Handbook of Satisfiability
Armin Biere, Marijn Heule, Hans van Maaren and Toby Walsh (Eds.)
IOS Press, 2009

Chapter 25

SAT Techniques for Modal and Description Logics

Roberto Sebastiani and Armando Tacchella

25.1. Introduction

In a nutshell, modal logics are propositional logics enriched with *modal operators*, —like $\Box$, $\Diamond$, $\Box_i$— which are able to represent complex facts like necessity, possibility, knowledge and belief. For instance, "$\Box\varphi$" and "$\Diamond\varphi$" may represent "necessarily φ" and "possibly φ" respectively, whilst "$\Box_1\Box_2\varphi$" may represent the fact that agent 1 knows that agent 2 knows the fact φ. Description logics are extensions of propositional logic which build on top of entities, concepts (unary relations) and roles (binary relations), which allow for representing complex concepts. For instance, the concept "MALE $\wedge$ $\exists$ CHILDREN ($\neg$ MALE $\wedge$ TEEN)", represents the set of fathers which have at least one teenager daughter.

The research in modal and description logics had followed two parallel routes until the seminal work by Schild [Sch91], who showed that the core modal logic K_m and the core description logic $\mathcal{ALC}$ are notational variants one of the other, and that analogous frameworks, results and algorithms had been conceived in parallel in the two communities. Since then, analogous results have been produced for a bunch of other logics, so that nowadays the two communities have substantially merged into one research flow.

In the last two decades, modal and description logics have provided a theoretical framework for important applications in many areas of computer science, including artificial intelligence, formal verification, database theory, distributed computing and, more recently, semantic web. For this reason, the problem of automated reasoning in modal and description logics has been thoroughly investigated (see, e.g., [Fit83, Lad77, HM92, BH91, Mas00]), and many approaches have been proposed for (efficiently) handling the satisfiability of modal and description logics, with a particular interest for the core logics K_m and $\mathcal{ALC}$ (see, e.g., [Fit83, BH91, GS00, HPS99, HS99, BGdR03, PSV02, SV06]). Moreover, a significant amount of benchmarks formulas have been produced for testing the effectiveness of the different techniques [HM92, GRS96, HS96, HPSS00, Mas99, PSS01, PSS03].

We briefly overview the main approaches for the satisfiability of modal and description logics which have been proposed in the literature. The "classic" *tableau-based* approach [Fit83, Lad77, HM92, Mas00] is based on the construction of propositional-tableau branches, which are recursively expanded on demand by generating successor nodes in a candidate Kripke model. In the *DPLL-based* approach [GS96a, SV98, GS00] a DPLL procedure, which treats the modal subformulas as propositions, is used as Boolean engine at each nesting level of the modal operators: when a satisfying assignment is found, the corresponding set of modal subformulas is recursively checked for modal consistency. Among the tools employing (and extending) this approach, we recall Ksat[GS96a, GGST00], *SAT [Tac99], Fact [Hor98b], Dlp [PS98], and Racer [HM01]. [1] This approach has lately been exported into the context of Satisfiability Modulo Theories - SMT [ACG00, ABC$^+$02], giving rise the so-called *on-line lazy approach* to SMT described in §26.4 (see also [Seb07]). The *CSP-based* approach [BGdR03] differs from the tableaux-based and DPLL-based ones mostly in the fact that a CSP engine is used instead of a tableaux/DPLL engine. KCSP is the representative tool of this approach. In the *translational* approach [HS99, AGHd00] the modal formula is encoded into first-order logic (FOL), and the encoded formula is then fed to a FOL theorem prover [AGHd00]. MSpass [HSW99] is the most representative tool of this approach. In the *Inverse-method* approach, a search procedure is based on the inverted version of a sequent calculus [Vor99, Vor01] (which can be seen as a modalized version of propositional resolution [PSV02]). Kꓘ is the representative tool of this approach. In the *Automata-theoretic* approach, or *OBDD-based* approach, (a OBDD-based symbolic representation of) a tree automaton accepting all the tree models of the input formula is implicitly built and checked for emptiness [PSV02, PV03]. KBDD [PV03] is the representative tool of this approach. [PV03] presents also an encoding of K-satisfiability into QBF-satisfiability – another PSPACE-complete problem – combined with the use of a state-of-the-art QBF solver Finally, in the *eager approach* [SV06, SV08] $K_m/\mathcal{ALC}$-formulas are encoded into SAT and then fed to a state-of-the-art SAT solver. K_m2SAT is the representative tool of this approach.

Most such approaches combine propositional reasoning with various techniques/encodings for handling the modalities, and thus are based on, or have largely benefited from, efficient propositional reasoning techniques. In particular, the usage of DPLL as a core Boolean reasoning technique produced a boost in the performance of the tools when it was adopted [GS96a, GS96b, Hor98b, PS98, HPS99, GGST00, HPSS00].

In this chapter we show how efficient Boolean reasoning techniques have been imported, used and integrated into reasoning tools for modal and description logics. To this extent, we focus on modal logics, and in particular mainly on K_m. Importantly, this chapter *does not* address the much more general issue of satisfiability in modal and description logics, because the reasoning techniques

[1] Notice that there is not an universal agreement on the terminology "tableau-based" and "DPLL-based". E.g., tools like Fact, Dlp, and Racer are often called "tableau-based", although they use a DPLL-like algorithm instead of propositional tableaux for handling the propositional component of reasoning [Hor98b, PS98, HPS99, HM01], because many scientists in these communities consider DPLL as an optimized version of propositional tableaux. The same issue holds for the Boolean system KE [DM94] and its derived systems.

which are specific for the different modal and description logics are orthogonal to the issue of Boolean reasoning. We refer the reader to the bibliography presented above and to [BCM+03] for a detailed description of those topics.

The chapter is organized as follows. In §25.2 we provide some background in modal logics. In §25.3 we describe a basic theoretical framework and we present and analyze the basic tableau-based and DPLL-based techniques. In §25.4 we present optimizations and extensions of the DPLL-based procedures. In §25.5 we present the automata-theoretic/OBDD-based approach. Finally, in §25.6 we present the eager approach.

25.2. Background

In this section we provide some background in modal logics. We refer the reader to, e.g., [Che80, Fit83, HM92] for a more detailed introduction.

25.2.1. The Modal Logic K_m

We start with some basic notions and notation (see, e.g., [Che80, Fit83, HM92] for more details). Given a non-empty set of primitive propositions $\mathcal{A} = \{A_1, A_2, \ldots\}$ and a set of m modal operators $\mathcal{B} = \{\Box_1, \ldots, \Box_m\}$, let the language Λ_m be the least set of formulas containing $\mathcal{A}$, closed under the set of propositional connectives $\{\neg, \wedge\}$ and the set of modal operators in $\mathcal{B}$. Notationally, we use capital letters $A_i, B_i, ...$ to denote primitive propositions and Greek letters $\alpha_i, \beta_i, \varphi_i, \psi_i$ to denote formulas in Λ_m. We use the standard abbreviations, that is: "$\Diamond_r\varphi_1$" for '$\neg\Box_r\neg\varphi_1$", "$\varphi_1 \vee \varphi_2$" for "$\neg(\neg\varphi_1 \wedge \neg\varphi_2)$", "$\varphi_1 \rightarrow \varphi_2$" for "$\neg(\varphi_1 \wedge \neg\varphi_2)$", "$\varphi_1 \leftrightarrow \varphi_2$" for "$\neg(\varphi_1 \wedge \neg\varphi_2) \wedge \neg(\varphi_2 \wedge \neg\varphi_1)$", "$\top$" and $\bot$ for the true and false constants respectively. Formulas like $\neg\neg\psi$ are implicitly assumed to be simplified into ψ; thus, if ψ is $\neg\phi$, then by "$\neg\psi$" we mean "ϕ". We often write "$(\bigwedge_i l_i) \rightarrow \bigvee_j l_j$" for the clause "$\bigvee_j \neg l_i \vee \bigvee_j l_j$", and "$(\bigwedge_i l_i) \rightarrow (\bigwedge_j l_j)$" for the conjunction of clauses "$\bigwedge_j(\bigvee_i \neg l_i \vee l_j)$". We call *depth* of φ, written *depth(φ)*, the maximum degree of nesting of modal operators in φ.

A K_m-formula is said to be in *Negative Normal Form (NNF)* if it is written in terms of the symbols $\Box_r$, $\Diamond_r$, $\wedge$, $\vee$ and propositional literals A_i, $\neg A_i$ (i.e., if all negations occur only before propositional atoms in $\mathcal{A}$). Every K_m-formula φ can be converted into an equivalent one $NNF(\varphi)$ by recursively applying the rewriting rules: $\neg\Box_r\varphi \Longrightarrow \Diamond_r\neg\varphi$, $\neg\Diamond_r\varphi \Longrightarrow \Box_r\neg\varphi$, $\neg(\varphi_1 \wedge \varphi_2) \Longrightarrow (\neg\varphi_1 \vee \neg\varphi_2)$, $\neg(\varphi_1 \vee \varphi_2) \Longrightarrow (\neg\varphi_1 \wedge \neg\varphi_2)$, $\neg\neg\varphi \Longrightarrow \varphi$.

A K_m-formula is said to be in *Box Normal Form (BNF)* [PSV02, PV03] if it is written in terms of the symbols $\Box_r$, $\neg\Box_r$, $\wedge$, $\vee$, and propositional literals A_i, $\neg A_i$ (i.e., if there are no diamonds, and if all negations occurs only before boxes or before propositional atoms in $\mathcal{A}$). Every K_m-formula φ can be converted into an equivalent one $BNF(\varphi)$ by recursively applying the rewriting rules: $\Diamond_r\varphi \Longrightarrow \neg\Box_r\neg\varphi$, $\neg(\varphi_1 \wedge \varphi_2) \Longrightarrow (\neg\varphi_1 \vee \neg\varphi_2)$, $\neg(\varphi_1 \vee \varphi_2) \Longrightarrow (\neg\varphi_1 \wedge \neg\varphi_2)$, $\neg\neg\varphi \Longrightarrow \varphi$.

The basic normal modal logic K_m can be defined axiomatically as follows. A formula in Λ_m is a theorem in K_m if it can be inferred from the following axiom schema:

$$K.\ \ (\Box_r\varphi_1 \wedge \Box_r(\varphi_1 \rightarrow \varphi_2)) \rightarrow \Box_r\varphi_2. \tag{25.1}$$

by means of tautological inference and of the application of the following inference rule:

$$\frac{\varphi}{\Box_r\varphi}\ (Necessitation), \tag{25.2}$$

for every formula φ, φ_1, φ_2 in Λ_m and for every $\Box_r \in \mathcal{B}$. The rule *Necessitation* characterizes most modal logics; the axiom K characterizes the normal modal logics.

The semantics of modal logics is given by means of Kripke structures. A *Kripke structure* for K_m is a tuple $M = \langle \mathcal{U}, \pi, \mathcal{R}_1, \ldots, \mathcal{R}_m \rangle$, where $\mathcal{U}$ is a set of states, π is a function $\pi : \mathcal{A} \times \mathcal{U} \longmapsto \{True, False\}$, and each $\mathcal{R}_r$ is a binary relation on the states of $\mathcal{U}$. With a little abuse of notation we write "$u \in M$" instead of "$u \in \mathcal{U}$". We call a pair M, u, a *situation.* The binary relation $\models$ between a modal formula φ and a pair M, u s.t. $u \in M$ is defined as follows:

$$\begin{array}{ll}
M, u \models A_i,\ A_i \in \mathcal{A} & \Longleftrightarrow \pi(A_i, u) = True; \\
M, u \models \neg\varphi_1 & \Longleftrightarrow M, u \not\models \varphi_1; \\
M, u \models \varphi_1 \wedge \varphi_2 & \Longleftrightarrow M, u \models \varphi_1 \textit{ and } M, u \models \varphi_2; \\
M, u \models \Box_r\varphi_1,\ \Box_r \in \mathcal{B} & \Longleftrightarrow M, v \models \varphi_1 \text{ for every } v \in M \text{ s.t. } \mathcal{R}_r(u, v) \text{ holds in } M.
\end{array}$$

We extend the definition of $\models$ to formula sets $\mu = \{\varphi_1, ..., \varphi_n\}$ as follows:

$$M, u \models \mu \iff M, u \models \varphi_i,\ \ for\ every\ \varphi_i \in \mu.$$

"$M, u \models \varphi$" should be read as "M, u *satisfy* φ in K_m" (alternatively, "M, u K_m-satisfy φ"). We say that a formula $\varphi \in \Lambda_m$ is satisfiable in K_m (K_m-satisfiable from now on) if and only if there exist M and $u \in M$ s.t. $M, u \models \varphi$. φ is *valid* for M, written $M \models \varphi$, if $M, u \models \varphi$ for every $u \in M$. φ is valid for a class of Kripke structures $\mathcal{K}$ if $M \models \varphi$ for every $M \in \mathcal{K}$. φ is said to be *valid in* K_m iff $M \models \varphi$ for every Kripke structure M. It can be proved that a Λ_m-formula φ is a theorem in K_m if and only if it is valid in K_m [Che80, Fit83, HM92].

When this causes no ambiguity we sometimes write "satisfiability" meaning "K_m-satisfiability". If $m = 1$, we simply write "K" for "K_1".

The problem of determining the K_m-satisfiability of a K_m-formula φ is decidable and PSPACE-complete [Lad77, HM92], even restricting the language to a single Boolean atom (i.e., $\mathcal{A} = \{A_1\}$) [Hal95]; if we impose a bound on the modal depth of the K_m-formulas, the problem reduces to NP-complete [Hal95]. Intuitively, every satisfiable formula φ in K_m can be satisfied by a Kripke structure M which is a finite tree and whose depth is given by $depth(\varphi) + 1$ (i.e., s.t. $|M| \leq |\varphi|^{depth(\varphi)}$). Such a structure can be spanned by an alternating-and/or search procedure, similarly to what is done with QBF. An encoding of K_m-satisfiability into QBF is presented in [Lad77, HM92]. For a detailed description on K_m, including complexity results, we refer the reader to [Lad77, HM92, Hal95].

Table 25.1. Axiom schemata and corresponding properties of $\mathcal{R}_r$ for the normal modal logics.

Axiom Schema	Property of $\mathcal{R}_r$	
B. $\neg\varphi \rightarrow \Box_r\neg\Box_r\varphi$	symmetric	$\forall\ u\ v.\ [\mathcal{R}_r(u,v) \Longrightarrow \mathcal{R}_r(v,u)]$
D. $\neg\Box_r\bot$	seriality	$\forall\ u.\ \exists\ v.\ [\mathcal{R}_r(u,v)]$
T. $\Box_r\varphi \rightarrow \varphi$	reflexive	$\forall\ u.\ [\mathcal{R}_r(u,u)]$
4. $\Box_r\varphi \rightarrow \Box_r\Box_r\varphi$	transitive	$\forall\ u\ v\ w.\ [\mathcal{R}_r(u,v)\ e\ \mathcal{R}_r(v,w) \Longrightarrow \mathcal{R}_r(u,w)]$
5. $\neg\Box_r\varphi \rightarrow \Box_r\neg\Box_r\varphi$	euclidean	$\forall\ u\ v\ w.\ [\mathcal{R}_r(u,v)\ e\ \mathcal{R}_r(u,w) \Longrightarrow \mathcal{R}_r(v,w)]$

Table 25.2. Properties of $\mathcal{R}_r$ for the various normal modal logics. The names between parentheses denote the names each logic is commonly referred with. (For better readability, we omit the pedex "$_m$" from the name of the logics.)

Logic $\mathcal{L} \in \mathcal{N}$ (Axiomatic Characterization)	Corresponding Properties of $\mathcal{R}_r$ (Semantic Characterization)
K	—
KB	symmetric
KD	serial
KT = KDT (T)	reflexive
K4	transitive
K5	euclidean
KBD	symmetric and serial
KBT = KBDT (B)	symmetric and reflexive
KB4 = KB5 = KB45	symmetric and transitive
KD4	serial and transitive
KD5	serial and euclidean
KT4 = KDT4 (S4)	reflexive and transitive
KT5 = KBD4 = KBD5 = KBT4 = KBT5 = KDT5 = KT45 = KBD45 = KBT45 = KDT45 = KBDT4 = KBDT5 = KBDT45 (S5)	reflexive, transitive and symmetric (equivalence)
K45	transitive and euclidean
KD45	serial, transitive and euclidean

25.2.2. Normal Modal Logics

We consider the class $\mathcal{N}$ of the normal modal logics. We briefly recall some of the standard definitions and results for these logics (see, e.g., [Che80, Fit83, HM92, Hal95]).

Given the language Λ_m, the class of normal modal logics on Λ_m, $\mathcal{N}$, can be described axiomatically as follows. The set of theorems in a logic $\mathcal{L}$ in $\mathcal{N}$ is the set of Λ_m-formulas which can be inferred by means of tautological inference and of the application of the Necessitation rule from the axiom schema K, plus a given subset of the axiom schemata $\{B, D, T, 4, 5\}$ described in the left column of Table 25.1. A list of normal modal logics built by combining such axiom schemata is presented in the left column of Table 25.2. Notice that each logic $\mathcal{L}$ is named after the list of its (modal) axiom schemata, and that many logics are equivalent, so that we have only 15 distinct logics out of the 32 possible combinations.

From the semantic point of view, the logics $\mathcal{L} \in \mathcal{N}$ differ from one another by imposing some restrictions on the relations $\mathcal{R}_r$ of the Kripke structures. As described in Table 25.1, each axiom schema in $\{B, D, T, 4, 5\}$ corresponds to a property on $\mathcal{R}_r$. In each logic $\mathcal{L}$, a formula φ can be satisfied only by Kripke structures whose relations $\mathcal{R}_r$'s verify the properties corresponding to $\mathcal{L}$, as described in Table 25.2. (E.g., φ is satisfiable in $KD4$ only by Kripke structures whose relations are both serial and transitive.) Consequently, φ is said to be *valid in* $\mathcal{L}$ if it is valid in the corresponding class of Kripke structures. For every $\mathcal{L}$ in $\mathcal{N}$, it can be proved that a Λ_m-formula is a theorem in $\mathcal{L}$ if and only if it is valid in $\mathcal{L}$ [Che80, Fit83].

The problem of determining the satisfiability in $\mathcal{L}$ in $\mathcal{N}$ ("$\mathcal{L}$-satisfiability" hereafter) of a Λ_m-formula φ is decidable for every $\mathcal{L}$. The computational complexity of the problem depends on the logic $\mathcal{L}$ and on many other factors, including the maximum number m of distinct box operators, the maximum number $|\mathcal{A}|$ of distinct primitive propositions, and the maximum modal depth of the formulas (denoted by *depth*). In the general case, for most logics $\mathcal{L} \in \mathcal{N}$ $\mathcal{L}$-satisfiability is PSPACE-complete; in some cases it may reduce to NP-complete, if $m = 1$ (e.g., with $K45$, $KD45$, $S5$), if *depth* is bounded (e.g., with K_m, T_m, $K45_m$, $KD45_m$, $S5_m$); in some cases it may reduce even to PTIME-complete is some of the features above combine with the fact that $|\mathcal{A}|$ is finite [Lad77, Hal95]. We refer the reader to [Lad77, HM92, Hal95, Ngu05] for a detailed description of these issues.

A *labeled formula* is a pair $\sigma : \varphi$, where φ is a formula in Λ and σ is a *label* (typically a sequence of integers) labeling a world in a Kripke structure for $\mathcal{L}$. If $\Gamma = \{\varphi_1, \ldots, \varphi_n\}$, we write $\sigma : \Gamma$ for $\{\sigma : \varphi_1, \ldots, \sigma : \varphi_n\}$. Intuitively, $\sigma : \varphi$ means "the formula φ in the world σ". For every $\mathcal{L} \in \mathcal{N}$, [Fit83, Mas94, Mas00] give a notion of *accessibility relation* between labels and gives the properties for these relations for the various logics $\mathcal{L}$. Essentially, they mirror the accessibility relation between the worlds they label.

25.2.3. Non-normal Modal Logics

We now consider the class of classical – also known as non-normal – modal logics. We briefly recall some of the standard definitions and results for these logics (see, e.g., [Che80, FHMV95]).

Given the language Λ_m, the basic classical modal logic on Λ_m, E_m, can be defined axiomatically as follows. The theorems in E_m are the set of formulas in Λ_m which can be inferred by tautological inference and by the application of the inference rule:

$$\frac{\varphi \leftrightarrow \psi}{\Box_r\varphi \leftrightarrow \Box_r\psi}\ (E). \tag{25.3}$$

As a consequence, the schemata

$$\begin{array}{l} N.\ \Box_r\top \\ M.\ \Box_r(\varphi \wedge \psi) \rightarrow \Box_r\varphi \\ C.\ (\Box_r\varphi \wedge \Box_r\psi) \rightarrow \Box_r(\varphi \wedge \psi) \end{array} \tag{25.4}$$

which are theorems in K_m do not hold in E_m. The three principles N, M, and C enforce closure conditions on the set of provable formulas which are not always desirable, especially if the $\Box_r$ operator has an epistemic (such as knowledge or belief) reading. If we interpret $\Box_r\varphi$ as "a certain agent r believes φ", then N enforces that r believes all the logical truths, M that r's beliefs are closed under logical consequence, and C that r's beliefs are closed under conjunction. These three closure properties are different forms of omniscience, and —as such— they might not be appropriate for modeling the beliefs of a real agent (see, e.g., [FHMV95]). By combining the schemata in (25.4) and using them as axiom schemata, we can get eight different combinations corresponding to eight distinct logics, where each logic is named after the list of its modal axiom schemata. The logic $EMCN_m$ corresponds to the basic normal modal logic K_m.

The semantics of classical modal logics is given by means of Montague-Scott structures. A *Montague-Scott structure* for E_m is a tuple $S = \langle \mathcal{U}, \pi, \mathcal{N}_1, \ldots, \mathcal{N}_m \rangle$, where $\mathcal{U}$ is a set of states, π is a function $\pi : \mathcal{A} \times \mathcal{U} \longmapsto \{True, False\}$, and each $\mathcal{N}_r$ is a relation $\mathcal{N}_r : \mathcal{U} \longmapsto \mathcal{P}(\mathcal{P}(\mathcal{U}))$, i.e., for each $u \in \mathcal{U}$, $\mathcal{N}_r(u) \subseteq \mathcal{P}(\mathcal{U})$. Notice that Montague-Scott structures are a generalization of Kripke structures, so the class of possible models for K_m is indeed a subclass of the possible models for E_m. In analogy with Section 25.2.1, we write "$u \in S$" instead of "$u \in \mathcal{U}$", and we call S, u a situation. The binary relation $\models$ between a modal formula φ and a pair S, u s.t. $u \in S$ is defined as follows:

$$\begin{array}{ll} S,u \models A_i,\ A_i \in \mathcal{A} & \Longleftrightarrow \pi(A_i,u) = True; \\ S,u \models \neg\varphi_1 & \Longleftrightarrow S,u \not\models \varphi_1; \\ S,u \models \varphi_1 \wedge \varphi_2 & \Longleftrightarrow S,u \models \varphi_1 \ \textit{and}\ S,u \models \varphi_2; \\ S,u \models \Box_r\varphi_1,\ \Box_r \in \mathcal{B} & \Longleftrightarrow \{v \mid M,v \models \varphi_1\} \in \mathcal{N}_r(u) \end{array} \tag{25.5}$$

We extend the definition of $\models$ to formula sets $\mu = \{\varphi_1, ..., \varphi_n\}$ as follows:

$$S,u \models \mu \iff S,u \models \varphi_i, \quad \textit{for every}\ \varphi_i \in \mu.$$

"$S, u \models \varphi$" should be read as "S, u *satisfy* φ in E_m" (alternatively, "S, u E_m-satisfy φ"). We say that a formula $\varphi \in \Lambda_m$ is satisfiable in E_m (E_m-satisfiable from now on) if and only if there exist S and $u \in S$ s.t. $S, u \models \varphi$. φ is *valid* for S, written $S \models \varphi$, if $S, u \models \varphi$ for every $u \in S$. φ is valid for a class of Montague-Scott structures $\mathcal{C}$ if $S \models \varphi$ for every $S \in \mathcal{C}$. φ is said to be *valid in* E_m iff $S \models \varphi$ for every Montague-Scott structure S. The semantics of the logic E is given by the relation defined in 25.5 only. The logics where one of M, C and N is an axiom require the following closure conditions on $\mathcal{N}_r$ to be satisfied for each r:

(M) if $U \subseteq V$ and $U \in \mathcal{N}_r(w)$ then $V \in \mathcal{N}_r(w)$ (closure by superset inclusion),
(C) if $U \in \mathcal{N}_r(w)$ and $V \in \mathcal{N}_r(w)$ then $U \cap V \in \mathcal{N}_r(w)$ (closure by intersection),
(N) $\mathcal{U} \in \mathcal{N}_r(w)$ (unit containment).

Notice that if an E_m structure S is such that $\mathcal{N}_r(u)$ satisfies all the above conditions for each world $u \in S$, then S is also a K_m structure.

In analogy with section 25.2.1, if $m = 1$, we simply write, e.g., "E" for "E_1", "EM" for "EM_1", and so on. For every non-normal logic $\mathcal{L}$, it can be proved that a Λ_m-formula is a theorem in $\mathcal{L}$ if and only if it is valid in $\mathcal{L}$ [Che80, Fit83].

The problem of determining the E_m-satisfiability of a E_m-formula φ is decidable, and the E_m-satisfiability problem is NP-complete [Var89]. Satisfiability is also NP-complete in all the classical modal logics that do not contain C as an axiom (EM, EN, EMN), while it is PSPACE-complete in the remaining ones (EC, EMC, ECN, EMCN). The satisfiability problems maintain the same complexity classes when considering multi-agent extensions.

25.2.4. Modal Logics and Description Logics

The connection between modal logics and terminological logics – also known as description logics – is due to a seminal paper by Klaus Schild [Sch91] where the description logic $\mathcal{ALC}$ [SSS91] is shown to be a notational variant of the modal logic K_m. Here we survey some of the results of [Sch91], and we refer the reader to [BCM+03] for further reading about the current state of the art in modal and description logics.

Following [Sch91], we start by defining the language of $\mathcal{ALC}$. Formally, the language $\mathcal{ALC}$ is defined by grammar rules of the form:

$$\begin{array}{l} C \rightarrow c \mid \top \mid C_1 \sqcap C_2 \mid \neg C \mid \forall R.C \\ R \rightarrow r \end{array} \tag{25.6}$$

where C, and C_i denote generical concepts, c denotes an atomic concept symbol and r a role symbol. The formal semantics of $\mathcal{ALC}$ is specified by an *extension function.* Let $\mathcal{D}$ be any set called the *domain.* An extension function ε over $\mathcal{D}$ is a function mapping concepts to subsets of D and roles to subsets of $\mathcal{D} \times \mathcal{D}$ such that

$$\begin{aligned} \varepsilon[\top] &= \mathcal{D} \\ \varepsilon[C \sqcap D] &= \varepsilon[C] \cap \varepsilon[D] \\ \varepsilon[\neg C] &= \mathcal{D} \setminus \varepsilon[C] \\ \varepsilon[\forall R.C] &= \{d \in \mathcal{D} \mid \forall \langle d, e\rangle \in \varepsilon[R] \quad e \in \varepsilon[C]\} \end{aligned} \tag{25.7}$$

Using extension functions, we can define the semantic notion of *subsumption*, *equivalence* and *coherence*: D *subsumes* C, written $\models C \sqsubseteq D$, iff for each extension function ε, $\varepsilon[C] \subseteq \varepsilon[D]$, whereas C and D are *equivalent*, written $\models C = D$ iff for each extension function ε, $\varepsilon[C] = \varepsilon[D]$. Finally, C is *coherent* iff there is an extension function ε with $\varepsilon[C] \neq \emptyset$. The following result allows us to concentrate on any of the above notions without loss of generality:

Lemma 1. *[Sch91] Subsumption, equivalence, and incoherence are log-space reducible to each other in any terminological logic comprising Boolean operations on concepts.*

Viewing $\mathcal{ALC}$ from the modal logic perspective (see 25.2.1), atomic concepts simply can be expounded as atomic propositions, and can be interpreted as the set of states in which such propositions hold. In this case "$\forall.$" becomes a modal operator since it is applied to formulas. Thus, e.g., $\neg c_1 \sqcup \forall r.(c_2 \sqcap c_3)$ can be expressed by the K_m-formula $\neg A_1 \vee \Box_r(A_2 \wedge A_3)$. The subformula $\Box_r(A_2 \wedge A_3)$ is to be read as "agent r *knows* $A_2 \wedge A_3$", and means that in every state accessible for r, both A_2 and A_3 hold.[2] Actually

[2]Notice that we replaced primitive concepts c_i with $i \in \{1, 2, 3\}$ with propositions A_i, assuming the obvious bijection between the two sets.

- the domain of an extension function can be read as a set of states $\mathcal{U}$,
- atomic concepts can be interpreted as the set of worlds in which they hold, if expounded as atomic formulas, and
- atomic roles can be interpreted as accessibility relations.

Hence $\forall R.C$ can be expounded as "all states in which agent R knows proposition C" instead of "all objects for which all R's are in C".

To establish the correspondence between $\mathcal{ALC}$ and K_m consider the function f mapping $\mathcal{ALC}$ concepts to K_m-formulas with $f(c_i) = A_i$ for $i \in 1, 2, \ldots$, i.e., f maps concept symbols to primitive propositions, $f(\top) = \top$, $f(C \sqcap D) = f(C) \wedge f(D)$, $f(\neg C) = \neg f(C)$ and $f(\forall R.C) = \Box_R f(C)$. It could easily be shown by induction on the complexity of $\mathcal{ALC}-$concepts that f is a linearly length-bounded isomorphism such that an $\mathcal{ALC}-$concept C is coherent iff the K_m-formula $f(C)$ is satisfiable. Formally:

Theorem 1. *[Sch91] $\mathcal{ALC}$ is a notational variant of the propositional modal logic K_m, and satisfiability in K_m has the same computational complexity as coherence in $\mathcal{ALC}$.*

By this correspondence, several theoretical results for K_m can easily be carried over to $\mathcal{ALC}$. We immediately know, for example, that without loss of generality, any decision procedure for K_m-satisfiability is also a decision procedure for $\mathcal{ALC}-$coherence.

There are other result (see, e.g., [BCM$^+$03]) that link normal modal logics, as described in 25.2.2, to various description logics that extend $\mathcal{ALC}$ in several ways. According to these results, decision procedures for expressive description logics may be regarded as decision procedures for various normal modal logics, e.g., KD, T, B, and the other way round.

25.3. Basic Modal DPLL

In this section we introduce the basic concepts of modal tableau-based and DPLL-based procedures, and we discuss their relation.

25.3.1. A Formal Framework

Assume w.l.o.g. that the $\Diamond_r$'s are not part of the language (each $\Diamond_r \varphi$ can be rewritten into $\neg\Box_r\neg\varphi$). We call *atom* every formula that cannot be decomposed propositionally, that is, every formula whose main connective is not propositional. Examples of atoms are, A_1, A_i (*propositional atoms*), $\Box_1(A_1 \vee \neg A_2)$ and $\Box_2(\Box_1 A_1 \vee \neg A_2)$ (*modal atoms*). A *literal* is either an atom or its negation. Given a formula φ, an atom [literal] is a *top-level atom [literal]* for φ if and only if it occurs in φ and under the scope of no boxes. $Atoms^0(\varphi)$ is the set of the top-level atoms of φ.

We call a *truth assignment* μ for a formula φ a truth value assignment to all the atoms of φ. A truth assignment is *total* if it assigns a value to all atoms in φ, *partial* otherwise. Syntactically identical instances of the same atom are always assigned identical truth values; syntactically different atoms, e.g., $\Box_1(\varphi_1 \vee \varphi_2)$

and $\Box_1(\varphi_2 \vee \varphi_1)$, are treated differently and may thus be assigned different truth values.

To this extent, we introduce a bijective function $\mathcal{L}2\mathcal{P}$ ("$\mathcal{L}$-to-Propositional") and its inverse $\mathcal{P}2\mathcal{L} := \mathcal{L}2\mathcal{P}^{-1}$ ("Propositional-to-$\mathcal{L}$"), s.t. $\mathcal{L}2\mathcal{P}$ maps top-level Boolean atoms into themselves and top-level non-Boolean atoms into fresh Boolean atoms — so that two atom instances in φ are mapped into the same Boolean atom iff they are syntactically identical— and distributes with sets and Boolean connectives. (E.g., $\mathcal{L}2\mathcal{P}(\{\Box_r\varphi_1, \neg(\Box_r\varphi_1 \vee \neg A_1)\})$ is $\{B_1, \neg(B_1 \vee \neg A_1)\}$.) $\mathcal{L}2\mathcal{P}$ and $\mathcal{P}2\mathcal{L}$ are also called *Boolean abstraction* and *Boolean refinement* respectively.

We represent a truth assignment μ for φ as a set of literals

$$\begin{aligned}\mu = \{\ & \Box_1\alpha_{11}, \ldots, \Box_1\alpha_{1N_1}, \neg\Box_1\beta_{11}, \ldots, \neg\Box_1\beta_{1M_1}, \\ & \vdots \\ & \Box_m\alpha_{m1}, \ldots, \Box_m\alpha_{mN_m}, \neg\Box_m\beta_{m1}, \ldots, \neg\Box_m\beta_{mM_m}, \\ & A_1, \ldots, \neg A_R, \neg A_{R+1}, \ldots, \neg A_S\},\end{aligned} \tag{25.8}$$

$\Box_r\alpha_i$'s, $\Box_r\beta_j$'s being modal atoms and A_i's being propositional atoms. Positive literals $\Box_r\alpha_i$ and A_k in μ mean that the corresponding atom is assigned to true, negative literals $\neg\Box_r\beta_i$ and $\neg A_k$ mean that the corresponding atom is assigned to false. If $\mu_2 \subseteq \mu_1$, then we say that μ_1 *extends* μ_2 *and that* μ_2 subsumes μ_1. A *restricted truth assignment*

$$\mu^r = \{\Box_r\alpha_{r1}, \ldots, \Box_r\alpha_{rN_r}, \neg\Box_r\beta_{r1}, \ldots, \neg\Box_r\beta_{rM_r}\} \tag{25.9}$$

is given by restricting μ to the set of atoms in the form $\Box_r\psi$, where $1 \leq r \leq m$. Trivially μ^r subsumes μ.

Notationally, we use the Greek letters μ, η to represent truth assignments. Sometimes we represent the truth assignments in (25.8) and (25.9) also as the formulas given by the conjunction of their literals:

$$\mu = \begin{cases} \bigwedge_{i=1}^{N_1} \Box_1\alpha_{1i} \wedge \bigwedge_{j=1}^{M_1} \neg\Box_1\beta_{1j} \wedge \\ \ldots \\ \bigwedge_{i=1}^{N_m} \Box_m\alpha_m \wedge \bigwedge_{j=1}^{M_m} \neg\Box_m\beta_{mj} \wedge \\ \bigwedge_{k=1}^{R} A_k \ \wedge \bigwedge_{h=R+1}^{S} \neg A_h, \end{cases} \tag{25.10}$$

$$\mu^r = \bigwedge_i \Box_r\alpha_{ri} \wedge \bigwedge_j \neg\Box_r\beta_{rj}. \tag{25.11}$$

For every logic $\mathcal{L}$, we say that an assignment μ [restricted assignment μ^r] is $\mathcal{L}$-satisfiable meaning that its corresponding formula (25.10) [(25.11)] is $\mathcal{L}$-satisfiable.

We say that a total truth assignment μ for φ *propositionally satisfies* φ, written $\mu \models_p \varphi$, if and only if $\mathcal{L}2\mathcal{P}(\mu) \models \mathcal{L}2\mathcal{P}(\varphi)$, that is, for all sub-formulas

φ_1, φ_2 of φ:

$$\begin{array}{ll}
\mu \models_p \varphi_1,\ \varphi_1 \in \mathit{Atoms}^0(\varphi) & \Longleftrightarrow \varphi_1 \in \mu, \\
\mu \models_p \neg\varphi_1 & \Longleftrightarrow \mu \not\models_p \varphi_1, \\
\mu \models_p \varphi_1 \wedge \varphi_2 & \Longleftrightarrow \mu \models_p \varphi_1 \textit{ and } \mu \models_p \varphi_2.
\end{array}$$

We say that a partial truth assignment μ *propositionally satisfies* φ if and only if all the total truth assignments for φ which extend μ propositionally satisfy φ. For instance, if $\varphi = \Box_1\varphi_1 \vee \neg\Box_2\varphi_2$, then the partial assignment $\mu = \{\Box_1\varphi_1\}$ is such that $\mu \models_p \varphi$. In fact, both $\{\Box_1\varphi_1\ \Box_2\varphi_2\}$ and $\{\Box_1\varphi_1, \neg\Box_2\varphi_2\}$ propositionally satisfy φ. Henceforth, if not otherwise specified, when dealing with propositional satisfiability we do not distinguish between assignments and partial assignments. Intuitively, if we consider a formula φ as a propositional formula in its top-level atoms, then $\models_p$ is the standard satisfiability in propositional logic. Thus, for every φ_1 and φ_2, we say that $\varphi_1 \models_p \varphi_2$ if and only if $\mu \models_p \varphi_2$ for every μ s.t. $\mu \models_p \varphi_1$. We say that φ is *propositionally satisfiable* if and only if there exist an assignment μ s.t. $\mu \models_p \varphi$. We also say that $\models_p \varphi$ (φ is *propositionally valid*) if and only if $\mu \models_p \varphi$ for every assignment μ for φ. Thus $\varphi_1 \models_p \varphi_2$ if and only if $\models_p \varphi_1 \to \varphi_2$, and $\models_p \varphi$ iff $\neg\varphi$ is propositionally unsatisfiable. Notice that $\models_p$ is stronger than $\models$, that is, if $\varphi_1 \models_p \varphi_2$, then $\varphi_1 \models \varphi_2$, but not vice versa. E.g., $\Box_r\varphi_1 \wedge \Box_r(\varphi_1 \to \varphi_2) \models \Box_r\varphi_2$, but $\Box_r\varphi_1 \wedge \Box_r(\varphi_1 \to \varphi_2) \not\models_p \Box_r\varphi_2$.

Example 1. *Consider the following K_2 formula φ and its Boolean abstraction $\mathcal{L}2\mathcal{P}(\varphi)$:*

$$\begin{array}{rl}
\varphi = & \{\underline{\neg\Box_1(\neg A_3 \vee \neg A_1 \vee A_2)}\ \vee A_1 \vee A_5\} \\
\wedge & \{\underline{\neg A_2} \vee \neg A_5 \vee \Box_1(\neg A_2 \vee A_4 \vee A_5)\} \\
\wedge & \{A_1 \vee \underline{\Box_2(\neg A_4 \vee A_5 \vee A_2)}\ \vee A_2\} \\
\wedge & \{\neg\Box_2(A_4 \vee \neg A_3 \vee A_1)\ \vee \underline{\neg\Box_1(A_4 \vee \neg A_2 \vee A_3)}\ \vee \neg A_5\} \\
\wedge & \{\neg A_3 \vee A_1 \vee \underline{\Box_2(\neg A_4 \vee A_5 \vee A_2)}\} \\
\wedge & \{\underline{\Box_1(\neg A_5 \vee A_4 \vee A_3)}\ \vee \Box_1(\neg A_1 \vee A_4 \vee A_3) \vee \neg A_1\} \\
\wedge & \{A_1 \vee \underline{\Box_1(\neg A_2 \vee A_1 \vee A_4)}\ \vee A_2\}
\end{array}$$

$$\begin{array}{rl}
\mathcal{L}2\mathcal{P}(\varphi) = & \{\underline{\neg B_1}\ \vee A_1 \vee A_5\} \\
\wedge & \{\underline{\neg A_2} \vee \neg A_5 \vee B_2\} \\
\wedge & \{A_1 \vee \underline{B_3}\ \vee A_2\} \\
\wedge & \{\neg B_4\ \vee \underline{\neg B_5}\ \vee \neg A_5\} \\
\wedge & \{\neg A_3 \vee A_1 \vee \underline{B_3}\} \\
\wedge & \{\underline{B_6}\ \vee B_7 \vee \neg A_1\} \\
\wedge & \{A_1 \vee \underline{B_8}\ \vee A_2\}
\end{array}$$

The partial assignment $\mu^p = \{B_6, B_8, \neg B_1, \neg B_5, B_3, \neg A_2\}$ satisfies $\mathcal{L}2\mathcal{P}(\varphi)$, so that the following assignment $\mu := \mathcal{P}2\mathcal{L}(\mu^p)$ propositionally satisfies φ:

$$\begin{array}{rlll}
\mu = & \Box_1(\neg A_5 \vee A_4 \vee A_3) \wedge & \Box_1(\neg A_2 \vee A_1 \vee A_4) \wedge & [\bigwedge_i \Box_1\alpha_{1i}] \\
& \neg\Box_1(\neg A_3 \vee \neg A_1 \vee A_2) \wedge & \neg\Box_1(A_4 \vee \neg A_2 \vee A_3) \wedge & [\bigwedge_j \neg\Box_1\beta_{1j}] \\
& \Box_2(\neg A_4 \vee A_5 \vee A_2) \wedge & & [\bigwedge_i \Box_2\alpha_{2i}] \\
& \neg A_2. & & [\bigwedge_k A_k\ \wedge \bigwedge_h \neg A_h]
\end{array}$$

μ gives rise to two restricted assignments μ^1 and μ^2:

$$\begin{array}{rlr}\mu^1 = & \Box_1(\neg A_5 \vee A_4 \vee A_3) \wedge \quad \Box_1(\neg A_2 \vee A_1 \vee A_4) \wedge & [\bigwedge_i \Box_1\alpha_{1i}] \\ & \neg\Box_1(\neg A_3 \vee \neg A_1 \vee A_2) \wedge \neg\Box_1(A_4 \vee \neg A_2 \vee A_3) & [\bigwedge_j \neg\Box_1\beta_{1j}] \\ \mu^2 = & \Box_2(\neg A_4 \vee A_5 \vee A_2) & [\bigwedge_i \Box_2\alpha_{2i}].\end{array}$$

We say that a collection $\mathcal{M} := \{\mu_1, \dots, \mu_n\}$ of (possibly partial) assignments propositionally satisfying φ is *complete* if and only if, for every total assignment η s.t. $\eta \models_p \varphi$, there exists $\mu_j \in \mathcal{M}$ s.t. $\mu_j \subseteq \eta$. Intuitively, $\mathcal{M}$ can be seen as a compact representation of the whole set of total assignments propositionally satisfying φ.

Proposition 1. *[SV98] Let φ be a formula and let $\mathcal{M} := \{\mu_1, \dots, \mu_n\}$ be a complete collection of truth assignments propositionally satisfying φ. Then, for every $\mathcal{L}$, φ is $\mathcal{L}$-satisfiable if and only if μ_j is $\mathcal{L}$-satisfiable for some $\mu_j \in \mathcal{M}$.*

We also notice the following fact.

Proposition 2. *[Seb01] Let α be a non-Boolean atom occurring only positively [resp. negatively] in φ. Let $\mathcal{M}$ be a complete set of assignments satisfying φ, and let*

$$\mathcal{M}' := \{\mu_j \setminus \{\neg\alpha\} | \ \mu_j \in \mathcal{M}\} \qquad [\textit{resp.}\ \{\mu_j \setminus \{\alpha\} \mid \mu_j \in \mathcal{M}\}].$$

Then (i) for every $\mu'_j \in \mathcal{M}'$, $\mu'_j \models_p \varphi$, and (ii) φ is $\mathcal{L}$-satisfiable if and only if there exist a $\mathcal{L}$-satisfiable $\mu'_j \in \mathcal{M}'$.

proposition 1 shows that the $\mathcal{L}$-satisfiability of a formula can be reduced to that of a complete collection of sets of literals (assignment), for every call. Proposition 2 says that, if we have non-Boolean atoms occurring only positively [resp. negatively] in the input formula, we can safely drop every negative [resp. positive] occurrence of them from all assignments in a complete set $\mathcal{M}$ preserving the completeness of $\mathcal{M}$. In general, $\mathcal{L}$-satisfiability of a conjunction of literals depends on $\mathcal{L}$ [Fit83, Che80]. The following propositions give a recursive definition for K_m.

Proposition 3. *[GS00] The truth assignment μ of Equation (25.10) is K_m-satisfiable if and only if the restricted truth assignment μ^r of Equation (25.11) is K_m-satisfiable, for all $\Box_r$'s.* [3]

Proposition 4. *[GS00] The restricted assignment μ^r of Equation (25.11) is K_m-satisfiable if and only if the formula*

$$\varphi^{rj} = \bigwedge_i \alpha_{ri} \wedge \neg\beta_{rj} \tag{25.12}$$

is K_m-satisfiable, for every $\neg\Box_r\beta_{rj}$ occurring in μ^r.

Notice that propositions 3 and 4 can be merged into one single theorem stating that μ is K_m-satisfiable if and only if φ^{rj} is K_m-satisfiable, for all r and j. Notice furthermore that the depth of every φ^{rj} is strictly smaller than the depth of φ.

[3]Notice that the component $\bigwedge_{k=1}^{R} A_k \ \wedge \bigwedge_{h=R+1}^{S} \neg A_h$ in (25.10) is consistent because μ is a truth assignment.

Example 2. *Consider the formula φ and the assignments μ, μ^1 and μ^2 in Example 1. μ propositionally satisfies φ. Thus, for proposition 1, φ is K_m-satisfiable if μ is K_m-satisfiable. By proposition 3, μ is K_m-satisfiable if and only if both μ^1 and μ^2 are K_m-satisfiable; by proposition 4, μ^2 is trivially K_m-satisfiable, as it contains no negated boxes, and μ^1 is K_m-satisfiable if and only if each of the formulas*

$$\begin{aligned}\varphi^{11} &= \bigwedge_i \alpha_{1i} \wedge \neg\beta_{11} = (\neg A_5 \vee A_4 \vee A_3) \wedge (\neg A_2 \vee A_1 \vee A_4) \wedge A_3 \wedge A_1 \wedge \neg A_2,\\ \varphi^{12} &= \bigwedge_i \alpha_{1i} \wedge \neg\beta_{12} = (\neg A_5 \vee A_4 \vee A_3) \wedge (\neg A_2 \vee A_1 \vee A_4) \wedge \neg A_4 \wedge A_2 \wedge \neg A_3\end{aligned}$$

is K_m-satisfiable. As they both are satisfiable propositional formulas, then φ is K_m-satisfiable.

Proposition 1 reduces the $\mathcal{L}$-satisfiability of a formula φ to the $\mathcal{L}$-satisfiability of a complete collection of its truth assignments, for every $\mathcal{L}$. If $\mathcal{L}$ is K_m, propositions 3 and 4 show how to reduce the latter to the K_m-satisfiability of formulas of smaller depth. This process can be applied recursively, decreasing the depth of the formula considered at each iteration. Following these observations, it is possible to test the K_m-satisfiability of a formula φ by implementing a recursive alternation of two basic steps [GS96a, GS96b]:

1. Propositional reasoning: using some procedure for propositional satisfiability, find a truth assignment μ for φ s.t. $\mu \models_p \varphi$;
2. Modal reasoning: check the K_m-satisfiability of μ by generating the corresponding restricted assignments μ_r's and formulas φ^{rj}'s.

The two steps recurse down until we get to a truth assignment with no modal atoms. At each level, the process is repeated until either a K_m-satisfiable assignment is found (in which case φ is K_m-satisfiable) or no more assignments are found (in which case φ is not K_m-satisfiable).

25.3.2. Modal Tableaux

We call "tableau-based" a system that implements and extends to other logics the Smullyan's propositional tableau calculus, as defined in [Smu68]. Tableau-based procedures basically consist of a control strategy applied on top of a tableau framework. By tableau framework for modal logics we denote a refutation formal system extending Smullyan's propositional tableau with rules handling the modal operators (modal rules). Thus, for instance, in our terminology Kris [BH91, BFH$^+$94], Crack [BFT95] and LWB [HJSS96] are tableau-based systems.

For instance, in the labeled tableau framework for normal modal logics in $\mathcal{N}$ described in [Fit83, Mas94, Mas00], branches are represented as sets of labeled formulas $u : \psi$, where u labels the state in which the formula ψ has to be satisfiable. At the first step the root $1 : \varphi$ is created, φ being the modal formula to be proved (un)satisfiable. At the i-th step, a branch is expanded by applying to a chosen labeled formula the rule corresponding to its main connective, and adding the resulting labeled formula to the branch. The rules are the following: [4]

[4] Analogous rules handling negated $\wedge$'s and $\vee$'s, double negations $\neg\neg$, single and double implications $\rightarrow$ and $\leftrightarrow$, diamonds $\Diamond_r$, n-ary $\wedge$'s and $\vee$'s, and the negation of all them, can

$$\frac{u:(\varphi_1 \wedge \varphi_2)}{u:\varphi_1, u:\varphi_2}\ (\wedge) \qquad \frac{u:(\varphi_1 \vee \varphi_2)}{u:\varphi_1 \qquad u:\varphi_2}\ (\vee), \tag{25.13}$$

$$\frac{u:\neg\Box_r\varphi}{u':\neg\varphi}\ (\neg\Box_r) \qquad \frac{u:\Box_r\varphi}{u'':\varphi}\ (\Box_r)\ . \tag{25.14}$$

The modal rules are constrained by the following applicability conditions:

- **$\neg\Box_r$-rule**: u' is a new state (u' is said to be *directly accessible* from u);
- **$\Box_r$-rule**: u'' is an existing state which is accessible from u via $\mathcal{R}_r$.

Distinct logics $\mathcal{L}$ differ for different notions of accessibility in the $\Box_r$-rule [Fit83, Mas94, Mas00].

Every application of the $\vee$-rule splits the branch into two sub-branches. A branch is *closed* when a formula ψ and its negation $\neg\psi$ occur in it. The procedure stops when all branches are closed (φ is $\mathcal{L}$-unsatisfiable) or no more rule is applicable (φ is $\mathcal{L}$-satisfiable).

For some modal logics it is possible to drop labels by using alternative sets of non-labeled modal rules [Fit83]. For instance in K_m it is possible to use unlabeled formulas and update branches according to the rules

$$\frac{\Gamma, \varphi_1 \wedge \varphi_2}{\Gamma, \varphi_1, \varphi_2}\ (\wedge) \qquad \frac{\Gamma, \varphi_1 \vee \varphi_2}{\Gamma, \varphi_1 \qquad \Gamma, \varphi_2}\ (\vee) \tag{25.15}$$

$$\frac{\mu}{\alpha_1 \wedge \ldots \wedge \alpha_m \wedge \neg\beta_j}\ (\Box_r/\neg\Box_r) \tag{25.16}$$

for each box-index $r \in \{1, ..., m\}$. Γ is an arbitrary set of formulas, and μ is a set of literals which includes $\neg\Box_r\beta_j$ and whose only positive $\Box_r$-atoms are $\Box_r\alpha_1, \ldots, \Box_r\alpha_m$.

This describes the tableau-based decision procedure of Figure 25.1, which is the restriction to K_m of the basic version of the Kris procedure described in [BH91]. Tableau-based formalisms for many modal logics are described, e.g., in [Fit83, Mas94]. Tableau-based procedures for many modal logics are described, e.g., in [BH91, BFH$^+$94, BFT95, HJSS96].

25.3.3. From Modal Tableaux to Modal DPLL

We call "DPLL-based" any system that implements and extends to other logics the Davis-Putnam-Longeman-Loveland procedure (DPLL) [DP60, DLL62]. DPLL-based procedures basically consist on the combination of a procedure handling purely-propositional component of reasoning, typically a variant of the DPLL algorithm, and some procedure handling the purely-modal component, typically consisting of a control strategy applied on top of a modal tableau rules.

be derived straightforwardly, and are thus omitted here. Following [Fit83], the $\wedge$-, $\vee$-, $\neg\Box_r$- and $\Box_r$-rules (and those for their equivalent operators) are often called α-, β-, π-, and ν-rules respectively.

```
function $K_m$-Tableau($\Gamma$)
    if $\psi_i \in \Gamma$ and $\neg\psi_i \in \Gamma$                              /* branch closed */
        then return False;
    if $(\varphi_1 \wedge \varphi_2) \in \Gamma$                                     /* $\wedge$-elimination */
        then return $K_m$-Tableau($\Gamma \cup \{\varphi_1, \varphi_2\} \backslash \{(\varphi_1 \wedge \varphi_2)\}$);
    if $(\varphi_1 \vee \varphi_2) \in \Gamma$                                       /* $\vee$-elimination */
        then return    $K_m$-Tableau($\Gamma \cup \{\varphi_1\} \backslash \{(\varphi_1 \vee \varphi_2)\}$) or
                       $K_m$-Tableau($\Gamma \cup \{\varphi_2\} \backslash \{(\varphi_1 \vee \varphi_2)\}$);
    for every $r \in \{1, ..., m\}$ do
        for every $\neg\Box_r\beta_j \in \Gamma$ do                                 /* branch expanded */
            if not $K_m$-Tableau($\{\neg\beta_j\} \cup \bigcup_{\Box_r\alpha_i \in \Gamma}\{\alpha_i\}$)
                    then return False;
    return True;
```

Figure 25.1. An example of a tableau-based procedure for K_m. We omit the steps for the other operators.

Thus, for instance, in our terminology Ksat [GS96a, GS00], Fact [Hor98b, Hor98a], Dlp [PS98], Racer [HM01] are DPLL-based systems. [5]

From a purely-logical viewpoint, it is possible to conceive a DPLL-based framework by substituting the propositional tableaux rules with some rules implementing the DPLL algorithms in a tableau-based framework [SV98]. For instance, one can conceive a DPLL-based framework for a normal logic $\mathcal{L}$ from Fitting or Massacci's frameworks (see §25.3.2) by substituting the $\vee$-rule (25.13) with the following rules:

$$\frac{u : (l \vee C)}{u : l \qquad u : \neg l}\ (Branch) \qquad \frac{u : l \qquad u : (\neg l \vee C)}{u : C}\ (Unit), \tag{25.17}$$

where l is a and C is a disjunction of literals. [6] More recent and richer formal frameworks for representing DPLL and DPLL-based procedures are described in [Tin02, NOT06].

As stated in §25.3.2, for some modal logics it is possible to drop labels by using alternative sets of non-labeled modal rules [Fit83]. If so, DPLL-based procedures can be implemented more straightforwardly. For instance, in K_m it is possible to use unlabeled formulas and update branches according to the following rules [SV98]:

$$\frac{\varphi}{\mu_1 \quad \mu_2 \quad \ldots \quad \mu_n}\ (DPLL) \qquad \frac{\mu}{\alpha_1 \wedge \ldots \wedge \alpha_m \wedge \neg\beta_j}\ (\Box_r/\neg\Box_r) \tag{25.18}$$

where the $\Box_r/\neg\Box_r$-rule is that of (25.16), and $\{\mu_1, \ldots, \mu_n\}$ is a *complete* set of assignments for φ, which can be produced by the DPLL algorithm.

[5] See footnote 1 in §25.1.

[6] Here we assume for simplicity that the input formula is in conjunctive normal form (CNF). Equivalent formalisms are available for non-CNF formulas [DM94, Mas98].

```
function Ksat($\varphi$)
    return Ksat$_F$($\varphi, \top$);

function Ksat$_F$($\varphi, \mu$)
    if $\varphi = \top$                                          /* base      */
        then return Ksat$_A(\mu)$;
    if $\varphi = \bot$                                          /* backtrack */
        then return False;
    if {a unit clause ($l$) occurs in $\varphi$}                 /* unit      */
        then return Ksat$_F$($assign(l, \varphi), \mu \wedge l$);
    $l$ := choose-literal($\varphi$);                            /* split     */
    return  Ksat$_F$($assign(l, \varphi), \mu \wedge l$)  or
            Ksat$_F$($assign(\neg l, \varphi), \mu \wedge \neg l$);

/* $\mu$ is $\bigwedge_i \Box_1\alpha_{1i} \wedge \bigwedge_j \neg\Box_1\beta_{1j} \wedge \ldots \wedge \bigwedge_i \Box_m\alpha_{mi} \wedge \bigwedge_j \neg\Box_m\beta_{mj} \wedge \bigwedge_k A_k \wedge \bigwedge_h \neg A_h$ */
function Ksat$_A$($\mu$)
    for each box index $r \in \{1...m\}$ do
        if not Ksat$_{AR}$($\bigwedge_i \Box_r\alpha_{ri} \wedge \bigwedge_j \neg\Box_r\beta_{rj}$)
            then return False;
    return True;

/* $\mu^r$ is $\bigwedge_i \Box_r\alpha_{ri} \wedge \bigwedge_j \neg\Box_r\beta_{rj}$ */
function Ksat$_{AR}$($\mu^r$)
    for each literal $\neg\Box_r\beta_{rj} \in \mu$ do
        if not Ksat($\bigwedge_i \alpha_{ri} \wedge \neg\beta_{rj}$)
            then return False;
    return True;
```

Figure 25.2. The basic version of Ksat algorithm.

25.3.4. Basic Modal DPLL for K_m

The ideas described in §25.3.3 were implemented in the Ksat procedure [GS96a, GS00], whose basic version is reported in Figure 25.2. This schema evolved from that of the PTAUT procedure in [AG93], and is based on the "classic" DPLL procedure [DP60, DLL62]. Ksat takes in input a modal formula φ and returns a truth value asserting whether φ is K_m-satisfiable or not. Ksat invokes Ksat$_F$(where "$_F$" stands for "Formula"), passing as arguments φ and (by reference) the empty assignment $\top$. Ksat$_F$ tries to build a K_m-satisfiable assignment μ propositionally satisfying φ. This is done recursively, according to the following steps:

- (base) If $\varphi = \top$, then μ satisfies φ. Thus, if μ is K_m-satisfiable, then φ is K_m-satisfiable. Therefore Ksat$_F$ invokes Ksat$_A(\mu)$ (where "$_A$" stands for *A*ssignment), which returns a truth value asserting whether μ is K_m-satisfiable or not.
- (backtrack) If $\varphi = \bot$, then μ does not satisfy φ, so that Ksat$_F$ returns *False*.
- (unit) If a literal l occurs in φ as a unit clause, then l must be assigned $\top$. [7] To obtain this, Ksat$_F$ is invoked recursively with arguments the formula returned by *assign*(l, φ) and the assignment obtained by adding l to μ.

[7] A notion of unit clause for non-CNF propositional formulas is given in [AG93].

assign(l, φ) substitutes every occurrence of l in φ with $\top$ and evaluates the result.

- (split) If none of the above situations occurs, then *choose-literal(φ)* returns an unassigned literal l according to some heuristic criterion. Then $\textsc{Ksat}_F$ is first invoked recursively with arguments *assign(l, φ)* and $\mu \wedge l$. If the result is negative, then $\textsc{Ksat}_F$ is invoked with arguments *assign($\neg l, \varphi$)* and $\mu \wedge \neg l$.

$\textsc{Ksat}_F$ is a variant of the "classic" DPLL algorithm [DP60, DLL62]. The $\textsc{Ksat}_F$ schema differs from that of classic DPLL by only two steps.

The first difference is the "base" case: when finding an assignment μ which propositionally satisfies the input formula, it simply returns "$True$". $\textsc{Ksat}_F$ instead is supposed also to check the K_m-satisfiability of the corresponding set of literals, by invoking $\textsc{Ksat}_A$ on μ. If the latter returns $true$, then the whole formula is satisfiable and $\textsc{Ksat}_F$ returns $True$ as well; otherwise, $\textsc{Ksat}_F$ backtracks and looks for the next assignment.

The second difference is in the fact that in $\textsc{Ksat}_F$ the pure-literal step [DLL62] is removed. [8] In fact the sets of assignments generated by DPLL with pure-literal might be incomplete and might cause incorrect results, as shown by the following example.

Example 3. *Let φ be the following formula:*

$$(\Box_1 A_1 \vee A_1) \wedge (\Box_1(A_1 \rightarrow A_2) \vee A_2) \wedge (\neg\Box_1 A_2 \vee A_2) \wedge (\neg A_2 \vee A_3) \wedge (\neg A_2 \vee \neg A_3).$$

φ is K_m-satisfiable, because $\mu = \{A_1, \neg A_2, \Box_1(A_1 \rightarrow A_2), \neg\Box_1 A_2\}$ is an assignment which propositionally satisfies φ and which is also modally consistent. It is easy to see that no satisfiable assignment propositionally satisfying φ assigns $\Box_1 A_1$ to true. As $\Box_1 A_1$ occurs only positively in φ, DPLL with the pure literal rule would assign $\Box_1 A_1$ to true as first step, which would lead the procedure to return $False$.

With these simple modifications, the embedded DPLL procedure works as an enumerator of a complete set of assignments, whose K_m-satisfiability is recursively checked by $\textsc{Ksat}_A$.

$\textsc{Ksat}_A(\mu)$ invokes $\textsc{Ksat}_{AR}(\mu_r)$ (where "$_{AR}$" stands for *R*estricted *A*ssignment) for every box index r. This is repeated until either $\textsc{Ksat}_{AR}$ returns a negative value (in which case $\textsc{Ksat}_A(\mu)$ returns *False*) or no more $\Box_r$'s are available (in which case $\textsc{Ksat}_A(\mu)$ returns *True*). $\textsc{Ksat}_{AR}(\mu_r)$ invokes $\textsc{Ksat}(\varphi^{rj})$ for any conjunct $\neg\Box_r\beta_{rj}$ occurring in μ_r. Again, this is repeated until either $\textsc{Ksat}$ returns a negative value (in which case $\textsc{Ksat}_{AR}(\mu_r)$ returns *False*) or no more $\neg\Box_r\beta_{rj}$'s are available (in which case $\textsc{Ksat}_{AR}(\mu_r)$ returns *True*). Notice that $\textsc{Ksat}_F$, $\textsc{Ksat}_A$ and $\textsc{Ksat}_{AR}$ are a direct implementation of propositions 1, 3 and 4, respectively. This guarantees their correctness and completeness.

[8] Alternatively, the application of the pure-literal rule can be restricted to atomic propositions only.

25.3.5. Modal DPLL vs. Modal Tableaux

[GS96a, GS96b, GS00, GGST98, GGST00, HPS99, HPSS00] presented extensive empirical comparisons, in which DPLL-based procedures outperformed tableau-based ones, with performance gaps that can reach orders of magnitude. (Similar performance gaps between tableau-based vs. DPLL-based procedures were obtained lately also in a completely-different context [ACG00].) Remarkably, most such results were obtained with tools implementing the "classic" DPLL procedure of §25.3.4, very far from the efficiency of current DPLL implementations.

We concentrate on the basic tableau-based and DPLL-based algorithms for K_m-satisfiability described in §25.3.2 and §25.3.4. Both procedures work (i) by enumerating truth assignments which propositionally satisfy the input formula φ and (ii) by recursively checking the K_m-satisfiability of the assignments found. Both algorithms perform the latter step in the same way. The key difference is thus in the way they handle propositional inference. [GS96b, GS00] remarked that, regardless the quality of implementation and the optimizations performed, tableau-based procedures have, with respect to DPLL-based procedures, two weaknesses which make them intrinsically less efficient, and whose effects get up to exponentially amplified when using them in modal inference. We consider them in turn.

Syntactic vs. semantic branching. In a propositional tableau truth assignments are generated as branches induced by the application of the $\vee$-rule to *disjunctive subformulas* of the input formula φ. Thus, they perform what we call *syntactic branching* [GS96b], that is, the branching in the search tree is induced by the syntactic structure of φ. As discussed in [D'A92, DM94], an application of the $\vee$-rule generates two subtrees which can be *mutually consistent*, i.e., which may share propositional models. [9] Therefore, the set of truth assignments enumerated by propositional tableau procedures grows exponentially with the number of disjunctions occurring positively in φ, regardless the fact that it may contain up to exponentially-many duplicated and/or subsumed assignments.

Things get even worse in the modal case. When testing K_m-satisfiability, unlike the propositional case where tableaux look for *one* assignment satisfying the input formula, the propositional tableaux are used to enumerate *all* the truth assignments, which must be recursively checked for K_m-consistency. This requires checking recursively possibly-many sub-formulas of the form $\bigwedge_i \alpha_{ri} \wedge \neg\beta_j$ of depth $d-1$, for which a propositional tableau will enumerate all truth assignments, and so on. At all levels of nesting, a redundant truth assignment introduces a redundant modal search tree. Thus, with modal formulas the redundancy of the propositional case propagates with the modal depth, and, in the worst case, the number of redundant truth assignments can become exponential.

DPLL instead, performs a search which is based on what we call *semantic branching* [GS96b], that is, a branching on the *truth value* of sub-formulas ψ of φ

[9] As pointed out in [D'A92, DM94], the propositional tableaux rules are unable to represent *bivalence*: "every proposition is either true or false, *tertium non datur*". This is a consequence of the elimination of the cut rule in cut-free sequent calculi, from which propositional tableaux are derived.

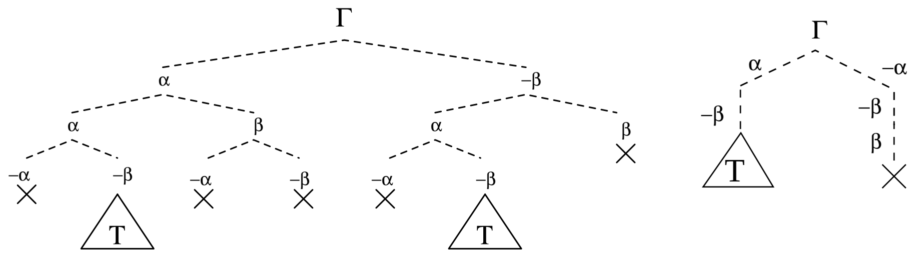

Figure 25.3. Search trees for the formula $\Gamma = (\alpha \vee \neg\beta) \wedge (\alpha \vee \beta) \wedge (\neg\alpha \vee \neg\beta)$. Left: a tableau-based procedure. Right: a DPLL-based procedure.

(typically atoms): [10]

$$\frac{\varphi}{\varphi[\psi/\top] \qquad \varphi[\psi/\bot]},$$

where $\varphi[\psi/\top]$ is the result of substituting with $\top$ all occurrences of ψ in φ and then simplify the result. Thus, every branching step generates two *mutually-inconsistent* subtrees. [11] Because of this, DPLL always generates non-redundant sets of assignments. This avoids any search duplication and, in the case of modal search, any recursive exponential propagation of such a redundancy.

Example 4. *Consider the simple formula $\Gamma = (\alpha \vee \neg\beta) \wedge (\alpha \vee \beta) \wedge (\neg\alpha \vee \neg\beta)$, where α and β are modal atoms s.t. $\alpha \wedge \neg\beta$ is not modally consistent. and let d be the depth of Γ. The only possible assignment propositionally satisfying Γ is $\mu = \alpha \wedge \neg\beta$. Look at Figure 25.3 left. Assume that in a tableau-based procedure, the $\vee$-rule is applied to the three clauses occurring in Γ in the order they are listed. Then two distinct but identical open branches are generated, both representing the assignment μ. Then the tableau expands the two open branches in the same way, until it generates two identical (and possibly big) closed modal sub-trees T of modal depth d, each proving the K_m-unsatisfiability of μ.*

This phenomenon may repeat itself at the lower level in each sub-tree T, and so on. For instance, if $\alpha = \Box_1((\alpha' \vee \neg\beta') \wedge (\alpha' \vee \beta'))$ and $\beta = \Box_1(\alpha' \wedge \beta')$, then at the lower level we have a formula Γ' of depth $d-1$ analogous to Γ. This propagates exponentially the redundancy with the depth d.

Finally, notice that if we considered the formula $\Gamma^K = \bigwedge_{i=1}^{K}(\alpha_i \vee \neg\beta_i) \wedge (\alpha_i \vee \beta_i) \wedge (\neg\alpha_i \vee \neg\beta_i)$, the tableau would generate 2^K identical truth assignments $\mu^K = \bigwedge_i \alpha_i \wedge \neg\beta_i$, and things would get exponentially worse.

Look at Figure 25.3, right. A DPLL-based procedure branches asserting $\alpha = \top$ or $\alpha = \bot$. The first branch generates $\alpha \wedge \neg\beta$, while the second gives $\neg\alpha \wedge \neg\beta \wedge \beta$, which immediately closes. Therefore, only one instance of $\mu = \alpha \wedge \neg\beta$ is generated. The same applies to μ^K.

[10]Notice that the notion of "semantic branching" introduced in [GS96b] is stronger than that lately used in [Hor98b, HPS99], the former corresponding to the latter plus the usage of unit-propagation.

[11] This fact holds for both "classic" [DP60, DLL62] and "modern" DPLL (see, e.g., [ZM02]), because in both cases two branches differ for the truth value of at least one atom, although for the latter case the explanation is slightly more complicate.

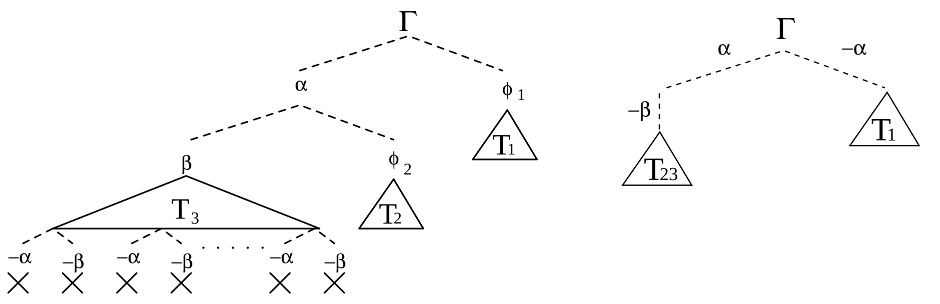

Figure 25.4. Search trees for the formula $\Gamma = (\alpha \vee \phi_1) \wedge (\beta \vee \phi_2) \wedge \phi_3 \wedge (\neg\alpha \vee \neg\beta)$. Left: a tableau-based procedure. Right: a DPLL-based procedure.

Detecting constraint violations. A propositional formula φ can be seen as a set of constraints for the truth assignments which possibly satisfy it. For instance, a clause $A_1 \vee A_2$ constrains every assignment not to set both A_1 and A_2 to $\bot$. Unlike tableaux, DPLL prunes a branch as soon as it violates some constraint of the input formula. (For instance, in KSAT this is done by the function *assign*.)

Example 5. *Consider the formula $\Gamma = (\alpha \vee \phi_1) \wedge (\beta \vee \phi_2) \wedge \phi_3 \wedge (\neg\alpha \vee \neg\beta)$, α and β being atoms, ϕ_1, ϕ_2 and ϕ_3 being sub-formulas, such that $\alpha \wedge \beta \wedge \phi_3$ is propositionally satisfiable and $\alpha \wedge \phi_2$ is K_m-unsatisfiable. Look at Figure 25.4, left. Again, assume that, in a tableau-based procedure, the $\vee$-rule is applied in order, left to right. After two steps, the branch α, β is generated, which violates the constraint imposed by the last clause $(\neg\alpha \vee \neg\beta)$. A tableau-based procedure is not able to detect such a violation until it explicitly branches on that clause, that is, only after having generated the whole sub-tableau T_3 for $\alpha \wedge \beta \wedge \phi_3$, which may be rather big. DPLL instead (Figure 25.4, right) avoids generating the violating assignment detects the violation and immediately prunes the branch.*

25.4. Advanced Modal DPLL

In this section we present the most important optimizations of the DPLL-based procedures described in §25.3, and one extension to non-normal modal logics.

25.4.1. Optimizations

As described in §25.3.4, the first DPLL-based tools of [GS96a, GS96b, SV98, GS00] were based on the "classic" recursive DPLL schema [DP60, DLL62]. Drastic improvements in performances were lately obtained by importing ideas and techniques from the SAT literature and/or by directly implementing tools on top of modern DPLL solvers, which are applied to the Boolean abstraction of the input formula [GGST98, GGST00, HPS99, Tac99, GGT01, HM01].

In particular, modern DPLL implementation are non-recursive, and are based on very efficient, destructive data structures to handle Boolean formulas and assignments. They benefit of sophisticated search techniques (e.g., backjumping, learning, restarts [MSS96, BS97, GSK98]), smart splitting heuristics (e.g., [MMZ$^+$01, GN02, ES04]), highly-engineered data structures and implementation tricks (e.g., the two-watched literal scheme [MMZ$^+$01]), and advanced preprocessing techniques [Bra01, BW03, EB05]. In particular, modern DPLL implementations perform *conflict analysis* on failed assignments μ's, which detect the *reason* of each failure, that is, a (typically much smaller) subset μ' of μ which alone causes the failure. When this happens, the procedure

- adds the negation of μ' as a new clause to the formula, so that no assignment containing μ' will be ever investigated again. This technique is called *learning*;
- backtracks to the highest point in the stack where one literal l in the learned clause $\neg\mu'$ is not assigned, it unit propagates l, and it proceeds with the search. This technique is called *backjumping*.

Backjumping and learning are of great interest in our discussion, as it will be made clear in §25.4.1.4. The other DPLL optimizations come for free by using state-of-the-art SAT solvers and they are substantially orthogonal to our discussion, so that they will not be discussed here.

We describe some further optimizations which have been proposed to the basic schema of §25.3.4. For better readability, the description will refer to the case of K_m, but they can be extended to other logics. Most of these techniques and optimizations have lately been adopted by the so-called lazy tools for Satisfiability Modulo Theories, SMT (see §26.4.3).

25.4.1.1. Normalizing atoms

One potential source of inefficiency for DPLL-based procedures is the occurrence in the input formula of equivalent though syntactically-different atoms (e.g., $\Box_r(A_1 \vee A_2)$ and $\Box_r(A_2 \vee A_1)$), or pairs atoms in which one is equivalent to the negation of the other (e.g. $\Box_r(A_1 \vee A_2)$ and $\Diamond_r(\neg A_1 \wedge \neg A_2)$). If two atoms ψ_1, ψ_2 are s.t. $\psi_1 \neq \psi_2$ and $\models \psi_1 \leftrightarrow \psi_2$ [resp. $\psi_1 \neq \neg\psi_2$ and $\models \psi_1 \leftrightarrow \neg\psi_2$], then they are recognized as distinct Boolean atoms $B_1 =_{def} \mathcal{L}2\mathcal{P}(\psi_1)$ and $B_2 =_{def} \mathcal{L}2\mathcal{P}(\psi_2)$, which may be assigned different [resp. identical] truth values by DPLL. This may cause the useless generation of many unsatisfiable assignments and the corresponding useless calls to $\textsc{Ksat}_A$ (e.g., up to $2^{|Atoms^0(\varphi)|-2}$ calls on assignments like $\{\Box_r(A_1 \vee A_2), \neg\Box_r(A_2 \vee A_1)...\}$).

In order to avoid these problems, it is wise to preprocess atoms so that to map as many as possible equivalent literals into syntactically identical ones [GS96a, GS00, HPS99]. This can be achieved by applying some rewriting rules, like, e.g.:

- *Drop dual operators*: $\Box_r(\varphi_1\vee\varphi_2), \Diamond_r(\neg\varphi_1\wedge\neg\varphi_2) \Longrightarrow \Box_r(\varphi_1\vee\varphi_2), \neg\Box_r(\varphi_1\vee\varphi_2)$, or even $(\varphi_1 \wedge \varphi_2), (\neg\varphi_1 \vee \neg\varphi_2) \Longrightarrow (\varphi_1 \wedge \varphi_2), \neg(\varphi_1 \wedge \varphi_2)$
- *Exploit associativity*: $\Box_r(\varphi_1 \vee (\varphi_2 \vee \varphi_3)), \Box_r((\varphi_1 \vee \varphi_2) \vee \varphi_3) \Longrightarrow \Box_r(\varphi_1 \vee \varphi_2 \vee \varphi_3)$,
- *Sort*: $\Box_r(\varphi_2 \vee \varphi_1 \vee \varphi_3), \Box_r(\varphi_3 \vee \varphi_1 \vee \varphi_2) \Longrightarrow \Box_r(\varphi_1 \vee \varphi_2 \vee \varphi_3)$.
- *Exploit properties of normal modal logics*: $\Box_r(\varphi_1 \wedge \varphi_2) \Longrightarrow \Box_r\varphi_1 \wedge \Box_r\varphi_2$ if $\mathcal{L} \in \mathcal{N}$.
- *Exploit specific properties of some logic $\mathcal{L}$*: $\Box_r\Box_r\varphi_1 \Longrightarrow \Box_r\varphi_1$ if $\mathcal{L}$ is S5.

Notice that pre-conversion to BNF (§25.2.1) goes in this direction.

Example 6. *Consider the modal atoms occurring in the formula φ in Example 1. For every modal atom in φ there are $3! = 6$ equivalent permutations, which are all mapped into one atom if the modal atoms are sorted. E.g., if we consider an equivalent formula φ' in which the second occurrence of the atom $\Box_2(\neg A_4 \vee A_5 \vee A_2)$, occurring in rows 3 and 5, is rewritten as $\Box_2(A_5 \vee \neg A_4 \vee A_2)$, then the latter will be encoded by $\mathcal{L}2\mathcal{P}$ into a different Boolean variable, namely B_9, which could be assigned by DPLL a different truth value wrt. B_3, generating an modally inconsistent assignment μ'. If all atoms in φ' are pre-sorted, then the problem does not occur.*

25.4.1.2. Early Pruning

Another optimization [GS96a, GS00, Tac99] was conceived after the empirical observation that most assignments found by DPLL are "trivially" K_m-unsatisfiable, that is, they will remain K_m-unsatisfiable even after removing some of their conjuncts. If an incomplete [12] assignment μ' is K_m-unsatisfiable, then all its extensions are K_m-unsatisfiable. If the unsatisfiability of μ' is detected on time, then this prevents checking the K_m-satisfiability of all the up to $2^{|Atoms^0(\varphi)|-|\mu'|}$ truth assignments which extend μ'.

This suggests the introduction of an intermediate K_m-satisfiability test on incomplete assignments just before the split. (Notice there is no need to introduce similar tests before unit propagation.) In the basic algorithm of Figure 25.2, this is done by introducing the three lines below in the function KSAT$_F$ of Figure 25.2, just before the "split":

```
if (Likely-Unsatisfiable($\mu$))          /* early-pruning */
    if not KSAT$_A$($\mu$)
        then return False;
```

(We temporarily ignore the test performed by *Likely-Unsatisfiable.*) KSAT$_A$ is invoked on the current incomplete assignment μ. If KSAT$_A(\mu)$ returns *False*, then all possible extensions of μ are unsatisfiable, and therefore KSAT$_F$ returns *False*. The introduction of this intermediate check, which is called *early pruning*, caused a drastic improvement in the overall performances [GS96a, GS00].

Example 7. *Consider the formula φ of Example 1. Suppose that, after three recursive calls,* KSAT$_F$ *builds the incomplete assignment:*

$$\mu' = \Box_1(\neg A_1 \vee A_4 \vee A_3) \wedge \Box_1(\neg A_2 \vee A_1 \vee A_4) \wedge \neg\Box_1(A_4 \vee \neg A_2 \vee A_3)$$

(rows 6, 7 and 4 of φ). If it is invoked on μ', KSAT$_A$ *will check the K_2-satisfiability of the formula*

$$(\neg A_1 \vee A_4 \vee A_3) \wedge (\neg A_2 \vee A_1 \vee A_4) \wedge \neg A_4 \wedge A_2 \wedge \neg A_3,$$

which is unsatisfiable. Therefore there will be no more need to select further literals, and KSAT$_F$ *will backtrack.*

[12]By *incomplete* assignment μ for φ we mean that μ has not assgined enough atoms to determine whether $\mu \models \varphi$ or not.

The intermediate consistency checks, however, may introduce some useless calls to $\textsc{Ksat}_A$.

One way of addressing this problem is to condition the calls to $\textsc{Ksat}_A$ in early-pruning steps to some heuristic criteria (here represented by the heuristic function *Likely-Unsatisfiable*). The main idea is to avoid invoking $\textsc{Ksat}_A$ when it is very unlikely that, since the last call, the new literals added to μ can cause inconsistency: e.g., when they are added only literals which are purely-propositional or contain new Boolean atoms [GS96a, GS00].

Another way is to make $\textsc{Ksat}_A$ work in an *incremental* way: if for some box index $r \in \{1...m\}$ no literal of the form $\Box_r\psi$ or $\neg\Box_r\psi$ has been added to μ since the last call to $\textsc{Ksat}_A$, then $\textsc{Ksat}_A$ can avoid performing the corresponding call to $\textsc{Ksat}_{AR}$; moreover, if for some box index $r \in \{1...m\}$ no positive $\Box_r\psi$'s have been added to μ since the last call to $\textsc{Ksat}_A$, then $\textsc{Ksat}_{AR}$ can avoid calling recursively $\textsc{Ksat}$ on the subformulas $(\bigwedge_i \alpha_{ri} \wedge \neg\beta_{rj})$ s.t. $\neg\Box_r\beta_{rj}$ was already passed to $\textsc{Ksat}_A$ in the last call [Tac99].

25.4.1.3. Caching

This section is an overview of [GT01] to which we refer for further reading. Consider the basic version of $\textsc{Ksat}$ algorithm in Figure 25.2. Without loss of generality, in the remainder of this section we assume that $|\mathcal{B}| = 1$, so that the call to $\textsc{Ksat}_A$ is the same as $\textsc{Ksat}_{AR}$. The extension to the case where $|\mathcal{B}| > 1$ is straightforward since it simply requires checking the different modalities in separate calls to $\textsc{Ksat}_{AR}$. Given two assignments μ and μ', it may be the case that $\textsc{Ksat}_A(\mu)$ and $\textsc{Ksat}_A(\mu')$ perform some equal subtests, i.e., recursive calls to $\textsc{Ksat}$. This is the case, e.g., when μ and μ' differ only for the propositional conjuncts and there is at least one conjunct of the form $\neg\Box\beta$. To prevent recomputation, the obvious solution is to cache both the formula whose satisfiability is being checked and the result of the check. Then, the cache is consulted before performing each subtest to determine whether the result of the subtest can be assessed on the basis of the cache contents.

In the following, we assume to have two different caching mechanisms, each using a separate caching structure:

- S-cache to store and query about satisfiable formulas, and
- U-cache to store and query about unsatisfiable formulas.

In this way, storing a subtest amounts to storing the formula in the appropriate caching structure. Of course, the issue is how to implement effective caching mechanisms allowing to reduce the number of subtests as much as possible. To this extent, the following considerations are in order:

1. if a formula $\bigwedge_{\alpha\in\Delta'} \alpha \wedge \neg\beta$ has already been determined to be satisfiable then, if $\Delta \subseteq \Delta'$, we can conclude that also $\bigwedge_{\alpha\in\Delta} \alpha \wedge \neg\beta$ is satisfiable, and
2. if a formula $\bigwedge_{\alpha\in\Delta'} \alpha \wedge \neg\beta$ has already been determined to be unsatisfiable then, if $\Delta \supseteq \Delta'$, we can conclude that also $\bigwedge_{\alpha\in\Delta} \alpha \wedge \neg\beta$ is unsatisfiable.

The above observations suggest the usage of caching mechanisms that allow for storing sets of formulas and for efficiently querying about subsets or supersets.

```
function KSAT_A(μ)
  $\Delta := \{\alpha \mid \Box\alpha$ is a conjunct of $\mu\}$;
  $\Gamma := \{\beta \mid \neg\Box\beta$ is a conjunct of $\mu\}$;
  if U-cache_get($\Delta, \Gamma$) return False;
  $\Gamma_r :=$ S-cache_get($\Delta, \Gamma$);
  $\Gamma_s := \emptyset$;
  foreach $\beta \in \Gamma_r$ do
    if not KSAT($\bigwedge_{\alpha\in\Delta} \alpha \wedge \neg\beta$) then
      if $\Gamma_s \neq \emptyset$ then S-cache_store($\Delta, \Gamma_s$);
      U-cache_store($\Delta, \beta$);
      return False
    else $\Gamma_s := \Gamma_s \cup \{\beta\}$;
  S-cache_store($\Delta, \Gamma_s$);
  return True.
```

Figure 25.5. KSAT$_A$: satisfiability checking for K with caching

In other words, given a subtest

$$\text{KSAT}(\bigwedge_{\alpha\in\Delta} \alpha \wedge \neg\beta), \qquad (25.19)$$

we want to be able to query our S-cache about the presence of a formula

$$\bigwedge_{\alpha\in\Delta'} \alpha \wedge \neg\beta \qquad (25.20)$$

with $\Delta \subseteq \Delta'$ (query for subsets or *subset-matching*). Analogously, given the subtest (25.19), we want to be able to query our U-cache about the presence of a formula (25.20) with $\Delta \supseteq \Delta'$ (query for supersets or *superset-matching*). In this way, caching a subtest avoids the recomputation of the very same subtest, *and* of the possibly many "subsumed" subtests.

Observations 1 and 2 are independent of the particular modal logic being considered. They are to be taken into account when designing caching structures for satisfiability in any modal logic. Of course, depending on the particular modal logic considered, some other considerations might be in order. For example, in K, we observe that in KSAT$_A$ there is a natural unbalance between satisfiable subtests and unsatisfiable ones. In fact, with reference to Figure 25.2, when testing an assignment μ

3. many subtests can be determined to be satisfiable, all sharing the same set Δ, and
4. at most one subtest may turn out to be unsatisfiable.

Observation 3 suggests that S-cache should be able to store satisfiable subtests sharing a common set Δ in a compact way. Therefore, S-cache associates the set Δ to the set $\Gamma' \subseteq \Gamma$, representing the "computed" satisfiable subtests $\bigwedge_{\alpha\in\Delta} \alpha \wedge \neg\beta$ for each $\beta \in \Gamma'$. Observation 4 suggests that U-cache should not care about subtests sharing a common Δ. Therefore, U-cache associates Δ to the single β for which the subtest $\bigwedge_{\alpha\in\Delta} \alpha \wedge \neg\beta$ failed.

Given the design issues outlined above, we can modify KSAT to yield the procedure KSAT$_A$ shown in Figure 25.5. In the Figure:

- U-cache_get(Δ, Γ) returns *True* if U-cache contains a set Δ' such that $\Delta \supseteq \Delta'$, Δ' is associated with β and $\beta \in \Gamma$;
- S-cache_get(Δ, Γ) returns the set $\Gamma \setminus \Gamma'$ where Γ' is the union over all the sets Γ'' such that for some set $\Delta' \supseteq \Delta$, Γ'' is associated to Δ' in S-cache.
- U-cache_store(Δ, β) stores in U-cache the set Δ and associates β to it;
- S-cache_store(Δ, Γ) stores in S-cache the set Δ and associates to it the set Γ.

The new issue is now to implement effective data structures for S-cache and U-cache supporting the above functions. Clearly, we expect that the computational costs associated to the above functions will be superior to the computational costs associated to other caching structures designed for "equality-matching", i.e., effectively supporting the functions obtained from the above by substituting "$\supseteq$" with "$=$". There is indeed a trade-off between "smart but expensive" and "simple but efficient" data-structures for caching. Of course, depending on

- the particular logic being considered, and
- the characteristics of the particular formula being tested,

we expect that one caching mechanism will lead to a faster decision process than the others.

Independently from the data-structure being used, the following (last) observation needs to be taken into account when dealing with modal logics whose decision problem is not in NP (e.g., K, S4):

5. testing the consistency of a formula may require an exponential number of subtests.

This is the case for the Halpern and Moses formulas presented in [HM85] for various modal logics. Observation 5 suggests that it may be necessary to bound the size of the cache, and introduce mechanisms for deciding which formulas to discard when the bound is reached. Further discussion about the implementation of caching and low-level optimizations can be found in [GT01].

25.4.1.4. Modal Backjumping

Another very important optimization, called *modal backjumping* [Hor98a, PS98], generalizes the idea of backjumping in DPLL. KSAT_A can be easily modified so that, when invoked on a K_m-unsatisfiable set of modal literals μ, it returns also the subset μ' of μ which caused the inconsistency of μ. We call μ', a *modal conflict set* of μ.

An easy way of computing μ' is that of returning the set $\mathcal{L}2\mathcal{P}(\{\Box_r \alpha_{ri}\}_i \cup \{\neg\Box_r \beta_{rj}\})$ corresponding to the first formula $\varphi^{rj} = \bigwedge_i \alpha_{ri} \wedge \neg\beta_{rj}$ which is found unsatisfiable by KSAT.

Example 8. *Consider the formula φ of Example 1. The assignment $\mu^p = \{B_6, B_8, B_2, \neg B_1, \neg B_5, B_3\}$ is found by KSAT_F, which satisfies $\mathcal{L}2\mathcal{P}(\varphi)$. Thus*

Ksat$_A$ is given as input

$$\begin{array}{lll} \mu = & \Box_1(\neg A_5 \vee A_4 \vee A_3) \wedge & \\ & \Box_1(\neg A_2 \vee A_1 \vee A_4) \ \wedge \Box_1(\neg A_2 \vee A_4 \vee A_5) \wedge & [\bigwedge_i \Box_1 \alpha_{1i}] \\ & \neg\Box_1(\neg A_3 \vee \neg A_1 \vee A_2) \ \wedge \neg\Box_1(A_4 \vee \neg A_2 \vee A_3) \wedge & [\bigwedge_j \neg\Box_1 \beta_{1j}] \\ & \Box_2(\neg A_4 \vee A_5 \vee A_2) & [\bigwedge_i \Box_2 \alpha_{2i}] \end{array}$$

and hence invokes Ksat$_{AR}$ *on the two restricted assignments:*

$$\begin{array}{lll} \mu^1 = & \Box_1(\neg A_5 \vee A_4 \vee A_3) \wedge & \\ & \Box_1(\neg A_2 \vee A_1 \vee A_4) \ \wedge \Box_1(\neg A_2 \vee A_4 \vee A_5) \wedge & [\bigwedge_i \Box_1 \alpha_{1i}] \\ & \neg\Box_1(\neg A_3 \vee \neg A_1 \vee A_2) \ \wedge \neg\Box_1(A_4 \vee \neg A_2 \vee A_3) & [\bigwedge_j \neg\Box_1 \beta_{1j}] \\ \mu^2 = & \Box_2(\neg A_4 \vee A_5 \vee A_2) & [\bigwedge_i \Box_2 \alpha_{2i}]. \end{array}$$

μ^2 is trivially K_m-satisfiable. μ^1 requires invoking Ksat *on the two formulas*

$$\begin{aligned} \varphi^{11} &= (\neg A_5 \vee A_4 \vee A_3) \ \wedge (\neg A_2 \vee A_1 \vee A_4) \wedge \\ &\quad (\neg A_2 \vee A_4 \vee A_5) \ \wedge A_3 \wedge A_1 \wedge \neg A_2, \\ \varphi^{12} &= (\neg A_5 \vee A_4 \vee A_3) \ \wedge (\neg A_2 \vee A_1 \vee A_4) \wedge \\ &\quad (\neg A_2 \vee A_4 \vee A_5) \ \wedge \neg A_4 \wedge A_2 \wedge \neg A_3. \end{aligned}$$

The latter is unsatisfiable, from which we can conclude that

$$\Box_1(\neg A_5 \vee A_4 \vee A_3) \wedge \Box_1(\neg A_2 \vee A_1 \vee A_4) \wedge \Box_1(\neg A_2 \vee A_4 \vee A_5) \wedge \neg\Box_1(A_4 \vee \neg A_2 \vee A_3)$$

is K_m-unsatisfiable, so that $\{B_6, B_8, B_2, \neg B_5,\}$ is a conflict set of μ^p.

The conflict set μ' found is then used to drive the backjumping mechanism of DPLL. Different strategies are possible. The DPLL-based modal tools [Hor98a, PS98] and earlier SMT tools [WW99] used to jump up to the most recent branching point s.t. at least one literal $l^p \in \mu'$ is not assigned. Intuitively, all open subbranches departing from the current branch at a lower decision point contain μ', so that there is no need to explore them; this allows for pruning all these subbranches from the search tree. (Notice that these strategies do not explicitly require adding the clause $\neg\mu'$ to φ.) More sophisticate versions of this technique, which mirror the most-modern backjumping techniques introduced in DPLL, were lately introduced in the context of SMT (see §26.4.3.5).

In substance, modal backjumping differs from standard Boolean backjumping only for the notion of conflict set used: whilst a Boolean conflict set μ is an assignment which causes a propositional inconsistency if conjoined to φ (i.e, s.t. $\mu \wedge \varphi \models_p \bot$), a modal conflict set is a set of literals which in K_m-inconsistent (i.e, s.t. $\mu \models \bot$).

25.4.1.5. Pure-literal filtering

This technique, which we call *pure-literal filtering*,[13] was implicitly proposed by [WW99] and then generalized and adopted in the *SAT tool [Tac99] (and lately imported into SMT [ABC$^+$02], see §26.4.3.7). The idea is that, if we have non-Boolean atoms occurring only positively [resp. negatively] in the input formula,

[13]Also called *triggering* in [WW99, ABC$^+$02].

we can safely drop every negative [resp. positive] occurrence of them from the assignment μ to be checked by $\textsc{Ksat}_A$. (The correctness and completeness of this process is a consequence of proposition 2 in §25.3.1.)

There are some benefits for this behavior. Let μ' be the reduced version of μ .

First, μ' might be K_m-satisfiable despite μ is K_m-unsatisfiable. If so, and if μ (and hence μ') propositionally satisfies φ, then $\textsc{Ksat}_F$ can stop, potentially saving a lot of search.

Second, if both μ' and μ are K_m-unsatisfiable, the call to $\textsc{Ksat}_A$ on μ' rather than that on μ can cause smaller conflict sets, in order to improve the effectiveness of backjumping and learning.

Third, checking the K_m-satisfiability of μ' rather than that of μ can be significantly faster. In fact, suppose $\Box_r\beta_{rj}$ occurs only positively in φ and it is assigned a negative value by $\textsc{Ksat}_F$, so that $\neg\Box_r\beta_{rj} \in \mu$ but $\neg\Box_r\beta_{rj} \notin \mu'$. Thus $\neg\Box_r\beta_{rj}$ will not occur in the restricted assignment μ^r fed to $\textsc{Ksat}_{AR}$, avoiding the call to $\textsc{Ksat}$ on $(\bigwedge_i \alpha_{ri} \wedge \neg\beta_{rj})$. This allows for extending the notion of "incrementality" of §25.4.1.2, by considering only the literals in μ' rather than those in μ.

25.4.2. Extensions to Non-Normal Modal Logics

This section briefly surveys some of the contents of [GGT01] to which we refer for further reading. Following the notation of [GGT01], we say that an assignment μ *satisfies* a formula φ if μ entails φ by propositional reasoning, and that a formula φ is *consistent* in a logic L (or L-*consistent*) if $\neg\varphi$ is not a theorem of L, i.e., if $\neg\varphi \notin$ L. Whether an assignment is consistent, depends on the particular classical modal logic L being considered. Furthermore, depending on the logic L considered, the consistency problem for L (i.e., determining whether a formula is consistent in L) belongs to different complexity classes. In particular, the consistency problem for E, EM, EN, EMN is NP-complete, while for EC, ECN, EMC it is PSPACE-complete (see [Var89, FHMV95]). Here, to save space, we divide these eight logics in two groups. We present the algorithms for checking the L-consistency of an assignment first in the case in which L is one of E, EM, EN, EMN, and then in the case in which L is one of the others.

25.4.2.1. Logics E, EM, EN, EMN

The following proposition is an easy consequence of the results presented in [Var89].

Proposition 5. *Let $\mu = \bigwedge_i \Box\alpha_i \wedge \bigwedge_j \neg\Box\beta_j \wedge \gamma$ be an assignment in which γ is a propositional formula. Let L be one of the logics E, EM, EN, EMN. μ is consistent in L if for each conjunct $\neg\Box\beta_j$ in μ one of the following conditions is satisfied:*

- *$(\alpha_i \equiv \neg\beta_j)$ is L-consistent for each conjunct $\Box\alpha_i$ in μ, and L=E;*
- *$(\alpha_i \wedge \neg\beta_j)$ is L-consistent for each conjunct $\Box\alpha_i$ in μ, and L=EM;*
- *$\neg\beta_j$ and $(\alpha_i \equiv \neg\beta_j)$ are L-consistent for each conjunct $\Box\alpha_i$ in μ, and L=EN;*
- *$\neg\beta_j$ and $(\alpha_i \wedge \neg\beta_j)$ are L-consistent for each conjunct $\Box\alpha_i$ in μ, and L=EMN.*

```
function Lsat_A($\mu$)
  foreach conjunct $\Box\beta_j$ do
    foreach conjunct $\Box\alpha_i$ do
      if $M[i,j] = Undef$ then $M[i,j] :=$ Lsat($\alpha_i \wedge \neg\beta_j$);
      if L $\in$ {EN,EMN} and $M[i,j] = True$ then $M[j,j] := True$;
      if L $\in$ {E,EN} and $M[i,j] = False$ then
        if $M[j,i] = Undef$ then $M[j,i] :=$ Lsat($\neg\alpha_i \wedge \beta_j$);
        if L = EN and $M[j,i] = True$ then $M[i,i] := True$;
        if $M[j,i] = False$ then return $False$
    end
    if L $\in$ {EN,EMN} then
      if $M[j,j] = Undef$ then $M[j,j] :=$ Lsat($\neg\beta_j$);
      if $M[j,j] = False$ then return $False$
  end;
  return $True$.
```

Figure 25.6. Lsat$_A$ for E, EM, EN, EMN

When implementing the above conditions, care must be taken in order to avoid repetitions of consistency checks. In fact, while an exponential number of assignments satisfying the input formula can be generated, at most n^2 checks are possible in L, where n is the number of “$\Box$” in the input formula. Given this upper bound, for each new consistency check, we can cache the result for a future possible re-utilization in a $n \times n$ matrix M. This ensures that at most n^2 consistency checks will be performed. In more detail, given an enumeration $\varphi_1, \varphi_2, \ldots, \varphi_n$ of the boxed subformulas of the input formula, M[i,j], with $i \neq j$, stores the result of the consistency check for $(\varphi_i \wedge \neg\varphi_j)$. M[i,i] stores the result of the consistency check for $\neg\varphi_i$. Initially, each element of the matrix M has value *Undef* (meaning that the corresponding test has not been done yet). The result is the procedure Lsat$_A$ in Figure 25.6, where the procedure Lsat is identical to the procedure Ksat modulo the call to Ksat$_A$ which must be replaced by Lsat$_A$.

Consider Figure 25.6 and assume that L=E or L=EN. Given a pair of conjuncts $\Box\alpha_i$ and $\neg\Box\beta_j$, we split the consistency test for $(\alpha_i \equiv \neg\beta_j)$ in two simpler sub-tests:

- first, we test whether $(\alpha_i \wedge \neg\beta_j)$ is consistent, and
- only if this test gives *False*, we test whether $(\neg\alpha_i \wedge \beta_j)$ is consistent.

Notice also that, in case L=EN or L=EMN, if we know that, e.g., $(\alpha_i \wedge \neg\beta_j)$ is consistent, then also $\neg\beta_j$ is consistent and we store this result in M[j,j]. The following proposition ensures the correctness of Lsat in the case of E, EM, EN and EMN.

Proposition 6. *Let $\mu = \bigwedge_i \Box\alpha_i \wedge \bigwedge_j \neg\Box\beta_j \wedge \gamma$ be an assignment in which γ is a propositional formula. Let L be one of the logics E, EM, EN, EMN. Assume that, for any formula φ whose depth is less than the depth of μ,* Lsat*(φ)*

- *returns True if φ is L-consistent, and*
- *False otherwise.*

```
function LSAT_A($\bigwedge_i \Box\alpha_i \wedge \bigwedge_j \neg\Box\beta_j \wedge \gamma$)
  $\Delta := \{\alpha_i \mid \Box\alpha_i$ is a conjunct of $\mu\}$;
  foreach conjunct $\Box\beta_j$ do
    $\Delta' := \Delta$;
    if L $\in$ {EC, ECN} then
      foreach conjunct $\Box\alpha_i$ do
        if $M[j,i] = Undef$ then $M[j,i] :=$ LSAT($\neg\alpha_i \wedge \beta_j$);
        if $M[j,i] = True$ then $\Delta' = \Delta' \setminus \{\alpha_i\}$
      end;
    if L $\in$ {ECN} or $\Delta' \neq \emptyset$ then
      if not LSAT($\bigwedge_{\alpha_i \in \Delta'} \alpha_i \wedge \neg\beta_j$) then return False
  end;
  return True.
```

Figure 25.7. LSAT$_A$ for EC, ECN, EMC

LSAT$_A$*(μ) returns True if μ is L-consistent, and False otherwise.*

25.4.2.2. Logics EC, ECN, EMC

The following proposition is an easy consequence of the results presented in [Var89].

Proposition 7. *Let $\mu = \bigwedge_i \Box\alpha_i \wedge \bigwedge_j \neg\Box\beta_j \wedge \gamma$ be an assignment in which γ is a propositional formula. Let Δ be the set of formulas α_i such that $\Box\alpha_i$ is a conjunct of μ. Let L be one of logics EC, ECN, EMC. μ is consistent in L if for each conjunct $\neg\Box\beta_j$ in μ one of the following conditions is satisfied:*

- *$((\bigwedge_{\alpha_i \in \Delta'} \alpha_i) \equiv \neg\beta_j)$ is L-consistent for each non empty subset Δ' of Δ, and L=EC;*
- *$((\bigwedge_{\alpha_i \in \Delta'} \alpha_i) \equiv \neg\beta_j)$ is L-consistent for each subset Δ' of Δ, and L=ECN;*
- *Δ is empty or $((\bigwedge_{\alpha_i \in \Delta} \alpha_i) \wedge \neg\beta_j)$ is L-consistent, and L=EMC;*

Assume that L=EC or L=ECN. The straightforward implementation of the corresponding condition may lead to an exponential number of checks in the cardinality $|\Delta|$ of Δ. More carefully, for each conjunct $\neg\Box\beta_j$ in μ, we can perform at most $|\Delta| + 1$ checks if

1. for each formula α_i in Δ, we first check whether $(\neg\alpha_i \wedge \beta_j)$ is consistent in L. Let Δ' be the set of formulas for which the above test fails. Then,
2. in case L=ECN or $\Delta' \neq \emptyset$, we perform the last test, checking whether $((\bigwedge_{\alpha_i \in \Delta'} \alpha_i) \wedge \neg\beta_j)$ is consistent in L.

Furthermore, the result of the consistency checks performed in the first step can be cached in a matrix M analogous to the one used in the previous subsection.

If L=EC or L=ECN, the procedure LSAT$_A$ in Figure 25.7 implements the above ideas. Otherwise, it is a straightforward implementation of the conditions in proposition 7. The following proposition ensures the correctness of LSAT in the case of E, EM, EN and EMN.

Proposition 8. *Let $\mu = \bigwedge_i \Box\alpha_i \wedge \bigwedge_j \neg\Box\beta_j \wedge \gamma$ be an assignment in which γ is a propositional formula. Let L be one of logics EC, ECN, EMC. Assume that, for any formula φ whose depth is less than the depth of μ, $Lsat(\varphi)$*

- *returns True if φ is L-consistent, and*
- *False otherwise.*

$\textsc{Lsat}_A(\mu)$ *returns True if μ is L-consistent, and False otherwise.*

25.5. The OBDD-based Approach

In this section we briefly survey the basics of the OBDD-based approach to implement decision procedures for modal K, and we refer the reader to [PSV02, PV03] for further details. The contents of this section and the next borrow from [PSV06], including basic notation and the description of the algorithms.

The OBDD-based approach is inspired by the automata-theoretic approach for logics with the tree-model -property. In that approach, one proceeds in two steps. First, an input formula is translated to a tree automaton that accepts all the tree models of the formula. Second, the automaton is tested for non-emptiness, i.e., whether it accepts some tree. The approach described in [PSV02] combines the two steps and carries out the non-emptiness test without explicitly constructing the automaton. The logic K is simple enough that the automaton's non-emptiness test consists of a single fixpoint computation, which starts with a set of states and then repeatedly applies a monotone operator until a fixpoint is reached. In the automaton that corresponds to a formula, each state is a *type*, i.e., a set of formulas satisfying some consistency conditions. The algorithms that we describe here start from some set of types and then repeatedly apply a monotone operator until a fixpoint is reached.

25.5.1. Basics

To aid the description of the OBDD-based algorithms we introduce some additional notation. The set of propositional atoms used in a formula is denoted $AP(\varphi)$, and, given a formula ψ, we call its set of subformulas $\mathsf{sub}(\psi)$. For $\varphi \in \mathsf{sub}(\psi)$, we can define $\mathsf{depth}(\varphi)$ in the usual way. If not stated otherwise, we assume all formulas to be in BNF. The *closure* of a formula $\mathsf{cl}(\psi)$ is defined as the smallest set such that, for all subformulas φ of ψ, if φ is not of the form $\neg\varphi'$, then $\{\varphi, \neg\varphi\} \subseteq \mathsf{cl}(\psi)$. The algorithms that we present here work on *types*, i.e., maximal sets of formulas that are consistent w.r.t. the Boolean operators, and where (negated) box formulas are treated as atoms. A set of formulas $a \subseteq \mathsf{cl}(\psi)$ is called a *ψ-type* (or simply a type if ψ is clear from the context) if it satisfies the following conditions:

- If $\varphi = \neg\varphi'$, then $\varphi \in a$ iff $\varphi' \notin a$.
- If $\varphi = \varphi' \wedge \varphi''$, then $\varphi \in a$ iff $\varphi' \in a$ and $\varphi'' \in a$.
- If $\varphi = \varphi' \vee \varphi''$, then $\varphi \in a$ iff $\varphi' \in a$ or $\varphi'' \in a$.

For a set of types T, we define the maximal accessibility relation $\Delta \subseteq T \times T$ as follows.

$$\Delta(t, t') \text{ iff for all } \Box\varphi' \in t, \text{ we have } \varphi' \in t'$$

```
$X := \mathsf{Init}(\psi)$
repeat
  $X' := X$
  $X := \mathsf{Update}(X')$
until $X = X'$
if exists $x \in X$ such that $\psi \in x$ then
  return "$\psi$ is satisfiable"
else
  return "$\psi$ is not satisfiable"
```

Figure 25.8. Basic schema for the OBDD-based algorithm.

In Figure 25.8 we present the basic schema for the OBDD-based decision procedures. The schema can be made to work in two fashions, called top-down and bottom-up in [PSV06], according to the definition of the accessory functions Init and Update. In both cases, since the algorithms operate with elements in a finite lattice $2^{\mathsf{cl}(\psi)}$ and use a monotone Update, they are bound to terminate. In the case of the top-down approach, the accessory functions are defined as:

- $\mathsf{Init}(\psi)$ is the set of all ψ-types.
- $\mathsf{Update}(T) := T \setminus \mathsf{bad}(T)$, where $\mathsf{bad}(T)$ are the types in T that contain unwitnessed negated box formulas. More precisely,

$$\begin{aligned}\mathsf{bad}(T) := \{t \in T \mid\ & \text{there exists } \neg\Box\varphi \in t \text{ and,}\\ & \text{forall } u \in T \text{ with } \Delta(t,u), \text{ we have } \varphi \in u\}.\end{aligned}$$

Intuitively, the top-down algorithm starts with the set of *all* types and remove those types with "possibilities" $\Diamond\varphi$ for which no "witness" can be found. In the bottom-up approach, the accessory functions are defined as:

- $\mathsf{Init}(\psi)$ is the set of all those types that do not require any witness, which means that they do not contain any negated box formula, or equivalently, that they contain all positive box formulas in $\mathsf{cl}(\psi)$. More precisely,

$$\mathsf{Init}(\psi) := \{t \subseteq \mathsf{cl}(\psi) \mid \text{t is a type and } \Box\varphi \in t \text{ for each } \Box\varphi \in \mathsf{cl}(\psi)\}.$$

- $\mathsf{Update}(T) := T \cup \mathsf{supp}(T)$, where $\mathsf{supp}(T)$ is the set of those types whose negated box formulas are witnessed by types in T. More precisely,

$$\begin{aligned}\mathsf{supp}(T) := \{t \subseteq \mathsf{cl}(\psi) \mid\ & t \text{ is a type and,}\\ & \text{for all } \neg\Box\varphi \in t, \text{ there exists } u \in T\\ & \text{with } \neg\varphi \in u \text{ and } \Delta(t,u)\}.\end{aligned}$$

Intuitively, the bottom-up algorithm starts with the set of types having no possibilities $\Diamond\varphi$, and adds those types whose possibilities are witnessed by a type in the set. Notice that the two algorithms described above, correspond to the two ways in which non-emptiness can be tested for automata for K.

25.5.2. Optimizations

The decision procedure described in the previous section handle the formula in three steps. First, the formula is converted into BNF. Then the initial set of types is generated – we can think of this set as having some memory efficient representation. Finally, this set is updated through a fixpoint process. The answer of the decision procedure depends on a simple syntactic check of this fixpoint. In the following we consider three orthogonal optimizations techniques. See [PSV06] for more details, and for a description of preprocessing techniques that may further improve the performances of the OBDD-based implementations.

25.5.2.1. Particles

The approaches presented so far strongly depend on the fact that the BNF is used and they can be said to be redundant: if a type contains two conjuncts of some subformula of the input, then it also contains the corresponding conjunction – although the truth value of the latter is determined by the truth values of the former. Working with a different normal form it is possible to reduce such redundancy. We consider K-formulas in NNF (negation normal form) and we assume hereafter that all the formulas are in NNF. A set $p \subseteq \mathsf{sub}(\psi)$ is a ψ-*particle* if it satisfies the following conditions:

- If $\varphi = \neg\varphi'$, then $\varphi \in p$ implies $\varphi' \notin p$
- If $\varphi = \varphi' \wedge \varphi''$, then $\varphi \in p$ implies $\varphi' \in p$ and $\varphi'' \in p$.
- If $\varphi = \varphi' \vee \varphi''$, then $\varphi \in p$ implies $\varphi' \in p$ or $\varphi'' \in p$.

Thus, in contrast to a type, a particle may contain both φ' and φ'', but neither $\varphi' \wedge \varphi''$ nor $\varphi' \vee \varphi''$. Incidentally, particles are closer than types to assignments over modal atoms as described in Section 25.3.4. For particles, $\Delta(\cdot,\cdot)$ is defined as types. From a set of particles P and the corresponding $\Delta(\cdot,\cdot)$, a Kripke structure K_p can be constructed in the same way as from a set of types (see [PSV06]).

The schema presented in Figure 25.8 can be made to work for particles as well. In the top-down algorithm:

- $\mathsf{Init}(\psi)$ is the set of all ψ-particles.
- $\mathsf{Update}(P) := P \setminus \mathsf{bad}(P)$, where $\mathsf{bad}(P)$ is the particles in P that contain unwitnessed diamond formulas and it is defined similarly to the case of types

Also in the case of particles, the bottom-up approach differs only for the definitions of Init and Update:

- $\mathsf{Init}(\psi) := \{p \subseteq \mathsf{sub}(\psi) \mid p$ is a particle and $\Diamond\varphi \notin p$ for all $\Diamond\varphi \in \mathsf{sub}(\psi)\}$ is the set of ψ-particles p that do not contain diamond formulas.
- $\mathsf{Update}(P) := P \cup \mathsf{supp}(P)$ where $\mathsf{supp}(P)$ is the set of witnessed particles defined similarly to witnessed types.

Just like a set of types can be encoded in some efficient way, e.g., a set of bit vectors using a BDD, the same can be done for particles. It is easy to see that bit vectors for particles may be longer than bit vectors for types because, for example, the input may involve subformulas $\Box A$ and $\Diamond\neg A$. The overall size of the BDD may, however, be smaller for particles since particles impose fewer constraints than types, and improvements in the run time of the algorithms may result because the particle-based Update functions require checking less formulas than the type-based ones.

25.5.2.2. Lean approaches

Even though the particle approach imposes less constraints than the type approach, it still involves redundant information: like types, particles may contain both a conjunction and the corresponding conjuncts. To further reduce the size of the corresponding BDDs, in [PSV06] it is proposed a representation where "non-redundant" subformulas are only kept track of. A set of "non-redundant" subformulas $\mathsf{atom}(\psi)$ is defined as the set of those formulas in $\mathsf{cl}(\psi)$ that are neither conjunctions nor disjunctions, i.e., each $\varphi \in \mathsf{atom}(\psi)$ is of the form $\Box\varphi'$, A, $\neg\Box\varphi'$, or $\neg A$. By definition of types, each ψ-type $t \subseteq \mathsf{cl}(\psi)$, corresponds one-to-one to a *lean type* $\mathsf{lean}(t) := t \cap \mathsf{atom}(\psi)$. To specify algorithms for lean types, a relation $\dot\in$ must be defined recursively as follows: $\varphi \,\dot\in\, t$ if

- $\varphi \in \mathsf{atom}(\psi)$ and $\varphi \in t$,
- $\varphi = \neg\varphi'$ and not $\varphi \,\dot\in\, t$,
- $\varphi = \varphi' \wedge \varphi''$, $\varphi' \,\dot\in\, t$, and $\varphi'' \,\dot\in\, t$, or
- $\varphi = \varphi' \vee \varphi''$, and $\varphi' \,\dot\in\, t$, or $\varphi'' \,\dot\in\, t$.

The top-down and bottom-up approach for types can be easily modified to work for lean types. It suffices to modify the definition of the functions bad and supp as follows:

$$\begin{aligned}\mathsf{bad}(T) := \{t \in T \mid\ & \text{there exists } \neg\Box\varphi \in t \text{ and,}\\ & \text{forall } u \in T \text{ with } \Delta(t,u), \text{ we have } \varphi \,\dot\in\, u\}.\end{aligned}$$

$$\begin{aligned}\mathsf{supp}(T) := \{t \subseteq \mathsf{cl}(\psi) \mid\ & t \text{ is a type and,}\\ & \text{for all } \neg\Box\varphi \in t, \text{ there exists } u \in T\\ & \text{with } \neg\varphi \,\dot\in\, u \text{ and } \Delta(t,u)\}.\end{aligned}$$

A lean optimization can also be defined for particles – details are given in [PSV06]. Notice that this approach bears also some resemblances with the approach used in [CGH97] to translate LTL to SMV.

25.5.2.3. Level based evaluation

Another variation of the basic algorithm presented in Figure 25.8 exploits the fact that K enjoys the finite-tree-model property, i.e., each satisfiable formula ψ of K has a finite tree model of depth bounded by the depth of nested modal operators $\mathsf{depth}(\psi)$ of ψ. We can think of such a model as being partitioned into *layers*, where all states that are at distance i from the root are said to be in layer i. Instead of representing a complete model using a set of particles or types, each layer in the model can be represented using a separate set. Since only a subset of all subformulas appears in one layer, the representation can be more compact.

We start by (re)defining $\mathsf{cl}(\cdot)$ as

$$\mathsf{cl}_i(\psi) := \{\varphi \in \mathsf{cl}(\psi) \mid \varphi \text{ occurs at modal depth } i \text{ in } \psi\}$$

and $\Delta(\cdot,\cdot)$ as

$$\Delta(t,t') \text{ iff for all } t \subseteq \mathsf{cl}_i(\psi),\ t' \subseteq \mathsf{cl}_{i+1}(\psi), \text{ and } \varphi' \in t' \text{ for all } \Box\varphi' \in t.$$

```
$d := \mathsf{depth}(\psi)$
$X_d := \mathsf{Init}_d(\psi)$
**for** $i := d - 1$ **downto** 0 **do**
  $X_i := \mathsf{Update}(X_{i+1}, i)$
**end**
**if** exists $x \in X_0$ such that $\psi \in x$ **then**
  **return** "$\psi$ is satisfiable"
**else**
  **return** "$\psi$ is not satisfiable"
```

Figure 25.9. Algorithm for the level-based optimization.

in order to adapt them to the layered approach. A sequence of sets of types $T = \langle T_0, T_1, \ldots, T_d\rangle$ with $T_i \subseteq 2^{\mathsf{cl}_i(\psi)}$ can still be converted into a tree Kripke structure (see [PSV06] for details).

A bottom-up algorithm for level-based evaluation can be defined as in Figure 25.9. The algorithm works bottom-up in the sense that it starts with the leaves of a tree model *at the deepest level* and then moves up the tree model toward the root, adding nodes that are "witnessed". In contrast, the bottom-up approach presented earlier starts with *all* leaves of a tree model. The accessory functions can be defined as follows:

- $\mathsf{Init}_i(\psi) := \{t \subseteq \mathsf{cl}_i(\psi) \mid t \text{ is a type } \}$.
- $\mathsf{Update}(T, i) := \{t \in \mathsf{Init}_i(\psi) \mid \text{for all } \neg\Box\varphi \in t \text{ there exists } u \in T \text{ with } \neg\varphi \in u \text{ and } \Delta_i(t, u)\}$.

For a set T of types of formulas at level $i + 1$, $\mathsf{Update}(T, i)$ represents all types of formulas at level i that are witnessed in T.

25.6. The Eager DPLL-based approach

Recently [SV06, SV08] have explored the idea of encoding $K_m/\mathcal{ALC}$-satisfiability into SAT and handle it by state-of-the-art SAT tools. A satisfiability-preserving encoding from $K_m/\mathcal{ALC}$ to SAT was proposed there, with a few variations and some important optimizations. As K_m-satisfiability is PSPACE-complete, the encoding is necessarily worst-case exponential (unless PSPACE=NP). However, the only source of exponentiality is the modal depth of the input formula: if the depth is bounded, the problem is NP-complete [Hal95], so that the encoding reduces to polynomial. In practice, the experiments presented there showed that this approach can handle most or all the problems which are at the reach of the other approaches, with performances which are comparable with, or even better than, those of the current state-of-the-art tools.

As this idea was inspired by the so-called "eager approach" to SMT [BGV99, SSB02, Str02] (see §26.3), we call this approach, *eager approach to modal reasoning*. In this section we present an overview of this approach.

25.6.1. The basic encoding

In order to make our presentation more uniform, and to avoid considering the polarity of subformulas, we adopt from [Fit83, Mas00] the representation of K_m-formulas from the following table:

α	α_1	α_2	β	β_1	β_2	π^r	π_0^r	ν^r	ν_0^r
$(\varphi_1 \wedge \varphi_2)$	φ_1	φ_2	$(\varphi_1 \vee \varphi_2)$	φ_1	φ_2	$\Diamond_r\varphi_1$	φ_1	$\Box_r\varphi_1$	φ_1
$\neg(\varphi_1 \vee \varphi_2)$	$\neg\varphi_1$	$\neg\varphi_2$	$\neg(\varphi_1 \wedge \varphi_2)$	$\neg\varphi_1$	$\neg\varphi_2$	$\neg\Box_r\varphi_1$	$\neg\varphi_1$	$\neg\Diamond_r\varphi_1$	$\neg\varphi_1$
$\neg(\varphi_1 \rightarrow \varphi_2)$	φ_1	$\neg\varphi_2$	$(\varphi_1 \rightarrow \varphi_2)$	$\neg\varphi_1$	φ_2				

in which non-literal K_m-formulas are grouped into four categories: α's (conjunctive), β's (disjunctive), π's (existential), ν's (universal). All such formulas occur in the main formula with positive polarity only. [14] This allows for disregarding the issue of polarity of subformulas.

We borrow some notation from the *Single Step Tableau (SST)* framework [Mas00, DM00]. We represent univocally states in $\mathcal{M}$ as labels σ, represented as non empty sequences of integers $1.n_1^{r_1}.n_2^{r_2}. \ldots .n_k^{r_k}$, s.t. the label 1 represents the root state, and $\sigma.n^r$ represents the n-th successor of σ through the relation $\mathcal{R}_r$. With a little abuse of notation, hereafter we may say "a state σ" meaning "a state labeled by σ". We call a *labeled formula* a pair $\langle\sigma : \psi\rangle$, s.t. σ is a state label and ψ is a K_m-formula.

Let $A_{[,\ ]}$ be an *injective* function which maps a labeled formula $\langle\sigma : \psi\rangle$, s.t. is not in the form $\neg\phi$, into a Boolean variable $A_{[\sigma,\ \psi]}$. Let $L_{[\sigma,\ \psi]}$ denote $\neg A_{[\sigma,\ \phi]}$ if ψ is in the form $\neg\phi$, $A_{[\sigma,\ \psi]}$ otherwise. Given a K_m-formula φ, $K_m 2SAT$ builds a Boolean CNF formula recursively as follows:

$$K_m2SAT(\varphi) := A_{[1,\ \varphi]} \wedge Def(1,\ \varphi) \quad (25.21)$$
$$Def(\sigma,\ A_i), := \top \quad (25.22)$$
$$Def(\sigma,\ \neg A_i) := \top \quad (25.23)$$
$$Def(\sigma,\ \alpha) := (L_{[\sigma,\ \alpha]} \rightarrow (L_{[\sigma,\ \alpha_1]} \wedge L_{[\sigma,\ \alpha_2]})) \wedge Def(\sigma,\ \alpha_1) \wedge Def(\sigma,\ \alpha_2) \quad (25.24)$$
$$Def(\sigma,\ \beta) := (L_{[\sigma,\ \beta]} \rightarrow (L_{[\sigma,\ \beta_1]} \vee L_{[\sigma,\ \beta_2]})) \wedge Def(\sigma,\ \beta_1) \wedge Def(\sigma,\ \beta_2) \quad (25.25)$$
$$Def(\sigma,\ \pi^{r,j}) := (L_{[\sigma,\ \pi^{r,j}]} \rightarrow L_{[\sigma.j,\ \pi_0^{r,j}]}) \wedge Def(\sigma.j,\ \pi_0^{r,j}) \quad (25.26)$$
$$Def(\sigma,\ \nu^r) := \bigwedge_{\langle\sigma:\pi^{r,i}\rangle} ((L_{[\sigma,\ \nu^r]} \wedge L_{[\sigma,\ \pi^{r,i}]}) \rightarrow L_{[\sigma.i,\ \nu_0^r]}) \ \wedge \bigwedge_{\langle\sigma:\pi^{r,i}\rangle} Def(\sigma.i,\ \nu_0^r). \quad (25.27)$$

Here by "$\langle\sigma : \pi^{r,i}\rangle$" we mean that $\pi^{r,i}$ is the j-th distinct π^r formula labeled by σ.

We assume that the K_m-formulas are represented as DAGs, so to avoid the expansion of the same $Def(\sigma,\ \psi)$ more than once. Moreover, following [Mas00], we assume that, for each σ, the $Def(\sigma,\ \psi)$'s are expanded in the order: α, β, π, ν. Thus, each $Def(\sigma,\ \nu^r)$ is expanded after the expansion of all $Def(\sigma,\ \pi^{r,i})$'s, so that $Def(\sigma,\ \nu^r)$ will generate one clause $((L_{[\sigma,\ \pi^{r,i}]} \wedge L_{[\sigma,\ \Box_r\nu_0^r]}) \rightarrow L_{[\sigma.i,\ \nu_0^r]})$ and one novel definition $Def(\sigma.i,\ \nu_0^r)$ for each $Def(\sigma,\ \pi^{r,i})$ expanded. [15]

Intuitively, $K_m2SAT(\varphi)$ mimics the construction of an SST tableau expansion [Mas00, DM00] , s.t., if there exists an open tableau $\mathcal{T}$ for $\langle 1 : \varphi\rangle$, then

[14] E.g., a $\wedge$-formula [resp. $\vee$-formula] occurring negatively is considered a positive occurrence of a β-formula [resp. an α-formula]; a $\Box_r$-formula [resp. a $\Diamond_r$-formula] occurring negatively is considered a positive occurrence of a π-formula [resp. a ν-formula].

[15] Notice that, e.g., an occurrence of $\Box_r\psi$ is considered a ν-formula if positive, a π-formula if negative.

there exists a total truth assignment μ which satisfies $K_m2SAT(\varphi)$, and vice versa. Thus, from the correctness and completeness of the SST framework, we have the following fact.

Theorem 2. *[SV08] A K_m-formula φ is K_m-satisfiable if and only if the corresponding Boolean formula $K_m2SAT(\varphi)$ is satisfiable.*

Notice that, due to (25.27), the number of variables and clauses in $K_m2SAT(\varphi)$ may grow exponentially with $depth(\varphi)$. This is in accordance to what stated in [Hal95].

Example 9 (NNF)**.** *Let φ_{nnf} be $(\Diamond A_1 \vee \Diamond(A_2 \vee A_3)) \wedge \Box\neg A_1 \wedge \Box\neg A_2 \wedge \Box\neg A_3$.* [16] *It is easy to see that φ_{nnf} is K_1-unsatisfiable: the $\Diamond$-atoms impose that at least one atom A_i is true in at least one successor of the root state, whilst the $\Box$-atoms impose that all atoms A_i are false in all successor states of the root state. $K_m2SAT(\varphi_{nnf})$ is:*

1. $A_{[1,\ \varphi_{nnf}]}$
2. $\wedge(\ A_{[1,\ \varphi_{nnf}]} \rightarrow (A_{[1,\ \Diamond A_1 \vee \Diamond(A_2 \vee A_3)]} \wedge A_{[1,\ \Box\neg A_1]} \wedge A_{[1,\ \Box\neg A_2]} \wedge A_{[1,\ \Box\neg A_3]})\ )$
3. $\wedge(\ A_{[1,\ \Diamond A_1 \vee \Diamond(A_2 \vee A_3)]} \rightarrow (A_{[1,\ \Diamond A_1]} \vee A_{[1,\ \Diamond(A_2 \vee A_3)]})\ )$
4. $\wedge(\ A_{[1,\ \Diamond A_1]} \rightarrow A_{[1.1,\ A_1]}\ )$
5. $\wedge(\ A_{[1,\ \Diamond(A_2 \vee A_3)]} \rightarrow A_{[1.2,\ A_2 \vee A_3]}\ )$
6. $\wedge(\ (A_{[1,\ \Box\neg A_1]} \wedge A_{[1,\ \Diamond A_1]}) \rightarrow \neg A_{[1.1,\ A_1]}\ )$
7. $\wedge(\ (A_{[1,\ \Box\neg A_2]} \wedge A_{[1,\ \Diamond A_1]}) \rightarrow \neg A_{[1.1,\ A_2]}\ )$
8. $\wedge(\ (A_{[1,\ \Box\neg A_3]} \wedge A_{[1,\ \Diamond A_1]}) \rightarrow \neg A_{[1.1,\ A_3]}\ )$
9. $\wedge(\ (A_{[1,\ \Box\neg A_1]} \wedge A_{[1,\ \Diamond(A_2 \vee A_3)]}) \rightarrow \neg A_{[1.2,\ A_1]}\ )$
10. $\wedge(\ (A_{[1,\ \Box\neg A_2]} \wedge A_{[1,\ \Diamond(A_2 \vee A_3)]}) \rightarrow \neg A_{[1.2,\ A_2]}\ )$
11. $\wedge(\ (A_{[1,\ \Box\neg A_3]} \wedge A_{[1,\ \Diamond(A_2 \vee A_3)]}) \rightarrow \neg A_{[1.2,\ A_3]}\ )$
12. $\wedge(\ A_{[1.2,\ A_2 \vee A_3]} \rightarrow (A_{[1.2,\ A_2]} \vee A_{[1.2,\ A_3]})\ )$

After a run of BCP, 3. reduces to the implicate disjunction $A_{[1, \Diamond A_1]} \vee A_{[1, \Diamond(A_2 \vee A_3)]}$. If the first element $A_{[1,\ \Diamond A_1]}$ is assigned to true, then by BCP we have a conflict on 4. and 6. If $A_{[1,\ \Diamond A_1]}$ is set to false, then the second element $A_{[1,\ \Diamond(A_2 \vee A_3)]}$ is assigned to true, and by BCP we have a conflict on 12. Thus $K_m2SAT(\varphi_{nnf})$ is unsatisfiable.

25.6.2. Optimizations

The following optimizations of the encoding have been proposed in [SV06, SV08] in order to reduce the size of the output propositional formula.

25.6.2.1. Pre-conversion to BNF

Before the encoding, some potentially useful preprocessing on the input formula can be performed. First, the input K_m-formulas can be converted into BNF. One potential advantage is that, when one $\Box_r\psi$ occurs both positively and negatively (like, e.g., in $(\Box_r\psi \vee ...) \wedge (\neg\Box_r\psi \vee ...) \wedge ...$), then both occurrences of $\Box_r\psi$ are labeled by the same Boolean atom $A_{[\sigma,\ \Box_r\psi]}$, and hence they are always assigned the

[16]For K_1-formulas, we omit the box and diamond indexes.

same truth value by DPLL; with NNF, instead, the negative occurrence $\neg\Box_r\psi$ is rewritten into $\Diamond_r\neg\psi$, so that two distinct Boolean atoms $A_{[\sigma,\ \Box_r\psi]}$ and $A_{[\sigma,\ \Diamond_r\neg\psi]}$ are generated; DPLL can assign them the same truth value, creating a hidden conflict which may require some extra Boolean search to reveal.

Example 10 (BNF)**.** *We consider the BNF variant of the φ_{nnf} formula of Example 9, $\varphi_{bnf} = (\neg\Box\neg A_1 \vee \neg\Box(\neg A_2 \wedge \neg A_3)) \wedge \Box\neg A_1 \wedge \Box\neg A_2 \wedge \Box\neg A_3$. As before, it is easy to see that φ_{bnf} is K_1-unsatisfiable. $K_m2SAT(\varphi_{bnf})$ is:*

$$\begin{array}{rl}
1. & \quad A_{[1,\ \varphi_{bnf}]} \\
2.\ \wedge & (\ A_{[1,\ \varphi_{bnf}]} \\
 & \quad \rightarrow (A_{[1,\ (\neg\Box\neg A_1\vee\neg\Box(\neg A_2\wedge\neg A_3))]} \wedge A_{[1,\ \Box\neg A_1]} \wedge A_{[1,\ \Box\neg A_2]} \wedge A_{[1,\ \Box\neg A_3]})\ ) \\
3.\ \wedge & (\ A_{[1,\ (\neg\Box\neg A_1\vee\neg\Box(\neg A_2\wedge\neg A_3))]} \rightarrow (\neg A_{[1,\ \Box\neg A_1]} \vee \neg A_{[1,\ \Box(\neg A_2\wedge\neg A_3)]})\ ) \\
4.\ \wedge & (\ \neg A_{[1,\ \Box\neg A_1]} \rightarrow A_{[1.1,\ A_1]}\ ) \\
5.\ \wedge & (\ \neg A_{[1,\ \Box(\neg A_2\wedge\neg A_3)]} \rightarrow \neg A_{[1.2,\ (\neg A_2\wedge\neg A_3)]}\ ) \\
6.\ \wedge & (\ (A_{[1,\ \Box\neg A_1]} \wedge \neg A_{[1,\ \Box\neg A_1]}) \rightarrow \neg A_{[1.1,\ A_1]}\ ) \\
7.\ \wedge & (\ (A_{[1,\ \Box\neg A_2]} \wedge \neg A_{[1,\ \Box\neg A_1]}) \rightarrow \neg A_{[1.1,\ A_2]}\ ) \\
8.\ \wedge & (\ (A_{[1,\ \Box\neg A_3]} \wedge \neg A_{[1,\ \Box\neg A_1]}) \rightarrow \neg A_{[1.1,\ A_3]}\ ) \\
9.\ \wedge & (\ (A_{[1,\ \Box\neg A_1]} \wedge \neg A_{[1,\ \Box(\neg A_2\wedge\neg A_3)]}) \rightarrow \neg A_{[1.2,\ A_1]}\ ) \\
10.\ \wedge & (\ (A_{[1,\ \Box\neg A_2]} \wedge \neg A_{[1,\ \Box(\neg A_2\wedge\neg A_3)]}) \rightarrow \neg A_{[1.2,\ A_2]}\ ) \\
11.\ \wedge & (\ (A_{[1,\ \Box\neg A_3]} \wedge \neg A_{[1,\ \Box(\neg A_2\wedge\neg A_3)]}) \rightarrow \neg A_{[1.2,\ A_3]}\ ) \\
12.\ \wedge & (\ \neg A_{[1.2,\ (\neg A_2\wedge\neg A_3)]} \rightarrow (A_{[1.2,\ A_2]} \vee A_{[1.2,\ A_3]})\ )
\end{array}$$

Unlike with NNF, $K_m2SAT(\varphi_{bnf})$ is found unsatisfiable directly by BCP. In fact, the unit-propagation of $A_{[1,\ \Box\neg A_1]}$ from 2. causes $\neg A_{[1,\ \Box\neg A_1]}$ in 3. to be false, so that one of the two (unsatisfiable) branches induced by the disjunction is cut a priori. With NNF, the corresponding atoms $A_{[1,\ \Box\neg A_1]}$ and $A_{[1,\ \Diamond A_1]}$ are not recognized to be one the negation of the other, s.t. DPLL may need exploring one Boolean branch more.

25.6.2.2. Lifting boxes and diamonds

The second form of preprocessing is, the K_m-formula can also be rewritten by recursively applying the K_m-validity-preserving “box/diamond-lifting rules”:

$$\begin{array}{ll}
(\ \Box_r\varphi_1 \wedge\ \Box_r\varphi_2) \Longrightarrow\ \Box_r(\varphi_1\wedge\varphi_2), & (\ \Diamond_r\varphi_1 \vee\ \Diamond_r\varphi_2) \Longrightarrow\ \Diamond_r(\varphi_1\vee\varphi_2), \\
(\neg\Box_r\varphi_1 \vee \neg\Box_r\varphi_2) \Longrightarrow \neg\Box_r(\varphi_1\wedge\varphi_2), & (\neg\Diamond_r\varphi_1 \wedge \neg\Diamond_r\varphi_2) \Longrightarrow \neg\Diamond_r(\varphi_1\vee\varphi_2).
\end{array} \tag{25.28}$$

This has the potential benefit of reducing the number of $\pi^{r,i}$ formulas, and hence the number of labels $\sigma.i$ to take into account in the expansion of the $Def(\sigma,\ \nu^r)$’s (25.27).

Example 11 (BNF with LIFT)**.** *If we apply the rules (25.28) to the formula of Example 10, then we have $\varphi_{bnflift} = \neg\Box(\neg A_1\wedge\neg A_2\wedge\neg A_3) \wedge \Box(\neg A_1\wedge\neg A_2\wedge\neg A_3)$. $K_m2SAT(\varphi_{bnflift})$ is thus:*

$$\begin{array}{rl}
1. & \quad A_{[1,\ \varphi_{bnflift}]} \\
2.\ \wedge & (\ A_{[1,\ \varphi_{bnflift}]} \rightarrow (\neg A_{[1,\ \Box(\neg A_1\wedge\neg A_2\wedge\neg A_3)]} \wedge A_{[1,\ \Box(\neg A_1\wedge\neg A_2\wedge\neg A_3)]})\ ) \\
3.\ \wedge & (\ \neg A_{[1,\ \Box(\neg A_1\wedge\neg A_2\wedge\neg A_3)]} \rightarrow \neg A_{[1.1,\ (\neg A_1\wedge\neg A_2\wedge\neg A_3)]}\ ) \\
4.\ \wedge & ((\ A_{[1,\ \Box(\neg A_1\wedge\neg A_2\wedge\neg A_3)]} \wedge \neg A_{[1,\ \Box(\neg A_1\wedge\neg A_2\wedge\neg A_3)]}) \rightarrow A_{[1.1,\ (\neg A_1\wedge\neg A_2\wedge\neg A_3)]}\ ) \\
5.\ \wedge & (\ \neg A_{[1.1,\ (\neg A_1\wedge\neg A_2\wedge\neg A_3)]} \rightarrow (A_{[1.1,\ A_1]} \vee A_{[1.1,\ A_2]} \vee A_{[1.1,\ A_3]})\ ) \\
6.\ \wedge & (\ A_{[1.1,\ (\neg A_1\wedge\neg A_2\wedge\neg A_3)]} \rightarrow (\neg A_{[1.1,\ A_1]} \wedge \neg A_{[1.1,\ A_2]} \wedge \neg A_{[1.1,\ A_3]})\ )
\end{array}$$

$K_m 2SAT(\varphi_{bnflift})$ is found unsatisfiable directly by BCP on 1. and 2..

One potential drawback of applying the lifting rules (25.28) is that, by collapsing a conjunction/disjunction of modal atoms into one single atom, the possibility of sharing box/diamond subformulas in the DAG representation of φ is reduced. To cope with this problem, it is possible to adopt a *controlled* policy for applying Box/Diamond-lifting, that is, to apply (25.28) only if neither atom has multiple occurrences.

25.6.2.3. Handling incompatible π^r and ν^r

A first straightforward optimization, in the BNF variant, avoids the useless encoding of incompatible π^r and ν^r formulas. In BNF, in fact, the same subformula $\Box_r\psi$ may occur in the same state σ both positively and negatively (e.g., if $\pi^{r,j}$ is $\neg\Box_r\psi$ and ν^r is $\Box_r\psi$). If so, $K_m 2SAT$ labels both those occurrences of $\Box_r\psi$ with the same Boolean atom $A_{[\sigma,\ \Box_r\psi]}$, and produces recursively two distinct subsets of clauses in the encoding, by applying (25.26) to $\neg\Box_r\psi$ and (25.27) to $\Box_r\psi$ respectively. However, the latter step (25.27) generates a *valid* clause $(A_{[\sigma,\ \Box_r\psi]} \wedge \neg A_{[\sigma,\ \Box_r\psi]}) \rightarrow A_{[\sigma.j,\ \psi]}$, which can be dropped. Consequently $A_{[\sigma.j,\ \psi]}$ no more occurs in the formula, so that also $Def(\sigma.i,\ \psi)$ can be dropped as well, as there is no more need of defining $\langle\sigma : \psi\rangle$.

Example 12. *In the formula φ_{bnf} of Example 10 the implication 6. is valid and can be dropped. In the formula $\varphi_{bnflift}$ of Example 11, not only 4., but also 6. can be dropped.*

25.6.2.4. On-the-fly Boolean Constraint Propagation

One major problem of the basic encoding of §25.6.1 is that it is *purely-syntactic*, that is, it does not consider the possible truth values of the subformulas, and the effect of their propagation through the Boolean and modal connectives. In particular, $K_m 2SAT$ applies (25.26) [resp. (25.27)] to *every* π-subformula [resp. ν-subformula], regardless the fact that the truth values which can be deterministically assigned to the labeled subformulas of $\langle 1 : \varphi\rangle$ may allow for dropping some labeled π-/ν-subformulas, and thus prevent the need of encoding them.

One solution to this problem is that of applying BCP on-the-fly during the construction of $K_m 2SAT(\varphi)$. If a contradiction is found, then $K_m 2SAT(\varphi)$ is $\bot$. When BCP allows for dropping one implication in (25.24)-(25.27) without assigning some of its implicate literals, namely $L_{[\sigma,\ \psi_i]}$, then $\langle\sigma : \psi_i\rangle$ needs not to be defined, so that $Def(\sigma,\ \psi)$ can be dropped. Importantly, dropping $Def(\sigma,\ \pi^{r,j})$ for some π-formula $\langle\sigma : \pi^{r,j}\rangle$ prevents generating the label $\sigma.j$ (25.26) and all its successor labels $\sigma.j.\sigma'$ (corresponding to the subtree of states rooted in $\sigma.j$), so that all the corresponding labeled subformulas are not encoded.

Example 13. *Consider Example 10. After building 1. – 3. in $K_m 2SAT(\varphi_{bnf})$, the atoms $A_{[1,\varphi_{bnf}]}$, $A_{[1,(\neg\Box\neg A_1 \vee \neg\Box(\neg A_2 \wedge \neg A_3))]}$, $A_{[1,\Box\neg A_1]}$, $A_{[1,\Box\neg A_2]}$ and $A_{[1,\Box\neg A_3]}$ can be deterministically assigned to true by applying BCP. This causes the removal from 3. of the first-implied disjunct $\neg A_{[1,\ \Box\neg A_1]}$, so that 4. is not generated. As label 1.1. is not defined, 6., 7. and 8. are not generated. Then after the construction of 5., 9., 10., 11. and 12., by applying BCP a contradiction is found, so that $K_m 2SAT(\varphi)$ is $\bot$.*

25.6.2.5. On-the-fly Pure-Literal Reduction

Another technique, evolved from that proposed in [PSV02, PV03], applies Pure-Literal reduction on-the-fly during the construction of $K_m2SAT(\varphi)$. When for some label σ all the clauses containing atoms $A_{[\sigma,\ \psi]}$ have been generated, if some of them occurs only positively [resp. negatively], then it can be safely assigned to true [resp. to false], and hence the clauses containing $A_{[\sigma,\ \psi]}$ can be dropped. As a consequence, some other atom $A_{[\sigma,\ \psi']}$ can become pure, so that the process is repeated until a fixpoint is reached.

Example 14. *Consider the formula φ_{bnf} of Example 10. During the construction of $K_m2SAT(\varphi_{bnf})$, after 1.-8. are generated, no more clause containing atoms in the form $A_{[1.1,\ \psi]}$ is to be generated. Then we notice that $A_{[1.1,\ A_2]}$ and $A_{[1.1,\ A_3]}$ occur only negatively, so that they can be safely assigned to false. Therefore, 7. and 8. can be safely dropped. Same discourse applies lately to $A_{[1.2,\ A_1]}$ and 9. The resulting formula is found inconsistent by BCP. (In fact, notice that in Example 10 $A_{[1.1,\ A_2]}$, $A_{[1.1,\ A_3]}$, and $A_{[1.2,\ A_1]}$ play no role in the unsatisfiability of $K_m2SAT(\varphi_{bnf})$.)*

References

[ABC+02] G. Audemard, P. Bertoli, A. Cimatti, A. Korniłowicz, and R. Sebastiani. A SAT Based Approach for Solving Formulas over Boolean and Linear Mathematical Propositions. In *Proceedings of 18th International Conference on Automated Deduction (CADE)*, volume 2392 of *LNAI*. Springer, 2002.

[ACG00] A. Armando, C. Castellini, and E. Giunchiglia. SAT-based procedures for temporal reasoning. In *Proceedings of 5th European Conference on Planning, (ECP)*, volume 1809 of *LNCS*. Springer, 2000.

[AG93] A. Armando and E. Giunchiglia. Embedding Complex Decision Procedures inside an Interactive Theorem Prover. *Annals of Mathematics and Artificial Intelligence*, 8(3–4):475–502, 1993.

[AGHd00] C. Areces, R. Gennari, J. Heguiabehere, and M. de Rijke. Tree-based heuristics in modal theorem proving. In *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI)*, pages 199–203, 2000.

[BCM+03] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.

[BFH+94] F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H. J. Profitlich. An Empirical Analysis of Optimization Techniques for Terminological Representation Systems or: Making KRIS get a move on. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management*, 4:109–132, 1994.

[BFT95] P. Bresciani, E. Franconi, and S. Tessaris. Implementing and testing expressive Description Logics: a preliminary report. In *Proc. International Workshop on Description Logics*, Rome, Italy, 1995.

[BGdR03] S. Brand, R. Gennari, and M. de Rijke. Constraint Programming for Modelling and Solving Modal Satisfability. In *Proceedings of 9th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 3010 of *LNAI*, pages 795–800. Springer, 2003.

[BGV99] R. Bryant, S. German, and M. Velev. Exploiting Positive Equality in a Logic of Equality with Uninterpreted Functions. In *Proceedings of 11th International Conference on Computer Aided Verification (CAV)*, volume 1633 of *LNCS*. Springer, 1999.

[BH91] F. Baader and B. Hollunder. A Terminological Knowledge Representation System with Complete Inference Algorithms. In *Proceedings of the First International Workshop on Processing Declarative Knowledge*, volume 572 of *LNCS*, pages 67–85, Kaiserslautern (Germany), 1991. Springer–Verlag.

[Bra01] R. Brafman. A simplifier for propositional formulas with many binary clauses. In *Proceedings of 17th International Joint Conference on Artificial Intelligence (IJCAI)*, 2001.

[BS97] R. J. Bayardo and R. C. Schrag. Using CSP Look-Back Techniques to Solve Real-World SAT instances. In *Proceedings of 14th National Conference on Artificial Intelligence (AAAI)*, pages 203–208. AAAI Press, 1997.

[BW03] F. Bacchus and J. Winter. Effective Preprocessing with Hyper-Resolution and Equality Reduction. In *Proceedings of 6th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2003.

[CGH97] E. M. Clarke, O. Grumberg, and K. Hamaguchi. Another look at ltl model checking. *Formal Methods in System Design*, 10(1):47–71, 1997.

[Che80] B. F. Chellas. *Modal Logic – an Introduction.* Cambridge University Press, 1980.

[D'A92] M. D'Agostino. Are Tableaux an Improvement on Truth-Tables? *Journal of Logic, Language and Information*, 1:235–252, 1992.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5(7), 1962.

[DM94] M. D'Agostino and M. Mondadori. The Taming of the Cut. *Journal of Logic and Computation*, 4(3):285–319, 1994.

[DM00] F. Donini and F. Massacci. EXPTIME tableaux for ALC. *Artificial Intelligence*, 124(1):87–138, 2000.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[EB05] N. Eén and A. Biere. Effective Preprocessing in SAT Through Variable and Clause Elimination. In *Proceedings of 8th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 3569 of *LNCS*. Springer, 2005.

[ES04] N. Eén and N. Sörensson. An extensible SAT-solver. In *Proceedings of 6th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 2919 of *LNCS*, pages 502–518. Springer, 2004.

[FHMV95] R. Fagin, J. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about knowledge*. The MIT press, 1995.

[Fit83] M. Fitting. *Proof Methods for Modal and Intuitionistic Logics*. D. Reidel Publishg, 1983.

[GGST98] E. Giunchiglia, F. Giunchiglia, R. Sebastiani, and A. Tacchella. More evaluation of decision procedures for modal logics. In *Proceedings of Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, Trento, Italy, 1998.

[GGST00] 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.

[GGT01] E. Giunchiglia, F. Giunchiglia, and A. Tacchella. SAT Based Decision Procedures for Classical Modal Logics. Journal of Automated Reasoning. Special Issue: Satisfiability at the start of the year 2000, 2001.

[GN02] E. Goldberg and Y. Novikov. BerkMin: A Fast and Robust SAT-Solver. In *Proc. DATE '02*, page 142, Washington, DC, USA, 2002. IEEE Computer Society.

[GRS96] F. Giunchiglia, M. Roveri, and R. Sebastiani. A new method for testing decision procedures in modal and terminological logics. In *Proc. of 1996 International Workshop on Description Logics - DL'96*, Cambridge, MA, USA, November 1996.

[GS96a] F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures - the case study of modal K. In *Proc. CADE'13*, LNAI, New Brunswick, NJ, USA, August 1996. Springer.

[GS96b] F. Giunchiglia and R. Sebastiani. A SAT-based decision procedure for ALC. In *Proc. of the 5th International Conference on Principles of Knowledge Representation and Reasoning - KR'96*, Cambridge, MA, USA, November 1996.

[GS00] F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures - the case study of modal K(m). *Information and Computation*, 162(1/2), October/November 2000.

[GSK98] C. P. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98)*, pages 431–437, Madison, Wisconsin, 1998.

[GT01] E. Giunchiglia and A. Tacchella. Testing for Satisfiability in Modal Logics using a Subset-matching Size-bounded cache. *Annals of Mathematics and Artificial Intelligence*, 33:39–68, 2001.

[Hal95] J. Y. Halpern. The effect of bounding the number of primitive propositions and the depth of nesting on the complexity of modal logic. *Artificial Intelligence*, 75(3):361–372, 1995.

[HJSS96] A. Heuerding, G. Jager, S. Schwendimann, and M. Seyfried. The

Logics Workbench LWB: A Snapshot. *Euromath Bulletin*, 2(1):177–186, 1996.

[HM85] J. Y. Halpern and Y. Moses. A guide to the modal logics of knowledge and belief: preliminary draft. In *Proceedings of 9th International Joint Conference on Artificial Intelligence*, pages 480–490, Los Angeles, CA, 1985. Morgan Kaufmann Publ. Inc.

[HM92] J. Y. Halpern and Y. Moses. A guide to the completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54(3):319–379, 1992.

[HM01] V. Haarslev and R. Moeller. RACER System Description. In *Proc. of International Joint Conference on Automated reasoning - IJCAR-2001*, volume 2083 of *LNAI*, Siena, Italy, July 2001. Springer-verlag.

[Hor98a] I. Horrocks. The FaCT system. In *Proc. Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'98*, number 1397 in LNAI, pages 307–312. Springer, May 1998.

[Hor98b] I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, 1998.

[HPS99] I. Horrocks and P. F. Patel-Schneider. Optimizing Description Logic Subsumption. *Journal of Logic and Computation*, 9(3):267–293, 1999.

[HPSS00] I. Horrocks, P. F. Patel-Schneider, and R. Sebastiani. An Analysis of Empirical Testing for Modal Decision Procedures. *Logic Journal of the IGPL*, 8(3):293–323, May 2000.

[HS96] A. Heuerding and S. Schwendimann. A benchmark method for the propositional modal logics K, KT, S4. Technical Report IAM-96-015, University of Bern, Switzerland, 1996.

[HS99] U. Hustadt and R. Schmidt. An empirical analysis of modal theorem provers. *Journal of Applied Non-Classical Logics*, 9(4), 1999.

[HSW99] U. Hustadt, R. A. Schmidt, and C. Weidenbach. MSPASS: Subsumption Testing with SPASS. In *Proc. 1999 International Workshop on Description Logics (DL'99), vol. 22, CEUR Workshop Proceedings*, pages 136–137, 1999.

[Lad77] R. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM J. Comp.*, 6(3):467–480, 1977.

[Mas94] F. Massacci. Strongly analytic tableaux for normal modal logics. In *In Proceedings of 12th International Conference on Automated Deduction*, volume 814 of *Lecture Notes in Computer Science*. Springer, 1994.

[Mas98] F. Massacci. Simplification: A general constraint propagation technique for propositional and modal tableaux. In *Proc. 2nd International Conference on Analytic Tableaux and Related Methods (TABLEAUX-97)*, volume 1397 of *LNAI*. Springer, 1998.

[Mas99] F. Massacci. Design and Results of Tableaux-99 Non-Classical (Modal) System Competition. In *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference (Tableaux'99)*, 1999.

[Mas00] F. Massacci. Single Step Tableaux for modal logics: methodology, computations, algorithms. *Journal of Automated Reasoning*, Vol. 24(3), 2000.

[MMZ$^+$01] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conference*, 2001.

[MSS96] J. P. Marques-Silva and K. A. Sakallah. GRASP - A new Search Algorithm for Satisfiability. In *Proc. ICCAD'96*, 1996.

[Ngu05] L. A. Nguyen. On the Complexity of Fragments of Modal Logics. In *Advances in Modal Logic*. King's College Publications, 2005.

[NOT06] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, November 2006.

[PS98] P. F. Patel-Schneider. DLP system description. In *Proc. DL-98*, pages 87–89, 1998.

[PSS01] P. F. Patel-Schneider and R. Sebastiani. A system and methodology for generating random modal formulae. In *Proc. IJCAR-2001*, volume 2083 of *LNAI*. Springer-verlag, 2001.

[PSS03] P. F. Patel-Schneider and R. Sebastiani. A New General Method to Generate Random Modal Formulae for Testing Decision Procedures. *Journal of Artificial Intelligence Research, (JAIR)*, 18:351–389, May 2003. Morgan Kaufmann.

[PSV02] G. Pan, U. Sattler, and M. Y. Vardi. BDD-Based Decision Procedures for K. In *In proceedings of 18th International Conference on Automated Deduction*, volume 2392 of *Lecture Notes in Computer Science*. Springer, 2002.

[PSV06] G. Pan, U. Sattler, and M. Y. Vardi. BDD-Based Decision Procedures for the Modal Logic K. *Journal of Applied Non-Classical Logics*, 16(2), 2006.

[PV03] G. Pan and M. Y. Vardi. Optimizing a BDD-based modal solver. In *Proceedings of 19th International Conference on Automated Deduction*, volume 2741 of *Lecture Notes in Computer Science*. Springer, 2003.

[Sch91] K. D. Schild. A correspondence theory for terminological logics: preliminary report. In *Proc. 12th Int. Joint Conf. on Artificial Inteligence, IJCAI*, Sydney, Australia, 1991.

[Seb01] R. Sebastiani. Integrating SAT Solvers with Math Reasoners: Foundations and Basic Algorithms. Technical Report 0111-22, ITC-IRST, Trento, Italy, November 2001.

[Seb07] R. Sebastiani. Lazy Satisfiability Modulo Theories. *Journal on Satisfiability, Boolean Modeling and Computation – JSAT.*, 3, 2007.

[Smu68] R. M. Smullyan. *First-Order Logic*. Springer-Verlag, NY, 1968.

[SSB02] O. Strichman, S. Seshia, and R. Bryant. Deciding separation formulas with SAT. In *Proc. of Computer Aided Verification, (CAV'02)*, LNCS. Springer, 2002.

[SSS91] M. Schmidt-Schauß and G. Smolka. Attributive Concept Descriptions with Complements. *Artificial Intelligence*, 48:1–26, 1991.

[Str02] O. Strichman. On Solving Presburger and Linear Arithmetic with SAT. In *Proc. of Formal Methods in Computer-Aided Design (FM-CAD 2002)*, LNCS. Springer, 2002.

[SV98] R. Sebastiani and A. Villafiorita. SAT-based decision procedures for normal modal logics: a theoretical framework. In *Proc. AIMSA'98*, volume 1480 of *LNAI*. Springer, 1998.

[SV06] R. Sebastiani and M. Vescovi. Encoding the Satisfiability of Modal and Description Logics into SAT: The Case Study of K(m)/ALC. In *Proc. SAT'06*, volume 4121 of *LNCS*. Springer, 2006.

[SV08] R. Sebastiani and M. Vescovi. Automated Reasoning in Modal and Description Logics via SAT Encoding: the Case Study of K(m)/ALC-Satisfiability. Technical report, DISI, University of Trento, Italy, 2008. Submitted for journal pubblication. Available as `http://disi.unitn.it/~rseba/sat06/`.

[Tac99] A. Tacchella. *SAT system description. In *Proc. 1999 International Workshop on Description Logics (DL'99), vol. 22, CEUR Workshop Proceedings*, pages 142–144, 1999.

[Tin02] C. Tinelli. A DPLL-based Calculus for Ground Satisfiability Modulo Theories. In *Proc. JELIA-02*, volume 2424 of *LNAI*, pages 308–319. Springer, 2002.

[Var89] M. Y. Vardi. On the complexity of epistemic reasoning. In *Proceedings, Fourth Annual Symposium on Logic in Computer Science*, pages 243–252, 1989.

[Vor99] A. Voronkov. KK: a theorem prover for K. In *CADE-16: Proceedings of the 16th International Conference on Automated Deduction*, number 1632 in LNAI, pages 383–387. Springer, 1999.

[Vor01] A. Voronkov. How to optimize proof-search in modal logics: new methods of proving redundancy criteria for sequent calculi. *ACM Transacrtions on Computational Logic*, 2(2):182–215, 2001.

[WW99] S. Wolfman and D. Weld. The LPSAT Engine & its Application to Resource Planning. In *Proc. IJCAI*, 1999.

[ZM02] L. Zhang and S. Malik. The quest for efficient boolean satisfiability solvers. In *Proc. CAV'02*, number 2404 in LNCS, pages 17–36. Springer, 2002.