Decision Procedures for Equality Logic with Uninterpreted Functions

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op woensdag 29 juni 2005 om 16.00 uur

door

Olga Tveretina

geboren te Tomsk, Rusland

Dit proefschrift is goedgekeurd door de promotor:

prof.dr.ir. J.F. Groote

Copromotor:
dr. H. Zantema

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Tveretina, Olga

Decision Procedures for Equality Logic with Uninterpreted Functions/ by
Olga Tveretina. - Eindhoven : Technische Universiteit Eindhoven, 2005.
Proefschrift. - ISBN 90-386-0624-9
NUR 993
Subject headings: automated theorem proving / formal proofs /
computer applications / theoretical computer science
CR Subject Classification (1998): I.2.3, F.4.1, B.5.2, D.2.4, F.3.1

To my daughter, Khrystyna

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

IPA dissertation series

Printing: Eindhoven University Press

Contents

Acknowledgements

I am greatly indebted to many people who helped me during this long thesis marathon.

First and foremost I have to thank dr. H. Zantema, who acted as my daily supervisor. He helped me to start my research and enabled me to develop my own ideas.

I am grateful to my promotor, prof.dr.ir. J.F. Groote, for giving me the opportunity to become a Ph.D. student in his group and his willingness to let me explore research fields of my own choice.

Thanks to all members of my committee for taking their time to serve on the committee, and for the feedback they provided. Special thanks go to dr. R.P. Nederpelt for his comments and suggestions on a previous draft of my thesis. I also appreciate the efforts of the other committee members, prof.dr. R. Drechsler and prof.dr. W.J. Fokkink.

Academic encouragement came from many fronts. I thank prof. Alexander I. Nosich for his subtle background influence in furthering my academic career.

Fortunately, I have not been working alone. I discovered the joy to work with Jaco van de Pol. He provided constructive remarks and appropriate criticism, and he was just a nice person to work with. My thanks also go to another co-author Bahareh Badban.

During these years in Eindhoven, I had the pleasure to interact with Natasha Stash, Slava Pranovich, Elena Lomonova, Evegeniya Balmashnova and Alexei Balmashnov. I am grateful to Arjan Mooij, who has been a perfect roommate and assisted me with many problems.

It has been a pleasure to work at the TU/e. With great pleasure I recall all the colleagues from the OAS group.

Last but not least, I would like to thank my friends and family members, who have supported me in many different ways. I would like to thank my mother and my sister for their love, encouragement and support throughout my whole life. I thank my daughter Khrystyna for her patience during the long time I was writing my thesis. I am very grateful to Jacques and Kitty van Hartingsveldt, and Jan and Ria Tuitel for their participation in my life. I also thank all my friends from Ukraine.

Finally, the greatest thanks go to my husband Rudie for his love and support.

Chapter 1

Introduction

"The study of mathematics, like the Nile, begins in muniteness, but ends in magnificence"

[C.C. Colton]

"Where shall I begin, please your Majesty?" He asked. "Begin at the beginning," the King said, very gravely, "and go on till you come to the end: then stop."

["Alice's Adventures in Wonderland", Lewis Carroll]

Automated deduction describes mechanized reasoning with a formal system, called an *automated theorem prover* (ATP). Automated theorem proving systems try to answer the question whether a given *hypothesis* is valid under a set of *axioms*. Depending on the kind of logic, the problem of answering this question varies from trivial to impossible. For the frequent case of propositional logic, the problem is *decidable* but *NP-complete*, and hence only exponential time algorithms are believed to exist. For first-order logic it is *semi-decidable*, i.e., given unbounded resources, any true hypothesis can eventually be proven, but invalid ones cannot always be recognized. Despite these theoretical limits, practical theorem provers can solve many hard problems in these logics. Of particular interest for us are the cases when hypotheses and axioms are expressed as formulae in first-order predicate logic or its subsets.

Since the 60's, a large variety of automated theorem provers based on numerous proof methods have been developed. Theorem provers like Otter [McCune and Wos, 1997], DISCOUNT [Avenhaus et al., 1995], SPASS [Weidenbach et al., 1996] and SETHEO [Schumann, 1997, Letz et al., 1992] have proved to be quite successful. Theorem provers are used for the verification of protocols [Schumann, 1997], the retrieval of mathematical theorems [Dahn and Wernhard, 1997], and the verification of hardware and software designs [Bryant et al., 2001, Pnueli et al., 1999]. Theorem provers can be used to synthesize larger programs from standard blocks and to prove the correctness of the resulting system [Stickel et al., 1994, Baalen et al., 1998].

Improvements in the design and implementation of an automated theorem prover are expected to lead to major enhancements of its performance. We are always trying to find new methods, strategies and data structures to improve the search process either by reducing the size of the search space or by reducing the amount of repetitive computations.

1.1 Formal Systems

A *formal system* is a set of rules that defines how to manipulate some defined set of symbols, i.e. a formal system consists of a formal language, axioms, and a set of *inference rules* or other means of deriving further statements. A *proof* is a sequence of syntactical transformations according to the inference rules. A formal systems is often referred to as a *calculus*.

The required properties of a satisfactory formal system are completeness, consistency and decidability. A formal systems is *complete* if it is possible either to prove or to disprove any proposition that can be expressed in the system. *Consistency* means that it is not possible to both prove and disprove a proposition in the system.

Formulae are finite strings of symbols. In this thesis we consider first-order predicate logic as a formal language. Atomic formulae are the simplest expressions in first-order logic, and more complex formulae can be constructed with *logical connectives* and *quantifiers*.

A *decision procedure* is a set of instructions (inference rules) that provides the mechanical means by which an answer to a question can be obtained in a finite number of steps. One of the things we expect from a decision procedure is that it yields only true conclusions, i.e. it is *sound*, and we would like it to yield all true conclusions, i.e. that it is *complete*.

In this thesis we mean by a decision procedure an algorithm that can reason about the satisfiability of logical formulae in a given decidable theory, and that always terminates with an answer “satisfiable” or “unsatisfiable”.

Purely syntactical reasoning is often sufficient to determine the validity or satisfiability of a formula. Therefore, we will mainly discuss techniques for theorem proving that are based on syntactical reasoning.

1.2 Reasoning with equality

The notation of equality is central to almost all formal methods. An equality relation plays an important role in most mathematical and verification problems as checking that systems like software, hardware, or protocols meet desired specifications. Thus, equations occur in more than half of the benchmarks in the TPTP problem library [Sutcliffe et al., 1994]. Functional programming languages like *Haskell* [Bird, 1998] are based on transformations of equations. Hence, equalities are suited to model complex formal expressions and to perform operations with them. Therefore, techniques for handling the equality relation are essential for the successful application of theorem provers to most interesting first order problems.

The equality relation is a congruence relation. Therefore, to reason about equality one can always add standard axioms (reflexivity, symmetry, transitivity, and substitution). However, symmetry and transitivity allow infinite derivations, and as a result this approach results in a huge, or even infinite, search space to be explored. Hence, an efficient handling of equality requires special rules. The most significant of these rules is *paramodulation* [Robinson and Wos, 1969] which allows to replace equals by equals.

Based on this principle, *superposition calculi* [Bachmair and Ganzinger, 1990, Nieuwenhuis and Rubio, 1992, Bachmair and Ganzinger, 1994] have been developed. They combine techniques like *resolution* with paramodulation and *ordering* restrictions. Approaches based on superposition calculi are very efficient.

1.3 Overview of the thesis

In this thesis we present a number of techniques for checking the satisfiability of decidable subsets of first order logic containing equality. An aim of this thesis is rather to develop novel calculi than an efficient tool for checking satisfiability.

As general logical framework we use a quantifier-free first-order predicate logic. Definitions of this framework and notations used throughout the thesis are given in Chapter 2.

Chapter 3 describes the basic procedures for checking satisfiability of propositional formulae: the DP procedure, the DPLL procedure and BDDs. These techniques are rather families of algorithms, than algorithms. Their behavior depends on choices made during execution.

Since we use standard notations, readers familiar with first-order logic and theorem proving can skip Chapter 2 and Chapter 3.

Most of the chapters starting from Chapter 4 are based on papers that have been published or appeared as technical reports [Tveretina and Zantema, 2002, Badban et al., 2004a,b, Tveretina and Zantema, 2003, 2004, Tveretina, 2004a,b, Pol and Tveretina, 2005].

In Chapter 4 we give a formal description of resolution and analyze the relation between DPLL and resolution in details [Tveretina and Zantema, 2002]. It is well-known that a DPLL refutation can be straightforwardly transformed into a resolution refutation of similar length. In this thesis, a transformation of a DPLL refutation to a resolution refutation of a number of steps which is essentially less than the number of unit resolution steps applied in the DPLL refutation, is introduced.

GDPLL, a generalization of the DPLL procedure [Badban et al., 2004a,b, Tveretina, 2004b,a, Pol and Tveretina, 2005] is presented in Chapter 6. It solves the satisfiability problem for decidable fragments of quantifier-free first-order logic. Sufficient properties are identified for proving soundness, termination and completeness of GDPLL.

Approaches to decide satisfiability of the logic of equality with uninterpreted functions (EUF) are described in Chapter 7. This type of logic has been proposed for verifying abstract hardware designs. Fast satisfiability checking over this logic is important for such verifications to be successful. In the past years various procedures for checking satisfiability of such formulae have been suggested.

In the past few years several different procedures for checking satisfiability of EUF-formulae, which are based on a reduction to equality logic, have been proposed. In Chapter 5 we give a new approach for deciding satisfiability of equality logic formulae in conjunctive normal form [Tveretina and Zantema, 2003, 2004]. Central in this approach is a single proof rule called equality resolution. For this single rule we prove soundness and completeness. Based on this rule we propose a complete procedure for checking satisfiability of this kind of formulae and prove its correctness.

A new procedure for checking satisfiability of EUF-formulae based on the DPLL method and substituting equals for equals to maintain the transitivity of equalities [Tveretina, 2004b,a] is presented in Chapter 8.

Finally, in Chapter 9 we extend BDDs for propositional logic to the equality logic with uninterpreted functions [Pol and Tveretina, 2005]. It is proved that all paths of these extended BDDs are satisfiable. It can be checked in constant time whether the formula is a tautology, a contradiction, or just satisfiable. This approach was implemented by Jaco van de Pol. The implementation works within the special purpose theorem prover for the μCRL toolset [Blom et al., 2003].

Chapter 2

Basic Concepts

"Mathematics maybe defined as the subject in which we never know what we are talking about, not whether what we are saying is true"

[Bertrand Russell]

"Contrariwise", continued Tweedledee, "if it was so, it might be; and if it were so, it would be; but as it isn't, it ain't. That's logic."

["Through the Looking Glass", Lewis Carroll]

In this chapter we introduce the logical framework and the notations used throughout the thesis. The classical notations of a logical language, a term, a formula, an interpretation and satisfiability are defined.

The chosen framework is first-order predicate logic (FOL). The reason to choose FOL is a tradeoff between expressive power of FOL and possibilities for automated reasoning. Many algorithms can be studied within a first-order framework. Automated theorem provers for FOL are increasingly being used in formal verification. For these applications, an efficient treatment of the equality relation is particularly important. Due to the special properties of the equality relation, proof search is hard for problems containing equality. We restrict our discussion to decidable subsets of FOL with equality. These subsets are powerful enough to specify many proof problems directly.

2.1 Syntax

In this section we introduce the logical framework and notations used throughout the thesis. The logical framework used in this thesis is a quantifier-free FOL with equality.

2.1.1 Terms

A *signature* is a tuple $\Sigma = (\mathsf{Fun}, \mathsf{Pr})$, where $\mathsf{Fun} = \{f, g, h, \dots\}$ is an enumerable set of *function symbols* and $\mathsf{Pr} = \{p, q, r \dots\}$ is a set of *predicate symbols.* The sets Fun and Pr are pairwise disjoint. For every function symbol and every predicate symbol its *arity* is defined, being a non-negative integer. The functions of arity zero are called *constants* , the predicates of arity zero are called *propositional variables.* We use $\mathsf{Const}(\Sigma)$, or for simplicity Const, to denote the set of constants.

Terms are the most important building blocks of formulae. They typically represent objects from the domain we want to reason about.

Definition 2.1 (**Terms, subterms**)

- *We use the lower case letters s, t, u to denote terms.*
- *Given a signature $\Sigma = (\mathsf{Fun}, \mathsf{Pr})$, the set $\mathsf{Term}(\Sigma)$ of* terms, *or for simplicity* Term, *over Σ is defined inductively: for $n \geq 0$, $f(t_1, \dots, t_n)$ is a term if $t_1, \dots, t_n$ are terms, $f \in \mathsf{Fun}$, and $\mathsf{ar}(f) = n$.*
- *The set $\mathsf{SubTerm}(t)$ of* subterms *of a term t is defined inductively: for each $f(t_1, \dots, t_n) \in \mathsf{Term}$, $n \geq 0$,*

$$\mathsf{SubTerm}(f(t_1, \dots, t_n)) = \{f(t_1, \dots, t_n)\} \cup \{\mathsf{SubTerm}(t_i) | 1 \leq i \leq n\}.$$

- *A subterm of a term t is called* proper *if it is distinct from t. The set of proper subterms of a term t is denoted by* $\mathsf{SubTerm}_p(t)$.
- *Let T be a set of terms. We define* $\mathsf{SubTerm}(T) = \bigcup_{t \in T} \mathsf{SubTerm}(t)$.

Note that the definition of terms can be given more concisely in 'abstract syntax' style as follows.

$\mathsf{Term} ::= \mathsf{Fun}(\mathsf{Term}, \ldots, \mathsf{Term})$.

Of course, we assume that the arity of the function symbol must be respected.

Definition 2.2 (**Depth of a term**)

- *The* depth *of a term t is denoted by* $\mathsf{depth}(t)$ *and inductively defined as follows.*
  - *For each $a \in \mathsf{Const}$, $\mathsf{depth}(a) = 1$.*
  - *For each term $f(t_1, \ldots, t_n)$,*

$$\mathsf{depth}(f(t_1, \ldots, t_n)) = 1 + \max(\mathsf{depth}(t_1), \ldots, \mathsf{depth}(t_n)).$$

2.1.2 Formulae and CNFs

While terms model domain objects and functions over them, atomic formulae (atoms) represent relations over objects, and formulae correspond to the specification of a proof problem. In the following we will define atoms and formulae the usual way, i.e. first order formulae consist of first order atoms connected by the usual propositional connectives.

Definition 2.3 (**Atomic formulae, Formulae**)

- *The set of atomic formulae* $\mathsf{At}(\Sigma)$ *, or for simplicity* At*, is defined as follows.*

  $\mathsf{At}(\Sigma) ::= \mathrm{Pr}(\mathsf{Term}, \ldots, \mathsf{Term})$,

  where the arity of the predicate symbol must be respected.
- *The set of formulae* $\mathsf{For}(\Sigma)$ *, or* For*, is defined as below.*

  $\mathsf{For}(\Sigma) ::= \mathsf{true} \mid \mathsf{false} \mid \mathsf{At} \mid \neg\mathsf{For} \mid \mathsf{For} \wedge \mathsf{For}$
- *The letters φ, ψ range over* For.
- *The abbreviation $\varphi \vee \psi$ stands for $\neg(\neg\varphi \wedge \neg\psi)$, $\varphi \to \psi$ stands for $\neg\varphi \vee \psi$, $\varphi \leftrightarrow \psi$ stands for $(\varphi \to \psi) \wedge (\psi \to \varphi)$, and $\mathsf{ITE}(\varphi, \psi, \chi)$ stands for $\neg(\neg(\varphi \wedge \psi) \vee (\neg(\varphi \wedge \chi))$.*

- *Given a formula φ,* $\mathsf{Const}(\varphi)$ *is the set of all constants occurring in φ (similar for terms, literals and clauses);* $\mathsf{Pr}(\varphi)$ *is the set of predicate symbols occurring in φ;* $\mathsf{At}(\varphi)$ *is the set of all atoms occurring in φ;*

Among all formulae, formulae in *conjunctive normal form* (CNF) are of special interest for us since many proof methods take CNFs as input. In this case a preliminary step is to translate general formulae into CNFs. The well-known Tseitin transformation [Tseitin, 1968] transforms an arbitrary formula to a CNF in such a way that the original formula is satisfiable if and only if the CNF is satisfiable.

Definition 2.4 (**Literals, Clauses, CNFs**)

- *The sets of literals* (Lit)*, clauses* (Cls) *and formulae in conjunctive normal form* (Cnf) *are defined below.*

  $\mathsf{Lit} ::= \mathsf{At} \mid \neg\mathsf{At}$

  $\mathsf{Cls} ::= \mathsf{Lit} \mid \mathsf{Cls} \vee \mathsf{Cls}$

  $\mathsf{Cnf} ::= \mathsf{Cls} \mid \mathsf{Cnf} \wedge \mathsf{Cnf}$

- *We will use the following convention: the letters l, m range over* Lit*; C, D range over* Cls*; φ, ψ range over* Cnf.

- *The set of positive literals* (Lit_p)*, the set of negative literals* (Lit_n) *and the set of purely positive clauses* (Cls_p) *are defined below.*

  $\mathsf{Lit}_p ::= \mathsf{At}$

  $\mathsf{Lit}_n ::= \neg\mathsf{At}$

  $\mathsf{Cls}_p ::= \mathsf{Lit}_p \mid \mathsf{Cls}_p \vee \mathsf{Cls}_p$

- $\mathsf{Lit}(\varphi)$, $\mathsf{Lit}_p(\varphi)$, $\mathsf{Lit}_n(\varphi)$ *are respectively the set of all literals, the set of all positive literals and the set of all negative literals which occur in φ.*

- $\mathsf{Cls}(\varphi)$ *and* $\mathsf{Cls}_p(\varphi)$ *are respectively the set of all clauses and the set of all pure positive clauses in φ.*

- *C is a unit clause if $|C| = 1$.*

- *We use $\perp$ to denote the empty clause.*

When working with CNFs we are neither concerned with the ordering of the clauses in a CNF nor with the ordering of the literals in a clause. Hence, all information we need can be presented as sets. Therefore, when it is convenient we will use set notations to represent CNFs.

We use the following notation throughout the thesis:

$\varphi|_l = \{C - \{\neg l\} \mid C \in \varphi,\ l \notin C\}$.

Example 2.5 Consider $\varphi \equiv \{\{r, q\}, \{\neg r, p\}\}$. Then $\varphi|_r \equiv \{\{p\}\}$.

2.1.3 Equalities

The equality relation plays an important role in modelling of many mathematical and verification problems. Statements, properties, descriptions are very often written in an equational form. For example, about half of the benchmarks in the TPTP problem library [Sutcliffe et al., 1994] contain equations. Therefore, handling of the equality relation is an important problem in automated theorem proving. Equality relations over terms are typically described by sets of *equalities*.

In general, the equality relation is symmetric, i.e. if some element x is equal to some element y then y is equal x. However, in this thesis we will distinguish between 'symmetrical' equalities and 'oriented' ones.

Definition 2.6 (**Equalities**)

- *An* equality e *is a pair* $(s, t) \in (\mathsf{Term} \times \mathsf{Term})$. *We write an equality as* $s \approx t$ *for the reserved predicate symbol* $\approx$. *We assume equalities to be symmetric, i.e. we identify* $s \approx t$ *and* $t \approx s$ *as the same equality.*
- *The set of equalities over* Σ *is defined by* $\mathsf{Eq}(\Sigma)$ *or if* Σ *is not relevant by* Eq.
- *Given an equality* $s \approx t$, *we define the set of terms occurring in* $s \approx t$ *as*
$$\mathsf{Term}(s \approx t) = \{s, t\}.$$
- *The set of subterms occurring in* $s \approx t$ *is defined as*
$$\mathsf{SubTerm}(s \approx t) = \mathsf{SubTerm}(s) \cup \mathsf{SubTerm}(t).$$

In Chapters 5, 8 we are not concerned with the orientation of an equality. Therefore, we consider $a \approx b$ and $b \approx a$ as the same equality. Nevertheless, $a \approx b$ and $b \approx a$ are syntactically different, and this is crucial in Chapter 9 when we define reduced ordered binary decision diagram for the equality logic with uninterpreted functions. To emphasize that equality is not symmetric we use a different symbol, i.e. when we write $a \simeq b$ it means that it differs from $b \simeq a$.

Definition 2.7 (**Oriented equalities**)

- *An* oriented equality e *is an oriented pair* $(s, t) \in (\mathsf{Term} \times \mathsf{Term})$. *We write an oriented equality as* $s \simeq t$ *for the predicate symbol* $\simeq$.
- *Given an equality* $s \simeq t$, *we define the set of terms occurring in* $s \approx t$ *as*
$$\mathsf{Term}(s \simeq t) = \{s, t\}.$$

- *The set of subterms occurring in $s \simeq t$ is defined as*

$$\mathsf{SubTerm}(s \simeq t) = \mathsf{SubTerm}(s) \cup \mathsf{SubTerm}(t).$$

In the following we will use the shortcut $x \not\approx y$ for $\neg(x \approx y)$, and we will use the shortcut $x \not\simeq y$ for $\neg(x \simeq y)$.

2.2 Semantics

Above we have described the formal language of first-order predicate logic. Now we discuss how such formulae can be interpreted. An interpretation consists of a *domain* and a mapping relative to $\mathcal{D}$ which assigns a value from the domain to every formula and every component of the formula. For first-order predicate formulae, an interpretation assigns values true and false to predicate symbols and values from the domain to function symbols.

A *structure* $\mathcal{D}$ over a signature $\Sigma = (\mathsf{Fun}, \mathsf{Pr})$ is defined to consist of

- a non-empty set D called the *domain*,
- for every $f \in \mathsf{Fun}$, $\mathsf{ar}(f) = n$, a map $f_D : D^n \to D$,
- for every $p \in \mathsf{Pr}$, $\mathsf{ar}(p) = n$, a map $p_D : D^n \to \{\mathsf{true}, \mathsf{false}\}$.

The *interpretation* $[\![t]\!]_{\mathcal{D}} : \mathsf{Term}(\Sigma) \to D$ of a term t is inductively defined as follows.

For $n \geq 0$, $[\![f(t_1, \ldots, t_n)]\!]_{\mathcal{D}} = f_D([\![t_1]\!]_{\mathcal{D}}, \ldots, [\![t_n]\!]_{\mathcal{D}})$, where $t_1, \ldots, t_n \in \mathsf{Term}$.

The interpretation $[\![\varphi]\!]_{\mathcal{D}} : \mathsf{For}(\Sigma) \to \{\mathsf{true}, \mathsf{false}\}$ of a formula φ is defined as follows.

- $[\![\mathsf{true}]\!]_{\mathcal{D}} = \mathsf{true}$
- $[\![\mathsf{false}]\!]_{\mathcal{D}} = \mathsf{false}$
- $[\![s \approx t]\!]_{\mathcal{D}} = \begin{cases} \mathsf{true} & \text{if } [\![s]\!]_{\mathcal{D}} = [\![t]\!]_{\mathcal{D}} \\ \mathsf{false} & \text{otherwise} \end{cases}$
- $[\![s \simeq t]\!]_{\mathcal{D}} = [\![s \approx t]\!]_{\mathcal{D}}$
- $[\![\neg\varphi]\!]_{\mathcal{D}} = \begin{cases} \mathsf{true} & \text{if } [\![\varphi]\!]_{\mathcal{D}} = \mathsf{false} \\ \mathsf{false} & \text{otherwise} \end{cases}$

- $[\![\varphi \wedge \psi]\!]_{\mathcal{D}} = \begin{cases} \mathsf{true} & \text{if } [\![\varphi]\!]_{\mathcal{D}} = \mathsf{true} \text{ and } [\![\psi]\!]_{\mathcal{D}} = \mathsf{true} \\ \mathsf{false} & \text{otherwise} \end{cases}$

  $[\![\mathsf{ITE}(\varphi, \psi, \chi)]\!]_{\mathcal{D}} = \begin{cases} [\![\psi]\!]_{\mathcal{D}} & \text{if } [\![\varphi]\!]_{\mathcal{D}} = \mathsf{true} \\ [\![\chi]\!]_{\mathcal{D}} & \text{otherwise} \end{cases}$

Note that ITE is only used in chapter 9.

Definition 2.8 *We say that a formula φ is* logically equivalent *to a formula ψ if for every structure $\mathcal{D}$, $[\![\varphi]\!]_{\mathcal{D}} = [\![\psi]\!]_{\mathcal{D}}$.*

2.3 Satisfiability

We define satisfiability for instances of first order predicate logic. In some instances of our framework for defining satisfiability all possible structures are allowed, in others we have restrictions on the structures that are allowed. Therefore in any instance we assume a notion of *admissible structure*. Depending on this notion of admissible structure we have the following definition of satisfiability.

Definition 2.9 (**Satisfiability**)

- *A structure $\mathcal{D}$* satisfies *a formula φ if $[\![\varphi]\!]_{\mathcal{D}} = \mathsf{true}$. φ is called* satisfiable *if it is satisfied in some admissible structure. Otherwise φ is called* unsatisfiable*.*
- *If each structure $\mathcal{D}$ satisfies φ then φ is a* tautology *(or a valid formula).*
- *If no structure $\mathcal{D}$ satisfies φ then φ is a* contradiction*.*

A particular logic consists of a signature and a set of admissible structures. By the latter, we can distinguish a completely uninterpreted setting (no restriction on structures) from a completely interpreted setting (only one structure is admissible). However, intermediate situations are possible as well.

2.4 Instances

In this section we precisely describe the different instances of the framework just described by specifying the signature and the admissible structures.

Since we use the symbol $\approx$ for reasoning over equality, in admissible structures we require for instances determined in this section that $\approx_D = Id_D$, where the function $Id_D : D \times D \to \{\mathsf{true}, \mathsf{false}\}$ is defined as follows.

$$Id_D(d_1, d_2) = \begin{cases} \mathsf{true} & \text{if } d_1 = d_2, \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

2.4.1 Propositional Logic

The first instance we consider is propositional logic. Here we have $\Sigma = \{\mathsf{Fun}, \mathsf{Pr}\}$, where $\mathsf{Fun} = \emptyset$ and Pr is a set of predicate symbols all having arity zero. In this way there are no terms at all occurring in atoms: an atom coincides with such a predicate symbol of arity zero. Hence, a CNF in this instance coincides with a usual propositional CNF. Since there are no terms in the formula, neither function symbols play a role, nor their interpretations. The only remaining ingredient of an interpretation is a map $p_D : D^0 \to \{\mathsf{true}, \mathsf{false}\}$ for every predicate symbol p. Since D^0 consists of one element independent of D, this interpretation is only a map from the atoms to $\{\mathsf{true}, \mathsf{false}\}$. Since the domain does not play a role there is no need for defining restrictions: as admissible structures we allow all structures.

2.4.2 Equality Logic

The next instance we consider is equality logic. By equality logic formulae we mean formulae built from atoms of the shape $a \approx b$, where a and b are constants, and the usual propositional connectives. Now we define equality formulae in CNF as an instance of the syntax described above.

For equality logic we have $\Sigma = \{\mathsf{Fun}, \mathsf{Pr}\}$, where Fun is a set of function symbols of arity zero, and $\mathsf{Pr} = \{\approx\}$. In this way the constants are the only terms, and all atoms are of the shape $a \approx b$ for constants a and b.

The admissible structures are defined to be all structures $\mathcal{D}$ for which $\approx_D = Id_D$.

Example 2.10 We consider

$$\varphi = a \approx b \wedge b \approx c \wedge a \not\approx c.$$

Assume φ is satisfiable. Then there is an admissible structure $\mathcal{D}$ such that $[\![\varphi]\!]_{\mathcal{D}} = \mathsf{true}$. Hence, we have

- $[\![a \approx b]\!]_{\mathcal{D}} = Id_D([\![a]\!]_{\mathcal{D}}, [\![b]\!]_{\mathcal{D}}) = \mathsf{true}$. Hence, $[\![a]\!]_{\mathcal{D}} = [\![b]\!]_{\mathcal{D}}$, and
- $[\![b \approx c]\!]_{\mathcal{D}} = Id_D([\![b]\!]_{\mathcal{D}}, [\![c]\!]_{\mathcal{D}}) = \mathsf{true}$. Hence, $[\![b]\!]_{\mathcal{D}} = [\![c]\!]_{\mathcal{D}}$, and
- $[\![a \approx c]\!]_{\mathcal{D}} = Id_D([\![a]\!]_{\mathcal{D}}, [\![c]\!]_{\mathcal{D}}) = \mathsf{false}$. Hence, $[\![a]\!]_{\mathcal{D}} \neq [\![c]\!]_{\mathcal{D}}$, contradiction.

Therefore, we proved that φ is unsatisfiable.

Roughly speaking an equality logic CNF is unsatisfiable if and only if a contradiction can be derived using the CNF itself and reflexivity, symmetry and transitivity of equality, see [Zantema and Groote, 2003]. In Chapter 4 we introduce a resolution-based procedure for Equality Logic.

In this basic version of equality logic there are no function symbols. The following instance is Equality Logic with Uninterpreted Functions.

2.4.3 Equality Logic with Uninterpreted Functions

An equality logic formula with uninterpreted functions is a boolean formula over atoms that are equalities between ground terms. For EUF-logic we have $\Sigma = (\mathsf{Fun}, \mathsf{Pr})$, where Fun is an arbitrary set of function symbols and Pr consists of the binary predicate symbol $\approx$. There is no restriction on the interpretation of the function symbols by which they are called uninterpreted.

As for Equality Logic the admissible structures are defined to be all structures $\mathcal{D}$ for which $\approx_D = Id_D$.

Example 2.11 We consider the following formula:

$$\varphi = (a \approx b \wedge b \approx c) \rightarrow f(a) \approx f(c).$$

For every admissible structure such that $[\![a \approx b \wedge b \approx c]\!]_{\mathcal{D}} = \mathsf{false}$, $[\![\varphi]\!]_{\mathcal{D}} = \mathsf{true}$.

Let us consider the case when $[\![a \approx b \wedge b \approx c]\!]_{\mathcal{D}} = \mathsf{true}$. Hence, we have

- $[\![a \approx b]\!]_{\mathcal{D}} = \mathsf{true}$. Hence, $[\![a]\!]_{\mathcal{D}} = [\![b]\!]_{\mathcal{D}}$.
- $[\![b \approx c]\!]_{\mathcal{D}} = \mathsf{true}$. Hence, $[\![b]\!]_{\mathcal{D}} = [\![c]\!]_{\mathcal{D}}$.

We conclude $[\![a]\!]_{\mathcal{D}} = [\![b]\!]_{\mathcal{D}} = [\![c]\!]_{\mathcal{D}}$. Therefore,

$$[\![f(a)]\!]_{\mathcal{D}} = f_D([\![a]\!]_{\mathcal{D}}) = f_D([\![c]\!]_{\mathcal{D}}) = [\![f(c)]\!]_{\mathcal{D}}.$$

Hence, $[\![\varphi]\!]_{\mathcal{D}} = \mathsf{true}$.

In Chapters 8 and 9 we discuss the way to deal with uninterpreted function symbols.

2.5 Term replacing

An equality relation allows to replace terms with terms without changing the meaning of the surrounding formula. Therefore one of the major operations in reasoning with equality is the replacement of subterms by *rewriting*, i.e. if we know $s \approx t$ we can replace any s with t in a true proposition and obtain a true proposition.

Definition 2.12 (**Replacement**)

- *A* replacement rule *is a pair (s,t), where $s,t \in \mathsf{Term}$.*
- *A* one-step replacement *obtained by (s,t) is a replacement of the form*

$$r[s] \rightarrow_{(s,t)} r[t],$$

where r is a term, and one occurrence of s has been replaced by t.

We will use terminology from *term rewrite systems* (TRS) [Baader and Nipkow, 1998]. In particular, by a *normal form* with respect to some TRS we mean a term to which no rules of the TRS are applicable. A system is *terminating* if no infinite rewrite sequence exists.

Example 2.13 *Consider a term $f(x)$. Then x is a normal form of $f(x)$ with respect to $\rightarrow_{(f(x),x)}$, and $f(x)$ has no normal form with respect to $\rightarrow_{(x,f(x))}$.*

Lemma 2.14 *Let $s,t,u \in \mathsf{Term}$ and $s \notin \mathsf{SubTerm}(t)$. Then u has a normal form with respect to $\rightarrow_{(s,t)}$.*

Proof Consider the term u. If $s \notin \mathsf{SubTerm}(u)$ then u is a normal form of u with respect to $\rightarrow_{(s,t)}$. Suppose $s \in \mathsf{SubTerm}(u)$. For this case we will give a proof by induction on the structure of u.

Base case. Suppose $u \equiv s$. Then $u \equiv s \rightarrow_{(s,t)} t$. Since $s \notin \mathsf{SubTerm}(t)$, t is a normal form of u with respect to $\rightarrow_{(s,t)}$.

Inductive step. Suppose $t_1, \dots, t_n$ are normal forms of $s_1, \dots, s_n$ with respect to $\rightarrow_{(s,t)}$. Suppose

$$u \equiv f(s_1, \dots, s_n).$$

Clearly, $u \rightarrow^*_{(s,t)} f(t_1, \dots, t_n)$. One of the following holds.

- $f(t_1, \dots, t_n) \equiv s$. Then $f(t_1, \dots, t_n) \rightarrow_{(s,t)} t$.

  We obtain that t is a normal form of u with respect to $\rightarrow_{(s,t)}$.

- $f(t_1, \dots, t_n) \not\equiv s$. Taking into account that $s \notin \mathsf{SubTerm}(t_i)$, $1 \leq i \leq n$, we conclude that $s \notin \mathsf{SubTerm}(f(t_1, \dots, t_n))$. Hence, $f(t_1, \dots, t_n)$ is a normal form of u with respect to $\rightarrow_{(s,t)}$.

We have shown the existence of a normal form with respect to $\rightarrow_{(s,t)}$. ⊠

Definition 2.15 (*Term replacing*) *Let $s, t \in \mathsf{Term}$, where $s \notin \mathsf{SubTerm}(t)$.*

- *Suppose $u \in \mathsf{Term}$. By $u[s := t]$ we mean the normal form of u with respect to $\rightarrow_{(s,t)}$.*
- *Suppose $\varphi \in \mathsf{Cnf}$. By $\varphi[s := t]$ we mean the normal form of φ obtained after applying the following rule. For every $u \in \mathsf{SubTerm}(\varphi)$, $u \rightarrow u[s := t]$.*
- *Suppose T is a set of terms. By $T[s := t]$ we mean the normal form of T obtained after applying the following rule. For every $u \in \mathsf{SubTerm}(T)$, $u \rightarrow u[s := t]$.*

Given $s, t, u \in \mathsf{Term}$ and $s \notin \mathsf{SubTerm}(t)$, in Lemma 2.14 we have shown that u has a normal form with respect to $\rightarrow_{(s,t)}$. Since for us only the existence of a normal form is important but not its uniqueness, we do not give a proof of confluence of the replacement. The reader can easily check that it is confluent. From the existence of a normal form and confluence we can conclude that it is unique.

Corollary 2.16 *Every $\varphi \in \mathsf{Cnf}$ has a unique normal form with respect to $\rightarrow_{(s,t)}$, where $s, t \in \mathsf{Term}$ and $s \notin \mathsf{SubTerm}(t)$.*

Proof An immediate consequence of Lemma 2.14 and confluence. ⊠

Example 2.17 *Consider the CNF $\varphi \equiv \{\{f(f(x)) \approx y\}, \{x \approx g(y)\}\}$. Then*

$$\varphi[f(x) := x] \equiv \{\{x \approx y\}, \{x \approx g(y)\}\}.$$

2.6 Inference rules

We are interested in checking the satisfiability of a logical formula. Purely syntactical reasoning is often sufficient to determine validity or satisfiability of a formula. Therefore, we discuss proof systems based on syntactical reasoning, and by a proof system we mean a set of inference rules, describing transformations of CNFs and conditions which guarantee that the derivation process will derive an empty clause from an unsatisfiable CNF.

If statements $Prem_1, \ldots, Prem_n$ together imply Con, then one can infer a statement Con from the set of statements $\{Prem_1, \ldots, Prem_n\}$. The inferred statement Con is called the *conclusion* of the inference. $Prem_1, \ldots, Prem_n$ are called *premisses* of the inference.

Note that sometimes we may be interested in inferences when we would like to infer more than one statement from premisses.

In the following we will consider two kinds of inference rules: *generating* rules and *replacing* rules. As in [Schulz, 2000], we use a double line for replacing rules.

Definition 2.18 (**Inference rules**)

- *A* generating inference rule *is a deduction scheme of the form*

  $$\frac{< premisses >}{< conclusion >} \qquad \text{if } < condition >$$

  It means that from the formulae written as the <premisses> we can conclude the formulae written as the <conclusion> if the <condition> is met.

- *A* replacing inference rule *is a deduction scheme of the form*

  $$\genfrac{}{}{}{}{< premisses >}{\overline{< conclusion >}} \qquad \text{if } < condition >$$

  It means that the set of formulae written as the <premisses> we may replace by the set of formulae written as the <conclusion> if the <condition> is met.

- *A* replacing rule with two conclusions *is a deduction scheme of the form*

  $$\genfrac{}{}{}{}{< premisses >}{\overline{< conclusion1 > \mid < conclusion2 >}} \qquad \text{if } < condition >$$

  Such a scheme is to be understood as indicating that we may replace the set of formulae written as the <premisses> with two sats of formulae written as <conclusion1> and <conclusion2> if the <condition> is met.

We are now ready to define what we mean by the validity of an inference rule.

As we mentioned in the introduction, purely syntactical reasoning is often sufficient to determine the validity or satisfiability of a formula. Therefore, we are interested in inference rules which preserve (un)satisfiability of the formula.

Definition 2.19 (**Valid inference rule**)

- *We say that an inference rule with one conclusion is valid if every interpretation which satisfies the set of formulae written as the <premisses> also satisfies the set of formulae written as the <conclusion>.*

- *We say that an inference rule with two conclusions is valid if every interpretation satisfying the set of formulae written as the < premisses > also satisfies at least one of the sets of formulae written as the <conclusion1> and the <conclusion2>.*

In this thesis we use a generating inference rule to describe the resolution rule in Chapter 3 and an extension of this rule for the equality logic in Chapter 5. Replacing inference rules with one and two conclusions are used to describe a proof system for the equality logic with uninterpreted functions in Chapter 8.

Definition 2.20 (**Proof system**)

- *A proof system is a set of inference rules.*
- *We say that a proof system is sound if it contains only valid inference rules*
- *We say that a proof system is complete if one can derive all true statements using the rules of the proof system.*

In the following we will speak about refutational proof systems. It means that the formula is unsatisfiable if and only if the empty clause can be derived by the rules of the proof system.

Chapter 3

Basic decision procedures for Propositional Logic

"Logic is the art of convincing us of some truth"

[Jean de La Bruyère]

"If any one of them can explain it," said Alice, "I'll give him sixpence. I don't believe there's an atom of meaning in it."

["Alice's Adventures in Wonderland", Lewis Carroll]

The SAT problem, the problem of deciding whether a given propositional formula is satisfiable, is one of the well-known NP-complete problems (Cook's theorem). It is a core problem in mathematical logic and computing theory. In practice, SAT is fundamental in solving many problems in automated reasoning, computer-aided design, integrated circuit design, computer architecture design, and computer network design.

Three basic methods have been developed for satisfiability testing: refutation search, model search, and local search.

- Refutation search seeks to discover a proof that a formula is unsatisfiable, usually employing resolution. If a complete search for a refutation fails, the formula is reported to be satisfiable.
- Model search seeks to discover a satisfying assignment, or model, for the formula. If a complete search for a model fails, the formula is reported to be unsatisfiable. The DPLL algorithm is the basis for many modern refinements. This basic algorithm consists of the unit clause and pure literal rules for simplification, and the splitting rule for searching. The splitting rule specifies that the splitting variable be chosen from a shortest clause.
- Several methods employ various local search heuristics to perform incomplete model searches. They are more properly characterized as max-sat methods. They cannot even report a formula to be unsatisfiable. They can only return "don't know" and give up based on resource limits, when they fail to discover a model. However, they have succeeded in finding models on much larger formulas than current complete methods can handle.

The most powerful complete SAT solvers, which have recently been proposed, are mostly based on resolution (the original DP and DPLL algorithms) and BDDs.

3.1 The DP procedure

The Davis-Putnam procedure (DP) [Davis and Putnam, 1960] was introduced in 1960 as a proof procedure for propositional logic. It is in essence a *resolution* procedure.

3.1.1 Resolution

Resolution is a technique invented by Robinson [Robinson, 1965] in the 1960's. It uses purely "syntactic" manipulation to determine the validity of a formula: for a given set of clauses, the resolution rule derives a new clause which is a logical consequence of two clauses from the set. In the propositional case we define the resolution inference rule by "cutting away" a pair of complementary literals in two clauses which are resolved upon.

In the following we use the notation $\uplus$ for *disjoint union*, i.e. we write $S_1 \uplus S_2$ when we are referring to the union $S_1 \cup S_2$ and $S_1 \cap S_2 = \emptyset$.

Definition 3.1 (**Resolution**) *The resolution rule is an inference rule of the following shape:*

$$\frac{\{l\} \uplus C \quad \{\neg l\} \uplus D}{C \cup D}$$

where $C, D \in \mathsf{Cls}$ *and* $l \in \mathsf{Lit}$.

In the following we use the well-known fact that a CNF ϕ is unsatisfiable if and only if the empty clause can be derived from φ by the resolution rule.

The inference rule *unit resolution* is resolution, such that at least one clause in the premisses is a unit clause.

Definition 3.2 (**Unit resolution**) *Unit resolution is an inference rule of the following shape*

$$\frac{\{l\} \uplus C \quad \{\neg l\}}{C}$$

where $C \in \mathsf{Cls}$ *and* $l \in \mathsf{Lit}$.

Theorem 3.3 *Resolution is sound and complete.*

Proof See [Robinson, 1965]. $\boxtimes$

In the following we will use the fact that resolution is a refutation system, i.e. a formula is unsatisfiable if and only if the empty clause can be derived from it by resolution. Resolution is the most widely studied approach to propositional theorem proving and to satisfiability checking. Many satisfiability algorithms are based on resolution. Some of the most widely used in propositional theorem proving are the DP and the DPLL procedures. The general idea of the procedures is to reduce a problem by eliminating one variable: the DP procedure does it by all possible resolution steps on a chosen variable, and the DPLL procedure branches on possible truth assignment to a chosen variable. In Chapter 4 we show that a DPLL-based refutation can be transformed to a resolution-based refutation of a similar length.

```
$DP(\varphi) : \{\mathsf{SAT}, \mathsf{UNSAT}\} =$
    begin
        if $(\bot \in \varphi)$ then return $\mathsf{UNSAT}$;
        if $(\varphi = \emptyset)$ then return $\mathsf{SAT}$;
        $p := \mathsf{ChooseLiteral}(\mathsf{At}(\varphi))$;
        $\varphi := \mathsf{Resolve}(\varphi, p)$;
        $\varphi := \mathsf{Reduce}(\varphi, p)$;
        $DP(\varphi)$;
    end;
```

Figure 3.1 The DP procedure

3.1.2 The original DP procedure

Given a CNF φ, the DP procedure repeats the following steps until either the empty clause is derived or no propositional variables are left.

- Choose a propositional variable $p \in \mathsf{At}(\varphi)$.
- Add to φ all possible resolvents using resolution over p.
- Throw out all clauses with both p and $\neg p$ in them.
- If the empty clause is derived then return SAT.
- If no clause is left then return UNSAT.

The algorithm is depicted in Figure 3.1.

The DP procedure includes the following functions.

- The function $\mathsf{ChooseLiteral}(\varphi)$ chooses a propositional variable appearing in one of the clauses.
- The function $\mathsf{Resolve}(\varphi, p)$ adds all possible resolvents to the CNF using resolution over the chosen propositional variable.
- The function $\mathsf{Reduce}(\varphi, p)$
  - throws out all clauses containing the chosen propositional variable p or a complimentary propositional variable $\neg p$;
  - removes all clauses containing both l and $\neg l$ in them.

Theorem 3.4 *The DP procedure is sound and complete.*

Unfortunately, in practice the original DP procedure is not efficient. If variables have many occurrences then many resolution steps should be done. Since the number of clauses increases drastically and clauses become longer and longer, in real world examples CNFs explode.

3.2 The DPLL procedure

The most powerful complete algorithms for solving SAT are based on the Davis-Putnam-Loveland procedure (DPLL) [Davis et al., 1962]. In DPLL, resolution steps are replaced by unit resolution in combination with backtracking search [Yugami, 1995].

Given a propositional formula in CNF, the DPLL procedure allows to check whether this formula is satisfiable or not. The basic DPLL procedure recursively implements the three rules: unit propagation, pure literal elimination and recursive splitting.

- *Unit propagation*: Given a unit clause $\{l\}$, remove all clauses that contain the literal l, including the clause itself and delete all occurrences of the literal $\neg l$ from all other clauses. Moreover, the literal l is assigned to true.
- *Pure literals elimination*: A *pure literal* is a literal that appears only in positive or only in negative form. Eliminate all clauses containing pure literals. Moreover, each pure literal is assigned to true.
- *Splitting*: Choose some literal $l \in \mathsf{Lit}(\varphi)$. Now φ is unsatisfiable if and only if both $\varphi \cup \{\{l\}\}$ and $\varphi \cup \{\{\neg l\}\}$ are unsatisfiable. Selecting a literal for splitting is nondeterministic, so various heuristics can be employed.

The algorithm starts with a set of clauses and simplifies the set of clauses simultaneously. The stopping cases are:

- An empty set of clauses is satisfiable–in this case, the entire algorithm terminates with "satisfiable".
- A set containing an empty clause is not satisfiable–in this case, the algorithm backtracks and tries a different value for an instantiated variable.

We would like to emphasize that the DPLL procedure stops with the answer 'satisfiable' if the empty set of clauses is derived for at least one branch, and it returns 'unsatisfiable' if the empty clause is derived for all branches. It can be easily extended to return a satisfying assignment.

The algorithm is depicted in Figure 3.2.

The DPLL procedure includes the following functions.

```
$DPLL(\varphi) : \{\mathsf{SAT}, \mathsf{UNSAT}\} =$
    begin
        $\varphi := \mathsf{Reduce}(\varphi)$;
        if $(\bot \in \varphi)$ then return $\mathsf{UNSAT}$;
        if $(\varphi = \emptyset)$ then return $\mathsf{SAT}$;
        $l := \mathsf{ChooseLiteral}(\mathsf{Lit}(\varphi))$;
        if $DPLL(\varphi \cup \{\{l\}\}) = \mathsf{SAT}$ then return $\mathsf{SAT}$;
        if $DPLL(\varphi \cup \{\{\neg l\}\}) = \mathsf{SAT}$ then return $\mathsf{SAT}$;
        return $\mathsf{UNSAT}$;
    end;
```

Figure 3.2 The DPLL procedure

- The function Reduce() eliminates pure literals and simplifies the CNF by unit resolution steps.
- The function ChooseLiteral() chooses a literal appearing in one of the clauses.

Theorem 3.5 *The DPLL procedure is sound and complete.*

3.3 BDDs

Binary Decision Diagrams (BDDs) are one of the biggest breakthroughs in computer-aided design in the last decade. BDDs are a canonical and efficient way to represent and manipulate Boolean functions and have been successfully used in numerous applications. The basic idea has been known for more than 30 years [Akers, 1978].

Reduced ordered BDDs [Bryant, 1986, Brace et al., 1991, Bryant, 1992a,b] form a canonical representation, making testing of satisfiability and equivalence straightforward. Unfortunately, their power is mostly restricted to propositional logic, which is often not sufficiently expressive.

A BDD represents a boolean formula as a finite, binary, ordered, directed acyclic graph. The leaves of this graph are labelled with true and false, and all internal nodes are labelled with boolean variables. Each internal node l is labelled with a boolean variable $var(l)$ and has two arcs toward its children: $low(l)$ corresponding to the case where the variable is assigned to false, and $high(l)$ corresponding to the case where the variable is assigned to true. For a given assignment to the variables, the function value can be determined by following a path from the root to a leaf.

We can define the set of BDDs as a restricted subset of formulas:

$BDD ::= \mathsf{false} \mid \mathsf{true} \mid \mathsf{ITE}(\mathsf{At}, BDD, BDD)$.

Although in implementations they are always treated as shared *directed acyclic graphs* (DAGs). When we define binary decision diagrams for equality logic with uninterpreted functions in Chapter 9 we also define them as a subset of formulas. The only difference with ordinary BDDs is that guards consist of equalities between ground terms.

Given a fixed order on the variables, a BDD can be transformed to an *Ordered* BDD (OBDD), in which the variables along each path occur in increasing order. Although the variables ordering can be chosen arbitrarily, in practice, the choice of a good order is critical for the efficiency of manipulation.

A *Reduced* (O)BDD (R(O)BDD) is an (ordered) graph, in which redundant tests do not occur, and the graph is maximally shared. For a fixed order, the ROBDD representation of a formula is *canonical*: two formulas are equivalent if and only if they are represented by the same ROBDD. This property has important consequences: a formula is a tautology if its ROBDD representation corresponds to a node labelled with true, a formula is unsatisfiable if its ROBDD representation corresponds to a node labelled with false, and all other formulas are satisfiable.

3.4 Hybrid procedures for SAT

The complementary properties of resolution and the DPLL procedure suggest combining both in a hybrid scheme. Resolution helps detecting inconsistent subproblems, but at the same time resolution is costly.

There has been interest in using resolution in combination with DPLL search, to reduce the amount of search required to solve SAT problems [GEL95,RIS00]. Resolution can shorten the size of the search tree, but it is only useful if the time spent on resolution is less than the time gained by reducing the search space.

In [Rish and Dechter, 2000] Rish and Dechter use directional resolution (a form of ordered resolution) to search for SAT problems. Directional resolution identifies pairs of resolvable clauses quickly. They also showed that two hybrid algorithms combining resolution and search performed better than pure search.

Drake et al. in [Drake et al., 2002] present two techniques that combine resolution and DPLL search, and describe the relationship between them. The first technique, neighbor resolution, uses a restricted form of resolution during search, while the second uses a single level of binary resolution as a preprocessing step. It is shown that the preprocessing technique can cut the search tree as much as neighbor resolution during search.

It is proved that resolution-based approaches and BDDs are fundamentally different techniques [Groote and Zantema, 2001], i.e. there are examples which are easy for BDDs and exponentially hard for any resolution-based procedure, and vice versa, there are examples easy for resolution and exponentially hard for for BDDs.

In [Huang and Darwiche, 2004] the DPLL procedure is used to construct OBDD for propositional formulas. The experiments show that this method runs orders a magnitude faster than a comparable implementation of a traditional approach.

3.5 Beyond propositional satisfiability

The last decade has witnessed an increasing interest in propositional procedures for checking satisfiability and using them as basic inference engines of decision procedures for much more expressive problems, like reasoning in modal and description logics, resource planning and temporal reasoning. By integrating SAT solvers with domain-specific procedures, they can be efficiently extended to work with more expressive domains.

A first order logic formula consists of first order atoms and propositional connectives. Then the propositional structure is left to a SAT checker, and a specific decision procedure on the top of the propositional method checks the satisfiability for conjunctions of literals. The DPLL procedure, and BDDs are natural candidates to be used as kernels of decision procedures for more expressive logics.

In [Tinelli, 2002] a parametric calculus for proving the satisfiability of ground formulas in a logical theory $\mathcal{T}$, a sequent calculus based on the DPLL procedure, is presented.

The propositional DPLL procedure is a basis for many expressive theorem provers, e.g. MathSat [Audemard et al., 2002] and STeP [Bjørner et al., 1997]. DPLL checks satisfiability by looking for a satisfiable interpretation in a binary tree. Each node in the search corresponds to a partial interpretation. When an interpretation satisfying the propositional structure of a formula is found, the decision procedure for the conjunction of first order atoms either agrees with this interpretation as satisfying the first order formula or finds it unsatisfiable. This approach is called *lazy notification* [Barrett et al., 2002]. Another approach, called *eager notification* [Barrett et al., 2002, Flanagan et al., 2003], is also possible in combination with the DPLL procedure. In this approach the full model does not have to be found, i.e. a specific decision procedure notifies regularly if the partial model is unsatisfiable.

There were several attempts to extend propositional BDDs to richer logics. Methods translating logic formulas containing equalities and uninterpreted functions are proposed in [Bryant et al., 1999a, Bryant and Velev, 2002]. Goel et al. [Goel et al., 1998] proposed to decide equality logic formulae by replacing all equalities with new boolean variables. Subsequently, the BDD for the resulting formula is calculated without taking into account the transitivity of equalities.

In [Groote and Pol, 2000], equational BDDs (EQ-BDDs) are defined, in which all paths are satisfiable by construction. That approach extends the notion of orderedness to capture the properties of reflexivity, symmetry, transitivity, and substitutivity. The advantage of the method is that satisfiability checking for a given BDD can be done immediately. However, it is restricted to the case when equalities do not contain function symbols. Although, in practise it works well.

EQ-BDDs have been extended in [Badban and Pol, 2004b]. Here some *interpreted* functions, viz. natural numbers with zero and successor were added. In [Badban and Pol, 2004a] an alternative solution was provided, with a different orientation of the equations.

'Extended' logics such as predicate logic are more expressive and often cannot be expressed in propositional logic. In certain cases properties expressed in higher order logics can be formulated in propositional logic at the expense of a generally enormous growth of the formula. In both cases verification techniques for propositional logic are not very helpful for establishing the validity of higher order logics.

In order to have the advantages of effective propositional proof procedures available for higher logics the techniques must be lifted to a higher level. In this thesis we describe a way to lift the DP and the DPLL procedures and BDD techniques to decidable subsets of first order logic with equality. In general, each prover for first-order predicate logic can handle equalities. One can always add the standard congruence axioms for equality (reflexivity, symmetry, transitivity, substitution axioms). However, the naive approach mostly results in a huge search space to be explored. Therefore, it is important to have specific efficient techniques to handle equalities.

Chapter 4

Transforming DPLL to Resolution

“ . . . there are good reasons to believe that nonstandard analysis, in some version or other, will be the analysis of the future.”

[Kurt Gödel]

“Logic is simply the architecture of human reason”

[Evelyn Waugh]

In this chapter we give a formal description of resolution and analyze the relation between DPLL and resolution in detail. It is well-known that a DPLL refutation can be straightforwardly transformed into a resolution refutation of a similar length [Goldberg, 1979]. We give a transformation of a DPLL refutation into a resolution refutation of a number of steps which is essentially less than the number of unit resolution steps applied in the DPLL refutation: in particular, we prove that if in the DPLL procedure s unit resolution steps are executed and r recursive calls are done, a resolution refutation exists of length at most $s-r$, being essentially better than s. We prove for suitable formulae that no shorter resolution refutation exists than we give by our transformation, so our bounds are tight.

4.1 Basic definitions and preliminaries

This section contains notations and definitions used throughout the chapter. In this chapter for the sake of convenience we will rather speak about a resolution step than about the resolution rule.

Definition 4.1 (**Resolution step**)

- *Suppose*
  - φ_1, φ_2 *are CNFs;*
  - $C \uplus \{p\}$, $D \uplus \{\neg p\} \in \varphi_1$, *where* $p \in A$.
  - $\varphi_2 = \varphi_1 \cup \{C \cup D\}$

  Then the transition from φ_1 *to* φ_2 *is called a* resolution step, *and it is denoted as*

$$\varphi_1 \overset{CDp}{\longmapsto} \varphi_2.$$

  C, D, p *can be omitted in the context where they are not relevant.*

- *If* $C = \emptyset$ *or* $D = \emptyset$ *then the resolution step is called a* unit resolution step. *We use the notation*

$$\varphi_1 \overset{l}{\underset{u}{\longmapsto}} \varphi_2$$

  for the unit resolution step, where l *is either* p *if* $C = \emptyset$ *or* $\neg p$ *if* $D = \emptyset$. *This unit resolution step is called an* l-step.

Definition 4.2 (**Resolution sequence**)

- *If* $\varphi_0 \longmapsto \cdots \longmapsto \varphi_n$ *then* $\varphi_0, \dots, \varphi_n$ *is called a* resolution sequence *of* length n.
- *If* $\varphi_0 \overset{l_1}{\underset{u}{\longmapsto}} \dots \overset{l_n}{\underset{u}{\longmapsto}} V_n$ *then* $\varphi_0, \dots, \varphi_n$ *is called a* unit resolution sequence.

Suppose φ_0 is a CNF and C is a clause. We say that C is *derived* from φ_0 in $n > 0$ resolution steps if there is a resolution sequence $\varphi_0, \dots, \varphi_n$ such that $\varphi_n = \varphi_{n-1} \cup \{C\}$ and $C \notin \varphi_{n-1}$. And we say that C is derived from φ_0 in 0 resolution steps if $C \in \varphi_0$.

Definition 4.3 (**Resolution refutation**) *A resolution sequence* $\varphi_0, \dots, \varphi_n$ *such that* $\perp \in \varphi_n$ *is called a* resolution refutation *and* n *is called the* length of the resolution refutation.

In the following we use the well-known fact that a *CNF* φ is unsatisfiable if there is a resolution refutation starting from φ.

We introduce some notation.

Suppose $\varphi \in \mathsf{Cnf}$ and $l \in \mathsf{Lit}$. We define

$$\mathcal{V}(\varphi, l) = \{\psi \in \mathsf{Cnf} \mid \forall C \in \varphi\ \exists D \in \psi : (D \equiv C) \vee (D \equiv C \cup \{l\})\}.$$

At first we present the lemmas we use in the following.

The main observation of the subsequent lemmas is that if the empty clause can be derived from some CNF $\varphi \cup \{\{l\}\}$ in n resolution steps then under some conditions $\neg l$ can be derived from φ in a smaller number of steps.

Lemma 4.4 *Suppose for* $\varphi_0, \varphi_1, \psi_0 \in \mathsf{Cnf}$ *the following holds.*

1. $\varphi_0 \longmapsto \varphi_1$,
2. $\psi_0 \in \mathcal{V}(\varphi_0, l)$.

Then there is a $\psi_1 \in \mathcal{V}(\varphi_1, l)$ *such that*

$$\psi_0 \longmapsto \psi_1.$$

Proof From $\psi_0 \in \mathcal{V}(\varphi_0, l)$ it follows that for each $C \in \varphi_0$ there is a $D \in \psi_0$ such that

$$D = C \text{ or } D = C \cup \{l\}.$$

It follows from $\varphi_0 \longmapsto \varphi_1$ that

$$\varphi_1 \ \equiv \ \varphi_0 \cup \{C_1 \cup C_2\},$$

where $C_1 \cup \{p\}, C_2 \cup \{\neg p\} \in \varphi_0$ for some $C_1, C_2 \in \mathsf{Cls}$ and $p \in \mathsf{At}$.

Since $\psi_0 \in \mathcal{V}(\varphi_0, l)$ then

$$\text{either } (C_1 \cup \{p\}) \in \psi_0 \text{ or } (C_1 \cup \{p\} \cup \{l\}) \in \psi_0,$$

and

$$\text{either } (C_2 \cup \{\neg p\}) \in \varphi_0 \text{ or } (C_2 \cup \{\neg p\} \cup \{l\}) \in \varphi_0.$$

Then one of the following is possible.

- $(C_1 \cup \{p\}), (C_2 \cup \{\neg p\}) \in \psi_0$.
  If we choose
  $$\psi_1 = \psi_0 \cup \{C_1 \cup C_2\}$$
  then clearly
  $$\psi_0 \longmapsto \psi_1.$$
  We check that $\psi_1 \in \mathcal{V}(\varphi_1, l)$.
  Consider an arbitrary $C \in \varphi_1$. Then one of the following holds.
  - $C \equiv C_1 \cup C_2$.
    Then there is a $D \in \psi_1$ such that $D \equiv C$.
  - $C \in \varphi_0$.
    Then by the lemma conditions there is a $D \in \psi_0$ such that $D \equiv C$ or $D \equiv C \cup \{l\}$.
    Since $\psi_0 \subseteq \psi_1$ then there is a $D \in \psi_1$ such that $D \equiv C$ or $D \equiv C \cup \{l\}$.

- For all other cases we choose
  $$\psi_1 = \psi_0 \cup \{C_1 \cup C_2 \cup \{l\}\}.$$
  We check that $\psi_1 \in \mathcal{V}(\varphi_1, l)$.
  Consider an arbitrary $C \in \varphi_1$. Then one of the following holds.
  - $C = C_1 \cup C_2$.
    Then there is a $D \in \psi_1$ such that $D \equiv C \cup \{l\}$.

- $C \in \varphi_0$.
  Then by the lemma conditions there is a $D \in \psi_0$ such that $D \equiv C$ or $D \equiv C \cup \{l\}$.
  Since $\psi_0 \subseteq \psi_1$ then there is a $D \in \psi_1$ such that $D \equiv C$ or $D \equiv C \cup \{l\}$.

⊠

Lemma 4.5 *Suppose for* $\varphi_0, \ldots, \varphi_n, \psi_0 \in \mathsf{Cnf}$ *and* $l \in \mathsf{Lit}$ *the following holds.*

1. $\varphi_0 \longmapsto \ldots \longmapsto \varphi_n$.
2. $\psi_0 \in \mathcal{V}(\varphi_0, l)$.

Then there is a a resolution sequence

$$\psi_0 \longmapsto \ldots \longmapsto \psi_n,$$

where $\psi_i \in \mathcal{V}(\varphi_i, l)$, $1 \leq i \leq n$.

Proof We will give a proof by induction on n.

Base case. $n = 1$. The lemma holds by Lemma 4.4.

Inductive step. Let the lemma hold for $n - 1$.

Then there is a sequence

$$\psi_0 \longmapsto \ldots \longmapsto \psi_{n-1},$$

where $\psi_i \in \mathcal{V}(\varphi_i, l), 1 \leq i \leq n - 1$.

By Lemma 4.4 there is a $\psi_n \in \mathcal{V}(\varphi_n, l)$ such that $\psi_{n-1} \longmapsto \psi_n$.

We obtain that there are $\psi_0, \ldots, \psi_n$ satisfying the lemma assumption.

⊠

Lemma 4.6 *Suppose for* $\varphi, \varphi_0, \ldots, \varphi_n \in \mathsf{Cnf}$

$$\varphi \cup \{\{l\}\} \equiv \varphi_0 \overset{l}{\underset{u}{\longmapsto}} \varphi_1 \longmapsto \ldots \longmapsto \varphi_n.$$

Then there is a resolution sequence

$$\varphi \longmapsto \psi_1 \longmapsto \ldots \longmapsto \psi_{n-1},$$

where $\psi_i \in \mathcal{V}(\varphi_{i+1}, \neg l)$, $1 \leq i \leq n - 1$.

Proof Since $\varphi \in \mathcal{V}(\varphi_1, \neg l)$, the lemma holds by Lemma 4.5.

⊠

Lemma 4.7 *Suppose for $\varphi_0, \ldots, \varphi_n \in \mathsf{Cnf}$ and $l \in \mathsf{Lit}$*

1. $\varphi \cup \{\{l\}\} = \varphi_0 \longmapsto \varphi_1 \longmapsto \ldots \longmapsto \varphi_n$.
2. $\varphi_j \overset{l}{\underset{u}{\longmapsto}} \varphi_{j+1}$, *where* $j \in \{i_1, \ldots, i_m\}$ *and* $\{i_1, \ldots, i_m\} \subseteq \{0, \ldots, n-1\}$.

Then there is a resolution sequence

$$\varphi \longmapsto \psi_1 \longmapsto \ldots \longmapsto \psi_{n-m},$$

where $\psi_{n-m} \in \mathcal{V}(\varphi_n, \neg l)$.

Proof We will give a proof by induction on m.

Base case. Suppose $m = 1$. Then the lemma holds by Lemma 4.6.

Inductive step. Let the lemma hold for $m - 1$. Then the lemma holds for m by Lemma 4.5 and Lemma 4.6. ⊠

Lemma 4.7 is needed to prove Lemma 4.8.

Lemma 4.8 *Suppose for $\varphi \in \mathsf{Cnf}$ and $l \in \mathsf{Lit}$, the empty clause can be derived from $\varphi \cup \{\{l\}\}$ in m resolution steps. Then one of the following holds.*

- *The empty clause can be derived from φ in at most m resolution steps.*
- *The clause $\{\neg l\}$ can be derived from φ in at most $m - 1$ resolution steps.*

Proof Suppose

$$\varphi_0 = \varphi \cup \{\{l\}\}.$$

Then by the lemma assumption there are $\psi_0, \ldots, \psi_m \in \mathsf{Cnf}$ such that

$$\varphi_0 \longmapsto \ldots \longmapsto \varphi_m,$$

where $\perp \in \varphi_m$.

Then one of the following holds.

1. There is a $\varphi \longmapsto \varphi_1 \longmapsto \ldots \longmapsto \varphi_m$ such that $\bot \in \varphi_m$. Then $\bot$ can be derived in m resolution steps from V.
2. For $j \in \{i_1, \ldots, i_k\}$, $\varphi_j \overset{l}{\underset{u}{\longmapsto}} \varphi_{j+1}$, where $\{i_1, \ldots, i_k\} \subseteq \{0, \ldots, m-1\}$.

   Then by Lemma 4.7 there is a

   $$\varphi \longmapsto \psi_1 \longmapsto \ldots \longmapsto \psi_{m-k},$$

   where $\psi_{m-k} \in \mathcal{V}(\varphi_m, \neg l)$.

   Since $\bot \in \varphi_m$ then by Lemma 4.7 either $\{\neg l\} \in \psi_{m-k}$ or $\bot \in \psi_{m-k}$.

   We can conclude that either $\{\neg l\}$ or $\bot$ can be derived from φ in $m-k$ resolution steps, where $k > 0$.

$\boxtimes$

Now we prove the basic theorem we use in the following section.

Theorem 4.9 *Suppose for $\varphi \in \mathsf{Cnf}$ and $l \in \mathsf{Lit}$*

1. *the empty clause can be derived from $\varphi \cup \{\{l\}\}$ in $m > 0$ resolution steps, and*
2. *the empty clause can be derived from $\varphi \cup \{\{\neg l\}\}$ in $n > 0$ resolution steps.*

If both hold, then the empty clause can be derived from φ in at most $m + n - 1$ resolution steps.

Proof By Lemma 4.8 the empty clause can be derived from φ either in $\min(m, n)$ or in $m - 1 + n - 1 + 1 = m + n - 1$ resolution steps. In both cases we have the number of resolution steps no more than $m + n - 1$. $\boxtimes$

Example 4.10 *Consider*

$\varphi \equiv \{\{p_1, p_2, p_3\}, \{\neg p_2, p_3\}, \{p_1, \neg p_3\}, \{\neg p_1, p_4\}, \{\neg p_4, p_5\}, \{\neg p_4, \neg p_5\}\}$.

There is a resolution of length 4 starting from $\varphi \cup \{\{\neg p_1\}\}$, namely

$$\varphi \cup \{\{\neg p_1\}\} \longmapsto \varphi_1 \longmapsto \varphi_2 \longmapsto \varphi_3 \longmapsto \varphi_4,$$

where $\varphi_1 = \varphi \cup \{\{\neg p_1\}\} \cup \{\{p_2, p_3\}\}$, $\varphi_2 = \varphi_1 \cup \{\{p_3\}\}$, $\varphi_3 = \varphi_2 \cup \{\{\neg p_3\}\}$, $\varphi_4 = \varphi_3 \cup \{\bot\}$.

There is a resolution derivation of length 4 *starting from* $\varphi \cup \{\{p_1\}\}$, *namely*

$$\varphi \cup \{\{p_1\}\} \longmapsto \varphi_1 \longmapsto \varphi_2 \longmapsto \varphi_3 \longmapsto \varphi_4,$$

where $\varphi_1 = \varphi \cup \{\{p_1\}\} \cup \{\{p_4\}$, $\varphi_2 = \varphi_1 \cup \{\{p_5\}\}$, $\varphi_3 = \varphi_2 \cup \{\{\neg p_5\}\}$, $\varphi_4 = \varphi_3 \cup \{\bot\}$.

By Theorem 4.9 there is a resolution derivation starting from φ *of length at most* 7.

In fact, there is a resolution derivation of length 6, *namely*

$$\varphi \longmapsto \varphi_1 \longmapsto \varphi_2 \longmapsto \varphi_3 \longmapsto \varphi_4 \longmapsto \varphi_5 \longmapsto \varphi_6,$$

where $\varphi_1 = \varphi \cup \{\{p_1, p_3\}\}$, $\varphi_2 = \varphi_1 \cup \{\{p_1\}\}$, $\varphi_3 = \varphi_2 \cup \{\{\neg p_1, p_5\}\}$, $\varphi_4 = \varphi_3 \cup \{\{\neg p_1, \neg p_5\}\}$, $\varphi_5 = \varphi_4 \cup \{\{\neg p_1\}\}$, $\varphi_6 = \varphi_5 \cup \{\bot\}$.

4.2 A DPLL derivation tree

In this section we define a DPLL derivation tree and the length of a DPLL proof.

As it is mentioned in Chapter 3, the DPLL procedure simplifies a CNF by implementing unit propagation and pure literal elimination. Now we will give a formal definition of a pure literal.

Definition 4.11 (**Pure literal**)

- *A literal* l *in a CNF* φ *is called pure if* $\neg l \notin \mathsf{Lit}(\varphi)$.
- *The procedure of deleting all pure literals from* φ *is denoted as pure_lit*(φ).

The DPLL procedure constructs a derivation tree. A DPLL derivation tree is nothing but a static representation of the recursive calls in the execution of the usual DPLL procedure.

Definition 4.12 (**A DPLL derivation tree**)

- *A DPLL derivation tree* T, *or for simplicity a DPLL tree* T, *is a labelled tree such that the following holds:*
  - *every node is labelled with a unit resolution sequence;*

  - *every non-leaf node has two successors;*
  - *if a non-leaf node is labelled with the unit resolution sequence $\varphi_1, \ldots, \varphi_n$ then its left successor node is labelled with a unit resolution sequence starting from pure_lit($\varphi_n|_p$) and its right successor node is labelled with a unit resolution sequence starting from pure_lit($\varphi_n|_{\neg p}$), where $p \in \mathsf{Pr}(\varphi_n)$.*

- *We say that a DPLL derivation tree is a derivation tree of φ if the root node is labelled with a unit resolution sequence starting from φ.*
- *A DPLL tree on an unsatisfiable formula is called a DPLL refutation.*

If T is a DPLL tree of φ then φ is satisfiable if and only if there is a leaf in T labelled with a unit resolution sequence $\varphi_1, \ldots, \varphi_n$ such that $\varphi_n = \emptyset$. Hence, building a DPLL tree implies a decision procedure for satisfiability. Therefore, we speak about a DPLL proof rather than a DPLL tree.

Definition 4.13 (**The length of a DPLL proof**)

- *Suppose a node is labelled with pure_lit($\varphi|_l$) $\longmapsto \varphi_1 \longmapsto \cdots \longmapsto \varphi_n$ and the number of $\neg l$ in φ is m. Then the number of unit resolution steps corresponding to the node is defined to be $n + m$.*
- *Suppose a DPLL tree T contains nodes $n_1, \ldots, n_k$, and the number of unit resolution steps corresponding to n_i is s_i, where $i \in \{1, \ldots, k\}$. Then the total number of unit resolution steps corresponding to T is defined to be $s_1 + \cdots + s_k$.*
- *The total number of unit resolution steps corresponding to a DPLL tree is called the* length of a DPLL proof.

4.3 Upper bounds on resolution refutation length

In this section we give two upper bounds on the resolution refutation length measured in the length of the DPLL refutation and the number of its nodes. The first one follows from a direct analysis of a DPLL refutation. The second one has an extra restriction on the resolution sequences used in the first result. The second bound is stronger. We even show that it is tight.

Theorem 4.14 *For each unsatisfiable $\varphi \in \mathsf{Cnf}$ exists a resolution refutation on φ of length at most*

$$s - (r - 1)/2,$$

where s is the length of a DPLL refutation on φ, and r is the number of nodes of this DPLL refutation on φ.

Proof We give a proof by induction on r.

Base case. Suppose $r = 1$. Then $s - (1-1)/2 = s$. We can conclude that the lemma holds.

Inductive step. Suppose the lemma holds for a DPLL refutation with the number of nodes at most $r-1$. Consider a DPLL refutation of φ with the number of nodes r. Consider the two subtrees rooted at the successors of the root. The number of nodes of each subtree is at most $r-2$. Then by the induction hypothesis the lemma holds for the subtrees.

Let the first subtree have a DPLL refutation of length s_1 and the number of its nodes be r_1. Let the other subtree have a DPLL refutation of length s_2 and the number of its nodes be r_2. Let s_0 be the number of unit resolution steps corresponding to the root.

We obtain from Theorem 4.9 that the length of a resolution refutation on φ is at most

$$s_0 + ((s_1 - (r_1 - 1)/2) + (s_2 - (r_2 - 1)/2) - 1) = s - (r-1)/2,$$

where $s = s_0 + s_1 + s_2$, $r = r_1 + r_2 + 1$. ⊠

Example 4.15 Consider the well-known pigeonhole formula P_n for $n = 2$.

$$P_2 \equiv \{\{p_{11}, p_{12}\}, \{p_{21}, p_{22}\}, \{p_{31}, p_{32}\}, \{\neg p_{11}, \neg p_{21}\}, \{\neg p_{12}, \neg p_{22}\},$$

$$\{\neg p_{11}, \neg p_{31}\}, \{\neg p_{12}, \neg p_{32}\}, \{\neg p_{21}, \neg p_{31}\}, \{\neg p_{22}, \neg p_{32}\}\}.$$

A DPLL refutation for P_2 is depicted in Figure 4.1. The nodes have the following labels.

$1:\ P_2$.

$2:\ P_2^2 \longmapsto P_2^2 \cup \{\{p_{22}\}\}$,
where $P_2^2 \equiv \{\{p_{21}, p_{22}\}, \{p_{31}, p_{32}\}, \{\neg p_{21}\}, \{\neg p_{12}, \neg p_{22}\}, \{\neg p_{31}\}, \{\neg p_{12}, \neg p_{32}\}, \{\neg p_{21}, \neg p_{31}\}, \{\neg p_{22}, \neg p_{32}\}\}$.

$3:\ P_2^3 \longmapsto P_2^3 \cup \{\{p_{32}\}\} \longmapsto P_2^3 \cup \{\{p_{32}\}\} \cup \{\bot\}$,

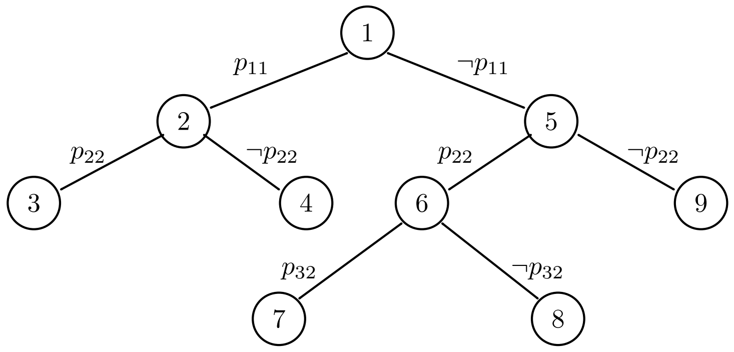

Figure 4.1 The DPLL refutation of P_2

where $P_2^3 = \{\{p_{31}, p_{32}\}, \{\neg p_{21}\}, \{\neg p_{12}\}, \{\neg p_{31}, \{\neg p_{12}, \neg p_{32}\}, \{\neg p_{21}, \neg p_{31}\}, \{\neg p_{32}\}, \{p_{22}\}\}$.

$4: \ P_2^4$,

where $P_2^4 = \{\{p_{21}\}, \{p_{31}, p_{32}\}, \{\neg p_{21}\}, \{\neg p_{31}\}, \{\neg p_{12}, \neg p_{32}\}, \{\neg p_{21}, \neg p_{31}\}, \bot\}$.

$5: \ P_2^5 \longmapsto P_2^5 \cup \{\{\neg p_{32}\}\}$,

where $P_2^5 = \{\{p_{12}\}, \{p_{21}, p_{22}\}, \{p_{31}, p_{32}\}, \{\neg p_{12}, \neg p_{22}\}, \{\neg p_{12}, \neg p_{32}\}, \{\neg p_{21}, \neg p_{31}\}, \{\neg p_{22}, \neg p_{32}\}\}$.

$6: \ P_2^6$,

where $P_2^6 = \{\{p_{12}\}, \{p_{31}, p_{32}\}, \{\neg p_{12}\}, \{\neg p_{12}, \neg p_{32}\}, \{\neg p_{21}, \neg p_{31}\}, \{\neg p_{32}\}\}$.

$7: \ P_2^7$,

where $P_2^7 = \{\{p_{12}\}, \{\neg p_{12}\}, \{\neg p_{21}, \neg p_{31}\}, \bot\}$.

$8: \ P_2^8 \longmapsto P_2^8 \cup \{\bot\}$,

where $P_2^8 = \{\{p_{12}\}, \{p_{31}\}, \{\neg p_{12}\}, \{\neg p_{21}, \neg p_{31}\}\}$.

$9: \ P_2^9 \longmapsto P_2^9 \cup \{\{p_{31}\}\} \longmapsto P_2^9 \cup \{\{p_{31}\}\} \cup \{\{\neg p_{21}\}\} \longmapsto P_2^9 \cup \{\{p_{31}\}\} \cup \{\{\neg p_{21}\}\} \cup \{\bot\}$,

where $P_2^9 = \{\{p_{12}\}, \{p_{21}\}, \{p_{31}, p_{32}\}, \{\neg p_{12}, \neg p_{32}\}, \{\neg p_{21}, \neg p_{31}\}, \{\neg p_{32}\}\}$.

The DPLL refutation on P_2 has $s = 20$ unit resolution steps and $r = 9$ nodes. Then by Theorem 4.14 there is a resolution refutation of P_2 of length at most 16.

We can improve the upper bound by making a restriction on the unit resolution sequences associated with nodes of the DPLL refutation.

Definition 4.16 *We say that a unit resolution sequence is* complete *if no unit resolution steps can be applied at the last CNF of the sequence.*

For the proofs of Theorem 4.20 and Theorem 4.21 we use the following lemmas.

Lemma 4.17 *Suppose $\varphi \in \mathsf{Cnf}$ contains no pure literals, and no unit resolution can be applied. Then φ contains no unit clauses.*

Proof We give a proof by contradiction.

Assume that

$$\varphi = \{\{l\}\} \cup \varphi',$$

where $\varphi' \in \mathsf{Cnf}$.

By the assumption of the lemma φ contains no pure literals so there is a $C \in V'$ such that $\neg l \in C$. Then at least one unit resolution step can be applied. We have a contradiction. We can conclude that φ contains no unit clauses. ⊠

Now we arrive at the crucial observation mentioned in the introduction.

Lemma 4.18 *Suppose for $\varphi \in \mathsf{Cnf}$ and $l \in \mathsf{Lit}$*

1. *φ contains no unit clauses,*
2. *the empty clause can be derived from $\varphi \cup \{\{l\}\}$ in $n \geq 2$ unit resolution steps.*

Then either the clause $\{\neg l\}$ or the empty clause can be derived from φ in at most $n-2$ resolution steps.

Proof We give a proof by induction on n.

Base case. Suppose $n = 2$.

Since φ does not contain unit clauses, the empty clause cannot be derived from φ in two unit resolution steps.

Inductive step. Let the lemma hold for $n-1$.

Suppose the empty clause can be derived from $\varphi \cup \{\{l\}\}$ in $n \geq 2$ unit resolution steps. Then one of the following holds.

- The unit resolution sequence contains more than one l-step.

  In this case the lemma holds by Lemma 4.7.

- The unit resolution sequence contains exactly one l-step.

  Since φ contains no unit clauses the first unit resolution step is an l-step, and this is the only l-step.

  We obtain

  $$\varphi \cup \{\{l\}\} \overset{l}{\underset{u}{\longmapsto}} \varphi \cup \{\{l\}\} \cup \{C\},$$

  where $C \in \mathsf{Cls}$.

  Then the empty clause can be derived from $\varphi \cup \{\{l\}\} \cup \{C\}$ in $n-1$ unit resolution step.

  As the resolution sequence contains only one l-step then the empty clause can be derived from $\varphi \cup \{C\}$ in $n-1$ unit resolution step. Since φ contains no unit clauses, C must be $\{l'\}$.

  By the induction hypothesis either $\{\neg l'\}$ or $\bot$ can be derived from $\varphi \cup \{\{l'\}\}$ in at most $n-3$ resolution steps. Since by an l-step we derived $\{l'\}$ we can conclude that

  $$\{\neg l, l'\} \in \varphi.$$

  We need one extra resolution step in case we derived $\{\neg l'\}$. We obtain that either $\{\neg l\}$ or $\bot$ can be derived from φ in at most $n-2$ resolution steps.

$\boxtimes$

Example 4.19 Consider

$$\varphi \equiv \{\{\neg p_1, p_2\}, \{\neg p_2, p_3\}, \{\neg p_2, p_4\}, \{\neg p_3, \neg p_4\}\},$$

and

$$l \equiv p_1.$$

There is a resolution derivation of the empty clause from $\varphi \cup \{\{p_1\}\}$ in 5 unit resolution steps, namely

$$\varphi \cup \{\{p_1\}\} \longmapsto \varphi_1 \longmapsto \varphi_2 \longmapsto \varphi_3 \longmapsto \varphi_4 \longmapsto \varphi_5,$$

where

$\varphi_1 = \varphi \cup \{\{p_2\}\}$, $\varphi_2 = \varphi_1 \cup \{\{p_3\}\}$, $\varphi_3 = \varphi_2 \cup \{\{p_4\}\}$, $\varphi_4 = \varphi_3 \cup \{\{\neg p_4\}\}$, $\varphi_5 = \varphi_4 \cup \{\bot\}$.

According Lemma 4.18 a resolution derivation of $\{\neg p_1\}$ from φ in at most 3 resolution steps exists. In fact, there is a resolution derivation of length 3, namely

$$\varphi \longmapsto \varphi_1' \longmapsto \varphi_2' \longmapsto \varphi_3',$$

where

$\varphi_1' = \varphi \cup \{\{\neg p_2, \neg p_4\}\}$, $\varphi_2' = \varphi_1' \cup \{\{\neg p_2\}\}$, $\varphi_3' = \varphi_2' \cup \{\{\neg p_1\}\}$.

Just like Theorem 4.9 was needed to prove Theorem 4.14 we now state Theorem 4.20 in order to prove Theorem 4.21.

Theorem 4.20 *Suppose for $\varphi \in \mathsf{Cnf}$ and $l \in \mathsf{Lit}$ the following holds.*

1. *φ contains no unit clauses.*
2. *The empty clause can be derived from $\varphi \cup \{\{l\}\}$ in $m > 2$ unit resolution steps.*
3. *The empty clause can be derived from $\varphi \cup \{\{\neg l\}\}$ in $n > 2$ unit resolution steps.*

Then the empty clause can be derived from φ in $m + n - 3$ resolution steps.

Proof If $\bot$ can be derived from $\varphi \cup \{\{l\}\}$ in $m > 2$ resolution steps then by Lemma 4.18 one of the following holds.

1. $\bot$ can be derived from φ in at most $m - 2$ resolution steps.
2. $\neg l$ can be derived from φ in at most $m - 2$ resolution steps.

   If $\bot$ can be derived from $\varphi \cup \{\{\neg l\}\}$ in $n > 2$ resolution steps then by Lemma 4.18 one of the following holds

3. $\bot$ can be derived from φ in at most $n - 2$ resolution steps.
4. l can be derived from φ in at most $n - 2$ resolution steps.

In cases 1 or 3, $\bot$ can be derived from φ in $\min(m-2, n-2)$ resolution steps. For $m > 2$ and $n > 2$,

$$\min(m - 2, n - 2) \leq m + n - 3.$$

When combining case 2 and case 4 we need one extra resolution step to get $\bot$. And it can be derived in

$$m - 2 + n - 2 + 1 = m + n - 3$$

resolution steps. ⊠

Theorem 4.21 *Suppose for unsatisfiable $\varphi \in \mathsf{Cnf}$ the following holds.*

1. *The number of nodes of the DPLL refutation of φ is $r \geq 3$,*
2. *The DPLL refutation of φ has length s,*
3. *Each unit resolution sequence associated with a node is complete.*

Then there is a resolution refutation starting from φ of length at most

$$s - r.$$

Proof We give a proof by induction on r.

Base case. Suppose $r = 3$. In this case the Lemma holds by Lemma 4.17 and Theorem 4.20.

Inductive step. Assume that the lemma holds for the subtrees rooted at the successors of the root.

Suppose one subtree has a DPLL refutation of length s_1 and the number of its nodes is r_1. Suppose that another subtree has a DPLL refutation of length s_2 and the number of its nodes is r_2. Let s_0 be a number of unit resolution steps corresponding to the root.

We obtain by Theorem 4.9 that there exists a resolution refutation of φ with length at most

$$s_0 + ((s_1 - r_1) + (s_2 - r_2) - 1) = s - r,$$

where $s = s_0 + s_1 + s_2$, $r = r_1 + r_2 + 1$. ⊠

Using the result a DPLL refutation can be transformed to a resolution refutation of shorter length.

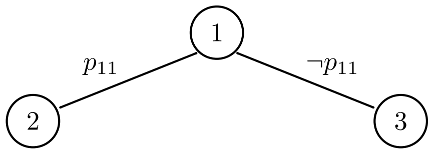

Figure 4.2 The DPLL refutation of P_2

Example 4.22 Consider the pigeonhole formula P_2 from Example 4.15. A DPLL refutation of P_2 is depicted in Figure 4.2.

The nodes have the following labels.

1 : P_2.

2 :

$$\begin{array}{lll} P_2^2 & \longmapsto & P_2^2 \cup \{\{p_{22}\}\} \\ & \longmapsto & P_2^2 \cup \{\{p_{22}\}\} \cup \{\{\neg p_{12}\}\} \\ & \longmapsto & P_2^2 \cup \{\{p_{22}\}\} \cup \{\{\neg p_{12}\}\} \cup \{\{p_{32}\}\} \\ & \longmapsto & P_2^2 \cup \{\{p_{22}\}\} \cup \{\{\neg p_{12}\}\} \cup \{\{p_{32}\}\} \cup \{\{\neg p_{32}\}\} \\ & \longmapsto & P_2^2 \cup \{\{p_{22}\}\} \cup \{\{\neg p_{12}\}\} \cup \{\{p_{32}\}\} \cup \{\{\neg p_{32}\}\} \cup \{\{\bot\}\} \end{array}$$

where $P_2^2 \equiv \{\{p_{21}, p_{22}\}, \{p_{31}, p_{32}\}, \{\neg p_{21}\}, \{\neg p_{12}, \neg p_{22}\}, \{\neg p_{31}\}, \{\neg p_{12}, \neg p_{32}\},$ $\{\neg p_{21}, \neg p_{31}\}, \{\neg p_{22}, \neg p_{32}\}\}$.

3 :

$$\begin{array}{lll} P_2^3 & \longmapsto & P_2^3 \cup \{\{\neg p_{22}\}\} \\ & \longmapsto & P_2^3 \cup \{\{\neg p_{22}\}\} \cup \{\{\neg p_{32}\}\} \\ & \longmapsto & P_2^3 \cup \{\{\neg p_{22}\}\} \cup \{\{\neg p_{32}\}\} \cup \{\{p_{21}\}\} \\ & \longmapsto & P_2^3 \cup \{\{\neg p_{22}\}\} \cup \{\{\neg p_{32}\}\} \cup \{\{p_{21}\}\} \cup \{\{p_{31}\}\} \\ & \longmapsto & P_2^3 \cup \{\{\neg p_{22}\}\} \cup \{\{\neg p_{32}\}\} \cup \{\{p_{21}\}\} \cup \{\{p_{31}\}\} \cup \{\{\neg p_{31}\}\} \\ & \longmapsto & P_2^3 \cup \{\{\neg p_{22}\}\} \cup \{\{\neg p_{32}\}\} \cup \{\{p_{21}\}\} \cup \{\{p_{31}\}\} \cup \{\{\neg p_{31}\}\} \cup \{\bot\} \end{array}$$

where $P_2^3 = \{\{p_{12}\}, \{p_{21}, p_{22}\}, \{p_{31}, p_{32}\}, \{\neg p_{12}, \neg p_{22}\}, \{\neg p_{12}, \neg p_{32}\}, \{\neg p_{21}, \neg p_{31}\},$ $\{\neg p_{22}, \neg p_{32}\}\}$.

The DPLL refutation depicted in Figure 4.2 is complete, i.e. for every node all possible unit resolution steps are done. The DPLL refutation satisfies the conditions of Theorem 4.21. A reader can check that $s = 14$ and $r = 3$. Then by Theorem 4.21 there is a resolution sequence of length at most $s - r = 14 - 3 = 11$, i.e. this upper bound is stronger than in Example 4.15.

Example 4.23 Consider

$$\varphi \equiv \{\{p, q, r\}, \{p, q, \neg r\}, \{p, \neg q, r\}, \{p, \neg q, \neg r\}, \{\neg p, q, r\},$$

$$\{\neg p, q, \neg r\}, \{\neg p, \neg q, r\}, \{\neg p, \neg q, \neg r\}.$$

The DPLL refutation of φ is depicted in Figure 4.3.

The root of the DPPL refutation is labelled with φ.

Node 2 is labelled with

$$\varphi_n|_p = \{q, r\}, \{q, \neg r\}, \{\neg q, r\}, \{\neg q, \neg r\}.$$

Node 3 is labelled with

$$\varphi_n|_{\neg p} = \{q, r\}, \{q, \neg r\}, \{\neg q, r\}, \{\neg q, \neg r\}.$$

Nodes $4, 5, 6$ and 7 are labelled with a resolution sequence

$$\{\{r\}, \{\neg r\}\} \longmapsto \{\{r\}, \{\neg r\}, \bot\}.$$

The DPLL refutation of φ has 7 nodes. A reader can easily check that by Definition 4.13 the number of unit resolution steps for the DPLL refutation of φ is 20. Then by Theorem 4.21 there is a resolution refutation of length at most 13.

In fact, there exists the resolution refutation

$$\varphi \longmapsto \varphi_1 \longmapsto \varphi_2 \longmapsto \varphi_3 \longmapsto \varphi_4 \longmapsto \varphi_5 \longmapsto \varphi_6 \longmapsto \varphi_7,$$

where $\varphi_1 = \varphi \cup \{\{p, q\}\}$, $\varphi_2 = \varphi_1 \cup \{\{p, \neg q\}\}$, $\varphi_3 = \varphi_2 \cup \{\{\neg p, q\}\}$, $\varphi_4 = \varphi_3 \cup \{\{\neg p, \neg q\}\}$, $\varphi_5 = \varphi_4 \cup \{\{p\}\}$ $\varphi_6 = \varphi_5 \cup \{\{\neg p\}\}$, $\varphi_7 = \varphi_6 \cup \{\bot\}$.

4.4 Tightness of the upper bound

One can wonder whether Theorem 4.21 can be improved further. In this section by analyzing a certain CNF class we show that it cannot be done.

Definition 4.24 *A CNF is* minimally unsatisfiable *if it is unsatisfiable and each of its subsets is satisfiable.*

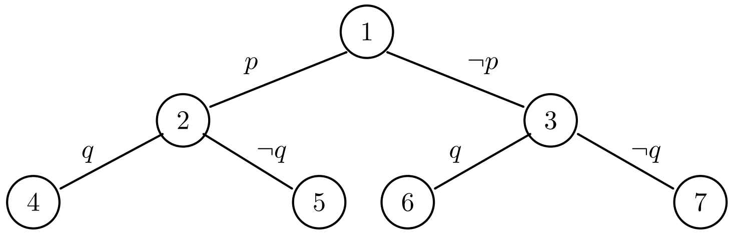

Figure 4.3 The DPLL refutation of φ

Definition 4.25 *We say that a directed graph $G(V,E)$, where V is a set of vertices and E is a set of edges is a* resolution graph *for the resolution sequence $\varphi_0, \ldots, \varphi_n$, where $\varphi_n \equiv \{C_1, \ldots, C_m\}$, if the following holds.*

- *$V = \{C_1, \ldots, C_m\}$.*
- *for each $(C, C') \in E$ there is a $r \in \{1, \ldots, n-1\}$, $p \in \mathsf{At}$, and $D \in V$ such that*

$$\varphi_{r+1} \equiv \varphi_r \cup \{C'\}, \text{ where } C' \equiv (C \cup D) \backslash \{p, \neg p\}.$$

Example 4.26 Consider $\varphi_0 \equiv \{\{p,q\},\{p,\neg q\},\{\neg p,q\},\{\neg p,\neg q\}\}$. This CNF is unsatisfiable. The empty clause can be derived by the resolution sequence

$$\varphi_0 \rightarrow \varphi_1 \rightarrow \varphi_2 \rightarrow \varphi_3,$$

where $\varphi_1 \equiv \varphi_0 \cup \{\{p\}\}$, $\varphi_2 \equiv \varphi_1 \cup \{\{\neg p\}\}$, $\varphi_3 \equiv \varphi_2 \cup \{\bot\}$.

The resolution graph representing this resolution sequence is depicted in Figure 4.4.

Lemma 4.27 *Suppose for $\varphi \in \mathsf{Cnf}$ the following holds.*

1. *φ is minimally unsatisfiable.*
2. *φ contains n clauses.*

Then the shortest resolution refutation of φ has length at least

$$n - 1.$$

Proof We give a proof by contradiction.

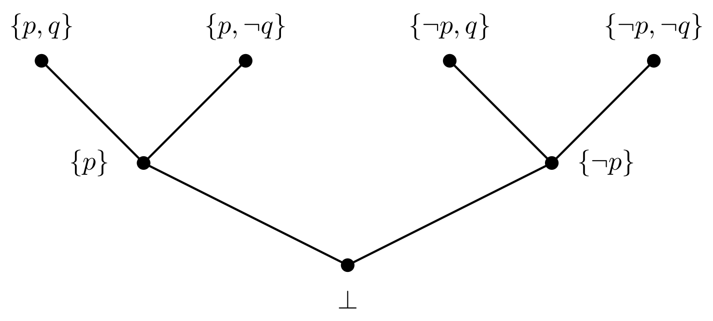

Figure 4.4 The resolution graph representing the resolution sequence $\varphi_0 \rightarrow \varphi_1 \rightarrow \varphi_2 \rightarrow \varphi_3$

Suppose

$$\varphi \longmapsto \varphi_1 \longmapsto \ldots \longmapsto \varphi_r$$

is a resolution refutation of φ.

Suppose $G(V^*, E^*)$ is a resolution graph of the resolution sequence $\varphi, \varphi_1 \ldots \varphi_r$.

By construction,

$$|V^*| = n + r \text{ and } |E^*| = 2r.$$

Assume that

$$r < n - 1.$$

We obtain that

$$|E^*| < |V^*| - 1$$

and we can conclude that G is a disconnected graph.

If G is a disconnected graph then there is a $C \in V$ such that there is no path from C to $\perp$ on G. In this case $\perp$ can be derived from $V \backslash C$. We have a contradiction with the assumption that φ is minimally satisfiable. ⊠

We introduce a class of CNFs to prove that the upper bound presented by Theorem 4.21 is tight.

We define for $n \geq 1$, $\varphi_n \equiv \{\{\neg p_1, q_1\}$,

$\{\neg p_2, \neg q_1, q_2\}, \ldots, \{\neg p_n, \neg q_{n-1}, q_n\}$,

$\{\neg q_n, q_{n+1}\}, \{\neg q_n, q_{n+2}\}, \{\neg q_{n+1}, \neg q_{n+2}\},$

$\{p_1, r_{11}\}, \{\neg r_{11}, r_{21}\}, \{\neg r_{11}, r_{31}\}, \{\neg r_{31}, \neg r_{21}\},$

$\ldots,$

$\{p_n, r_{1n}\}, \{\neg r_{1n}, r_{2n}\}, \{\neg r_{1n}, r_{3n}\}, \{\neg r_{2n}, \neg r_{3n}\}\}.$

Lemma 4.28 φ_n*,* $n \geq 1$*, is minimally unsatisfiable.*

Proof We prove that φ_n is minimally unsatisfiable by giving a satisfying assignment for ψ obtained by deleting a clause from φ_n.

1. $\psi \equiv \varphi_n \backslash \{\neg p_1, q_1\}$,

   $\forall i \in \{1, \ldots, n\}$ $p_i = 1$, $r_{1i} = 0$, $r_{2i} = 0$, $r_{3i} = 0$; $\forall i \in \{1, \ldots, n+2\}$ $q_i = 0$.

2. $\psi \equiv \varphi_n \backslash \{\neg p_k, \neg q_{k-1}, q_k\}$, $2 \leq k \leq n$,

   $\forall i \in \{1, \ldots, n\}$ $p_i = 1$, $r_{1i} = 0$, $r_{2i} = 0$, $r_{3i} = 0$; $\forall i \in \{1, \ldots, n+2\} \backslash \{k\}$ $q_i = 0$; $q_k = 1$.

3. $\psi \equiv \varphi_n \backslash \{\neg q_n, q_{n+1}\}$,

   $\forall i \in \{1, \ldots, n\}$ $p_i = 1$, $r_{1i} = 0$, $r_{2i} = 0$, $r_{3i} = 0$; $\forall i \in \{1, \ldots, n+2\} \backslash \{n+1\}$ $q_i = 1$; $q_{n+1} = 0$.

4. $\psi \equiv \varphi_n \backslash \{\neg q_n, q_{n+2}\}$,

   $\forall i \in \{1, \ldots, n\}$ $p_i = 1$, $r_{1i} = 0$, $r_{2i} = 0$, $r_{3i} = 0$; $\forall i \in \{1, \ldots, n+1\}$ $q_i = 1$; $q_{n+2} = 0$.

5. $\psi \equiv \varphi_n \backslash \{\neg q_{n+1}, \neg q_{n+2}\}$,

   $\forall i \in \{1, \ldots, n\}$ $p_i = 1$, $r_{1i} = 0$, $r_{2i} = 0$, $r_{3i} = 0$; $\forall i \in \{1, \ldots, n+2\}$ $q_i = 1$.

6. $\psi \equiv \varphi_n \backslash \{p_k, r_{1k}\}$, $1 \leq k \leq n$,

   $\forall i \in \{1, \ldots, n\} \backslash \{k\}$ $p_i = 1$; $p_k = 0$; $\forall i \in \{1, \ldots, n\}$ $r_{1i} = 0$, $r_{2i} = 0$, $r_{3i} = 0$; $\forall q_i \in \{0, \ldots, k-1\}$ $q_i = 1$; $\forall q_i \in \{k, \ldots, n+2\}$ $q_i = 0$;

7. $\psi \equiv \varphi_n \backslash \{\neg r_{1k}, r_{2k}\}$, $1 \leq k \leq n$,

   $\forall i \in \{1, \ldots, n\} \backslash \{k\}$ $p_i = 1$; $p_k = 0$; $\forall i \in \{1, \ldots, n\} \backslash \{k\}$ $r_{1i} = 0$; $r_{1k} = 1$; $\forall i \in \{1, \ldots, n\}$ $r_{2i} = 0$; $\forall i \in \{1, \ldots, n\} \backslash \{k\}$ $r_{3i} = 0$; $r_{3k} = 1$; $\forall q_i \in \{0, \ldots, k-1\}$ $q_i = 1$; $\forall q_i \in \{k, \ldots, n+2\}$ $q_i = 0$.

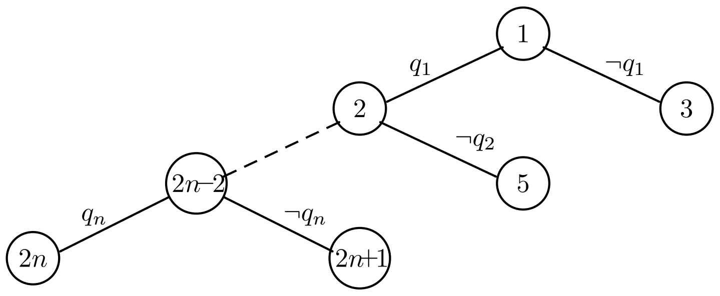

Figure 4.5 The DPLL refutation of φ_n

8. $\psi \equiv \varphi_n \backslash \{\neg r_{1k}, r_{3k}\}$,

   $\forall i \in \{1,\dots,n\}\backslash\{k\}\ p_i = 1;\ p_k = 0;\ \forall i \in \{1,\dots,n\}\backslash\{k\}\ r_{1i} = 0;\ r_{1k} = 1;$ $\forall i \in \{1,\dots,n\}\backslash\{k\}\ r_{2i} = 0;\ r_{2k} = 1;\ \forall i \in \{1,\dots,n\}\ r_{3i} = 0;\ \forall q_i \in \{0,\dots,k-1\}\ q_i = 1;\ \forall q_i \in \{k,\dots,n+2\}\ q_i = 0.$

9. $\psi \equiv \varphi_n \backslash \{\neg r_{2k}, \neg r_{3k}\}$,

   $\forall i \in \{1,\dots,n\}\backslash\{k\}\ p_i = 1;\ p_k = 0;\ \forall i \in \{1,\dots,n\}\ r_{1i} = 0;\ \forall i \in \{1,\dots,n\}\backslash\{k\}$ $r_{2i} = 0;\ r_{2k} = 1;\ \forall i \in \{1,\dots,n\}\backslash\{k\}\ r_{3i} = 0;\ r_{3k} = 1;\ \forall q_i \in \{0,\dots,k-1\}$ $q_i = 1;\ \forall q_i \in \{k,\dots,n+2\}\ q_i = 0.$

$\boxtimes$

Lemma 4.29 *φ_n, $n \geq 1$, has a DPLL refutation of length $7n+3$ and the number of its nodes is $2n+1$.*

Proof The DPLL refutation of φ_n is depicted in Figure 4.5.

Node 1 is labelled with φ_n.

For $i = 1,\dots,n-1$, Node $2i+1$ is labelled with the resolution sequence

$$\begin{aligned}
\psi_{2i+1} &\longmapsto \psi_{2i+1} \cup \{r_{1i}\} \\
&\longmapsto \psi_{2i+1} \cup \{r_{1i}\} \cup \{r_{2i}\} \\
&\longmapsto \psi_{2i+1} \cup \{r_{1i}\} \cup \{r_{2i}\} \cup \{r_{3i}\} \\
&\longmapsto \psi_{2i+1} \cup \{r_{1i}\} \cup \{r_{2i}\} \cup \{r_{3i}\} \cup \{\neg r_{3i}\} \\
&\longmapsto \psi_{2i+1} \cup \{r_{1i}\} \cup \{r_{2i}\} \cup \{r_{3i}\} \cup \{\neg r_{3i}\} \cup \bot
\end{aligned}$$

where $\psi_{2i+1} = \{\{\neg p_i\}, \{\neg p_{i+2}, \neg q_{i+1}, q_{i+2}\}, \dots, \{\neg p_n, \neg q_{n-1}, q_n\}, \{\neg q_n, q_{n+1}\},$ $\{\neg q_n, q_{n+2}\}, \{\neg q_{n+1}, \neg q_{n+2}\}, \{p_1, r_{11}\}, \{\neg r_{11}, r_{21}\}, \{\neg r_{11}, r_{31}\}, \{\neg r_{31}, \neg r_{21}\}, \dots,$ $\{p_n, r_{1n}\}, \{\neg r_{1n}, r_{2n}\}, \{\neg r_{1n}, r_{3n}\}, \{\neg r_{2n}, \neg r_{3n}\}\}$.

Node $2n+1$ is labelled with the resolution sequence

$$\begin{array}{lll} \psi_{2n+1} & \longmapsto & \psi_{2n+1} \cup \{r_{1n}\} \\ & \longmapsto & \psi_{2n+1} \cup \{r_{1n}\} \cup \{r_{2n}\} \\ & \longmapsto & \psi_{2n+1} \cup \{r_{1n}\} \cup \{r_{2n}\} \cup \{r_{3n}\} \\ & \longmapsto & \psi_{2in+1} \cup \{r_{1n}\} \cup \{r_{2n}\} \cup \{r_{3n}\} \cup \{\neg r_{3n}\} \\ & \longmapsto & \psi_{2n+1} \cup \{r_{1n}\} \cup \{r_{2n}\} \cup \{r_{3n}\} \cup \{\neg r_{3n}\} \cup \bot \end{array}$$

where $\psi_{2n+1} = \{\{\neg p_n, \}, \{\neg q_{n+1}, \neg q_{n+2}\}, \{p_1, r_{11}\}, \{\neg r_{11}, r_{21}\}, \{\neg r_{11}, r_{31}\},$ $\{\neg r_{31}, \neg r_{21}\}, \dots, \{p_n, r_{1n}\}, \{\neg r_{1n}, r_{2n}\}, \{\neg r_{1n}, r_{3n}\}, \{\neg r_{2n}, \neg r_{3n}\}\}$.

For $i = 1, \dots, n-1$, Node $2i$ is labelled with the CNF

$\psi_{2i} = \{\{\neg p_{2i+1}, q_{2i+1}\}, \{\neg p_{2i+2}, \neg q_{2i+1}, q_{2i+2}\}, \dots, \{\neg p_n, \neg q_{n-1}, q_n\}\ \{\neg q_n, q_{n+1}\},$ $\{\neg q_n, q_{n+2}\}, \{\neg q_{n+1}, \neg q_{n+2}\}, \{p_1, r_{11}\}, \{\neg r_{11}, r_{21}\}, \{\neg r_{11}, r_{31}\}, \{\neg r_{31}, \neg r_{21}\}, \dots,$ $\{p_n, r_{1n}\},\ \{\neg r_{1n}, r_{2n}\},\ \{\neg r_{1n}, r_{3n}\},\ \{\neg r_{2n}, \neg r_{3n}\}\}$.

Node $2n$ is labelled with the resolution sequence

$$\begin{array}{lll} \psi_{2n} & \longmapsto & \psi_{2n} \cup \{\neg q_{n+2}\} \\ & \longmapsto & \psi_{2n} \cup \{\neg q_{n+2}\} \cup \bot \end{array}$$

where $\psi_{2n} = \{q_{n+1}\}, \{q_{n+2}\}, \{\neg q_{n+1}, \neg q_{n+2}\}, \{p_1, r_{11}\}, \{\neg r_{11}, r_{21}\}, \{\neg r_{11}, r_{31}\},$ $\{\neg r_{31}, \neg r_{21}\}, \dots, \{p_n, r_{1n}\}, \{\neg r_{1n}, r_{2n}\}, \{\neg r_{1n}, r_{3n}\}, \{\neg r_{2n}, \neg r_{3n}\}\}$.

Suppose s_i, $1 \le i \le 2n+1$, is a number of unit resolution steps corresponding to a node i. Then taking into account Definition 4.13 a reader can check that $s_1 = 0$; $s_{2i} = 1$, $1 \le i \le n-1$; $s_{2i+1} = 6$, $1 \le i \le n$; $s_{2n} = 4$. We obtain that a total number of unit resolution steps is $s = s_1 + \cdots + s_{2n+1} = 7n + 1$.

We obtain that φ_n, $n \ge 1$, has a DPLL refutation of length $7n+3$ and the number of its nodes is $2n+1$. ⊠

Now we are ready to prove the tightness of our main result.

Theorem 4.30 *Suppose a DPLL refutation of φ_n, $n \geq 1$, has a length s and the number of its nodes is r. Then there is no resolution refutation of φ_n of a length less than $s - r$.*

Proof Consider φ_n, $n \geq 1$. By Lemma 4.29, φ_n has a DPLL refutation of length $7n + 3$ and the number of its nodes is $2n + 1$. By Theorem 4.21, there is a resolution refutation of φ_n of length less or equal $5n + 2$.

Since φ_n has $5n + 3$ clauses then by Lemma 4.27 there is no resolution refutation on φ_n of length less than $5n + 2$. ⊠

Chapter 5

A resolution-based proof system for Equality Logic

"The perfection of mathematical beauty is such ... that whatsoever is most beautiful and regular is also found to be the most useful and excellent"

[Sir D'Arcy Wentworth Thompson]

The main problem dealt with in this chapter is: given an equality formula, decide whether it is satisfiable or not. We describe a resolution-based approach to deal with equality formulae. One of the problems that our method must deal with is detecting inconsistent combinations of atoms, such as

$$a_1 \approx a_2, \ldots, a_{n-1} \approx a_n, a_1 \not\approx a_n.$$

It is well-known that by the combination of paramodulation and resolution the empty clause can be deduced from an unsatisfiable set of clauses [Robinson and Wos, 1969]. However, earlier resolution-based proof systems for logics with equality had the tendencies either to generate numerous useless resolvents or to generate clauses containing new literals. Our system has a number of advantages. For instance in our proof system in the newly generated clause only literals occur that already occurred in the original formula.

5.1 Commutation of proof systems

In the following we need a commutation property between proof systems in different settings. Therefore we now develop the desired commutation results for arbitrary proof systems.

Here we mean by a proof system a given set of statements and a set or rules to derive new statements from the given set. In the following we use F and G to denote a given set of statements, e.g. clauses. For a proof system S we use the notation $F \rightarrow_{\mathsf{S}} G$ for $G = F \cup \{C\}$, where C is a statement deduced from F by the proof system S.

For every relation $\rightarrow$ we write $\rightarrow^*$ for its reflexive transitive closure, i.e., we write $F \rightarrow^* G$ if $F_0, \ldots, F_n$ exist for $n \geq 0$ satisfying

$$F = F_0 \rightarrow F_1 \rightarrow \cdots \rightarrow F_n = G.$$

We write $F \sqsubseteq G$ if for each $C \in G$ there is a $D \in F$ such that $D \subseteq C$.

Definition 5.1 *A proof system S is $\sqsubseteq$-monotonic if it follows from $F \rightarrow_s^* F'$ that for each $G \sqsubseteq F$ there is a G' such that $G \rightarrow_s^* G'$ and $G' \sqsubseteq F'$.*

Definition 5.2 *Suppose S_1 and S_2 are proof systems and $F \rightarrow_{\mathsf{S}_1} F' \rightarrow_{\mathsf{S}_2} F''$ for some F, F', F''. We say that a proof system S_1 commutes over a proof system S_2 if for some finite n sets $G_1, \ldots, G_n, G$ exist such that*

1. *$F \rightarrow_{\mathsf{S}_2} G_i$, $1 \leq i \leq n$.*

2. $\bigcup_{i=1}^{n} G_i \rightarrow^*_{s_1} G$, *where* $G \sqsubseteq F''$.

Lemma 5.3 *Suppose for* $\sqsubseteq$*-monotonic proof systems* S_1 *and* S_2 *and some* F, F', F'' *the following holds.*

1. S_1 *commutes over* S_2.

2. $F \rightarrow^*_{\mathsf{S}_1} F' \rightarrow_{\mathsf{S}_2} F''$.

Then there are G', G'' *such that*

$$F \rightarrow^*_{\mathsf{S}_2} G' \rightarrow^*_{\mathsf{S}_1} G'',$$

where $G'' \sqsubseteq F''$.

Proof Suppose $F \equiv F_0 \rightarrow_{\mathsf{S}_1} F_1 \rightarrow_{s_1} \cdots \rightarrow_{\mathsf{S}_1} F_n \equiv F'$.

We give a proof by induction on n.

Base case. $n = 0$. The lemma holds trivially.

Inductive step. Assume that the lemma holds for $n-1$. By definition there are $G_1, \ldots, G_m$ for some finite m such that

1. $F_{n-1} \rightarrow_{\mathsf{S}_2} G_i$ for any $i \in \{1, \ldots, m\}$

2. $\bigcup_{i=1}^{m} G_i \rightarrow_{\mathsf{S}_1} G$, where $G \sqsubseteq F''$.

By the induction hypothesis, there are $G'_1, \ldots, G'_m, G''_1, \ldots, G''_m$, $1 \leq i \leq n$, such that for the $i \in \{1, \ldots, n\}$

$$F \rightarrow^*_{\mathsf{S}_2} G'_i \rightarrow^*_{\mathsf{S}_1} G''_i,$$

and

$$G''_i \sqsubseteq G_i.$$

By Definition 5.1 there is a $G'' \sqsubseteq G$ such that

$$\bigcup_{i=1}^{m} G''_i \rightarrow^*_{\mathsf{S}_1} G''.$$

We choose $G' = \bigcup_{i=1}^{m} G'_i$. Then $F \rightarrow^*_{\mathsf{S}_2} G' \rightarrow^*_{\mathsf{S}_1} G''$, where $G'' \sqsubseteq F''$. ⊠

Now we are ready to prove the theorem stating that any derivation consisting of any mix of S_1-steps and S_2-steps, can be rearranged in such a way that first only S_2-steps are applied and then only S_1-steps.

Theorem 5.4 *Let S_1 and S_2 be $\sqsubseteq$-monotonic proof systems, S_1 commute over S_2, S be the union of S_1 and S_2, and $F \rightarrow^*_{\mathsf{S}} F'$ for some F, F'. Then there are G, G' such that $F \rightarrow^*_{\mathsf{S}_2} G \rightarrow^*_{\mathsf{S}_1} G'$, and $G' \sqsubseteq F'$.*

Proof Suppose

$$F = F_0 \rightarrow_{\mathsf{S}} F_1 \rightarrow_{\mathsf{S}} \cdots \rightarrow_{\mathsf{S}} F_n = F'.$$

We give a proof by induction on n.

Base case. $n = 0$. The theorem trivially holds.

Inductive step. Suppose the theorem holds for $n-1$. Then by the induction hypothesis there are F_1', F_2' such that $F \rightarrow^*_{\mathsf{S}_2} F_1' \rightarrow^*_{\mathsf{S}_1} F_2'$, where $F_2' \sqsubseteq F_{n-1}$.

If $F_{n-1} \rightarrow_{\mathsf{S}_1} F_n$ then the theorem holds by Definition 5.1. If $F_{n-1} \rightarrow_{\mathsf{S}_2} F_n$ then the theorem holds by Lemma 5.3. ⊠

The next theorem is a further generalization of Theorem 5.4: now not only two proof systems are involved but an arbitrary number.

Theorem 5.5 (**Commutation property**) *Let S be the union of $\sqsubseteq$-monotonic proof systems $\mathsf{S}_1, \ldots, \mathsf{S}_n$. Let S_i commute over S_j for each $i > j$.*

If $F \rightarrow^*_{\mathsf{S}} F'$ then there are $F_1, \ldots, F_n$ such that $F \rightarrow^*_{\mathsf{S}_1} F_1 \rightarrow^*_{\mathsf{S}_2} \cdots \rightarrow^*_{\mathsf{S}_n} F_n$, and $F_n \sqsubseteq F'$.

Proof The proof is by induction on n.

Base case. $k = 1$. Trivially holds.

Inductive step. Let the theorem hold for $n-1$. Let S' be the union of systems $\mathsf{S}_2, \ldots, \mathsf{S}_n$. By Theorem 5.4 there are F_1, F_1' such that $F \rightarrow^*_{\mathsf{S}_1} F_1 \rightarrow^*_{\mathsf{S}'} F_1'$, $F_1' \sqsubseteq F'$. By induction hypothesis there are $F_2, \ldots, F_n$ such that $F_1 \rightarrow^*_{\mathsf{S}_2} \cdots \rightarrow^*_{\mathsf{S}_n} F_n$, where $F_n \sqsubseteq F_1'$. Then $F_1, \ldots, F_n$ satisfy the theorem. ⊠

5.1.1 A contradictory cycle

In Chapter 4 we defined a minimally unsatisfiable set of clauses. By a minimally unsatisfiable set of clauses we mean a set of clauses satisfying the following conditions: it is unsatisfiable and each of its subsets is satisfiable.

Lemma 5.6 *The set of clauses* $\{\{a_1 \approx a_2\}, \ldots, \{a_{n-1} \approx a_n\}, \{a_1 \not\approx a_n\}\}$ *is minimally unsatisfiable.*

Proof Obviously, this set of clauses is unsatisfiable. After removing an arbitrary clause this set will be satisfiable. Each subset of a satisfiable set of clauses is also satisfiable.

⊠

An important notion in this chapter is a *contradictory cycle.*

Definition 5.7 (**Contradictory cycle**)

- *A* contradictory cycle θ *is defined to be a set of literals of the shape*

$$a_1 \approx a_2, \ldots, a_{n-1} \approx a_n, a_1 \not\approx a_n,$$

  where $a_1, \ldots, a_n$ *are distinct constants.*

- *A contradictory cycle* θ *is called a contradictory cycle of a CNF* φ *if all literals of* θ *are contained in* $\mathsf{Lit}(\varphi)$.

When drawing a graph consisting of the constants from a CNF φ as nodes, and equalities from $\mathsf{Lit}(\varphi)$ as solid edges and inequalities from $\mathsf{Lit}(\varphi)$ as dashed edges, then a contradictory cycle of φ corresponds exactly to a cycle in this graph in which one edge is dashed and all other edges are solid. For a given CNF such a graph is easily made and such cycles are easily established by looking for solid paths from the one end of a dashed edge to the other end. In Theorems 5.16 and 5.18 we will see that unsatisfiability of a CNF is preserved by removing clauses containing literals that are not on a contradictory cycle.

5.2 Proof Systems

In this section we present the proof systems that play a role in this chapter.

5.2.1 Resolution and Paramodulation

Resolution for first order logic [Robinson, 1965] can be presented as follows (it is a generalization of Definition 3.1).

$$\text{Resolution:} \quad \frac{\{A_1\} \cup C, \{\neg A_2\} \cup D}{(C \cup D)\sigma}$$

where σ is a unifier of A_1 and A_2.

If a language contains not only predicates and functions but also equalities between two statements then one more rule is needed to avoid adding *equality axioms.* This rule is *paramodulation* [Robinson and Wos, 1969, Bachmair and Ganzinger, 1990].

It is well-known that resolution together with paramodulation forms a sound and complete refutation mechanism for first order logic with equality. This means that a formula is unsatisfiable if and only if the empty clause can be derived from it by means of paramodulation and resolution.

Paramodulation generates all versions of clauses, modulo "equal information". The paramodulation operation takes two clauses, where one clause must contain a positive equality literal, and derives a new clause, where the equality literal's argument must unify with a subterm in another clause. After unification of the argument and the subterm, the subterm is replaced by the other argument of the equality literal.

$$\text{Paramodulation:} \quad \frac{\{s \approx t\} \cup C, \{l[r]\} \cup D}{(\{l[t]\} \cup C \cup D)\sigma}$$

where s, t, r are terms, $l[r]$ is a (possibly negative) literal that contains a term r as subexpression, and σ is a unifier of r and s.

One of the problems with paramodulation is that it generates all equal variants of clauses without knowing which of the variants will be useful. As a result the combination of resolution and paramodulation is inefficient for applications. It is intrinsically exponential.

5.2.2 E(quality) R(esolution)

In this section we describe and prove a refutationally complete clausal inference system which is based on a combination of paramodulation and resolution.

The simplest version of paramodulation is substitution. If we know $a \approx b$ and an expression containing b then we can deduce the same expression where b is substituted with a. Hence, in the particular case of equality logic, paramodulation boils down to the following transitivity rule.

$$\text{Transitivity:} \quad \frac{\{a \approx b\} \cup C, \{b \approx c\} \cup D}{\{a \approx c\} \cup C \cup D}$$

For equality logic the resolution rule can be presented as follows.

$$\text{Resolution:} \quad \frac{\{a \approx b\} \cup C, \{a \not\approx b\} \cup D}{C \cup D}$$

We mentioned above that the combination of paramodulation and resolution is a sound and complete refutation mechanism. Taking into account that the transitivity rule is a special case of paramodulation, we obtain that a CNF φ is unsatisfiable if and only if the empty clause can be derived from φ by only using the transitivity rule and the resolution rule.

Instead of these two rules we now introduce one single rule, the equality resolution rule (ER), that is complete on its own as we will show in the following.

$$\text{ER:} \quad \frac{\{x_1 \approx x_2\} \cup C_1, \ldots, \{x_{n-1} \approx x_n\} \cup C_{n-1}, \{x_1 \not\approx x_n\} \cup C_n}{C_1 \cup \cdots \cup C_n}$$

Clearly, for every contradictory cycle $\{a_1 \approx a_2, \ldots, a_{n-1} \approx a_n, a_1 \not\approx a_n\}$ we have a corresponding instance of the ER rule.

5.2.3 Soundness and completeness of the ER Rule

In this section we prove soundness and completeness of the ER rule. The proof is based on the commutation property of proof systems presented in 5.1.

Suppose $\varphi' = \varphi \cup \{C\}$. Then we will use the following notations.

- $\varphi \rightarrow_t \varphi'$ if C is derived from φ by the transitivity rule,
- $\varphi \rightarrow_r \varphi'$ if C is derived from φ by the resolution rule,
- $\varphi \rightarrow_{er}^{\theta} \varphi'$ if C is derived by the ER rule using the contradictory cycle θ,
- $\varphi \rightarrow_{er} \varphi'$ if C is derived by the ER rule,
- $\varphi \rightarrow_{\theta} \varphi'$ if the ER rule is applied for a fixed contradictory cycle θ.

We now come to the proof of soundness of the ER rule.

Theorem 5.8 (**Soundness**) *Let $\varphi \rightarrow_{er} \psi$. Then φ is satisfiable if and only if ψ is satisfiable.*

Proof Let

$$\psi = \varphi \cup \{C_1 \cup \cdots \cup C_n\},$$

where $C_1 \cup \{x_1 \approx x_2\}, \ldots, C_{n-1} \cup \{x_{n-1} \approx x_n\}, C_n \cup \{x_1 \not\approx x_n\} \in \varphi$.

($\Rightarrow$) Let ψ be satisfiable. Then φ is satisfiable as it is a subset of a satisfiable set of clauses.

($\Leftarrow$) Let φ be satisfiable. Suppose $\mathcal{D}$ is a structure such that $[\![\varphi]\!]_{\mathcal{D}} = \mathsf{true}$.

The set $\theta = \{\{x_1 \approx x_2\}, \ldots, \{x_{n-1} \approx x_n\}, \{x_1 \not\approx x_n\}\}$ is minimally unsatisfiable by Lemma 5.6. Then there is an element l in θ such that $[\![l]\!]_{\mathcal{D}} = \mathsf{false}$. Hence, $[\![C_i]\!]_{\mathcal{D}} = \mathsf{true}$ for some $i \in \{1, \ldots, n\}$. We conclude $[\![\psi]\!]_{\mathcal{D}} = \mathsf{true}$. ⊠

In order to prove completeness we will use a commutation property and Theorem 5.4. We start with proving the commutation property for the transitivity proof system and the ER proof system.

Lemma 5.9 *The transitivity rule commutes over the ER rule.*

Proof It is easily observed that the transitivity rule and the ER rule are $\sqsubseteq$-monotonic.

We assume the following.

- $\varphi \rightarrow_t \varphi' \rightarrow_{er} \varphi''$, where $\varphi' = \varphi \cup \{C\}$, $\varphi'' = \varphi' \cup \{D\}$.
- $C = C_1 \cup C_2 \cup \{x \approx z\}$, where $C_1 \cup \{x \approx y\}, C_2 \cup \{y \approx z\} \in \varphi$.
- $\{l_1, \ldots, l_n\}$ is a contradictory cycle.
- $D = \bigcup_{i=1}^{n} D_i$, where $D_i \cup \{l_i\} \in \varphi'$, $1 \le i \le n$.

We have to check that the transitivity and the ER proof systems satisfy Definition 5.2.

Suppose $D_i \not\equiv C$, $1 \le i \le n$. We choose

$$G_1 = \varphi \cup \{D\} \text{ and } G = G_1 \cup \{C\}.$$

In this case $G = \varphi''$. One can see that this choice satisfies Definition 5.2.

Suppose $D_i \equiv C$ for some $i \in \{1, \ldots, n\}$.

W.l.o.g. we can assume that $i = 1$. We do the following case analysis.

1. $l_1 \equiv (x \approx z)$.

   We have $D = C_1 \cup C_2 \cup \bigcup_{i=2}^{n} D_i$.

   Literals $x = y, y = z, l_2, \ldots, l_n$ form a contradictory cycle. Then D can be derived from φ by ER rule. Choose

   $$G_1 = \varphi \cup \{D\} \text{ and } G = G_1 \cup \{C\}.$$

   In this case $G = \varphi''$.

2. $l_1 \in C_1 \cup C_2$.

   Let $C_1' = C_1 \cup \{x \approx y\}$, $C_2' = C_2 \cup \{y \approx z\}$, $D_j = C_j' \backslash \{l_1\} \cup \bigcup_{i=2}^{n} D_i$.

   If $l_1 \in C_1$ or $l_1 \in C_2$ then D_1 or D_2 can be derived from φ by the ER rule. Assume that

   $$G_i = \varphi \cup \{D_i\} \text{ for } i \in \{1, 2\}.$$

   If $l_1 \not\equiv (x \approx y)$ and $l_1 \not\equiv (y \approx z)$ then D can be derived from D_1 and D_2 by the transitivity rule. If $l_1 \not\equiv (x \approx y)$ then $D_1 \subseteq D$. If $l_1 \not\equiv (y \approx z)$ then $D_2 \subseteq D$.

   Hence, if $l_1 \in C_1$ and $l_1 \in C_2$ then there are G_1, G_2, G satisfying Definition 5.2, and if either $l_1 \in C_1$ or $l_1 \in C_1$ then there are G_1, G or G_2, G satisfying Definition 5.2.

$\boxtimes$

Theorem 5.10 (**Completeness of the ER rule**) *The set of clauses φ is unsatisfiable if and only if the empty clause can be derived from φ by the ER rule.*

Proof

($\Rightarrow$) Assume that the empty clause can be derived from φ by the ER rule. Then by Theorem 9.22 the original set of clauses is unsatisfiable.

($\Leftarrow$) Assume that the original set φ of clauses is unsatisfiable. According to a well-known paramodulation result, the empty clause can be derived from φ by transitivity and resolution. Since resolution is a particular case of the ER rule we conclude that $\varphi \to_s \psi$ for some ψ containing $\bot$, and S is the combination of the transitivity rule and the ER rule. By Lemma 5.9 and Theorem 5.4, there are φ', φ'' such that

$$\varphi \to_{er}^{*} \varphi' \to_{t}^{*} \varphi'' \text{ and } \varphi'' \sqsubseteq \psi.$$

```
Procedure E-DP($\varphi$);
    begin
        $\Theta$ := ContrCycle($\varphi$);
        while ($\Theta \neq \emptyset$) do
        begin
            choose $\theta \in \Theta$;
            $\Theta := \Theta \backslash \{\theta\}$;
            if $\perp \in \varphi$ return(UNSAT);
            $\varphi := \varphi \cup$ ER($\varphi, \theta$);
        end
        return(SAT);
    end
```

Figure 5.1 The E-DP procedure

Since $\perp \in \psi$ then $\perp \in \varphi''$. By the transitivity rule $\perp$ cannot be derived. Hence, $\perp \in \varphi'$. ⊠

5.3 The E-DP procedure

In this section we describe the E-DP procedure which is an extension of the propositional DP procedure. Based on this we present in the following a modified version which is more efficient.

Given a nonempty CNF φ containing nonempty clauses, the E-DP procedure forms the set of all contradictory cycles Θ contained in φ and then repeats the following steps.

- Choose a contradictory cycle $\theta \in \Theta$ and remove θ from Θ.
- Add all possible clauses derived from φ by the ER rule over θ.

We give a precise version of the procedure.

In this procedure the function `ContrCycle`(φ) forms the set of all possible contradictory cycles.

The function `ER`(φ, θ) forms the set of clauses derived from φ by all possible θ-steps.

The procedure ends when either the empty clause is derived or no contradictory cycles are left. If the empty clause is derived the output of the procedure is "unsatisfiable". If the empty clause is not derived during the procedure the output is "satisfiable".

5.3.1 Soundness and completeness of the procedure

In this section we prove the completeness of the DP procedure. Let $\theta_1, \ldots, \theta_n$ be contradictory cycles of an unsatisfiable CNF φ_0. Based on the completeness of the ER rule we show that there is a finite sequence $\varphi_1, \ldots, \varphi_n$ such that for any $i \in \{1, \ldots, n\}$ φ_i consists of all clauses contained in φ_{i-1} and clauses derived from φ_{i-1} in one θ_i-step, and φ_n contains the empty clause.

Lemma 5.11 *Let each θ_i-step be a proof system s_i, where $i \in \{1, 2\}$. Then s_1 commutes over s_2.*

Proof

We prove that s_1 and s_2 satisfy Definition 5.2. Assume

$$\varphi \rightarrow_{\theta_1} \varphi_1 \rightarrow_{\theta_2} \varphi_2.$$

Suppose

$$\varphi_1 = \varphi \cup \{C\},$$

and

$$\varphi_2 = \varphi_1 \cup \{D\},$$

where $D = \bigcup\limits_{i=1}^{m} D_i \backslash \{l_i\}$ for some $D_1, \ldots, D_n \in \varphi_1$.

Then one of the following holds.

1. $D_i \not\equiv C$, for every $i \in \{1, \ldots, m\}$.

   Choose

   $$G_1 = \varphi \cup \{D\} \text{ and } G = G_1 \cup \{C\}.$$

   The lemma holds.

2. $D_i \equiv C$ for some $i \in \{1, \ldots, m\}$.

   W.l.o.g. we can assume that $i = 1$.

   Let

   $$C = \bigcup_{i=1}^{r} C_i \backslash \{l'_i\},$$

for some $C_1, \ldots, C_r \in \varphi$.

W.l.o.g we can assume that $l_1 \in C_{i_1}$ for some $i \in \{1, \ldots, k\}$.

We define

(a) For every $i \in \{1, \ldots, k\}$,

$$G_i = F \cup \{C_i^*\},$$

where

$$C_i^* = C_i \backslash \{l_1\} \cup \bigcup_{j=2}^{m} D_j \backslash \{l_j\}.$$

(b)

$$G = \bigcup_{i=1}^{k} G_i \cup \{C\} \cup \{D^*\},$$

where

$$D^* = \bigcup_{i=1}^{k} C_i^* \backslash \{l_i^{'}\} \cup \bigcup_{i=k+1}^{r} C_i \backslash \{l_i^{'}\}.$$

We will show that $D^* \subseteq D$.

$$\begin{aligned}
D^* &= \bigcup_{i=1}^{k} C_i^* \backslash \{l_i^{'}\} \cup \bigcup_{i=k+1}^{r} C_i \backslash \{l_i^{'}\} \\
&= \bigcup_{i=1}^{k} (C_i \backslash \{l_1\} \cup \bigcup_{j=2}^{m} D_j \backslash \{l_j\}) \backslash \{l_i^{'}\} \cup \bigcup_{i=k+1}^{r} C_i \backslash \{l_i^{'}\} \\
&= \bigcup_{i=1}^{k} (C_i \backslash \{l_i^{'}\}) \backslash \{l_1\} \cup \bigcup_{i=k+1}^{r} C_i \backslash \{l_i^{'}\} \cup \bigcup_{j=2}^{m} D_j \backslash \{l_j\} \\
&= C \backslash \{l_1\} \cup \bigcup_{j=2}^{m} D_j \backslash \{l_j\} \\
&= \bigcup_{j=1}^{m} D_j \backslash \{l_j\} \\
&\subseteq D.
\end{aligned}$$

Then for each $i \in \{1, \ldots, k\}$ G_i consists of clauses from φ and a clause derived from φ by a θ_2-step, and C and D^* can be derived from $\bigcup_{i=1}^{k} G_i$ by a θ_1-step. It can be easily checked that $G \sqsubseteq \varphi_2$.

⊠

Theorem 5.12 *Let for $\varphi, \psi \in \mathsf{Cnf}$,*

$$\varphi \rightarrow_{er}^{*} \psi.$$

Let $\{\theta_1, \ldots, \theta_n\}$ be the set of all contradictory cycles in φ. Then there are $\varphi_1, \ldots, \varphi_m$ such that

$$\varphi \rightarrow_{\theta_1}^{*} \varphi_1 \rightarrow_{\theta_2}^{*} \cdots \rightarrow_{\theta_m}^{*} \varphi_m,$$

where $\varphi_m \sqsubseteq \psi$.

Proof The theorem follows from Theorem 5.5.

⊠

Theorem 5.13 *Let for $\varphi, \psi, \psi' \in \mathsf{Cnf}$ and a contradictory cycle θ,*

$$\psi = \varphi \cup ER(\varphi, \theta) \text{ and } \psi \rightarrow_{\theta} \psi'.$$

Then $\psi \sqsubseteq \psi'$.

Proof If each $C \in \psi \backslash \varphi$ does not contain a literal from θ then the theorem trivially holds.

We show that for each $C \cup \{l\} \in \psi \backslash \varphi$, where $l \in \theta$, there is a $D \cup \{l\} \in \varphi$ such that $D \subseteq C$.

Suppose

$$\theta = \{l_1, \ldots, l_n\}.$$

$C \cup \{l_i\} \in \psi \backslash \varphi$ for some $i \in \{1, \ldots, n\}$; $C \cup \{l_i\} = D_1 \cup \cdots \cup D_n$, where $D_1 \cup \{l_1\}, \ldots, D_n \cup \{l_n\} \in \varphi$. Then $D_i \subseteq C$.

Let $\psi' = \psi \cup \{C\}$. We shall show that there is a $D \in \psi$ such that $D \subseteq C$.

Let $C = C_1 \cup \cdots \cup C_n$, where $C_1 \cup \{l_1\}, \ldots, C_n \cup \{l_n\} \in G$; $C_1 \cup \{l_1\}, \ldots, C_r \cup \{l_r\} \in \psi \backslash \varphi$ for some $1 \leq r \leq n$. As it was shown above for any $i \in \{1, \ldots, r\}$ there is a $D_i \cup \{l_i\} \in \varphi$ such that $D_i \subseteq C_i$. Then $D = D_1 \cup \cdots \cup D_r \cup C_{r+1} \cup \cdots \cup C_n$ can be derived by a θ-step, $D \subseteq C$ and $D_1, \ldots, D_r, C_{r+1}, \ldots, C_n \in \varphi$. So $D \in G$. ⊠

Theorem 5.14 (Soundness and completeness of the E-DP procedure) *Let φ be a CNF. Then φ is unsatisfiable if and only if the output of the E-DP procedure is the empty clause.*

Proof

If there is a derivation of the empty clause by the ER rule then φ is unsatisfiable by Theorem 9.22.

If φ is unsatisfiable then by Theorem 8.4 there is a derivation of the empty clause from φ by the ER rule.

Let $\{\theta_1, \dots, \theta_n\}$ be the set of all contradictory cycles in φ. Then by Theorem 5.12 there are $\varphi_1, \dots, \varphi_m$ such that $\varphi \rightarrow^*_{\theta_1} \varphi_1 \rightarrow^*_{\theta_2} \cdots \rightarrow^*_{\theta_m} \varphi_m$ and $\bot \in \varphi_m$. By Theorem 5.13 $\varphi_i = \varphi_{i-1} \cup ER(\varphi_{i-1}, \theta_i)$ for any $i \in \{1, \dots, m\}$. It implies that the empty clause can be derived by the E-SAT procedure. ⊠

5.4 Removing redundant clauses

In the following we use the fact that φ is satisfiable if and only if it is satisfiable on the set of natural numbers $\mathbf{N}$. So w.l.o.g. we may and shall assume that $D \equiv \mathbf{N}$.

5.4.1 Removing disequalities

Definition 5.15 *Let φ be a CNF. The relation $=_\varphi$ on $\mathsf{Const}(\varphi)$ is defined by $x =_\varphi y$ if and only if $x \approx y \in \mathsf{Lit}(\varphi)$.*

The relation $\sim_\varphi$ is defined to be the equivalence relation generated by $=_\varphi$, i.e., the reflexive, symmetric, transitive closure of the relation $=_\varphi$. By E^x_φ we denote the equivalence class of x with respect to $\sim_\varphi$.

Theorem 5.16 *Let $x \nsim_\varphi y$. Then φ is satisfiable iff $\varphi|_{x \not\approx y}$ is satisfiable.*

Proof (⇒) Suppose φ is satisfiable. Then $\varphi|_{x \not\approx y}$ is satisfiable as a subset of satisfiable set of clauses.

(⇒) Suppose $\varphi|_{x \not\approx y}$ is satisfiable, and $[\![\varphi]\!]_{\mathcal{D}} = \mathsf{true}$ for a structure $\mathcal{D}$. We define a new structure $\mathcal{D}'$ in such a way that it preserves satisfiability of $\varphi|_{x \not\approx y}$ but also $[\![\varphi]\!]_{\mathcal{D}'} = \mathsf{true}$.

Choose a number N satisfying

$$N > [\![x']\!]_{\mathcal{D}} - [\![y']\!]_{\mathcal{D}}$$

for all $x', y' \in \mathsf{Const}(\varphi)$.

We define $D' \equiv \mathbf{N}$, and

$$z_{\mathcal{D}'} = \begin{cases} z_{\mathcal{D}} + N, \text{ if } z \in E^x_\varphi, \\ z_{\mathcal{D}}, \text{ in other cases.} \end{cases} \tag{5.1}$$

Since $[\![\varphi|_{x \not\approx y}]\!]_{\mathcal{D}}$ then for every $C \in \varphi|_{x \not\approx y}$ there is an $l \in C$ such that $[\![l]\!]_{\mathcal{D}} = true$. We check that $[\![l]\!]_{\mathcal{D}'} = \mathsf{true}$.

1. $l \;\equiv\; (x' \approx y')$, where $x'', y'' \in \mathsf{Const}(\varphi)$.

   If $x' \not\sim_\varphi x$ then $[\![x'']\!]_{\mathcal{D}'} = [\![x'']\!]_{\mathcal{D}} = [\![y'']\!]_{\mathcal{D}} = [\![y'']\!]_{\mathcal{D}'}$.

   If $x'' \sim_\varphi x$ then $[\![x'']\!]_{\mathcal{D}'} = [\![x'']\!]_{\mathcal{D}} + N = [\![y'']\!]_{\mathcal{D}} + N = [\![y'']\!]_{\mathcal{D}'}$.

   In both cases $[\![l]\!]_{\mathcal{D}'} = \mathsf{true}$.

2. $l \;\equiv\; (x'' \not\approx y'')$, where $x'', y'' \in \mathsf{Const}(\varphi)$.

   (a) $x'' \sim_\varphi y''$.
   If $x'' \sim_\varphi x$ then $[\![x'']\!]_{\mathcal{D}'} = [\![x'']\!]_{\mathcal{D}} + N \neq [\![y'']\!]_{\mathcal{D}} + N = [\![y'']\!]_{\mathcal{D}'}$.
   If $x'' \not\sim_\varphi x$ then $[\![x'']\!]_{\mathcal{D}'} = [\![x'']\!]_{\mathcal{D}} \neq [\![y'']\!]_{\mathcal{D}} = [\![y'']\!]_{\mathcal{D}}$.

   (b) $x'' \not\sim_\varphi y''$.
   Using that $N > A(x') - A(y')$ for all $x', y' \in \mathsf{Const}(\varphi)$ we have:
   If $x'' \sim_\varphi x$ and $y'' \not\sim_\varphi x$ then $[\![x'']\!]_{\mathcal{D}'} = [\![x'']\!]_{\mathcal{D}} + N > [\![y'']\!]_{\mathcal{D}} = [\![y'']\!]_{\mathcal{D}'}$.
   If $x'' \not\sim_\varphi x$ and $y'' \sim_\varphi x$ then $[\![x'']\!]_{\mathcal{D}'} = [\![x'']\!]_{\mathcal{D}} < [\![y'']\!]_{\mathcal{D}} + N = [\![y'']\!]_{\mathcal{D}'}$.
   If $x'' \not\sim_\varphi x$ and $y'' \not\sim_\varphi x$ then $[\![x'']\!]_{\mathcal{D}'} = [\![x'']\!]_{\mathcal{D}} \neq [\![y'']\!]_{\mathcal{D}} = [\![y'']\!]_{\mathcal{D}'}$.

   In all cases $[\![l]\!]_{\mathcal{D}'} = \mathsf{true}$.

Since $N > [\![x']\!]_{\mathcal{D}} - [\![y']\!]_{\mathcal{D}}$ for all $x', y' \in \mathsf{Const}(\varphi)$ and $y \not\sim_F x$ we have $[\![x]\!]_{\mathcal{D}'} = [\![x]\!]_{\mathcal{D}} + N > [\![y]\!]_{\mathcal{D}} = [\![y]\!]_{\mathcal{D}}$. So $[\![x \not\approx y]\!]_{\mathcal{D}'} = \mathsf{true}$. ⊠

Example 5.17 As an example we have taken the formula from [Pnueli et al., 1999] raised during the verification of translators (compilers, code generators). After abstracting concrete functions, performing the Ackermann reduction the following CNF is obtained:

$$\begin{aligned}\varphi_1 &= \{\{x_1 \not\approx x_2, x_3 \not\approx x_4, y_1 \approx y_2\}, \{y_1 \not\approx y_3, y_2 \not\approx y_4, z_1 \approx z_2\},\\ &\quad \{y_1 \approx y_3\}, \{y_2 \approx y_4\}, \{z_1 \approx z_3\}, \{z_2 \not\approx z_3\}\}.\end{aligned}$$

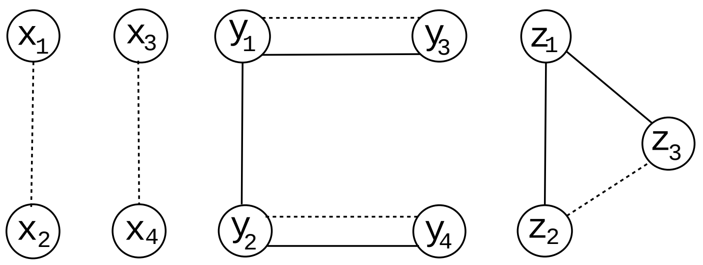

Figure 5.2 The graph corresponding to φ_1

As sketched before we may draw a graph consisting of the constants from φ_1 as nodes, and equalities from $\mathsf{Lit}(\varphi_1)$ as solid edges and inequalities from $\mathsf{Lit}(\varphi_1)$ as dashed edges. The result is given in Figure 5.2.

In graph notation, $x \sim y$ means that there is a path from node x to node y purely consisting of solid edges. Hence, in this example we clearly have $x_1 \nsim x_2$. Then by Theorem 5.16 we may remove all clauses containing the literal $x_1 \not\approx x_2$ without changing satisfiability behavior. Therefore, we may remove the first clause, resulting in

$$\varphi_2 = \{\{y_1 \not\approx y_3, y_2 \not\approx y_4, z_1 \approx z_2\}, \{y_1 \approx y_3\}, \{y_2 \approx y_4\}, \{z_1 \approx z_3\}, \{z_2 \not\approx z_3\}\}$$

We see that constants x_1, x_2, x_3 and x_4 do not occur in φ_2, and the resulting graph for φ_2 is given in Figure 5.3.

5.4.2 Removing equalities

Theorem 5.18 *Let φ be a CNF and let $x \approx y \in \mathsf{Lit}(\varphi)$ not be contained in any contradictory cycle of φ. Then φ is satisfiable if and only if $\varphi|_{x \approx y}$ is satisfiable.*

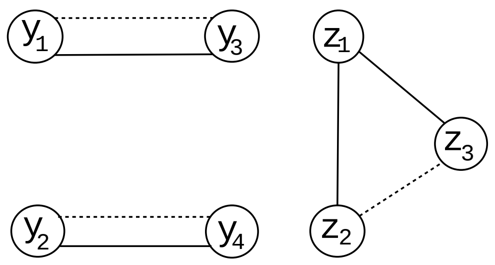

Figure 5.3 The graph corresponding to φ_2

Proof ($\Rightarrow$) Let φ be satisfiable. Then $\varphi|_{x\approx y}$ is satisfiable as a subset of a satisfiable set of clauses.

($\Leftarrow$) Conversely assume that $\varphi|_{x\approx y}$ is satisfiable. Then we may assume that $[\![\varphi|_{x\approx y}]\!]_{\mathcal{D}} = \mathsf{true}$ for some structure $\mathcal{D}$. We show that there is a structure $\mathcal{D}'$ such that $[\![\varphi]\!]_{\mathcal{D}'} = \mathsf{true}$.

We define the set P_x as follows:

- $x \in P_x$.
- $z \in P_x$ if $(z \approx z') \in \mathsf{Lit}(\varphi)$ and $[\![z]\!]_{\mathcal{D}} = [\![z']\!]_{\mathcal{D}}$ for some $z' \in P_x$.

Choose

$$N > \max_{x \in \mathsf{Const}(\varphi)} [\![x]\!]_{\mathcal{D}}.$$

Then for each $z \in \mathsf{Const}(\varphi)$ we may define $z_{\mathcal{D}'}$ as follows:

$$z_{\mathcal{D}'} = \begin{cases} N, \text{ if } z \in E^x_\varphi, \\ z_{\mathcal{D}}, \text{ in other cases.} \end{cases} \tag{5.2}$$

We show that for each $l \in \mathsf{Lit}(\varphi|_{x\approx y})$, if $[\![l]\!]_{\mathcal{D}} = \mathsf{true}$ then $[\![l]\!]_{\mathcal{D}'} = \mathsf{true}$.

Let us take an arbitrary $l \in \mathsf{Lit}(\varphi|_{x\approx y})$. Then one of the following holds:

1. $l \equiv (x' \approx y')$, where $x', y' \in \mathsf{Const}$.
   Then either
   $x', y' \in P_x \cup P_y$ and $[\![x']\!]_{\mathcal{D}'} = N = [\![y']\!]_{\mathcal{D}'}$

or

$x', y' \in \mathsf{Const}(\varphi) \backslash (P_x \cup P_y)$ and $[\![x']\!]_{\mathcal{D}'} = [\![x']\!]_{\mathcal{D}} = [\![y']\!]_{\mathcal{D}} = [\![y']\!]_{\mathcal{D}'}$.

2. $l \equiv (x' \not\approx y')$, where $x', y' \in \mathsf{Const}$.

   We consider the case when $x', y' \in P_x \cup P_y$.

   If $x' \in P_x$ and $y' \in P_y$ or vise versa then $x \approx y$ is contained in a contradictory cycle of φ, contradicting the assumption of the theorem.

   If $x', y' \in P_x$ then $[\![x']\!]_{\mathcal{D}} = [\![y']\!]_{\mathcal{D}}$ and $[\![l]\!]_{\mathcal{D}} \neq \mathsf{true}$. In this case we also have a contradiction. By symmetry in case when $x', y' \in P_y$ we have a contradiction.

   Then one of the following holds:

   (a) $x' \notin P_x \cup P_y$, $y' \notin P_x \cup P_y$. Then $[\![x']\!]_{\mathcal{D}'} = [\![x']\!]_{\mathcal{D}} \neq [\![y']\!]_{\mathcal{D}} = [\![y']\!]_{\mathcal{D}'}$.

   (b) $x' \in P_x \cup P_y$, $y' \notin P_x \cup P_y$. Then $[\![x']\!]_{\mathcal{D}'} = N > [\![y']\!]_{\mathcal{D}} = [\![y']\!]_{\mathcal{D}'}$ as $N > \max\limits_{x \in \mathsf{Const}(\varphi)} [\![x]\!]_{\mathcal{D}}$.

   (c) $x' \notin P_x \cup P_y$, $y' \in P_x \cup P_y$. Then $[\![y']\!]_{\mathcal{D}'} = N > [\![x']\!]_{\mathcal{D}} = [\![x']\!]_{\mathcal{D}'}$ as $N > \max\limits_{x \in \mathsf{Const}(\varphi)} [\![x]\!]_{\mathcal{D}}$.

   In all cases $[\![l]\!]_{\mathcal{D}'} = \mathsf{true}$.

Hence, $[\![\varphi|_{x \approx y}]\!]_{\mathcal{D}'} = \mathsf{true}$.

Since by definition of $\mathcal{D}'$, $[\![x \approx y]\!]_{\mathcal{D}'} = \mathsf{true}$, we conclude $[\![\varphi]\!]_{\mathcal{D}'} = \mathsf{true}$. ⊠

In the graph representation Theorem 5.18 states that every clause may be removed containing an equality corresponding to a solid edge for which no path between the end points of the edge exists containing exactly one dashed edge.

Example 5.19 Consider the formula φ_1 from Example 5.17. By applying Theorem 5.18 we may remove all clauses containing the equality $y_1 \approx y_2$. As a result we again obtain φ_2, of which the graph is given in Figure 5.3. Note that clauses containing the equality $z_1 \approx z_2$ may not be removed.

The fact that by applying Theorem 5.16 and Theorem 5.18 in this particular example the same clause is removed, is a coincidence; in more complicated examples one sees that the combination of both theorems is more powerful than applying only one of them.

```
Procedure E-DP($\varphi$);
    begin
        $\Theta := \emptyset$;
        while ($\varphi \neq \emptyset$) do
        begin
            RemoveRedundant ($\varphi$);
            $\theta :=$ ShortestContrCycle ($\varphi, \Theta$);
            $\Theta := \Theta \cup \{\theta\}$;
            if $\bot \in \varphi$ return(UNSAT);
            $\varphi := \varphi \cup$ ER($\varphi, \theta$);
            end
            return(SAT);
        end
```

Figure 5.4 The optimized E-DP procedure

5.5 The optimized E-DP procedure

The search space of the saturation-based procedures can grow very rapidly. The procedure becomes more efficient when we have criteria to remove redundant clauses from the search space. In the optimized procedure we use *subsumption* introduced by Robinson [Robinson, 1965] for general resolution. We obtain additional criteria to remove redundant clauses by means of the theorems proved in section 4.

The number of contradictory cycles in a formula can be exponential in the size of a formula. This is a potential source of inefficiency. To avoid this problem an optimized procedure does not collect the set of all contradictory cycles as a first step.

The optimized procedure repeats the following steps.

- Choose a contradictory cycle θ of the shortest length not contained in Θ and add it to Θ.
- Remove redundant clauses.
- Add all possible clauses derived from φ by the ER rule over θ.
- Remove clauses containing literals which are not in contradictory cycles not contained in Θ.

The procedure `ShortestContrCycle` (φ, Θ) chooses a contradictory cycle of the shortest length not contained in Θ. The procedure `RemoveRedundant`(φ) repeatedly removes clauses from φ by the following rules:

- if a clause C is a subclause of a clause C' then C' is removed (subsumption);
- remove a clause containing $x \not\approx y$ for which $x \not\sim y$, see Theorem 5.16;
- remove a clause containing $x \approx y$ for which $x \approx y$ is not contained in any contradictory cycle, see Theorem 5.18;

until nothing can be removed any more. The function `ER`(φ, θ) forms the set of clauses derived from F by all possible θ-steps.

The procedure ends if either the set of clauses is empty or the empty clause is derived. If the empty clause is derived then the output is "unsatisfiable". If the set of clauses is empty then the output is "satisfiable".

Theorem 5.20 (**Soundness and completeness**) *Let φ be a CNF. Then φ is unsatisfiable if and only if the output of the optimized E-DP procedure is the empty clause.*

Proof If there is a derivation of the empty clause by the ER rule then φ is unsatisfiable by Theorem 9.22. If φ is unsatisfiable then the empty clause can be derived by optimized procedure by Theorem 5.16, Theorem 5.18 and Theorem 5.14. ⊠

Example 5.21 Consider the formula φ_1 from Example 5.17. Removing redundant clauses yields the CNF φ_2:

$$\varphi_2 = \{\{y_1 \not\approx y_3, y_2 \not\approx y_4, z_1 \approx z_2\}, \{y_1 \approx y_3\}, \{y_2 \approx y_4\}, \{z_1 \approx z_3\}, \{z_2 \not\approx z_3\}\}.$$

The contradictory cycles contained in φ_2 are the following: $\theta_1 = \{y_1 \approx y_3, y_1 \not\approx y_3\}$, $\theta_2 = \{y_2 \approx y_4, y_2 \not\approx y_4\}$, $\theta_3 = \{z_1 \approx z_2, z_1 \approx z_3, z_2 \not\approx z_3\}$.

(1) $y_1 \not\approx y_3 \vee y_2 \not\approx y_4 \vee z_1 \approx z_2$	
(2) $y_1 \approx y_3$	
(3) $y_2 \approx y_4$	
(4) $z_1 \approx z_3$	
(5) $z_2 \not\approx z_3$	
(6) $y_2 \not\approx y_4 \vee z_1 \approx z_2$	(1,2)
(7) $z_1 \approx z_2$	(3,6)
(8) $\perp$	(4,5,7)

The empty clause is derived by the optimized procedure.

5.6 Example

As an example we consider a formula that is related to the pigeon hole formula in proposition calculus. Just like the pigeon hole formula our formula is parameterized by a number n. It is easily seen to be contradictory by a meta argument, and its shape is the conjunction of two subformulae. In our formula there are $n+1$ variables $x_1, \ldots, x_n, y$. The first subformula states that all values of $x_1, \ldots, x_n$ are different. The second subformula states that the value of y occurs in every subset of size $n-1$ of $\{x_1, \ldots, x_n\}$, hence it will occur at least twice in $\{x_1, \ldots, x_n\}$, contradicting the property of the first subformula. Hence the total formula

$$\varphi_n \quad \equiv \quad \bigwedge_{1 \leq i < j \leq n} x_i \not\approx x_j \;\wedge\; \bigwedge_{j=1}^{n} (\bigvee_{i \in \{1,\ldots,n\}, i \neq j} x_i \approx y)$$

is unsatisfiable. It is easy to see that φ_n is minimally unsatisfiable, hence in any proof of unsatisfiability all $\frac{n(n+1)}{2}$ clauses have to be used. The goal now is to prove unsatisfiability of φ_n automatically.

We applied the bit vector encoding to this formula, i.e., in this formula every $z \approx w$ is replaced by $\bigwedge_i (z_i \leftrightarrow w_i)$ for i running from 1 to $\lceil \log(n+1) \rceil$ and then a standard SAT approach is applied for the resulting propositional formula. It turned out that both for a BDD-based approach and a resolution based approach this is a hard job. For $n = 50$ or even lower a combinatory explosion comes up.

However, by applying the approach introduced in this paper proving unsatisfiability of φ_n can be done polynomially in n. It turns out that all contradictory cycles in φ_n are of length 3 and are of the shape $\theta_{ij} = \{x_i \approx y, x_j \approx y, x_i \not\approx x_j\}$ for $1 \leq i < j \leq n$; the total number of these contradictory cycles is $\frac{n(n-1)}{2}$.

Now we study the behavior of our procedure consecutively proceeding all these contradictory cycles. Write C_j for the clause $\bigvee_{i \in \{1,\ldots,n\}, i \neq j} x_i \approx y$ for $j = 1, \ldots, n$, and write C_{jn} for the clause obtained from C_j by removing $x_n \approx y$, for $j = 1, \ldots, n-1$. As a first contradictory cycle choose $\theta_{1,n}$. Then by applying a $\theta_{1,n}$-step on C_1, C_n and $x_1 \not\approx x_n$ we obtain the new clause C_{1n}. Another number of $\theta_{1,n}$-steps is possible, but each of them yields a clause in which C_{1n} is contained, hence it will be removed. Also C_1 and C_n are supersets of C_{1n} and will be removed. So after treating this first contradictory cycle apart from the inequalities only the following $n-1$ clauses remain: $C_2, \ldots, C_{n-1}, C_{1n}$. As a second contradictory cycle choose $\theta_{2,n}$. Applying a corresponding step on C_2, C_{1n} and $x_2 \not\approx x_n$ yields the new clause C_{2n}. Since this is a subclause of all other clauses generated by $\theta_{2,n}$-steps, and also of C_2, after treating this second contradictory cycle apart from the inequalities only the following $n-1$ clauses remain: $C_3, \ldots, C_{n-1}, C_{1n}, C_{2n}$.

This pattern continues after choosing the $n-1$-th contradictory cycle $\theta_{n-1,n}$, apart from the inequalities only the following $n-1$ clauses remain:

$C_{1n}, C_{2n}, \ldots, C_{n-1,n}$. Since now no equality occurs any more involving the variable x_n, there is no contradictory cycle any more containing the inequalities $x_i \not\approx x_n$ for $i = 1, \ldots, n-1$. It turns out that the remaining E-CNF is exactly φ_{n-1}. Continuing with consecutively choosing $\theta_{1,n-1}$, $\theta_{2,n-1}, \ldots$, after $n-2$ steps the remaining E-CNF is exactly φ_{n-2}. This goes on until the remaining E-CNF is exactly φ_2 consisting of the three unit clauses $x_1 \approx y$, $x_2 \approx y$, $x_1 \not\approx x_2$ from which the empty clause is derived in one single θ_{12}-step.

We conclude that all $\frac{n(n-1)}{2}$ contradictory cycles were processed before the empty clause was derived. Surprisingly, after removing redundant clauses, in intermediate steps the total number of clauses was never greater than the original number of clauses.

Chapter 6

A generalization of the DPLL procedure

“A system is nothing more than the subordination of all aspects of the universe to any one such aspect”

[T.H. Huxley]

In this chapter we present GDPLL, a generalization of the DPLL procedure. It solves the satisfiability problem for decidable fragments of quantifier-free first-order logic. Sufficient properties are identified for proving soundness, termination and completeness of GDPLL. The original DPLL procedure is an instance. Subsequently the GDPLL instances for equality logic, and the logic of equality with uninterpreted functions are presented. In [Ganzinger et al., 2004] a different generalization of the DPLL procedure is presented.

6.1 GDPLL

The DPLL procedure, due to Davis, Putnam, Logemann, and Loveland, is the basis of some of the most successful propositional satisfiability solvers. The original DPLL procedure was developed as a proof-procedure for first-order logic. It has been used so far almost exclusively for propositional logic because of its highly inefficient treatment of quantifiers. In this chapter, we present the general version of the procedure, and we adopt it for some fragments of first order logic. The satisfiability problem is decidable in these logics.

Most of the techniques relevant in the setting of the DPLL procedure are also applicable to GDPLL. Essentially, the DPLL procedure consists of the following three rules: the unit clause rule, the splitting rule, and the pure literal rule. Both the unit clause rule and the pure literal rule reduce the formula according to some criteria. Thus, in GDPLL we may assume a function Reduce which performs all rules for formula reduction. GDPLL has a splitting rule, which carries out a case analysis with respect to an atom a. The current set of clauses φ splits into two sets: the one where a is true, and another where a is false.

In the following we assume a function $\mathsf{Reduce} : \mathsf{Cnf} \to \mathsf{Cnf}$. We define the set $\mathsf{R} = \{\varphi \in \mathsf{Reduce}(\mathsf{Cnf}) \mid \bot \notin \varphi\}$.

In the following we also assume functions

- $\mathsf{Eligible} : \mathsf{R} \to \mathsf{At}$,
- $\mathsf{SatCriterion} : \mathsf{R} \to \{\mathsf{true}, \mathsf{false}\}$,
- Filter, where $\mathsf{Filter}(\varphi, a)$ is defined for $\varphi \in \mathsf{R}$ and $a \in \mathsf{Eligible}(\varphi)$.

We now introduce the requirements on the functions above: for all $\psi \in \mathsf{Cnf}$, for all $\varphi \in \mathsf{R}$, and for all $a \in \mathsf{Eligible}(\varphi)$ the functions should satisfy the following properties.

1. $\mathsf{Reduce}(\psi)$ is satisfiable iff ψ is satisfiable,

```
GDPLL($\varphi$) : {SAT, UNSAT} =
    begin
        $\varphi$ := Reduce($\varphi$);
        if ($\bot \in \varphi$) then return UNSAT;
        if (SatCriterion($\varphi$)) then return SAT;
        choose $a \in$ Eligible($\varphi$);
        if GDPLL(Filter($\varphi, a$)) = SAT then return SAT;
        if GDPLL(Filter($\varphi, \neg a$)) = SAT then return SAT;
        return UNSAT;
    end;
```

Figure 6.1 The GDPLL procedure

2. φ is satisfiable iff at least one of $\mathsf{Filter}(\varphi, a)$ and $\mathsf{Filter}(\varphi, \neg a)$ is satisfiable,
3. $\mathsf{Reduce}(\mathsf{Filter}(\varphi, a)) \prec \varphi$ and $\mathsf{Reduce}(\mathsf{Filter}(\varphi, \neg a)) \prec \varphi$, for some well-founded order $\prec$ on $\mathsf{Reduce}(\mathsf{Cnf})$.
4. if $\mathsf{SatCriterion}(\varphi) = \mathsf{true}$ then φ is satisfiable,
5. if $\mathsf{SatCriterion}(\varphi) = \mathsf{false}$ then $\mathsf{Eligible}(\varphi) \neq \emptyset$.

Figure 6.1 shows the pseudo-code of the skeleton of the algorithm. The procedure takes as an input $\varphi \in \mathsf{Cnf}$. GDPLL proceeds until either the function SatCriterion has returned true for at least one branch, or the empty clause has been derived for all branches. Respectively, either SAT or UNSAT is returned.

6.1.1 Soundness and completeness of GDPLL

Theorem 6.1 (**Soundness and completeness**) *Let* $\varphi \in \mathsf{Cnf}$. *Then the following properties hold:*

- *If* φ *is satisfiable then* $\mathsf{GDPLL}(\varphi) = \mathsf{SAT}$.
- *If* φ *is unsatisfiable then* $\mathsf{GDPLL}(\varphi) = \mathsf{UNSAT}$.

Proof Let $\varphi \in \mathsf{Cnf}$. We apply induction on $\prec$, which is well-founded by property 3. So assume (induction hypothesis) that the theorem holds for all ψ such that $\mathsf{Reduce}(\psi) \prec \mathsf{Reduce}(\varphi)$. By property 1, $\mathsf{Reduce}(\varphi)$ is satisfiable if φ is satisfiable, and $\mathsf{Reduce}(\varphi)$ is unsatisfiable if φ is unsatisfiable.

Let $\bot \in \mathsf{Reduce}(\varphi)$. Then trivially φ is unsatisfiable, and $\mathsf{GDPLL}(\varphi)$ returns UNSAT.

Let $\perp \notin \mathsf{Reduce}(\varphi)$. Assume that for all ψ such that $\mathsf{Reduce}(\psi) \prec \mathsf{Reduce}(\varphi)$, $\mathsf{GDPLL}(\psi)$ returns UNSAT if ψ is unsatisfiable, and $\mathsf{GDPLL}(\psi)$ returns SAT if ψ is satisfiable.

If $\mathsf{SatCriterion}(\mathsf{Reduce}(\varphi)) = \mathsf{true}$ then by property 4, φ is satisfiable, and $\mathsf{GDPLL}(\varphi) = \mathsf{SAT}$.

If $\mathsf{SatCriterion}(\mathsf{Reduce}(\varphi)) = \mathsf{false}$ then by property 5, $\mathsf{Eligible}(\mathsf{Reduce}(\varphi)) \neq \emptyset$.

By property 3, for all $\varphi \in \mathsf{Cnf}$ and all $a \in \mathsf{Eligible}(\varphi)$,

- $\mathsf{Reduce}(\mathsf{Filter}(\mathsf{Reduce}(\varphi), a)) \prec \mathsf{Reduce}(\varphi)$,
- $\mathsf{Reduce}(\mathsf{Filter}(\mathsf{Reduce}(\varphi), \neg a)) \prec \mathsf{Reduce}(\varphi)$.

Let $\mathsf{Reduce}(\varphi)$ be unsatisfiable. Then by property 2, $\mathsf{Filter}(\mathsf{Reduce}(\varphi), a)$ and $\mathsf{Filter}(\mathsf{Reduce}(\varphi), \neg a)$ are unsatisfiable. We can apply the induction hypothesis. Then both $\mathsf{GDPLL}(\mathsf{Filter}(\mathsf{Reduce}(\varphi), a))$ and $\mathsf{GDPLL}(\mathsf{Filter}(\mathsf{Reduce}(\varphi), \neg a))$ return UNSAT. By definition of GDPLL, $\mathsf{GDPLL}(\varphi)$ also returns UNSAT.

Let $\mathsf{Reduce}(\varphi)$ be satisfiable. By property 2, at least one of $\mathsf{Filter}(\mathsf{Reduce}(\varphi), a)$ and $\mathsf{Filter}(\mathsf{Reduce}(\varphi), \neg a)$ is unsatisfiable. By the induction hypothesis, at least one of $\mathsf{GDPLL}(\mathsf{Filter}(\mathsf{Reduce}(\varphi), a))$ and $\mathsf{GDPLL}(\mathsf{Filter}(\mathsf{Reduce}(\varphi), \neg a))$ return SAT, and by definition of GDPLL, $\mathsf{GDPLL}(\varphi)$ also returns SAT. ⊠

6.2 Instances for the GDPLL procedure

In this section we will define the functions Eligible, Filter, Reduce and SatCriterion for propositional logic and equality logic.

6.2.1 GDPLL for Propositional Logic

Two main operations of the DPLL procedure are *unit propagation* and *purification*. Unit clauses can only be satisfied by a specific assignment to the corresponding propositional variable, and the complementary assignment will lead to a contradiction. Hence all occurrences of this variable can be eliminated. Elimination of the variable can create a new unit clause, so this process has to be repeated until no unit clauses are left. Purification can be applied if the formula contains pure literals. Such literals can be eliminated by assigning true in the positive case and false in the negative case. Note that this cannot introduce unit clauses.

It can be seen that the DPLL procedure for propositional logic is a particular case of GDPLL, where unit resolution and purification are performed by Reduce. In

```
Reduce($\varphi$);
    begin
        $\psi := \varphi$;
        while (there is a unit clause in $\psi$)
        begin;
            $l :=$ UnCl($\psi$);
            $\psi := \psi|_l$;
        end;
        while (there is a pure literal in $\psi$)
        begin;
            $l :=$ PureLiteral($\psi$);
            $\psi := \psi|_l$;
        end;
    return $\psi$;
    end;
```

Figure 6.2 The Reduce function for propositional logic

case of propositional logic, we let an eligible atom be an arbitrary atom, i.e. to coincide with the original DPLL procedure, we choose

$$\mathsf{Eligible}(\varphi) = \mathsf{At}(\varphi).$$

We define SatCriterion as follows

$$\mathsf{SatCriterion}(\varphi) = \begin{cases} \mathsf{true} & \text{if } \varphi = \emptyset, \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

Figure 6.2 shows the function Reduce. The function $\mathsf{UnCl}(\psi)$ returns a unit clause contained in $\mathsf{Cls}(\psi)$, and the function $\mathsf{PureLiteral}(\psi)$ returns a pure literal contained in $\mathsf{Lit}(\psi)$.

We define for all $\varphi \in \mathsf{Reduce}(\mathsf{Cnf})$ and all $l \in \mathsf{Lit}(\varphi)$

$$\mathsf{Filter}(\varphi, l) = \varphi \wedge l.$$

Definition 6.2 (**Ordering on formulae**) *Given* $\varphi_1, \varphi_2 \in \mathsf{Cnf}$*, we define* $\varphi_1 \prec \varphi_2$ *if* $|\mathsf{Pr}(\varphi_1)| < |\mathsf{Pr}(\varphi_2)|$.

The defined order is trivially well-founded.

Example 6.3 Consider

$$\varphi_1 \equiv \{\{\neg p, q, r\}, \{\neg q, r\}, \{\neg r\}\},$$

$$\varphi_2 \equiv \{\{\neg p, r\}, \{p, r\}, \{\neg r\}, \{p, \neg r\}, \{\neg p\}\}.$$

According to the definition $\varphi_2 \prec \varphi_1$.

Theorem 6.4 *The functions* Reduce*,* $\mathsf{Eligible}$*,* Filter*,* $\mathsf{SatCriterion}$ *satisfy the Properties 1–5.*

Proof

1. Property 1 holds since unit clauses can only be satisfied by a specific assignment to a corresponding propositional variable, and the complementary assignment will lead to contradiction, and pure literals can be eliminated by assigning true in the positive case and false in the negative case.

2. By definition of Filter, Property 2 trivially holds.

3. We prove Property 3. We have to prove that $\mathsf{Reduce}(\varphi \wedge l) \prec \varphi$ for all $\varphi \in \mathsf{Cnf}$ and for all $l \in \mathsf{Lit}(\varphi)$.

   We consider the case when $\varphi|_l$ contains no unit clauses and pure literals. All other cases can be easily proven by induction. Let $l \equiv p$ for some $p \in \mathrm{Pr}$. Since by the theorem conditions $l \in \mathsf{Lit}(\varphi)$ then trivially $\mathsf{Pr}(\varphi|_l) \subseteq \mathsf{Pr}(\varphi)\backslash\{p\}$.

   Using Definition 6.2 one can see that from

   $$|\mathsf{Pr}(\mathsf{Reduce}(\varphi \wedge p))| = |\mathsf{Pr}(\varphi|_p)| \leq |\mathsf{Pr}(\varphi)\backslash\{p\}| < |\mathsf{Pr}(\varphi)|$$

   it follows that $\mathsf{Reduce}(\varphi \wedge p) \prec \varphi$.

   The case $l \equiv \neg p$ for some $p \in \mathsf{Pr}$ is similar.

4. We check Property 4. By definition, the function $\mathsf{SatCriterion}(\varphi)$ returns true only if $\varphi = \emptyset$, which is satisfiable by definition.

5. Property 5 follows from the fact that if $\mathsf{SatCriterion}(\varphi) = \mathsf{false}$ then by the definition of $\mathsf{SatCriterion}$ there is $C \in \varphi$ such that $C \neq \bot$. Then $\mathsf{Lit}(\varphi) \neq \emptyset$, and $\mathsf{Eligible}(\varphi) \neq \emptyset$.

$\boxtimes$

We have defined the functions $\mathsf{Eligible}$, Reduce, Filter, and $\mathsf{SatCriterion}$. One can see that GDPLL now coincides with the DPLL procedure for propositional logic.

In the situation when φ consists of relatively few clauses compared to the number of variables in each clause, splitting can be very inefficient. The following theorem allows the procedure to stop when every clause in φ contains at least one negative literal.

Theorem 6.5 (SAT criterion) *Let $\varphi \in \mathsf{Cnf}$ contain no purely positive clause. Then φ is satisfiable.*

Proof For propositional logic the domain can be an arbitrary set. Let some set D be a domain. Since no term contains function symbols their interpretation plays no role. For all $p \in \mathsf{Pr}$ we define $[\![p]\!]_{\mathcal{D}} = \mathsf{false}$. Regarding the theorem conditions for all $C \in \varphi$ there is $l \in C$ such that $l \equiv \neg p$ for some $p \in \mathsf{Pr}$. We have that $[\![l]\!]_{\mathcal{D}} = \mathsf{true}$ and $[\![C]\!]_{\mathcal{D}} = \mathsf{true}$ for all $C \in \varphi$. By definition of a formula interpretation $[\![\varphi]\!]_{\mathcal{D}} = \mathsf{true}$. ⊠

Example 6.6 Consider

$$\varphi \equiv \{\{\neg p, q, r\}, \{\neg q, r\}, \{\neg r\}\}.$$

One can see easily that the formula is satisfiable.

Using the above theorem we can define the function SatCriterion.

$$\mathsf{SatCriterion}(\varphi) = \begin{cases} \mathsf{true} & \text{if } C \cap \mathsf{Lit}_n \neq \emptyset \text{ for all } C \in \varphi, \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

One can easily check that the function SatCriterion satisfies Properties 4 and 5. In following sections we define the function SatCriterion in a similar way.

6.2.2 GDPLL for Equality Logic

We now define the functions Eligible, Filter, Reduce and SatCriterion for equality logic. The function Reduce removes all clauses containing a literal of the shape $x \approx x$ and literals of the shape $x \not\approx x$ from other clauses. In the following we consider $x \approx y$ and $y \approx x$ as the same atom.

In case of propositional logic we may choose any atom contained in a CNF to apply the split rule. The correctness of GDPLL is not immediate for other instances. For equality logic we define an atom to be eligible if it occurs as a positive literal in the formula, i.e. $\mathsf{Eligible}(\varphi) = \mathsf{Lit}_p(\varphi)$.

Example 6.7 Let us consider the formula

$$\varphi \equiv \{\{x \approx y\}, \{y \approx z\}, \{x \not\approx z\}\}.$$

One can see that $(x \approx z) \notin \mathsf{Eligible}(\varphi)$ since it occurs in φ only as a negative literal $x \not\approx z$.

We define the function $\mathsf{SatCriterion}$, so that it indicates that there are no purely positive clauses left:

$$\mathsf{SatCriterion}(\varphi) = \begin{cases} \mathsf{true} & \text{if } C \cap \mathsf{Lit}_n \neq \emptyset \text{ for all } C \in \varphi, \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

Example 6.8 Consider

$$\varphi \equiv \{\{x \approx y, y \not\approx z\}, \{x \approx z, x \not\approx y, y \approx z\}, \{x \not\approx z\}\}.$$

One can easily see that the formula is satisfied by an assignment α such that $\alpha(x') \neq \alpha(x'')$ for all $x', x'' \in \mathsf{Lit}_n(\varphi)$.

We denote by $\varphi[x := y]$ the formula φ, where all occurrences of x are replaced by y.

We define the function Filter as follows.

- $\mathsf{Filter}(\varphi, x \approx y) = \varphi|_{x \approx y}[x := y]$,
- $\mathsf{Filter}(\varphi, x \not\approx y) = \varphi|_{x \not\approx y} \wedge (x \not\approx y)$.

Definition 6.9 (**Ordering on formulae**) *Given* $\varphi_1, \varphi_2 \in \mathsf{Cnf}$*, we define* $\varphi_1 \prec \varphi_2$ *if* $|\mathsf{Lit}_p(\varphi_1)| < |\mathsf{Lit}_p(\varphi_2)|$.

The defined order is trivially well-founded.

Example 6.10 Consider

$$\varphi_1 \equiv \{\{x \approx y, y \not\approx z\}, \{x \approx z, x \not\approx y, y \approx z\}, \{x \not\approx z\}\},$$

$$\varphi_2 \equiv \{\{x \approx y, y \not\approx z\}, \{x \not\approx y, y \approx z\}, \{x \not\approx z\}, \{x \not\approx y, y \not\approx z\}\}.$$

Since $\mathsf{Lit}_p(\varphi_2) = \mathsf{Lit}_p(\varphi_1) \backslash \{x \approx z\}$ one can see that by the definition $\varphi_2 \prec \varphi_1$.

Theorem 6.11 *The functions* Reduce*,* Eligible*,* Filter*,* SatCriterion *satisfy the Properties 1-5.*

Proof

1. Property 1 holds since $[\![x \approx x]\!]_{\mathcal{D}} = \mathsf{true}$ for all admissible $\mathcal{D}$ and all $\alpha : \mathsf{Var} \rightarrow D$, i.e. removing clauses containing $x \approx x$ from the formula and the literal $x \not\approx x$ from all clauses can be done without influencing the satisfiability of the formula.
2. We prove Property 2. For each $a \in \mathsf{At}$, φ is satisfiable iff at least one of $\varphi \wedge a$ and $\varphi \wedge \neg a$ is satisfiable. Trivially, $\varphi \wedge (x \approx y)$ is satisfiable iff $\varphi|_{x \approx y}[x := y]$ is satisfiable for all $x, y \in \mathsf{Var}$. From this we can conclude that the property holds.
3. We will prove Property 3.

   At first we prove that $\varphi|_{x \approx y}[x := y] \prec \varphi$ and $\varphi|_{x \not\approx y} \wedge (x \not\approx y) \prec \varphi$ for all $\varphi \in \mathsf{Cnf}$ and all $(x \approx y) \in \mathsf{Lit}(\varphi)$.

   - Let $l \equiv x \approx y$.
     It follows from the definition of $\varphi|_{x \approx y}$ and the fact $(x \approx y) \in \mathsf{Lit}(\varphi)$ that
     $$|\mathsf{Lit}_p(\varphi|_{x \approx y})| < |\mathsf{Lit}_p(\varphi)|.$$
     One can easily check that for all $\psi|_{(x \approx y)} \in \mathsf{Cnf}$
     $$|\mathsf{Lit}_p(\psi|_{x \approx y}[x := y])| \leq |\mathsf{Lit}_p(\psi|_{x \approx y})|.$$
     We obtain that
     $$|\mathsf{Lit}_p(\varphi|_{x \approx y}[x := y])| \leq |\mathsf{Lit}_p(\varphi|_{x \approx y})| < |\mathsf{Lit}_p(\varphi)|.$$
     We can conclude that
     $$\varphi|_{x \approx y}[x := y] \prec \varphi.$$
   - Let $l \equiv x \not\approx y$.
     Since by the theorem conditions $(x \approx y) \in \mathsf{Lit}_p(\varphi)$ then
     $$|\mathsf{Lit}_p(\varphi|_{x \not\approx y})| < |\mathsf{Lit}_p(\varphi)|.$$
     We have that
     $$|\mathsf{Lit}_p(\varphi|_{x \not\approx y} \wedge (x \not\approx y))| = |\mathsf{Lit}_p(\varphi|_{x \not\approx y})| < |\mathsf{Lit}_p(\varphi)|.$$
     We can conclude that
     $$\varphi|_{x \not\approx y} \wedge (x \not\approx y) \prec \varphi.$$

Regarding the definition of the function Reduce, we obtain that for all $\varphi \in \mathsf{Reduce}(\mathsf{Cnf})$ and all $a \in \mathsf{Eligible}(\varphi)$

$$\mathsf{Reduce}(\mathsf{Filter}(\varphi, a)) \prec \varphi, \mathsf{Reduce}(\mathsf{Filter}(\varphi, \neg a)) \prec \varphi.$$

4. Let $\mathsf{SatCriterion}(\varphi) = \mathsf{true}$. Then either $\varphi = \emptyset$ or every clause in φ contains at least one negative literal. If $\varphi = \emptyset$ then by definition φ is satisfiable. Let us consider the remaining case. Let D be a domain such that $|D| \geq |\mathsf{Var}(\varphi)|$. We choose an assignment α such that $\alpha(x) \neq \alpha(y)$ for all $x, y \in \mathsf{Var}(\varphi)$. Regarding the definition of a CNF interpretation we have that $[\![\varphi]\!]_{\mathcal{D}} = \mathsf{true}$.

5. Property 5 follows from the fact that if $\mathsf{SatCriterion}(\varphi) = \mathsf{false}$ then there is some $C \in \varphi$ such that $C \neq \bot$ and $C \cap \mathsf{Lit}_p = C$. We obtain that $\mathsf{Lit}_p(\varphi) \neq \emptyset$, and $\mathsf{Eligible}(\varphi) \neq \emptyset$.

$\boxtimes$

Many variations of this instance for GDPLL are possible. Here we choose to do the main job in Filter, while Reduce only removes trivialities $x \approx x$ from its argument. In fact in this version Reduce is only required in the first call of GDPLL as a kind of preprocessing, the other calls of Reduce may be omitted since atoms of the shape $x \approx x$ are not created by Filter.

Chapter 7

Checking satisfiability of EUF-formulas

“A choice function exists in constructive mathematics, because a choice is implied by the very meaning of existence”

[Errett Bishop]

“Every person thinks his own intellect is perfect, and his own child handsome”

[Sa’di]

“Would you tell me, please, which way I ought to go from here?” “That depends a good deal on where you want to get to,” said the Cat. “I don’t much care where” said Alice. “Then it doesn’t matter which way you go,” said the Cat.

[“Alice’s Adventures in Wonderland”, Lewis Carroll]

The logic of equality with uninterpreted functions has been proposed for verifying hardware designs [Burch and Dill, 1994]. Hardware and software systems grow in scale and functionality. As a result of increasing complexity, the likelihood of errors is much greater. Hence, formal verification has become an increasingly important technique to establish the correctness of hardware designs. A formal hardware/software verification problem is to formally establish that an implementation satisfies a specification.

EUF formulae have been successfully applied for the verification of pipelined processors [Burch and Dill, 1994, Bryant et al., 1999a,b, Bryant and Velev, 2000]. The EUF-logic provides a means of abstracting the manipulation of data by a processor when verifying the correctness of its control logic, i.e. uninterpreted functions are used to represent all blocks that transform and evaluate values. The method has two phases: the first phase consists of the constructing of a logical formula which is valid if and only if the implementation is correct with respect to the specification. During the second phase a decision procedure checks validity of the formula. The validity problem for this logic is decidable.

The EUF-logic was also used for translation validation [Pnueli et al., 1999, 2001, 2002, Rodeh and Shtrichman, 2001], i.e checking the correctness of a compiler's translation by verifying the equivalence between the source and target codes.

In general, this type of logic is mainly used for proving the equivalence between systems. While verifying the equivalence between two formulae, it is possible to replace all functions, except the equality sign and propositional operators, with uninterpreted functions. This abstraction process does not preserve validity and may transform a valid formula into an invalid one, e.g.

$$x + y \approx y + x$$

is valid, but

$$f(x, y) \approx f(y, x)$$

is not.

However, deciding validity of a general formula φ^G can be replaced by checking unsatisfiability of the the negated EUF-formula $\neg\varphi^{EUF}$: if $\neg\varphi^{EUF}$ is unsatisfiable then φ^G is valid.

7.1 Satisfiability of the EUF-formulae

In the past few years several decision procedures for checking satisfiability of EUF-formulae have been suggested [Shostak, 1979, Barrett et al., 1996, Bjørner et al., 1997, Barrett et al., 2002]. Roughly we can speak about three main approaches.

- *Checking satisfiability of a formula as it is (without transforming it).* There have been several decision procedures developed to check satisfiability of EUF logic formulae.
  - *Congruence closure.* A traditional approach to check satisfiability of EUF formulae is by applying congruence closure algorithms [Shostak, 1978, Nelson and Oppen, 1980, Bachmair and Tiwari, 2000]. This is mainly applied to the conjunction of equalities. This approach works as follows: an equivalence class for each set of variables that have the same value, is maintained. The property of functional consistency is enforced by computing the congruence closure of these classes, i.e. if the arguments of two function instances are in one equivalence class, the function instances are also in one equivalence class.
  - *The DPLL-based approach in combination with congruence closure.* Using the congruence closure approaches one can check satisfiability of conjunctions of equalities. Disjunctions can be treated by case splitting [Nieuwenhuis and Oliveras, 2003].
  - *The* GDPLL*-based method.* In Chapter 8, the GDPLL-based method is introduced.
  - *The BDD-based approach.* Chapter 9 describes a BDD-based representation for the EUF formulae.
- *Lazy encoding.* A main idea of this approach is to find first satisfying interpretations (if those exist) for the propositional structure of a formula and then to verify the interpretations against the theory. The drawback is that the number of interpretations satisfying the propositional abstraction of a formula can be exponential in the size of a formula.
- *Eager encoding.* This approach is based on the idea that if a first-order statement is true, then it must be possible to prove it by analyzing propositional formulae generated from the formula. A general idea is to transform a formula to a formula in a simpler logic preserving (un)satisfiability, and then to use a decision procedure for this logic. In case of the EUF logic there are two possible ways to perform such a transformation: to reduce an EUF-formula to an equality logic formula and then to use a decision procedure for the equality logic, or to transform an EUF-formula to an equality logic formula first, then an equality logic formula can be reduced to a propositional one and any standard SAT checker can be applied.

7.2 Reduction of the EUF Logic to Equality Logic

As shown by Ackermann [Ackermann, 1954], deciding the validity of EUF formulae can be reduced to checking the satisfiability of equality logic formulae. Ackermann proposed to replace each application of an uninterpreted function symbol with a new variable and for each pair of function instances to add a constraint which enforces the property of functional consistency. It means that while replacing any two subterms of the form $f(a)$ and $f(b)$ by new variables f_1 and f_2, we have to add a constraint of the form $a = b \to f_1 = f_2$. For example, checking the satisfiability of the formula

$$f(a) \not\approx f(b)$$

is replaced by deciding the satisfiability of the formula

$$f_1 \not\approx f_2 \wedge (a \approx b \to f_1 \approx f_2).$$

The complexity of satisfiability checking for equality logic formulae arises from the fact that the properties of equality have to be taken into account. For example, the formula $(a \approx b) \wedge (b \approx c) \wedge (a \not\approx c)$ is unsatisfiable since it violates the transitivity of equality.

A number of decision procedures exist for the theory of equality over propositional logic.

- *Range allocation.* Due to the *finite domain property*, which states that an equality logic formula is satisfiable if and only if it is satisfiable over a finite domain, some approaches [Pnueli et al., 2002] find a small domain for each variable, which is large enough to maintain satisfiability. Given an upper bound n on this domain, each variable can be encoded as a vector of $\lceil log(n) \rceil$ bits, and then existing BDD techniques can be used to check the state-space spanned by these variables.

- *Positive Equality.* This approach identifies terms that have certain syntactic properties in the formula, and replaces them with unique constants [Bryant et al., 1999a]. Then it finds a small domain for every variable, which is large enough to maintain the satisfiability of a formula. The state-space is checked with a BDD-based tool.

- *Equational BDD.* In [Groote and Pol, 2000], equational BDDs (EQ-BDDs) are defined, in which all paths are satisfiable by construction. That approach extends the notion of orderedness to capture the properties of reflexivity, symmetry, transitivity, and substitutivity. The advantage of the method is that satisfiability checking for a given BDD can be done immediately. However, it is restricted to the case when equalities do not contain function symbols.

- *Equality Resolution.* This approach is described in Chapter 5.

7.3 Reduction to Propositional Logic

As mentioned in Section 7.1, we can divide approaches for checking satisfiability of equality logic formulae via a transformation to propositional ones into two main groups: lazy encoding and eager encoding.

7.3.1 Lazy encoding

An EUF formula φ is a propositional combination of equalities between terms. Assume φ^* is a propositional abstraction of φ, i.e. φ^* is a formula φ, where every equality e is replaced with a proposition p. Trivially, such a transformation does not preserve unsatisfiability. For example, for the formula

$$\varphi \equiv (a \approx b) \wedge (b \approx c) \wedge (f(a) \not\approx f(c))$$

its propositional abstraction is

$$\varphi^* \equiv p \wedge q \wedge \neg r.$$

Conversely, if an interpretation satisfies φ then this interpretation also satisfies φ^*. This observation defines a strategy to check satisfiability of φ the following steps have be performed.

- To find an interpretation I satisfying φ^*.
- If such interpretation is found, to check whether the conjunction of the corresponding equalities is satisfiable in the EUF logic.
- If all interpretations satisfying φ^* have been checked and none of them satisfies φ then φ is unsatisfiable. Otherwise, φ is satisfiable.

Example 7.1 Consider the formula

$$\varphi^E \equiv v_1 \approx v_3 \wedge v_1 \approx v_4 \wedge v_1 \not\approx v_2 \wedge (v_1 \not\approx v_2 \vee v_2 \approx v_3) \wedge (v_1 \not\approx v_2 \vee v_3 \approx v_4).$$

It will be replaced by the formula

$$\varphi^P \equiv e_{13} \wedge e_{14} \wedge \neg e_{12} \wedge (\neg e_{12} \vee e_{23}) \wedge (\neg e_{12} \vee e_{34}).$$

Suppose for some structure $\mathcal{D}$, $[\![e_{12}]\!]_{\mathcal{D}} = \mathsf{false}$, $[\![e_{13}]\!]_{\mathcal{D}} = \mathsf{true}$, $[\![e_{14}]\!]_{\mathcal{D}} = \mathsf{true}$, $[\![e_{23}]\!]_{\mathcal{D}} = \mathsf{true}$, $[\![e_{34}]\!]_{\mathcal{D}} = \mathsf{true}$. This assignment satisfies the formula φ^P, i.e. $[\![\varphi^P]\!]_{\mathcal{D}} = \mathsf{true}$. At the same time this structure is not admissible for the equality logic formula φ^E since it violates the transitivity of equalities. For example, if $[\![e_{13}]\!]_{\mathcal{D}} = \mathsf{true}$ and $[\![e_{23}]\!]_{\mathcal{D}} = \mathsf{true}$ then e_{12} cannot be interpreted as false.

Assume for a structure $\mathcal{D}'$, $[\![e_{12}]\!]_{\mathcal{D}} = \mathsf{false}$, $[\![e_{13}]\!]_{\mathcal{D}} = \mathsf{true}$, $[\![e_{14}]\!]_{\mathcal{D}} = \mathsf{true}$, $[\![e_{23}]\!]_{\mathcal{D}} = \mathsf{false}$, $[\![e_{34}]\!]_{\mathcal{D}} = \mathsf{true}$. This interpretation satisfies both formulae.

In [Fontaine and Gribomont, 2002] a BDD-based approach for the combination of theories is presented. It is noted that BDDs, when they are used for first order logic, are not canonical representations any more. For example, BDDs representing $(x \approx y) \wedge p(x)$ and $(x \approx y) \wedge p(y)$ are different although they are logically equivalent. Special constraints have to be added to remove unsatisfiable paths.

Goel et al. [Goel et al., 1998] proposed to decide equality logic formulae by replacing all equalities with new propositional variables, i.e. to replace an equality $v_i \approx v_j$ with a new variable e_{ij}. In this approach the BDD for the resulting formula is calculated without taking into account the transitivity of equalities, and for assignments satisfying the BDD, it is inspected on whether they also satisfy the original equality logic formula.

7.3.2 Eager encoding

A different approach is based on a transformation of an equality formula φ^E to an equiv-satisfiable propositional formula φ^P. Checking satisfiability of φ^P can be done by any standard SAT checker. According to [Zantema and Groote, 2003] for such a transformation some properties are desirable:

- the size of φ^P is not too big in comparison with φ^E;
- the structure of φ^P reflects the structure of φ^E;
- the variables of φ^P correspond to equalities in φ^E.

We can distinguish three main approaches: bit vector encoding, adding of transitivity constraints [Bryant and Velev, 2002], and equality substitution [Zantema and Groote, 2003].

- *Bit vector encoding.* Assume φ^E is an equality logic formula. In this approach for every $a \in \mathsf{Const}$, $\lceil \log(|\mathsf{Const}(\varphi)|) \rceil$ new propositional variables a_i are introduced. An equiv-satisfiable propositional formula φ^P is obtained from φ^E by replacing every equality $a \approx b$ with

  $$\bigwedge_i (a_i \leftrightarrow b_i).$$

  According to [Zantema and Groote, 2003], although bit vector encoding transforms an equality formula into a propositional one of relatively small size and the structure of the formula is preserved, it gives a very bad performance on proving unsatisfiability.

- *Transitivity constraints.* In this approach an equality formula φ^E is transformed to the propositional formula φ^P by adding all possible transitivity constraints of the shape $(a \approx b) \wedge (b \approx c) \rightarrow (a \approx c)$ and replacing all equalities with new propositions.

The drawback of this approach is a relatively big size of the propositional formula. Bryant et al. [Bryant and Velev, 2002] analyze which transitivity properties may be relevant.

- *Equality Substitution.* This technique was introduced in [Zantema and Groote, 2003]. As bit vector encoding, this approach transforms an equality logic formula to a propositional one by performing a substitution on the equalities of the formula, while the structure of the formulae is unchanged. It is shown that the best experimental results are obtained for two transformations: adding transitivity constraints and equality substitution. Equality substitution gives rise to the smallest formulae between these two approaches.

Chapter 8

The GDPLL-based procedure for the EUF-logic

"The sees haunted by the sun never fail to find its way between the stones in the ground. And the pure logician, if no sun draws him forth, remains entangled in his logic"

[Antoine de Saint-Exupéry]

"It is no exaggeration to say that a straightforward realistic approach to mathematics has yet to be tried. It is time to make the attempt"

[Errett Bishop]

In this chapter we present a GDPLL-based approach for checking satisfiability of formulae in the EUF-logic. Our approach is based on a generalized version of the propositional DPLL procedure.

Part of our method is a technique for reducing a formula that can be of interest itself. The procedure can incorporate some optimization techniques developed by the SAT community for the DPLL method.

We can see from the considered example that our approach can be efficient for some formulae. However, at present we cannot make general conclusions about the efficiency of the procedure.

In Chapter 6 we presented GDPLL, a generalization of the propositional DPLL procedure, which solves the satisfiability problem for decidable fragments of quantifier-free first-order logic. In this chapter we show how the GDPLL procedure can be used to decide the satisfiability of equality logic formulae with uninterpreted functions in CNF.

The GDPLL procedure is done recursively, according to the following steps.

- φ is replaced by a CNF $\mathsf{Reduce}(\varphi)$ such that φ is satisfiable iff $\mathsf{Reduce}(\varphi)$ is satisfiable.
- If $\bot \in \varphi$, then for any structure $\mathcal{D}$, $[\![\varphi]\!]_{\mathcal{D}} = \mathsf{false}$. Therefore, $\mathsf{GDPLL}(\varphi)$ returns "unsatisfiable".
- If for $\varphi \in \mathsf{RCnf}$, $\mathsf{SatCriterion}(\varphi) = \mathsf{true}$ then $\mathsf{GDPLL}(\varphi)$ returns "satisfiable".
- If none of the above situations occur , then the procedure chooses a literal l from $\mathsf{Eligible}(\varphi)$ according to some heuristic criterion, and proceeds with two cases $\mathsf{Filter}(\varphi, l)$ and $\mathsf{Filter}(\varphi, \neg l)$.

Therefore, the GDPLL procedure is an algorithm with four blocks, namely Eligible(), Reduce(), Filter(), SatCriterion(), which have to be defined for the particular logic. They correspond to choosing an atomic formula for splitting, reducing an intermediate formula, and a satisfiability criterion. It was shown in Chapter 6 that if Properties 1-5 are satisfied then the procedure is sound and complete. In the following we will show how to fulfill the modules of the GDPLL procedure for the case of the EUF-logic.

8.1 The reduction rules

The function Reduce is defined by means of a set of reduction rules, that can be applied in any order. In this section we define reduction rules for EUF-CNFs.

Starting with an arbitrary CNF, we can simplify all clauses contained in it.

Definition 8.1 (**Simplified clauses**)

- *Suppose C is a clause. By $C\downarrow$ we mean the normal form obtained from C after applying the following simplification rule.*

$$C \to C\backslash\{t \not\approx t\} \text{ if for some term } t,\ (t \not\approx t) \in C.$$

- *A clause C is called simplified if $C \equiv C\downarrow$.*

Given a CNF, we can simplify it by repeatedly applying the following simplification rules.

Definition 8.2 (**Simplified CNFs**)

- *Suppose φ is a CNF. By $\varphi\downarrow$ we mean the normal form of φ after applying the following rules.*
  - *$C \to C\downarrow$ for some clause $C \in \varphi$.*
  - *$\varphi \to \varphi\backslash\{C\}$ for a clause $C \in \varphi$ if for some term t, $(t \approx t) \in C$.*
- *A CNF φ is called simplified if $\varphi \equiv \varphi\downarrow$.*

Now we can define a system of reduction rules. Starting with an arbitrary CNF, we can transform it by repeatedly applying the reduction rules.

Definition 8.3 (**Reduction rules on CNFs**) *We define a reduction system* Reduce *as follows.*

1. *$\{\{s \approx t\}\} \uplus \varphi \to \{\{s \approx t\}\} \uplus \varphi[s := t]$ if $s, t \in \mathsf{SubTerm}(\varphi)$ and $s \notin \mathsf{SubTerm}(t)$.*
2. *$\varphi \to \varphi\downarrow$.*

Rule 1 of the reduction system Reduce allows to substitute equals for equals. Rule 2 simplifies a CNF by removing all equalities of the form $t \approx t$ from a formula. The transformation by the reduction rules yields a logically equivalent CNF.

Definition 8.4 (**Reduced CNFs**) *We define a reduced CNF to be a normal form with respect to the reduction system* Reduce*.*

By the following corollary we show which shape a reduced CNF may have.

Proposition 1 *If φ is a reduced formula then the following holds.*

1. *If $\varphi \equiv \{\{s \approx t\}\} \uplus \varphi'$ then either $s \notin \mathsf{SubTerm}(\varphi')$ or $t \notin \mathsf{SubTerm}(\varphi')$.*
2. *φ does not contain equalities of the form $t \approx t$.*

Proof If φ does not satisfy 1 then Rule 1 of the reduction system Reduce can be applied. If φ does not satisfy 2 then Rule 2 of the reduction system Reduce can be applied. ⊠

We prove in Section 8.1.1 that the set of reduction rules is terminating, so at least one normal form exists. Unfortunately, the rules are not confluent as it is shown by the following example. So the normal form is not unique.

Example 8.5 Consider $\varphi = \{\{a \approx f(b)\}, \{a \approx g(c)\}, \{f(b) \approx h(a, c)\}\}$.

1. Applying Rule 1 of the reduction system Reduce on $a \approx f(b)$, we can replace a with $f(b)$. Therefore, the reduced formula is
$$\varphi' = \{\{a \approx f(b)\}, \{f(b) \approx g(c)\}, \{f(b) \approx h(f(b), c)\}\}.$$
2. We can also replace $f(b)$ with a. The result is the reduced formula
$$\varphi'' = \{\{a \approx f(b)\}, \{a \approx g(c)\}, \{a \approx h(a, c)\}\}.$$

8.1.1 Termination

Now we prove termination of the reduction system and of the corresponding GDPLL procedure.

In the following we use a definition of *non-propagated equalities*, i.e. equalities to which Rule 1 of the reduction system can be applied.

Definition 8.6 (**Non-propagated equalities**)

- *An equality $s \approx t$ is called non-propagated in a CNF φ if*
  - $\{s \approx t\} \in \varphi$,
  - $s, t \in \mathsf{SubTerm}(\varphi \backslash \{\{s \approx t\}\})$.
- *The set of all non-propagated equalities in φ is denoted by* $\mathsf{NPEq}(\varphi)$.

In Definition 8.3(1) the $s \approx t$ is non-propagated in $\{\{s \approx t\}\} \cup \varphi$, whereas the $s \approx t$ is propagated in $\{\{s \approx t\}\} \cup \varphi[s := t]$. Note that $s \notin \mathsf{SubTerm}(\varphi[s := t])$.

We start with a technical lemma which is used in Lemma 8.8 and in Theorem 8.10. Lemma 8.8 is also a technical lemma.

Lemma 8.7 *Suppose $s, t \in \mathsf{Term}$, where $s \notin \mathsf{SubTerm}(t)$, and T is a set of terms. Then*

$$|T \cup \{s, t\}| \geq |T[s := t] \cup \{s, t\}|.$$

Proof We can observe that for an arbitrary set of terms T' and arbitrary terms s', t' the following holds.

- $|T'| = |T'[s' := t']|$ if for all distinct $u, v \in T'$, $u[s' := t'] \not\equiv v[s' := t']$,
- $|T'| > |T'[s' := t]'|$ if there are distinct $u, v \in T$ such that $u[s' := t'] \equiv v[s' := t']$.

Hence,

$$\begin{array}{rcl} |T \cup \{s,t\}| & = & |(T\backslash\{s,t\}) \cup \{s,t\}| \\ & \geq & |(T\backslash\{s,t\})[s := t] \cup \{s,t\}| \\ & = & |T[s := t] \cup \{s,t\}| \end{array}$$

⊠

Lemma 8.8 *Suppose $\varphi, \varphi', \psi \in \mathsf{Cnf}$ and $\varphi \equiv \{\{s \approx t\}\} \uplus \varphi'$, where $s, t \in \mathsf{SubTerm}(\varphi')$ and $s \notin \mathsf{SubTerm}(t)$, and $\psi \equiv \{\{s \approx t\}\} \cup \varphi'[s := t]$. Then one of the following holds.*

1. *$|\mathsf{SubTerm}(\psi)| < |\mathsf{SubTerm}(\varphi)|$ or*
2. *$|\mathsf{SubTerm}(\psi)| \leq |\mathsf{SubTerm}(\varphi)|$ and $|\mathsf{NPEq}(\psi)| < |\mathsf{NPEq}(\varphi)|$.*

Proof

By Lemma 9.26, $|\mathsf{SubTerm}(\psi)| \leq |\mathsf{SubTerm}(\varphi)|$. Hence, it is sufficient to show that from $|\mathsf{NPEq}(\psi)| \geq |\mathsf{NPEq}(\varphi)|$ it follows that $|\mathsf{SubTerm}(\psi)| < |\mathsf{SubTerm}(\varphi)|$.

Suppose $|\mathsf{NPEq}(\psi)| \geq |\mathsf{NPEq}(\varphi)|$. Then there is some $C \in \varphi$ such that $C \notin \mathsf{NPEq}(\varphi)$ and $C[s := t] \in \mathsf{NPEq}(\psi)$.

Suppose $C \equiv \{u \approx v\}$, where u, v are terms. Since $C \notin \mathsf{NPEq}(\varphi)$ then at least one of u and v is not in $\mathsf{SubTerm}(\varphi \backslash \{C\})$. W.l.o.g. we can assume that $u \notin \mathsf{SubTerm}(\varphi \backslash \{C\})$.

We denote $T = \mathsf{SubTerm}(\varphi') \backslash \{u\}$. Since $C[s := t] \in \mathsf{NPEq}(\psi)$, $u[s := t] \in T[s := t]$. Hence, taking into account Lemma 9.26,

$$\begin{array}{rcl}
|\mathsf{SubTerm}(\psi)| & = & |\mathsf{SubTerm}(\varphi'[s := t] \cup \{\{s \approx t\}\})| \\
& = & |\mathsf{SubTerm}(\varphi'[s := t]) \cup \mathsf{SubTerm}(s \approx t)| \\
& = & |\mathsf{SubTerm}(T[s := t]) \cup \mathsf{SubTerm}(u[s := t]) \cup \mathsf{SubTerm}(s \approx t)| \\
& = & |\mathsf{SubTerm}(T[s := t]) \cup \mathsf{SubTerm}(s \approx t)| \\
& = & |\mathsf{SubTerm}(T[s := t] \cup \{s \cup t\})| \\
& \leq & |\mathsf{SubTerm}(T \cup s \cup t)| \\
& < & |\mathsf{SubTerm}(T \cup u \cup s \cup t)| \\
& = & |\mathsf{SubTerm}(\varphi' \cup \{\{s \approx t\}\})| \\
& = & |\mathsf{SubTerm}(\varphi)|
\end{array}$$

⊠

Definition 8.9 *To each $\varphi \in$ Cnf we relate a pair of numbers, using $\mathsf{norm}(\varphi)$ as below:*

$$\mathsf{norm}(\varphi) = (|\mathsf{SubTerm}(\varphi)| + |\mathsf{Lit}(\varphi)|, |\mathsf{NPEq}(\varphi)|).$$

Theorem 8.10 *The reduction system* Reduce *is terminating.*

Proof We prove termination of the reduction system Reduce by showing that after applying each step of the reduction system on a formula, $\mathsf{norm}(\varphi)$ decreases with respect to the lexicographic order $\prec_{lex}$ on pairs. Suppose $\varphi \to \psi$.

1. Suppose $\varphi \equiv \{\{s \approx t\}\} \uplus \varphi'$, where $s, t \in \mathsf{SubTerm}(\varphi')$ and $s \notin \mathsf{SubTerm}(t)$, and $\psi \equiv \{\{s \approx t\}\} \uplus \varphi'[s := t]$.

   Taking into account Lemma 9.26, we conclude

   $$\begin{array}{rcl}
|\mathsf{Lit}(\psi)| & = & |\mathsf{Lit}(\varphi'[s := t] \cup \{\{s \approx t\}\})| \\
& \leq & |\mathsf{Lit}(\varphi' \cup \{\{s \approx t\}\})| \\
& = & |\mathsf{Lit}(\varphi)|
\end{array}$$

   By Lemma 8.8, either $|\mathsf{SubTerm}(\psi)| < |\mathsf{SubTerm}(\varphi)|$ or $|\mathsf{SubTerm}(\psi)| = |\mathsf{SubTerm}(\varphi)|$ and $|\mathsf{NPEq}(\psi)| < |\mathsf{NPEq}(\varphi)|$.

We obtain, that either $|\mathsf{SubTerm}(\psi)| + |\mathsf{Lit}(\psi)| < |\mathsf{SubTerm}(\varphi)| + |\mathsf{Lit}(\varphi)|$ or $|\mathsf{SubTerm}(\psi)|+|\mathsf{Lit}(\psi)| = |\mathsf{SubTerm}(\varphi)|+|\mathsf{Lit}(\varphi)|$ and $|\mathsf{NPEq}(\psi)| < |\mathsf{NPEq}(\varphi)|$.

2. Suppose $\psi \equiv \varphi\downarrow$.

   Then

$$\begin{array}{rcl} |\mathsf{Lit}(\psi)| & \leq & |\mathsf{Lit}(\varphi)| - |\{t \approx t \mid (t \approx t) \in \mathsf{Lit}(\varphi)\}| - |\{t \not\approx t \mid (t \not\approx t) \in \mathsf{Lit}(\varphi)\}| \\ & < & |\mathsf{Lit}(\varphi)| \end{array}$$

   Obviously, $\mathsf{SubTerm}(\psi) \leq \mathsf{SubTerm}(\varphi)$. We conclude, $|\mathsf{SubTerm}(\psi)|+|\mathsf{Lit}(\psi)| < |\mathsf{SubTerm}(\varphi)| + |\mathsf{Lit}(\varphi)|$.

Hence, $\psi \prec_{lex} \varphi$. ⊠

We have proved that the reduction system Reduce is terminating.

8.2 Satisfiability criterion

In this section we will consider conditions under which an EUF-CNF is satisfiable. These conditions will form a basis for the function SatCriterion.

Definition 8.11 (**The core of a CNF**) *Let $\varphi \in \mathsf{Cnf}$. Then the set of positive clauses of length more than one contained in φ is called the* core *of φ and denoted by $\mathsf{Core}(\varphi)$.*

Let φ be a reduced CNF not containing the empty clause; $\mathsf{Core}(\varphi) = \emptyset$. We will give a proof that such a CNF φ is satisfiable.

Consider a reduced CNF φ such that $\mathsf{Core}(\varphi) = \emptyset$. Since $\mathsf{Core}(\varphi) = \emptyset$ every clause of length more than one contains at least one negative literal. Let $\psi \in \mathsf{Cnf}$ be obtained from φ by removing from all clauses of length more than one all literals except one negative literal. Trivially, if $[\![\psi]\!]_{\mathcal{D}} = \mathsf{true}$ for some structure $\mathcal{D}$ then $[\![\varphi]\!]_{\mathcal{D}} = \mathsf{true}$, i.e. φ is satisfiable if ψ is satisfiable. It means that w.l.o.g. we can restrict ourself to the case when a CNF contains only unit clauses.

The set of CNFs containing only unit clauses is denoted by UCnf.

At first we introduce two binary relations on the set of terms contained in φ.

Definition 8.12 *Let $\varphi \in \mathsf{UCnf}$. The binary relation $\sim_\varphi$ is the smallest relation over $\mathsf{Term}(\varphi) \times \mathsf{Term}(\varphi)$ such that:*

1. *$s \sim_\varphi t$, if $\{s \approx t\} \in \varphi$.*
2. *$\sim_\varphi$ is reflexive, symmetric, and transitive.*

Definition 8.13 *The binary relation $\cong_\varphi$ is the smallest relation over $\mathsf{SubTerm}(\varphi) \times \mathsf{SubTerm}(\varphi)$ such that:*

1. *$s \cong_\varphi t$, if $\{s \approx t\} \in \varphi$.*
2. *$f(s_1, \ldots, s_n) \cong_\varphi f(t_1, \ldots, t_n)$, if $s_i \cong_\varphi t_i$, $1 \leq i \leq n$, and $f(s_1, \ldots, s_n)$, $f(t_1, \ldots, t_n) \in \mathsf{SubTerm}(\varphi)$.*
3. *$\cong_\varphi$ is reflexive, symmetric, and transitive.*

We will prove a lemma stating that for reduced CNFs the introduced binary relations are equivalent.

Lemma 8.14 *Let $\varphi \in \mathsf{UCnf}$ be reduced. Then for each $s, t \in \mathsf{Term}$*

$$s \sim_\varphi t \text{ if and only if } s \cong_\varphi t.$$

Proof

($\Rightarrow$) Suppose $s \sim_\varphi t$. Then by Definitions 8.12 and 8.13, we obtain $s \cong_\varphi t$.

($\Leftarrow$) Suppose $s \cong_\varphi t$. We give a proof by contradiction.
Assume that

$$s \not\sim_\varphi t.$$

It means that Condition (2) of Definition 8.13 was applied at least one time.

W.l.o.g. we can assume that we applied the condition one time. Then there are

$$f(s_1, \ldots, s_n), f(t_1, \ldots, t_n) \in \mathsf{SubTerm}(\psi)$$

such that

$$s_i \sim_\varphi t_i, 1 \leq i \leq n.$$

In this case there are $u_0, \ldots, u_n \in \mathsf{Term}(\varphi)$ such that

$$\{s_i \equiv u_0 \;\approx\; u_1\}, \{u_1 \;\approx\; u_2\}, \ldots, \{u_{n-1} \;\approx\; u_n \equiv t_n\} \in \varphi,$$

where $i \in \{1, \ldots, n\}$.

In this case Rule 1 of the reduction system Reduce would be applicable. This contradicts that φ is reduced. We obtain that

$$s \sim_\varphi t.$$

⊠

Lemma 8.15 *Suppose* $\varphi \in \mathsf{UCnf}$, φ *is reduced and* $\{s \not\approx t\} \in \varphi$ *for some* $s, t \in \mathsf{Term}$. *Then* $s \not\cong_\varphi t$.

Proof At first we prove by contradiction that $s \not\sim_\varphi t$.

Assume that

$$s \sim_\varphi t.$$

Then one of the following holds.

- $s \equiv t$. Then $\{s \not\approx s\} \in \varphi$. In this case Rule 2 can be applied. This contradicts that φ is reduced.
- There are $u_0, \ldots, u_n \in \mathsf{Term}(\varphi)$ such that

  $$\{s \equiv u_0 \ \approx \ u_1\}, \{u_1 \ \approx \ u_2\}, \ldots, \{u_{n-1} \ \approx \ u_n \equiv t\} \in \varphi.$$

  Then Rule 1 can be applied. This contradicts that φ is reduced.

We can conclude that

$$s \not\sim_\varphi t.$$

By Lemma 8.14 we obtain that

$$s \not\cong_\varphi t.$$

⊠

Theorem 8.16 $\varphi \in \mathsf{UCnf}$ *is unsatisfiable if and only if there exist* $s, t \in \mathsf{Term}(\varphi)$ *such that*

$$\{s \not\approx t\} \in \varphi \text{ and } s \cong_\varphi t.$$

Proof See [Shostak, 1978]. ⊠

Theorem 8.17 (**Satisfiability criterion**) *Suppose φ is a reduced CNF, $\bot \notin \varphi$, and $\mathsf{Core}(\varphi) = \emptyset$. Then φ is satisfiable.*

Proof From the assumption of the theorem we conclude that every clause of length more than one contains at least one negative literal. Let $\psi \in \mathsf{Cnf}$ be obtained from φ by removing from all clauses all literals except one negative literal. Hence, ψ is reduced by construction. By Lemma 8.15 and Theorem 8.16 ψ is satisfiable. Then $[\![\psi]\!]_{\mathcal{D}} = \mathsf{true}$ for some structure $\mathcal{D}$. One can see, that $[\![\varphi]\!]_{\mathcal{D}} = \mathsf{true}$ for the same structure $\mathcal{D}$. We can conclude that φ is satisfiable.

⊠

8.3 The GDPLL building blocks for the EUF-logic

We now come to the definition of the building blocks for the GDPLL procedure.

The procedure GDPLL for the EUF-logic invokes the following functions.

- We define the function $\mathsf{Reduce}(\varphi)$ to be any normal form of φ with respect to the reduction system Reduce.
- For each $\varphi \in \mathsf{R}$, the function SatCriterion() is defined as follows:

$$\mathsf{SatCriterion}(\varphi) = \begin{cases} \mathsf{true} & \text{if } \mathsf{Core}(\varphi) = \emptyset, \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

- For each $\varphi \in \mathsf{R}$, the function Eligible() is defined as below:

$$\mathsf{Eligible}(\varphi) = \mathsf{Lit}(\mathsf{Core}(\varphi)).$$

- For each $\varphi \in \mathsf{R}$ and for each $l \in \mathsf{Eligible}(\varphi)$, the function Filter() is defined as

$$\mathsf{Filter}(\varphi, l) = \{\{l\}\} \cup \varphi|_l \text{ and } \mathsf{Filter}(\varphi, \neg l) = \{\{\neg l\}\} \cup \varphi|_{\neg l}.$$

Example 8.18 As an example we consider the formula originating from the verification of translators (compilers, code generators) [Pnueli et al., 1999], where concrete functions have been replaced by uninterpreted function symbols.

$$\varphi_0 \equiv \{\{u_1 \approx f(x_1, y_1)\}, \{u_2 \approx f(x_2, y_2)\}, \{z \approx g(u_1, u_2)\},$$

$$\{z \not\approx g(f(x_1, y_1), f(x_2, y_2))\}\}.$$

After applying Rule 1, we obtain

$$\varphi_1 \equiv \{\{u_1 \approx f(x_1, y_1)\}, \{u_2 \approx f(x_2, y_2)\}, \{z \approx g(u_1, u_2)\}, \{z \not\approx g(u_1, f(x_2, y_2))\}\},$$

$$\varphi_2 \equiv \{\{u_1 \approx f(x_1, y_1)\}, \{u_2 \approx f(x_2, y_2)\}, \{z \approx g(u_1, u_2)\}, \{z \not\approx g(u_1, u_2)\}\},$$

$$\varphi_3 \equiv \{\{u_1 \approx f(x_1, y_1)\}, \{u_2 \approx f(x_2, y_2)\}, \{z \approx g(u_1, u_2)\}, \{z \not\approx z\}\}.$$

After applying Rule 2, we obtain

$$\varphi_4 \equiv \{\{u_1 \approx f(x_1, y_1)\}, \{u_2 \approx f(x_2, y_2)\}, \{z \approx g(u_1, u_2)\}, \perp\}.$$

Since $\perp \in \varphi_4$, φ_4 is unsatisfiable and therefore φ_0 is unsatisfiable.

Therefore, the function Reduce() is defined by means of reduction rules, that can be applied in any order. The reduction rules allow to replace equals for equals, and simplifying a formula by removing all equalities of the form $t \approx t$. So, all work specific for the EUF-logic is done by the function Reduce.

The function Eligible() allows us to choose literals from the purely positive clauses of length more than one, i.e. from the core of a formula. Hence, we may terminate with SAT as soon as the core of a reduced formula is empty and the formula does not contain the empty clause.

In Example 8.19 we show that choosing to split on an arbitrary positive literal can lead to a non-terminating derivation.

Example 8.19 The figure 8.1(a) shows sample derivations from a CNF $\varphi_0 \equiv \{\{a \approx b, b \approx c\}, \{f(a) \approx f(c)\}\}$ in tree notation. Splitting on a literal $f(a) \approx f(c)$ can lead to a non-terminating derivation since for a positive branch φ_0 is derived. For a negative branch the CNF $\varphi_1 \equiv \{\{a \approx b\}, \{b \approx c\}\}$ is derived.

Splitting on a literal contained in Core(φ) leads to a terminating derivation. For the given CNF φ_0, Core(φ_0) $= \{a \approx b, b \approx c\}$. A terminating derivation is depicted in Figure 8.1(b). After splitting on a literal $a \approx b$, for a positive branch a reduced CNF $\varphi_2 \equiv \{\{a \approx b\}, \{f(a) \approx f(c)\}\}$ is derived and for a negative branch a reduced CNF $\varphi_3 \equiv \{\{b = c\}, \{f(a) \approx f(c)\}\}$ is derived. Both φ_2 and φ_3 are satisfiable according Theorem 8.17.

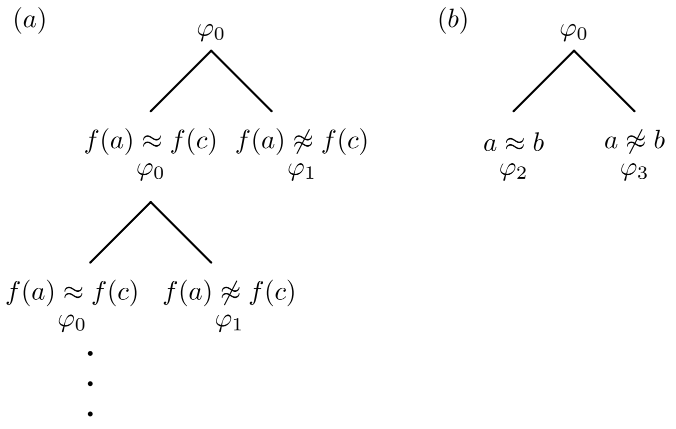

Figure 8.1 a) An example of a non-terminating derivation $\varphi_0 \equiv \{\{a \approx b, b \approx c\}, \{f(a) \approx f(c)\}\}$. b) An example of a terminating derivation $\varphi_0 \equiv \{\{a \approx b, b \approx c\}, \{f(a) \approx f(c)\}\}$

8.4 Soundness and completeness

In this section we prove that the GDPLL procedure for the EUF-logic is sound and complete. One can see that the rules of the reduction system Reduce preserve (un)satisfiability of a formula.

Lemma 8.20 *Let $\varphi \in$ Cnf. Then φ is satisfiable if and only if* Reduce(φ) *is satisfiable.*

Proof We show that every reduction step preserves (un)satisfiability. So, let $\varphi \to \psi$.

($\Rightarrow$) Suppose φ is satisfiable. Let $\mathcal{D}$ be an arbitrary structure such that $[\![\varphi]\!]_{\mathcal{D}} =$ true.

1. Suppose $\varphi \equiv \{\{s \approx t\}\} \uplus \varphi'$, where $s, t \in \mathsf{SubTerm}(\varphi')$ and $s \notin \mathsf{SubTerm}(t)$, and $\psi \equiv \{\{s \approx t\}\} \uplus \varphi'[s := t]$.

   Taking into account $[\![s \approx t]\!]_{\mathcal{D}} =$ true, we obtain $[\![s]\!]_{\mathcal{D}} = [\![t]\!]_{\mathcal{D}}$, and as a result $[\![\varphi'[s := t]]\!]_{\mathcal{D}} =$ true. Hence, $[\![\psi]\!]_{\mathcal{D}} =$ true.

2. Let $\psi \equiv \varphi \downarrow$.

   Hence, $\psi = \{C \downarrow \mid (C \in \varphi) \wedge (\forall t \in \mathsf{Term} : (t \approx t) \notin \mathsf{C})\}$. Then for each $C \in \varphi$, $[\![C]\!]_{\mathcal{D}} =$ true. Since for each $t \in$ Term, $[\![t \not\approx t]\!]_{\mathcal{D}} =$ false, we obtain that $[\![C \downarrow]\!]_{\mathcal{D}} =$ true. Therefore, $[\![\psi]\!]_{\mathcal{D}} =$ true.

($\Leftarrow$) Suppose ψ is satisfiable, and $\mathcal{D}$ is a structure such that $[\![\psi]\!]_{\mathcal{D}} =$ true.

1. Suppose $\varphi \equiv \{\{s \approx t\}\} \uplus \varphi'$, where $s, t \in \mathsf{SubTerm}(\varphi')$ and $s \notin \mathsf{SubTerm}(t)$, and $\psi \equiv \{\{s \approx t\}\} \uplus \varphi'[s := t]$.

   Taking into account $[\![s \approx t]\!]_{\mathcal{D}} = \mathsf{true}$, we obtain $[\![s]\!]_{\mathcal{D}} = [\![t]\!]_{\mathcal{D}}$, and as a result $[\![\varphi']\!]_{\mathcal{D}} = \mathsf{true}$. Hence, $[\![\varphi]\!]_{\mathcal{D}} = \mathsf{true}$.

2. Assume $\psi \equiv \varphi \downarrow$. Then for each $C \in \varphi$ one of the following holds.

   - $C \subseteq D$, where $D \in \psi$. Since $[\![\psi]\!]_{\mathcal{D}} = \mathsf{true}$ then $[\![D]\!]_{\mathcal{D}} = \mathsf{true}$. Hence, $[\![C]\!]_{\mathcal{D}} = \mathsf{true}$.
   - $(t \approx t) \in C$, where $t \in \mathsf{Term}$. Since for each $t \in \mathsf{Term}$, $[\![t \approx t]\!]_{\mathcal{D}} = \mathsf{true}$ then $[\![C]\!]_{\mathcal{D}} = \mathsf{true}$.

   Since for each $C \in \varphi$, $[\![C]\!]_{\mathcal{D}} = \mathsf{true}$, we conclude that $[\![\varphi]\!]_{\mathcal{D}} = \mathsf{true}$.

$\boxtimes$

Lemma 8.21 *Let $\varphi \in \mathsf{Cnf}$, and $s, t \in \mathsf{Term}$. Then φ is unsatisfiable iff both $\{\{s \approx t\}\} \cup \varphi|_{s\approx t}$ and $\{\{s \not\approx t\}\} \cup \varphi|_{s\not\approx t}$ are unsatisfiable.*

Proof (⇒) Let $\mathcal{D}$ be an arbitrary structure. Since φ is unsatisfiable, $[\![\varphi]\!]_{\mathcal{D}} = \mathsf{false}$. Hence, $[\![\{\{s \approx t\}\} \cup \varphi]\!]_{\mathcal{D}} = \mathsf{false}$ and $[\![\{\{s \not\approx t\}\} \cup \varphi]\!]_{\mathcal{D}} = \mathsf{false}$.

W.l.o.g. we can consider the case when $[\![s \approx t]\!]_{\mathcal{D}} = \mathsf{true}$. In this case $[\![\{\{s \not\approx t\}\} \cup \varphi|_{s\not\approx t}]\!]_{\mathcal{D}} = \mathsf{false}$.

Consider $\{\{s \approx t\}\} \cup \varphi|_{s\approx t}$. Since $[\![s \approx t]\!]_{\mathcal{D}} = \mathsf{true}$, there is some $C \in \varphi$ such that $C \backslash \{s \not\approx t\} = \mathsf{false}$. Hence, $[\![\{\{s \approx t\}\} \cup \varphi|_{s\approx t}]\!]_{\mathcal{D}} = \mathsf{false}$.

(⇐) Let $\mathcal{D}$ be an arbitrary structure. Since both $\{\{s \approx t\}\} \cup \varphi|_{s\approx t}$ and $\{\{s \not\approx t\}\} \cup \varphi|_{s\not\approx t}$ are unsatisfiable, $[\![\{\{s \approx t\}\} \cup \varphi|_{s\approx t}]\!]_{\mathcal{D}} = \mathsf{false}$ and $[\![\{\{s \approx t\}\} \cup \varphi|_{s\approx t}]\!]_{\mathcal{D}} = \mathsf{false}$.

By definition, either $[\![s \approx t]\!]_{\mathcal{D}} = \mathsf{true}$ or $[\![s \not\approx t]\!]_{\mathcal{D}} = \mathsf{true}$. W.l.o.g. we can assume that $[\![s \approx t]\!]_{\mathcal{D}} = \mathsf{true}$. Therefore, $[\![\varphi]\!]_{\mathcal{D}} = \mathsf{false}$, and φ is unsatisfiable.

$\boxtimes$

Theorem 8.22 (**Soundness and Completeness**) *A CNF φ is unsatisfiable if and only if the* $\mathsf{GDPLL}(\varphi)$ *returns "unsatisfiable".*

Proof In order to apply Theorem 5, we have to check the following properties.

Properties 1 and 2 have been proved in Lemma 8.20 and in Lemma 8.21. Properties 4 and Property 5 have been proved in Theorem 8.17.

To prove Property 3 we define a well-founded order $\prec$ as follows. For all $\varphi, \psi \in$ $\mathsf{Reduce}(\mathsf{Cnf})$ $\psi_1 \prec \psi_2$ if $\sum_{C \in \mathsf{Core}(\psi_1)} |C| < \sum_{C \in \mathsf{Core}(\psi_2)} |C|$.

By definition of the function $\mathsf{Filter}()$, for all $l \in \mathsf{Eligible}(\varphi)$

$$\mathsf{Core}(\mathsf{Filter}(\varphi, l)) \equiv \{C \in \mathsf{Core}(\varphi) |\ l \notin C\}.$$

By definition of the function $\mathsf{Eligible}()$, there is some $C \in \mathsf{Core}(\varphi)$ such that $l \in C$. Hence, for every $l \in \mathsf{Eligible}(\varphi)$, $\mathsf{Filter}(\varphi, l) \prec \varphi$.

Consider $\mathsf{Filter}(\varphi, \neg l)$. By definition of the function $\mathsf{Filter}()$, for all $l \in \mathsf{Eligible}(\varphi)$

$$\mathsf{Core}(\mathsf{Filter}(\varphi, \neg l)) \equiv \{C \backslash \{l\} |\ C \in \mathsf{Core}(\varphi)\}$$

Since there is some $C \in \mathsf{Core}(\varphi)$ such that $l \in C$, we conclude that for every $l \in \mathsf{Eligible}(\varphi)$, $\mathsf{Filter}(\varphi, \neg l) \prec \varphi$.

⊠

8.5 The EUF-DPLL calculus

In this section we describe the EUF-DPLL calculus. The GDPLL procedure for the EUF-logic can also be presented as a calculus, whose rules are depicted in Figure 8.2. The three essential operations of the method are: equality propagation, tautology atom removing and splitting.

The approach can be described as follows. Given a CNF φ, apply *equality propagation* to non-propagated equalities. Moreover, perform *tautology atom removing*, i.e. (1) delete all clauses containing a literal of the shape $t \approx t$, and (2) remove literals of the shape $t \not\approx t$ from all clauses. If the normal form φ^* contains the empty clause then backtrack. If φ^* is a reduced CNF not containing the empty clause and $\mathsf{Core}(\varphi^*)$ is an empty set then terminate with the answer "satisfiable". If none of the above situations occur then choose an arbitrary literal l from $\mathsf{Core}(\varphi^*)$ and check the satisfiability of $\varphi^* \cup \{\{l\}\}$ and $\varphi^* \cup \{\{\neg l\}\}$ separately.

The *equality propagation* rule models the substitutivity of equality. The *tautology atom removing* allows to remove clauses which are trivially satisfiable and to delete literals which are unsatisfiable for any structure. The *splitting* rule decomposes a problem into two smaller problems. The splitting rule is the only non-deterministic rule in the calculus.

We would like to emphasize that this set of rules is minimal from the point of completeness and it cannot be reduced.

Equality propagation: $\dfrac{\{\{s \approx t\}\} \cup \varphi}{\{\{s \approx t\}\} \cup \varphi[t := s]}$ if $\varphi \in \mathsf{Cnf}$; $s, t \in \mathsf{Term}(\varphi)$; $s \notin \mathsf{SubTerm}(t)$

Tautology atom removing: $\dfrac{\varphi}{\varphi|_{t \approx t}}$ if $\varphi \in \mathsf{Cnf}$; $(t \approx t) \in \mathsf{Eq}(\varphi)$

Splitting: $\dfrac{\varphi}{\{\{l\}\} \cup \varphi \mid \{\{\neg l\}\} \cup \varphi}$ if $l \in \mathsf{Lit}(\mathsf{Core}(\varphi))$

Figure 8.2 The rules of EUF-DPLL calculus

Unit propagation in the calculus is achieved by the combined use of the equality propagation rule and the tautology atom removing rule.

Example 8.23 Consider the formula

$$\varphi \equiv \{\{a \approx b\}, \{f(c) \not\approx (c), a \not\approx b\}\}.$$

The EUF-calculus first implements unit propagation by applying the equality propagation rule

$$\varphi \equiv \{\{a \approx b\}, \{f(c) \not\approx (c), b \not\approx b\}\},$$

and subsequently the tautology atom removing rule

$$\varphi \equiv \{\{a \approx b\}, \{f(c) \not\approx (c)\}.$$

The crucial difference with the propositional DPLL method is that in the propositional case given a formula $\{\{l\}\} \cup \varphi$, a literal l can be assigned to true and then $\{\{l\}\} \cup \varphi$ can be replaced by $\varphi|_l$. In case of the EUF logic such an operation can violate the satisfiability of a formula.

Example 8.24 Consider the formula

$$\varphi \equiv a \approx b \wedge f(a) \not\approx f(b).$$

Unit propagation implemented as in the propositional case does not maintain (un)satisfiability: the formula φ is unsatisfiable, but the formula

$$\varphi' \equiv f(a) \not\approx f(b)$$

is satisfiable.

The crucial difference with superposition-based calculi is that no reduction ordering plays a role when substituting a term with another term. Reduction orderings on terms are important for termination of superposition-based calculi, i.e. every clause derived via the rules of superposition-based calculi is smaller than the maximal 'parent' clause. We have proved termination of the approach in case of a single restriction: while replacing a term t with a term s, s cannot be a subterm of t.

The EUF-DPLL calculus constructs a derivation tree.

Definition 8.25 (**An EUF-DPLL derivation tree**)

- *An EUF-DPLL derivation tree, or for simplicity a derivation tree, is a labelled tree such that the following holds:*
  - *every node is labelled with a CNF;*
  - *every non-leaf node has one or two successors;*
  - *for each non-leaf node, the CNFs labelling its successor nodes are obtained by applying one of the rules of the EUF-DPLL calculus.*
- *We say that a derivation tree is a derivation tree of φ if the root node is labelled with φ.*

Definition 8.26 (**A successful branch, a failed branch**)

- *A branch in an EUF-DPLL derivation tree is successful if its leaf is labelled with a CNF φ such that* $\mathsf{SatCriterion}(\varphi) = \mathsf{true}$*;*
- *A branch in an EUF-DPLL derivation tree is failed if its leaf is labelled with a CNF φ such that $\bot \in \varphi$.*

Definition 8.27 (**An EUF-DPLL refutation tree**) *An EUF-DPLL derivation tree is called a refutation tree if each of its leaves is labelled with a CNF φ such that $\bot \in \varphi$.*

The purpose of the EUF-DPLL calculus is starting from a CNF φ to derive a CNF ψ such that $\mathsf{SatCriterion}(\psi) = \mathsf{true}$. In this case φ is satisfiable. Otherwise φ is unsatisfiable.

Depending on the strategy how to apply the rules of the EUF-DPLL calculus, for every CNF φ more than one derivation tree can be constructed. But the EUF-DPLL calculus is terminating in the sense that every finite derivation tree of a CNF φ is finite.

Lemma 8.28 *Suppose φ is an unsatisfiable CNF. Then each of its derivation trees is a refutation tree.*

Proof It follows from Lemma 8.20 and Lemma 8.21. ⊠

Theorem 8.29 (**Soundness and completeness**) *The* EUF-DPLL *calculus is sound and complete.*

Proof It follows immediately from Theorem 8.22. ⊠

8.6 The optimization of the reduction system

DPLL-based systems are really efficient only in combination with good optimization strategies. Modern implementations use a number of optimizing techniques to choose literals for branching and to perform unit resolution. Not all approaches suitable for propositional logic work automatically for the EUF logic. Some techniques might become incorrect, and others being still correct, may lose their efficiency. Analysis which of the optimizations are still effective for the case of the EUF logic is beyond the scope of this thesis.

In this section we introduce techniques which can be used either as a preprocessing step or while executing the procedure.

Definition 8.30 (**A reducible term, a fresh constant**)

- *A term t is called* reducible *in φ if the following holds.*
  - $t \in \mathsf{SubTerm}(\varphi)$,
  - $\mathsf{depth}(t) > 1$,
  - *if a term $s \in \mathsf{SubTerm}_p(t)$ then $s \notin \mathsf{Term}(\varphi)$.*
- *A constant a is called* fresh *in φ if $a \notin \mathsf{Const}(\varphi)$.*

Example 8.31 *Let us consider the formula from Example 8.18. The terms $f(x_1, y_1)$ and $f(x_2, y_2)$ are reducible.*

Definition 8.32 (**Optimized reduction rules on CNFs**) *We define an optimized reduction system* Opt-Reduce *as follows.*

1. $\{\{s \approx t\}\} \uplus \varphi \to \{\{s \approx t\}\} \uplus \varphi[s := t]$ *if* $s, t \in \mathsf{SubTerm}(\varphi)$ *and* $s \notin \mathsf{SubTerm}(t)$.
2. $\varphi \to \varphi\downarrow$.
3. $\varphi \to \varphi[t := a]$ *if* t *is a reducible term in* φ *and* a *is a fresh constant in* φ.
4. $\{\{a \approx b\}\} \uplus \varphi \to \varphi[a := b]$ *if* $a, b \in \mathsf{Const}(\varphi)$.

Example 8.33 We consider the formula from Example 8.18.

$$\varphi_0 : \{\{u_1 \approx f(x_1, y_1)\}, \{u_2 \approx f(x_2, y_2)\}, \{z \approx g(u_1, u_2)\},$$

$$\{z \not\approx g(f(x_1, y_1), f(x_2, y_2))\}\}.$$

After applying the term reduction rule, we obtain

$$\varphi_1 : \{\{u_1 \approx v_1\}, \{u_2 \approx v_2\}, \{z \approx g(u_1, u_2)\}, \{z \not\approx g(v_1, v_2)\}\}$$

where a term $f(x_1, y_1)$ is replaced by a fresh variable v_1 and a term $f(x_2, y_2)$ is replaced by a fresh variable v_2.

After applying the equality propagation rule the equality elimination rule, we obtain

$$\varphi_2 : \{\{u_2 \approx v_2\}, \{z \approx g(u_1, u_2)\}, \{z \not\approx g(u_1, v_2)\}\}$$

$$\varphi_3 : \{\{z \approx g(u_1, u_2)\}, \{z \not\approx g(u_1, u_2)\}\}$$

$$\varphi_4 : \{\{z \approx g(u_1, u_2)\}, \{z \not\approx z\}\}.$$

After applying the tautology atoms removing rule, we obtain

$$\varphi_5 : \{\{z \approx g(u_1, u_2)\}, \{\bot\}\}.$$

8.6.1 Soundness and completeness of the optimized procedure

Given a CNF φ, we have to prove that $\mathsf{Opt\text{-}Reduce}(\varphi)$ preserves the (un)satisfiability of φ.

Lemma 8.34 *Suppose for some $\varphi \in \mathsf{Cnf}$*

- *t is a reducible term in φ,*
- *a is a fresh constant in φ.*

Then φ is satisfiable if and only if $\varphi[t := a]$ is satisfiable.

Proof Suppose

$$t \equiv f(t_1, \ldots, t_n),$$

where $f \in \mathsf{Fun}$, $\mathsf{ar}(f) = n$, $t_1, \ldots, t_n \in \mathsf{Term}$.
Suppose

$$\mathsf{Const}(t) = \{a_1, \ldots, a_m\}.$$

Let D be a domain and $\mathcal{D}$ be a structure.
Suppose

$$D^* = \{d_1^*, \ldots, d_m^*\}$$

and

$$D^* \cap D = \emptyset.$$

We assume that

$$\overline{D} = D \cup D^*.$$

We define a function $\epsilon : \overline{D} \Rightarrow D$ as follows.

$$\epsilon(d) = \begin{cases} d, \text{if } d \in D, \\ [\![a_i]\!]_{\mathcal{D}}, \text{if } d = d_i^* \text{ for some } i \in \{1, \ldots, n\}. \end{cases}$$

For each term s we define an interpretation $[\![s]\!]_{\overline{\mathcal{D}}}$ as follows.

- $[\![a]\!]_{\overline{\mathcal{D}}} = [\![t]\!]_{\mathcal{D}}$,
- for each $a_i \in \mathsf{Const}(t)$, $1 \leq i \leq m$,

$$[\![a_i]\!]_{\overline{\mathcal{D}}} = d_i^*,$$

- for each constant b such that $b \notin \mathsf{Const}(t) \cup \{a\}$,

$$[\![b]\!]_{\overline{\mathcal{D}}} = [\![b]\!]_{\mathcal{D}},$$

- for each term $d(s_1, \ldots, s_r)$,

$$[\![g(s_1, \ldots, s_r)]\!]_{\overline{\mathcal{D}}} = \begin{cases} [\![t]\!]_{\mathcal{D}}, \text{if } g(t_1, \ldots, t_m) \equiv t, \\ g_D(\epsilon([\![s_1]\!]_{\overline{\mathcal{D}}}, \ldots, \epsilon([\![s_r]\!]_{\overline{\mathcal{D}}}), \text{otherwise.} \end{cases}$$

Let a function $P : \mathsf{Term} \to \mathsf{Term}$ be inductively defined as follows.

- $P(b) = b$, if $b \in \mathsf{Const} \backslash \{a\}$,
- $P(t) = a$,
- $P(g(s_1, \ldots, s_r) = g(P(s_1), \ldots, P(s_r))$ in other cases.

We will show that

$$[\![s]\!]_{\overline{\mathcal{D}}} = \begin{cases} d_i^*, \text{ if } s \equiv a_i, 1 \leq i \leq m, \\ [\![P(s)]\!]_{\mathcal{D}}, \text{ in other cases.} \end{cases} \tag{8.1}$$

We prove Statement 8.1 by induction on the structure of s.

Base case. Let $s \equiv b$, where $b \in \mathsf{Const}$. It holds by definition of $[\![s]\!]_{\overline{\mathcal{D}}}$.

Inductive step. Let it hold for some $s_1, \ldots, s_r \in \mathsf{Term}$. Suppose

$$s \equiv g(s_1, \ldots, s_r), \text{ where } g \in \mathsf{Fun} \text{ and } \mathsf{ar}(g) = r.$$

Then one of the following holds.

- $g(s_1, \ldots, s_r) \equiv f(t_1, \ldots, t_n)$. In this case Statement 8.1 holds by definition.

- $g(s_1, \ldots, s_r) \not\equiv f(t_1, \ldots, t_n)$. Then we have

$$\begin{aligned} [\![s]\!]_{\overline{\mathcal{D}}} &= [\![g(s_1, \ldots, s_r)]\!]_{\overline{\mathcal{D}}} \\ &= g_D(\epsilon([\![s_1]\!]_{\overline{\mathcal{D}}}), \ldots, \epsilon([\![s_r]\!]_{\overline{\mathcal{D}}})) \\ &= g_D([\![P(s_1)]\!]_{\mathcal{D}}, \ldots, [\![P(s_r)]\!]_{\mathcal{D}}) \\ &= [\![s]\!]_{\mathcal{D}}. \end{aligned}$$

We have proved Statement 8.1.

Now we extend $P(\varphi)$ inductively for formulae as follows.

- $P(s_1 \approx s_2) \;\equiv\; (P(s_1) \approx P(s_2))$,
- $P(s_1 \not\approx s_2) \;\equiv\; (P(s_1) \not\approx P(s_2))$,
- $P(\{l_1, \ldots, l_r\}) = \{P(l_1), \ldots, P(l_r)\}$,
- $P(\{C_1, \ldots, C_r\}) = \{P(C_1, \ldots, P(C_r)\}$.

Since for arbitrary $s_1, s_2 \in \mathsf{SubTerm}(\varphi)$

$$s_1, s_2 \notin \mathsf{Term}(\varphi)$$

then by Statement 8.1 and by the definition of P we obtain that for each $l \in \mathsf{Lit}(\varphi)$

$$[\![l]\!]_{\overline{\mathcal{D}}} = [\![P(l)]\!]_{\mathcal{D}},$$

for each $C \in \varphi$

$$[\![C]\!]_{\overline{\mathcal{D}}} = [\![P(C)]\!]_{\mathcal{D}},$$

for the formula φ

$$[\![\varphi]\!]_{\overline{\mathcal{D}}} = [\![P(\varphi)]\!]_{\mathcal{D}}.$$

It follows from the definition of $\varphi[t := a]$ and P that

$$\varphi[t := a] = P(\varphi).$$

Since by the theorem conditions if $s \in \mathsf{SubTerm}_p(t)$ then $s \notin \mathsf{Term}(\varphi)$ we can conclude that

$$[\![\varphi[t := a]]\!]_{\overline{\mathcal{D}}} = [\![\varphi]\!]_{\mathcal{D}}.$$

We obtain that $\varphi[t := a]$ is satisfiable if and only if φ is satisfiable. $\boxtimes$

Lemma 8.35 *Let $\varphi \in \mathsf{Cnf}$. Then φ is satisfiable if and only if* $\mathsf{Opt\text{-}Reduce}(\varphi)$ *is satisfiable.*

Proof

We show that every reduction step preserves (un)satisfiability. Since we have proved in Lemma 8.20 that step 1 and step 2 preserve (un)satisfiability, we have to prove the lemma only for step 3 and step 4. So, let $\varphi \to \psi$.

($\Rightarrow$) Suppose φ is satisfiable. Let $\mathcal{D}$ be an arbitrary structure such that $[\![\varphi]\!]_{\mathcal{D}} = \mathsf{true}$.

1. Suppose $\varphi \equiv \{\{a \approx b\}\} \uplus \varphi'$, where $a, b \in \mathsf{Const}(\varphi')$, and $\psi \equiv \varphi'[a := b]$.

   Taking into account $[\![a \approx b]\!]_{\mathcal{D}} = \mathsf{true}$, we obtain $[\![a]\!]_{\mathcal{D}} = [\![b]\!]_{\mathcal{D}}$, and as a result $[\![\varphi'[a := b]]\!]_{\mathcal{D}} = \mathsf{true}$.

2. Suppose $\psi \equiv \varphi[t := a]$, where t is a reducible term in φ and a is a fresh variable in φ. By Lemma 8.34, $[\![\psi]\!]_{\mathcal{D}} = \mathsf{true}$.

($\Leftarrow$) Suppose ψ is satisfiable, and $\mathcal{D}$ is a structure such that $[\![\psi]\!]_{\mathcal{D}} = \mathsf{true}$.

1. Suppose $\varphi \equiv \{\{a \approx b\}\} \uplus \varphi'$, where $a, b \in \mathsf{Const}(\varphi')$, and $\psi \equiv \varphi'[a := b]$.

   Since $a \notin \mathsf{Const}(\psi)$ we can choose $[\![a]\!]_{\mathcal{D}} = [\![b]\!]_{\mathcal{D}}$, and as a result $[\![\varphi']\!]_{\mathcal{D}} = \mathsf{true}$. Hence, $[\![\varphi]\!]_{\mathcal{D}} = \mathsf{true}$.

2. Suppose $\psi \equiv \varphi[t := a]$, where t is a reducible term in φ and a is a fresh variable in φ. By Lemma 8.34, $[\![\psi]\!]_{\mathcal{D}} = \mathsf{true}$.

$\boxtimes$

Theorem 8.36 (**Soundness and Completeness**) *A CNF φ is unsatisfiable if and only if the optimized* GDPLL(φ) *returns "unsatisfiable".*

Proof It follows from Theorem 8.22 and Lemma 8.35. $\boxtimes$

8.7 The extended EUF-DPLL calculus

The optimization techniques described in Section 8.6 can be also presented as rules of the calculus. The extended calculus includes the rules of the EUF-DPLL calculus and additionally the *term reduction* rule and the *equality elimination* rule.

The term reduction rule replaces a reducible term with a fresh constant and the equality elimination rule deletes a unit clause consisting of a positive equality

Equality propagation: $$\frac{\{\{s \approx t\}\} \cup \varphi}{\{\{s \approx t\}\} \cup \varphi[t := s]}$$ if $\varphi \in \mathsf{Cnf}$; $s, t \in \mathsf{Term}(\varphi)$; $s \notin \mathsf{SubTerm}(t)$

Tautology atom removing: $$\frac{\varphi}{\varphi|_{t \approx t}}$$ if $\varphi \in \mathsf{Cnf}$; $(t \approx t) \in \mathsf{Eq}(\varphi)$

Term Reduction: $$\frac{\varphi}{\varphi[t := a]}$$ if t is a reducible term and a is a fresh constant in φ

Equality elimination: $$\frac{\{\{a \approx b\}\} \cup \varphi}{\varphi}$$ if $a, b \in \mathsf{Const}$, $(a \approx b) \notin \mathsf{NPEq}(\{\{a \approx b\}\} \cup \varphi)$

Splitting: $$\frac{\varphi}{\{\{l\}\} \cup \varphi \mid \{\{\neg l\}\} \cup \varphi}$$ if $l \in \mathsf{Lit}(\mathsf{Core}(\varphi))$

Figure 8.3 The rules of the extended EUF-DPLL calculus

between two constants if this equality is not contained in the set of non-propagated equalities of the formula.

The extended calculus is easily proven to be terminating. Using termination, we prove it is also sound and complete.

Theorem 8.37 *The extended* EUF-DPLL *calculus is sound and complete.*

Proof It follows from Theorem 8.36. ⊠

Chapter 9

A BDD-representation for the EUF-logic

"If the only tool you have is a hammer, then every problem you see looks like a nail"

[Confucian saying]

"Can you do addition?" the White Queen asked. "What's one and one and one and one and one and one and one and one and one and one?" "I do not know," said Alice. "I lost count."

["Through the Looking Glass", Lewis Carroll]

This chapter presents a data structure called Binary Decision Diagrams for representing EUF formulae (EUF-BDDs). We define EUF-BDDs similar to BDDs, but we allow equalities between terms as labels instead of boolean variables. This method extends the approach introduced in [Groote and Pol, 2000]. However, the changed orientation of [Badban and Pol, 2004a] is essential for the completeness of our method.

We provide an approach to build a reduced ordered EUF-BDD (EUF-ROBDD) and prove that every path to a leaf is satisfiable by construction. Hence, one can immediately check whether an EUF-ROBDD is a tautology, a contradiction, or satisfiable. Moreover, EUF-ROBDDs are logically equivalent representations of EUF-formulae, which are concise for many examples. So EUF-ROBDDs can also be used to represent state spaces in symbolic model checking with data. Unfortunately, the unicity of the representation is lost. Nevertheless, our approach is still applicable for checking the equivalence of two formulae. A prototype implementation of this method works within the special purpose theorem prover for the μCRL toolset [Blom et al., 2003].

9.1 Definition of EUF-BDD

This chapter presents a new data structure called an EUF-BDD for representing and manipulating formulae containing equalities and uninterpreted functions. We will consider EUF-BDDs as a restricted subset of formulae.

Definition 9.1 *We define the set* B *of EUF-BDDs as follows.*

$$\mathsf{B} := \mathsf{true} \mid \mathsf{false} \mid \mathsf{ITE}(\mathsf{Eq}, \mathsf{B}, \mathsf{B})$$

It is straightforward to show that every formula defined above is equivalent to at least one EUF-BDD.

EUF-BDDs are nested ITE formulae which are represented in implementations as directed acyclic graphs. The difference between BDDs representing boolean formulae and EUF-BDDs is, that in the latter case internal nodes are labelled with equalities, i.e. an EUF-BDD can be represented as a rooted, directed acyclic graph with nodes of out-degree zero labelled by true and false, and a set of nodes of out-degree two labelled by equalities between ground terms. For a node l the two outgoing edges are given by two functions low(l) and high(l).

Throughout this chapter we will use T and S to denote EUF-BDDs.

Note, that in this chapter equalities are oriented. Therefore, we use a predicate symbol $\simeq$ instead of $\approx$, to emphasize that $s \simeq t$ and $t \simeq s$ are not the same equalities.

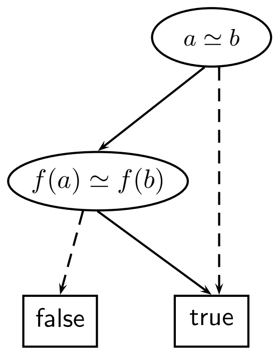

Figure 9.1 Dashed lines represent low/false edges, and solid ones represent high/true edges

Example 9.2 *The EUF-BDD representing the property of functional consistency* $a \simeq b \rightarrow f(a) \simeq f(b)$ *can be depicted as in Figure 9.1.*

The EUF-BDD can be written as $\mathsf{ITE}(a \simeq b, \mathsf{ITE}(f(a) \simeq f(b), \mathsf{true}, \mathsf{false}), \mathsf{true})$*.*

We introduce an order on terms below.

Definition 9.3 (**Simplification order on terms**) *We call an order on the set* Term *a simplification order, if it satisfies the following conditions:*

1. *For all* $s, t \in \mathsf{Term}$*,* $s \prec t$ *if* $s \in \mathsf{SubTerm}_p(t)$*.*
2. *For each* $f \in \mathsf{Fun}$ *and for all* $1 \leq i, j \leq n$*, and* $s_i, t \in \mathsf{Term}$*, if* $s_j \prec t$ *then* $f(s_1, \ldots, s_j, \ldots, s_n) \prec f(s_1, \ldots, t, \ldots, s_n)$*.*
3. *The order is total and well-founded.*

An example of such an order is the recursive path order [Dershowitz, 1979], which is also used in the implementation (see Section 9.4).

Definition 9.4 (**Order on equalities**) *Given a simplification order* $\prec$ *on terms, the total well-founded order on the set* Eq *is defined as follows.*

$$(s \simeq t) \prec (u \simeq v) \text{ if either } s \prec u \text{ or } s \equiv u \text{ and } t \prec v.$$

As it is mentioned in Chapter 2 we use terminology from term rewrite systems. In particular, by a normal form with respect to some TRS we mean a term to which no rules of the TRS are applicable. A system is terminating if no infinite rewrite sequence exists.

A first operation on EUF-BDDs is a simplification of equalities as defined below. Note that the auxiliary true symbols introduced in Definition 9.5 are eliminated by Definition 9.6.

Definition 9.5 (**Simplified equalities**)

- *Suppose e is an equality. By $e\downarrow$ we mean the normal form of e obtained after applying the following simplification rules.*
  - $t \simeq t \rightarrow \mathsf{true}$*, for each* $t \in \mathsf{Term}$.
  - $s \simeq t \rightarrow t \simeq s$ *for all* $s, t \in \mathsf{Term}$ *such that* $s \prec t$.
- *An equality e is called* simplified *if* $e \equiv e\downarrow$.

Starting with any EUF-BDD we have to simplify all equalities associated with the EUF-BDD nodes.

Definition 9.6 (**Simplified EUF-BDDs**)

- *Suppose T is an EUF-BDD. By $T\downarrow$ we mean the normal form of T obtained after applying the following rules.*
  - *For every node l in T associated with an equality $e(l)$,* $e(l) \rightarrow e(l)\downarrow$.
  - $\mathsf{ITE}(\mathsf{true}, T_1, T_2) \rightarrow T_1$.
- *An EUF-BDD T is called* simplified *if* $T \equiv T\downarrow$.

In the following we mean by $t[s]$ a term t such that $s \in \mathsf{SubTerm}(t)$, by $e[s]$ we mean an equality, where $s \in \mathsf{SubTerm}(e)$, and by $T[s]$ we mean an EUF-BDD T, where there is a node l in T associated with an equality $e(l)$ and $s \in \mathsf{SubTerm}(e(l))$.

Given the order on equalities, we can define a system of reduction rules as in [Groote and Pol, 2000], but the equations are oriented differently, as in [Badban and Pol, 2004a]. Starting with an arbitrary simplified EUF-BDD, we can transform it by repeatedly applying the following reduction rules.

Definition 9.7 (**Reduction rules on simplified EUF-BDDs**) *We define a TRS* Reduce-Order *as follows.*

1. $\mathsf{ITE}(e, T, T) \to T$
2. $\mathsf{ITE}(e, T_1, \mathsf{ITE}(e, T_2, T_3)) \to \mathsf{ITE}(e, T_1, T_3)$
3. $\mathsf{ITE}(e, \mathsf{ITE}(e, T_1, T_2), T_3)) \to \mathsf{ITE}(e, T_1, T_3)$
4. $\mathsf{ITE}(e_1, \mathsf{ITE}(e_2, T_1, T_2), T_3) \to \mathsf{ITE}(e_2, \mathsf{ITE}(e_1, T_1, T_3), \mathsf{ITE}(e_1, T_2, T_3))$, *if* $e_1 \succ e_2$.
5. $\mathsf{ITE}(e_1, T_1, \mathsf{ITE}(e_2, T_2, T_3)) \to \mathsf{ITE}(e_2, \mathsf{ITE}(e_1, T_1, T_2), \mathsf{ITE}(e_1, T_1, T_3))$, *if* $e_1 \succ e_2$.
6. $\mathsf{ITE}(s \simeq t, T_1[s], T_2) \to \mathsf{ITE}(s \simeq t, T_1[t]\downarrow, T_2)$, *if* $s \succ t$.

Rules $1-5$ are the rules for simplifying BDDs for propositional logic, eliminating redundant tests and ensuring the right ordering. Rule 6 allows to substitute equals for equals. Note that we immediately apply simplification after a substitution. The transformation by the reduction rules yields a logically equivalent EUF-BDD.

Definition 9.8 (**EUF-ROBDDs**) *We define an EUF-ROBDD to be a simplified EUF-BDD which is a normal form with respect to the TRS* Reduce-Order.

By a *reduced ordered* EUF-BDD (EUF-ROBDD) we mean an EUF-BDD such that the equalities labelling the nodes are *oriented*, i.e. for a given order $\prec$ on terms, if a node l is associated with an equality $s \simeq t$ then $s \succ t$; the equalities along a path appear only in a fixed order; for each EUF-ROBDD of the form $\mathsf{ITE}(s \simeq t, T_1, T_2)$, s doesn't occur in T_1.

Example 9.9 Consider $\varphi \equiv (x \simeq y \wedge y \simeq z) \to f(x) \simeq f(z)$. For a given order $x \prec y \prec z$, the derivation of an EUF-ROBDD is depicted in Figure 9.2. The EUF-ROBDD consists of one node true. In the picture, we combined several steps in one arrow. Note that intermediate EUF-BDDs should always be kept simplified. In the middle arrow of the picture, we explicitly show a simplification step.

9.2 Termination

An EUF-ROBDD is a normal form with respect to the TRS Reduce-Order. We show that for each EUF-BDD an EUF-ROBDD exists, i.e. the system of reduction rules is terminating.

We use *recursive path order* (RPO) [Dershowitz, 1979] to prove termination. The main idea of recursive path order is that two terms are compared by first comparing

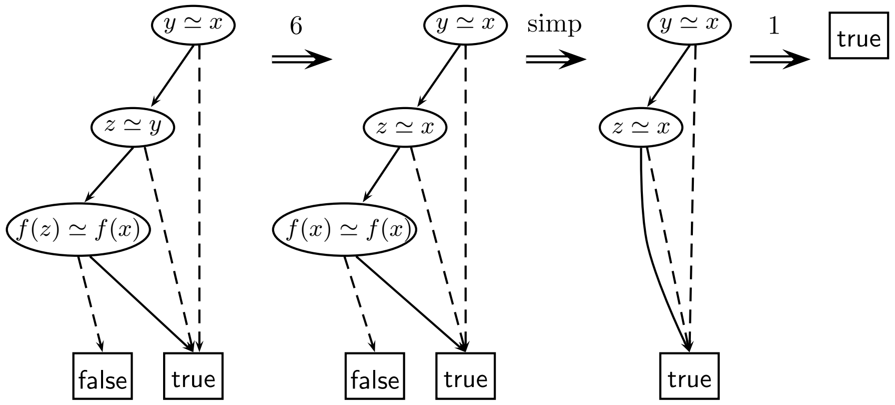

Figure 9.2 The derivation of the EUF-ROBDD for $(x \simeq y \wedge y \simeq z) \rightarrow f(x) \simeq f(z)$

their root symbols and then recursively comparing their subterms. We do it by viewing BDDs as binary trees, i.e. $\mathsf{ITE}(e, T_1, T_2)$ can be seen as the tree $e(T_1, T_2)$.

Definition 9.10 (*Recursive path order for BDDs*) *We say that for simplified EUF-BDDs S and T,*

$$S \succ_{rpo} T, \text{ where } S \equiv f(S_1, S_2), T \equiv g(T_1, T_2)$$

if one of the following holds.

1. *$S_1 \succeq_{rpo} T$ or $S_2 \succeq_{rpo} T$.*

2. *$f \succ g$, and $S \succ_{rpo} T_1$ and $S \succ_{rpo} T_2$.*

3. *$f \equiv g$, and $S \succ_{rpo} T_1$ and $S \succ_{rpo} T_2$, and either $S_1 \succ_{rpo} T_1$ or ($S_1 \equiv T_1$ and $S_2 \succ_{rpo} T_2$).*

Lemma 9.11 *Let $T[s]$ be a simplified EUF-BDD, and s and t be terms such that $s \succ t$, for some order satisfying Definition 9.3. Then $T[s] \succ_{rpo} T[t]\downarrow$.*

Proof Suppose $T[s]$ contains an internal node labelled with an equality $e[s]$. Consider $e[t]\downarrow$. Then one of the following holds.

- $e[t]\downarrow \equiv \mathsf{true}$.

  Suppose $T[s] \equiv e[s](T_1, T_2)$ contains one internal node which is labelled by an equality $e[s]$.

  By Definition 9.6, $T[t] \downarrow\equiv T_1$. Therefore, taking into account Definition 9.10(1), $T[s] \succ_{rpo} T[t]$.

  As the induction hypothesis we assume that for each simplified EUF-BDD $T'[s]$ containing at most $n-1$ internal nodes, $T'[s] \succ_{rpo} T'[t]\downarrow$.

  Suppose $T[s] \equiv e(T_1, T_2)$ has n internal nodes.

  If $s \in \mathsf{SubTerm}(e)$ then by Definition 9.6, $T[t]\downarrow\equiv T_1$. Therefore, taking into account Definition 9.10(1), $T[s] \succ_{rpo} T[t]$.

  If $s \notin \mathsf{SubTerm}(e)$ then s occurs either in T_1 or in T_2. By the induction hypothesis, if s occurs in T_1, then $T_1[s] \succ T_1[t]\downarrow$, and if s occurs in T_2, then $T_2[s] \succ T_2[t]\downarrow$. Hence, by 1 and 3 of Definition 9.10, $T[s] \succ T[t]\downarrow$.

- $e[t]\downarrow \not\equiv \mathsf{true}$.

  First we show that for each equality $e[s]$, $e[s]\downarrow \succ e[t]\downarrow$.

  Suppose $e[s]\downarrow\equiv u \simeq v$ and $e[t] \equiv u' \simeq v'$. Then one of the following holds.

  - $u' \succ v'$. Then $e[t]\downarrow\equiv u' \simeq v'$, and one of the following holds.
    - $u' \prec u$. In this case by Definition 9.4, $e[s]\downarrow\succ e[t]\downarrow$.
    - $u' \equiv u$ and $v' \prec v$. By Definition 9.4, $e[s]\downarrow\succ e[t]\downarrow$.
  - $u' \prec v'$. Then $e[t]\downarrow\equiv v' \simeq u'$. Since $v' \preceq v \prec u$, by Definition 9.4, $e[s]\downarrow\succ e[t]\downarrow$.

  Suppose $T[s]$ contains one internal node labelled by an equality $e[s]$. Since $T[s]$ is simplified, $e[s] \equiv e[s]\downarrow$. Taking into account that $e[s] \succ e[t]\downarrow$ and Definition 9.10(1,2), we obtain $T[s] \succ_{rpo} T[t]\downarrow$.

  As the induction hypothesis we assume that for each simplified EUF-BDD $T'[s]$ containing at most $n-1$ internal nodes, $T'[s] \succ_{rpo} T'[t]\downarrow$.

  Suppose $T[s] \equiv e(T_1, T_2)$ has n internal nodes.

  If $s \in \mathsf{SubTerm}(e)$ then $e \succ e[t]\downarrow$. Therefore taking into account 1 and 2 of Definition 9.10, we obtain $T[s] \succ T[t]\downarrow$.

  If $s \notin \mathsf{SubTerm}(e)$ then s occurs either in T_1 or in T_2. By induction hypothesis, if s occurs in T_1, then $T_1[s] \succ T_1[t]\downarrow$, and if s occurs in T_2, then $T_2[s] \succ T_2[t]\downarrow$. Hence, by 1 and 3 of Definition 9.10, $T[s] \succ T[t]\downarrow$.

$\boxtimes$

Theorem 9.12 *The rewrite system* Reduce-Order *is terminating.*

Proof We show that every rewrite rule is contained in $\succ_{rpo}$.

1. $e(T,T) \succ_{rpo} T$ by 1 of Definition 9.10.
2. $e(T_1, e(T_2, T_3)) \succ_{rpo} e(T_1, T_3)$ by 1 and 3 of Definition 9.10.
3. $e(e, T_1, T_2), T_3)) \succ_{rpo} e(T_1, T_3)$ by 1 and 3 of Definition 9.10.
4. Suppose $e_1 \succ e_2$. Then
   - $e_1(e_2(T_1, T_2), T_3) \succ_{rpo} e_1(T_1, T_3)$ by 1 and 3 of Definition 9.10.
   - $e_1(e_2(T_1, T_2), T_3) \succ_{rpo} e_1(T_2, T_3)$ by 1 and 3 of Definition 9.10.

   We conclude $e_1(e_2(T_1, T_2), T_3) \succ_{rpo} e_2(e_1(T_1, T_3), e_1(T_2, T_3))$ by 2.
5. Suppose $e_1 \succ e_2$. Then
   - $e_1(T_1, e_2(T_2, T_3)) \succ_{rpo} e_1(T_1, T_2)$ by 1 and 3.
   - $e_1(T_1, e_2(T_2, T_3)) \succ_{rpo} e_1(T_1, T_3)$ by 1 and 3.

   We conclude $e_1(T_1, e_2(T_2, T_3)) \succ_{rpo} e_2(e_1(T_1, T_2), e_1(T_1, T_3))$ by 2.
6. Suppose $s \succ t$. Then
   $T_1[s] \succ_{rpo} T_1[t] \downarrow$ by Lemma 9.11.
   Hence $(s \simeq t)(T_1[s], T_2) \succ_{rpo} (s \simeq t)(T_1[t] \downarrow, T_2)$ by 1 and 3 of Definition 9.10.

⊠

Theorem 9.13 *Every EUF-BDD is equivalent to some EUF-ROBDD.*

Proof It follows from the termination of Reduce-Order. ⊠

Checking equivalence of two boolean functions can be done by comparing their ROBDD representations: equivalent formulae have identical ROBDDs. Unfortunately, the canonicity property of EUF-ROBDDs is violated as we show with the following example. Nevertheless, our approach allows to check whether φ and ψ are equivalent: by Corollary 9.28, it can be done by verifying whether $\varphi \leftrightarrow \psi$ is a tautology.

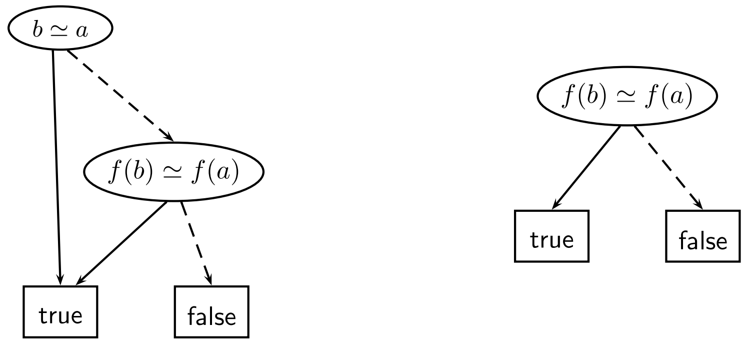

Figure 9.3 The EUF-ROBDD representation for $a \simeq b \vee f(a) \simeq f(b)$ and $f(a) \simeq f(b)$

Example 9.14 Consider two logically equivalent formulae $a \simeq b \vee f(a) \simeq f(b)$ and $f(a) \simeq f(b)$. For a given order $a \prec b \prec c \prec f(b) \prec f(c)$, their EUF-ROBDD representations are $\mathsf{ITE}(b \simeq a, \mathsf{true}, \mathsf{ITE}(f(b) \simeq f(a), \mathsf{true}, \mathsf{false}))$ and $\mathsf{ITE}(f(b) \simeq f(a), \mathsf{true}, \mathsf{false})$, which is depicted in Figure 9.3. No reduction rule is applicable, but the EUF-ROBDDs are not identical.

We have not yet studied strategies for choosing an ordering on equalities. A good ordering is crucial since it yields a compact representation: for some boolean functions, the ROBDD sizes are linear in the number of variables for one ordering, and exponential for another. In the boolean case the ordering of boolean variables can be chosen arbitrarily, but the ordering between equalities for the EUF logic must obey certain limitations. It is not clear how these limitations in the orderings will affect the size of EUF-ROBDD.

The basic procedure is presented as a term-rewrite system which converts an EUF-BDD into an EUF-ROBDD. We used a top-down approach to construct an EUF-ROBDD. Hence, the advantage of a polynomial APPLY algorithm [Bryant, 1986] is lost. Another line of possible research could be to study how to construct EUF-BDDs using a bottom-up approach like the one used for classical BDDs.

9.3 Satisfiability of paths in EUF-ROBDDs

In this section we prove that if the EUF-ROBDD corresponding to a formula φ consists of one node true then φ is a tautology. If the EUF-ROBDD consists of one node false then φ is a contradiction. In all other cases φ is satisfiable.

When BDDs are used to represent formulae including equalities and uninterpreted functions, a path to the true leaf in the BDD might not be consistent, i.e. the set of literals occurring along the path does not have a model. We show that each path in an EUF-ROBDD is satisfiable by construction.

For proving satisfiability of a path, we see it as a conjunction of literals occurring along the path, where $\wedge$ is considered modulo associativity and commutativity.

We use letters α and β to denote finite sequences of literals, ϵ for the empty sequence and $\alpha.\beta$ for the concatenation of sequences α and β.

Definition 9.15 (**EUF-BDD paths**)

- *We define the set* $\mathsf{Path}(T)$ *of all paths contained in an EUF-BDD* T *inductively as follows.*
  - $\mathsf{Path}(\mathsf{true}) = \epsilon$,
  - $\mathsf{Path}(\mathsf{false}) = \epsilon$,
  - $\mathsf{Path}(\mathsf{ITE}(e, T_1, T_2)) = \{e.\alpha \mid \alpha \in \mathsf{Path}(T_1)\} \cup \{\neg e.\alpha \mid \alpha \in \mathsf{Path}(T_2)\}$.
- *For a given path* $\alpha \equiv l_1 . \dots . l_n$, *we use an abbreviation* φ_α *to denote a formula* $l_1 \wedge \dots \wedge l_n$.
- *The formula* φ_α *is called the formula corresponding to path* α.
- *We say that* α *is a* satisfiable path *if* φ_α *is satisfiable.*

Example 9.16 Consider an EUF-BDD
$\mathsf{ITE}(a \simeq b, \mathsf{true}, \mathsf{ITE}(f(a) \simeq g(c), \mathsf{ITE}(b \simeq c, \mathsf{false}, \mathsf{true})), \mathsf{false})$. The EUF-BDD and the path $a \not\simeq b.f(a) \simeq g(c).b \not\simeq c$ are depicted in Figure 9.4.

Definition 9.17 (**Proper formulae**) *We say that a formula* $\varphi \equiv l_1 \wedge \dots \wedge l_n$ *is* proper *if the following holds.*

1. *for every* $(s \simeq t) \in \mathsf{Eq}(\varphi)$, $s \succ t$.
2. *for every* $s \simeq t \in \{l_1, \dots, l_n\}$ *and for every* $e \in \mathsf{Eq}(\varphi)$, *if* $e \succ (s \simeq t)$ *then* $s \notin \mathsf{SubTerm}(e)$.

Lemma 9.18 *A formula* φ *corresponds to a path in an EUF-ROBDD if and only if it is a proper formula.*

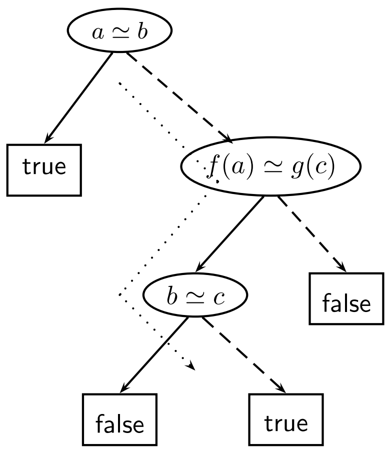

Figure 9.4 The path $a \not\simeq b.f(a) \simeq g(c).b \not\simeq c$

Proof Assume $\varphi \equiv l_1 \wedge \cdots \wedge l_n$ and $l_1 \ldots l_n$ is a corresponding path in an EUF-BDD.

($\Rightarrow$) Suppose a formula φ corresponds to a path in an EUF-ROBDD. We have to check Definition 9.17.

1. Definition 9.17(1) holds (otherwise not all equalities are simplified).

2. Definition 9.17(2) holds (otherwise Rule 6 would be applicable).

Hence, φ is a proper formula.

($\Leftarrow$) Suppose φ is a proper formula. W.l.o.g. we can assume that $l_i \not\equiv l_j$ for all $i \neq j$.

Consider the path $l_1 \ldots l_n$. Since we consider $\wedge$ modulo associativity, w.l.o.g. we can assume that for every $1 \leq i < j \leq n$, $e(l_i) \prec e(l_j)$.

All equalities are simplified by Definition 9.17(1). By Definition 9.17(2), for every $l_i \equiv s_i \simeq t_i$, $1 \leq i < n$, $s_i \notin \mathsf{SubTerm}(l_{i+1}, \ldots, l_n)$.

Hence, no rule of Definition 9.7 is applicable. Therefore, $l_1 \ldots l_n$ is a path in an EUF-ROBDD. ⊠

9.3.1 Satisfiability of reduced formulae

To prove satisfiability of paths in EUF-ROBDDs we use a satisfiability criterium from [Tveretina, 2004b].

Before turning to a proof that every path in an EUF-ROBDD is satisfiable, we need to give a definition of a non-propagated equality and a definition of a reduced formula. In [Tveretina, 2004b] the definition of non-propagated equalities is given for CNFs. Here, for sake of simplicity, we rather speak of formulae, but actually we are interested in the case when a formula is a conjunction of literals. Since we see a path as a conjunction of literals, where $\wedge$ is considered modulo associativity and commutativity, this corresponds to a set of unit clauses, as in [Tveretina, 2004b].

In Chapter 8 we gave a definition of non-propagated equalities. For sake of convenience let us recall this definition in slightly different notations.

Definition 9.19 (**Non-propagated equality**)

- *An equality $s \simeq t$ is called* non-propagated *in a formula φ if the following holds.*
  - $\varphi \equiv (s \simeq t) \wedge \psi$ *for some formula ψ.*
  - $s, t \in \mathsf{SubTerm}(\psi)$.
- *The set of all non-propagated equalities in φ is denoted by $\mathsf{NPEq}(\varphi)$.*

Example 9.20 Consider a formula

$$\varphi \equiv (a \simeq b) \wedge (f(a) \simeq f(b)) \wedge ((c \not\simeq f(a)) \vee (b \simeq g(a))).$$

According to Definition 9.19, only $a \simeq b$ is a non-propagated equality in φ.

Definition 9.21 (**Reduced formula**) *We say that*

$$\varphi \equiv l_1 \wedge \cdots \wedge l_n,$$

where $l_i \in \mathsf{Lit}$, for all $1 \leq i \leq n$, is reduced *if the following holds.*

- $\mathsf{NPEq}(\varphi) = \emptyset$,
- *for each $t \in \mathsf{Term}$, $(t \not\simeq t) \notin \mathsf{Lit}(\varphi)$.*

In the following RCnf *is used to denote the set of reduced formulae.*

For our purpose it is sufficient to give a satisfiability criterium for the case when a formula is a conjunction of literals.

Theorem 9.22 *Every $\varphi \in \mathsf{RCnf}$ is satisfiable.*

Proof It follows from Theorem 8.17. $\boxtimes$

9.3.2 Satisfiability of EUF-ROBDD paths

In this section we prove that every path in an EUF-ROBDD is satisfiable. For a given path α, we will transform φ_α into the logically equivalent reduced formula φ_α^{red}.

The idea of the proof is that a path in a (simplified) EUF-ROBDD contains segments of the form $s \not\simeq t_0. \cdots .s \not\simeq t_n.s \simeq t$. The term s doesn't occur as a subterm in any t_i, nor in any of the other segments. We obtain a path corresponding to $\varphi_\alpha^{red} \in \mathsf{RCnf}$ by propagating $s \simeq t$, i.e. replacing the segment by $t \not\simeq t_0. \cdots .t \not\simeq t_n.s \simeq t$. Note that this operation doesn't introduce new subterms, so propagated equalities in other segments remain propagated. The result is an equivalent formula in RCnf, hence it is satisfiable.

Example 9.23 For a given order $a \prec b \prec c \prec f(d) \prec g(a)$, the EUF-ROBDD representation of $\mathsf{ITE}(f(d) \simeq a, \mathsf{true}, \mathsf{ITE}(f(d) \simeq b, \mathsf{true}, \mathsf{ITE}(f(d) \simeq c, \mathsf{ITE}(g(a) \simeq c, \mathsf{true}, \mathsf{false}), \mathsf{false})))$ is depicted in Figure 9.5.

Consider the path $f(d) \not\simeq a.f(d) \not\simeq b.f(d) \simeq c.g(a) \simeq c$. The formula corresponding to the path contains one non-propagated equality $f(d) \simeq c$. By propagating this equality, i.e. replacing $f(d) \not\simeq a \wedge f(d) \not\simeq b$ with $c \not\simeq a \wedge c \not\simeq b$, we obtain a reduced formula $c \not\simeq a \wedge c \not\simeq b \wedge f(d) \simeq c \wedge g(a) \simeq c$.

The formula corresponding to the path and the reduced formula are logically equivalent. By Theorem 9.22, the reduced formula is satisfiable. Therefore, the path is also satisfiable.

We start with a technical lemma which is used in the proofs of Lemma 9.25 and Lemma 9.26

Lemma 9.24 *Let for $s, t \in \mathsf{Term}$, $\alpha.(s \simeq t).\beta$ be a path in an EUF-ROBDD. Then the following holds.*

1. *For every $(s' \simeq t') \in \alpha \cup \beta$, $s \notin \mathsf{SubTerm}(s' \simeq t')$.*

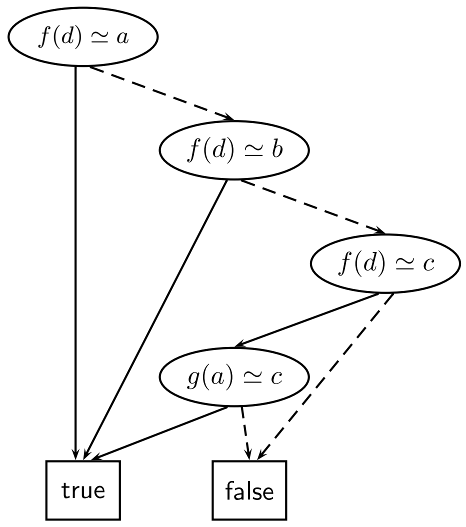

Figure 9.5 The EUF-ROBDD representation of a formula $\mathsf{ITE}(f(d) \simeq a, \mathsf{true}, \mathsf{ITE}(f(d) \simeq b, \mathsf{true}, \mathsf{ITE}(f(d) \simeq c, \mathsf{ITE}(g(a) \simeq c, \mathsf{true}, \mathsf{false}), \mathsf{false})))$.

2. $s \notin \mathsf{SubTerm}_p(\alpha.(s \simeq t).\beta)$.

Proof

1. *Since* α *is a path in the EUF-ROBDD,* $s \notin \mathsf{SubTerm}(\beta)$ *(otherwise Rule 6 of Definition 9.7 would be applicable).*

   Consider

$$(u \simeq v) \in \alpha$$

   Then the following holds (we use the first property of Definition 9.3).

   - $s \notin \mathsf{SubTerm}(u)$ *since* $u \prec s$ *(if* $u \equiv s$ *then Rule 6 of Definition 9.7 would be applicable, and if* $u \succ s$*, Rule 4 or 5 would be applicable).*
   - $s \notin \mathsf{SubTerm}(v)$ *since* $v \prec u \preceq s$ *(every EUF-ROBDD is simplified).*

   Hence,

$$s \notin \mathsf{SubTerm}(u \simeq v).$$

2. - *$s \notin \mathsf{SubTerm}_p(\beta)$ (otherwise Rule 6 of Definition 9.7 would be applicable).*
   - *For each $(u \simeq v) \in \alpha$, $s \notin \mathsf{SubTerm}_p(u \simeq v)$ by Lemma 9.24(1).*
   - *For each $(u \not\simeq v) \in \alpha$,*
     - *$s \notin \mathsf{SubTerm}_p(u)$ since $u \preceq s$ (otherwise Rule 4 or Rule 5 of Definition 9.7 would be applicable), and*
     - *$s \notin \mathsf{SubTerm}_p(v)$ since $v \prec u \preceq s$ (the EUF-ROBDD is a simplified EUF-BDD).*
   - *$s \notin \mathsf{SubTerm}_p(t)$ because $t \prec s$ (the EUF-BDD is simplified).*

   Hence,

$$s \notin \mathsf{SubTerm}_p(\alpha.s \simeq t.\beta).$$

⊠

Suppose $s, t \in \mathsf{Term}$ and $s \notin \mathsf{SubTerm}(t)$. As we defined in Chapter 2, by $\varphi[s := t]$ we mean the formula obtained from φ, where all occurrences of s are repeatedly replaced by t until no occurrence of s is left.

Again we prove a technical lemma which we use in the proof of Lemma 9.26.

Lemma 9.25 *Suppose $\alpha \equiv \alpha_1.(s \simeq t).\alpha_2$ is a path in an EUF-ROBDD, and $\beta \equiv \beta_1.(s \simeq t).\beta_2$, where $\beta_1 \subseteq \alpha_2$ and $\beta_2 \subseteq \alpha_2$.*

Then $\mathsf{SubTerm}(\varphi_\beta) = \mathsf{SubTerm}(\varphi_{\beta_1}[s := t] \wedge (s \simeq t) \wedge \varphi_{\beta_2}[s := t])$.

Proof We use the trivial observation that for $s, t, u \in \mathsf{Term}$, if $s \notin \mathsf{SubTerm}(u)$ then $\mathsf{SubTerm}(u[s := t]) \equiv \mathsf{SubTerm}(u)$. By Lemma 9.24, $s \notin \mathsf{SubTerm}_p(\alpha_1 \cup \alpha_2)$. Since $(\beta_1 \cup \beta_2) \subseteq (\alpha_1 \cup \alpha_2)$, we conclude $s \notin \mathsf{SubTerm}_p(\beta_1 \cup \beta_2)$.

Hence,

$$\begin{aligned}
\mathsf{SubTerm}(\varphi_\beta) &= \mathsf{SubTerm}(\varphi_{\beta_1} \wedge (s \simeq t) \wedge \varphi_{\beta_2}) \\
&= \mathsf{SubTerm}(\varphi_{\beta_1} \wedge \varphi_{\beta_2}) \cup \mathsf{SubTerm}(\{s \simeq t\}) \\
&= \mathsf{SubTerm}(\varphi_{\beta_1}[s := t] \wedge \varphi_{\beta_2}[s := t]) \cup \mathsf{SubTerm}(\{s \simeq t\}) \\
&= \mathsf{SubTerm}(\varphi_{\beta_1}[s := t] \wedge (s \simeq t) \wedge \varphi_{\beta_2}[s := t])
\end{aligned}$$

⊠

Lemma 9.26 *Let $\alpha \equiv \alpha_1.(s \simeq t).\alpha_2$ be a path in an EUF-ROBDD, and $(s \simeq t) \in \mathsf{NPEq}(\varphi_\alpha)$, where $s, t \in \mathsf{Term}$. Then the following holds.*

1. $|\mathsf{NPEq}(\varphi_{\alpha_1}[s := t] \wedge (s \simeq t) \wedge \varphi_{\alpha_2}[s := t])| < |\mathsf{NPEq}(\varphi_\alpha)|$.

2. $\varphi_{\alpha_1}[s := t] \wedge (s \simeq t) \wedge \varphi_{\alpha_2}[s := t]$ *is a proper formula.*

Proof

1. By Lemma 9.25

$$\mathsf{SubTerm}(\varphi_{\alpha_1}[s := t] \wedge (s \simeq t) \wedge \varphi_{\alpha_2}[s := t]) = \mathsf{SubTerm}(\varphi_\alpha).$$

Let us consider an arbitrary equality $(u \simeq v) \in \alpha$ such that

- $(u \simeq v) \not\equiv (s \simeq t)$,
- $(u \simeq v) \notin \mathsf{NPEq}(\varphi_\alpha)$.

By Lemma 9.24(1),

$$s \notin \mathsf{SubTerm}(u \simeq v).$$

Taking into account a trivial observation that for $s, t, u \in \mathsf{Term}$, if $s \notin \mathsf{SubTerm}(u)$ then $\mathsf{SubTerm}(u[s := t]) \equiv \mathsf{SubTerm}(u)$, we obtain

$$(u \simeq v)[s := t] \equiv (u \simeq v).$$

We show that $(u \simeq v) \notin \mathsf{NPEq}(\varphi_{\alpha_1}[s := t] \wedge (s \simeq t) \wedge \varphi_{\alpha_2}[s := t])$.

Consider $\alpha' \equiv \alpha \backslash \{u \simeq v\}$. One can see that either $(u \simeq v) \in \alpha_1$ or $(u \simeq v) \in \alpha_2$. W.l.o.g. we can consider the case $(u \simeq v) \in \alpha_1$. Let $\alpha'_1 \equiv \alpha_1 \backslash \{u \simeq v\}$.

By Lemma 9.25,

$$\mathsf{SubTerm}(\varphi_{\alpha'}) = \mathsf{SubTerm}(\varphi_{\alpha'_1}[s := t] \wedge (s \simeq t) \wedge \varphi_{\alpha_2}[s := t]).$$

Taking into account Definition 9.19, we obtain that

$$(u \simeq v) \notin \mathsf{NPEq}(\varphi_{\alpha_1}[s := t] \wedge (s \simeq t) \wedge \varphi_{\alpha_2}[s := t]).$$

We can conclude that for each $(u \simeq v) \in \alpha$ such that $(u \simeq v) \notin \mathsf{NPEq}(\varphi_\alpha)$,

$$(u \simeq v)[s := t] \notin \mathsf{NPEq}(\varphi_{\alpha_1}[s := t] \wedge (s \simeq t) \wedge \varphi_{\alpha_2}[s := t]),$$

i.e. the size of the set $\mathsf{NPEq}(\varphi_{\alpha_1}[s := t] \wedge (s \simeq t) \wedge \varphi_{\alpha_2}[s := t])$ cannot be bigger than the size of the set $\mathsf{NPEq}(\varphi_\alpha)$.

Since $(s \simeq t) \in \mathsf{NPEq}(\varphi_\alpha)$ and $(s \simeq t) \notin \mathsf{NPEq}(\varphi_{\alpha_1}[s := t] \wedge (s \simeq t) \wedge \varphi_{\alpha_2}[s := t])$, we can conclude that

$$|\mathsf{NPEq}(\varphi_{\alpha_1}[s := t] \wedge (s \simeq t) \wedge \varphi_{\alpha_2}[s := t])| < |\mathsf{NPEq}(\varphi_\alpha)|.$$

2. Taking into account Lemma 9.24 and a definition of an EUF-BDD, we obtain that there are $t_1, \ldots, t_k$, $k \geq 0$ such that

$$\alpha_1 \equiv \alpha.s \not\simeq t_1.\ldots.s \not\simeq t_k,$$

where $t_i \prec t$,$1 \leq i \leq k$, and $s \notin \mathsf{SubTerm}(\alpha' \cup \alpha_2)$.

We check now whether the formula $\varphi_{\alpha_1}[s := t] \wedge (s \simeq t) \wedge \varphi_{\alpha_2}[s := t]$ satisfies the conditions of Definition 9.17.

We denote $\overline{\alpha}$ $s \not\simeq t_1.\ldots.s \not\simeq t_k$, $\overline{\alpha}^* = s \not\simeq t_1[s := t].\ldots.s \not\simeq t_k[s := t] = t \not\simeq t_1.\ldots.t \not\simeq t_k$, and $A = \varphi_{\alpha_1} \wedge (s \simeq t) \wedge \varphi_{\alpha_2}$, $\overline{A} = \varphi_{\alpha_1}[s := t] \wedge (s \simeq t) \wedge \varphi_{\alpha_2}[s := t]$.

(a) We check Definition 9.17(1).
Consider an arbitrary $l \in \mathsf{Lit}(\overline{A})$. Then the following holds.

- $l \in \overline{\alpha}^*$. Then $l \equiv (t \not\simeq t_i)$ for some $i \in \{1, \ldots, k\}$. We conclude that $e(l)$ is not of the shape $r \simeq r$.
- $l \in \alpha[s := t] \wedge (s \simeq t) \wedge \alpha_2[s := t]$. We have shown that

$$\alpha[s := t] \wedge (s \simeq t) \wedge \alpha_2[s := t] \equiv \alpha \wedge (s \simeq t) \wedge \alpha_2.$$

Hence, $e(l)$ is not of the shape $r \simeq r$.

(b) We check Definition 9.17(2).
By the observation above for each $(u \simeq v) \in \mathsf{Lit}(A)$,

$$(u \simeq v)[s := t] \equiv (u \simeq v).$$

Consider an arbitrary $(u \simeq v) \in \mathsf{Eq}(\overline{A})$.
Then one of the following holds.

- $(u \simeq v) \in \mathsf{Eq}((s \simeq t) \wedge \varphi_{\alpha_2}[s := t])$.
Taking into account

$$(s \simeq t) \wedge \varphi_{\alpha_2}[s := t] \equiv (s \simeq t) \wedge \varphi_{\alpha_2}$$

we obtain that Definition 9.17(2) holds.
- $(u \simeq v) \in \mathsf{Eq}(\varphi_{\alpha_1}[s := t])$. Then taking into account Lemma 9.24 and Lemma 9.25 we obtain that Definition 9.17(2) also holds.

$\boxtimes$

Theorem 9.27 *Every path in an EUF-BDD is satisfiable.*

Proof

Consider an arbitrary path α in an EUF-ROBDD. We have to prove that φ_α is satisfiable. We claim that there exists a formula $\varphi_\alpha^{red} \in \mathsf{RCnf}$, which is satisfiable iff φ_α is. But φ_α^{red} is satisfiable by Theorem 9.22, which finishes the proof. We now prove the claim by induction on $|\mathsf{NPEq}(\varphi_\alpha)|$.

If $|\mathsf{NPEq}(\varphi_\alpha)| = 0$, note that $r \not\simeq r$ cannot occur in simplified paths, so $\varphi_\alpha \in \mathsf{RCnf}$, and we are finished.

Next, if $|\mathsf{NPEq}(\varphi_\alpha)| > 0$, then φ_α is of the form $\varphi_1 \wedge (s \simeq t) \wedge \varphi_2$, where $s \simeq t$ is not propagated. Define $\varphi' := \varphi_1[s := t] \wedge (s \simeq t) \wedge \varphi_2[s := t]$. By Lemma 9.26(1), $|\mathsf{NPEq}(\varphi')| < |\mathsf{NPEq}(\varphi_\alpha)|$, and by Lemma 9.26(2), no occurrences of $r \not\simeq r$ are introduced. Taking into account Lemma 9.26(2) and by the induction hypothesis, we find $\varphi'_{red} \in \mathsf{RCnf}$ is equivalent to φ'.

We show that φ_α and φ' are logically equivalent. Suppose $[\![\varphi_\alpha]\!]_\mathcal{D} = \mathsf{true}$ for a structure $\mathcal{D}$. Hence, $[\![s]\!]_\mathcal{D} = [\![t]\!]_\mathcal{D}$. For each $l[s := t] \in \mathsf{Lit}(\varphi')$, taking into account that $[\![l]\!]_\mathcal{D} = \mathsf{true}$, we obtain that $[\![l[s := t]]\!]_\mathcal{D} = \mathsf{true}$. Hence, $[\![\varphi']\!]_\mathcal{D} = \mathsf{true}$. Assume that $[\![\varphi']\!]_\mathcal{D} = \mathsf{true}$ for a structure $\mathcal{D}$. Taking into account the shape of φ', we obtain $[\![s]\!]_\mathcal{D} = [\![t]\!]_\mathcal{D}$. Hence, taking into account that for each $l \in \varphi_\alpha$, $[\![l[s := t]]\!]_\mathcal{D} = \mathsf{true}$, we obtain that $[\![l]\!]_\mathcal{D} = \mathsf{true}$. Hence, $[\![\varphi_\alpha]\!]_\mathcal{D} = \mathsf{true}$. We conclude that φ_α and φ' are logically equivalent.

$\boxtimes$

Corollary 9.28 *From Theorem 9.27*

- *The only tautological EUF-ROBDD is* true.
- *The only contradictory EUF-ROBDD is* false.
- *All other EUF-ROBDDs are satisfiable.*

Proof By Theorem 9.27, if an EUF-ROBDD is a tautology then every path in it is satisfiable. So all leaves must be equivalent to true. By Rule 1 of Definition 9.7, the EUF-ROBDD will be reduced to a node true. Similarly, the only contradictory EUF-ROBDD is a node false. Hence, all other EUF-BDDs are satisfiable. $\boxtimes$

9.4 Implementation and Applications

The first prototype implementation of equational BDDs is presented in [Groote and Pol, 2000]. The theory did not contain function symbols and it had a different order on equalities. The benchmarks were taken from [Strichman]. They originate from compiler optimization: formulae representing the source code and the target code of a compilation step have to be equivalent. The obtained results are comparable with the results in [Pnueli et al., 1999].

The current approach, described in this chapter, was implemented by Jaco van de Pol as an extension of the previous implementation [Pol, 2001]. It is complete for the theory with equality and uninterpreted function symbols. The implementation works within the special purpose theorem prover for the μCRL toolset [Blom et al., 2001, 2003]. The language μCRL is a process algebraic language that was especially developed to take account data in the study of communicating processes. It is intended to study the description and analysis techniques for large distributed systems (see: http://homepages.cwi.nl/∼mcrl/).

The prover is used to discharge proof obligations generated in protocol verifications, in particular to prove process invariants and confluence of internal computation steps. It is also used in the symbolic model checker with data proposed in [Groote and Willemse, 2004]. In the latter application it is essential to have a concise representation of formulae, which is provided by EUF-BDDs. The prover itself deals with the data part only.

As the programming language was used C and the ATerm library [Brand et al., 2000]. The ATerm implementation is based on maximal subterm sharing and automatic garbage collection. Using the ATerm library, terms are represented as maximally shared DAGs. Therefore, syntactical identity of terms can be checked in constant time.

The implementation was applied to many existing case studies (see for instance [Blom and Pol, 2002, Blom et al., 2003] for a description). The new implementation is as efficient as the previous one which could not handle uninterpreted functions.

The resulting EUF-ROBDDs can be visualized for small formulae.

The EUF-BDD for

$$f(f(f(f(f(a))))) \approx b \wedge f(f(a)) \approx f(a)$$

is depicted in Figure 9.6.

The EUF-BDD for

$$(f(b) \simeq f(c) \Rightarrow a \simeq b) \Leftrightarrow (f(b) \simeq f(c) \Rightarrow a \simeq c)$$

is depicted in Figure 9.7.

We have not yet studied strategies for choosing an ordering on equalities. A good ordering is crucial since it yields a compact representation: for some boolean functions, the ROBDD sizes are linear in the number of variables for one ordering, and exponential for another. If in boolean case the guards ordering for BDDs can be chosen arbitrarily, for the EUF logic the ordering between equalities must obey

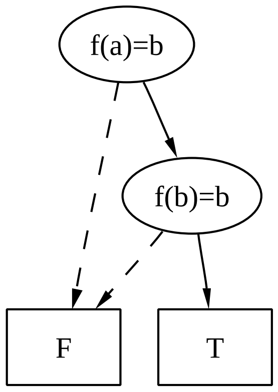

Figure 9.6 EUF-ROBDD representing $f(f(f(f(f(a))))) \approx b \wedge f(f(a)) \approx f(a)$ obtained by the implementation

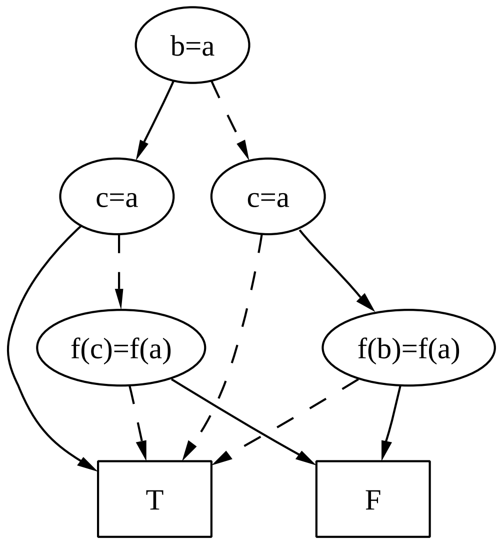

Figure 9.7 EUF-ROBDD $(f(b) \simeq f(c) \Rightarrow a \simeq b) \Leftrightarrow (f(b) \simeq f(c) \Rightarrow a \simeq c)$ obtained by the implementation

certain limitations. It is not clear how these limitations in the orderings will affect the size of the EUF-ROBDD.

Chapter 10

Conclusion

“If what you did yesterday still looks good to you, then your goals for tomorrow are not big enough”

[Ling Fu Yu]

This thesis is concerned with new decision procedures in automated reasoning. We considered the satisfiability problem for decidable subsets of first-order logic with equality. Mainly, we were interested in the logic of equality with uninterpreted functions. This kind of logic has been proposed for processor verification. Fast satisfiability checking is of great importance for such verification to be successful.

As a basis for the decision procedures we took well-known resolution-based techniques such as the DP and DPLL methods, and BDDs. We briefly surveyed major propositional methods and discussed how they can be used to decide subsets of first-order predicate logic.

We started by analyzing the relation between the size of a DPLL proof and a resolution proof. We gave a transformation of a DPLL refutation to a resolution refutation in a number of steps that is essentially less than the number of unit resolution steps applied in the DPLL refutation.

In this thesis, we have presented several techniques for checking satisfiability of EUF formulae. To achieve this goal, we have generalized a variety of techniques applied to propositional formulae. A particular important contribution is a generalization of the well-known DPLL procedure for propositional logic, called GDPLL. Sufficient properties are identified for proving soundness, termination and completeness of GDPLL. The original DPLL procedure can be presented as an instance of GDPLL. Subsequently the GDPLL instances for equality logic, and the EUF logic are presented.

Traditionally, decision procedures determine validity, or dually unsatisfiability, of formulae in conjunctive normal form. However, for many problems CNF is not a natural representation. We have lifted the propositional non-clausal DPLL procedure and BDDs to the EUF-logic. EUF-BDDs can be used in the symbolic model checker with data proposed in [Groote and Willemse, 2004]. In the latter application it is essential to have a concise representation of formulae, which is provided by EUF-BDDs.

10.1 Future work

As it is mentioned in the introduction, an aim of this thesis was rather to develop novel calculi than an efficient tool for checking satisfiability. Modern efficient theorem provers combine different techniques and heuristics to be able to check satisfiability of large formulae.

There is always room for improvements. The choice of the literal to which to apply a splitting rule is crucial for efficiency of DPLL-based systems. There are a lot of described approaches for it in the SAT literature, but not all propositional techniques can be immediately used in case of EUF-logic. One of the desirable extensions of the work could be studying which heuristics are based on properties of propositional logic and which ones could be lifted to a more general case.

As already mentioned in Chapter 9, we have not yet studied strategies for choosing an ordering on equalities while constructing EUF-BDDs. A good ordering is crucial since it yields a compact representation: for some boolean functions, the ROBDD sizes are linear in the number of variables for one ordering, and exponential for another. While in the boolean case the guards ordering for BDDs can be chosen arbitrarily. For the EUF logic the ordering between equalities must obey certain limitations. It is not clear how these limitations in the orderings will effect the size of the EUF-ROBDD.

The basic procedure is presented as a term-rewrite system which converts an EUF-BDD into an EUF-ROBDD. We used a top-down approach to construct an EUF-ROBDD. Hence, the advantage of a polynomial APPLY algorithm is lost. Another line of possible research could be studying how to construct EUF-BDDs using a bottom-up approach like the one used for the classical BDDs.

References

W. Ackermann. *Solvable cases of the decision problem.* Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, 1954.

S.B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, c-27(6): 509–516., June 1978.

G. Audemard, P. Bertoli, A. Cimatti, A. Kornilowicz, and R. Sebastiani. A SAT based approach for solving formulas over boolean and linear mathematical propositions. In Andrei Voronkov, editor, *CADE-18: Proceedings of the 18th International Conference on Automated Deduction*, volume 2392 of *LNCS*, pages 195–210. Springer-Verlag, 2002.

J. Avenhaus, J. Denzinger, and M. Fuchs. DISCOUNT: A system for distributed equational deduction. In J. Hsiang, editor, *RTA*, volume 914 of *LNCS*, pages 397–402. Springer-Verlag, 1995.

F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

J. Van Baalen, P. Robinson, M.R. Lowry, and T. Pressburger. Explaining synthesized software. In *The Thirteenth IEEE Conference on Automated Software Engineering (ASE'98)*, pages 240–248, 1998.

L. Bachmair and H. Ganzinger. On restrictions of ordered paramodulation with simplification. In Mark E. Stickel, editor, *CADE*, volume 449 of *LNCS*, pages 427–441. Springer-Verlag, 1990.

L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *J. Log. Comput.*, 4(3):217–247, 1994.

L. Bachmair and A. Tiwari. Abstract congruence closure and specializations. In David McAllester, editor, *Conference on Automated Deduction, CADE '2000*, volume 1831 of *LNAI*, pages 64–78. Springer-Verlag, 2000.

B. Badban and J. C. van de Pol. An algorithm to verify formulas by means of (0,s,=)-BDDs. In *Proceedings of the 9th Annual Computer Society of Iran*

Computer Conference (CSICC 2004), Tehran, Iran, 2004a. The results are also available in [Badban and Pol, 2002].

B. Badban, J. van de Pol, O. Tveretina, and H. Zantema. Generalizing DPLL and satisfiability for equalities. CSR Technical Report 04/14, Technical University of Eindhoven, 2004a.

B. Badban and J.C. van de Pol. Two solutions to incorporate zero, successor and equality in binary decision diagrams. Technical Report SEN-R0231, CWI, Amsterdam, December 2002.

B. Badban and J.C. van de Pol. Zero, sucessor and equality in BDDs. *Accepted for Annals of Pure and Applied Logic*, 2004b. The results are also available in [Badban and Pol, 2002].

B. Badban, J. van de Pol, O. Tveretina, and H. Zantema. Solving satisfiability of ground term algebras using DPLL and unification. In *Workshop on Unification*, Cork, July 2004b.

C.W. Barrett, D.L. Dill, and J.R. Levitt. Validity checking for combinations of theories with equality. In M. Srivas and A. Camilleri, editors, *Formal Methods in Computer-Aided Design (FMCAD'96)*, volume 1166 of *LNCS*, pages 187–201. Springer-Verlag, November 1996.

C.W. Barrett, D.L. Dill, and A. Stump. Checking satisfiability of first-order formulas by incremental translation to SAT. In E. Brinksma and K. Guldstrand, editors, *14th International Conference on Computer-Aided Verification*, volume 2404 of *LNCS*, pages 236–249. Springer-Verlag, 2002.

R. Bird. *Introduction to Functional Programming using Haskell.* Prentice Hall, 1998.

N. Bjørner, M.E. Stickel, and T.E. Uribe. A practical integration of first-order reasoning and decision procedures. In William McCune, editor, *CADE*, volume 1249 of *LNCS*, pages 101–115. Springer-Verlag, 1997.

S. C. C. Blom, J. F. Groote, I. van Langevelde, B. Lisser, and J. C. van de Pol. New developments around the μcrl tool set. In *Proceedings of FMICS 2003*, volume 80 of *ENTCS*, 2003.

S.C.C. Blom, W.J. Fokkink, J.F. Groote, I. van Langevelde, B. Lisser, and J.C. van de Pol. μCRL: A toolset for analysing algebraic specifications. In G. Berry, editor, *CAV 2001*, volume 2102 of *LNCS*, pages 250–254. Springer-Verlag, 2001.

S.C.C. Blom and J.C. van de Pol. State space reduction by proving confluence. In E. Brinksma and K.G. Larsen, editors, *Proceedings of CAV'02*, LNCS 2404, pages 596–609. Springer-Verlag, 2002.

K.S. Brace, R.L. Rudell, and R.E. Bryant. Efficient implementation of a BDD package. In *27th ACM/IEEE conference on Design automation*, pages 40 – 45, 1991.

M. van den Brand, H. A. de Jong, P. Klint, and P. A. Olivier. Efficient annotated terms. *Software - Practice and Experience (SPE)*, 30(3):259–291, 2000.

R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

R. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24(3):293–318, 1992a.

R. Bryant and M. Velev. Boolean satisfiability with transitivity constraints. *ACM Transactions on Computational Logic*, 3(4):604–627, October 2002.

R.E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992b.

R.E. Bryant, S. German, and M.N. Velev. Exploiting positive equality in a logic of equality with uninterpreted functions. In N. Halbwachs and D. Peled, editors, *Computer-Aided Verification (CAV'99)*, volume 1633 of *LNCS*, pages 470–482. Springer-Verlag, July 1999a.

R.E. Bryant, S. German, and M.N. Velev. Processor verification using efficient decision procedures for a logic of uninterpreted functions. In N.V. Murray, editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, volume 1617 of *LNAI*, pages 1–13. Springer-Verlag, June 1999b.

R.E. Bryant, S. German, and M.N. Velev. Processor verification using efficient reductions of the logic of uninterpreted functions to propositional log. *ACM Transactions on Computational Logic*, 2(1):93–134, January 2001.

R.E. Bryant and M.N. Velev. Boolean satisfiability with transitivity constraints. In E.A. Emerson and A.P. Sistla, editors, *Computer-Aided Verification (CAV'00)*, volume 1855 of *LNCS*, pages 85–98. Springer-Verlag, July 2000.

J.R. Burch and D.L. Dill. Automated verification of pipelined microprocesoor control. In D.L. Dill, editor, *Computer-Aided Verification (CAV'94)*, volume 818 of *LNCS*, pages 68–80. Springer-Verlag, June 1994.

I. Dahn and C. Wernhard. First order proof problems extracted from an article in the mizar mathematical library. In *International Workshop on First order Theorem Proving*, number 97-50 in RISC-Linz Report Series, pages 58–62, Linz, 1997.

M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the Association for Computing Machinery*, 7:394–397, July 1962.

M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, July 1960.

N. Dershowitz. A note on simplification orderings. *Information Processing Letters*, 9(5):212–215, 1979.

L. Drake, A.M. Frisch, and T. Walsh. Adding resolution to the DPLL procedure for boolean satisfiability. In *SAT 2002*, 2002.

Cormac Flanagan, Rajeev Joshi, Xinming Ou, and James B. Saxe. Theorem proving using lazy proof explication. In *CAV*, pages 355–367, 2003.

P. Fontaine and E.P. Gribomont. Using BDDs with combinations of theories. volume 2514 of *LNCS*, pages 190–201. Springer-Verlag, 2002.

H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast decision procedures. In Rajeev Alur and Doron Peled, editors, *CAV 2004*, volume 3114 of *LNCS*, pages 175–188. Springer-Verlag, 2004.

A. Goel, K. Sajid, H. Zhou, A. Aziz, and V. Singhal. BDD based procedures for a theory of equality with uninterpreted functions. In A. J. Hu and M. Y. Vardi, editors, *Computer-Aided Verification (CAV'98)*, volume 1427 of *LNCS*, pages 244–255. Springer-Verlag, 1998.

A. Goldberg. Average case complexity of the satisfiability problem. In *4th Workshop on Automated Deduction*, pages 1–6, Austin Texas, 1979.

J.F. Groote and J.C. van de Pol. Equational binary decision diagrams. In M. Parigot and A. Voronkov, editors, *Logic for Programming and Reasoning (LPAR'2000)*, volume 1955 of *LNAI*, pages 161–178, 2000.

J.F. Groote and T.A.C. Willemse. Parameterised boolean equation systems (extended abstract). In P. Gardner and N. Yoshida, editors, *CONCUR 2004*, volume 3170 of *LNCS*, pages 308–324, 2004.

J.F. Groote and H. Zantema. Resolution and binary decision diagrams cannot simulate each other polynomially. In M. Broy D. Bjorner and A. Zamulin, editors, *Ershov Memorial Conference*, volume 2244 of *LNCS*, pages 33–38, 2001.

J. Huang and A. Darwiche. Using DPLL for efficient OBDD construction. In *the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT 2004)*, 2004.

R. Letz, J. Schumann, S. Bayerl, and W. Bibel. SETHEO: A high-performance theorem prover. *J. Autom. Reasoning*, 8(2):183–212, 1992.

W. McCune and L. Wos. Otter - the CADE-13 competition incarnations. *J. Autom. Reason.*, 18(2):211–220, 1997. ISSN 0168-7433. doi: http://dx.doi.org/10.1023/A:1005843632307.

G. Nelson and D. Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM*, 27 (2):356 – 364, April 1980.

R. Nieuwenhuis and A. Oliveras. Congruence closure with integer offsets. In M Vardi and A Voronkov, editors, *10h Int. Conf. Logic for Programming, Artif. Intell. and Reasoning (LPAR)*, volume 2850 of *LNAI*, pages 78–90, 2003.

R. Nieuwenhuis and A. Rubio. Theorem proving with ordering constrained clauses. In *CADE*, volume 607 of *LNCS*, pages 477–491. Springer-Verlag, 1992.

A. Pnueli, Y. Rodeh, and O. Shtrichman. Range allocation for equivalence logic. In R. Hariharan, M. Mukund, and V. Vinay, editors, *Foundations of Software Technology and Theoretical Computer Science (FSTTCS'01)*, volume 2245 of *LNCS*, pages 317–333. Springer-Verlag, December 2001.

A. Pnueli, Y. Rodeh, O. Shtrichman, and M. Siegel. Deciding equality formulas by small domains instantiations. In Nicolas Halbwachs and Doron Peled, editors, *Computer-Aided Verification (CAV'99)*, volume 1633 of *LNCS*, pages 455–469. Springer-Verlag, 1999.

A. Pnueli, Y. Rodeh, O. Shtrichman, and M. Siegel. The small model property: how small can it be? *Information and Computation*, 178(1):279 – 293, October 2002.

J.C. van de Pol. A prover for the μCRL toolset with applications – Version 0.1. Technical Report SEN-R0106, CWI, Amsterdam, 2001.

J.C. van de Pol and O. Tveretina. A BDD-representation for the logic of equality and uninterpreted functions. Technical report, CWI, 2005.

I. Rish and R. Dechter. Resolution versus search: Two strategies for sat. *Journal of Automated Reasoning*, 24(1/2):225–275, 2000.

G. Robinson and L. Wos. Paramodulation and theorem-proving in first-order theories with equality. *Machine inteligence*, 4:135–150, 1969.

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

Y. Rodeh and O. Shtrichman. Finite instantiations in equivalence logic with uninterpreted functions. In *Computer Aided Verification (CAV'01)*, volume 2102 of *LNCS*, pages 144–154. Springer-Verlag, July 2001.

S. Schulz. *Learning Search Control Knowledge for Equational Deduction*, volume 230. IOS Press, 2000.

J. Schumann. Automatic verification of cryptographic protocols with SETHEO. In William McCune, editor, *CADE*, volume 1249 of *LNCS*, pages 87–100. Springer-Verlag, 1997.

R.E. Shostak. An algorithm for reasoning about equality. *Communications of the ACM*, 21:583–585, July 1978.

R.E. Shostak. A practical decision procedure for arithmetic with function symbols. *Journal of the ACM*, 26(2):351–360, April 1979.

M.E. Stickel, R.J. Waldinger, M.R. Lowry, T. Pressburger, and I. Underwood. Deductive composition of astronomical software from subroutine libraries. In Alan Bundy, editor, *CADE*, volume 814 of *LNCS*, pages 341–355. Springer-Verlag, 1994.

O. Strichman. Benchmarks for satisfiability checking of equality formulas. See: http://iew3.technion.ac.il/ ofers/sat/bench.htm.

G. Sutcliffe, C.B. Suttner, and T. Yemenis. The TPTP problem library. In Alan Bundy, editor, *CADE*, volume 814 of *LNCS*, pages 252–266. Springer-Verlag, 1994.

C. Tinelli. A DPLL-based calculus for ground satisfiability modulo theories. In G. Ianni and S. Flesca, editors, *Proceedings of the 8th European Conference on Logics in Artificial Intelligence (Cosenza, Italy)*, volume 2424 of *LNAI*, pages 308–319. Springer-Verlag, 2002.

G.S. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic, Part 2*, pages 115–125. Consultant Bureau, New York-London, 1968.

O. Tveretina. Deciding satisfiability of equality logic formulas with uninterpreted functions. In *Joint Annual Workshop of ERCIM/CoLogNet on Constraint Solving and Constraint Logic Programming*, Lausanne, June 2004a.

O. Tveretina. A decision procedure for equality logic with uninterpreted functions. In B. Buchberger and J. A. Campbell, editors, *Artificial Intelligence and Symbolic Mathematical Computation*, volume 3249 of *LNAI*, pages 63–76. Springer-Verlag, 2004b.

O. Tveretina and H. Zantema. Transforming DPLL to resolution. CS-report 02-07, Technical University of Eindhoven, July 2002.

O. Tveretina and H. Zantema. A proof system and a decision procedure for equality logic. CS-report 03-02, Technical University of Eindhoven, 2003.

O. Tveretina and Hans Zantema. A proof system and a decision procedure for equality logic. In Martin Farach-Colton, editor, *LATIN 2004: Theoretical Informatics*, volume 2976 of *LNCS*, pages 530–539, 2004.

C. Weidenbach, B. Gaede, and G. Rock. SPASS & FLOTTER version 0.42. In *CADE-13: Proceedings of the 13th International Conference on Automated Deduction*, pages 141–145. Springer-Verlag, 1996. ISBN 3-540-61511-3.

N. Yugami. Theoretical analysis of Davis-Putnam procedure and propositional satisfiability. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 282–288, 1995.

H. Zantema and J.F. Groote. Transforming equality logic to propositional logic. In *Proceedings of 4th International Workshop on First-Order Theorem Proving (FTP'03)*, volume 86 of *ENTCS*, 2003.

Table of Frequently Used Symbols

$\approx$	Equality predicate symbol, used in infix notation for non-oriented equalities (see p.11).
$\simeq$	Equality predicate symbol, used in infix notation for oriented equalities (see p.11).
$\equiv$	Syntactic identity of two elements.
$\|S\|$	Cardinality of a set S.
$S \uplus T$	Disjoint union of S and T.
$\mathbf{N}$	Set of natural numbers, $\mathbf{N} = \{0, 1, 2, \ldots\}$.
$u[s := t]$	The normal form of u with respect to $\rightarrow_{(s,t)}$ (see p.16).
$\phi[s := t]$	The normal form of ϕ with respect to $\rightarrow_{(s,t)}$ (see p.16).
$\perp$	The empty clause.
$\phi\|_l$	The CNF obtained from the CNF ϕ by deleting all clauses containing the literal l and removing the literal $\neg l$ from all other clauses (see p.10).

Index

Samenvatting

In dit proefschrift presenteren we een aantal technieken om vervulbaarheid (satisfiability) vast te stellen binnen beslisbare delen van de eerste orde logica met gelijkheid. Het doel van dit proefschrift is voornamelijk het ontwikkelen van nieuwe technieken in plaats van het ontwikkelen van een efficiënte implementatie om vervulbaarheid vast te stellen. Als algemeen logisch raamwerk gebruiken we de eerste orde predikaten logica zonder kwantoren.

We beschrijven enkele basisprocedures om vervulbaarheid van propositionele formules vast te stellen: de DP procedure, de DPLL procedure, en een techniek gebaseerd op BDDs. Deze technieken zijn eigenlijk families van algoritmen in plaats van losse algoritmen. Hun gedrag wordt bepaald door een aantal keuzen die ze maken gedurende de uitvoering.

We geven een formele beschrijving van resolutie, en we analyseren gedetailleerd de relatie tussen resolutie en DPLL. Het is bekend dat een DPLL bewijs van onvervulbaarheid (refutation) rechtstreeks kan worden getransformeerd naar een resolutie bewijs van onvervulbaarheid met een vergelijkbare lengte. In dit proefschrift wordt een transformatie geïntroduceerd van zo'n DPLL bewijs naar een resolutie bewijs dat de kortst mogelijke lengte heeft.

We presenteren GDPLL, een generalisatie van de DPLL procedure. Deze is bruikbaar voor het vervulbaarheidsprobleem voor beslisbare delen van de eerste orde logica zonder kwantoren. Voldoende eigenschappen worden geïdentificeerd om de correctheid, de beëindiging en de volledigheid van GDPLL te bewijzen.

We beschrijven manieren om vervulbaarheid vast te stellen binnen de logica met gelijkheid en niet-geïnterpreteerde functies (EUF). Dit soort logica is voorgesteld om abstracte hardware ontwerpen te verifiëren. Het snel kunnen vaststellen van vervulbaarheid binnen deze logica is belangrijk om dergelijke verificaties te laten slagen. In de afgelopen jaren zijn er verschillende procedures voorgesteld om de vervulbaarheid van dergelijke formules vast te stellen.

Wij beschrijven een nieuwe aanpak om vervulbaarheid vast te stellen van formules uit de logica met gelijkheid die in de conjunctieve normaal vorm zijn gegeven. Centraal in deze aanpak staat één enkele bewijsregel genaamd gelijkheidsresolutie.

Voor deze ene regel bewijzen wij correctheid en volledigheid. Op grond van deze regel stellen we een volledige procedure voor om vervulbaarheid van dit soort formules vast te stellen, en we bewijzen de correctheid ervan.

Daarnaast presenteren we nog een nieuwe procedure om vervulbaarheid vast te stellen van EUF-formules, gebaseerd op de GDPLL methode.

Tot slot breiden we BDDs voor propositionele logica uit naar logica met gelijkheid. We bewijzen dat alle paden in deze uitgebreide BDDs vervulbaar zijn. In een constante hoeveelheid tijd kan vastgesteld worden of de formule een tautologie is, een tegenspraak is, of slechts vervulbaar is.

Curriculum Vitae

Olga Tveretina was born on August 14, 1965 in Tomsk (Russia). In 1982 she finished her secondary school. She studied Applied Mathematics at the Tomsk State University and at the Tartu State University. She obtained her master degree with honours in 1987. From 1987 till 1989 she worked as an engineer-programmer in the Ministry of Finance in Estonia. She worked subsequently as a researcher and teaching assistant at the Kiev Polytechnical University (Ukraine) and at the Kharkov Technical University (Ukraine). After being a PhD student in Bilkent (Turkey) she moved to the Netherlands to start a PhD project at the Technical University of Eindhoven. This research project, called "Integrating techniques for verification of distributed systems", was funded by the NWO and led to this thesis.

Titles in the IPA Dissertation Series

Mathematics and Natural Sciences, UL. 2004-03

S. Maneth. *Models of Tree Translation.* Faculty of Mathematics and Natural Sciences, UL. 2004-04

Y. Qian. *Data Synchronization and Browsing for Home Environments.* Faculty of Mathematics and Computer Science and Faculty of Industrial Design, TU/e. 2004-05

F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-06

L. Cruz-Filipe. *Constructive Real Analysis: a Type-Theoretical Formalization and Applications.* Faculty of Science, Mathematics and Computer Science, KUN. 2004-07

E.H. Gerding. *Autonomous Agents in Bargaining Games: An Evolutionary Investigation of Fundamentals, Strategies, and Business Applications.* Faculty of Technology Management, TU/e. 2004-08

N. Goga. *Control and Selection Techniques for the Automated Testing of Reactive Systems.* Faculty of Mathematics and Computer Science, TU/e. 2004-09

M. Niqui. *Formalising Exact Arithmetic: Representations, Algorithms and Proofs.* Faculty of Science, Mathematics and Computer Science, RU. 2004-10

A. Löh. *Exploring Generic Haskell.* Faculty of Mathematics and Computer Science, UU. 2004-11

I.C.M. Flinsenberg. *Route Planning Algorithms for Car Navigation.* Faculty of Mathematics and Computer Science, TU/e. 2004-12

R.J. Bril. *Real-time Scheduling for Media Processing Using Conditionally Guaranteed Budgets.* Faculty of Mathematics and Computer Science, TU/e. 2004-13

J. Pang. *Formal Verification of Distributed Systems.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-14

F. Alkemade. *Evolutionary Agent-Based Economics.* Faculty of Technology Management, TU/e. 2004-15

E.O. Dijk. *Indoor Ultrasonic Position Estimation Using a Single Base Station.* Faculty of Mathematics and Computer Science, TU/e. 2004-16

S.M. Orzan. *On Distributed Verification and Verified Distribution.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-17

M.M. Schrage. *Proxima - A Presentation-oriented Editor for Structured Documents.* Faculty of Mathematics and Computer Science, UU. 2004-18

E. Eskenazi and A. Fyukov. *Quantitative Prediction of Quality Attributes for Component-Based Software Architectures.* Faculty of Mathematics and Computer Science, TU/e. 2004-19

P.J.L. Cuijpers. *Hybrid Process Algebra.* Faculty of Mathematics and Computer Science, TU/e. 2004-20

N.J.M. van den Nieuwelaar. *Supervisory Machine Control by Predictive-Reactive Scheduling.* Faculty of Mechanical Engineering, TU/e. 2004-21

E. Ábrahám. *An Assertional Proof System for Multithreaded Java -Theory and Tool Support- .* Faculty of Mathematics and Natural Sciences, UL. 2005-01

R. Ruimerman. *Modeling and Remodeling in Bone Tissue.* Faculty of Biomedical Engineering, TU/e. 2005-02

C.N. Chong. *Experiments in Rights Control - Expression and Enforcement.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-03

H. Gao. *Design and Verification of Lock-free Parallel Algorithms.* Faculty of Mathematics and Computing Sciences, RUG. 2005-04