B \Rightarrow \Box C$	if $A, B \Rightarrow C$
$\Box$-rule:	$\Box A \Rightarrow \Box B$	if $A \Rightarrow B$

Simplification rules			
$\vee$-simp1:	$\bot \vee D \rightarrow D$	$\Diamond$-simp:	$\Diamond\bot \rightarrow \bot$
$\vee$-simp2:	$A \vee A \vee D \rightarrow A \vee D$	$\wedge$-simp:	$\bot, N \rightarrow \bot$

Figure 15. Modal resolution rules of [59] for $\mathbf{K}$. (The symbols A, B, C, D, D' denote clauses, N denotes a set of clauses, and (A, N) denotes the union of $\{A\}$ and N. No distinction is made between a set N of clauses and the conjunction of its elements.)

D_l is a modal formula in disjunctive normal form. A formula in disjunctive normal form is also called a *(modal) clause*. Any modal formula φ can be transformed into an equivalent formula in conjunctive normal form $\mathrm{cnf}(\varphi)$. In the following, we do not distinguish between a conjunction of clauses and a set of clauses.

The calculus $C_{\mathbf{K}}$ of [59] is given by the set of axioms, resolution rules, and simplification rules shown in Figure 15. The intended meaning of $A, B \Rightarrow C$ and $A \Rightarrow C$ is that the conjunction of the formulae on the left-hand side of $\Rightarrow$ implies the formula on its right-hand side. In contrast, the meaning of $A, B \rightarrow C$ is that occurrences of A and B in a conjunction can be simplified to, that is, replaced by, C. Analogously, $A \rightarrow C$, means that occurrences of A can be replaced by C. Every formula A has a unique normal form $\mathrm{nf}(A)$ under the simplification rules of Figure 15 (modulo commutativity and associativity of $\vee$ and $\wedge$).

Various extensions of $\mathbf{K}$ have been considered, including extensions by the axiom schemas $\mathbf{D}$, $\mathbf{T}$, and $\mathbf{4}$. For each of these axiom schemas the calculus $C_{\mathbf{K}}$ needs to be extended with additional rules: for $\mathbf{D}$ with $\Box\bot \Rightarrow \bot$, for $\mathbf{T}$ with $\Box A, B \Rightarrow C$ if $A, B \Rightarrow C$, while for $\mathbf{4}$ with the two rules $\Box A, \Box B \Rightarrow \Box C$ if $\Box A, B \Rightarrow C$ and $\Box A, \Diamond(B, N) \Rightarrow \Diamond(B, C, N)$ if $\Box A, B \Rightarrow C$. We denote the calculi obtained by adding these rules to $C_{\mathbf{K}}$ by $C_{\mathbf{KD}}$, $C_{\mathbf{KT}}$, and $C_{\mathbf{K4}}$, respectively.

Let L be one of $\mathbf{K}$, $\mathbf{KD}$, $\mathbf{KT}$, $\mathbf{K4}$. Given sets of clauses N and (C, N) we say (C, N) can be *derived in one step* from N in C_L iff either there are clauses A and B in N such that $A, B \Rightarrow C'$ in C_L or there is a clause A in N such that $A \Rightarrow C'$ in C_L, and $C = \mathrm{nf}(C')$ in C_L. A *derivation* of N' from N in C_L is a sequence $N = N_0, N_1, \ldots, N_n = N'$ such that for every i, $0 \leq i < n$, N_{i+1} can be derived from N_i in one step. A *refutation* of N in C_L is a derivation of $\bot$ from N in C_L. If a refutation of N exists, then N is C_L*-refutable*.

In [59] it is shown that a modal formula φ is valid in L iff $\mathrm{cnf}(\neg\varphi)$ is C_L-refutable. This soundness and completeness result is shown in [10] to also hold for a number of refinements of this modal resolution method and the extension by subsumption deletion.

So far, little work seems to have been conducted on devising specialised and efficient data structures and algorithms for modal resolution methods. Due to the extra structural information

that modal formulae carry, which is reflected in the more complicated clausal form, the data structures and algorithms developed for efficient propositional and first-order resolution provers cannot be utilised easily to implement modal resolution methods.

5.3 Sequent-based approaches

Sequent calculi were introduced by Gentzen [80] as a tool for studying natural deduction. The central property of sequent calculi is cut elimination which usually yields consistency as an easy corollary. The first sequent calculi and cut elimination results for modal logics have been established in the early fifties [39], see [89] for further historic references.

A *sequent* is a structure of the form $\Gamma \vdash \Delta$, where Γ and Δ are (finite) lists, multisets, or sets of formulae; Δ is also quite often restricted to be a singleton set or the empty set. A *sequent calculus* for a logic L consists of two parts: (i) a finite set of axioms, (ii) a finite set of rules of the form $\frac{S_1}{S}$ or $\frac{S_1\ S_2}{S}$ with conclusion S and premises S_1 and S_2, where S, S_1, and S_2 denote sequents. The rules can usually be divided into two major groups: *logical rules*, which introduce a new logical formula either on the left or on the right of the turnstile $\vdash$, and *structural rules*, which operate on the structure of the sequents. Of particular interest, both from a proof-theoretical and a computational point of view is the *cut rule*, a rule of the form

$$\frac{\Gamma_1 \vdash \Delta_1, A \qquad A, \Gamma_2 \vdash \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}$$

where, in general, A is an arbitrary formula, called the *cut formula*. A *sequent calculus proof* of a *goal sequent* S is a tree whose nodes are labelled with sequents, such that (i) the root of the tree is labelled with S, (ii) each leaf node is an instance of an axiom of the calculus, and (iii) each sequent labelling a non-leaf node n follows by one of the rules of the calculus from the sequents labelling the children of n. This notion of a proof does not prescribe a particular approach to the construction of the proof of a sequent S. However, it is quite natural to proceed by *backward reasoning*, that is, to start with a tree consisting only of the root node labelled with S and to apply rules from bottom to top, taking the sequent labelling a node of the tree to be the conclusion of a rule and adding children to the tree labelled with the premises of the rule. The construction is complete if all the current leaf nodes are labelled with instances of axioms. In contrast, in *forward reasoning* one would start with one or more leaf nodes labelled with instances of axioms and build the tree toward its root node labelled with S. This approach is basically taken in the *inverse method*, see Section 5.4. For further details on sequent calculi see Sections 7 and 8 of Chapter 2 of this handbook.

From a computational point of view, sequent calculi pose several challenges and also provide insights that can help to improve systems based on tableau calculi or the inverse method.

First, the cut rule is problematic for backward reasoning, since we can choose an arbitrary formula to be the cut formula. We can try to show that we can restrict ourselves to cut formulae which are subformulae of formulae in the goal sequent S while retaining completeness of the calculus. The result would be a calculus with *analytic cut*. However, from a practical point of view, while for analytic cuts we can only choose finitely many different cut formulae, the search space may still be too large. Alternatively, one can try to show that for any sequent S there exists a proof without any application of the cut rule. In such a case, we can omit the cut rule from the calculus and obtain a *cut-free sequent calculus*. While for some modal logics it is rather straightforward to devise cut-free sequent calculi, for others it is much more challenging, for example, for $\mathbf{S5}$ [30, 151, 152], and for some it is an open problem, for example, for $\mathbf{PDL}$

and converse $\mathbf{PDL}$ [127]. We are also not aware of cut-free systems for modal logics with the common knowledge operator.

Second, in the presence of the axiom schema $\mathbf{4}$, systems based on sequent calculi face the same non-termination problems as systems based on tableau calculi. Recall from Section 7 of Chapter 2 one of the additional rules required for $\mathbf{K4}$,

$$\frac{\Box Y_1, \ldots, \Box Y_m, Y_1, \ldots, Y_m, A \vdash \Diamond Z_1, \ldots, \Diamond Z_n, Z_1, \ldots, Z_n}{\Gamma, \Box Y_1, \ldots, \Box Y_m, \Diamond A \vdash \Diamond Z_1, \ldots, \Diamond Z_n, \Delta}$$

where Γ and Δ are sequences of formulae not containing a $\Box$-formula and $\Diamond$-formula, respectively. Here, the premise is not necessarily 'simpler' than the conclusion which can lead to situations in which this rule can be applied infinitely many times when using backward reasoning. To ensure termination a form of *loop-check* has to be used, that is, a check which detects whenever the 'same' sequent occurs twice on a branch of a proof. If in turn we would like to formulate the rules of our calculus in such a way that the applicability of rules does not depend on information about the whole branch of a proof or even the whole proof, additional *history information* has to accompany each sequent in a proof. What minimal history information for loop-checks is necessary to ensure termination on a variety of modal logics, including $\mathbf{KT}$ and $\mathbf{S4}$, is investigated in [96, 98]. These results transfer directly to tableau calculi.

Finally, sequent-based systems face the same problems as tableau-based systems when trying to prove formulae involving disjunctions on the left or conjunctions on the right of the turnstile. Naturally, similar solutions as presented in Section 4, in particular, simplification and forms of intelligent backtracking, have also been considered in the context of sequent-based systems, most notably in the work of [25, 96, 97].

5.4 Inverse method

The inverse method is a variant of the sequent calculus [51, 135] which carries its name because it works from sub-goals to goals, whereas standard sequent-based approaches work in the other direction. For example, if it has already been proven that ϕ is false and ψ is true, then the inverse method will deduce from this that $\phi \rightarrow \psi$ is true. For this kind of forward reasoning to work, we need to be able to focus on an acceptably small set of axioms, and an acceptably small set of goals and sub-goals. For many modal logics, we can restrict our attention to such 'acceptably small' sets of formulae since they enjoy the sub-formula property, i.e. every valid formula ϕ has a derivation in which only (negated or unnegated) sub-formulae of ϕ occur [51]. Calculi for modal logics using the inverse method have been developed in [139, 140, 190]. The inverse method has been shown to be suitable for efficient modal logic theorem proving and is amenable to optimisations [190].

Interestingly, the inverse method is closely related to automata-based approaches [13]. More precisely, the algorithm that decides emptiness of automata (the problem to which satisfiability of a variety of modal logics can be reduced, see Section 5.1) can be viewed as being a notational variant of the inverse method. Both start with propositional axioms (in the automata emptiness test, these correspond to unreachable states), and saturate these axioms using basically the same deduction rules. As a consequence, it should be possible to translate a variety of automata-based decision procedures into the inverse method, thus obtaining an efficient implementation (or a good starting point for its implementation) basically for free.

6 OTHER REASONING PROBLEMS

In this chapter, we have focused on one specific reasoning problem, satisfiability or, dually, validity. There are, however, other interesting reasoning problems for modal logics that are useful for certain applications. We will discuss some of them in this section and we refer the reader to Section 5 of Chapter 13 of this handbook for reasoning problems that are motivated by applications of description logics.

6.1 Model checking

Model checking is the problem of deciding whether $\mathfrak{M}, w \models \varphi$ for a given Kripke structure $\mathfrak{M}$, a world w, and a modal formula φ. It is used for system verification, e.g. to verify a piece of soft- or hardware, as follows:

- $\mathfrak{M}$ represents the system: worlds are viewed as *states* the systems can be in,
- modal parameters represent actions which take the system from one state into another (or several others),
- w is some initial state, and
- φ is a temporal logic formula describing a desired behaviour of the system.

Whereas satisfiability algorithms have to reason w.r.t. *all* structures (possibly from a given class), model checking is concerned with a single structure, and thus quite different: model checking is often less complex than satisfiability, and there are industrial strength implementations of model checking algorithms capable of handling large systems and formulae from rather expressive logics. We refer the interested reader to Chapter 17 of this handbook and [34, 35].

6.2 Proof checking

Proof checking is the problem of deciding whether a given derivation P is a proof of a given formula φ, commonly with respect to a fixed calculus C for a logic L. It requires a language in which we are able to formalise derivations. The formalisation of a derivation may simply be a sequence or a tree-structure of formulae, but may also contain additional information about which and how inference rules of the calculus C have been used in each step of the derivation.

The motivation for proof checking is the fact that advanced theorem proving systems are rarely verified. Thus, like any other piece of software they invariably include errors which can lead the system to provide incorrect answers, including, providing an incorrect proof P for a given formula φ. Simplifying theorem proving systems to an extend that would allow their verification in all likelihood results in systems which are too slow to be useful. However, such system may still be sufficiently powerful to check the correctness of a given derivation P. Thus, a natural approach is to use a highly optimised but unverified system to find a proof P for a given formula φ which is then independently checked for correctness by a slower, verified system.

Proof checking has received considerable attention in the context of higher-order logic [155, 195] and is taken seriously in the context of first-order logic [136]. However, we are not aware of any work in this direction in the context of modal logics, although the problem of incorrect theorem provers also exists in this field. Note that in the context of the translation approach we can rely on first-order proof checkers augmented with a verified program for translating modal formulae into first-order clause sets.

Even more complex is the problem of verifying the non-existence of a proof. For the modal logics we have considered in this chapter we would also expect decision procedures to correctly determine in finite time that a given formula φ has no proof. A justification for this can be given by a model or representation of a model $\mathfrak{M}$ for $\neg\varphi$, produced by the decision procedure. A verified model checker could then be used to independently verify that $\mathfrak{M}$ is indeed a model of $\neg\varphi$.

6.3 Computing correspondences

Recall from Chapter 1 of this handbook the notion of a modal formula $\varphi(p_1, \ldots, p_n)$ over propositional variables $p_1, \ldots, p_n$ being *true in frame* $\mathfrak{F}$ iff for every world w and every valuation mapping V for its propositional variables we have $(\mathfrak{F}, V), w \models \varphi$, and the notion of a modal formula φ defining a class of frames iff φ is true in precisely the frames in the class. It straightforward to see that a modal formula φ over $p_1, \ldots, p_n$ is true in a frame $\mathfrak{F}$ iff the monadic second-order formula $\forall P_{p_1} \ldots P_{p_n} \forall x\, \pi_r(\varphi, x)$ is true in the class of all models over the frame $\mathfrak{F}$. There are methods for reducing such second-order formulae to equivalent first-order formulae and there are methods for reducing the second-order logic formulation of modal axioms to the corresponding frame properties. Computing the first-order equivalents of modal formulae (if they exists) amounts to the elimination of the universal or existential monadic second-order quantifiers. For example, if we are interested in establishing the *relational frame properties* corresponding to a modal formula φ, then we either have to eliminate the universal monadic second-order quantifiers from $\forall P_{p_1} \ldots P_{p_n} \forall x\, \pi_r(\varphi, x)$, or, equivalently, the existential monadic second-order quantifiers from $\exists P_{p_1} \ldots P_{p_n} \exists x\, \pi_r(\neg\varphi, x)$. There can be no algorithm which is guaranteed to find a first-order equivalent formula if there exists one. Still, a number of automated algorithms are known which provide a partial solution to the quantifier elimination problem, namely SCAN [75, 58], DLS [55, 185] and SQEMA [37]. SCAN and DLS are based on a form of resolution while SQEMA can be viewed as a modalized DLS algorithm. Here we briefly review the SCAN algorithm, but more details of DLS and other quantifier elimination algorithms can be found in [36, 147].

The SCAN algorithm involves three stages:

(i) transformation to clausal form and (inner) Skolemisation;

(ii) C-resolution;

(iii) reverse Skolemisation (unskolemisation).

The input of SCAN is a second-order formula of the form $\exists Q_1 \ldots \exists Q_k\, \psi$, where the Q_i are unary predicate variables and ψ is a first-order formula. In the first stage SCAN converts ψ into clausal normal form by transformation into conjunctive normal form, Skolemisation, and clausifying the Skolemised formula. In the second stage SCAN performs a special kind of constraint resolution, called *C-resolution*, the two main inference rules are given in Figure 16. It generates all and only resolvents and factors with the second-order variables that are to be eliminated, which in the case of computing frame correspondence properties includes all existentially quantified second-order variables. When all C-resolvents and C-factors with respect to a particular Q_i-literal and the rest of the clause set have been generated, purity deletion removes all clauses in which this literal occurs. The subsumption deletion rule is optional for the sake of soundness, but helps simplify clause sets in the derivation.

If the C-resolution stage terminates, it yields a set N of clauses in which the specified second-order variables are eliminated. This set is satisfiability equivalent to the original second-order

formula. If no clauses remain after purity deletion, then the original formula is a tautology; if C-resolution produces the empty clause, then it is unsatisfiable. If N is non-empty, finite and does not contain the empty clause, then in the third stage, SCAN attempts to restore the quantifiers from the Skolem functions by reversing Skolemisation. This is not always possible, for instance if the input formula is not first-order definable.

If the input formula is not first-order definable and stage two terminates successfully yielding a non-empty set not containing the empty clause then SCAN produces equivalent second-order formulae in which the specified second-order variables are eliminated but quantifiers involving Skolem functions occur and the reverse Skolemisation typically produces Henkin quantifiers. If SCAN terminates and reverse Skolemisation is successful, then the result is a first-order formula logically equivalent to the second-order input formula.

SCAN can compute the frame correspondence properties for very many well-known axioms including $\mathbf{T}$, $\mathbf{4}$, and $\mathbf{5}$. Recent work has in fact shown that the SCAN algorithm is complete for the class of all Sahlqvist formulae, in the sense that, when given a Sahlqvist formula it will successfully compute an equivalent first-order formula for it [88].

6.4 Model generation

A problem closely related to the satisfiability problem is the problem of generating (counter-) models. Ideally we want to construct finite models if they exist. It is possible to use both tableau and resolution methods to prove that logics have the finite model property and also to give procedures for constructing standard Kripke models.

Although tableau provers do not always output models, it is well-known that tableau procedures implicitly generate models (of some kind) for satisfiable input problems. This is especially true for semantic tableau procedures which are defined by structural rules and use explicit accessibility relations. Modal tableau procedures of the kind described in Section 4 which use propagation rules for handling the additional axioms do construct models but often they are just skeleton models which need to be completed with respect to the relational correspondence properties and then give standard Kripke models.

In first-order logic it is well-known that hyperresolution like tableau methods can be employed both as a reasoning method and a Herbrand model builder [31, 65]. It has been shown that the methods using R^{hyp} and the relational translation described in Section 3 require hardly any extra effort to construct a modal model [49, 121, 180]. It is usually a simple matter to read off a Kripke model from the saturated set of ground unit clauses which represents a Herbrand model. In general this set will be infinite in the limit, but when R^{hyp} is a decision procedure then the set is finitely bounded and consequently a finite Kripke model can be defined.

In more detail, a *Herbrand interpretation* is a set of ground atoms. By definition a ground atom A is *true* in an interpretation H iff $A \in H$ and it is *false* in H iff $A \notin H$. Now, extend the

C-Resolution: $$\frac{C \vee Q(s_1, \ldots, s_n) \quad \neg Q(t_1, \ldots, t_n) \vee D}{C \vee D \vee s_1 \not\approx t_1 \vee \ldots \vee s_n \not\approx t_n}$$

provided the two premises have no variables in common and are distinct clauses

C-Factoring: $$\frac{C \vee Q(s_1, \ldots, s_n) \vee Q(t_1, \ldots, t_n)}{C \vee Q(s_1, \ldots, s_n) \vee s_1 \not\approx t_1 \vee \ldots \vee s_n \not\approx t_n}$$

Figure 16. The calculus of SCAN

definition as expected to the Boolean combination of ground atoms. A clause C is true in H iff for all ground substitutions σ there is a literal L in $C\sigma$ which is true in H. A set N of clauses is true in H iff all clauses in N are true in H. If a set N of clauses is true in an interpretation H then H is referred to as a *Herbrand model* of N. It is proved in [49, 121] that the combination of the relational translation and $\mathsf{R}^{\mathsf{hyp}}$ can be used as a finite Herbrand model generator for the modal logics $\mathbf{K}_n$, $\mathbf{K}_n^{\smile}$ and the extensions with $\mathbf{T}$, $\mathbf{D}$, $\mathbf{B}$ (actually more general results are proved).

In general Herbrand models are not unique and can be large. Therefore it is useful to have a method for generating minimal Herbrand models. An interpretation H is a *minimal Herbrand model* for a set N of clauses iff H is a Herbrand model of N and for no Herbrand model H' of N, $H' \subset H$ holds. Various approaches to generating minimal Herbrand models with hyperresolution are known [26, 31, 94, 144]. It follows from [31] and investigations of $\mathbf{GF}^-$ and the class $\mathcal{BU}$ in [81, 82] that with a moderate extension of $\mathsf{R}^{\mathsf{hyp}}$, denoted here by $\mathsf{R}^{\mathsf{hyp}}_{\mathsf{min}}$, it is possible to guarantee the generation of all and only minimal Herbrand models for any modal and description logic reducible to a decidable class of range restricted clauses. It is necessary to use a depth-first strategy, a complement splitting rule should be used so that the first model generated is a minimal Herbrand model, and a model constraint propagation rule is necessary to prevent the generation of non-minimal Herbrand models (see Figure 17). The procedure $\mathsf{R}^{\mathsf{hyp}}_{\mathsf{min}}$ is generally sound and complete and is a minimal Herbrand model building procedure for range-restricted clauses [31]. An alternative is to use the generalisation [81, 82] of an approach of [144].

It is not difficult to see that model generation procedures and the mentioned minimal Herbrand model generation procedures can be developed by using hyperresolution and the other translation methods. Because of the close connection to tableau, corresponding tableau procedures can be defined and all results carry over to the tableau setting (see [49, 121]).

6.5 Bisimulation

Chapter 1 of this handbook has introduced the notion of a bisimulation between two Kripke models. A *bisimulation* between models $\mathfrak{M} = \langle W, R, V\rangle$ and $\mathfrak{M}' = \langle W', R', V'\rangle$ is a binary relation $E \subseteq W \times W'$ such that whenever $E(w, w')$ the following three properties hold:

Atomic: for all propositional variables p, $w \in V(p)$ iff $w \in V'(p)$;

Zig: if $R_i(w, v)$ for some i, then there exists v' in $\mathfrak{M}'$ such that $E(v, v')$ and $R'_i(w', v')$; and

Zag: if $R'_i(w', v')$ for some i, then there exists v in $\mathfrak{M}$ such that $E(v, v')$ and $R_i(w, v)$.

Complement splitting:
$$\frac{N \cup \{C \vee D\}}{N \cup \{C, \neg D\} \quad | \quad N \cup \{D\}}$$
where D is a ground clause.

Model constraint propagation:
$$\frac{N}{N \cup \{\neg A_1 \vee \ldots \vee \neg A_n\}}$$
where $\{A_1, \ldots, A_n\}$ is the finite Herbrand model of an open branch which is complete with respect to $\mathsf{R}^{\mathsf{hyp}}$. The model constraint propagation rule extends all branches in the derivation tree (to the right) which are not complete with respect to $\mathsf{R}^{\mathsf{hyp}}_{\mathsf{min}}$.

Figure 17. Additional rules for minimal Herbrand model generation

One important property of bisimilar models is that they satisfy the same μ-calculus formulae, that is, let $E(w, w')$ hold then a μ-calculus formula φ is true at w in $\mathfrak{M}$ iff φ is true at w' in $\mathfrak{M}'$. The notion of bisimulation not only plays an important rôle in modal logic, as an equivalence principle between Kripke models, but also in other fields, for example, concurrency theory, set theory, and formal verification. An algorithm for 'on the fly' verification of bisimulations is presented in [66].

A related problem is that of *bisimulation minimisation*, that is, the problem of finding the minimal Kripke model bisimilar to a given Kripke model. In particular, in the context of formal verification by model checking (see Section 6.1 and Chapter 17 of this handbook for further details), bisimulation minimisation provides an easily and automatically computable way to reduce the number of states of a model while preserving the truth and falsehood of the formulae that hold in it.

Let $\mathfrak{M} = \langle W, R, V\rangle$ be a Kripke model and E be an equivalence relation on W. Let $[w]_E$ denote the equivalence class of a world $w \in W$ with respect to E. The set of all equivalence classes is a *partition* of W and $\mathfrak{M}$. The bisimulation minimisation of a Kripke model $\mathfrak{M}$ is the quotient $\mathfrak{M}/E = \langle W', R', V'\rangle$ where

$$\begin{array}{rcl} W' & = & \{[w]_E \mid w \in W\}, \\ R' & = & \{([w]_E, [w']_E) \mid w, w' \in W \wedge R'(w, w')\}, \text{ and} \\ V'(p) & = & \{[w]_E \mid w \in V(p)\} \end{array}$$

for every propositional variable p such that E is the maximal equivalence relation on W which is also a bisimulation between $\mathfrak{M}$ and itself. A partition P is *stable* with respect to E iff for each pair $[w]_E$, $[w']_E$ of equivalence classes with respect to E either $[w]_E \subseteq E^{-1}([w']_E)$ or $[w]_E \cap E^{-1}([w']_E) = \emptyset$.

In the computation of the bisimulation minimisation of a Kripke model we can basically follow two strategies. One is a negative strategy in which we start with the coarsest partition P such that $E(w, w')$ iff $w \in V(p)$ iff $w' \in V(p)$ for every positional variable p and split classes whenever P is not stable. Another is a positive strategy in which we start with the finest partition P in which each equivalence class consists of a single world and the bisimulation minimisation is constructed via a sequence of steps in which we merge two or more classes. An algorithm following the negative strategy is presented in [156] which has the optimal worst-case running time, namely $O(|R|log|W|)$. An implementation of this algorithm is presented in [67]. Other algorithms following a negative strategy are presented in [28, 130]. They take advantage of the fact that in a number of applications we are only interested in the part of a Kripke model reachable from a designated start world. In this case, equivalence classes associated with unreachable worlds need not be taken into account when considering the stability of an equivalence class associated with a reachable world. An algorithm following the positive strategy is presented in [157]. Recently, [57] has introduced an algorithm combines both the positive and negative strategy by using the algorithms of [156] and [157] as subroutines. For a range of special cases this algorithm terminates in time $O(|R| + |W|)$.

Finally, [71] presents on-the-fly model checkers for invariant properties incorporating the bisimulation minimisation algorithms of [28, 130, 156]. From an empirical comparison they draw the conclusion that in this context an optimised version of the algorithm of [156] performs better than the other two.

6.6 Modal logic programming

The problem of extending logic programming languages with modal operators has received a lot of attention in the late 1980s and early 1990s, at about the same time most of the direct resolution methods mentioned in Section 5.2 were developed and also work on the translation methods described in Section 3 intensified. Consequently, work in this area can again be divided between direct approaches and translation approaches.

Following the direct approach, [21, 20] presents a declarative semantics and an SLD resolution calculus for a class of modal logic programs in modal logics $\mathbf{KD}$, $\mathbf{KT}$, and $\mathbf{S4}$, while [22, 23] present a framework for developing the fixpoint and operational semantics of a class of multi-modal logic programs where additional properties of modal operators can be described by axiom schemas of the form $[i_1][i_2]\cdots[i_m]p \rightarrow [j_1][j_2]\cdots[j_n]p$, so-called inclusion axioms. More recent work includes [142] presenting a fixpoint semantics, least model semantics, and an SLD resolution calculus for modal logic programs in modal logics extending $\mathbf{K}$ with a non-empty selection of the axiom schemas $\mathbf{B}$, $\mathbf{D}$, $\mathbf{T}$, $\mathbf{4}$ and $\mathbf{5}$. Also, modal logic programs in [142] are as expressive as the general modal Horn fragment which allows arbitrary occurrences of the modal operators $\Box$ and $\Diamond$ in programs clauses and goals.

Following the translation approach, [50] applies the functional translation to multi-modal logic programs in the modal logics $\mathbf{KD}$, $\mathbf{KT}$, $\mathbf{KD4}$, $\mathbf{KT4}$, $\mathbf{KF}$ ($\mathbf{F}$ is the functionality axiom), and simple inclusion axioms of the form $[i]p \rightarrow [j]p$. In these logics, the functional translation of goals and program clauses in the general modal Horn fragment are in the first-order Horn fragment. For computations SLD resolution extended by theory unification is used. In [146] presents an application of the *semi-functional translation* [145, 148] to modal logic programs in modal logics $\mathbf{KB}$ and $\mathbf{KDB}$, as well as $\mathbf{KD}$, $\mathbf{KT}$, and their extension by one or both of the axiom schemas $\mathbf{4}$ and $\mathbf{5}$. The semi-functional translation combines features of the relational and functional translation. For modal formulae in negation normal form, subformulae of the form $\Box\varphi$ are translated using the relational translation, while subformulae of the form $\Diamond\varphi$ are translated using the functional translation. A *functional simulator axiom* needs to be added to the translation to link the relational and functional aspects of the translation. The semi-functional translation has the advantage over the functional translation that the frame properties of many modal logics, including the ones listed above, can be specified by simple first-order Horn theories without equality. Consequently, the use of theory unification and theory resolution can be avoided. Furthermore, if the semi-functional translation is applied to goals and program clauses in the general modal Horn fragment, then the resulting first-order clauses are themselves Horn. Together with the fact that the frame properties are expressed by Horn clauses, this implies that unmodified SLD resolution can be used to execute the translated modal logic programs.

The functional and semi-functional translation have been incorporated into MSPASS [120, 174]. Implementations of systems based on the direct approach include MOLOG [62, 63], MProlog [141, 143], and TIM [21]. However, just as for the direct resolution approaches described in Section 5.2, little work seems to have been conducted on developing specialised and efficient data structures and algorithms for such systems, with the exception of [2] which describes an abstract machine model for MOLOG, in analogue to the Warren Abstract Machine model for Prolog [191].

There has also been considerable work on temporal logic programming. For surveys on this work which also cover some of the approaches to modal logic programming mentioned above see [154, 70, 84].

There is currently renewed interest in modal and temporal logic programming in the context of multi-agent system development [24, 53, 69] and related areas.

7 REVIEW AND DISCUSSION

In this chapter we have examined computational approaches to modal logics. Although we have considered a variety of computational approaches and reasoning problems, we have focused on the use of translation-based and tableau-based algorithms for deciding the satisfiability of a formula, both with and without reference to a background theory. This focus was motivated by the dominance of translation-based and tableau-based approaches in implemented systems, and by the importance of satisfiability testing in applications such as the verification of multi-agent systems and ontology engineering.

The reason for the dominance of these two approaches is that they have proved amenable to implementation and optimisation techniques that dramatically improve typical case performance; the use of such techniques is crucial if reasoning systems are to be effective in applications. The applicability and effectiveness of optimisation techniques and refinements is, however, highly dependent on the logic under consideration and on the class of problem being solved. For example, in the context of tableau-based algorithms, caching must be used with care in the presence of converse modalities, and semantic branching search, while highly effective for randomly generated problems, may be ineffective (and perhaps even counter productive) for problems derived from ontology engineering applications. Similarly, in the context of translation-based algorithms, hyperresolution may be the most suitable approach for randomly generated problems in the modal logic $\mathbf{K}_n$, while ordered resolution is more effective for problems derived from ontology engineering applications.

Regarding the two approaches, both have advantages and disadvantages. Tableau-based methods generally require full implementation, but this allows the implementor to choose and fine-tune the optimisations, data structures, and algorithms for effective operation in the intended application. In contrast, no major implementation effort is needed for translation-based methods, but a careful choice of translation, refinement of resolution, and operational parameters is required to guarantee termination and effectiveness of the first-order logic prover on the class of problems being solved. The choice of approach may ultimately depend on the logic in question: tableau-based methods seem to have some advantages in the presence of graded modalities (counting), for example, whereas translation-based methods can handle and may be better for boolean modal logics (role negation). Currently, tableau-based approaches are the most widely used in ontology applications, with description logic systems such as FaCT++, Racer and Pellet [159, 90, 160]. In contrast, translation-based methods have a number of other uses, for example, computing correspondence properties and modal logic programming.

The use of implemented systems in realistic applications brings with it new challenges, both with respect to the expressive power of the logics being used, and the size and complexity of the problems to be solved. The W3C standard ontology language OWL, for example, corresponds to a logic with transitive, converse and graded modalities, as well as nominals, and ontology applications may call for reasoning with respect to very large background theories. For the logic corresponding to OWL, a tableau-based algorithm has only recently been introduced [110], a translation-based algorithm using the basic superposition calculus is still under development [112], and the development of computational and optimisation techniques is the subject of considerable ongoing research. Similarly, as mentioned in the introduction, agent frameworks consist of complex multi-modal logics, typically including a dynamic component, allowing the representation of dynamic activity via a temporal or a dynamic logic. Ongoing research is focusing on developing advanced computational methods and optimisation techniques for such frameworks.

BIBLIOGRAPHY

[1] M. Abadi and Z. Manna. Modal theorem proving. In J. H. Siekmann, editor, *Proceedings of the 8th International Conference on Automated Deduction (CADE-8)*, volume 230 of *Lecture Notes in Computer Science*, pages 172–189. Springer, 1986.

[2] J.-M. Alliot, A. Herzig, and M. Lima Marques. Implementing Prolog extensions: a parallel inference machine. In *Proceedings of the International Conference on Fifth Generation Computer Systems '92*, pages 833–842. IOS Press, 1992.

[3] H. Andréka, I. Németi, and J. van Benthem. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27(3):217–274, 1998.

[4] C. Areces. *Logic Engineering. The Case of Description and Hybrid Logics*. PhD thesis, ILLC, University of Amsterdam, 2000.

[5] C. Areces, P. Blackburn, and M. Marx. A road-map on complexity for hybrid logics. In *Annual Conference of the European Association for Computer Science Logic (CSL '99)*, volume 1683 of *Lecture Notes in Computer Science*, pages 307–321. Springer, 1999.

[6] C. Areces, H. De Nivelle, and M. de Rijke. Prefixed resolution: A resolution method for modal and description logics. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, volume 1632 of *Lecture Notes in Artificial Intelligence*, pages 187–201. Springer, 1999.

[7] C. Areces, M. de Rijke, and H. De Nivelle. Resolution in modal, description and hybrid logic. *Journal of Logic and Computation*, 11(5):717–736, 2001.

[8] C. Areces, R. Gennari, J. Heguiabehere, and M. de Rijke. A simple ordering for deciding modal logic. Forthcoming.

[9] C. Areces, R. Gennari, J. Heguiabehere, and M. de Rijke. Tree-based heuristic in modal theorem proving. In W. Horn, editor, *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI 2000)*, pages 199–203. IOS Press, 2000.

[10] Y. Auffray, P. Enjalbert, and J.-J. Hebrard. Strategies for modal resolution: Results and problems. *Journal of Automated Reasoning*, 6:1–38, 1990.

[11] F. Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. Technical Report RR-90-13, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Kaiserslautern, Germany, 1990. An abridged version appeared in *Proceedings of IJCAI-91*, pp. 446–451.

[12] F. Baader, J. Hladik, C. Lutz, and F. Wolter. From tableaux to automata for description logics. *Fundamenta Informaticae*, 57(2-4):247–279, 2003.

[13] F. Baader and S. Tobies. The inverse method implements the automata approach for modal satisfiability. In *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR-01)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 92–106. Springer, 2001.

[14] L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

[15] L. Bachmair and H. Ganzinger. Ordered chaining calculi for first-order theories of transitive relations. *Journal of the ACM*, 45(6):1007–1049, 1998.

[16] L. Bachmair and H. Ganzinger. Resolution theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 2, pages 19–99. Elsevier, 2001.

[17] L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic paramodulation. *Information and Computation*, 121(2):172–192, 1995.

[18] L. Bachmair, H. Ganzinger, and U. Waldmann. Superposition with simplification as a decision procedure for the monadic class with equality. In G. Gottlob, A. Leitsch, and D. Mundici, editors, *Proceedings of the Third Kurt Gödel Colloquium (KGC '93)*, volume 713 of *Lecture Notes in Computer Science*, pages 83–96. Springer, 1993.

[19] A. B. Baker. *Intelligent Backtracking on Constraint Satisfaction Problems: Experimental and Theoretical Results*. PhD thesis, University of Oregon, 1995.

[20] P. Balbiani, L. Farinas del Cerro, and A. Herzig. Declarative semantics for modal logic programs. In *Proceedings of the 1988 International Conference on Fifth Generation Computer Systems*, pages 507–514. ICOT, 1988.

[21] P. Balbiani, A. Herzig, and M. Lima Marques. TIM: The Toulouse Inference Machine for non-classical logic programming. In H. Boley and M. M. Richter, editors, *Proceedings of the International Workshop on Processing Declarative Knowledge (PDK '91)*, volume 567 of *Lecture Notes in Artificial Intelligence*, pages 366–382. Springer, 1991.

[22] M. Baldoni. *Normal Multimodal Logics: Automatic Deduction and Logic Programming Extension*. PhD thesis, Dipartimento di Informatica, Universit degli Studi di Torino, Torino, Italy, April 1998.

[23] M. Baldoni, L. Giordano, and A. Martelli. A framework for a modal logic programming. In M. J. Maher, editor, *Joint International Conference and Symposium on Logic Programming*, pages 52–66. MIT Press, 1996.

[24] M. Baldoni, A. Martelli, V. Patti, and L. Giordano. Programming rational agents in a modal action logic. *Annals of Mathematics and Artificial Intelligence*, 41(2-4):207–257, 2004.

[25] P. Balsiger, A. Heuerding, and S. Schwendimann. Logics Workbench 1.0. In H. de Swart, editor, *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX '98)*, volume 1397 of *Lecture Notes in Artificial Intelligence*, pages 35–35. Springer, 1998.

[26] P. Baumgartner, J. D. Horton, and B. Spencer. Merge path improvements for minimal model hyper tableaux. In N. V. Murray, editor, *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX '99)*, volume 1617 of *Lecture Notes in Artificial Intelligence*, pages 51–65. Springer, 1999.

[27] P. Baumgartner and R. A. Schmidt. Improved bottom-up model generation. In U. Furbach and N. Shankar, editors, *Proceedings of the Third International Joint Conference on Automated Reasoning (IJCAR 2006)*, Lecture Notes in Artificial Intelligence. Springer, 2006. To appear.

[28] A. Bouajjani, J.-C. Fernandez, and N. Halbwachs. Minimal model generation. In E. M. Clarke and R. P. Kurshan, editors, *Proceedings of the 2nd International Workshop on Computer Aided Verification (CAV '90)*, volume 531 of *Lecture Notes in Computer Science*, pages 197–203. Springer, 1991.

[29] T. Boy de la Tour. An optimality result for clause form translation. *Journal of Symbolic Computation*, 14:283–301, 1992.

[30] T. Braüner. A cut-free Gentzen formulation of the modal logic S5. *Logic Journal of the IGPL*, 8(5):629–643, 2000.

[31] F. Bry and A. Yahya. Positive unit hyperresolution tableaux for minimal model generation. *Journal of Automated Reasoning*, 25(1):35–82, 2000.

[32] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Description logic framework for information integration. In A. G. Cohn, L. K. Schubert, and S. C. Shapiro, editors, *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR '98)*, pages 2–13. Morgan Kaufmann, 1998.

[33] M. Cialdea. Resolution for some first order modal systems. *Theoretical Computer Science*, 85:213–229, 1991.

[34] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[35] E. M. Clarke and B.-H. Schlingloff. Model checking. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 24, pages 1635–1790. Elsevier, 2001.

[36] W. Conradie, V. Goranko, and D. Vakarelov. Elementary canonical formulae: a survey on syntactic, algorithmic, and model-theoretic aspects. In R. A. Schmidt, I. Pratt-Hartmann, M. Reynolds, and H. Wansing, editors, *Advances in Modal Logic, Volume 5*, pages 17–51. King's College Publications, London, 2005.

[37] W. Conradie, V. Goranko, and D. Vakarelov. Algorithmic correspondence and completeness in modal logic. i. the core algorithm SQEMA. *Logical Methods in Computer Science*, 2(1:5):1–26, 2006.

[38] S. A. Cook and R. A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, 1979.

[39] H. B. Curry. The elimination theorem when modality is present. *Journal of Symbolic Logic*, 17:249–265, 1952.

[40] M. D'Agostino. Are tableaux an improvement on truth-tables? *Journal of Logic, Language, and Information*, 1:235–252, 1992.

[41] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.

[42] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[43] G. De Giacomo. Eliminating 'converse' from converse PDL. *Journal of Logic, Language and Information*, 5(2):193–208, 1996.

[44] G. De Giacomo and M. Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics (extended abstract). In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, pages 205–212. AAAI Press, 1994.

[45] G. De Giacomo and F. Massacci. Combining deduction and model checking into tableaux and algorithms for converse-pdl. *Information and Computation*, 160(1–2):117–137, 2000.

[46] H. De Nivelle. *Ordering refinements of resolution*. PhD thesis, Technische Universiteit Delft, The Netherlands, 1996.

[47] H. De Nivelle. Splitting through new proposition symbols. In R. Nieuwenhuis and A. Voronkov, editors, *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2001)*, volume 2250 of *Lecture Notes in Artificial Intelligence*, pages 172–185. Springer, 2001.

[48] H. De Nivelle and M. de Rijke. Deciding the guarded fragment by resolution. *Journal of Symbolic Computation*, 35(1):21–58, 2003.

[49] H. De Nivelle, R. A. Schmidt, and U. Hustadt. Resolution-based methods for modal logics. *Logic Journal of the IGPL*, 8(3):265–292, 2000.

[50] F. Debart, P. Enjalbert, and M. Lescot. Multimodal logic programming using equational and order-sorted logic. *Theoretical Computer Science*, 105(1):141–166, 1992.

[51] A. Degtyarev and A. Voronkov. The inverse method. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 21, pages 179–272. Elsevier Science, 2001.

[52] N. Dershowitz and D. A. Plaisted. Rewriting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 9, pages 533–607. Elsevier, 2001.

[53] J. Dix, S. Kraus, and V. S. Subrahmanian. Temporal Agent Reasoning. *Artificial Intelligence*, 127(1):87–135, 2001.

[54] C. Dixon, M. Fisher, and M. Wooldridge. Resolution for temporal logics of knowledge. *Journal of Logic and Computaton*, 8(3):345–372, 1998.

[55] P. Doherty, W. Lukaszewics, and A. Szalas. Computing circumscription revisited: A reduction algorithm. *Journal of Automated Reasoning*, 18(3):297–336, 1997.

[56] F. M. Donini and F. Massacci. Exptime tableaux for $\mathcal{ALC}$. *Artificial Intelligence*, 124(1):87–138, 2000.

[57] A. Dovier, C. Piazza, and A. Policriti. An efficient algorithm for computing bisimulation equivalence. *Theoretical Computer Science*, 311(1-3):221–256, 2004.
[58] T. Engel. Quantifier elimination in second-order predicate logic. Diplomarbeit, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, 1996.
[59] P. Enjalbert and L. Fariñas del Cerro. Modal resolution in clausal form. *Theoretical Computer Science*, 65(1):1–33, 1989.
[60] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1996.
[61] L. Fariñas del Cerro. A simple deduction method for modal logic. *Information Processing Letters*, 14(2):49–51, April 1982.
[62] L. Fariñas del Cerro. MOLOG: A system that extends PROLOG with modal logic. *New Generation Computing*, 4:35–50, 1986.
[63] L. Fariñas del Cerro and A. Herzig. MOLOG. `http://www.irit.fr/ACTIVITES/EQ_ALG/Herzig/molog.html`.
[64] L. Fariñas del Cerro and A. Herzig. Modal deduction with applications in epistemic and temporal logics. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming: Epistemic and Temporal Reasoning*, volume 4, pages 499–594. Clarendon Press, Oxford, 1995.
[65] C. Fermüller, A. Leitsch, U. Hustadt, and T. Tammet. Resolution decision procedures. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 25, pages 1791–1849. Elsevier, 2001.
[66] J. Fernandez and L. Mounier. Verification bisimulations 'on the fly'. In J. Quemada, J. A. Mañas, and E. Vázquez, editors, *In Proceedings of the Third International Conference on Formal Description Techniques (FORTE '90)*, pages 95–110. North-Holland, 1990.
[67] J.-C. Fernandez. An implementation of an efficient algorithm for bisimulation equivalence. *Science of Computer Programming*, 13(2–3):219–236, 1990.
[68] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Science*, 18:194–211, 1979.
[69] M. Fisher, C. Ghidini, and B. Hirsch. Organising logic-based agents. In M. G. Hinchey, J. L. Rash, W.t Truszkowski, C. Rouff, and D. F. Gordon-Spears, editors, *Revised Papers of the Second International Workshop on Formal Approaches to Agent-Based Systems (FAABS 2002)*, volume 2699 of *Lecture Notes in Artificial Intelligence*, pages 15–27. Springer, 2003.
[70] M. Fisher and R. Owens. An introduction to executable modal and temporal logics. In *Proceedings of the IJCAI-93 Workshop on Executable Modal and Temporal Logics*, volume 897 of *Lecture Notes in Artificial Intelligence*, pages 1–20. Springer, 1995.
[71] K. Fisler and M. Y. Vardi. Bisimulation and model checking. In L. Pierre and T. Kropf, editors, *Proceedings of the 10th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME '99)*, volume 1703 of *Lecture Notes in Computer Science*, pages 338–341. Springer, 1999.
[72] M. Fitting. Destructive modal resolution. *Journal of Logic and Computation*, 1(1):83–97, 1990.
[73] E. Franconi and G. Ng. The i.com tool for intelligent conceptual modelling. In *Working Notes of the ECAI2000 Workshop on Knowledge Representation Meets Databases (KRDB2000)*, pages 45–53. CEUR, 2000.
[74] J. W. Freeman. *Improvements to Propositional Satisfiability Search Algorithms*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1995.
[75] D. M. Gabbay and H. J. Ohlbach. Quantifier elimination in second-order predicate logic. *South African Computer Journal*, 7:35–43, 1992. Also published in B. Nebel, C. Rich, W. R. Swartout, editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR '92)*, pages 425–436. Morgan Kaufmann, 1992.
[76] H. Ganzinger and H. De Nivelle. A superposition decision procedure for the guarded fragment with equality. In *Proceedings of the Fourteenth Annual IEEE Symposium on Logic in Computer Science (LICS '99)*, pages 295–303. IEEE Computer Society Press, 1999.
[77] H. Ganzinger, U. Hustadt, C. Meyer, and R. A. Schmidt. A resolution-based decision procedure for extensions of K4. In M. Zakharyaschev, K. Segerberg, M. de Rijke, and H. Wansing, editors, *Advances in Modal Logic, Volume 2*, volume 119 of *Lecture Notes*, chapter 9, pages 225–246. CSLI Publications, Stanford, 2001.
[78] J. Gaschnig. Performance measurement and analysis of certain search algorithms. Technical report CMU-CS-79-124, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA, 1979.
[79] C. Geissler and K. Konolige. A resolution method for quantified modal logics of knowledge and belief. In J. Y. Halpern, editor, *Proceedings of the 1st Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 309–324. Morgan Kaufmann, 1986.
[80] G. Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935. English translation in M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131, North-Holland, 1969.
[81] L. Georgieva, U. Hustadt, and R. A. Schmidt. Computational space efficiency and minimal model generation for guarded formulae. In R. Nieuwenhuis and A. Voronkov, editors, *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2001)*, volume 2250 of *Lecture Notes in Artificial Intelligence*, pages 85–99. Springer, 2001.

[82] L. Georgieva, U. Hustadt, and R. A. Schmidt. A new clausal class decidable by hyperresolution. In A. Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction (CADE-18)*, volume 2392 of *Lecture Notes in Artificial Intelligence*, pages 260–274. Springer, 2002. The long version is Preprint CSPP-18, University of Manchester, UK.

[83] L. Georgieva, U. Hustadt, and R. A. Schmidt. Hyperresolution for guarded formulae. *Journal of Symbolic Computation*, 36(1–2):163–192, 2003.

[84] M. Gergatsoulis. Temporal and modal logic programming languages. In A. Kent and J. G. Williams, editors, *Encyclopedia of Microcomputers*, chapter Volume 27, Supplement 6, pages 393–408. Marcel Dekker, 2001.

[85] E. Giunchiglia, F. Giunchiglia, R. Sebastiani, and A. Tacchella. Sat vs. translation based decision procedures for modal logics: A comparative evaluation. *Journal of Applied Non-Classical Logics*, 10(2):145–172, 2000.

[86] E. Giunchiglia and A. Tacchella. A subset-matching size-bounded cache for satisfiability in modal logics. In R. Dyckhoff, editor, *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2000)*, volume 1847 of *Lecture Notes in Artificial Intelligence*, pages 237–251. Springer, 2000.

[87] F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures - the case study of modal K. In M. McRobbie and J. K. Slaney, editors, *Proceedings of the 13th Conference on Automated Deduction (CADE-13)*, volume 1104 of *Lecture Notes in Artificial Intelligence*, pages 583–597, New Brunswick, NJ, USA, 1996. Springer.

[88] V. Goranko, U. Hustadt, R. Schmidt, and D. Vakarelov. SCAN is complete for all Sahlqvist formulae. In R. Berghammer, B. Möller, and G. Struth, editors, *Revised Selected Papers of the 7th International Seminar on Relational Methods in Computer Science and the 2nd International Workshop on Kleene Algebra*, volume 3051 of *Lecture Notes in Computer Science*, pages 149–162. Springer, 2004.

[89] R. Goré. Tableau methods for modal and temporal logics. In M. D'Agostino, D. Gabbay, R. Hähnle, and J. Posegga, editors, *Handbook of Tableau Methods*, pages 297–396. Kluwer, 1999.

[90] V. Haarslev and R. Möller. Consistency testing: The RACE experience. In R. Dyckhoff, editor, *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2000)*, volume 1847 of *Lecture Notes in Artificial Intelligence*, pages 1–18. Springer, 2000.

[91] Volker Haarslev and Ralf Möller. High performance reasoning with very large knowledge bases: A practical case study. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*, pages 161–168. Morgan Kaufmann, 2001.

[92] R. Hähnle. Tableaux and related methods. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, chapter 3, pages 101–178. Elsevier Science Publishers (North-Holland), Amsterdam, 2001.

[93] J. Y. Halpern and Y. Moses. A guide to completeness and complexity for modal logic of knowledge and belief. *Artificial Intelligence*, 54:319–379, 1992.

[94] R. Hasegawa, H. Fujita, and M. Koshimura. Efficient minimal model generation using branching lemmas. In D. McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction (CADE-17)*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 184–199. Springer, 2000.

[95] A. Herzig. A new decidable fragment of first order logic, June 1990. In Abstracts of the 3rd Logical Biennial, Summer School & Conference in honour of S. C. Kleene, Varna, Bulgaria.

[96] A. Heuerding. *Sequent Calculi for Proof Search in Some Modal Logics*. PhD thesis, Universität Bern, Switzerland, 1996.

[97] A. Heuerding, G. Jäger, S. Schwendimann, and M. Seyfried. The Logics Workbench LWB: A snapshot. *Euromath Bulletin*, 2(1):177–186, 1996.

[98] A. Heuerding, M. Seyfried, and H. Zimmermann. Efficient loop-check for backward proof search in some non-classical propositional logics. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Proceedings of the 5th International Workshop on Theorem Proving with Analytic Tableaux and Related Methods (TABLEAUX '96)*, volume 1071 of *Lecture Notes in Artificial Intelligence*, pages 210–225. Springer, 1996.

[99] J. Hladik. Implementation and optimisation of a tableau algorithm for the guarded fragment. In U. Egly and C. G. Fermüller, editors, *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2002)*, volume 2381 of *Lecture Notes in Artificial Intelligence*, pages 145–159. Springer, 2002.

[100] Jörg Hoffmann and Jana Koehler. A new method to index and query sets. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 462–467. Morgan Kaufmann, 1999.

[101] B. Hollunder and F. Baader. Qualifying number restrictions in concept languages. In *Proceedings of the 2nd International Conference on the Principles of Knowledge Representation and Reasoning (KR '91)*, pages 335–346. Morgan Kaufmann, 1991.

[102] I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.

[103] I. Horrocks. Implementation and optimisation techniques. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 306–346. Cambridge University Press, 2003.

[104] I. Horrocks and P. F. Patel-Schneider. Optimizing description logic subsumption. *Journal of Logic and Computation*, 9(3):267–293, 1999.

[105] I. Horrocks and U. Sattler. A description logic with transitive and inverse roles and role hierarchies. *Journal of Logic and Computation*, 9(3), 1999.

[106] I. Horrocks and U. Sattler. Ontology reasoning in the $\mathcal{SHOQ}(D)$ description logic. In B. Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*, pages 199–204. Morgan Kaufmann, 2001.

[107] I. Horrocks and U. Sattler. Optimised reasoning for $\mathcal{SHIQ}$. In F. van Harmelen, editor, *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI 2002)*, pages 277–281. IOS Press, July 2002.

[108] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proceedings of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR '99)*, volume 1705 of *Lecture Notes in Artificial Intelligence*, pages 161–180. Springer, 1999.

[109] I. Horrocks and S. Tobies. Reasoning with axioms: Theory and practice. In A. G. Cohn, F. Giunchiglia, and B. Selman, editors, *Proceedings of the 7th International Conference on the Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 285–296. Morgan Kaufmann, 2000.

[110] Ian Horrocks and Ulrike Sattler. A tableaux decision procedure for $\mathcal{SHOIQ}$. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pages 448–453, 2005.

[111] U. Hustadt. *Resolution-Based Decision Procedures for Subclasses of First-Order Logic*. PhD thesis, Univ. d. Saarlandes, Saarbrücken, Germany, 1999.

[112] U. Hustadt, B. Motik, and U. Sattler. Reducing $\mathcal{SHIQ}^-$ description logic to disjunctive datalog programs. In D. Dubois, C. Welty, and M.-A. Williams, editors, *Proceedings of the 9th International Conference on Knowledge Representation and Reasoning (KR 2004)*, pages 152–162. AAAI Press, 2004.

[113] U. Hustadt, B. Motik, and U. Sattler. A decomposition rule for decision procedures by resolution-based calculi. In F. Baader and A. Voronkov, editors, *Proceedings of the 11th International Conference on Logic for Programming, Artificial Intelligence (LPAR 2004)*, volume 3452 of *Lecture Notes in Artificial Intelligence*, pages 21–35. Springer, 2005.

[114] U. Hustadt and R. A. Schmidt. On evaluating decision procedures for modal logic. In M. E. Pollack, editor, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 202–207. Morgan Kaufmann, 1997.

[115] U. Hustadt and R. A. Schmidt. Simplification and backjumping in modal tableau. In H. de Swart, editor, *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX '98)*, volume 1397 of *Lecture Notes in Artificial Intelligence*, pages 187–201. Springer, 1998.

[116] U. Hustadt and R. A. Schmidt. An empirical analysis of modal theorem provers. *Journal of Applied Non-Classical Logics*, 9(4):479–522, 1999.

[117] U. Hustadt and R. A. Schmidt. Maslov's class K revisited. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, volume 1632 of *Lecture Notes in Artificial Intelligence*, pages 172–186. Springer, 1999.

[118] U. Hustadt and R. A. Schmidt. On the relation of resolution and tableaux proof systems for description logics. In T. Dean, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 110–115. Morgan Kaufmann, 1999.

[119] U. Hustadt and R. A. Schmidt. Issues of decidability for description logics in the framework of resolution. In R. Caferra and G. Salzer, editors, *Automated Deduction in Classical and Non-Classical Logics*, volume 1761 of *Lecture Notes in Artificial Intelligence*, pages 191–205. Springer, 2000.

[120] U. Hustadt and R. A. Schmidt. MSPASS: Modal reasoning by translation and first-order resolution. In R. Dyckhoff, editor, *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2000)*, volume 1847 of *Lecture Notes in Artificial Intelligence*, pages 67–71. Springer, 2000.

[121] U. Hustadt and R. A. Schmidt. Using resolution for testing modal satisfiability and building models. *Journal of Automated Reasoning*, 28(2):205–232, 2002.

[122] U. Hustadt, R. A. Schmidt, and L. Georgieva. A survey of decidable first-order fragments and description logics. *Journal of Relational Methods in Computer Science*, 1:251–276, 2004.

[123] W. H. Joyner Jr. Resolution strategies as decision procedures. *Journal of the ACM*, 23(3):398–417, 1976.

[124] Y. Kazakov and H. De Nivelle. A resolution decision procedure for the guarded fragment with transitive guards. In D. A. Basin and M. Rusinowitch, editors, *Proceedings of the Second International Joint Conference on Automated Reasoning (IJCAR 2004)*, volume 3097 of *Lecture Notes in Artificial Intelligence*, pages 122–136. Springer, 2004.

[125] Holger Knublauch, Ray Fergerson, Natalya Noy, and Mark Musen. The protégé OWL plugin: An open development environment for semantic web applications. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *Proceedings of the 2004 International Semantic Web Conference (ISWC 2004)*, volume 3298 of *Lecture Notes in Computer Science*, pages 229–243. Springer, 2004.

[126] K. Konolige. Resolution and quantified epistemic logics. In J. H. Siekmann, editor, *Proceedings of the 8th International Conference on Automated Deduction (CADE-8)*, volume 230 of *Lecture Notes in Computer Science*, pages 199–208. Springer, 1986.

[127] M. Kracht. *Tools and Techniques in Modal Logic*, volume 142 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 1999.

[128] O. Kupferman and M.Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proceedings of the 30th ACM SIGACT Symposium on Theory of Computing (STOC-98)*, pages 224–233. ACM Press, 1998.

[129] R. E. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM Journal of Computing*, 6(3):467–480, 1977.

[130] D. Lee and M. Yannakakis. Online minimization of transition systems (extended abstract). In *Proceedings of the 24th Annual ACM symposium on Theory of Computing*, pages 264–274. ACM Press, 1992.

[131] Thorsten Liebig and Olaf Noppens. Ontotrack: Combining browsing and editing with reasoning and explaining for OWL Lite ontologies. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *Proceedings of the 2004 International Semantic Web Conference (ISWC 2004)*, volume 3298 of *Lecture Notes in Computer Science*, pages 244–258. Springer, 2004.

[132] C. Lutz. Complexity of terminological reasoning revisited. In H. Ganzinger, D. A. McAllester, and A. Voronkov, editors, *Proceedings of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR '99)*, volume 1705 of *Lecture Notes in Artificial Intelligence*, pages 181–200. Springer, 1999.

[133] C. Lutz. NExpTime-complete description logics with concrete domains. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR-01)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 45–60. Springer, 2001.

[134] C. Lutz, U. Sattler, and S. Tobies. A suggestion of an n-ary description logic. In P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, and P. Patel-Schneider, editors, *Proceedings of the 1999 International Workshop on Description Logics (DL '99)*, pages 81–85. Linköping University, 1999.

[135] S. Ju. Maslov. The inverse method for establishing deducibility for logical calculi. In V. P. Orevkov, editor, *The Calculi of Symbolic Logic I: Proc. of the Steklov Institute of Mathematics edited by I. G. Petrovskiĭ and S. M. Nikol'skiĭ, Nr. 98 (1968)*, pages 25–96. Amer. Math. Soc., Providence, Rhode Island, 1971.

[136] W. McCune and O. Shumsky. Ivy: A preprocessor and proof checker for first-order logic. In M. Kaufmann, P. Manolios, and J. Moore, editors, *Computer-Aided Reasoning: ACL2 Case Studies*, pages 265–282. Kluwer, 2000.

[137] Deborah L. McGuinness and Jon R. Wright. An industrial strength description logic-based configuration platform. *IEEE Intelligent Systems*, pages 69–77, 1998.

[138] P. Mika, D. Oberle, A. Gangemi, and M. Sabou. Foundations for service ontologies: Aligning OWL-S to dolce. In S.I. Feldman, M. Uretsky, M. Najork, and C.E. Wills, editors, *Proceedings of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 563–572. ACM Press, 2004.

[139] G. Mints. Gentzen-type systems and resolution rules. Part I: Propositional lo gic. In *Proceedings of COLOG-88*, volume 417 of *Lecture Notes in Computer Science*, pages 198–231. Springer, 1990.

[140] G. Mints, V. Orevkov, and T. Tammet. Transfer of sequent calculus strategies to resolution for S4. In H. Wansing, editor, *Proof Theory of Modal Logic*, volume 2 of *Applied Logic Series*, pages 17–31. Kluwer, 1996.

[141] L. Nguyen. MProlog. `http://www.mimuw.edu.pl/~nguyen/mprolog/`.

[142] L. Nguyen. A fixpoint semantics and an sld-resolution calculus for modal logic programs. *Fundamenta Informaticae*, 55(1):63–100, 2003.

[143] L. A. Nguyen. The modal logic programming system MProlog. In J. J. Alferes and J. A. Leite, editors, *Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA 2004)*, volume 3229 of *Lecture Notes in Artificial Intelligence*, pages 266–278. Springer, 2004.

[144] I. Niemelä. A tableau calculus for minimal model reasoning. In *Proceedings of the 5th International Workshop on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX '96)*, volume 1071 of *Lecture Notes in Artificial Intelligence*, pages 278–294. Springer, 1996.

[145] A. Nonnengart. First-order modal logic theorem proving and functional simulation. In R. Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 80–85. Morgan Kaufmann, 1993.

[146] A. Nonnengart. How to use modalities and sorts in prolog. In C. MacNish, D. Pearce, and L. M. Pereira, editors, *Proceedings of the European Workshop on Logics in Artificial Intelligence (JELIA '94)*, volume 838 of *Lecture Notes in Artificial Intelligence*, pages 365–378. Springer, 1994.

[147] A. Nonnengart, H. J. Ohlbach, and A. Szalas. Quantifier elimination for second-order predicate logic. To appear in *Logic, Language and Reasoning: Essays in honour of Dov Gabbay*, Part I, Kluwer.

[148] H. J. Ohlbach, A. Nonnengart, M. de Rijke, and D. Gabbay. Encoding two-valued nonclassical logics in classical logic. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 21, pages 1403–1486. Elsevier Science, 2001.

[149] H. J. Ohlbach and R. A. Schmidt. Functional translation and second-order frame properties of modal logics. *Journal of Logic and Computation*, 7(5):581–603, 1997.

[150] H. J. Ohlbach, R. A. Schmidt, and U. Hustadt. Translating graded modalities into predicate logic. In H. Wansing, editor, *Proof Theory of Modal Logic*, volume 2 of *Applied Logic Series*, pages 253–291. Kluwer, 1996.

[151] M. Ohnishi and K. Matsumoto. Gentzen method in modal calculi. *Osaka Mathematical Journal*, 9:113–130, 1957.

[152] M. Ohnishi and K. Matsumoto. Gentzen method in modal calculi II. *Osaka Mathematical Journal*, 11:115–120, 1959.

[153] F. Oppacher and E. Suen. HARP: A tableau-based theorem prover. *Journal of Automated Reasoning*, 4:69–100, 1988.

[154] M. A. Orgun and W. Ma. An overview of temporal and modal logic programming. In D. M. Gabbay and H. J. Ohlbach, editors, *Proceedings of the First International Conference on Temporal Logic (ICTL '94)*, volume 827 of *Lecture Notes in Artificial Intelligence*, pages 445–479. Springer, 1994.

[155] S. Owre, S. Rajan, J. M. Rushby, N. Shankar, and M. K. Srivas. PVS: Combining specification, proof checking, and model checking. In R. Alur and T. A. Henzinger, editors, *Proceedings of the 8th International Conference on Computer-Aided Verification (CAV '96)*, volume 1102 of *Lecture Notes in Computer Science*, pages 411–414. Springer, 1996.

[156] R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[157] R. Paige, R. E. Tarjan, and R. Bonic. A linear time solution to the single function coarsest partition problem. *Theoretical Computer Science*, 40:67–84, 1985.

[158] G. Pan and M. Y. Vardi. Optimizing a BDD-based modal solver. In F. Baader, editor, *Proceedings of the 19th International Conference on Automated Deduction (CADE-19)*, volume 2741 of *Lecture Notes in Artificial Intelligence*, pages 75–89. Springer, 2003.

[159] P. F. Patel-Schneider and I. Horrocks. DLP and FaCT. In *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX '99)*, volume 1397 of *Lecture Notes in Artificial Intelligence*, pages 19–23. Springer, 1999.

[160] Pellet OWL reasoner. Maryland Information and Network Dynamics Lab, 2003.

[161] D. A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2:293–304, 1986.

[162] V. R. Pratt. Models of program logics. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, 1979.

[163] A. Prior. *Past, Present and Future*. Oxford University Press, 1967.

[164] P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9(3):268–299, 1993.

[165] W. V. Quine. Variables explained away. In *Proceedings of the American Philosophy Society*, volume 104, pages 343–347, 1960.

[166] W. V. Quine. Algebraic logic and predicate functors. In R. Rudner and I. Scheffler, editors, *Logic and Art: Esssays in Honor of Nelson Goodman*. Bobbs-Merrill, 1971.

[167] A. L. Rector, W. A. Nowlan, and A. Glowinski. Goals for concept representation in the GALEN project. In *Proceedings of the 17th Annual Symposium on Computer Applications in Medical Care (SCAMC '93)*, pages 414–418, 1993.

[168] A. Riazanov and A. Voronkov. Splitting without backtracking. In B. Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*, pages 611–617. Morgan Kaufmann, 2001.

[169] A. Riazanov and A. Voronkov. The design and implementation of Vampire. *AI Communications*, 15(2–3):91–110, 2002.

[170] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41, 1965.

[171] U. Sattler. Description logics for the representation of aggregated objects. In W. Horn, editor, *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI 2000)*, pages 239–243. IOS Press, 2000.

[172] W. J. Savitch. Relationsship between nondeterministic and deterministic tape complexities. *Journal of Computer and System Science*, 4:177–192, 1970.

[173] K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 466–471. Morgan Kaufmann, 1991.

[174] R. A. Schmidt. MSPASS. `http://www.cs.man.ac.uk/~schmidt/mspass`.

[175] R. A. Schmidt. *Optimised Modal Translation and Resolution*. PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, 1997.

[176] R. A. Schmidt. E-unification for subsystems of S4. In T. Nipkow, editor, *Proceedings of the 9th International Conference on Rewriting Techniques and Applications (RTA '98)*, volume 1379 of *Lecture Notes in Computer Science*, pages 106–120. Springer, 1998.

[177] R. A. Schmidt. Decidability by resolution for propositional modal logics. *Journal of Automated Reasoning*, 22(4):379–396, 1999.

[178] R. A. Schmidt. Decidability by resolution for propositional modal logics. *Journal of Automated Reasoning*, 22(4):379–396, 1999.

[179] R. A. Schmidt and U. Hustadt. A resolution decision procedure for fluted logic. In D. McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction (CADE-17)*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 433–448. Springer, 2000.

[180] R. A. Schmidt and U. Hustadt. Mechanised reasoning and model generation for extended modal logics. In H. C. M. de Swart, E. Orlowska, G. Schmidt, and M. Roubens, editors, *Theory and Applications of Relational Structures as Knowledge Instruments*, volume 2929 of *Lecture Notes in Computer Science*, pages 38–67. Springer, 2003.

[181] R. A. Schmidt and U. Hustadt. A principle for incorporating axioms into the first-order translation of modal formulae. In F. Baader, editor, *Proceedings of the 19th International Conference on Automated Deduction (CADE-19)*, volume 2741 of *Lecture Notes in Artificial Intelligence*, pages 412–426. Springer, 2003. The long version is Preprint CSPP-22, University of Manchester, UK.

[182] R. A. Schmidt and U. Hustadt. First-order resolution methods for modal logics. In A. Podelski, A. Voronkov, and R. Wilhelm, editors, *Volume in memoriam of Harald Ganzinger*, Lecture Notes in Computer Science. Springer, 2006. To appear.
[183] S. Schulz. E: A Brainiac theorem prover. *Journal of AI Communications*, 15(2/3):111–126, 2002.
[184] E. Spaan. *Complexity of Modal Logics*. PhD thesis, University of Amsterdam, 1993.
[185] A. Szalas. On the correspondence between modal and classical logic: An automated approach. *Journal of Logic and Computation*, 3(6):605–620, 1993.
[186] S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH Aachen, 2001.
[187] J. van Benthem. Temporal logic. In D. Gabbay, C. Hogger, and J. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 4*, pages 241–350. Oxford Scientific Publishers, 1996.
[188] M. Y. Vardi. Reasoning about the past with two-way automata. In *Proceedings of the 25th Int. Colloq. on Automata, Languages, and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 628–641. Springer, 1998.
[189] U. Visser, H. Stuckenschmidt, G. Schuster, and T. Vögele. Ontologies for geographic information processing. *Computers & Geosciences*, 28(1):103–117, 2002.
[190] A. Voronkov. Theorem proving in non-standard logics based on the inverse method. In *Proceedings of the 11th Conference on Automated Deduction (CADE-11)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 648–662. Springer, 1992.
[191] D. H. D. Warren. An abstract prolog instruction set. Technical Note 309, SRI International, Menlo Park, CA, USA, 1983.
[192] C. Weidenbach. SPASS. `http://spass.mpi-sb.mpg.de`.
[193] C. Weidenbach. Combining superposition, sorts and splitting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 27, pages 1965–2013. Elsevier, 2001.
[194] C. Weidenbach et al. System description: SPASS version 1.0.0. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, volume 1632 of *Lecture Notes in Artificial Intelligence*, pages 378–382. Springer, 1999.
[195] W. Wong. Validation of HOL proofs by proof checking. *Formal Methods in System Design*, 14(2):193–212, 1999.
[196] M. Wooldridge. *Reasoning about rational agents*. MIT Press, 2000.
[197] C. Wroe, C. A. Goble, A. Roberts, and M. Greenwood. A suite of DAML+OIL ontologies to describe bioinformatics web services and data. *International Journal of Cooperative Information Systems*, 12(2):597–624, 2003.