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

List of Tables

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 [18].

A propositional modal language is the well known propositional language augmented by a collection of *modal operators* [5]. 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 allow 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 [5]. 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 $\langle a \rangle$, whose reading are "is necessary from the point of view of an agent a" and "is possible from the point of view of an 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 is an index from 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 [7].

A set of worlds, their accessibility relations and a valuation function define a structure known as a *Kripke model*, a structure proposed by Kripke to semantically analyse modal logics [17]. 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. 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 [18].

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* or *variables* $\mathcal{P} = \{p, q, r, \ldots\}$, the negation symbol $\neg$, the disjunction symbol $\vee$ and the modal connectives $\boxed{a}$, that express the notion of necessity, for each a in a finite, non-empty fixed set of indexes $\mathcal{A} = \{1, \ldots, n\}, n \in \mathbb{N}$.

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 operator $\langle a \rangle$ is the dual of $\boxed{a}$, for each $a \in \mathcal{A}$, that is, $\langle a \rangle\varphi$ 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:

- $\varphi \wedge \psi \stackrel{\text{def}}{=} \neg(\neg\varphi \vee \neg\psi)$ (conjuction)
- $\varphi \Rightarrow \psi \stackrel{\text{def}}{=} \neg\varphi \vee \psi$ (implication)
- $\varphi \Leftrightarrow \psi \stackrel{\text{def}}{=} (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$ (equivalence)
- **false** $\stackrel{\text{def}}{=} \varphi \wedge \neg\varphi$ (*falsum*)
- **true** $\stackrel{\text{def}}{=}$ $\neg$**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 $\Box\varphi$ and $\Diamond\varphi$, for a well-formed formula φ.

We define a *literal* as 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 $\Diamond\!\!\!\!\cdot\;\, l$, 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. $mdepth(\varphi \vee \psi) = \max\{mdepth(\varphi), mdepth(\psi)\}$
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 $\varphi = \boxed{a} \Diamond\!\!\!\!\cdot\;\, p \vee \Diamond\!\!\!\!\cdot\;\, q, a \in \mathcal{A}$, 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. $\tau(\varphi \vee \psi, \lambda, ml) = \{(\varphi \vee \psi, \lambda, ml)\} \cup \tau(\varphi, \lambda.1, ml) \cup \tau(\psi, \lambda.2, ml)$

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 $\boxed{a} \Diamond\!\!\!\!\cdot\;\, (p \wedge \boxed{a} p)$, 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, there 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 are 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 $\langle \mathcal{M}, t \rangle \models \varphi$.

A formula $\varphi \in \mathsf{WFF}_{\mathsf{K}_n}$ is said to be *locally satisfiable* if there exists a Kripke model $\mathcal{M} = (W, w_0, R_1, \ldots, R_n, \pi)$ such that $\langle \mathcal{M}, w_0 \rangle \models \varphi$. In this case we simply write $\mathcal{M} \models_L \varphi$ to mean that $\mathcal{M}$ locally satisfies φ. A model $\mathcal{M} = (W, w_0, R_1, \ldots, R_n, \pi)$ is said to *globally satisfy* a formula φ, denoted $\mathcal{M} \models_G \varphi$, if for all $w \in W$, we have $\langle \mathcal{M}, w \rangle \models \varphi$. A formula φ is said to be *globally satisfiable* if there is a model $\mathcal{M}$ such that $\mathcal{M}$ globally satisfies φ. We say that a set $\mathcal{F}$ of formulae is locally satisfiable if there is a model that locally satisfies every $\varphi \in \mathcal{F}$. Global satisfiability of sets is defined analogously. A formula is said to be *valid* if it is locally satisfiable in all models.

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 $\Box\Box\Box\neg p$.

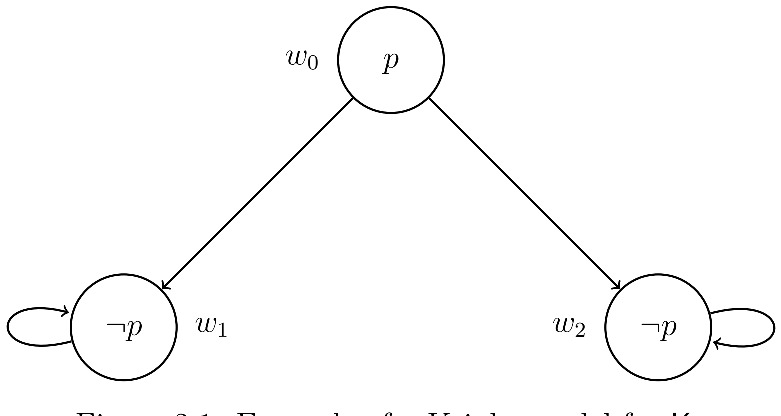
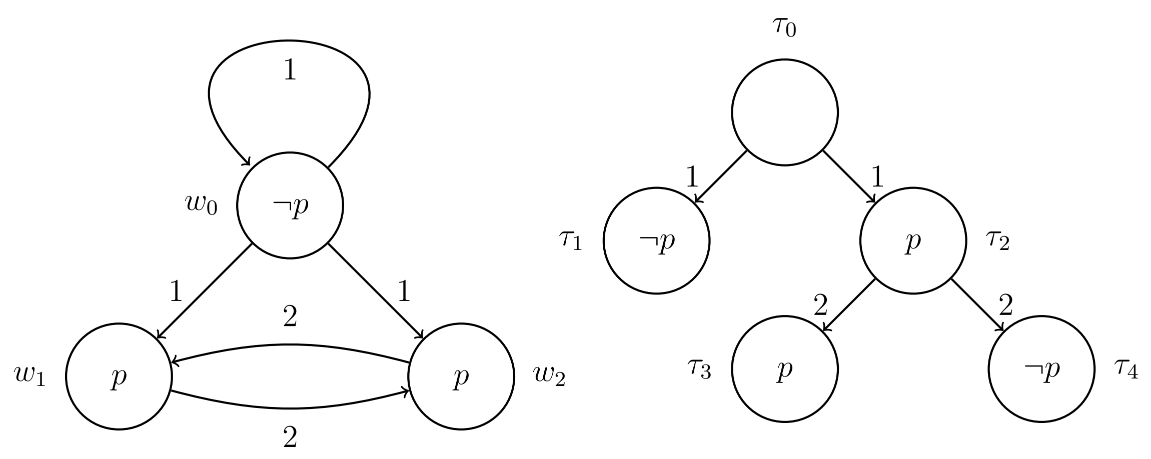


Figure 2.1: Example of a Kripke model for K_n

Example 2. (*Tree-like* model) Consider the formula $\varphi = \boxed{1}(p \Rightarrow \Diamond p)$. 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 equivalent to a tree.



(a) Kripke model (b) Tree model

Figure 2.2: Models that satisfy φ of Example 2

As trees play an important role in computer science, we will take this opportunity to define them. 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 over these nodes. T contains a unique node $r_0 \in T$ (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 $\neg S^* tt$ [2].

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 longest path from the root to any world in W.

The following theorems have been adapted from the ones presented in [2].

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 $\mathcal{M} \models_L \varphi$ if and only if there is a tree-like model $\mathcal{M}'$ such that $\mathcal{M}' \models_L \varphi$. Moreover, $\mathcal{M}'$ is finite and its depth is bounded by mdepth(φ).*

The proof of Theorem 2.2.1 presented in [5] constructs a tree-like model $\mathcal{M}'$ as a *generated submodel* of $\mathcal{M}$, that is, it restricts the binary relations to consider only a subset of W. Then, using the satisfiability invariance of generated submodels, also proved in [5], which states that a formula is satisfiable in a model if, and only if, is satisfiable in its generated submodels, the proof becomes trivial.

Theorem 2.2.2. *Let $\varphi, \varphi' \in \mathsf{WFF}_{\mathsf{K}_n}$ and $\mathcal{M} = (W, w_0, R_1, \dots, R_n, \pi)$ be a tree-like model such that $\mathcal{M} \models_L \varphi$. 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 $mdepth(\varphi')$.*

The proof of Theorem 2.2.2 is by induction on the structure of the formula and shows that a subformula φ' of φ is satisfied at a node with distance m of the root of the tree-like model. As determining the satisfiability of a formula depends only on its subformulae, only the subtrees of height $mdepth(\varphi')$ starting at level m need to be checked. The bound on the height of the subtrees follows from Theorem 2.2.1 [19].

The *local satisfiability problem* for K_n corresponds to determining the existence of a model in which a formula is locally satisfied. This problem is proven to be PSPACE-complete [25]. The *global satisfiability problem* for a modal logic is equivalent to the local satisfiability problem of the logic obtained by adding the universal modality, $\boxed{*}$, to the original modal language [16]. 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 $\mathcal{M} = (W, w_0, R_1, \dots, R_n, \pi)$ 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 $\mathcal{M} \models_G \varphi$, if, and only if, $\mathcal{M}^* \models_L \boxed{*}\varphi$. The global satisfiability problem is proven to be EXPTIME [25].

2.3 Proof Systems and Normal Forms

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 [13]. 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 [11].

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 [13]. A sound and complete calculus for K_n finds a proof 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 repercussion of doing so, we verify if the consequences turn out to be impossible, if that is the case, we conclude that the original formula has been proved [13]. The calculus for K_n we use in this work is resolution based, which is one of the most common examples of refutational systems. This calculus was proposed by [19] and is presented 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 in [19], it was developed with regard to computer implementations and it uses formulae translated into a special *normal form.*

A normal form is an elegant representation of an equivalence class and the equivalence relation in question may determine what kind of normal form is used [1]. 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 may help calculi to provide constructive proofs of many standard results [12], that is, a proof that demonstrates the existence of a mathematical object by providing a method for creating this object. This happens because formulae translated into a normal form have a specific, normalized structure, possibly resulting in less operators to handle with, 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 [6].

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 [22], and was claimed to be suitable to be performed by computer, as it has only one inference rule that is applied exhaustively.

3.1.1 Modal-Layered Resolution

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 [18]. 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 $\mathcal{M}^* = (W, w_0, R_1, \ldots, R_n, R_*, \pi)$ 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 [21].

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:

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

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 [19], 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.

3.3 Modal-Layered Resolution Calculus for K_n

The motivation for the use of this labelled clausal normal form in a calculus is that inference rules can then be guided by the semantic information given by the labels and applied to smaller sets of clauses, reducing the number of unnecessary inferences, and therefore improving the efficiency of the proof procedure [20].

This calculus comprises a set of inference rules, given in Table 3.1, for dealing with propositional and modal reasoning. In the following, we denote by σ the result of unifying the labels in the premises for each rule. Formally, unification is given by a function

Table 3.1: Inference rules

$$\text{[LRES]} \quad \begin{array}{rl} ml_1 : & D \vee l \\ ml_2 : & D' \vee \neg l \\ \hline \sigma(\{ml_1, ml_2\}) : & D \vee D' \end{array} \qquad \text{[MRES]} \quad \begin{array}{rl} ml_1 : & l_1 \Rightarrow \boxed{a} l \\ ml_2 : & l_2 \Rightarrow \neg \boxed{a} l \\ \hline \sigma(\{ml_1, ml_2\}) : & \neg l_1 \vee \neg l_2 \end{array}$$

$$\text{[GEN2]} \quad \begin{array}{rl} ml_1 : & l'_1 \Rightarrow \boxed{a} l_1 \\ ml_2 : & l'_2 \Rightarrow \boxed{a} \neg l_1 \\ ml_3 : & l'_3 \Rightarrow \Diamond_a l_2 \\ \hline \sigma(\{ml_1, ml_2, ml_3\}) : & \neg l'_1 \vee \neg l'_2 \vee \neg l'_3 \end{array}$$

$$\text{[GEN1]} \quad \begin{array}{rl} ml_1 : & l'_1 \Rightarrow \boxed{a} \neg l_1 \\ \vdots & \\ ml_m : & l'_m \Rightarrow \boxed{a} \neg l_m \\ ml_{m+1} : & l' \Rightarrow \Diamond_a \neg l \\ ml_{m+2} : & l_1 \vee \ldots \vee l_m \vee l \\ \hline ml : & \neg l'_1 \vee \ldots \vee \neg l'_m \vee \neg l' \end{array} \qquad \text{[GEN3]} \quad \begin{array}{rl} ml_1 : & l'_1 \Rightarrow \boxed{a} \neg l_1 \\ \vdots & \\ ml_m : & l'_m \Rightarrow \boxed{a} \neg l_m \\ ml_{m+1} : & l' \Rightarrow \Diamond_a l \\ ml_{m+2} : & l_1 \vee \ldots \vee l_m \\ \hline ml : & \neg l'_1 \vee \ldots \vee \neg l'_m \vee \neg l' \end{array}$$

where $ml = \sigma(\{ml_1, \ldots, ml_{m+1}, ml_{m+2} - 1\})$ where $ml = \sigma(\{ml_1, \ldots, ml_{m+1}, ml_{m+2} - 1\})$

$\sigma : \mathscr{P}(\mathbb{N} \cup \{*\}) \longrightarrow \mathbb{N} \cup \{*\}$, where $\sigma(\{ml, *\}) = ml$ and $\sigma(\{ml\}) = ml$, otherwise, σ is undefined. The following inference rules can only be applied if the unification of their labels is defined (where $* - 1 = *$). Note that for GEN1 and GEN3, if the modal clauses occur at the modal level ml, then the literal clause occurs at the next modal level, $ml + 1$.

The proofs for termination, soundness and completeness of this calculus can be found in [19].

3.4 K$_{\text{S}}$P

In this section, we present K$_{\text{S}}$P, the theorem prover presented in [20] for the basic multi-modal logic K_n, which implements a variation of the set of support strategy [26] for the modal resolution-based calculus described in Section 3.3.

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 [8]. 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 deterministic 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 [15]. In essence, SAT solvers provide a generic combinatorial reasoning and search platform. Beyond that, the source code of many implementations of such solvers is freely available and can be used as a basis for the development of decision procedures for more expressive logics [14].

In the context of SAT solvers for propositional provers, the underlying representational formalism is propositional logic [15]. We are interested in formulae in *Conjunctive Normal Form* (CNF): φ is in CNF if it is a conjunction of *clauses*, denoted ω, 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 variables and three clauses. We use the constant **false** 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*.

A propositional formula φ takes value in the set $\{false, true\}$. In algorithms for SAT, variables can be *assigned* a logic value in the same set and, alternatively, variables may also be *unassigned*. A *truth assignment* (or just assignment) to a set of variables $\mathcal{P}$, is the valuation function π as defined in Definition 5. As in propositional logic we have a unit set as the set of possible worlds W, we can omit this set from the function signature and just write $\pi : \mathcal{P} \longrightarrow \{false, true\}$, for simplicity. A *satisfying assignment* for φ is an assignment π such that φ evaluates to *true* under π. A *partial assignment* for a formula φ is a truth assignment to a subset of the variables in φ. For a partial assignment ρ for a CNF formula φ, $\varphi|_\rho$ denotes the simplified formula obtained by replacing the variables 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 [15].

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 an assignment [9].

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 [15]. 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 [10], which performs a backtrack search in the space of partial truth assignments. 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 [15].

Algorithm 1, DPLL-recursive(φ, ρ), sketches the basic DPLL procedure on CNF formulae [10]. 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 of the procedure. The end of each recursive call, which takes φ back to fewer assigned literals, is called the *backtracking step.*

A partial assignment ρ is maintained during the search and output if the formula turns out to be satisfiable. To increase efficiency, a key procedure in SAT solvers is the *unit propagation* [4], where unit clauses are immediately set to $true$ as outlined in Algorithm 1. In most implementations of DPLL, logical inferences can be derived with unit propagation. Thus, this procedure is used for identifying variables which must be assigned a specific value. If $\varphi|_\rho$ contains the empty clause, a *conflict* condition is declared, the corresponding clause of φ from which it came is said to be *violated* by ρ, and the algorithm backtracks. The literals whose negation do not appear in the formula, called *pure literals*, are also set to $true$ as a preprocessing step and, in some implementations, during the simplification process after every branching.

Algorithm 1: DPLL-recursive(φ, ρ)

```
$(\varphi, \rho) \leftarrow$ UnitPropagate($\varphi, \rho$)
if $\varphi$ contains the empty clause then
    return UNSAT
end
if $\varphi$ has no clauses left then
    Output $\rho$
    return $SAT$
end
$l \leftarrow$ a literal not assigned by $\rho$
if $DPLL\text{-}recursive(\varphi|_l, \rho \cup \{l\}) = SAT$ then
    return $SAT$
end
return $DPLL\text{-}recursive(\varphi|_{\neg l}, \rho \cup \{\neg l\})$

sub UnitPropagate($\varphi, \rho$)
    while $\varphi$ contains no empty clause but has a unit clause $\omega$ do
        $l \leftarrow$ the literal in $\omega$ not assigned by $\rho$
        $\varphi \leftarrow \varphi|_l$
        $\rho \leftarrow \rho \cup \{l\}$
    end
    return $(\varphi, \rho)$
```

Variants of this algorithm form the most widely used family of complete algorithms for the SAT problem. They are frequently implemented in an iterative manner, resulting in significantly reduced memory usage. The efficiency of state-of-the-art SAT solvers relies heavily on various features that have been developed, analysed and tested over the last two decades. These include fast unit propagation using watched literals, deterministic and randomized restart strategies, effective clause deletion mechanisms, smart static and dynamic branching heuristics and learning mechanisms. We will discuss learning mechanisms in the next section and refer the reader to [15] for more details about other strategies.

4.2 Conflict-Driven Clause Learning

One of the main reasons for the widespread use of SAT in many applications is that solvers based on clause learning are effective in practice [15]. The main idea is to cache "causes of conflict" as learned clauses, and utilize this information to prune the search in a different part of the search space encountered later. Since their inception in the mid-90s, *Conflict-Driven Clause Learning* (CDCL) SAT solvers have been applied, in many cases with remarkable success, to a number of practical applications [4]. The organization of CDCL SAT solvers is primarily inspired by the DPLL procedure.

In CDCL SAT solvers, each variable p is characterized by a number of properties, including the *value*, the *antecedent* and the decision level, denoted respectively by $\nu(p) \in \{false, true, u\}, \alpha(p) \in \varphi \cup \{nil\}$, and $\delta(p) \in \{-1, 0, 1, \ldots, |\mathcal{P}|\}$. A variable p that is assigned a value as the result of unit propagation is said to be *implied.* The unit clause ω used for implying this value is said to be the antecedent of p, that is, $\alpha(p) = \omega$. For variables that are decision variables or are unassigned, the antecedent is *nil*. Hence, antecedents are only defined for variables whose value is implied by other assignments. The decision level of a variable p denotes the depth of the decision tree at which the variables is assigned a value in $\{false, true\}$ or $\delta(p) = -1$ if p is still unassigned. The decision level associated with variables used for branching steps is specified by the search process, and denotes the current depth of the *decision stack.* Hence, a variable p associated with a decision assignment is characterized by having $\alpha(p) = nil$ and $\delta(p) > 0$. More formally, the decision level of p with antecedent ω is given by:

$$\delta(p) = \max(\{0\} \cup \{\delta(p')|p' \in \omega \land p' \neq p\}) \tag{4.1}$$

i.e. the decision of an implied literal is either the highest decision level of the implied literals in a unit clause, or it is 0 in case the clause is unit. The notation $p = v@d$ is used to denote that $\nu(p) = v$ and $\delta(p) = d$. Moreover, the decision level of a literal is defined as the decision level of its variable, that is, $\delta(l) = \delta(p)$ if $l = p$ or $l = \neg p$.

Example 4. Consider the formula

$$\begin{aligned}\varphi &= \omega_1 \land \omega_2 \land \omega_3 \\ &= (p \lor \neg s) \land (p \lor r) \land (\neg r \lor q \lor s)\end{aligned}$$

Assume that the decision assignment is $s = false@1$. Unit propagation yields no additional implied assignments. Assume that the second decision is $p = false@2$. Unit propagation yields the implied assignment $r = true@2$ and $q = true@2$. Moreover, $\alpha(r) = \omega_2$ and $\alpha(q) = \omega_3$.

During the execution of a DPLL based SAT solver, assigned variables as well as their antecedents define a directed acyclic graph $I = (V_I, E_I)$ referred to as the *implication graph* [23]. The vertices of this graph are defined by all assigned variables and one special node κ, $V_I \subseteq \mathcal{P} \cup \{\kappa\}$. The edges in the implication graph are obtained from the antecedent of each assigned variable: if $\omega = \alpha(p)$ then there is a directed edge from each variable in ω, other than p, to p. If unit propagation yields an unsatisfied clause ω_i, then a special vertex κ is used to represent the unsatisfied clause. In this case, the antecedent of κ is defined by $\alpha(\kappa) = \omega_j$.

The edges of I are formally defined below. Let $z, z_1, z_2 \in V_I$ be vertices in I, in order to derive the conditions for existence of edges in I, a number of predicates need to be defined first.

Definition 11 The predicate $\gamma(z, \omega)$ takes value 1 if, and only if, z is a literal in ω, and is defined as follows:

$$\gamma(z,\omega) = \begin{cases} 1 & \text{if } z \in \omega \vee \neg z \in \omega \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

This predicate can now be used for testing the value of a literal in z in a given clause. The predicate $\nu_0(z, \omega)$ takes value 1 if, and only if, z is a literal in ω and its value is $false$:

$$\nu_0(z,\omega) = \begin{cases} 1 & \text{if } \gamma(z,\omega) \wedge z \in \omega \wedge \nu(z) = false \\ 1 & \text{if } \gamma(z,\omega) \wedge \neg z \in \omega \wedge \nu(z) = true \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

The predicate $\nu_1(z, \omega)$ takes value 1 if, and only if, z is a literal in ω and its value is $true$:

$$\nu_1(z,\omega) = \begin{cases} 1 & \text{if } \gamma(z,\omega) \wedge z \in \omega \wedge \nu(z) = true \\ 1 & \text{if } \gamma(z,\omega) \wedge \neg z \in \omega \wedge \nu(z) = false \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

As a result, there is an edge from z_1 to z_2 in I if, and only if, the following predicate takes value 1:

$$\epsilon(z_1,z_2) = \begin{cases} 1 & \text{if } z_2 = \kappa \wedge \gamma(z_1, \alpha(\kappa)) \\ 1 & \text{if } z_2 \neq \kappa \wedge \alpha(z_2) = \omega \wedge \nu_0(z_1,\omega) \wedge \nu_1(z_2,\omega) \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

Consequently, the set of edges E_I of the implication graph I is given by:

$$E_I = \{(z_1, z_2) | \epsilon(z_1, z_2) = 1\} \tag{4.6}$$

$$\begin{tikzcd}
t = false@3 \arrow[dr, "\omega_1"] & & \\
& q = false@5 \arrow[dr, "\omega_3"] & \\
p = false@5 \arrow[ur, "\omega_1"] \arrow[dr, "\omega_2"'] & & s = true@5 \\
& r = false@5 \arrow[ur, "\omega_3"'] &
\end{tikzcd}$$

Figure 4.1: Implication graph for Example 5

Finally, observe that a labeling function for associating a clause with each edge can also be defined.

Definition 12 Let $\iota : V_I \times V_I \longrightarrow \varphi$ be a labeling function. Then $\iota(z_1, z_2)$, with $z_1, z_2 \in V_I$ and $(z_1, z_2) \in E_I$, is defined by $\iota(z_1, z_2) = \alpha(z_2)$.

Example 5. (Implication graph without conflict). Consider the CNF formula:

$$\begin{aligned}\varphi &= \omega_1 \wedge \omega_2 \wedge \omega_3 \\ &= (p \vee t \vee \neg q) \wedge (p \vee \neg r) \wedge (q \vee r \vee s)\end{aligned}$$

Assume decision assignment $t = false@3$. Moreover, assume that the current decision assignment is $p = false@5$. The resulting implication graph is shown in Figure 4.1.

Example 6. (Implication graph with conflict). Consider the CNF formula:

$$\begin{aligned}\psi &= \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6 \\ &= (p \vee t \vee \neg q) \wedge (p \vee \neg r) \wedge (q \vee r \vee s) \wedge (\neg s \vee \neg u) \wedge (y \vee \neg s \vee \neg x) \wedge (u \vee x)\end{aligned}$$

Assume decision assignments $y = false@2$ and $t = false@3$. Moreover, assume the current decision assignment $p = false@5$. The resulting implication graph is shown in Figure 4.2 and yields a conflict because the clause ω_6 becomes unsatisfied.

Algorithm 2, adapted from [4], shows the standard organization of a CDCL SAT solver, which essentially follows the organization of DPLL. With respect to DPLL, the main differences are the call to function ConflictAnalysis each time a conflict is identified, and the call to Backtrack when backtracking takes place. Moreover, the Backtrack procedure allows for backtracking non-chronologically.

In addition to the main CDCL function, the following auxiliary functions are used:

$$\begin{tikzcd}
t = false@3 \arrow[dr, "\omega_1"] & & & & \\
 & q = false@5 \arrow[dr, "\omega_3"] & & u = false@5 \arrow[dr, "\omega_6"] & \\
p = false@5 \arrow[ur, "\omega_1"] \arrow[dr, "\omega_2"'] & & s = true@5 \arrow[ur, "\omega_4"] \arrow[dr, "\omega_5"'] & & \kappa \\
 & r = false@5 \arrow[ur, "\omega_3"'] & & x = false@5 \arrow[ur, "\omega_6"'] & \\
 & & y = false@5 \arrow[ur, "\omega_5"'] & &
\end{tikzcd}$$

Figure 4.2: Implication graph for Example 6

```
Algorithm 2: CDCL($\varphi, \rho$)
if UnitPropagate($\varphi, \rho$) yields a conflict then
    return UNSAT
end
$dl \leftarrow 0$
while $\neg$AllVariablesAssigned($\varphi, \rho$) do
    $l \leftarrow$ PickBranchingVariable($\varphi, \rho$)
    $dl \leftarrow dl + 1$
    $\rho \leftarrow \rho \cup \{l\}$
    if UnitPropagate($\varphi, \rho$) yields a conflict then
        $\beta \leftarrow$ ConflictAnalysis($formula, \rho$)
        if $\beta < 0$ then
            return UNSAT
        end
        else
            Backtrack($\varphi$,$\rho$,$\beta$)
            $dl \leftarrow \beta$
        end
    end
end
```

- `UnitPropagate`: same as in DPLL, consists of the iterated application of the unit propagation procedure. If an unsatisfied clause is identified, then a conflict indication is returned.

- `AllVariablesAssigned`: tests whether all variables have been assigned, in which case the algorithm terminates indicating that the CNF formula is satisfiable.

- `PickBranckingVariable`: consists of selecting a variable to assign and deciding its value.

- `ConflictAnalysis`: consists of analyzing the most recent conflict and learning a new clause from the conflict. The organization of this procedure is described in Section 4.2.1.
- `Backtrack`: backtracks to the decision level computed by `ConflictAnalysis`.

Arguments to the auxiliary functions are assumed to be passed by reference. Hence, φ and ρ are supposed to be modified during execution of these functions.

The typical CDCL algorithm shown does not account for a few often used techniques, as for instance, search restarts and deletion policies. Search restarts cause the algorithm to restart itself, but keeping the learnt clauses. Clause deletion policies are used to decide learnt clauses that can be deleted, which allows the memory usage of the SAT solver to be kept under control.

4.2.1 Conflict Analysis

Each time the CDCL SAT solver identifies a conflict due to unit propagation, the conflict analysis procedure is invoked. As a result, one or more new clauses are learnt, and a backtracking decision level is computed. This procedure analyses the structure of unit propagation and decides which literals to include in the learnt clause.

The decision levels associated with assigned variables define a partial order of the variables. Starting from a given unsatisfied clause (represented in the implication graph as the vertex κ), the conflict analysis procedure visits variables implied at the most recent decision level, i.e., the current largest decision level, identifies the antecedents of these variables and keeps from the antecedents the literals assigned at decision levels less than the one being considered. This process is repeated until the most recent decision variable is visited.

Let d be the current decision level, let p be the decision variable, let $\nu(p) = v$ be the decision assignment and let ω be an unsatisfied clause identified with unit propagation. In terms of the implication graph, the conflict vertex κ is such that $\alpha(\kappa) = \omega$. Moreover, take $\odot$ to be the resolution operator. For two clauses ω_i and ω_j, for which there is a unique variable q such that one clause has a literal q and the other has literal $\neg q$, $\omega_i \odot \omega_j$ contains all the literals of ω_i and ω_j with the exception of q and $\neg q$.

The clause learning procedure used in SAT solvers can be defined by a sequence of selective resolution operations [3, 24], that at each step yields a new temporary clause.

Definition 13 First, define a predicate that holds if a clause ω has an implied literal l assigned at the current decision level d:

$$\xi(\omega, l, d) = \begin{cases} 1 & \text{if } l \in \omega \wedge \delta(l) = d \wedge \alpha(l) \neq nil \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

Let $\omega^{d,i}$, with $i = 0, 1, \ldots$, be the intermediate clause obtained after i resolution operations. Using the predicated defined by 4.7, this intermediate clause can be defined as follows:

$$\omega^{d,i} = \begin{cases} \alpha(\kappa) & \text{if } i = 0 \\ \omega^{d,i-1} \odot \alpha(l) & \text{if } i \neq 0 \wedge \xi(\omega^{d,i-1}, l, d) = 1 \\ \omega^{d,i-1} & \text{if } i \neq 0 \wedge \forall\, l\ \xi(\omega^{d,i-1}, l, d) = 0 \end{cases} \tag{4.8}$$

Equation 4.8 can be used for formalizing the clause learning procedure. The first condition, $i = 0$, denotes the initialization step given in κ in I, where all literals in the unsatisfied clause are added to the first intermediate clause. Afterwards, at each step i, a literal l assigned at the current decision level d is selected and the intermediate clause $\omega^{d,i-1}$ is resolved with the antecedent of l.

For an iteration i such that $\omega^{d,i} = \omega^{d,i-1}$, then a *fixed point* is reached, and $\omega_L \overset{\text{def}}{=} \omega^{d,i}$ represents the new learnt clause. Observe that the number of resolution operations represented by 4.8 is no greater than $|\mathcal{P}|$.

Example 7. (Clause learning)

Modern SAT solvers implement an additional refinement of Definition 13, by further exploiting the structure of implied assignments induced by unit propagation, which is a key aspect of the clause learning procedure [23]. The idea of exploiting the structure induced by unit propagation was further exploited with *Unit Implication Points* (UIPs). A UIP is a vertex u in the implication graph, such that every path from the decision vertex p to the conflict vertex κ contains u, and it represents an alternative decision assignment at the current decision level that results in the same conflict. The main motivation for identifying UIPs is to reduce the size of learnt clauses.

Clause learning finds other applications besides the key efficiency improvements to CDCL SAT solvers. One example is *clause reuse*. In a large number of applications, clauses learnt from a given CNF formula can often be reused for related CNF formulae.

Moreover, for unsatisfiable subformulae, the clauses learnt by a CDCL SAT solver encode a resolution refutation of the original formula. Given the way clauses are learnt in this solvers, each learnt clause can be explained by a number of resolution steps, each of which is a trivial resolution step. As a result, the resolution refutation can be obtained from the learnt clauses in linear time and space on the number of learnt clauses.

For unsatisfiable formulae, the resolution refutations obtained from the clauses learnt by a SAT solver serve as certificate for validating the correctness of the SAT solver. Moreover, resolution refutations based on clause learning find practical applications [4].

Besides allowing producing a resolution refutation, learnt clauses also allow identifying a subset of clauses that is also unsatisfiable. For example, a *minimally unsatisfiable subformula* can be derived by iteratively removing a single clause and checking unsatisfiability. Unnecessary clauses are discarded, and eventually a minimally unsatisfiable subformula is obtained.

4.3 Modern CDCL Solvers

4.3.1 MiniSat

4.3.2 Glucose

Bibliography

[1] *Normal Form Theory*, pages 173–184. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. 9

[2] 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

[3] P. Beame, H. A. Kautz, and A. Sabharwal. Towards understanding and harnessing the potential of clause learning. *CoRR*, abs/1107.0044, 2011. 21

[4] A. Biere, M. Heule, H. van Maaren, and T. Walsh. Conflict-driven clause learning sat solvers. *Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications*, pages 131–153, 2009. 15, 17, 19, 23

[5] P. Blackburn, M. De Rijke, and Y. Venema. *Modal Logic: Graph. Darst*, volume 53. Cambridge University Press, 2002. 2, 7

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

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

[8] S. A. Cook. The complexity of theorem proving procedures. In *stoc71*, pages 151–158, 1971. 14

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 3rd Edition.* MIT Press, 2009. 15

[10] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, July 1962. 15

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

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

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

[14] E. Giunchiglia, A. Tacchella, and F. Giunchiglia. Sat-based decision procedures for classical modal logics. *Journal of Automated Reasoning*, 28(2):143–171, 2002. 14

[15] C. P. Gomes, H. Kautz, A. Sabharwal, and B. Selman. Satisfiability solvers. *Foundations of Artificial Intelligence*, 3:89–134, 2008. 14, 15, 16

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

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

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

[19] 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. 8, 9, 12, 13

[20] C. Nalon, U. Hustadt, and C. Dixon. Ksp: A resolution-based prover for multimodal k. In N. Olivetti and A. Tiwari, editors, *Automated Reasoning: 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 – July 2, 2016, Proceedings*, pages 406–415, Cham, 2016. Springer International Publishing. 12, 13

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

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

[23] J. P. M. Silva and K. A. Sakallah. Grasp—a new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, pages 220–227. IEEE Computer Society, 1997. 17, 22

[24] J. P. M. Silva and K. A. Sakallah. Boolean satisfiability in electronic design automation. In *Proceedings of the 37th Conference on Design Automation (DAC-00)*, pages 675–680, NY, June 5–9 2000. ACM/IEEE. 21

[25] E. Spaan. *Complexity of Modal Logics*. PhD thesis, University of Amsterdam, 1993. 8

[26] L. Wos, G. A. Robinson, and D. F. Carson. Efficiency and completeness of the set of support strategy in theorem proving. *Journal of the ACM (JACM)*, 12(4):536–541, 1965. 13