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

Resumo

Palavras-chave: lógicas modais, resolução, sat-solvers

Abstract

Keywords: modal logics, resolution, sat-solvers

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Modal logics have been widely studied in Computer Science for allowing the characterization of complex systems that express notions in terms of knowledge, belief etc. Once a system has been specified in a logic language, it is possible to use proof methods to verify properties of such systems. In the literature, there are several theorem provers for modal logics. In this work, we focus on K$_S$P [25], a theorem prover for the basic multimodal language K_n, which implements the clausal resolution method proposed in [24]. Clauses are labelled by the modal level at which they occur, helping to restrict unnecessary applications of the resolution inference rules. Several refinements and simplification techniques in order to reduce the search space for a proof are implemented. To get the best performance for a particular formula, or class of formulae, it is important to choose the right strategy and optimizations. Currently, K$_S$P leaves these choices to the user, so we are interested in steps towards the implementation of an "auto-mode" in which the prover makes choices on its own, based on an analysis of the input.

K$_S$P performs well if the set of propositional symbols are uniformly distributed over the modal levels. However, when there is a high number of propositional symbols in just one particular level, the performance deteriorates. One reason is that the specific normal form we use always generates satisfiable sets of propositional clauses (i.e. clauses without modal operators). As resolution relies on saturation, this can be very time consuming. We are currently investigating the use of other tools in order to speed up the saturation process. For instance, *Boolean Satisfiability Solvers* can often solve hard structured problems with over a million variables and several million constraints [17]. We believe that we can take advantage of the theoretical and practical efforts that have been directed in improving the efficiency of such solvers.

Our implementation, which is work in progress, uses a SAT solver based on clause learning by conflict analysis. We feed this solver with the satisfiable sets of clauses generated and, each time it identifies a conflict in these sets due to unit propagation from some variable assignment, one or more new clauses are learnt from the conflict analysis procedure, which analyses the structure of unit propagation and decides which literals to include in the new clauses [4]. As mentioned before, as we already know that these sets are satisfiable, we are not particularly interested in the model generated by the SAT solver, but we believe that by carefully choosing the set of clauses and making use of the learnt clauses generated by MiniSat we may be able to reduce the time K_SP spends during saturation.

Chapter 2

Modal Logics

2.1

This chapter formally introduces K_n, a *propositional modal logic language*, semantically determined by an account of necessity and possibility [23].

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 extend classical logic by adding operators, known as modalities, 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 over 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 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 [22]. The satisfiability and validity of a formula depend on this structure. For instance, given a Kripke model that contains a set of possible worlds, a binary relation of accessibility between worlds and a valuation function that returns the value of a propositional symbols in a specific world, we say that a formula $\boxed{a} p$ is satisfiable at some world w of this model, if the valuation function establishes that p is true at all worlds accessible from w trough the accessibility relation indexed by a.

In the following, we will formally define the modal language. The syntax and semantics of K_n are given in Sections 2.2 and 2.3, respectively, and the definitions presented in these two sections follow those in [23].

2.2 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 $\Diamond_a$ is the dual 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:

- $\varphi \wedge \psi \stackrel{\text{def}}{=} \neg(\neg\varphi \vee \neg\psi)$ (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)
- $\textbf{false} \stackrel{\text{def}}{=} \varphi \wedge \neg\varphi$ (*falsum*)
- $\textbf{true} \stackrel{\text{def}}{=} \neg\textbf{false}$ (*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 $\Diamond_a l$, with $l \in \mathcal{L}$ and $a \in \mathcal{A}$. If l is a literal, we call $\neg l$ its complement and say that l and $\neg l$ form, in either order, a complementary pair.

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 $\varphi = \boxed{a}\,\Diamond_a p \vee \Diamond_a q, a \in \mathcal{A}$, 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 $\boxed{a}\Diamond(p \wedge \boxed{a}p)$, 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.

2.3 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

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

where W is a non-empty set of possible worlds with a distinguinshed world w_0, the root of $\mathfrak{M}$; each R_a, $a \in \mathcal{A}$, is a binary relation on W, that is, $R_a \subseteq W \times W$, and $\pi : W \times \mathcal{P} \longrightarrow \{\mathit{false}, \mathit{true}\}$ 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 $\mathfrak{M} = (W, w_0, R_1, \ldots, R_n, \pi)$ be a Kripke model, $w \in W$ and $\varphi, \psi \in \mathsf{WFF}_{\mathsf{K}_n}$. The *satisfiability relation*, denoted by $\langle \mathfrak{M}, w \rangle \models \varphi$, between a world w and a formula φ, is inductively defined by:

1. $\langle \mathfrak{M}, w \rangle \models p$ if, and only if, $\pi(w, p) = \mathit{true}$, for all $p \in \mathcal{P}$;
2. $\langle \mathfrak{M}, w \rangle \models \neg\varphi$ if, and only if, $\langle \mathfrak{M}, w \rangle \not\models \varphi$;

3. $\langle \mathfrak{M}, w\rangle \models \varphi \vee \psi$ if, and only if, $\langle \mathfrak{M}, w\rangle \models \varphi$ or $\langle \mathfrak{M}, w\rangle \models \psi$;

4. $\langle \mathfrak{M}, w\rangle \models \boxed{a}\varphi$ if, and only if, for all $t \in W$, $(w, t) \in R_a$ implies $\langle \mathfrak{M}, t\rangle \models \varphi$.

A formula $\varphi \in \mathsf{WFF}_{\mathsf{K}_n}$ is said to be *locally satisfiable* if there exists a Kripke model $\mathfrak{M} = (W, w_0, R_1, \ldots, R_n, \pi)$ such that $\langle \mathfrak{M}, w_0\rangle \models \varphi$. In this case we simply write $\mathfrak{M} \models_L \varphi$ to mean that $\mathfrak{M}$ locally satisfies φ. A model $\mathfrak{M} = (W, w_0, R_1, \ldots, R_n, \pi)$ is said to *globally satisfy* a formula φ, denoted $\mathfrak{M} \models_G \varphi$, if for all $w \in W$, we have $\langle \mathfrak{M}, w\rangle \models \varphi$. A formula φ is said to be *globally satisfiable* if there is a model $\mathfrak{M}$ such that $\mathfrak{M}$ 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 $\mathfrak{M}$ be the model illustrated in Figure 2.1. Take $\mathfrak{M} = (W, w_0, R, \pi)$, 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) $\pi(w, p) = \begin{cases} \textit{true} & \text{if } w = w_0 \\ \textit{false} & \text{otherwise} \end{cases}$

Note that both p and $\Box\neg p$ are satisfied in $\mathfrak{M}$. 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 propositional classical logic. Other examples of formulae satisfied by this model are: $p \wedge \Diamond\neg p$, $\Box\Box\neg p$ and $\Box\Box\Box\neg p$.

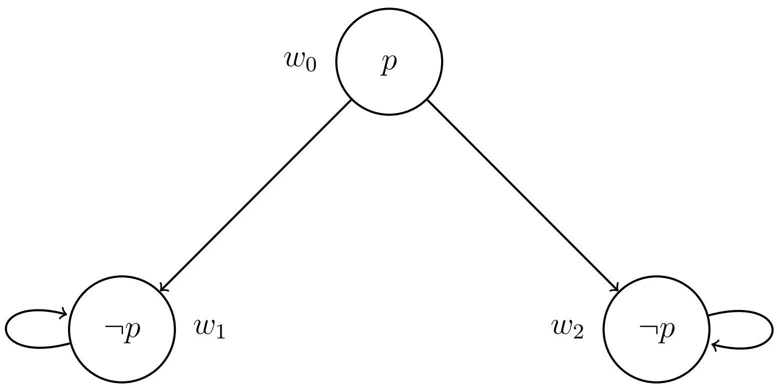
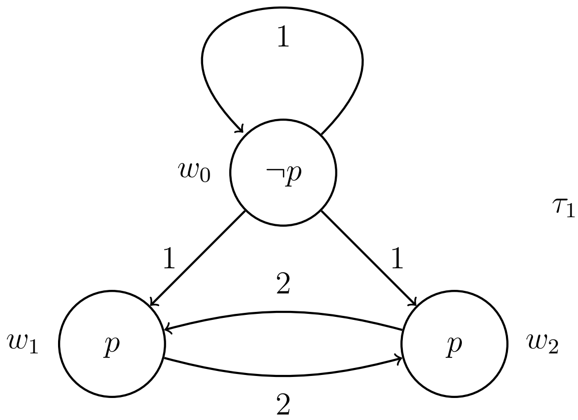
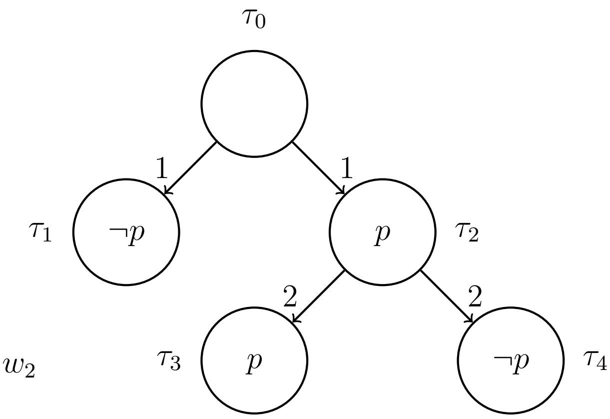


Figure 2.1: Example of a Kripke model for K_n



(a) Kripke model



(b) Tree model

Figure 2.2: Models that satisfy φ of Example 2

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.

As trees play an important role in computer science, we will take this opportunity to define them. By a tree $\mathfrak{T}$ 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 $\neg S^* tt$ [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 $\mathfrak{M} = (W, w_0, R_1, \ldots, R_a, \pi)$ 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 $\mathfrak{M}$. We sometimes say *depth* of $\mathfrak{M}$ to mean the longest path from the root to any world in W.

The following theorems have been adapted from the ones presented in [1].

Theorem 2.3.1. *Let $\varphi \in \mathsf{WFF}_{\mathsf{K}_n}$ be a formula and $\mathfrak{M} = (W, w_0, R_1, \ldots, R_n, \pi)$ be a model. Then $\mathfrak{M} \models_L \varphi$ if and only if there is a tree-like model $\mathfrak{M}'$ such that $\mathfrak{M}' \models_L \varphi$. Moreover, $\mathfrak{M}'$ is finite and its depth is bounded by mdepth(φ).*

The proof of Theorem 2.3.1 presented in [5] constructs a tree-like model $\mathfrak{M}'$ as a *generated submodel* of $\mathfrak{M}$, 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 2.3.2. *Let $\varphi, \varphi' \in \mathsf{WFF}_{\mathsf{K}_n}$ and $\mathfrak{M} = (W, w_0, R_1, \ldots, R_n, \pi)$ be a tree-like model such that $\mathfrak{M} \models_L \varphi$. If $(\varphi', \lambda', ml) \in \tau(\varphi, 1, 0)$ and φ' is satisfied in $\mathfrak{M}$, then there is $w \in W$, with $depth(w) = ml$, such that $\langle \mathfrak{M}, w \rangle \models \varphi'$. Moreover, the subtree rooted at w has height equals to $mdepth(\varphi')$.*

The proof of Theorem 2.3.2 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 φ depends only on its subformulae φ', only the subtrees of height $mdepth(\varphi')$ starting at level ml need to be checked. The bound on the height of the subtrees follows from Theorem 2.3.1 [24].

The *local satisfiability problem* for K_n corresponds to determining the existence of a model in which a formula is locally satisfied, while the *global satisfiability problem* corresponds to determining the existence of a model in which a formula is globally satisfied. These problems are proven to be PSPACE-complete [30], for local satisfiability, and EXPTIME , for global satisfiability[30].

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 [18]. Let K_n^* be the logic obtained by adding $\boxed{*}$ to K_n. A model $\mathfrak{M}^*$ for K_n^* is the pair $(\mathfrak{M}, R_*)$, where $\mathfrak{M} = (W, w_0, R_1, \ldots, R_n, \pi)$ 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 $\mathfrak{M}^*$, written $\langle \mathfrak{M}^*, w \rangle \models \boxed{*}\varphi$, if, and only if, for all $w' \in W$, we have that $\langle \mathfrak{M}^*, w' \rangle \models \varphi$. Therefore, let $\varphi \in \mathsf{WFF}_{\mathsf{K}_n}$ be a formula, we say that $\mathfrak{M} \models_G \varphi$, if, and only if, $\mathfrak{M}^* \models_L \boxed{*}\varphi$.

2.4 Proof Systems and Normal Forms

The *proof* of theorems, or the *deduction* of consequences of assumptions, in mathematics, typically proceeds by putting sentences in a list [21]. The assumptions are called *axioms*.

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 derivation of formulae from formulae through strict symbol manipulation [12]. 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 [15]. Therefore, a sound and complete calculus allows us to produce a proof that formulae are valid.

A sound and complete calculus for K_n finds a proof for a formula of this logic if, and only if, this formula is valid. A proof for this logic is defined below.

Definition 7 A K_n-*proof*, or a *proof for* K_n, is a finite sequence of formulae, each of which is an axiom, or follows from one or more earlier items in the sequence by applying a syntactic rule. The axioms for K_n are all instances of propositional tautologies plus the axiom K: $\boxed{a}(\varphi \Rightarrow \psi) \Rightarrow (\boxed{a}\varphi \Rightarrow \boxed{a}\psi)$ [5].

Propositional tautologies may contain modalities, for example, $\Diamond p \vee \neg\Diamond q$ is a tautology. As tautologies are valid, they are a safe starting point for modal reasoning.

There are many kinds of proof systems. They can be loosely divide into two broad categories: *synthetic* and *analytic* [15]. An analytic proof system decomposes the formula being proved into simpler and simpler parts. A synthetic proof system, on the other hand, builds its way up to the formula being proved. Analytic proof systems tend to be easier to use, since the field of play is sharply limited to the formula being proved [15].

The most common example of synthetic calculus is an *axiom system*. Certain formulae are taken as axioms. A proof starts with these axioms and, using inference rules that produce new formulae, builds up a sequence that finally ends with the formula being proved.

Tableau systems are one of the more common analytic proof systems. These are *refutational systems*, i.e., to prove a formula, we begin by negating it, then analysing the entailment of doing so. If it is the case that the consequences turn out to be impossible, we conclude that the original formula has been proved [15]. There are many varieties of tableaux, and the interesting thing about these systems is that, usually, proofs can be represented by trees of formulae.

Another way of categorizing proof systems involves the chaining in which the rules are applied to form a line of reasoning. If the chaining starts from a set of conditions and moves toward some (possibly remote) conclusion, the system is called *forward chaining*. If the conclusion is known, for instance, if it is a goal to be achieved, but the path to this conclusion is unknown, then reasoning backwards is called for, and the system is a *backward chaining* [13].

Backward chaining is a standard technique in automated deduction, often taking the form of a version of Robinson's *resolution rule* [20]. The fundamental question is to determine whether or not a given formula follows from a given set of formulae, and there are various techniques which can be applied to guide the search for a proof.

In the literature, there are several kinds of proof systems proposed for K_n. The one we use in this work is based in resolution, it was proposed by [24] and is presented in Chapter 3, resolution is also briefly introduced in the same chapter. The calculus used is proved to be sound and complete for K_n in [24], 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 [19]. 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 [14], that is, a proof that demonstrates the existence of a mathematical object by providing a method for creating this object.

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, for example.

The normal form used in this work is a layered normal form called Separated Normal Form with Modal Levels, originally proposed in [23]. This normal form is introduced in Chapter 3, as well as the translation rules for formulae in K_n. The proof that this translation preserves satisfiability can be found in [23].

Chapter 3

Modal-Layered Resolution

Resolution appeared in the early 1960's through investigations on performance improvements of refutational systems based on the *Herbrand's Theorem*, which allows a kind of reduction of first-order logic to propositional logic [6]. In particular, Prawitz' studies of such systems brought back the concept of unification. J. A. Robinson incorporated this concept on a refutational system, creating what was later known as resolution [27].

Resolution relies on saturation. A saturation-based theorem proving is usually characterized by a process in which new formulae are derived from given ones by thorough application of specified inference rules, with the ultimate goal of obtaining a contradiction. In addition, the current set of formulae is analyzed to identify the most promising inference rules to be applied next and to eliminate redundancies [2].

Numerous resolution systems have been proposed in the literature. The standard system has only one inference rule [2], showed in Equation 3.1, that takes two *clauses* with literals that form a complementary pair and generates a *resolvent*, where each clause, denoted by $\mathcal{C}$, is a disjunction of literals.

$$\text{[RES]} \ \frac{\mathcal{C}_i \lor l \qquad \neg l \lor \mathcal{C}_j}{\mathcal{C}_i \lor \mathcal{C}_j} \tag{3.1}$$

where $\mathcal{C}_i$ and $\mathcal{C}_j$ are possibly empty clauses, l is a literal and $\mathcal{C}_i \lor \mathcal{C}_j$ is the resolvent.

We use the constant **false** to denote the *empty clause*, i.e., the clause that contains no literals. Due to the associativity and commutativity properties of disjunction, one may see a clause as a set of literals. In this work we might abuse of set notation, writing $l \in \mathcal{C}$, when l is a literal of $\mathcal{C}$, and $\mathcal{C}_i \subseteq \mathcal{C}_j$, when all literals in $\mathcal{C}_i$ are also literals in $\mathcal{C}_j$.

A clause set is said to be *saturated* when no new relevant information can be generated [11]. Saturation, up to redundancy, of the initial clause set is a quite useful criteria when we are thinking in terms of termination proof and completeness proof [15] of a calculus.

Most of these systems work exclusively with clauses in a specific normal form. Resolution, as proposed by Robinson, is a refutationally complete theorem proving method [27]. Therefore, to show that a formula φ is valid, $\neg\varphi$ is translated into a normal form, then the inference rule is applied until either no new resolvents can be generated or a contradiction is obtained. This means that the search for a contradiction proceeds by saturating the given clause set, exhaustively applying the inference rule [2]. The contradiction implies that $\neg\varphi$ is unsatisfiable and hence, that φ is valid.

Resolution-based provers are widely implemented and tested. Besides reliable implementations, we can also profit from several complete strategies which can be extended to deal with modal resolution [23]. Such provers for multimodal logics require pruning of the search space for a proof in order to deal with the inherent intractability of the satisfiability problem for such logics. In [24] is presented a clausal modal-layered resolution calculus for K_n, which divides the clause set according to the modal depth at which each clause occurs. This calculus is introduced in Section 3.2.

3.1 Clausal Resolution

Clausal resolution is a simple and adaptable proof system for classical logics. It was proposed by Robinson in 1965 [27], and was claimed to be suitable to be performed by a computer, as, for propositional logic, it has only one inference rule that is applied. Robinson emphasizes that, from the theoretical point of view, an inference rule need only to be sound and effective, that is, it allows only logical consequences of premisses to be deduced and it must be algorithmically decidable whether an alleged application of the rule is indeed an application of it.

The single inference rule of this system of logic entails the *resolution principle*, namely: *From any two clauses $\mathcal{C}_i$ and $\mathcal{C}_j$ which contain a complementary pair of literals, one may infer a resolvent of $\mathcal{C}_i$ and $\mathcal{C}_j$* [27]. This one rule is machine-oriented, rather then human-oriented, in the sense of the preceding remarks, once that is no more need to a single step in a deduction to be simple enough to be apprehended as correct by a human mind in a single intellectual act.

In his paper, Robinson presents a formulation of first-order logic designed to be used as the basic theoretical instrument of the proposed computer theorem-proving program. For the purpose of this work, we are only interested in the calculus for propositional logic. Therefore, we will neither discuss the theory behind the Herbrand's Theorem nor the definition of the unification procedure, but, if curious, the reader can refer to [27] for more details.

Table 3.1: Derivation of $\mathcal{K}$ in Example 3

1.	$\neg p \vee q$		
2.	$\neg p \vee r$		
3.	$\neg r \vee q$		
4.	$\neg q \vee r$		
5.	$\neg r$		
6.	p		
7.	$\neg p \vee q$	[RES,3,2]	$= 1$
8.	$\neg p \vee r$	[RES,4,1]	$= 2$
9.	$\neg q \vee q$	[RES,4,3]	$=$ **true**
10.	$\neg r \vee r$	[RES,4,3]	$=$ **true**
11.	$\neg p$	[RES,5,2]	
12.	$\neg q$	[RES,5,4]	
13.	q	[RES,6,1]	
14.	r	[RES,6,2]	
15.	**false**	[RES,11,6]	contradiction

In the sense of this last remark, the only representational formalism needed is propositional logic. As resolution has only the RES rule, a proof consists of repeated application of this rule to the propositional clauses. These applications are sufficient to derive an empty clause if and only if the initial formula is unsatisfiable [17].

Example 3. Consider the set of clauses $\mathcal{K} = \{(\neg p \vee q), (\neg p \vee r), (\neg r \vee q), (\neg q \vee r), \neg r, p\}$. Table 3.1 shows an example of derivation of the empty clause for $\mathcal{K}$, proving that this set is unsatisfiable.

Despite its simplicity, unrestricted resolution is hard to implement efficiently due to the difficulty of finding good choices of clauses to resolve [17]; natural choices typically yield huge storage requirements. Thus, Robinson established two principles, namely *purity principle* and *subsumption principle*, to discuss the question of developing efficient resolution systems. A refutation system based on resolution incorporating the first principle tends to derive a smaller number of clauses. On the other hand, the incorporation of the second principle helps to increase the rate of which a contradiction is derived [27]. Such principles are called *search principles.* A third search principle is presented in terms of subsumption.

Definition 8 If $\mathcal{K}$ is any finite set of clauses, $\mathcal{C}$ is a clause in $\mathcal{K}$ and l a literal in $\mathcal{C}$ with the property that no literal in any other clause in $\mathcal{K}$ form a complementary pair with l, then l is said to be *pure* in $\mathcal{K}$.

The purity principle is then based on Theorem 3.1.1.

Theorem 3.1.1. *If $\mathcal{K}$ is any finite set of clauses, and $l \in \mathcal{C} \in \mathcal{K}$ is a pure literal in $\mathcal{K}$, then $\mathcal{K}$ is satisfiable if and only if $\mathcal{K} - \{\mathcal{C}\}$ is satisfiable.*

Definition 9 If $\mathcal{C}_i$ and $\mathcal{C}_j$ are two distinct nonempty clauses, we say that $\mathcal{C}_i$ *subsumes* $\mathcal{C}_j$ in the case that $\mathcal{C}_i \subseteq \mathcal{C}_j$.

Theorem 3.1.2 establishes the basic property of subsumption.

Theorem 3.1.2. *If $\mathcal{K}$ is any finite set of clauses, and $\mathcal{C}_j$ is any clause in $\mathcal{K}$ which is subsumed by some clause in $\mathcal{K} - \{\mathcal{C}_j\}$, then $\mathcal{K}$ is satisfiable if and only if $\mathcal{K} - \{\mathcal{C}_j\}$ is satisfiable.*

Theorems 3.1.1 and 3.1.2 are both proved in [27].

A particularly useful application of the subsumption principle is the following: suppose a resolvent $\mathcal{C}_R$ of $\mathcal{C}_i$ and $\mathcal{C}_j$ subsumes $\mathcal{C}_i$, then in adding $\mathcal{C}_R$ as a result of resolving $\mathcal{C}_i$ and $\mathcal{C}_j$, we may simultaneously delete, by the subsumption principle, $\mathcal{C}_i$. This combined operation entails the replacement of $\mathcal{C}_i$ by $\mathcal{C}_R$; accordingly this third principle is named as the *replacement principle.*

The application, to a finite set $\mathcal{K}$ of clauses, of any of the three search principles described, produces a set $\mathcal{K}'$ which either has fewer clauses than $\mathcal{K}$ or has the same number, but with one or more shorter clauses [27].

Example 4. Let $\mathcal{K}$ be the set of clauses showed in Table 3.2a. Prior to the thorough application of the RES rule, one may search for opportunities to apply the search principles, in order to reduce the number of clauses and increase the rate of which a contradiction is derived.

In this example, the third clause can be eliminated from the set of clauses through an application of the purity principle, as expressed in Table 3.2b, once there is no other clause that has the complementary pair of the literal u. Furthermore, as the clause 5 subsumes the clause 4, this last one can also be eliminated from the set of clauses, leaving only the clauses in Table 3.2c. The application of RES to clauses 1 and 2 generates the resolvent $\neg q$, which subsumes both clauses, hence, by the replacement principle, both may be replaced by the clause 7, as showed in Table 3.2d. Finally, the application of RES to the clauses 5 and 7, generates the clause 8 as a resolvent, and then resolving this one with the clause 6 will generate the empty clause, thus, a contradiction.

Applying the search principles allowed us to go from a set of six clauses with 4 literals to a set of four with only 2 literals.

Table 3.2: Derivation schemes for Example 4

(a) Initial clauses		(b) Purity principle		(c) Subsumption		(d) Replacement		
1.	$\neg q \vee \neg r$	1.	$\neg q \vee \neg r$	1.	$\neg q \vee \neg r$	5.	$\neg p \vee q$	
2.	$\neg q \vee r$	2.	$\neg q \vee r$	2.	$\neg q \vee r$	6.	p	
3.	$u \vee \neg p$	4.	$\neg p \vee q \vee \neg r$	5.	$\neg p \vee q$	7.	$\neg q$	[RES,1,2]
4.	$\neg p \vee q \vee \neg r$	5.	$\neg p \vee q$	6.	p	8.	$\neg p$	[RES,5,7]
5.	$\neg p \vee q$	6.	p			9.	**false**	[RES,6,8]
6.	p							

There are further principles of the same general sort, possibly less simple than the ones presented earlier, which Robinson considers to be merely a brief view of the possible approaches to the efficiency problem of resolution systems.

3.2 Modal-Layered Resolution Calculus for K_n

The calculus presented in this section requires a translation into a more expressive modal language, where labels are used to express semantic properties of a formula. This transformation is presented in Section 3.2.1. Furthermore, this calculus makes use of labelled resolution in order to avoid unnecessary applications of the inference rules [24]. For instance, we do not apply resolution to clauses at different modal levels, since they are not, in fact, contradictory.

3.2.1 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 [23], hence, all the definitions in this section are taken from [23]. A formula in SNF_{ml} is a conjunction of *clauses* where the modal level in which they occur is made explicit in the syntax 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 $\mathfrak{M}^* = (W, w_0, R_1, \ldots, R_n, R_*, \pi)$ be a tree-like model and take $\varphi \in \mathsf{WFF}_{\mathsf{K}_n}$.

Definition 10 Satisfiability of labelled formulae is given by:

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

2. $\mathfrak{M}^* \models_L * : \varphi$ if, and only if, $\mathfrak{M}^* \models \boxed{*}\varphi$

Clauses in SNF_{ml} are defined as follows.

Definition 11 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 [26].

Definition 12 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 13 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 [24], 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 locally 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 5.

3.2.2 Calculus

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 applications, and therefore improving the efficiency of the proof procedure [25].

The modal-layered resolution calculus, proposed by Nalon, Hustadt and Dixon in [24], comprises a set of inference rules, given in Table 3.3, 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 $\sigma : \mathscr{P}(\mathbb{N} \cup \{*\}) \longrightarrow$

Table 3.3: Inference rules

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

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

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

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

$\mathbb{N} \cup \{*\}$, where $\sigma(\{ml, *\}) = ml$ and $\sigma(\{ml\}) = ml$, otherwise, σ is undefined. The inference rules showed in Table 3.3 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, $ml + 1$.

Definition 14 Let $\mathcal{K}$ be a set of clauses in SNF_{ml}. A *derivation* from $\mathcal{K}$ is a sequence of sets $\mathcal{K}_0, \mathcal{K}_1, \ldots$ where $\mathcal{K}_0 = \mathcal{K}$ and, for each $i > 0$, $\mathcal{K}_{i+1} = \mathcal{K}_i \cup \{D\}$, where D is the resolvent obtained from $\mathcal{K}_i$ by an application of either LRES, MRES, GEN1, GEN2 or GEN3. It is also required that D is in simplified form, $D \notin \mathcal{K}_i$ and that D is not a tautology. A *refutation* for $\mathcal{K}$ is a finite derivation that contains the empty clause [24].

The proofs for termination, soundness and completeness of this calculus can be found in [24].

Example 6.

Example 7.

3.2.3 K$_S$P

In this section, we briefly introduce K$_S$P, the theorem prover presented in [25] for the basic multimodal logic K_n, which implements a variation of the set of support strategy [31] for the modal resolution-based calculus described in Section 3.2.

K$_S$P was designed to support experimentation with different combinations of refinements of its basic calculus. Refinements and options for processing and preprocessing the input are coded as independently as possible in order to allow for the easy addition and testing of new features, even though this may not lead to optimal performance, since techniques need to be applied one after another, whereas most tools would apply them all together, but this helps to evaluate how the different options independently contribute to achieve efficiency [25].

The results showed in [25] indicates that K_SP works well on problems with high modal depth where the separation of modal layers can be exploited to improve the efficiency of reasoning. Although, as with all provers that provide a variety of strategies and optimizations, to get the best performance for a particular formula or class or formulae, it is important to choose the right strategy and optimizations. K_SP leaves this choice to the user. The same applies to the transformation to the layered normal form.

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 [17]. 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 [16].

In the context of SAT solvers for propositional provers, the underlying representational formalism is propositional logic [17]. We are interested in formulae in *Conjunctive Normal Form* (CNF): φ is in CNF if it is a conjunction of clauses. 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. 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 a value in the set $\{\mathit{false}, \mathit{true}\}$. 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 an 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 $\pi : \mathcal{P} \longrightarrow \{\mathit{false}, \mathit{true}\}$, 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 [17].

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 [17]. 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 describes a backtracking algorithm to the search problem of finding a satisfying assignment for a formula in the space of partial 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 [17].

Algorithm 1, DPLL-recursive(φ, ρ), where ρ corresponds to a partial assignment of the CNF formula φ, 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

Algorithm 1: DPLL-recursive(φ, ρ)

$(\varphi, \rho) \leftarrow$ `UnitPropagate(`φ, ρ`)`
if φ *contains the empty clause* **then**
  return UNSAT
end
if φ *has no clauses left* **then**
  Output ρ
  return SAT
end
$l \leftarrow$ a literal not assigned by ρ
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(`φ, ρ`)`
  while φ *contains no empty clause but has a unit clause* $\mathcal{C}$ **do**
    $l \leftarrow$ the literal in $\mathcal{C}$ not assigned by ρ
    $\varphi \leftarrow \varphi|_l$
    $\rho \leftarrow \rho \cup \{l\}$
  end
  return (φ, ρ)

to *true* as a preprocessing step and, in some implementations, during the simplification process after every branching.

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 [17] 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 [17]. 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 $\nu(p) \in \{\mathit{false}, \mathit{true}, u\}$, where $\nu(p) = u$ means that p is still unassigned, $\alpha(p) \in \varphi \cup \{\mathit{nil}\}$, 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 $\mathcal{C}$ used for implying this value is said to be the antecedent of p, that is, $\alpha(p) = \mathcal{C}$. 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 $\{\mathit{false}, \mathit{true}\}$ 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 $\alpha(p) = \mathit{nil}$ and $\delta(p) > 0$. More formally, the decision level of p with antecedent $\mathcal{C}$ is given by:

$$\delta(p) = \max(\{0\} \cup \{\delta(p') \mid p' \in \mathcal{C} \wedge p' \neq p\}) \tag{4.1}$$

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 8. Consider the formula

$$\begin{aligned} \varphi &= \mathcal{C}_1 \wedge \mathcal{C}_2 \wedge \mathcal{C}_3 \\ &= (p \vee \neg s) \wedge (p \vee r) \wedge (\neg r \vee q \vee s) \end{aligned}$$

Assume that the decision assignment is $s = \mathit{false}@1$. Unit propagation yields no additional implied assignments. Assume that the second decision is $p = \mathit{false}@2$. Unit propagation yields the implied assignments $r = \mathit{true}@2$ and $q = \mathit{true}@2$. Therefore, $\pi = \{(s, \mathit{false}), (p, \mathit{false}), (r, \mathit{true}), (q, \mathit{true})\}$ is a satisfying assignment for φ, since π makes φ true. Moreover, $\alpha(r) = \mathcal{C}_2$ and $\alpha(q) = \mathcal{C}_3$.

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* [28]. The vertices of this graph are defined by all assigned variables and one special node ε, $V_I \subseteq \mathcal{P} \cup \{\varepsilon\}$. The edges in the implication graph are obtained from the antecedent of each assigned variable: if $\alpha(p) = \mathcal{C}$ then there is a directed edge from each variable in

$\mathcal{C}$, other than p, to p. If unit propagation yields an unsatisfied clause $\mathcal{C}_i$, then a special vertex ε is used to represent the unsatisfied clause. In this case, the antecedent of ε is defined by $\alpha(\varepsilon) = \mathcal{C}_i$.

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 15 The predicate $\gamma(z, \mathcal{C})$ takes value 1 if, and only if, z is a literal in $\mathcal{C}$, and is defined as follows:

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

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

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

The predicate $\nu_1(z, \mathcal{C})$ takes value 1 if, and only if, z is a literal in $\mathcal{C}$ and its value is *true*:

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

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:

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

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

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

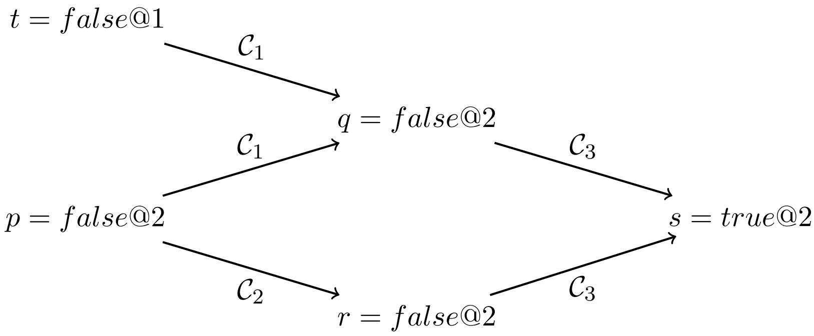


Figure 4.1: Implication graph for Example 9

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

> **Definition 16** 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 9. (Implication graph without conflict). Consider the CNF formula:

$$\begin{aligned}\varphi &= \mathcal{C}_1 \wedge \mathcal{C}_2 \wedge \mathcal{C}_3 \\ &= (p \vee t \vee \neg q) \wedge (p \vee \neg r) \wedge (q \vee r \vee s)\end{aligned}$$

Assume the decision assignment $t = \mathit{false}@1$ has been taken. Moreover, assume that the current decision assignment is $p = \mathit{false}@2$. Unit propagation yields the implied assignments $q = \mathit{false}@2$, $r = \mathit{false}@2$ and $s = \mathit{true}@2$. The resulting implication graph is shown in Figure 4.1. As all variables have been assigned a value and the implication graph does not contain the vertex ε, this set of decision assignments forms a satisfying assignment for φ.

Example 10. (Implication graph with conflict). Consider the CNF formula:

$$\begin{aligned}\psi &= \mathcal{C}_1 \wedge \mathcal{C}_2 \wedge \mathcal{C}_3 \wedge \mathcal{C}_4 \wedge \mathcal{C}_5 \wedge \mathcal{C}_6 \\ &= (p \vee t \vee \neg q) \wedge (p \vee \neg r) \wedge (q \vee r \vee s) \wedge (\neg s \vee \neg u) \wedge (y \vee \neg s \vee \neg x) \wedge (u \vee x)\end{aligned}$$

Assume the decision assignments $y = \mathit{false}@2$ and $t = \mathit{false}@3$. Moreover, assume the current decision assignment $p = \mathit{false}@5$. Unit propagation yields the implied assignments $q = \mathit{false}@2$, $r = \mathit{false}@2$, $s = true@2$, $u = \mathit{false}@5$ and $x = false@5$. These last two assignments generate a conflict once the clause $\mathcal{C}_6$ becomes unsatisfied. Therefore, the

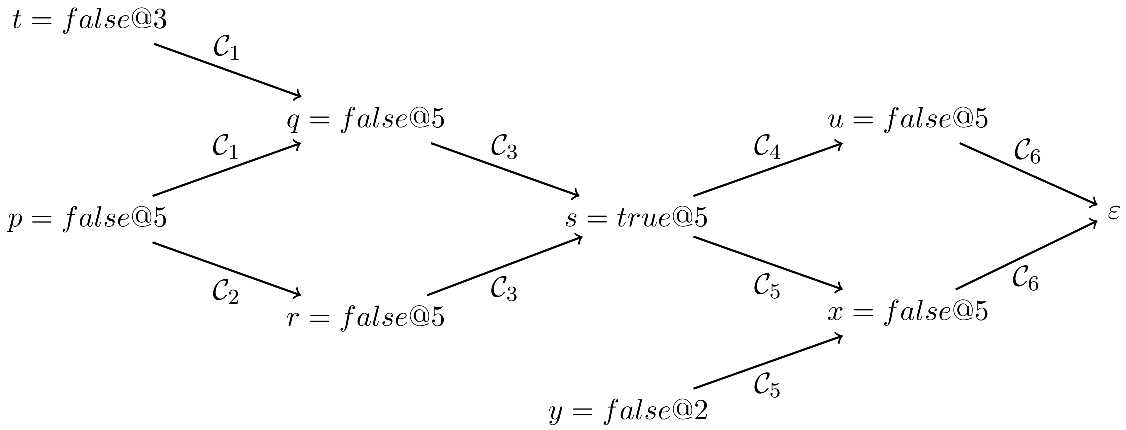


Figure 4.2: Implication graph for Example 10

resulting implication graph has the conflict vertex, as shown in Figure 4.2, with $\alpha(\varepsilon) = \mathcal{C}_6$. Hence, this set of decision assignments does not satisfy ψ.

Algorithm 2, adapted from [4], shows the standard structure of a CDCL SAT solver, which essentially follows the one from 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.

Algorithm 2: CDCL(φ, ρ)

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

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

- `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 ε), 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 $\mathcal{C}$ be an unsatisfied clause identified with unit propagation.

In terms of the implication graph, the conflict vertex ε is such that $\alpha(\varepsilon) = \mathcal{C}$. Moreover, take $\odot$ to be the resolution operator. For two clauses $\mathcal{C}_i$ and $\mathcal{C}_j$, for which there is a unique variable q such that one clause has a literal q and the other has literal $\neg q$, $\mathcal{C}_i \odot \mathcal{C}_j$ contains all the literals of $\mathcal{C}_i$ and $\mathcal{C}_j$ 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, 29], that at each step yields a new temporary clause.

Definition 17 First, define a predicate that holds if a clause $\mathcal{C}$ has an implied literal l assigned at the current decision level d:

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

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

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

Equation 4.8 can be used for formalizing the clause learning procedure. The first condition, $i = 0$, denotes the initialization step given in ε 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 $\mathcal{C}^{d,i-1}$ is resolved with the antecedent of l.

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

Example 11. (Clause learning) Consider Example 10. The application of clause learning to this example results in the intermediate clauses shown in Table 4.1. The resulting learnt clause is $\mathcal{C}_L = \mathcal{C}^{5,6} = (p \vee t \vee y)$. Alternatively, this clause can be obtained by inspecting the graph in Figure 4.2 and selecting the literals assigned at decision levels less than the current decision level 5, i.e. $t = \mathit{false}@3$ and $y = \mathit{false}@2$, and by selecting the current decision assignment $p = \mathit{false}@5$.

Modern SAT solvers implement an additional refinement of Definition 17, by further exploiting the structure of implied assignments induced by unit propagation, which is a key aspect of the clause learning procedure [28]. The idea of exploiting the structure

Table 4.1: Resolution steps during clause learning

$\mathcal{C}^{5,0} = \{u, x\}$	Literals in $\alpha(\varepsilon)$
$\mathcal{C}^{5,1} = \{\neg s, x\}$	Resolve with $\alpha(u) = \mathcal{C}_4$
$\mathcal{C}^{5,2} = \{\neg s, y\}$	Resolve with $\alpha(x) = \mathcal{C}_5$
$\mathcal{C}^{5,3} = \{q, r, y\}$	Resolve with $\alpha(s) = \mathcal{C}_3$
$\mathcal{C}^{5,4} = \{p, t, r, y\}$	Resolve with $\alpha(q) = \mathcal{C}_1$
$\mathcal{C}^{5,5} = \{p, t, y\}$	Resolve with $\alpha(r) = \mathcal{C}_2$
$\mathcal{C}^{5,6} = \{p, t, y\}$	No more resolution operations given

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 ε 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 the production of 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

Apart from conflict analysis, modern solvers include techniques like lazy data structures, search restarts, conflict-driven branching heuristics and clause deletion strategies [4].

4.3.1 MiniSat

4.3.2 Glucose

Chapter 5

Discussion and Future Work

5.1 Related Work

5.2 Future Work

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. 8

[2] L. Bachmair and H. Ganzinger. Resolution theorem proving. *Handbook of automated reasoning*, 1:19–99, 2001. 12, 13

[3] P. Beame, H. A. Kautz, and A. Sabharwal. Towards understanding and harnessing the potential of clause learning. *CoRR*, abs/1107.0044, 2011. 29

[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. 2, 22, 24, 27, 30

[5] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic: Graph. Darst*, volume 53. Cambridge University Press, 2002. 3, 8, 9, 10

[6] M. A. Casanova. *Programação em lógica e a linguagem Prolog.* E. Blucher, 1987. 12

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

[8] S. A. Cook. The complexity of theorem proving procedures. In *stoc71*, pages 151–158, 1971. 21

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 3rd Edition.* MIT Press, 2009. 22

[10] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, July 1962. 22

[11] N. Dershowitz and C. Kirchner. Abstract saturation-based inference. In *Logic in Computer Science, 2003. Proceedings. 18th Annual IEEE Symposium on*, pages 65–74. IEEE, 2003. 12

[12] N. Eisinger and J. Ohlbach. Deduction systems based on resolution. Technical report, Saarbruecken, 1991. 9

[13] E. Feigenbaum, P. McCorduck, and H. P. Nii. *The rise of the expert company*, volume 240. Times Books New York, 1988. 10

[14] K. Fine. Normal forms in modal logic. *Notre Dame J. Formal Logic*, 16(2):229–237, 04 1975. 11

[15] M. Fitting and R. L. Mendelsohn. *First-order modal logic*, volume 277. Springer Science & Business Media, 2012. 9, 10, 12

[16] E. Giunchiglia, A. Tacchella, and F. Giunchiglia. Sat-based decision procedures for classical modal logics. *Journal of Automated Reasoning*, 28(2):143–171, 2002. 21

[17] C. P. Gomes, H. Kautz, A. Sabharwal, and B. Selman. Satisfiability solvers. *Foundations of Artificial Intelligence*, 3:89–134, 2008. 1, 14, 21, 22, 23

[18] V. Goranko and S. Passy. Using the universal modality: gains and questions. *Journal of Logic and Computation*, 2(1):5–30, 1992. 9

[19] H. Hanβmann. *Normal Form Theory*, pages 173–184. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. 11

[20] J. A. Harland, D. J. Pym, and M. Winikoff. Forward and backward chaining in linear logic. *Electronic Notes in Theoretical Computer Science*, 37:1–16, 2000. 10

[21] S. C. Kleene. *Mathematical logic*. Courier Corporation, 2002. 9

[22] S. A. Kripke. Semantical analysis of modal logic I. *Zeitschr. Math. Logik Grund. Math*, 9:67–96, 1963. 4

[23] C. Nalon and C. Dixon. Clausal resolution for normal modal logics. *J. Algorithms*, 62(3-4):117–134, 2007. 3, 4, 11, 13, 16

[24] 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. 1, 9, 10, 11, 13, 16, 18, 19

[25] 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. 1, 18, 19, 20

[26] D. A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293–304, 1986. 17

[27] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, Jan. 1965. 12, 13, 14, 15

[28] 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. 24, 29

[29] 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. 29

[30] E. Spaan. *Complexity of Modal Logics.* PhD thesis, University of Amsterdam, 1993. 9

[31] 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. 19