Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Combined Proof Methods for Multimodal Logic

Daniella Angelos

Dissertação apresentada como requisito parcial para
qualificação do Mestrado em Informática

Orientadora
Prof.a Dr.a Cláudia Nalon

Brasília
2017

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Combined Proof Methods for Multimodal Logic

Daniella Angelos

Dissertação apresentada como requisito parcial para
qualificação do Mestrado em Informática

Prof.a Dr.a Cláudia Nalon (Orientadora)
CIC/UnB

Prof. Dr.   Dr.

Prof. Dr. Bruno Luiggi Macchiavello Espinoza
Coordenador do Programa de Pós-graduação em Informática

Brasília, de de 2017

Abstract

Keywords: modal logics, resolution, sat-solvers

Contents

List of Figures

Chapter 1

Introduction

Chapter 2

Modal Logics

This chapter formally introduces K_n, a *propositional modal logic language*, semantically determined by an account of necessity and possibility [14].

A propositional modal language is the well known propositional language augmented by a collection of *modal operators* [2]. In classical logic, propositions or sentences are evaluated to either true or false, in any model. Propositional logic and predicate logic, for instance, do not allow for any further possibilities. However, in natural language, we often distinguish between various modalities of truth, such as *necessarily* true, *known to be* true, *believed to be* true or yet true *in some future*, for example. Therefore, one may think that classical logics lacks expressivity in this sense.

Modal logics add operators to express one or more of these different modes of truth. Different modalities define different languages. The key concept behind these operators is that they allow us to reason over relations among different contexts or interpretations, an abstraction that here we think as *possible worlds.* The purpose of the modal operators is to permit the information that holds at other worlds to be examined — but, crucially, only at worlds visible or accessible from the current one via an accessibility relation [2]. Then, the evaluation of a modal formula depends on the set of possible worlds and the accessibility relations defined for these worlds. It is possible to define several accessibility relations between worlds, and different modal logics are defined by different relations.

The modal language which is the focus of this work is the extension of the classical propositional logic plus the unary operators $\boxed{a}$ and $\Diamond_{a}$, whose reading are "is necessary by the agent a" and "is possible by the agent a", respectively. This language, known as K_n, is characterized by the schema $\boxed{a}(\varphi \Rightarrow \psi) \Rightarrow (\boxed{a}\varphi \Rightarrow \boxed{a}\psi)$ (axiom K), where a represents an agent in a finite, fixed set, and φ, ψ are well-formed formulae. The addition of other axioms defines different systems of modal logics and it imposes restrictions on the class of models where formulae are valid [4].

Worlds and their accessibility relations define a structure known as a *Kripke model*, a structure proposed by Kripke to semantically analyse modal logics [13]. The satisfiability and validity of a formula depend on this structure. For example, given a Kripke model that contains a set of possible worlds, a binary relation of accessibility between worlds and a valuation function that maps in which worlds a proposition symbol holds, we say that a formula $\Box p$ is satisfiable at some world w of this model, if the valuation function establishes that p is true at all worlds accessible from w.

In the following, we will formally define the modal language we will be working with. The syntax and semantics of K_n are showed in Sections 2.1 and 2.2, respectively, and the definitions presented in these two sections follow those in [14].

2.1 Syntax

The language of K_n is equivalent to its set of *well-formed formulae*, denoted by $\mathsf{WFF}_{\mathsf{K}_n}$, which is constructed from a denumerable set of *propositional symbols* $\mathcal{P} = \{p, q, r, \ldots\}$, the negation symbol $\neg$, the disjunction symbol $\vee$ and the modal connectives $\boxed{a}$, that express the notion of necessity, for each index a in a finite, non-empty fixed set of labels $\mathcal{A} = \{1, \ldots, n\}, n \in \mathbb{N}$.

Definition 1 The set of well-formed formulae, $\mathsf{WFF}_{\mathsf{K}_n}$, is the least set such that:

1. $p \in \mathsf{WFF}_{\mathsf{K}_n}$, for all $p \in \mathcal{P}$
2. if $\varphi, \psi \in \mathsf{WFF}_{\mathsf{K}_n}$, then so are $\neg\varphi$, $(\varphi \vee \psi)$ and $\boxed{a}\varphi$, for each $a \in \mathcal{A}$

The operators $\Diamond_a$ are the duals of $\boxed{a}$, for each $a \in \mathcal{A}$, that is, $\Diamond_a\varphi$ can be defined as $\neg\boxed{a}\neg\varphi$, with $\varphi \in \mathsf{WFF}_{\mathsf{K}_n}$. Other logical operators are also used as abbreviations. In this work, we consider the usual ones:

- $\varphi \wedge \psi \overset{\text{def}}{=} \neg(\neg\varphi \vee \neg\psi)$ (conjuction)
- $\varphi \Rightarrow \psi \overset{\text{def}}{=} \neg\varphi \vee \psi$ (implication)
- $\varphi \Leftrightarrow \psi \overset{\text{def}}{=} (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$ (equivalence)
- $\mathbf{false} \overset{\text{def}}{=} \varphi \wedge \neg\varphi$ (*falsum*)
- $\mathbf{true} \overset{\text{def}}{=} \neg\mathbf{false}$ (*verum*)

Parentheses may be omitted if the reading is not ambiguous. When $n = 1$, we often omit the index in the modal operators, i.e., we just write $\Box\varphi$ (or 'box' φ) and $\Diamond\varphi$ (or 'diamond' φ), for a well-formed formula φ.

We define as *literal* a propositional symbol $p \in \mathcal{P}$ or its negation $\neg p$, and denote by $\mathcal{L}$ the set of all literals. A *modal literal* is a formula of the form $\boxed{a}\, l$ or $\Diamond l$, with $l \in \mathcal{L}$ and $a \in \mathcal{A}$.

The following definitions are also needed later. The *modal depth* of a formula is recursively defined as follows:

Definition 2 We define $mdepth : \mathsf{WFF}_{\mathsf{K}_n} \longrightarrow \mathbb{N}$ inductively as:

1. $mdepth(p) = 0$
2. $mdepth(\neg\varphi) = mdepth(\varphi)$
3. $mdepth(\varphi \lor \psi) = \max\{mdepth(\varphi), mdepth(\psi)\}$
4. $mdepth(\boxed{a}\,\varphi) = mdepth(\varphi) + 1$

With $p \in \mathcal{P}$ and $\varphi, \psi \in \mathsf{WFF}_{\mathsf{K}_n}$.

This function represents the maximal number of nesting operators in a formula. For instance, if $\varphi = \boxed{a}\Diamond p \lor \Diamond q, a \in \mathcal{A}$, then $mdepth(\varphi) = 2$.

The *modal level* of a formula (or a subformula) is given relative to its position in the *annotated syntactic tree.*

Definition 3 Let Σ be the alphabet $\{1, 2, .\}$ and Σ^* the set of all finite sequences over Σ. We define $\tau : \mathsf{WFF}_{\mathsf{K}_n} \times \Sigma^* \times \mathbb{N} \longrightarrow \mathscr{P}(\mathsf{WFF}_{\mathsf{K}_n} \times \Sigma^* \times \mathbb{N})$ as the partial function inductively defined as follows:

1. $\tau(p, \lambda, ml) = \{(p, \lambda, ml)\}$
2. $\tau(\neg\varphi, \lambda, ml) = \{(\neg\varphi, \lambda, ml)\} \cup \tau(\varphi, \lambda.1, ml)$
3. $\tau(\boxed{a}\,\varphi, \lambda, ml) = \{(\boxed{a}\,\varphi, \lambda, ml)\} \cup \tau(\varphi, \lambda.1, ml + 1)$
4. $\tau(\varphi \lor \psi, \lambda, ml) = \{(\varphi \lor \psi, \lambda, ml)\} \cup \tau(\varphi, \lambda.1, ml) \cup \tau(\psi, \lambda.2, ml)$

With $p \in \mathcal{P}, \lambda \in \Sigma^*, ml \in \mathbb{N}$ and $\varphi, \psi \in \mathsf{WFF}_{\mathsf{K}_n}$.

The function τ applied to $(\varphi, 1, 0)$ returns the annotated syntactic tree for φ, where each node is uniquely identified by a subformula, its position in the tree (or path order) and its modal level. For instance, p occurs twice in the formula $\boxed{a}\Diamond(p \land \boxed{a}\,p)$, at the position 1.1.1, with modal level 2, and again at position 1.1.2.1, with modal level 3.

Definition 4 Let φ be a formula and let $\tau(\varphi, 1, 0)$ be its annotated syntactic tree.

We define *mlevel* : $\mathsf{WFF}_{\mathsf{K}_n} \times \mathsf{WFF}_{\mathsf{K}_n} \times \Sigma^* \longrightarrow \mathbb{N}$, as: if $(\varphi', \lambda, ml) \in \tau(\varphi, 1, 0)$ then $mlevel(\varphi, \varphi', \lambda) = ml$.

This function represents the maximal number of operators in which scope a subformula occurs.

2.2 Semantics

The semantics of K_n is presented in terms of Kripke structures.

Definition 5 A Kripke model for $\mathcal{P}$ and $\mathcal{A} = \{1, \ldots, n\}$ is given by the tuple

$$\mathcal{M} = (W, w_0, R_1, \ldots, R_n, \pi) \tag{2.1}$$

where W is a non-empty set of possible worlds with a distinguinshed world w_0, the root of $\mathcal{M}$; each R_a, $a \in \mathcal{A}$, is a binary relation on W, that is, $R_a \subseteq W \times W$, and $\pi : W \times \mathcal{P} \longrightarrow \{false, true\}$ is the valuation function that associates to each world $w \in W$ a truth-assignment to propositional symbols.

We write $R_a wv$ to denote that v is accessible from w through the accessibility relation R_a, that is $(w, v) \in R_a$, and $R_a^* wv$, to mean that v is reachable from w through a finite number of steps, that is, it exists a sequence $(w_1, \ldots, w_k)$ of worlds such that $R_a w_i w_{i+1}$, for all $i \leq k$, where $w_1 = w$ and $w_k = v$, with $a \in \mathcal{A}$, $w, v, w_i \in W$ and $i, k \in \mathbb{N}$. Note that R_a^* is the *transitive closure* of R_a, the least transitive set that contains all elements of R_a. In this work, we will also use the *transitive and reflexive closure*, denoted by R_a^+, the least transitive and reflexive set that contains all elements of R_a.

Satisfiability and *validity* of a formula is defined in terms of the *satisfiability relation.*

Definition 6 Let $\mathcal{M} = (W, w_0, R_1, \ldots, R_n, \pi)$ be a Kripke model, $w \in W$ and $\varphi, \psi \in \mathsf{WFF}_{\mathsf{K}_n}$. The *satisfiability relation*, denoted by $\langle \mathcal{M}, w \rangle \models \varphi$, between a world w and a formula φ, is inductively defined by:

1. $\langle \mathcal{M}, w \rangle \models p$ if, and only if, $\pi(w, p) = true$, for all $p \in \mathcal{P}$;
2. $\langle \mathcal{M}, w \rangle \models \neg\varphi$ if, and only if, $\langle \mathcal{M}, w \rangle \not\models \varphi$;
3. $\langle \mathcal{M}, w \rangle \models \varphi \vee \psi$ if, and only if, $\langle \mathcal{M}, w \rangle \models \varphi$ or $\langle \mathcal{M}, w \rangle \models \psi$;
4. $\langle \mathcal{M}, w \rangle \models \boxed{a}\varphi$ if, and only if, for all $t \in W$, $(w, t) \in R_a$ implies $\langle \mathcal{M}, t \rangle \models \varphi$.

Satisfiability is defined with respect to the root of a model. A formula $\varphi \in \mathsf{WFF}_{\mathsf{K}_n}$ is said to be *satisfiable* if there exists a Kripke model $\mathcal{M} = (W, w_0, R_1, \dots, R_n, \pi)$ such that $\langle \mathcal{M}, w_0 \rangle \models \varphi$. In this case we simply write $\mathcal{M} \models \varphi$ to mean that $\mathcal{M}$ statisfies φ in w_0. A formula is said to be *valid* if it is satisfiable in all models. We say that a set $\mathcal{F}$ of formulae is satisfiable if every $\varphi \in \mathcal{F}$ is satisfiable.

The satisfiability problem for K_n corresponds to determining the existence of a model in which a formula is satisfied. This problem is proven to be PSPACE-complete [19].

Example 1. Let $\mathcal{M}$ be the model illustrated in Figure 2.1. Take $\mathcal{M} = (W, w_0, R, \pi)$, for $\mathcal{P} = \{p\}$ and $\mathcal{A} = \{1\}$, where

(i) $W = \{w_0, w_1, w_2\}$

(ii) $R = \{(w_1, w_1), (w_2, w_2), (w_0, w_1), (w_0, w_2)\}$

(iii) $\pi(w, p) = \begin{cases} true & \text{if } w = w_0 \\ false & \text{otherwise} \end{cases}$

Note that both p and $\Box\neg p$ are satisfied in $\mathcal{M}$. This is a rather simple example to illustrate that, even though some sentence evaluates to true in the current context, one can see the same sentence occurring with the opposite valuation through an accessibility relation. This kind of reasoning is not possible in classical logic. Other examples of formulae satisfied by this model are: $p \wedge \Diamond\neg p$, $\Box\Box\neg p$ and $\Box\Box\Box\neg p$.

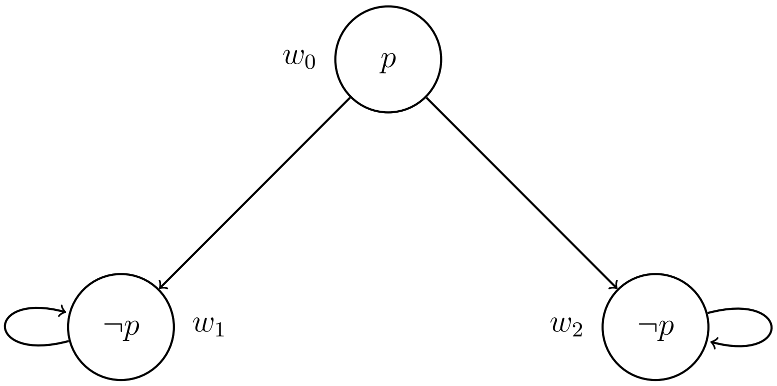
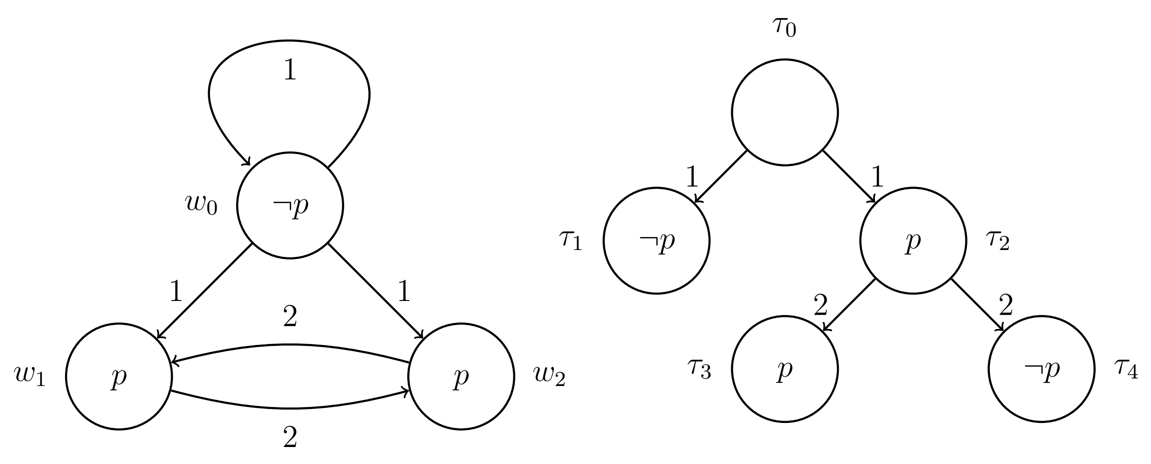


Figure 2.1: Example of a Kripke model for K_n

Example 2. (*Tree-like* model) Consider the formula $\varphi = \boxed{1}(p \Rightarrow \Diamond\!\!\!\!\raisebox{0.1ex}{\tiny 1}\,\, p)$. The Figure 2.2 contains examples of models that satisfy φ, hence, φ is satisfiable. Note that the model from the figure 2.2b has a graphical representation similar to a tree. Finite trees are ubiquitous in computer science, they are used to represent knowledge or data from the most diverse fields, such as linguistics, programming language etc. As trees play such an



(a) Kripke model (b) Tree model

Figure 2.2: Models that satisfy φ of Example 2

important role, we will take this opportunity to define them, and next, we will mention some interesting results concerning our modal language.

By a tree $\mathcal{T}$ we mean a relational structure (T, S) where T is a set of nodes and S is a binary relation among these nodes. T contains a unique $r_0 \in T$ (called the *root*) such that all other nodes in T are reachable from r_0, that is, for all $t \in T$, with $t \neq r_0$, we have $S^* r_0 t$, besides that, every element of T, distinct from r_0, has a unique S-predecessor, and the transitive and reflexive closure S^+ is acyclic, that is, for all $t \in T$, we have $\neg S^* tt$ [1].

A *tree model* is a Kripke model (W, w_0, R, π), with $\mathcal{A} = \{1\}$, where (W, R) is a tree and w_0 is its root. A *tree-like model* for K_n is a model $(W, w_0, R_1, \ldots, R_n, \pi)$, with $\mathcal{A} = \{1, \ldots, n\}$, such that $(W, \cup_{i \in \mathcal{A}} R_i)$ is a tree, with w_0 as the root.

Let $\mathcal{M} = (W, w_0, R_1, \ldots, R_a, \pi)$ be a tree-like model for K_n. We define the *depth* : $W \longrightarrow \mathbb{N}$ of a world $w \in W$, as the length of the path from w_0 to w through the union of the relations in $\mathcal{M}$. We sometimes say *depth* of $\mathcal{M}$ to mean the largest path from the root to any world in W.

The following theorems are particular cases of the ones presented in [1].

Theorem 2.2.1. *Let $\varphi \in \mathsf{WFF}_{\mathsf{K}_n}$ be a formula and $\mathcal{M} = (W, w_0, R_1, \ldots, R_n, \pi)$ be a model. Then $\mathcal{M} \models \varphi$ if and only if there is a tree-like model $\mathcal{M}'$ such that $\mathcal{M}' \models \varphi$. Moreover, $\mathcal{M}'$ is finite and its depth is bounded by mdepth(φ).*

Theorem 2.2.2. *Let $\varphi, \varphi' \in \mathsf{WFF}_{\mathsf{K}_n}$ and $\mathcal{M} = (W, w_0, R_1, \ldots, R_n, \pi)$ be a tree-like model such that $\mathcal{M} \models \varphi$. If $(\varphi', \lambda', ml) \in \tau(\varphi, 1, 0)$ and φ' is satisfied in $\mathcal{M}$, then there is $w \in W$, with depth$(w) = ml$, such that $\langle \mathcal{M}, w \rangle \models \varphi'$. Moreover, the subtree rooted at w has height equals to mdepth(φ').*

For the *global satisfiability* problem of a modal logic we need to add the universal modality, $\boxed{*}$, to the original modal language [12]. Let K_n^* be the logic obtained by adding $\boxed{*}$ to K_n. A model $\mathcal{M}^*$ for K_n^* is the pair $(\mathcal{M}, R_*)$, where $\mathcal{M} = (W, w_0, R_1, \ldots, R_n, \pi)$ is a tree-like model for K_n and $R_* = W \times W$. The global satisfiability problem is equivalent to the satisfiability problem in the following sense: a formula $\boxed{*}\varphi$ is satisfied at the world $w \in W$, in the model $\mathcal{M}^*$, written $\langle \mathcal{M}^*, w \rangle \models \boxed{*}\varphi$, if, and only if, for all $w' \in W$, we have that $\langle \mathcal{M}^*, w' \rangle \models \varphi$. Therefore, let $\varphi \in \mathsf{WFF}_{\mathsf{K}_n}$ be a formula, we say that φ is globally satisfiable in a model $\mathcal{M}$, denoted $\mathcal{M} \models_G \varphi$, if, and only if, $\mathcal{M}^* \models \boxed{*}\varphi$.

2.3 Proof Systems and Normal Forms

Informally, a proof is an argument that can convince if determined formula is satisfiable or not. Formally, a proof is a finite object constructed according to fixed syntactic rules that refer only to the structure of formulae and not to their intended meaning [10]. The set of syntactic rules that define proofs are said to specify a *proof system* or *calculus*. These rules allow the derivation of formulae from formulae through strict symbol manipulation [8].

A proof system is *sound* for a particular logic if any formula that has a proof is a valid formula of the logic, and it is *complete* for a particular logic if any valid formula has a proof [10]. A sound and complete calculus for K_n finds a model for a formula if, and only if, this formula is satisfiable.

There are several kinds of proof systems, and they can be divided in many categories as, for instance, *refutational systems*. To prove a formula using a refutational system, we begin by negating it, then analysing the consequences of doing so, using a kind of tree structure, we verify if the consequences turn out to be impossible, if that is the case, we conclude that the original formula has been proved [10]. The calculus for K_n we use in this work is based in resolution, which is one of the most common examples of refutational systems. This calculus is introduced in Chapter 3, resolution is also briefly introduced in the same chapter. The resolution calculus used is proved to be sound and complete for K_n, it was developed with regard to computer implementations and it uses formulae into a special *normal form*.

A normal form is an elegant representation of an equivalence class. The equivalence relation in question may determine what kind of normal form is used. The relation considered in proof systems for logic languages relates two formulae if whenever one is satisfiable, the other one also is. Therefore, the transformation rules defined for a specific normal form used in a proof system for a logic must preserve satisfiability, that is, the formula obtained by applying a transformation rule is satisfiable if, and only if, the original one also is. Normal forms can provide constructive proofs of many standard results [9].

This happens because formulae translated into a normal form have a specific, normalized structure, possibly resulting in less operators, which may implicate into a smaller number of rules for a proof system. Hence, a calculus that is planned to be implemented in a computer may take great advantage of normal forms, since the smaller number of rules reduces the chances of implementation errors.

The normal form used in this work is a layered normal form called Separated Normal Form with Modal Levels and it's introduced in Chapter 3, along side with its transformation rules for formulae in K_n. These rules are proved to preserve satisfiability of formulae.

Chapter 3

Modal-Layered Resolution

Resolution appeared in the early 1960's through investigations on performance improvements of refutational procedures based on *Herbrand Theorem*. In particular, Prawitz' studies on such procedures brought back the idea of unification. J. A. Robinson incorporated the concept of unification on a refutation method, creating what was later known as resolution [3].

The standard rule for resolution systems takes two or more premises with literals that are contradictory, and generates a resolvent. Most of these systems work exclusively with clauses in a specific normal form. Resolution systems are refutational systems, that is, to show that a formula φ is valid, $\neg\varphi$ is translated into a normal form. The inference rules are applied until either no new resolvents can be generated or a contradiction is obtained. The contradiction implies that $\neg\varphi$ is unsatisfiable and hence, that φ is valid.

3.1 Clausal Resolution

Clausal resolution was proposed as a proof method for classical logic by Robinson in 1965 [17], and was claimed to be suitable to be performed by computer, as it has only one inference rule that is applied exhaustively.

3.2 Separated Normal Form with Modal Levels

Formulae in K_n can be transformed into a layered normal form called *Separated Normal Form with Modal Levels*, denoted by SNF_{ml}, proposed in [14]. A formula in SNF_{ml} is a conjunction of *clauses* where the modal level in which they occur is emphasized as a label.

We write $ml : \varphi$ to denote that φ occurs at modal level $ml \in \mathbb{N} \cup \{*\}$. By $* : \varphi$ we mean that φ is true at all modal levels. Formally, let $\mathrm{WFF}^{ml}_{\mathsf{K}_n}$ denote the set of formulae with the modal level annotation, $ml : \varphi$, such that $ml \in \mathbb{N} \cup \{*\}$ and $\varphi \in \mathsf{WFF}_{\mathsf{K}_n}$. Let $\mathcal{M}^* = (W, w_0, R_1, \dots, R_n, R_*, \pi)$ be a tree-like model and take $\varphi \in \mathsf{WFF}_{\mathsf{K}_n}$.

Definition 7 Satisfiability of labelled formulae is given by:

1. $\mathcal{M}^* \models ml : \varphi$ if, and only if, for all worlds $w \in W$ such that $depth(w) = ml$, we have $\langle \mathcal{M}^*, w \rangle \models \varphi$
2. $\mathcal{M}^* \models * : \varphi$ if, and only if, $\mathcal{M}^* \models \boxed{*}\varphi$

Observe that the labels in formulae work as a kind of *weak* universal operator, allowing us to reason about a set of formulae that are all satisfied at a given modal level.

Clauses in SNF_{ml} are defined as follows.

Definition 8 Clauses in SNF_{ml} are in one of the following forms:

1. Literal clause $\quad ml : \bigvee_{b=1}^{r} l_b$
2. Positive a-clause $\quad ml : l' \Rightarrow \boxed{a}\, l$
3. Negative a-clause $\quad ml : l' \Rightarrow \diamondsuit_a l$

where $r, b \in \mathbb{N}, ml \in \mathbb{N} \cup \{*\}$ and $l, l', l_b \in \mathcal{L}$.

Positive and negative a-clauses are together known as *modal a-clauses*, the index a can be omitted if it is clear from the context.

The transformation of a formula $\varphi \in \mathsf{WFF}_{\mathsf{K}_n}$ into SNF_{ml} is achieved by first transforming φ into its *Negation Normal Form*, and then, recursively applying rewriting and renaming [16].

Definition 9 Let $\varphi \in \mathsf{WFF}_{\mathsf{K}_n}$. We say that φ is in Negation Normal Form (NNF) if it contains only the operators $\neg, \vee, \wedge, \boxed{a}$ and $\diamondsuit_a$. Also, only propositions are allowed in the scope of negations.

Let φ be a formula and t a propositional symbol not occurring in φ. The translation of φ is given by $0 : t \wedge \rho(0 : t \Rightarrow \varphi)$ – for global satisfiability, the translation is given by $* : t \wedge (\rho(* : t \Rightarrow \varphi)$ – where ρ is the *translation function* defined below. We refer to clauses of the form $0 : D$, for a disjunction of literals D, as *initial clauses*.

Definition 10 The translation function $\rho : \mathrm{WFF}^{ml}_{\mathsf{K}_n} \longrightarrow \mathrm{WFF}^{ml}_{\mathsf{K}_n}$ is defined as follows:

$$\begin{aligned}
\rho(ml : t \Rightarrow \varphi \wedge \psi) &= \rho(ml : t \Rightarrow \varphi) \wedge \rho(ml : t \Rightarrow \psi) \\
\rho(ml : t \Rightarrow \boxed{a}\varphi) &= (ml : t \Rightarrow \boxed{a}\varphi), \text{ if } \varphi \text{ is a literal} \\
&= (ml : t \Rightarrow \boxed{a}t') \wedge \rho(ml + 1 : t' \Rightarrow \varphi), \text{ otherwise} \\
\rho(ml : t \Rightarrow \Diamond_{a}\varphi) &= (ml : t \Rightarrow \Diamond_{a}\varphi), \text{ if } \varphi \text{ is a literal} \\
&= (ml : t \Rightarrow \Diamond_{a}t') \wedge \rho(ml + 1 : t' \Rightarrow \varphi), \text{ otherwise} \\
\rho(ml : t \Rightarrow \varphi \vee \psi) &= (ml : \neg t \vee \varphi \vee \psi), \text{ if } \psi \text{ is a disjunction of literals} \\
&= \rho(ml : t \Rightarrow \varphi \vee t') \wedge \rho(ml : t' \Rightarrow \psi), \text{ otherwise}
\end{aligned}$$

Where $t, t' \in \mathcal{L}$, $\varphi, \psi \in \mathsf{WFF}_{\mathsf{K}_n}$, $ml \in \mathbb{N} \cup \{*\}$ and $r, b \in \mathbb{N}$.

As the conjunction operator is commutative, associative and idempotent, we will commonly refer to a formula in SNF_{ml} as a set of clauses.

The next lemma, taken from [15], shows that the transformation into SNF_{ml} preserves satisfiability.

Lemma 3.2.1. *Let $\varphi \in \mathsf{WFF}_{\mathsf{K}_n}$ be a formula and let t be a propositional symbol not occurring in φ. Then:*

(*i*) *φ is satisfiable if, and only if, $0 : t \wedge \rho(0 : t \Rightarrow \varphi)$ is satisfiable;*

(*ii*) *φ is globally satisfiable if, and only if, $* : t \wedge \rho(* : t \Rightarrow \varphi)$ is satisfiable;*

Example 3.

Chapter 4

Satisfiability Solvers

The problem of determining whether a formula in classical propositional logic is satisfiable has the historical honor of being the first problem ever shown to be NP-Complete [5]. Great theoretical and practical efforts have been directed in improving the efficiency of solvers for this problem, known as *Boolean Satisfiability Solvers*, or just *SAT solvers*. Despite the worst-case exponential run time of all the algorithms known, satisfiability solvers are increasingly leaving their mark as a general purpose tool in the most diverse areas [11]. In essence, SAT solvers provide a generic combinatorial reasoning and search platform.

In the context of SAT solvers for propositional provers, the underlying representational formalism is propositional logic [11]. We are interested in formulae in *Conjunctive Normal Form* (CNF): φ is in CNF if it is a conjunction of *clauses*, where each clause is a disjunction of literals. For example, $\varphi = (p \vee \neg q) \wedge (\neg p \vee r \vee s) \wedge (q \vee r)$ is a CNF formula with four literals and three clauses. We use the symbol $\emptyset$ to denote the *empty clause*, i.e., the clause that contains no literals. A clause with only one literal is referred to as a *unit clause*, and a clause with two literals, as a *binary clause*. When every clause of φ has k literals, we refer to φ as a k-CNF formula.

A propositional formula φ takes value in the set $\{false, true\}$. A *truth assignment* (or just assignment) to a set of literals $\mathcal{L}$ is a map $\sigma : \mathcal{L} \longrightarrow \{false, true\}$. A *satisfying assignment* for φ is an assignment σ such that φ evaluates to *true* under σ. A *partial assignment* for a formula φ is a truth assignment to a subset of the literals in φ. For a partial assignment ρ for a CNF formula φ, $\varphi|_\rho$ denotes the simplified formula obtained by replacing the literals appearing in ρ with their specified values, removing all clauses with at least one *true* literal, and deleting all occurrences of $false$ literals from the remaining clauses.

Therefore, the *Boolean Satisfiability Problem* (SAT) can be expressed as: Given a CNF formula φ, does φ have a satisfying assignment? If this is the case, φ is said to be *satisfiable*, otherwise, φ is *unsatisfiable*. One can be interested not only in the answer of this decision problem, but also in finding the actual assignment that satisfies the formula, when it exists. All practical SAT solvers do produce such assignment [6].

4.1 The DPLL Procedure

A *complete* solution method for the SAT problem is one that, given the input formula φ, either produces a satisfying assignment for φ or proves that it is unsatisfiable. One of the most surprising aspects of the relatively recent practical progress of SAT solvers is that the best complete methods remain variants of a process introduced in the early 1960's: the Davis-Putnam-Logemann-Loveland, or DPLL, procedure [11], which performs a backtrack search in the space of partial truth assignments [7]. A key feature of DPLL is efficient pruning of the search space based on falsified clauses. Since its introduction, the main improvements to DPLL have been smart branch selection heuristics, extensions like clause learning and randomized restarts, and well-crafted data structures such as lazy implementations and watched literals for fast unit propagation [11].

Algorithm 1, DPLL-recursive(φ, ρ), sketches the basic DPLL procedure on CNF formulae [7]. The main idea is to repeatedly select an unassigned literal l in the input formula and recursively search for a satisfying assignment for $\varphi|_l$ and $\varphi|_{\neg l}$. The step where such an l is chosen is called a *branching step*. Setting l to true or false when making a recursive call is referred to as *decision*, and is associated with a decision level which equals the recursion depth at that stage. The end of each recursive call, which takes φ back to fewer assigned literals, is called the *backtracking step*.

Algorithm 1: DPLL-recursive(φ, ρ)

```
1  $(\varphi, \rho) \leftarrow$ UnitPropagate$(\varphi, \rho)$
2  if $\varphi$ contains the empty clause then
3      return UNSAT
4  end
5  if $\varphi$ has no clauses left then
6      Output $\rho$
7      return SAT
8  end
9  $l \leftarrow$ a literal not assigned by $\rho$
10 if DPLL-recursive$(\varphi|_l, \rho \cup \{l\}) = SAT$ then
11     return SAT
12 end
13 return DPLL-recursive$(\varphi|_{\neg l}, \rho \cup \{\neg l\})$
```

4.2 Conflict Driven Clause Learning

Algorithm 2.

```
Algorithm 2: CDCL($F, \nu$)
  Input  :
  Output:
1 if $UnitPropagate(F, \nu) == CONFLICT$ then
2     return $UNSAT$
3 end
4 $dl \leftarrow 0$
5 while not $AllVariablesAssigned(F, \nu)$ do
6     $(x, \nu) \leftarrow$ PickBranchingVariable($F, \nu$)
7 end
8 return $SAT$
```

4.3 MiniSat and Glucose

Minisat [18]

Chapter 5

Methodology

To implement a calculus for a logic, one needs a representation of formulae and operations corresponding to the inference rules. Somewhat more abstractly, one has to define a state transition system. The states represent (sets of) formulae with their interrelationships, providing information on the development of the derivations up to the respective point and on their possible continuations. The transitions model the changes to the states as inference rules are applied. Better state transition systems for the same calculus can be obtained by refining the states, for instance such that they indicate directly where inference rules can be or have been applied. More common improvements define additional transitions, which are not based on the rules of the calculus, but simplify the formulae or eliminate redundancies in the search space. The state transition systems described in this chapter are based on sets of clauses and on graph structures imposed on them.

Bibliography

[1] 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*, pages 199–203. IOS Press, 2000. 7

[2] P. Blackburn, M. De Rijke, and Y. Venema. *Modal Logic: Graph. Darst*, volume 53. Cambridge University Press, 2002. 2

[3] M. A. Casanova. *Programação em lógica e a linguagem Prolog.* E. Blucher, 1987. 10

[4] B. F. Chellas. *Modal logic — an introduction.* Press Syndicate of the University of Cambridge, London, 1980. 2

[5] S. A. Cook. The complexity of theorem proving procedures. In *stoc71*, pages 151–158, 1971. 13

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 3rd Edition.* MIT Press, 2009. 14

[7] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, July 1962. 14

[8] N. Eisinger and J. Ohlbach. Deduction systems based on resolution. Technical report, Saarbruecken, 1991. 8

[9] K. Fine. Normal forms in modal logic. *Notre Dame J. Formal Logic*, 16(2):229–237, 04 1975. 8

[10] M. Fitting and R. L. Mendelsohn. *First-order modal logic*, volume 277. Springer Science & Business Media, 2012. 8

[11] C. P. Gomes, H. Kautz, A. Sabharwal, and B. Selman. Satisfiability solvers. *Foundations of Artificial Intelligence*, 3:89–134, 2008. 13, 14

[12] V. Goranko and S. Passy. Using the universal modality: gains and questions. *Journal of Logic and Computation*, 2(1):5–30, 1992. 8

[13] S. A. Kripke. Semantical analysis of modal logic I. *Zeitschr. Math. Logik Grund. Math*, 9:67–96, 1963. 3

[14] C. Nalon and C. Dixon. Clausal resolution for normal modal logics. *J. Algorithms*, 62(3-4):117–134, 2007. 2, 3, 10

[15] C. Nalon, U. Hustadt, and C. Dixon. A modal-layered resolution calculus for k. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 185–200. Springer, 2015. 12

[16] D. A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293–304, 1986. 11

[17] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, Jan. 1965. 10

[18] N. Sorensson and N. Een. Minisat v1. 13-a sat solver with conflict-clause minimization. *SAT*, 2005(53), 2005. 15

[19] E. Spaan. *Complexity of Modal Logics*. PhD thesis, University of Amsterdam, 1993. 6