EFFICIENT EQUATIONAL REASONING FOR THE INST-GEN FRAMEWORK

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Engineering and Physical Sciences

2011

By
Christoph Sticksel
School of Computer Science

Contents

Word Count: 70 700

Abstract

We can classify several quite different calculi for automated reasoning in first-order logic as instantiation-based methods (IMs). Broadly speaking, unlike in traditional calculi such as resolution where the first-order satisfiability problem is tackled by deriving logical conclusions, IMs attempt to reduce the first-order satisfiability problem to propositional satisfiability by intelligently instantiating clauses.

The Inst-Gen-Eq method is an instantiation-based calculus which is complete for first-order clause logic modulo equality. Its distinctive feature is that it combines first-order reasoning with efficient ground satisfiability checking, which is delegated in a modular way to any state-of-the-art ground solver for satisfiability modulo theories (SMT). The first-order reasoning modulo equality employs a superposition-style calculus which generates the instances needed by the ground solver to refine a model of a ground abstraction or to witness unsatisfiability.

The thesis addresses the main issue in the Inst-Gen-Eq method, namely efficient extraction of instances, while providing powerful redundancy elimination techniques. To that end we introduce a novel labelled unit superposition calculus with sets, AND/OR trees and ordered binary decision diagrams (OBDDs) as labels. The different label structures permit redundancy elimination each to a different extent.

We prove completeness of redundancy elimination from labels and further integrate simplification inferences based on term rewriting. All presented approaches, in particular the three labelled calculi are implemented in the iProver-Eq system and evaluated on standard benchmark problems.

Lay Abstract

How can we be sure that software which we entrust more and more vital tasks to is free of flaws? Incidents of software maliciously being exploited or accidentally failing have had consequences that range from severe financial losses to threats to the lives of people. Good engineering is just a weak safeguard, and in particular in an ever faster paced and more complex world no guarantee that no flaw in the final product has been overlooked. Only formal verification can prove that a program will always behave as specified or that a specification is actually without loopholes.

Unfortunately, verification of real world applications is tantamount to finding a needle in a haystack. The problems are too large and too complex to tackle for any one person even with all human intuition and creativity. This is where automated reasoning comes in. It offers a mechanisation of the process of finding logical conclusions, sifting through the haystack and in relevant practical cases actually finding a needle. Then, either defects in the implementation or specification can be fixed, or the reliability of the software has been formally established.

This research is focused on a recent approach to automated reasoning that has already shown success in a range of cases. The goal is to allow specifications to use more expressive theories, leading to more concise formulations, in turn making the automated reasoning process more powerful. Ultimately, automated reasoning for verification will have an integral place in design processes, resulting in better and safer software.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the "Copyright") and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the "Intellectual Property") and any reproductions of copyright works in the thesis, for example graphs and tables ("Reproductions"), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see `http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=487`), in any relevant Thesis restriction declarations deposited in the University Library, The University Library's regulations (see `http://www.manchester.ac.uk/library/aboutus/regulations`) and in The University's policy on Presentation of Theses

Acknowledgements

First and foremost I have to thank my supervisors Konstantin Korovin and Renate Schmidt who I have worked with in such an enjoyable and successful way for the last three years.

Konstantin gave me deep insights into the topic and we had many fruitful discussions that led to the ideas developed in this thesis and also resulted in our two publications. I appreciate Renate's invaluable guidance and feedback throughout the PhD project. For the thesis I am particularly glad for her diligent proofreading, both down to missing commas and with many suggestions that have made the text much more accessible.

I am gratefully acknowledging the funding by the School of Computer Science and the Engineering and Physical Sciences Research Council (EPSRC). Further, the Royal Society have funded a cluster of computers I could access and the scale of the evaluations I was able to run had a great influence on the research.

On a personal level I am thankful to my office-mates with whom I could share and participate in technical and not-so-technical issues. Last but not least, I could have not done without the support from my wife and my family.

Chapter 1

Introduction

Information systems are ubiquitous. Many aspects of our daily lives depend on both software and hardware correctly executing the task we entrust them with. Accidental failures, malicious exploits or flawed designs of these systems can have severe consequences and good engineering should make all efforts to prevent these. With systems becoming more and more complex, simple approaches based on thorough testing are neither appropriate nor possible and formal methods become an essential part of the toolbox.

The interest in formal methods for system design and verification is growing and major industry players now integrate formal methods in their processes to develop dependable systems. One notable example is the design of the Intel Core™ i7 processor, where for the first time formal verification became the "primary validation vehicle" for the core execution cluster, with coverage oriented testing in a merely supplementary role [Kaivola et al., 2009]. Another example of active research in an industry setting is Microsoft, where several tools for verification of software have been developed. For instance, the Windows Driver Development Kit contains SLAM [Ball et al., 2010], which checks if a piece of software like a device driver satisfies mandatory behavioural properties, or Pex [Tillmann and de Halleux, 2008] integrated into the Visual Studio development environment for "white box test generation", which automatically produces a "small test suite with high code coverage".

Academia has long recognised verification as a "grand challenge" [Beckert et al., 2006] and automated reasoning as the heart of formal methods plays a central role in it. Most verification tasks can be directly posed in first-order logic or discharge proof obligations as first-order formulae.

In other application areas we can translate into certain fragments of first-order logic important non-classical logics such as description logics for reasoning in knowledge databases [Hustadt et al., 2004b] or more general many modal logics [Schmidt, 1999]. Moreover, the translated first-order problems can be efficiently decided with calculi for first-order reasoning. Therefore we can consider first-order logic it as the standard logic and due to its generality certainly as one of the most important logics.

Automated reasoning for first-order logic has a rather long history in Computer Science, but the canonical problem all other tasks are reduced to has always been the following: Prove the unsatisfiability of a formula. Despite the inherent undecidability of this problem, calculi have been developed that make it feasible for larger and larger fragments of first-order logic.

A number of diverse calculi and variants have been proposed, they have been in and out of fashion at times and perform differently for different kinds of problems. In this thesis we are interested in so-called instantiation-based methods that have recently gained attention as they promise some advantages over longer established methods (see, e.g., Baumgartner [2007]).

A first simple instantiation-based method was already presented in Gilmore [1960], but it was soon overshadowed by the success of the resolution calculus by Robinson [1965]. Two developments have lead to the recent interest and the development of "modern" instantiation-based calculi.

The first is the success of solvers for the decidable problem of propositional satisfiability with the DPLL method [Davis and Putnam, 1960, Davis et al., 1962]. Although this method was originally formulated as a first-order calculus, it is virtually exclusively employed for propositional reasoning. Sophisticated techniques were developed such that DPLL-based solvers for propositional satisfiability, so called SAT solvers, can tackle huge problems and have become valuable tools for certain verification tasks.

A second reason was the discovery of the decidable fragment of effectively propositional logic, also called the Bernays-Schönfinkel fragment, that can encode problems from domains as diverse as bounded model checking [Pérez and Voronkov, 2007], logic programming [Eiter et al., 2005], knowledge representation [Hustadt et al., 2004a] and hardware verification [Emmer et al., 2010]. Instantiation-based methods by their nature are decision procedures for this fragment and outperform traditional methods.

However, instantiation-based methods are not only restricted to the Bernays-Schönfinkel fragment. They are general calculi for first-order logic and have attractive features, some of which are complementary to other contemporary methods.

At the core of instantiation-based methods is a theorem, ultimately due to Herbrand's Theorem, which we state in the following form.

Theorem 1.1. *Let $\varphi(\bar{x})$ be a quantifier-free formula, then $\forall \bar{x}\ \varphi(\bar{x})$ is unsatisfiable if and only if there is a finite number of ground terms $\bar{t}_1, \ldots, \bar{t}_n$ such that $\bigwedge_i \varphi(\bar{t}_i)$ is unsatisfiable.*

We immediately obtain a refutationally complete method, which is the prototype of an instantiation-based method, by repeating two steps.

1. Guess a finite number of ground instances of $\forall \bar{x}\ \varphi(\bar{x})$.
2. Test ground satisfiability.

Since techniques for SAT solving are well explored and mature industrial-strength tools exist, instantiation-based methods can rely on such approaches for the second part. The central question and the distinguishing feature between instantiation-based methods is therefore how to efficiently find a set of ground instances of a first-order formula to witness first-order unsatisfiability.

An instantiation-based method searches in an intelligent way for ground instances of the input formula. In general, a quantifier-free formula has an infinite number of ground instances, but by Theorem 1.1 it suffices to find a finite subset. This approach is orthogonal to traditional methods, which usually derive logical conclusions or analyse the input formula. A number of properties make instantiation-based methods interesting in their own regard as well as a complement to traditional methods.

- They cover a different search space, solving a different set of problems. This is in particular exemplified by the decidable Bernays-Schönfinkel fragment, where formulae only have a finite number of ground instances. Hence, instantiation-based methods terminate after having produced all these instances. Therefore instantiation-based methods are natural decision procedures and particularly strong, unlike other methods in their standard formulations, which do not necessarily decide this fragment.

- The search for instances is usually guided by partial models, although not with the intention of finding a model of the input formula, but rather to find a set of unsatisfiable ground instances. As a side effect, in most instantiation-based methods it is easy to extract a model after the input formula has been found satisfiable.

- Since propositional reasoning is an integral part, instantiation-based methods usually show a good performance on propositional and nearly propositional formulae. Other methods do not have this advantage and in particular resolution is well-known to be weak on propositional formulae.

- Finally, unlike other methods, where new and longer formulae can be derived by recombining formulae, instantiation-based methods have the advantage of preserving the structure of formulae.

In many applications it is indispensable to have equational reasoning robustly built into the calculus. While theoretical foundations of equational reasoning in several instantiation-based methods have been laid already some time ago (Baumgartner and Tinelli [2005], Ganzinger and Korovin [2004], Letz and Stenz [2002]), due to a number of challenging issues only now practical implementations of the calculi appear.

In this thesis we take up equational reasoning with the Inst-Gen-Eq calculus as presented in Ganzinger and Korovin [2004], which is a novel approach not found in other calculi. The distinctive feature of the Inst-Gen-Eq method is a modular combination of superposition style first-order reasoning to find ground instances and ground satisfiability based on any off-the-shelf solver. The extension of the SAT problem to ground satisfiability modulo equality remains decidable, and instead of a SAT solver, we use a solver for satisfiabiliy modulo theories (SMT) to decide ground satisfiability modulo equality.

Our research hypothesis is that instantiation-based reasoning and in particular the Inst-Gen-Eq method can benefit from a robust integration of equational reasoning. We further test the hypothesis that in the Inst-Gen-Eq method the separation into and cooperation between first-order reasoning and ground reasoning as conceived in the non-equational Inst-Gen method extends to equational reasoning.

A major practical challenge in Inst-Gen-Eq is the efficient extraction of relevant substitutions from superposition proofs, which are used for instance generation. Here we need to explore potentially all non-redundant superposition proofs resulting in contradictions, extract relevant substitutions and efficiently propagate redundancy elimination from instantiation into superposition derivations. Non-trivial issues arise when combining literals equal up to renaming, where it is crucial to preserve the structure of proofs. For refutational completeness the necessary instances have to be generated, while for efficiency redundancy elimination with constraints should be preserved.

The main original contribution of the thesis is the Inst-Gen-Eq method with unit superposition on labelled literals to handle different literal variants in an elegant, uniform and efficient way. The labelling initially distinguishes between literal variants and the inference systems provide a merging rule to combine literal variants such that further inferences are simultaneously performed on all merged literal variants. The labelling approach has the advantages of replacing the extraction of substitutions from proofs and enabling redundancy elimination with both constraints and subsumption. We define and discuss inference systems for three label structures, namely sets, AND/OR trees and ordered binary decision diagrams (OBDDs), some enabling redundancy elimination to a greater extent at the cost of a higher computational complexity.

Besides redundancy elimination based on labels we lift simplification inferences based on rewriting and add a demodulation inference rule to the labelled inference systems. This is another non-trivial contribution due to interactions between the first-order reasoning and the ground satisfiability solving.

The methodology in the thesis comprises both formal proofs and empirical evaluation. We give an extensive completeness proof of the Inst-Gen-Eq method that features a general notion of redundancy to justify the concrete techniques presented afterwards. All theoretical aspects have been implemented in the iProver-Eq system using state-of-the art techniques and the system is efficient enough to be on a par with leading theorem proving systems. In particular, the system implements the three labelled calculi and we give experimental evidence of strengths and weaknesses.

Finally, there are directions for further work to move beyond equational reasoning, where further significant contributions can be expected.

Putting applications back into focus, we find that there is an amount of background knowledge required to solve problems in particular domains. This results in the approach of satisfiability modulo theories (SMT), which is a currently highly active research topic. Since virtually all theories of interest have the theory of equality embedded, the work in this thesis is an essential step towards a possible further development of the Inst-Gen method to reasoning modulo theories as already described in Ganzinger and Korovin [2006].

The main value of this line of research, building in robust equational reasoning into Inst-Gen and further extending the calculus to reason modulo theories, lies in connecting three areas of active research in automated reasoning: first-order reasoning, propositional satisability (SAT) solving and satisfiability modulo theories (SMT). By their nature, instantiation-based methods combine first-order reasoning and SAT solving. In particular the Inst-Gen method exhibits this combination in a clear modular way and has led to valuable insights. A hot topic in first-order reasoning is the integration of theory reasoning, while a complementary topic in SMT solving is to lift the efficient ground reasoning to first-order. Extending the Inst-Gen calculus to equational reasoning and to theories beyond has the potential to benefit these two concurrent developments.

The thesis is organised into the following chapters.

Chapter 2 provides a common ground of definitions and notation for the chapters to follow.

Chapter 3 gives an overview of the non-equational Inst-Gen method and demonstrates equational reasoning with Inst-Gen-Eq. We contrast the method with other instantiation-based methods and in particular their approaches to equational reasoning.

Chapter 4 formally defines the Inst-Gen-Eq method and proves refutational completeness. The proof provides more details than in the original publication by Ganzinger and Korovin [2004] and contributes a more restrictive inference rule based on superposition rather than ordered paramodulation. We also extend the notion of redundancy in order to allow simplifications with unit clauses.

Chapter 5 contains the main contribution of the thesis, introducing three labelled calculi for the Inst-Gen-Eq method and proving their completeness. The

label structures of sets, AND/OR trees and OBDDs differ in the way labels are merged, the precision of redundancy elimination and the existence of normal forms.

Chapter 6 is concerned with concrete techniques for redundancy elimination. Justified by the extended notion of redundancy in the proof of completeness we contribute two techniques for simplification that use unit clauses: subsumption by unit clauses and demodulation.

Chapter 7 is a system description of iProver-Eq, the implementation of the Inst-Gen-Eq calculus based on the non-equational system iProver. All of the features studied in previous chapters are implemented in iProver-Eq.

Chapter 8 backs up the theoretical contributions with experimental evidence, demonstrating the effects of the main contributions in the iProver-Eq system. We also report on a comparison with other systems.

Chapter 9 concludes and discusses further work.

Parts of Chapters 5 and 7 of this thesis were published as Korovin and Sticksel [2010a] and Korovin and Sticksel [2010b], respectively.

Chapter 2

Preliminaries

We use standard terminology and notions of first-order logic. In order to keep this thesis self-contained, we give definitions and notations of the central elements in this chapter.

Definition 2.1. A *signature* $\Sigma = \langle \Sigma_f, \Sigma_P \rangle$ is a pair of sets of *function symbols* and disjoint *predicate symbols*. We assume a unary function arity: $\Sigma_f \cup \Sigma_P \to \mathbb{N}$ that assigns to each function symbol $f \in \Sigma_f$ and each predicate symbol $P \in \Sigma_P$ a non-negative integer $\text{arity}(f) \geq 0$ and $\text{arity}(P) \geq 0$. Function symbols with arity zero are also called constant symbols and we assume that every signature Σ contains at least one constant symbol.

Definition 2.2. Let Σ be a signature and X be a set of *variables* with $X \cap \Sigma = \emptyset$. A *term* is either

(i) a variable $x \in X$ or

(ii) a functional term $f(t_1, \ldots, t_n)$ where $f \in \Sigma_f$ with $\text{arity}(f) = n$ and each t_i is a term.

By $\mathrm{T}(\Sigma, X)$ we denote the set of all terms built over the signature Σ with variables from X.

The set of variables of a term is recursively defined as

$$\text{var}(t) = \begin{cases} \{x\} & \text{if } t = x \\ \bigcup_{i=1}^{n} \text{var}(t_i) & \text{if } t = f(t_1, \ldots, t_n). \end{cases}$$

A term t is *ground* if $\text{var}(t) = \emptyset$ and we denote the set of ground terms over a signature Σ as $\mathrm{T}(\Sigma) = \mathrm{T}(\Sigma, \emptyset)$.

The set of *subterms* of a term t contains the term t itself and if $t = f(t_1, \ldots, t_n)$ also the subterms of $t_1, \ldots, t_n$.

A *position* in a term t is a sequence of integers, it denotes a subterm of t. The *empty sequence* λ denotes the term t itself and the position $p = q.i$ denotes the i-th subterm of the subterm at position q. For instance the subterm at position 2.1 in $f(a, g(b))$ is b.

We write $t[u]_p$ to express that the term u is a subterm of t at position p. Ambiguously we denote by $t[v]_p$ the result of replacing the subterm of t at position p with the term v.

Conventionally the symbols x, y, z, u, v and w denote variables, a, b and c are constant symbols and f, g and h function symbols, each decorated with $'$ and subscripts as necessary.

Definition 2.3. A *substitution* σ is a total function from variables to terms, which is the identity on all variables except a finite set, which we call the *domain* $\mathrm{dom}(\sigma)$ of the substitution. We write $\sigma = [t_1/x_1, \ldots, t_n/x_n]$ in order to explicitly list the image t_i of each variable x_i in the domain. We call the set $\{t_1, \ldots, t_n\}$ the *range* $\mathrm{rng}(\sigma)$ of the substitution.

The *restriction* of a substitution σ to a domain $Y \subset X$, written as $\sigma|_Y$ is the substitution where $x\sigma|_Y = x\sigma$ if $x \in \mathrm{dom}(\sigma) \cap Y$ and $x\sigma|_Y = x$ otherwise.

When applying a substitution σ to a term t we simultaneously replace each variable x in t with the term $\sigma(x)$. We usually write $t\sigma$ instead of $\sigma(t)$to denote the result of applying the substitution σ to the term t.

We extend the notion of applying a substitution to sets of terms and to substitutions by defining

$$S\sigma = \{t\sigma \mid t \in S\}$$

and

$$\sigma\tau = [t_1/x_1, \ldots, t_n/x_n]\,\tau = [t_1\tau/x_1, \ldots, t_n\tau/x_n]\,,$$

respectively.

If the domains of two substitutions σ and ρ are disjoint, the *union* of the substitutions is defined as

$$x(\sigma \cup \tau) = \begin{cases} x\sigma & \text{if } x \in \mathrm{dom}(\sigma), \\ x\tau & \text{if } x \in \mathrm{dom}(\tau) \text{ and} \\ x & \text{otherwise.} \end{cases}$$

If a substitution is injective, that is $\mathrm{rng}(\sigma) = \{x_1, \ldots, x_n\}$ where each x_i is a variable and $x\sigma \neq y\sigma$ if $x \neq y$, we call σ a *renaming*.

A substitution σ is *proper* if there is at least one variable $x \in \mathrm{dom}(\sigma)$ such that $x\sigma$ is not a variable. Otherwise a substitution is *non-proper*. The substitution $[z/x, z/y]$ is non-proper and not injective, therefore not a renaming.

A substitution σ is *more general* than a substitution τ if there is a substitution μ, which is not a renaming, such that $\sigma\mu = \tau$. A substitution σ *unifies* two terms $t \in \mathrm{T}(\Sigma, X)$ and $t' \in \mathrm{T}(\Sigma, X)$ if $t\sigma = t'\sigma$. The substitution σ is a *most general unifier* for t and t' if $t\sigma = t'\sigma$ and there is no substitution τ that unifies t and t' and is more general than σ. All substitutions which are a most general unifier for two terms t and t' are equal up to renaming and we arbitrarily choose one representative denoted $\mathrm{mgu}(t, t')$.

We write $\sigma \equiv \tau$ if $x\sigma = x\tau$ for each x.

Definition 2.4. An *atom* $P(t_1, \ldots, t_n)$ is a predicate, which consists of a predicate symbol $P \in \Sigma_P$ with $\mathrm{arity}(P) = n$ and n terms.

Definition 2.5. An *equation* is an atom with the binary predicate symbol $\simeq$, written in infix notation as $l \simeq r$. We view equations as multisets of size two and do not distinguish between $l \simeq r$ and $r \simeq l$.

A *disequation* is the negation of an equation, we usually write $l \not\simeq r$ instead of $\neg(l \simeq r)$.

In most parts of this thesis we consider without loss of generality pure equational logic, where equality $\simeq$ is the only predicate symbol in the signature. We translate a formula F containing a predicate other than $\simeq$ into a purely equational formula in the following way. We extend the signature with the distinct constant symbol $\top$ and a fresh function symbol $f_P \notin \Sigma_f$ for each non-equational predicate P in F such that $\mathrm{arity}(f_P) = \mathrm{arity}(P)$. We then replace each atom $P(t_1, \ldots, t_n)$ with $f_P(t_1, \ldots, t_n) \simeq \top$ and remove all predicate symbols except $\simeq$ from Σ_P.

In order to prevent producing ill-formed purely equational formulae, which cannot be translated back into the original signature by reversing the above procedure, we take a two-sorted approach: no function symbol f_P and neither the constant $\top$ may occur in the range of a substitution. In this way, we effectively prevent unifications of terms with atoms.

Definition 2.6. A *literal* L is an equation $l \simeq r$ or a disequation $l \not\simeq r$. For convenience and to avoid multiple negations we define the *complement* $\overline{L}$ of a literal as $\overline{l \simeq r} = l \not\simeq r$ and $\overline{l \not\simeq r} = l \simeq r$. A literal is *ground* if both terms l and r are ground.

A *clause* C is a multiset of literals and is to be interpreted as the disjunction of its literals: $C = \{L_1, \ldots, L_n\} = L_1 \vee \cdots \vee L_n$. We also write $L \vee C$ for the clause $\{L\} \cup C$.

We call a clause *ground* if all its literals are ground.

For any two clauses we assume variable-disjointness and we do not distinguish between variants of clauses. We say two clauses C and C' are *equal up to renaming* if there is a renaming ρ such that $C = C'\rho$.

Since any first-order formula can be effectively translated into a set of clauses, we restrict the presentation to such sets of clauses.

Definition 2.7. A *closure* is a pair of a clause C and a substitution θ written as $C \cdot \theta$ where

(i) $\mathrm{dom}(\theta) = \mathrm{var}(C)$ and

(ii) for all variables $x \in \mathrm{rng}(\theta)$ we have $x \notin \mathrm{var}(C)$.

We call a closure *ground* if $C\theta$ is ground.

We say two closures $C \cdot \theta$ and $C' \cdot \theta'$ are *equal up to renaming* if there is are renamings ρ and μ such that $C = C'\rho$ and $C\theta = C'\theta'\mu$.

An essential ingredient in equational reasoning is the notion of rewriting, which is based on term orderings.

Definition 2.8. A *simplification order* $\succ$ on terms over $\mathrm{T}(\Sigma, X)$ is a total and strict ordering satisfying the following properties

(i) $\succ$ is *well-founded*: every chain $s_1 \succ \cdots \succ s_n$ has a least element, there is not infinite chain

(ii) $\succ$ is *monotonic*: for all terms s, t and u if $s \succ t$, then also $u[s] \succ u[t]$.

(iii) $\succ$ is *closed under substitutions*: for all terms s and t if $s \succ t$, then $s\sigma \succ t\sigma$ for all substitutions σ.

(iv) $\succ$ satisfies the *subterm property*: $u[s]_p \succ s$ for all terms s and u and all positions $p \neq \lambda$

We extend $\succ$ from terms to multisets of terms and based on this to literals and clauses in one of the standard ways, described in Nieuwenhuis and Rubio [2001], obtaining a total, well-founded and monotone ordering.

We denote a simplification order on ground terms over $\mathrm{T}(\Sigma)$ by $\succ_{\mathrm{gr}}$.

In this thesis it suffices to consider rewriting of ground terms, resulting in the following slightly simpler definition of rewrite systems.

Definition 2.9. A *rewrite rule* $l \to r$ is a pair of ground terms. We call a rewrite rule *oriented* with respect to a ground ordering $\succ_{\mathrm{gr}}$ if $l \succ_{\mathrm{gr}} r$. A set of rewrite rules is a *rewrite system*.

We say a rewrite system R reduces a term $s[l]_p$ to $s[r]_p$ if it contains a rule $l \to r$. The term $s[l]_p$ is *reducible* at position p. If in a term s there is no position p reducible by a rule in R, we call the term s *irreducible*.

We extend the notion of irreducibility from terms to substitutions in the natural way: a substitution σ is irreducible with respect to a rewrite system R if none of the terms in $\mathrm{rng}(\sigma)$ is reducible by a rule $l \to r \in R$.

Let $\mathcal{L}$ be a set of ground literal closures and R be a ground rewrite system. We denote with $\mathrm{irred}_R(\mathcal{L})$ the set of closures $L \cdot \theta \in \mathcal{L}$ with θ irreducible with respect to R.

Let us denote by $\to_R$ the relation on ground terms induced by reducibility in the rewrite system R, such that $l \to_R r$ if l is reducible to r by some rule in R. Let $\overset{*}{\to}_R$ be the reflexive and transitive closure of the relation $\to_R$. We have $l \overset{*}{\to}_R r$ if $l = r$, if $l \to_R r$ or if there are terms $u_1, \ldots, u_n$ such that l is reducible to u_1, for each $1 < i < n$ the term u_i is reducible to u_{i+1} and u_n is reducible to r.

A rewrite system is called *terminating* if there is no infinite chain of terms $t_1 \to_R t_2 \to_R \cdots$ and *confluent* if for each term l, u_1 and u_2 with $l \overset{*}{\to}_R u_1$ and $l \overset{*}{\to}_R u_2$ there is a term r such that $u_1 \overset{*}{\to}_R r$ and $u_2 \overset{*}{\to}_R r$. A rewrite system is *convergent* if it is both terminating and confluent.

As usual, we are only concerned with Herbrand interpretations, where a term t is interpreted "as itself".

Definition 2.10. In order to speak about truth values, we view a Herbrand *interpretation* I as a set of ground literals over $\mathrm{T}(\Sigma)$. A ground literal L is true

in an interpretation I if it is a consequence modulo equality from literals in I. We write $I \models L$ and say "L is true in I" or "L holds in I". In contrast we say "L is false in I" if $I \models \overline{L}$. A non-ground literal L is true in I if all its ground instances are true in L.

An interpretation is *consistent* if for each ground literal L the complement $\overline{L}$ does not hold in I. If neither $I \models L$ nor $I \models \overline{L}$ we say L is *undefined* and we call I *total* if for each literal L either $I \models L$ or $I \models \overline{L}$. Otherwise I is *partial*.

A clause C is true in an interpretation I, written as $I \models C$, if at least one literal $L \in C$ holds in I. A set of clauses S is true in an interpretation if each clause $C \in S$ is true in I. We call I a *model* for S and write $I \models S$.

A set of clauses is *satisfiable* if it has a model.

Overloading the notation for $\models$ we also write $S \models C$ for a set of clauses S and a clause C if in every model for S at least one literal of C is true. For brevity we also use $\models$ with rewrite systems R intending $R \models C$ to be read as $S_R \models C$ with $S_R = \{l \simeq r \mid l \to r \in R\}$.

Definition 2.11. We use the symbol $\Box$ to denote either a literal or a clause that is false in every interpretation. We call it a *contradiction* or the *empty clause*.

Chapter 3

Inst-Gen, Inst-Gen-Eq and Related Work

In this chapter we present the Inst-Gen method as found in Ganzinger and Korovin [2003] and Ganzinger and Korovin [2004]. We first introduce the method for the non-equational case and its extension to equational reasoning in a less formal way and demonstrate it on two examples. We exhibit the challenges when moving to equational reasoning as it is considered for the remainder of the thesis. Finally we mention other instantiation-based methods and their approaches to equational reasoning.

3.1 Non-equational Reasoning with Inst-Gen

The main idea of the Inst-Gen method is as follows. Given a set of first-order clauses S we first form its ground abstraction $S\bot$ by mapping all variables to the same ground term, conventionally denoted by $\bot$. Overloading notation, we use $\bot$ also for the substitution that maps all variables to the ground term $\bot$. If the ground abstraction $S\bot$ is unsatisfiable, we have found an unsatisfiable set of ground instances of S. By Herbrand's Theorem the original set S is also unsatisfiable and the procedure terminates. Otherwise, there is a model $I_\bot$ of the ground abstraction $S\bot$ and the first-order instantiation process is guided by means of a selection function sel based on $I_\bot$. A selection function assigns to each first-order clause C in S exactly one literal $\mathrm{sel}(C) = L$ from C such that $I_\bot \models L\bot$. At least one such literal always exists as the ground abstraction of the clause is true in the model $I_\bot$.

Let us for the moment treat the equational predicate $\simeq$ as any other predicate and consider first-order logic without equality. If the set of selected (not necessarily ground) literals, seen as unit clauses, does not contain unifiable complementary literal pairs, a model for the clause set S exists and it has thus been proved satisfiable. Otherwise, there are two selected literals inconsistent in first-order. We instantiate the clauses these literals are selected in, such that the inconsistency can already be witnessed in the ground abstraction. Thus the ground model is refined in order to resolve the inconsistency.

The Inst-Gen method applies the following inference rule to the set of clauses S up to saturation. The inferences are interleaved with checking the ground abstraction $S\bot$ for unsatisfiability and updating the selection function sel to a model of $S\bot$ if it is satisfiable.

Definition 3.1 (Inst-Gen inference rule for non-equational reasoning)**.**

$$\frac{L \vee C \qquad \overline{L'} \vee D}{(L \vee C)\,\sigma \qquad \big(\overline{L'} \vee D\big)\,\sigma}$$

where

(i) $\mathrm{sel}\,(L \vee C) = L$, (ii) $\mathrm{sel}\,\big(\overline{L'} \vee D\big) = \overline{L'}$, (iii) $\sigma = \mathrm{mgu}\,(L, L')$.

The Inst-Gen inference rule detects inconsistent selected literals and generates clause instances such that the ground model has to be refined on the conflict. It is related to the resolution inference rule (see Bachmair and Ganzinger [2001]) with a few important differences. The Inst-Gen rule can be applied if two clauses can be resolved, that is contain complementary unifiable literals. However, while a resolution inference would combine the two clauses into the clause $(C \vee D)\,\sigma$, the Inst-Gen calculus keeps the clauses separate and instead creates two instances with the most general unifier σ. A second important difference is the selection function which is based on a model of the ground abstraction in the Inst-Gen case, whereas only negative literals are selected in resolution. For an Inst-Gen inference the literals L and $\overline{L'}$ have to be selected, while in resolution inferences the selection function together with an ordering determines eligible literals.

A main feature of the Inst-Gen approach is the delegation of the ground satisfiability check to an off-the-shelf SAT solver. Figure 3.1 on the facing page gives a bird's eye view of the resulting procedure. The dashed line separates first-order reasoning and ground reasoning; the latter is being dealt with in a black

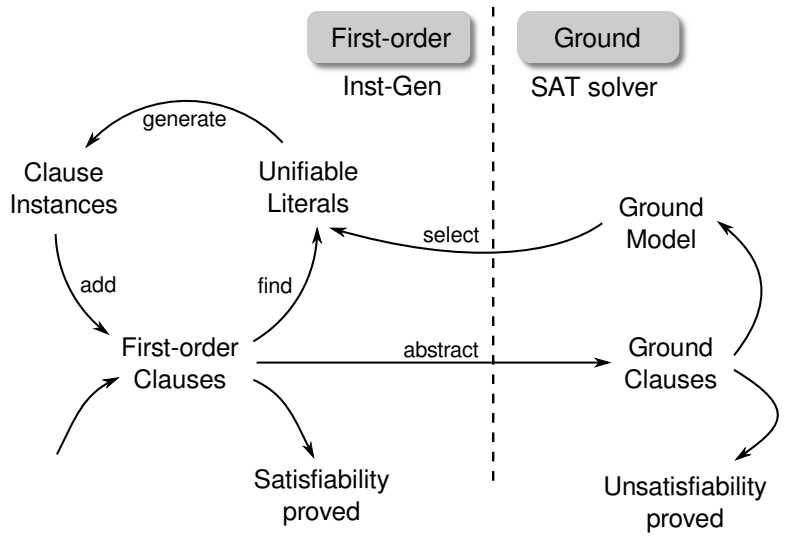


Figure 3.1: The Inst-Gen method without equality

boxed way by the SAT solver. The set of first-order clauses S is abstracted to a set of ground clauses $S\bot$ and passed to the SAT solver. If the ground abstraction is unsatisfiable, the procedure stops, having proved unsatisfiability of the first-order clause set S. Otherwise, a model for the ground abstraction returned by the SAT solver is used to select one literal in each first-order clause. If there is a unifiable complementary pair of selected literals, the Inst-Gen inference from Definition 3.1 is applied, resulting in new clause instances which are added to the first-order clause set. Should there be no unifiable complementary literals generating new clause instances, the procedure has proved satisfiability of the first-order clause set.

Example 3.1. Let us consider the following unsatisfiable clause set.

$$\underline{\neg Q(f(x))} \tag{1}$$

$$\underline{\neg P(f(f(y)))} \tag{2}$$

$$\underline{P(f(z))} \ \vee \ Q(z) \tag{3}$$

By mapping all variables in all clauses to the distinguished constant $\bot$ we

obtain the ground abstraction.

$$\neg Q(f(\bot)) \tag{$1_\bot$}$$
$$\neg P(f(f(\bot))) \tag{$2_\bot$}$$
$$P(f(\bot)) \ \vee \ Q(\bot) \tag{$3_\bot$}$$

The ground clauses ($1_\bot$)-($3_\bot$) are satisfiable with a model $I_\bot$ in which, say, the first literals in each clause, namely, $\neg Q(f(\bot))$, $\neg P(f(f(\bot)))$ and $P(f(\bot))$ are true. In practice, finding the ground model $I_\bot$ is delegated to the SAT solver. The selection function sel now selects the first literal in each first-order clause (1)-(3), since $I_\bot \models \neg Q(f(x))\bot$, $I_\bot \models \neg P(f(f(y)))\bot$ and $I_\bot \models P(f(z))\bot$. We have underlined the selected literals in the clauses (1)-(3) above.

We attempt to extend the ground model to a first-order model by assuming it contains all ground instances of a selected first-order literal. However, this fails since we have

$$\neg P(f(f(y))) \models \neg P(f(f(a))$$

for the selected literal in clause (2) as well as

$$P(f(z)) \models P(f(f(a))$$

for the selected literal in clause (3). The interpretation induced by the selection function on the first-order clauses is contradictory because it contains both $P(f(f(a))$ and its negation $\neg P(f(f(a))$.

The Inst-Gen method now tries to refine the model by generating clause instances from the conflicting selected literals above. We perform the following Inst-Gen inference.

$$\frac{\underline{\neg P(f(f(y)))} \qquad \underline{P(f(z)) \vee Q(z)}}{\neg P(f(f(y))) \qquad P(f(f(y)) \vee Q(f(y))}[f(y)/z]$$

The two premises are separately instantiated with the most general unifier $[f(y)/z]$. The first conclusion is a variant of the premise and can be omitted since its ground abstraction is the same as the ground abstraction of the premise and therefore does not contribute to the refinement of the ground abstraction. The second conclusion is added to the clause set after its variables have been

made disjoint from the variables occurring there.

$$P(f(f(u))) \ \vee \ Q(f(u)) \tag{4}$$

The ground abstraction of the clause set with the new instance is

$$\neg Q(f(\bot)) \tag{$1_\bot$}$$
$$\neg P(f(f(\bot))) \tag{$2_\bot$}$$
$$P(f(\bot)) \ \vee \ Q(\bot) \tag{$3_\bot$}$$
$$P(f(f(\bot))) \ \vee \ Q(f(\bot)). \tag{$4_\bot$}$$

The ground clauses ($1_\bot$), ($2_\bot$) and ($4_\bot$) are unsatisfiable, which is detected by the SAT solver. Hence we have found an unsatisfiable set of ground instances and by Herbrand's Theorem the initial first-order clause set (1)-(3) is unsatisfiable.

The Inst-Gen method can be extended to reasoning modulo equality, robustly dealing with the particular properties of the equational predicate $\simeq$, by the way of the congruence it induces on terms. While it is straightforward to replace the SAT solver with a ground solver modulo equality, the first-order reasoning needs non-trivial adaptions.

In particular, it is not sufficient to employ an atomic inference rule on literal pairs for instance generation as in the non-equational case. While the Inst-Gen inference rule guarantees that at least one of the conclusions is a proper instance of its premise (see Ganzinger and Korovin [2003]), this property does not hold in the equational case.

Equational reasoning in first-order logic is commonly performed with para-modulation-based calculi, which are a kin to resolution. The central inference rule is the following superposition rule (from Bachmair and Ganzinger [1998]).

Definition 3.2 (Superposition)**.**

$$\frac{l \simeq r \ \vee \ C \qquad s[l'] \simeq t \ \vee \ D}{(s[r] \simeq t \ \vee C \ \vee \ D)\sigma}\,(\sigma) \qquad \frac{l \simeq r \ \vee \ C \qquad s[l'] \not\simeq t \ \vee \ D}{(s[r] \not\simeq t \ \vee C \ \vee \ D)\sigma}\,(\sigma)$$

where

(i) $\sigma = \mathrm{mgu}(l, l')$,

(ii) l' is not a variable,

(iii) $l\sigma \succ r\sigma$,

(iv) $s[l']\sigma \succ t\sigma$,

(v) $(s[l'] \simeq t)\,\sigma \succ (l \simeq r)\,\sigma$.

As the following example shows, generating instances from the superposition rule in the style the Inst-Gen inferences rule generates instances from the resolution rule is incomplete.

Example 3.2. Consider the following unsatisfiable purely equational clause set.

$$\underline{h(x) \simeq x} \ \vee \ x \not\simeq a \tag{1}$$

$$\underline{f(h(y)) \simeq g(z)} \tag{2}$$

$$\underline{f(a) \not\simeq g(u)} \tag{3}$$

The ground abstraction is satisfiable modulo equality with a model that contains the first literal in each clause and the selection function thus selects the first literals in clauses (1)-(3), which we have underlined.

A superposition inference is possible between the selected literals in clauses (1) and (2). If we were to generate instances from superposition, the inference would be the following.

$$\frac{h(x) \simeq x \qquad f(h(y)) \simeq g(z)}{h(x) \simeq x \qquad f(h(x)) \simeq g(z)}[x/y]$$

Instead of combining the two premises into one conclusion, both are instantiated with the most general unifier. However, in this case only two variants of the premises are generated which have no effect on the ground abstraction.

Since no further inference steps are possible, and the ground abstraction is satisfiable, the Inst-Gen method would stop without showing the unsatisfiability of the clause set.

While we approach equational reasoning by extending the calculus in this thesis, an alternative approach is to add axioms to the clause set that capture the semantics of the equational predicate $\simeq$.

$$x \simeq x \tag{R}$$

$$x \not\simeq y \ \vee \ y \simeq x \tag{S}$$

$$x \not\simeq y \vee y \not\simeq z \ \vee \ x \simeq z \tag{T}$$

$$x_1 \not\simeq y_1 \vee \cdots \vee x_n \not\simeq y_n \ \vee \ f(x_1, \ldots, x_n) \simeq f(y_1, \ldots, y_n) \tag{F$_f$}$$

$$x_1 \not\simeq y_1 \vee \cdots \vee x_n \not\simeq y_n \vee \neg P(x_1, \ldots, x_n) \ \vee \ P(y_1, \ldots, y_n) \tag{M$_P$}$$

The monotonicity axioms (F$_f$) and (M$_P$) have to be instantiated for every function symbol f and predicate symbol P, respectively, in the signature Σ.

The axiomatic approach in resolution theorem proving is usually inferior to paramodulation-based approaches, which can make use of powerful ordering restrictions from rewriting and we expect similar benefits over an axiomatic treatment of equality in Inst-Gen. However, we notice that if the input clause set is in the Bernays-Schönfinkel fragment of first-order logic, it does not contain function symbols, no functional monotonicity axiom (F$_f$) is needed and the remaining axioms do not introduce function symbols, either. Since instantiation-based methods, including Inst-Gen, are strong in the Bernays-Schönfinkel fragment, it seems feasible to tackle equational reasoning in this fragment with a simple axiomatic handling of equality. We postpone this thought until the evaluation in Chapter 8 and now turn to equational reasoning with a paramodulation-based approach that can handle full first-order logic with equality.

3.2 Equational Reasoning in Inst-Gen-Eq

In the remainder of the thesis we consider the extension of the Inst-Gen method to equational reasoning, called Inst-Gen-Eq. Figure 3.2 on the next page shows an overview of the reasoning process, which is similar to non-equational reasoning in Figure 3.1 on page 31 with some adaptions.

Since the ground reasoning has to be performed modulo equality, a SAT solver is not sufficient. In order to keep with the Inst-Gen principle of delegating the ground reasoning to an off-the-shelf tool, we use a solver for satisfiability modulo theories (SMT) of which many are available. Virtually all of them are able to

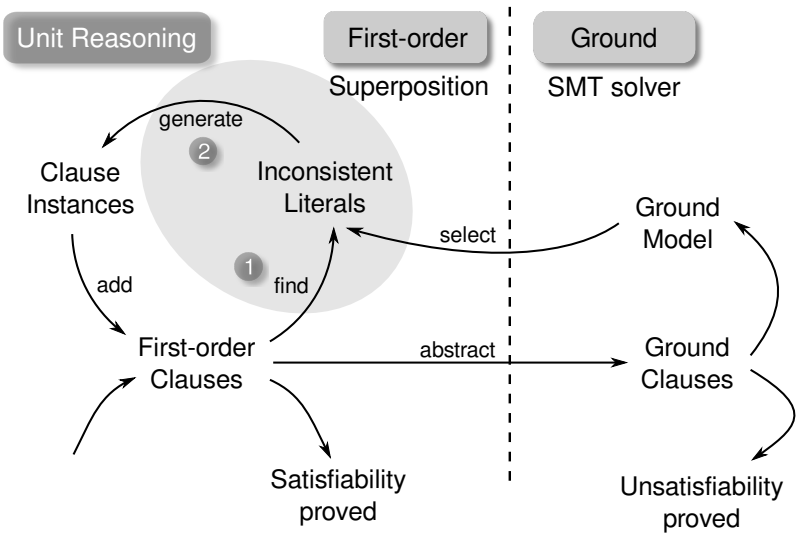


Figure 3.2: The Inst-Gen method with equality

reason in the theory of equality with uninterpreted function symbols (EUF), which is exactly the theory we require.

The main focus for equational reasoning is the shaded area in the first-order part: superposition reasoning with selected literals which we regard as unit clauses. In Example 3.2 above we have the set of selected literals

$$\{\ h(x) \simeq x,\ f(h(y)) \simeq g(z),\ f(a) \not\simeq g(u)\ \}$$

which is unsatisfiable if seen as the conjunction of literals in first-order modulo equality. We call this set of inconsistent literals a conflict. In order to refine the ground abstraction, clause instances have to be generated from this conflict. As we are showing in one of the following sections, there is a proper substitution for at least one literal in every conflict if the selection is based on a model of the ground abstraction.

For equational reasoning we are in a situation where we have to generate instances to refine the ground abstraction not only from pairs of selected literals, but from sets of selected literals of any finite size. We note that a conflict can be as small as a singleton set, for example

$$\{\ f(x) \not\simeq f(a)\ \}.$$

In order ① to find conflicts in the set of selected literals and ② to generate

instances from conflicts, we step back from the atomic Inst-Gen inference rule in Definition 3.1 on page 30, which combines finding conflicts and the generation of instances. Instead, we use two paramodulation-style inference rules.

Definition 3.3 (Unit Superposition Calculus)**.**

Unit Superposition

$$\frac{l \simeq r \qquad s[l'] \simeq t}{(s[r] \simeq t)\,\sigma}\,(\sigma) \qquad\qquad \frac{l \simeq r \qquad s[l'] \not\simeq t}{(s[r] \not\simeq t)\,\sigma}\,(\sigma)$$

where for some grounding substitution θ with $\mathrm{dom}\,(\theta) = \mathrm{var}\,(\{l\sigma, r\sigma, s[l']\sigma, t\sigma\})$

(i) $\sigma = \mathrm{mgu}(l, l')$,

(ii) l' is not a variable,

(iii) $l\sigma\theta \succ_{\mathrm{gr}} r\sigma\theta$,

(iv) $s[l']\sigma\theta \succ_{\mathrm{gr}} t\sigma\theta$,

(v) $\mathrm{var}\,(\{l, r\}) \cap \mathrm{var}\,(\{s[l'], t\}) = \emptyset$.

Equality Resolution

$$\frac{(l \not\simeq r)}{\square}\,(\sigma)$$

where $\sigma = \mathrm{mgu}(l, r)$

If a set of selected literals is a conflict, it is possible to derive a contradiction $\square$ with the inference rules of the unit superposition calculus. We annotate each inference step with the substitution σ to the right of the inference line and by composing the substitutions along each branch in a derivation we find substitutions to instantiate clauses with.

Example 3.3. Let us reconsider the clause set from Example 3.2 on page 34.

$$\underline{h(x) \simeq x} \;\vee\; x \not\simeq a \tag{1}$$

$$\underline{f(h(y)) \simeq g(z)} \tag{2}$$

$$\underline{f(a) \not\simeq g(u)} \tag{3}$$

There is a proof of a contradiction from selected literals L_1, L_2 and L_3 from

clauses (1), (2) and (3), respectively, with the unit superposition calculus.

$$\dfrac{\dfrac{\dfrac{\overset{L_1}{h(x) \simeq x} \qquad \overset{L_2}{f(h(y)) \simeq g(z)}}{f(x) \simeq g(z)}\,[x/y] \qquad \overset{L_3}{f(a) \not\simeq g(u)}}{g(z) \not\simeq g(u)}\,[a/x]}{\square}\,[z/u] \tag{$*$}$$

For each leaf literal we now compose the substitutions on its branch ending in a contradiction in the proof tree, obtaining

$$\sigma_1 = [x/y][a/x][z/u] \mid_{\mathrm{var}(L_1)=\{x\}} = [a/x],$$

$$\sigma_2 = [x/y][a/x][z/u] \mid_{\mathrm{var}(L_2)=\{y,z\}} = [a/y]$$

and

$$\sigma_3 = [a/x][z/u] \mid_{\mathrm{var}(L_3)=\{u\}} = [z/u]$$

for L_1, L_2 and L_3, respectively. Since σ_3 is a renaming and leads to the a variant of clause (3) only, we do not need to consider it. Applying σ_1 and σ_2 to clause (1) and (2), respectively, results in two new instances.

$$h(a) \simeq a \ \vee \ a \not\simeq a \tag{4}$$

$$f(h(a)) \simeq g(w). \tag{5}$$

The ground abstraction of the whole clause set is

$$h(\bot) \simeq \bot \ \vee \ \bot \not\simeq a \tag{$1_\bot$}$$

$$f(h(\bot)) \simeq g(\bot) \tag{$2_\bot$}$$

$$f(a) \not\simeq g(\bot) \tag{$3_\bot$}$$

$$h(a) \simeq a \ \vee \ a \not\simeq a \tag{$4_\bot$}$$

$$f(h(a)) \simeq g(\bot). \tag{$5_\bot$}$$

Clauses ($3_\bot$), ($4_\bot$) and ($5_\bot$) are unsatisfiable and we have found ground instances of the initial clause set (1)-(3) which is hence proved to be unsatisfiable.

The Inst-Gen-Eq method proceeds in analogy to the non-equational Inst-Gen method in the previous section, see again Figure 3.2 on page 36. A set of first-order clauses S is grounded to a set of clauses $S\bot$, which is passed to an SMT

solver to be tested for satisfiability modulo equality. Then either unsatisfiability of $S\bot$ is found by the solver and thus S is proved unsatisfiable. Otherwise, there is a model of $S\bot$, which is used to select one literal in each first-order clause in S by means of the selection function sel. In order to find conflicts in the set of selected literals, we saturate the set under inferences from the unit superposition calculus. Every contradiction $\Box$ that is found in the process, yields substitutions for clauses in S, which are in turn propagated to the ground solver. The process then follows another cycle, where either the ground solver finds unsatisfiability of $S\bot$ or returns a model of it. If the set of selected literals is saturated under unit superposition inferences and all clause instances from conflicts have been added to the clause set, we have found satisfiability of the original clause set.

Since Inst-Gen-Eq only produces instances of clauses, it is sound. The variables in clauses are universally quantified and therefore each clause already implies all its instances. We formally prove completeness in the following chapter.

3.3 Related Instantiation-based Methods

We briefly turn to contemporary instantiation-based methods, their proof procedures and the way equational reasoning is integrated. There are surveys [Baumgartner and Thorstensen, 2009] and comparisons [Jacobs and Waldmann, 2005] of instantiation-based methods, which are only concerned with non-equational reasoning. No such surveys exist for equational reasoning, showing that equational reasoning in instantiation-based methods is a recent and ongoing effort without definite answers so far.

Instantiation-based methods can be classified mainly in two aspects: the way ground solving is integrated and the control structure used in the proof procedure. Inst-Gen is a two-level method, since the ground solving is done in a separate step by a separate tool. In other methods the ground solving is part of the calculus. Further, Inst-Gen is a saturation-based procedure and as such more related to resolution, while some methods use tableaux structures making them resemble more to tableaux methods.

Let us give two relevant and recent examples of instantiation-based calculi and show the relation to and differences with the Inst-Gen method.

The Disconnection Calculus originated in Billon [1996] as the "Disconnection Method" and was significantly extended by Letz and Stenz [2001b]. An implementation is described by Letz and Stenz [2001a].

From a set of clauses as input, a clausal tableau is built, that is, each node is a literal. The literals can contain free variables, but contrary to standard tableaux calculi, the variables are non-rigid and there are no destructive substitutions that would require backtracking to a previous tableau. The calculus is confluent and the only choice points for backtracking are at nodes where different branches have to be explored.

The central concept in the calculus are links between clauses. A link between two clauses exists, when two clauses contain complementary unifiable literals. In this case a resolution inference or an Inst-Gen inference could be applied. Another ingredient is an initial path, which is a selection of one literal in each input clause. Only links with the literals on the initial path are allowed, there is no restriction on the links with literals that are added to the tableau.

Inferences are performed between linked literals on a branch of the tableaux and consist of instantiating the linked clauses in the same way as in the Inst-Gen method and extending the tableau. Afterwards the link is removed, the literals are "disconnected". Propositional satisfiability is tested for each branch separately by grounding all literals with a distinct constant, branches, where the grounding is unsatisfiable are closed and if all branches of the tableau become closed, unsatisfiability is detected.

The Disconnection method is similar to the Inst-Gen method in that the calculi share the intuition behind their inference rule and the principle of grounding. However, Inst-Gen is a two-level method and delegates the propositional satisfiability check, while Disconnection is a one-level method, since the propositional satisfiability check is a part of the proof procedure. Another difference is that Inst-Gen saturates the set of clauses under inferences, whereas Disconnection employs a tableau as control structure.

Equational reasoning in the Disconnection calculus is performed by extending the notion of a link to be similar to paramodulation. A link exists between an equation and a literal if the left-hand side of the equation can be unified with a subterm of the literal. In a similar way than before, the tableau is extended with instances from the instance and the link is disconnected.

There are several other instantiation-based methods based on the principle

of links between clauses, for example Ordered Semantic Hyperlinking (OSHL) [Plaisted and Zhu, 2000] or Partial Instantiation [Hooker et al., 2002].

Model Evolution as defined in Baumgartner and Tinelli [2003] can be viewed as a lifting of the powerful propositional DPLL procedure for SAT solving to first-order logic. A predecessor of Model Evolution was the FDPLL calculus in Baumgartner [2000], but Model Evolution goes a step further by lifting more aspects admitting to a more powerful implementation in the theorem prover Darwin, see Baumgartner et al. [2005].

Sequents of the form $\Lambda \vdash \Phi$ are processed, where Λ is an initially empty context to be interpreted as a candidate model for the clause set and Φ contains the remaining clauses. The calculus tries to extend the partial candidate model to a complete model by asserting truth values for literals in Φ in the context Λ. If it does not succeed, it backtracks to an earlier choice point and returns unsatisfiability if no more choice points are available. If the clause set Φ is empty, the context Λ can be interpreted as a model for the initial clause set. The calculus has several rules, each extending the current context or trying to find a contradiction.

The procedure is controlled by a tableau with sequents $\Lambda \vdash \Phi$ at its nodes. Candidate models are local to their branch. There is no separate propositional solver involved as this is part of the calculus, similar to closing a branch as in the Disconnection calculus. Therefore, Model Evolution is also a one-level method.

The relationship to resolution and the Inst-Gen calculus is only distant. The calculus does not consider links between clauses, it is only guided by the candidate model and inferences only operate on a single clause. The procedure is controlled by a tableaux where rule applications are local to a branch and generate new clauses to saturate the a clause set as in resolution and Inst-Gen. Model Evolution can be classified as instance-based nevertheless, because it refines candidate models with instances of literals Φ and in case of satisfiability, a model is easily retrieved from the context Λ in a representation similar to the Disconnection calculus.

There are two different extensions of Model Evolution to equational reasoning. The first approach [Baumgartner and Tinelli, 2005] that has been revised and implemented in the E-Darwin system is described in Baumgartner et al. [2010]. A constraint notation is added to clauses in the clause set Φ and equational reasoning is performed with superposition on the context Λ.

A more recent approach in Baumgartner and Waldmann [2009] takes a different turn and combines the superposition with model evolution by labelling each atom as either a split atom or a superposition atom. Split atoms are treated with the Model Evolution calculus, while superposition atoms are used for superposition. If all atoms are labelled as split atoms or superposition atoms, the calculus becomes identical to Model Evolution or superposition, respectively. Although this idea looks promising and the approach of combining an instantiation-based method with a paramodulation-based method is attractive, the calculus is not mature enough, requiring an extensive formalism for the interaction between the two components and lacks an implementation.

Chapter 4

Completeness of Inst-Gen-Eq

After the brief introduction of the Inst-Gen-Eq method in the previous chapter in the context of and in contrast to other instantiation-based methods, we now give a comprehensive proof of its completeness. While Inst-Gen-Eq has already been proved complete in Ganzinger and Korovin [2004] and the proof we present closely follows theirs, the main contributions of this section are two extensions. The original proof has used ordered unit paramodulation to reason on literals, whereas we strengthen the ordering constraints and consider unit superposition, which is not an obvious extension in an instantiation-based framework. Moreover, we extend the saturation process in order to justify the special treatment of unit clauses in Chapter 6. Introducing the proof in this chapter keeps the thesis self-contained and allows to adapt presentation and notation bearing in mind the following chapters. Finally, we have the opportunity to explicitly spell out and to describe each step in the proof.

Let us first remind ourselves of the important notion of a ground closure (see Definition 2.7 on page 26) before we give a more detailed overview of the proof. A *ground closure* $C \cdot \theta$ is a pair consisting of a clause C and a substitution θ such that $C\theta$ is ground. We assume that $\mathrm{dom}\,(\theta) = \mathrm{var}\,(C)$ and consider equality of closures modulo renaming: we do not distinguish between a closure $C \cdot \theta$ and $D \cdot \rho$ if $C\theta = D\rho$ and C is a variant of D.

We think of a ground closure $C \cdot \theta$ as representing the ground instance $C\theta$ of the clause C. We also call a closure $C \cdot \theta$ a ground instance of a set of clauses S if $C \in S$. Considering ground instances not as clauses but instead as closures is necessary to distinguish between and to compare ground instances from different clauses. Although this enables us to define redundancy in a fine-grained way, in a

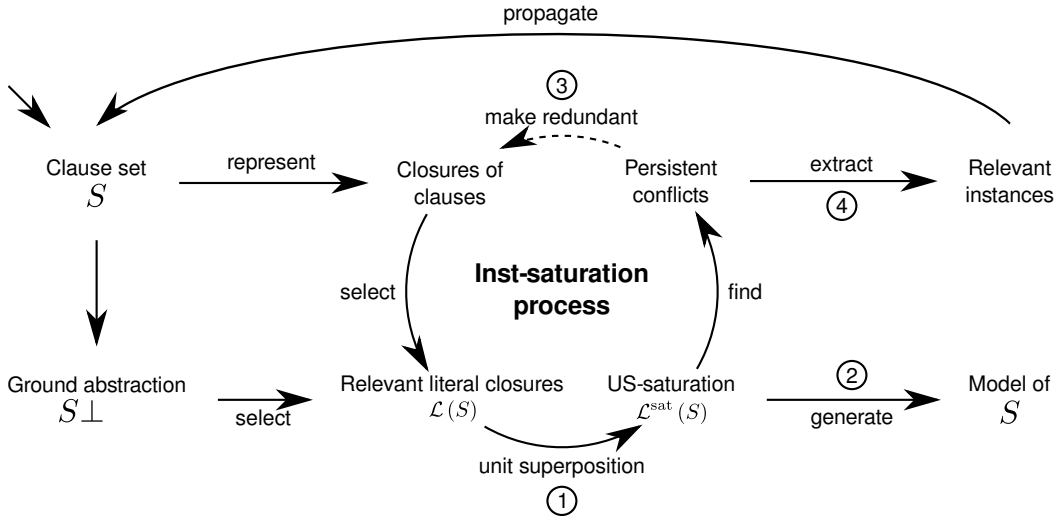


Figure 4.1: Road map for the completeness proof of the Inst-Gen-Eq method

practical implementation it is not sensible to work on the level of ground closures. Therefore, in following sections we lift the results from closures to clauses and cover redundancy with constraints on clauses.

Figure 4.1 shows a road map for the proof of completeness of the Inst-Gen-Eq method as an overview of this chapter. We begin with a set of clauses S and its ground abstraction $S\bot$. The clauses are represented by ground closures in an Inst-saturation process. A redundancy criterion and the selection function sel based on a model of the ground abstraction $S\bot$ lead to a set of relevant literal closures $\mathcal{L}(S)$ from the ground closures of the clauses in S. Applying the unit superposition calculus in a fair way we obtain the US-saturation $\mathcal{L}^{\mathrm{sat}}(S)$ which is closed under unit superposition inferences and free from US-redundancy (① in the figure). If the US-saturation does not contain a contradiction $\Box$, we can generate a model of the initial clause set S (② in the figure). Otherwise, there are finite sets of closures from which a contradiction $\Box$ can be derived. We call these sets conflicts. For completeness, a fair Inst-saturation process must make all conflicts redundant (③ in the figure). An effective way to make a conflict redundant is to extract relevant clause instances from the proof of the contradiction $\Box$ and to propagate the clause instances to the clause set S (④ in the figure).

In the next sections we follow the above sequence to show that Inst-Gen-Eq is refutationally complete. That is, if the initial clause set S is unsatisfiable, a fair Inst-saturation process eventually obtains a clause set S' with an unsatisfiable abstraction $S'\bot$. On the other hand, the clause sets produced by adding instances

from conflicts in a satisfiable initial clause set S always have satisfiable ground abstractions. Further, if the Inst-saturation process encounters a saturated set of literal closures without conflicts, satisfiability of S can be concluded.

The proof techniques of model generation and saturation processes that we employ in the following have been used in completeness proofs of the resolution calculus [Bachmair and Ganzinger, 2001] or several variants of the paramodulation calculus, see [Bachmair et al., 1995, Nieuwenhuis and Rubio, 2001]. However, in order to prove completeness of Inst-Gen-Eq, we have to take into account the nested Inst-saturation and US-saturation processes.

Since closures and literal closures are of little practical use, we lift the proof of completeness to first-order literals and subsequently introduce a constraint notation to lift the concept of redundancy on closures. We finish the chapter with remarks on the combination of ground solving, Inst-saturation and US-saturation towards a practical implementation.

4.1 Unit Superposition on Literal Closures

We begin by refining the unit superposition calculus from literals (Definition 3.3 on page 37) to ground literal closures. The following definition is a stronger version of the unit paramodulation calculus given by Ganzinger and Korovin [2004], which is in turn based on the standard superposition calculus, see, e.g., Nieuwenhuis and Rubio [2001].

Definition 4.1 (Unit Superposition on closures)**.**

Unit Superposition

$$\frac{(l \simeq r) \cdot \theta_{\mathrm{l}} \qquad (s[l'] \simeq t) \cdot \theta_{\mathrm{r}}}{(s[r] \simeq t)\sigma \cdot \rho|_{\mathrm{var}(\{s[r]\sigma, t\sigma\})}}\,(\sigma) \qquad \frac{(l \simeq r) \cdot \theta_{\mathrm{l}} \qquad (s[l'] \not\simeq t) \cdot \theta_{\mathrm{r}}}{(s[r] \not\simeq t)\,\sigma \cdot \rho|_{\mathrm{var}(\{s[r]\sigma, t\sigma\})}}\,(\sigma)$$

where

(i) $\sigma = \mathrm{mgu}(l, l')$,

(ii) l' is not a variable,

(iii) $l\theta_{\mathrm{l}} \succ_{\mathrm{gr}} r\theta_{\mathrm{l}}$,

(iv) $s[l']\theta_{\mathrm{r}} \succ_{\mathrm{gr}} t\theta_{\mathrm{r}}$,

(v) ρ is such that $(\theta_{\mathrm{l}} \cup \theta_{\mathrm{r}}) = \sigma\rho$

(vi) $l\theta_{\mathrm{l}} = l'\theta_{\mathrm{r}}$,

(vii) $\mathrm{var}(\{l, r\}) \cap \mathrm{var}(\{s[l'], t\}) = \emptyset$.

Equality Resolution

$$\frac{(l \not\simeq r) \cdot \theta}{\square}\,(\sigma)$$

where

(i) $\sigma = \mathrm{mgu}(l, r)$, (ii) $l\theta = r\theta$.

Conditions (i)-(iv) of the unit superposition inference rule and condition (i) on the equality resolution inference rule are identical to conditions on the corresponding standard superposition inference rules in the literature (see Definition 3.2 on page 34, or Nieuwenhuis and Rubio [2001]).

Conditions (v) and (vi) on the superposition inference and condition (ii) on the equality resolution inference connect the ground literals that the literal closures in the premises represent, in order to make the reasoning on ground instances visible. We use the most general unifier σ for the terms l and l' as in a first-order superposition inference and $l\theta_\mathrm{l} = l'\theta_\mathrm{r}$ in condition (vi) ensures that the superposition inference also applies for the ground literals represented by the closures. The same holds for the equality resolution inference, where we only consider literal closures to be contradictory if the represented ground literals are. In a unit superposition inference, by means of condition (v) we form the union of the variable-disjoint substitutions θ_l and θ_r from the premises and "pull out" the substitution σ. We restrict the resulting ρ and take the closure $(s[r] \simeq t)\,\sigma \cdot \rho|_{\mathrm{var}(\{s[r]\sigma, t\sigma\})}$ as the conclusion. Hence a unit superposition inference and an equality resolution inference on closures can be viewed both as a first-order inference and as a ground inference, the former by dropping the substitution part of the closures and the latter by considering the ground literals represented by the closures.

Since closures $C \cdot \theta$ are defined such that $\mathrm{dom}\,(\theta) = \mathrm{var}\,(C)$, that is the substitution θ affects exactly the variables of the clause, closures produced by inference rules have to respect this definition. Restricting the domain of ρ to $\rho|_{\mathrm{var}(\{s[r]\sigma, t\sigma\})}$ in the conclusion ensures that the closure is well-defined.

Due to condition (vii) the literals in the literal closures in premises are variable disjoint and therefore the domains of the substitutions θ_l and θ_r in the literal closures are variable disjoint. The union $\theta_\mathrm{l} \cup \theta_\mathrm{r}$ of the two substitutions in condition (v) can thus always be formed.

It is important to notice that given two variable disjoint premises for the first inference rule which satisfy conditions (i)-(iv), there is always a substitution ρ

satisfying the remaining conditions (v)-(vii). Due to conditions (i) and (vi) both σ and $\theta_{\mathrm{l}} \cup \theta_{\mathrm{r}}$ are unifying substitutions for l and l'. Since σ is the most general unifier, either $\sigma = \theta_{\mathrm{l}} \cup \theta_{\mathrm{r}}$ and thus $\rho = []$ or there is a substitution ρ such that $\sigma\rho = \theta_{\mathrm{l}} \cup \theta_{\mathrm{r}}$.

Unit superposition as defined above has three properties that play important roles in the completeness proof of the Inst-Gen-Eq calculus.

Lemma 4.1. *In a superposition inference the literal in the conclusion follows from literals in the premise instantiated with the most general unifier σ*

$$(l \simeq r)\,\sigma\rho, (s[l'] \simeq t)\,\sigma\rho \models (s[r] \simeq t)\,\sigma\rho$$

and

$$(l \simeq r)\,\sigma\rho, (s[l'] \not\simeq t)\,\sigma\rho \models (s[r] \not\simeq t)\,\sigma\rho,$$

for every substitution ρ that grounds $(l \simeq r)\,\sigma$, $(s[l'] \simeq t)\,\sigma$ and $(s[l'] \not\simeq t)\,\sigma$, respectively.

Proof. The ground equation $(l \simeq r)\,\sigma\rho$ induces a rewrite rule $l\sigma\rho \to r\sigma\rho$. Since σ is a unifier for l and l', we have $l\sigma = l'\sigma$ and also $l\sigma\rho = l'\sigma\rho$ for every substitution ρ. Hence the rewrite rule reduces $(s[l'] \simeq t)\,\sigma\rho$ to $(s[r] \simeq t)\,\sigma\rho$.

Therefore, if the two ground equations $(l \simeq r)\,\sigma\rho$ and $(s[l'] \simeq t)\,\sigma\rho$ are true, the ground equation $(s[r] \simeq t)\,\sigma\rho$ also holds.

We can give a parallel proof with $(s[r] \not\simeq t)\,\sigma\rho$ as the right premise. □

In order to define a notion of redundancy, we need the following ordering on ground literal closures, which is based on the ordering $\succ_{\mathrm{gr}}$ on ground terms.

Definition 4.2. Let $\succ_{\mathrm{l}}$ be an arbitrary total well-founded extension of $\succ_{\mathrm{gr}}$ from ground literals to ground literal closures such that if $L\theta \succ_{\mathrm{gr}} M\rho$ then $L\cdot\theta \succ_{\mathrm{l}} M\cdot\rho$.

Let $\mathcal{L}$ be a set of ground literal closures and $L \cdot \theta$ be a ground literal closure. By $\mathcal{L}_{L\cdot\theta\succ_{\mathrm{l}}}$ we denote the set of literal closures in $\mathcal{L}$ smaller than $L \cdot \theta$:

$$\mathcal{L}_{L\cdot\theta\succ_{\mathrm{l}}} = \{M \cdot \rho \in \mathcal{L} \mid L \cdot \theta \succ_{\mathrm{l}} M \cdot \rho\}\,.$$

Lemma 4.2. *In every unit superposition inference the literal closure in the conclusion is smaller in $\succ_l$ than at least one of the premises.*

Proof. Let $(l \simeq r) \cdot \theta_\mathrm{l}$ and $(s[l'] \simeq t) \cdot \theta_\mathrm{r}$ be the premises of a unit superposition inference. Since $l\theta_\mathrm{l} = l\sigma\rho \succ_\mathrm{gr} r\theta_\mathrm{l} = r\sigma\rho$ as well as $l\sigma = l'\sigma$, in the conclusion we have $s[l']\theta_\mathrm{r} = s[l']\sigma\rho \succ_\mathrm{gr} s[r]\sigma\rho$ and with monotonicity of $\succ_\mathrm{gr}$ it follows that $(s[l'] \simeq t) \cdot \theta_\mathrm{r} \succ_\mathrm{l} (s[r] \simeq t)\,\sigma \cdot \rho'$ with $\rho' = \rho|_{\mathrm{var}(\{s[r]\sigma, t\sigma\})}$.

With a parallel argument for the inference with the premises $(l \simeq r) \cdot \theta_\mathrm{l}$ and $(s[l'] \not\simeq t) \cdot \theta_\mathrm{r}$ we can conclude $(s[l'] \not\simeq t) \cdot \theta_\mathrm{r} \succ_\mathrm{l} (s[r] \not\simeq t)\,\sigma \cdot \rho|_{\mathrm{var}(\{s[r]\sigma, t\sigma\})}$.

The empty clause $\square$ is smaller than any term, therefore the conclusion of an equality resolution inference is smaller than its premise. $\square$

Lemma 4.3. *Let R be a ground rewrite system and let θ_l, θ_r, σ and ρ be the substitutions in a unit superposition inference. If $\theta_\mathrm{l} \cup \theta_\mathrm{r}$ is irreducible in R then ρ also is.*

Proof. Let $\theta_\mathrm{l} \cup \theta_\mathrm{r}$ be an irreducible substitution and assume ρ reducible, that is there is a variable x such that $x\rho = s$ and the term s is reduced by a rewrite rule $(s \to t) \in R$.

If there is a term $v[x] \in \mathrm{rng}\,(\sigma)$ then there is a variable x' such that $x'\sigma = v[x]$ and $x'\,(\theta_\mathrm{l} \cup \theta_\mathrm{r}) = x'\sigma\rho = v[x]\rho = v[s]$. Now we have $v[s] \in \mathrm{rng}\,(\theta_\mathrm{l} \cup \theta_\mathrm{r})$ reducible by $(s \to t) \in R$ which contradicts the assumption of irreducibility of $\theta_\mathrm{l} \cup \theta_\mathrm{r}$.

Otherwise, $x\sigma = x$ and $x\,(\theta_\mathrm{l} \cup \theta_\mathrm{r}) = x\sigma\rho = x\rho = s$. We have reducibility of $s \in \mathrm{rng}\,(\theta_\mathrm{l} \cup \theta_\mathrm{r})$ by $(s \to t) \in R$ contradicting the assumption of irreducibility of $\theta_\mathrm{l} \cup \theta_\mathrm{r}$.

Hence ρ must be irreducible and so also its restriction $\rho|_{\mathrm{var}(\{s[r]\sigma, t\sigma\})}$. $\square$

Let us define redundancy of literal closures based on the ordering $\succ_\mathrm{l}$.

Definition 4.3. Let $\mathcal{L}$ be a set of literal closures and $L \cdot \theta$ be a literal closure. We say that $L \cdot \theta$ is *US-redundant* in $\mathcal{L}$, if for every convergent ground rewrite system R oriented by $\succ_\mathrm{gr}$, where θ is irreducible with regard to R, the literal closure $L \cdot \theta$ follows from smaller irreducible closures and the rewrite system R:

$$R \cup \mathrm{irred}_R\,(\mathcal{L}_{L \cdot \theta \succ_\mathrm{l}}) \models L\theta.$$

Let $\mathcal{R}_\mathrm{US}\,(\mathcal{L})$ denote the set of all US-redundant literal closures in $\mathcal{L}$.

Informally, a literal closure $L \cdot \theta$ is redundant if it follows from smaller closures. However, we have to take into account the candidate model from the model generation proof we present later in this section. US-redundancy has to hold in

every rewrite system R built from smaller irreducible literal closures, where the rewriting system must not affect terms in the substitution θ.

Irreducibility of the substitution in a literal closure is an important concept in the model generation proof later and is therefore required in the definition of redundancy. It is necessary to accept some not obvious properties as in the following example.

Example 4.1 (Ganzinger and Korovin [2004])**.** Let $f(a) \succ_{\text{gr}} f(b) \succ_{\text{gr}} a \succ_{\text{gr}} b$ and

$$\mathcal{L} = \{(f(x) \simeq b) \cdot [a/x], (a \simeq b) \cdot [], (f(b) \not\simeq b) \cdot []\}$$

be a set of literal closures. Both the set of literals of the closures

$$\{f(x) \simeq b, a \simeq b, f(b) \not\simeq b\}$$

and the set of represented ground literals

$$\{f(a) \simeq b, a \simeq b, f(b) \not\simeq b\}$$

are inconsistent.

No inference is possible with the unit superposition calculus between literal closures in $\mathcal{L}$: the left-hand side term $f(x)$ of the first literal closure only unifies with the left-hand side term $f(b)$ of the last literal closure, but since the ground terms $f(x)\,[a/x]$ and $f(b)\,[]$ are different, no inference can be performed. Because $a \succ_{\text{gr}} b$, the equation in the second literal closure can only be applied as a left premise to literals with a subterm a, but the unit superposition inference rule does not allow the subterm to be a variable, which is why no inference is performed between the first two literal closures.

Therefore the inconsistency remains undetected. However, this does not harm completeness of the Inst-Gen-Eq method due to two observations.

We have $(f(x) \simeq b) \cdot [a/x] \succ_1 (a \simeq b) \cdot []$ and the rewrite system R induced by a candidate model in the model generation later contains the rule $a \to b$ before $(f(x) \simeq b) \cdot [a/x]$. Since the substitution $[a/x]$ is reducible by $a \to b$, the literal closure $(f(x) \simeq b) \cdot [a/x]$ does not contribute to the candidate model.

On the other hand, we always consider each literal closure of a literal. The model generation guarantees that if the inconsistency is not redundant, there is a literal closure $(f(x) \simeq b) \cdot [b/x]$, for instance, so that the unit superposition

calculus does find a contradiction.

Having presented inference rules and a notion of redundancy, we can now precisely define a saturation process. In contrast to Ganzinger and Korovin [2004] we consider the saturation process modulo a distinguished set of literal closures. Intuitively, there are literal closures that are ground instances in every model of the input clause set and that can thus be efficiently used for simplification by rewriting. From the point of view of the saturation process it suffices to assume a partition of the input set of literal closures into a pair of sets. Incorporating above intuition, in the next section we give a condition that has to be satisfied by the partition. We present concrete simplification inferences based on this saturation process in Chapter 6.

Definition 4.4. A *US-saturation process* is a sequence of pairs $\{\langle \mathcal{N}^i, \mathcal{D}^i \rangle\}_{i=1}^{\infty}$, where $\mathcal{N}^i$ is a set of literal closures and $\mathcal{D}^i$ is a set of positive literal closures. Each pair $\langle \mathcal{N}^{i+1}, \mathcal{D}^{i+1} \rangle$ with $i > 1$, called a *successor state*, is obtained from $\langle \mathcal{N}^i, \mathcal{D}^i \rangle$ by either

(i) adding to $\mathcal{N}^i$ the conclusion of a unit superposition inference with a left premise $(l \simeq r) \cdot \theta_{\mathrm{l}} \in \mathcal{N}^i \cup \mathcal{D}^i$ and a right premise $(s[l'] \simeq t) \cdot \theta_{\mathrm{r}} \in \mathcal{N}^i$ or $(s[l'] \not\simeq t) \cdot \theta_{\mathrm{r}} \in \mathcal{N}^i$, or the conclusion of an equality resolution inference with a premise $(l \not\simeq r) \cdot \theta$ in $\mathcal{N}^i$,

(ii) adding to $\mathcal{D}^i$ the conclusion of a unit superposition inference with both premises $(l \simeq r) \cdot \theta_{\mathrm{l}}$ and $(s[l] \simeq t) \cdot \theta_{\mathrm{r}}$ in $\mathcal{D}^i$,

(iii) removing a literal closure from $\mathcal{N}^i$ which is US-redundant in $\mathcal{N}^i \cup \mathcal{D}^i$ or

(iv) removing a literal closure from $\mathcal{D}^i$ which is US-redundant in $\mathcal{D}^i$.

In a later chapter we present simplification inferences such as demodulation with unit clauses. To justify these inferences, the set $\mathcal{D}$ will contain closures from unit clauses. However, for the rest of this chapter we can assume $\mathcal{D}$ to be empty.

Definition 4.5. Let us denote by $\mathcal{N}^\infty$ and $\mathcal{D}^\infty$ the sets of *persistent closures* in the US-saturation process $\{\langle \mathcal{N}^i, \mathcal{D}^i \rangle\}_{i=1}^{\infty}$, which are the lower limits of $\{\mathcal{N}^i\}_{i=1}^{\infty}$ and $\{\mathcal{D}^i\}_{i=1}^{\infty}$, respectively:

$$\mathcal{N}^\infty = \liminf_{i \to \infty} \mathcal{N}^i = \bigcup_{n=1}^{\infty} \left(\bigcap_{m=n}^{\infty} \mathcal{N}^m \right)$$

$$\mathcal{D}^\infty = \liminf_{i\to\infty} \mathcal{D}^i = \bigcup_{n=1}^{\infty} \left(\bigcap_{m=n}^{\infty} \mathcal{D}^m \right)$$

A literal closure in $\mathcal{N}$ and $\mathcal{D}$ is persistent if it is in the saturation process for every i greater than a fixed i_0. Further, a saturation process is fair if in the limit it reaches sets $\mathcal{N}^\infty$ and $\mathcal{D}^\infty$ which are closed under non-redundant inferences from their respective persistent literal closures. This is formalised in the next definition.

Definition 4.6. A US-saturation process is *fair* if

(i) for every unit superposition inference with one premise in $\mathcal{N}^\infty$ and the second premise in $\mathcal{N}^\infty \cup \mathcal{D}^\infty$, there is an i such that the conclusion is in $\mathcal{N}^i$ or US-redundant in $\mathcal{N}^i \cup \mathcal{D}^i$ and

(ii) for every unit superposition inference with both premises in $\mathcal{D}^\infty$ there is a j such that the conclusion is in $\mathcal{D}^j$ or US-redundant in $\mathcal{D}^j$.

In the following we assume that a saturation process follows a deterministic strategy and associate a fixed fair US-saturation process with each pair of sets of literal closures. We only consider the limit of the saturation process modulo redundancy.

Definition 4.7. Let $\langle \mathcal{N}, \mathcal{D} \rangle$ be a pair of a set of literal closures $\mathcal{N}$ and a disjoint set of positive literal closures $\mathcal{D}$. Let $\{\langle \mathcal{N}^i, \mathcal{D}^i \rangle\}_{i=1}^{\infty}$ be an arbitrary but fixed and fair US-saturation process with $\mathcal{N}^1 = \mathcal{N}$ and $\mathcal{D}^1 = \mathcal{D}$. We call the set

$$\langle \mathcal{N}, \mathcal{D} \rangle^{\text{sat}} = \left(\mathcal{N}^\infty \setminus \mathcal{R}_{\text{US}} \left(\mathcal{N}^\infty \cup \mathcal{D}^\infty \right) \right) \cup \left(\mathcal{D}^\infty \setminus \mathcal{R}_{\text{US}} \left(\mathcal{D}^\infty \right) \right)$$

the *US-saturation* of $\langle \mathcal{N}, \mathcal{D} \rangle$.

This section has introduced the unit superposition calculus on literal closures, a notion of redundancy and a saturation process. Returning to the proof road map in Figure 4.1 on page 44, we are now equipped with a fair US-saturation process (① in the figure) that provides us with the US-saturation $\langle \mathcal{N}, \mathcal{D} \rangle^{\text{sat}}$ to any pair of sets of literal closures $\langle \mathcal{N}, \mathcal{D} \rangle$.

4.2 Model Generation from Inst-Saturation

Step ② in Figure 4.1 on page 44 is the generation of a model for S from a US-saturated set of literal closures. To that end we now consider ground closures

which are instances of clauses in S. We first define a separate ordering $\succ_{\text{cl}}$ on ground closures which extends the ordering $\succ_{\text{l}}$ on ground literal closures.

Definition 4.8. Let $\succ_{\text{cl}}$ be an arbitrary total well-founded ordering on ground closures $C \cdot \theta$ and $D \cdot \rho$ such that $C \cdot \theta \succ_{\text{cl}} D \cdot \rho$ if

(i) $C\theta \succ_{\text{gr}} D\rho$ or

(ii) $C\theta = D\rho$ and $C\sigma = D$ for some proper substitution σ.

In addition to comparing two closures by the ground clauses they represent, the ordering $\succ_{\text{cl}}$ takes into account the clauses and makes the closure $C\sigma \cdot \theta$ smaller than $C \cdot \sigma\theta$, where the clause $C\sigma$ is a proper instance of C.

In order to define saturation on ground instances of clauses in S we need a new notion of redundancy, making use of the ordering $\succ_{\text{cl}}$ on ground closures.

Definition 4.9. Let S be a set of clauses. The ground closure $C \cdot \theta$ is *Inst-redundant* in S if there exist ground closures $C_1 \cdot \theta_1, \ldots, C_n \cdot \theta_n$ of clauses $C_1, \ldots, C_n$ in S such that $C \cdot \theta \succ_{\text{cl}} C_i \cdot \theta_i$ for each i and $C_1\theta_1, \ldots, C_n\theta_n \models C\theta$.

Let $\mathcal{S}$ be a set of closures, then by $\mathcal{R}_{\text{Inst}}(\mathcal{S})$ we denote the set of closures which are Inst-redundant in $\mathcal{S}$.

A clause C is *Inst-redundant* in S if each ground closure $C \cdot \theta$ is Inst-redundant in S.

We note that Inst-redundancy is as strong as standard redundancy in resolution as in Bachmair and Ganzinger [2001]. By their notion, a clause C is considered redundant with respect to a set of clauses S if there are clauses $C_1, \ldots, C_n$ in S such that $C \succ C_i$ for all i and $C_1, \ldots, C_n \models C$ for an ordering $\succ$ on clauses. If $C \succ C_i$ for two clauses, then for all their ground instances we have $C\theta \succ_{\text{gr}} C_i\theta_i$ and therefore by Definition 4.8 also $C \cdot \theta \succ_{\text{cl}} C_i \cdot \theta_i$ for all ground closures of C and C_i. Thus, all ground closures of C are Inst-redundant and finally C is Inst-redundant.

However, the more important property of Inst-redundancy is how it relates the closures of a clause C to the closures of an instance of C. If S contains a proper instance $C\sigma$ of a clause C, then all ground closures $C \cdot \sigma\theta$ are Inst-redundant. For each closure $C \cdot \sigma\theta$ there is a closure $C\sigma \cdot \theta$ which is smaller in $\succ_{\text{cl}}$ and trivially $C\sigma\theta \models C\sigma\theta$. A closure of a clause C representing a ground clause $C\theta$ is therefore redundant if there is an instance of C which has a closure

representing the same ground clause $C\theta$. The notion of Inst-redundancy is the key to completeness of Inst-Gen-Eq. By instantiating a clause we are making closures of the instantiated clause redundant.

Central to the US-saturation process are ground literal closures from instances of S. It suffices to consider a subset of all literal closures from all ground instances.

Definition 4.10. Let S be a set of clauses and $I_\perp$ be a model for its ground abstraction $S\perp$. A *selection function* sel assigns to each clause $C \in S$ a literal $L \in C$ such that $I_\perp \models L\perp$. We call such a selection function *based on* $I_\perp$.

Definition 4.11. Let sel be a selection function based on a model $I_\perp$ of the abstraction $S\perp$ of S. The set of *S-relevant literal closures* $\mathcal{L}(S)$ contains all ground literal closures $L \cdot \theta|_{\mathrm{var}(L)}$ such that

(i) $L \vee C \in S$

(ii) $(L \vee C) \cdot \theta$ is not Inst-redundant in S

(iii) $L = \mathrm{sel}(L \vee C)$.

A literal closure $L\cdot\theta$ is relevant in S if the literal L is selected in a clause $L \vee C$ and there is a closure of the clause $L \vee C$ which is not Inst-redundant and contains $L \cdot \theta$. We then apply the unit superposition calculus to the relevant literal closures until we arrive at saturation in the following sense.

Definition 4.12. Let S be a set of clauses and $\langle \mathcal{N}, \mathcal{D} \rangle$ be a partition of the S-relevant literal closures $\mathcal{L}(S)$ such that each closure $(l \simeq r) \cdot \theta \in \mathcal{D}$ follows from S, i.e. $S \models (l \simeq r)\,\theta$ holds. Let $\langle \mathcal{N}, \mathcal{D} \rangle^{\mathrm{sat}}$ be the US-saturation of $\langle \mathcal{N}, \mathcal{D} \rangle$. The set of clauses S is *Inst-saturated* with regard to a selection function sel and a partition $\langle \mathcal{N}, \mathcal{D} \rangle$ if $\langle \mathcal{N}, \mathcal{D} \rangle^{\mathrm{sat}}$ does not contain a contradiction $\Box$.

The set of relevant literal closures $\mathcal{L}(S)$ is partitioned into $\langle \mathcal{N}, \mathcal{D} \rangle$ such that every closure $(l \simeq r) \cdot \theta \in \mathcal{D}$ follows from S. This property enables efficient simplification inferences with literal closures from $\mathcal{D}$, described in Chapter 6. The partitioning ensures that these simplifications in the US-saturation process remain valid throughout the incremental Inst-saturation process and never have to be reversed.

A crucial theorem for completeness of Inst-Gen-Eq is the following.

Theorem 4.4. *If a set of clauses S is Inst-saturated with regard to a selection function* sel *and a partition $\langle \mathcal{N}, \mathcal{D} \rangle$ and $S\perp$ is satisfiable, then S is also satisfiable.*

We prove the theorem by explicitly constructing a model for S from an Inst-saturated set of clauses in the following way.

Definition 4.13 (Model generation)**.** Let the literal closures $L \cdot \theta = (l \simeq r) \cdot \theta$, respectively $L \cdot \theta = (l \not\simeq r) \cdot \theta$, be in the US-saturation $\langle \mathcal{N}, \mathcal{D} \rangle^{\text{sat}}$ of the partition $\langle \mathcal{N}, \mathcal{D} \rangle$ of the S-relevant literal closures $\mathcal{L}(S)$. Since we regard equations as unordered multisets, we may assume $l\theta \succ_{\text{gr}} r\theta$ without loss of generality. The US-saturation $\langle \mathcal{N}, \mathcal{D} \rangle^{\text{sat}}$ does not contain literal closures with $l\theta = r\theta$, since they are tautologies and as such US-redundant.

Suppose that sets of ground literals $\varepsilon_{M \cdot \rho}$ and sets of ground rewrite rules $\delta_{M \cdot \rho}$ have been defined for all literal closures $M \cdot \rho$ in $\langle \mathcal{N}, \mathcal{D} \rangle^{\text{sat}}$ smaller than $L \cdot \theta$ in $\succ_{\text{l}}$. Let $I_{L \cdot \theta \succ_{\text{l}}}$ be a partial interpretation, defined as the set union of all ground literals $\varepsilon_{M \cdot \rho}$ where $L \cdot \theta \succ_{\text{l}} M \cdot \rho$. Let $R_{L \cdot \theta \succ_{\text{l}}}$ be the rewrite system consisting of all rewrite rules $\delta_{M \cdot \rho}$ where $L \cdot \theta \succ_{\text{l}} M \cdot \rho$.

For a positive given $L \cdot \theta = (l \simeq r) \cdot \theta$ let $\varepsilon_{L \cdot \theta} = \{(l \simeq r)\,\theta\}$ and $\delta_{L \cdot \theta} = \{l \to r\}$ and for a negative given $L \cdot \theta = (l \not\simeq r) \cdot \theta$ let $\varepsilon_{L \cdot \theta} = \{(l \not\simeq r)\,\theta\}$ and $\delta_{L \cdot \theta} = \emptyset$ if

(i) $l\theta$ is irreducible in $R_{L \cdot \theta \succ_{\text{l}}}$ and

(ii) $L\theta$ is undefined in $I_{L \cdot \theta \succ_{\text{l}}}$, i.e. neither $I_{L \cdot \theta \succ_{\text{l}}} \models L\theta$ nor $I_{L \cdot \theta \succ_{\text{l}}} \models \overline{L}\theta$.

If (i) and (ii) are satisfied, we say the literal closure $L \cdot \theta$ is *productive*. Else, let $\varepsilon_{L \cdot \theta} = \emptyset$ and $\delta_{L \cdot \theta} = \emptyset$.

Let $I_S = \bigcup_{L \cdot \theta \in \langle \mathcal{N}, \mathcal{D} \rangle^{\text{sat}}} \varepsilon_{L \cdot \theta}$ be an interpretation and $R_S = \bigcup_{L \cdot \theta \in \langle \mathcal{N}, \mathcal{D} \rangle^{\text{sat}}} \delta_{L \cdot \theta}$ be a rewrite system.

The ordering $\succ_{\text{l}}$ is well-founded, hence induction over $L \cdot \theta$ with $\succ_{\text{l}}$ is well-founded. We show that the above model generation is sound, yielding a consistent interpretation and a convergent rewrite system.

Lemma 4.5. *The interpretation I_S is consistent and the rewrite system R_S is convergent.*

Proof. Due to condition (i) a literal closure $L \cdot \theta$ is only productive if the interpretation $I_{L \cdot \theta \succ_{\text{l}}}$ does not entail its complement. By induction over $\langle \mathcal{N}, \mathcal{D} \rangle^{\text{sat}}$ with $\succ_{\text{l}}$ it follows that I_S is consistent.

Since for each rule $(l \to r)\,\theta$ in R_S we have $l\theta \succ_{\text{gr}} r\theta$ and $\succ_{\text{gr}}$ is well-founded, the rewrite system R_S is terminating.

Further, we can show that R_S is left-reduced, that is there are no two different rules $(l \to r)\,\theta$ and $(s \to t)\,\rho$ where $s\rho$ is a subterm of $l\theta$. By condition (ii), if a literal closure $(l \simeq r) \cdot \theta$ produces a rule $(l \to r)\,\theta$, then $l\theta$ is irreducible in $R_{(l \simeq r) \cdot \theta \succ_1}$, which contains the rules produced by smaller literal closures in $\succ_1$. Now assume that there is a literal closure $(s \simeq t) \cdot \rho \succ_{\text{gr}} (l \simeq r) \cdot \theta$ producing the rule $(s \to t)\,\rho$. We have $(s \simeq t)\,\rho \succ_{\text{gr}} (l \simeq r)\,\theta$. Since $\succ_{\text{gr}}$ is a simplification order and as such obeys the subterm property, we have $s\rho \succ_{\text{gr}} l\theta$ and thus $s\rho$ cannot be a subterm of $l\theta$.

We now know R_S is left-reduced and terminating. By standard theorems in rewriting (see Baader and Nipkow [1998]), the system R_S is confluent and hence convergent, since it is terminating and confluent. □

Let us now show that the interpretation I_S as constructed from the US-saturation $\langle \mathcal{N}, \mathcal{D} \rangle^{\text{sat}}$ does indeed satisfy S: every total and consistent extension of I_S is a model for S. We proceed by contradiction and assume there are ground closures false in some interpretation I and let $C \cdot \theta$ be the minimal such closure with regard to $\succ_{\text{cl}}$. We call it the *minimal Inst-counterexample* to I.

We lead the assumption of the existence of a minimal Inst-counterexample to a contradiction by first showing that there is an S-relevant literal closure $L \cdot \theta'$ from $C \cdot \theta$, which is false in I. It follows that there is a, possibly different, not US-redundant literal closure $M \cdot \rho$ in the US-saturation $\langle \mathcal{N}, \mathcal{D} \rangle^{\text{sat}}$, which is also false in I. Since $M \cdot \rho$ is false, it is not productive, but we show that it satisfies (i) and (ii) in Definition 4.13, which is a contradiction. Hence we cannot have a minimal Inst-counterexample $C \cdot \theta$ and I is a model for all ground instances of S.

Lemma 4.6. *The minimal Inst-counterexample $C \cdot \theta$ is not Inst-redundant in S.*

Proof. If the closure $C \cdot \theta$ were Inst-redundant in S, we would have ground closures $C_1 \cdot \theta_1, \ldots, C_n \cdot \theta_n$, which are instances of clauses from S, such that $C \cdot \theta \succ_{\text{cl}} C_i \cdot \theta_i$ for each i and $C_1\theta_1 \ldots, C_n\theta_n \models C\theta$. However, since $C \cdot \theta$ is false in I, for at least one i the closure $C_i \cdot \theta_i$ would have to be false in I and hence $C \cdot \theta$ would not be the minimal Inst-counterexample. □

Lemma 4.7. *There is an S-relevant ground literal closure $L \cdot \theta'$ with $L \in C$, which is false in I.*

Proof. Since $C \cdot \theta$ is not Inst-redundant, its selected literal closure $L \cdot \theta' = \text{sel}\,(C) \cdot \theta|_{\text{var}(L)}$ is S-relevant. Moreover, because the closure $C \cdot \theta$ is false in I, each of its literal closures, including $L \cdot \theta'$, is. □

Lemma 4.8. *The substitution θ from $C \cdot \theta$ is irreducible in R_S.*

Proof. Assume that there is a variable $x \in \operatorname{dom}(\theta)$ such that $x\theta = x\theta[u]_p$ is reducible at position p by a rewrite rule $u \to v \in R_S$. We can define a substitution θ' by changing θ at x so that $x\theta' = x\theta[v]_p$. Now we have a closure $C \cdot \theta \succ_{\mathrm{cl}} C \cdot \theta'$, which is false in I and thus contradicts the minimality of the Inst-counterexample $C \cdot \theta$. □

Lemma 4.9. *There is a literal closure $M \cdot \rho$ in $\langle \mathcal{N}, \mathcal{D} \rangle^{sat}$, which is false in I, not US-redundant and ρ is irreducible in R_S.*

Proof. By Definition 4.7 on page 51 we have

$$\langle \mathcal{N}, \mathcal{D} \rangle^{\mathrm{sat}} = \big(\mathcal{N}^\infty \setminus \mathcal{R}_{\mathrm{US}}(\mathcal{N}^\infty \cup \mathcal{D}^\infty)\big) \cup \big(\mathcal{D}^\infty \setminus \mathcal{R}_{\mathrm{US}}(\mathcal{D}^\infty)\big).$$

Due to Definition 4.12 we have $S \models (l \simeq r)\,\theta$ for every $(l \simeq r) \cdot \theta \in \mathcal{D}$ and, since unit superposition inferences are sound, also $S \models (s \simeq t)\,\rho$ for the literal closures $(s \simeq t) \cdot \rho \in \mathcal{D}^\infty$. Every such $(s \simeq t) \cdot \rho$ is either productive or reducible in $R_{(s \simeq t) \cdot \rho \succ_1}$ to a productive literal closure. Hence, we have $I_S \models (s \simeq t) \cdot \rho$ for every $(s \simeq t) \cdot \rho \in \mathcal{D}^\infty$.

The S-relevant ground literal closure $L \cdot \theta'$ from Lemma 4.7 is false in I and thus not in $\mathcal{D}^\infty \setminus \mathcal{R}_{\mathrm{US}}(\mathcal{D}^\infty)$. If $L \cdot \theta'$ is in $\mathcal{N}^\infty \setminus \mathcal{R}_{\mathrm{US}}(\mathcal{N}^\infty \cup \mathcal{D}^\infty)$, then let $M \cdot \rho = L \cdot \theta'$. Since then $L \cdot \theta' \notin \mathcal{R}_{\mathrm{US}}(\mathcal{N}^\infty \cup \mathcal{D}^\infty)$, it is not US-redundant in $\langle \mathcal{N}, \mathcal{D} \rangle^{\mathrm{sat}}$. As of Lemma 4.8 the substitution θ and therefore also $\theta' = \theta|_{\mathrm{var}(L)}$ is irreducible in R_S.

If $L \cdot \theta' \notin \langle \mathcal{N}, \mathcal{D} \rangle^{\mathrm{sat}}$, then it is US-redundant in $\langle \mathcal{N}, \mathcal{D} \rangle^{\mathrm{sat}}$ and θ' is irreducible in R_S. By Definition 4.3 on page 48 we have[1]

$$R_S \cup \operatorname{irred}_{R_S}\big(\{M \cdot \rho \in \langle \mathcal{N}, \mathcal{D} \rangle^{\mathrm{sat}} \mid L \cdot \theta' \succ_1 M \cdot \rho\}\big) \models L\theta.$$

There is an $M \cdot \rho \in \operatorname{irred}_{R_S}\big(\langle \mathcal{N}, \mathcal{D} \rangle^{\mathrm{sat}}\big)$, which is false in I since $L \cdot \theta'$ is false in I. Because $M \cdot \rho \in \langle \mathcal{N}, \mathcal{D} \rangle^{\mathrm{sat}}$, it is not US-redundant in $\langle \mathcal{N}, \mathcal{D} \rangle^{\mathrm{sat}}$. Further, the substitution ρ is irreducible in R_S. □

Let us now consider the minimal wrt. $\succ_1$ literal closure $M \cdot \rho = (s \simeq t) \cdot \rho$, respectively $M \cdot \rho = (s \not\simeq t) \cdot \rho$, in $\operatorname{irred}_{R_S}\big(\{M \cdot \rho \in \langle \mathcal{N}, \mathcal{D} \rangle^{\mathrm{sat}} \mid L \cdot \theta' \succ_1 M \cdot \rho\}\big)$, which is false in I and call it again the *minimal US-counterexample.*

[1] For readability we choose not to use the abbreviated notation $\mathcal{L}_{L \cdot \theta \succ_1}$ as in the definition but the longer form $\{L' \cdot \theta' \in \mathcal{L} \mid L \cdot \theta \succ_1 L' \cdot \theta'\}$ instead.

Lemma 4.10. *The minimal US-counterexample $M \cdot \rho$ is not productive.*

Proof. Assume that $s\rho = r\rho$, then $M \cdot \rho = (s \simeq r) \cdot \rho$ is a tautology and not false in any interpretation. Further, if $M \cdot \rho = (s \not\simeq r) \cdot \rho$, then an equality resolution inference is applicable and the US-saturation contains a contradiction. Therefore we have $s\rho \succ_{\text{gr}} r\rho$ without loss of generality, since an equation is a multiset.

The literal closure $M \cdot \rho$ is false in I, therefore it cannot follow from the interpretation $\bigcup_{L\cdot\theta \in \langle \mathcal{N}, \mathcal{D} \rangle^{\text{sat}}} I_{L\cdot\theta \succ_{\text{l}}}$ and is thus not productive. □

Lemma 4.11. *The term $s\rho$ is irreducible in R_S.*

Proof. We note that the substitution ρ is irreducible in R_S as we have seen in Lemma 4.9. Assume that $s\rho$ is reducible in R_S, then there is a position p in $s\rho = s[u']_p\rho$ that is reducible by $(u \to v)\,\tau \in R_S$, which is produced by the closure $(u \simeq v) \cdot \tau \in \langle \mathcal{N}, \mathcal{D} \rangle^{\text{sat}}$. We can apply a unit superposition inference from the premises $(u \simeq v) \cdot \tau$ and $M \cdot \rho = (s[u'] \simeq t) \cdot \rho$ (or $M \cdot \rho = (s[u'] \not\simeq t) \cdot \rho$, respectively), since the side conditions are satisfied as follows:

(i) Since $u'\rho = u\tau$, there is a $\sigma = \text{mgu}\,(u, u')$,

(ii) u' is not a variable,

(iii) $u\tau \succ_{\text{gr}} v\tau$,

(iv) $s[u']\rho \succ_{\text{gr}} t$ and

(v) $u\tau = u'\rho$

We obtain the conclusion $K \cdot \mu' = (s[v] \simeq t)\,\sigma \cdot \mu'$ (or $K \cdot \mu' = (s[v] \not\simeq t)\,\sigma \cdot \mu'$, respectively), with $\mu = (\tau \cup \rho)$ and $\mu' = \mu|_{\text{var}(\{s[v]\sigma, t\sigma\})}$. The conclusion is false in I as the right premise $M \cdot \rho$ is, whereas the left premise is true in I.

In order to show that $K \cdot \mu'$ is not US-redundant in $\langle \mathcal{N}, \mathcal{D} \rangle^{\text{sat}}$, we assume the opposite. Due to Lemma 4.3 on page 48, the substitution in the conclusion of a unit superposition inference from closures with irreducible substitutions is also irreducible. Thus μ' is irreducible and we can apply Definition 4.3 on page 48, by which $K \cdot \mu'$ follows from smaller irreducible closures:

$$R_S \cup \text{irred}_{R_S}\left(\left\{P \cdot \nu \in \langle \mathcal{N}, \mathcal{D} \rangle^{\text{sat}} \mid K \cdot \mu' \succ_{\text{l}} P \cdot \nu\right\}\right) \models K \cdot \mu'.$$

We thus have a $P \cdot \nu \in \langle \mathcal{N}, \mathcal{D} \rangle^{\text{sat}}$, which is false in I, and $M \cdot \rho \succ_1 K \cdot \mu' \succ_1 P \cdot \nu$, which contradicts the minimality of $M \cdot \rho$ and our assumption of US-redundancy in $\langle \mathcal{N}, \mathcal{D} \rangle^{\text{sat}}$ of $K \cdot \mu$.

On the other hand, the literal closure $K \cdot \mu'$ is the conclusion of a unit superposition inference from premises in $\langle \mathcal{N}, \mathcal{D} \rangle^{\text{sat}}$ and if $K \cdot \mu'$ is not US-redundant, it is in $\langle \mathcal{N}, \mathcal{D} \rangle^{\text{sat}}$. However, this also contradicts the minimality of the US-counterexample $M \cdot \rho$. We conclude the irreducibility of $s\rho$ in R_S. □

Lemma 4.12. *There is no minimal US-counterexample $M \cdot \rho$.*

Proof. By Lemma 4.11 we have that $s\rho$ is irreducible in R_S. As we also know that $M \cdot \rho$ is not productive (Lemma 4.10) and $M \cdot \rho \in \langle \mathcal{N}, \mathcal{D} \rangle^{\text{sat}}$, condition (ii) in Definition 4.13 must not be satisfied and it must be the case that $I_{M \cdot \rho \succ_1} \models \overline{M} \cdot \rho$.

Let us consider the two possible cases:

Case $M \cdot \rho = (s \simeq t) \cdot \rho$. We have $I_{M \cdot \rho \succ_1} \models (s \not\simeq t)\,\rho$. Since $s\rho$ is irreducible in R_S, there must be a $(u \not\simeq v) \cdot \tau$ producing some $(s \simeq t')\,\tau$ in $I_{M \cdot \rho \succ_1}$ such that t' is reducible in $R_{M \cdot \rho \succ_1}$ to t. However, this is impossible as

$$\overline{M}\rho = (u \not\simeq v)\,\tau \succ_{\text{gr}} (s \simeq t')\,\rho \succ_{\text{gr}} (s \simeq tr)\,\rho = M\rho$$

and hence $(u \not\simeq v) \cdot \tau \succ_1 (s \simeq t) \cdot \rho$.

Case $M \cdot \rho = (s \not\simeq t) \cdot \rho$. We have $I_{M \cdot \rho \succ_1} \models (s \simeq t) \cdot \rho$. Since $s\rho$ is irreducible in R_S, we have $s\rho = t\rho$. However, then equality resolution is applicable to $M \cdot \rho = (s \not\simeq t) \cdot \rho$ and $\langle \mathcal{N}, \mathcal{D} \rangle^{\text{sat}}$ contains a contradiction. Again, we have a contradiction. □

Backtracking over our assumptions shows that there cannot be a minimal US-counterexample $M \cdot \rho \in \langle \mathcal{N}, \mathcal{D} \rangle^{\text{sat}}$ and thus no minimal Inst-counterexample $C \cdot \theta$, which is a ground instance in S. Therefore, the interpretation I is a model for all ground instances of S.

Hence, we have proved Theorem 4.4, which is part ② in Figure 4.1 on page 44: if S is Inst-saturated, that is the US-saturation of its relevant literal closures does not contain the contradiction, we can construct a model I of S as shown in Definition 4.13.

4.3 An Effective Inst-Saturation Process

In the previous section we have considered Inst-saturation only statically by assuming an Inst-saturated set of clauses, by Definition 4.12 on page 53 a set of clauses for which the US-saturation of the relevant literal closures does not contain a contradiction.

We now turn to the process of reaching Inst-saturation in this sense, which is illustrated as a circle in the middle of Figure 4.1 on page 44. So far we have defined the notion of relevant literal closures (Definition 4.11 on page 53) and have shown how to obtain the US-saturation of a set of literal closures (Definition 4.7 on page 51). In order to close the circle, we have to remove contradictions from the US-saturation. This requires eliminating certain literal closures from the set of relevant literal closures, which we achieve by making closures redundant (part ③ in Figure 4.1 on page 44). An Inst-saturation process is fair if it eventually removes all contradictions in the US-saturation.

In this section we prove the significance of a fair Inst-saturation process: we can always find an unsatisfiable ground abstraction if the input clause set is unsatisfiable. Moreover, all ground abstractions from a satisfiable set of clauses remain satisfiable.

Let us now formally define the Inst-saturation process.

Definition 4.14. An *Inst-saturation process* is a possibly infinite sequence of tuples $\left\{\left\langle S^i, I^i_\bot, \mathrm{sel}^i\right\rangle\right\}_{i=1}^{\infty}$, where S^i is a set of clauses. If at any i the ground abstraction $S^i\bot$ is unsatisfiable, the process stops with the result "unsatisfiable". Otherwise $I^i_\bot$ is a model of $S^i\bot$ and sel^i a selection function based on $I^i_\bot$.

Given a triple $\left\langle S^i, I^i_\bot, \mathrm{sel}^i\right\rangle$, which we call a state in the saturation process, a *successor state* $\left\langle S^{i+1}, I^{i+1}_\bot, \mathrm{sel}^{i+1}\right\rangle$ is obtained by modifying the set of clauses S^i such that either

(i) $S^{i+1} = S^i \cup N$ where N is a set of clauses such that $S^i \models N$ or

(ii) $S^{i+1} = S^i \setminus \{C\}$ where C is Inst-redundant in S^i.

An Inst-saturation process adds to the clause set S^i in the current state consequences of S^i or removes Inst-redundant clauses, by Definition 4.9 on page 52 clauses which follow from smaller clauses in S^i. The satisfiability of the clause set S^1 is preserved throughout the saturation process.

Lemma 4.13. *The set of clauses S^{i+1} in a successor state is satisfiable if and only if S^i is satisfiable.*

Proof. Neither adding consequences nor removing redundant clauses changes the set of entailed clauses. In particular the empty clause is never redundant and only follows from unsatisfiable clause sets, hence the set S^{i+1} is satisfiable if and only if S^i is. □

Satisfiability is not always preserved for the ground abstraction between the current state $S^i\bot$ and a successor state $S^{i+1}\bot$. In the introductory Example 3.3 on page 37 a satisfiable ground abstraction becomes unsatisfiable after adding instances in a successor state. However, for a satisfiable clause set we can show that no unsatisfiable ground abstraction can be obtained in the saturation process.

Lemma 4.14. *If the clause set S^i is satisfiable, then the ground abstractions of all successor states $S^j\bot$ for $j \geq i$ are satisfiable.*

Proof. From the above Lemma 4.13 we conclude that all clause sets S^j with $j \geq i$ are satisfiable. By Herbrand's Theorem a set of clauses is satisfiable if and only if all sets of ground instances of the clause set are satisfiable. Consequently, in particular the sets of ground instances $S^j\bot$ with $j \geq i$ are satisfiable. □

In the following we are only interested in the clauses in the limit of the saturation process, where we do not consider clauses that only intermittently appear.

Definition 4.15. For an Inst-saturation process $\left\{\left\langle S^i, I^i_\bot, \mathrm{sel}^i\right\rangle\right\}_{i=1}^{\infty}$ let us denote by S^∞ the set of *persistent clauses* which is the lower limit of the sequence $\{S^i\}_{i=1}^{\infty}$.

$$S^\infty = \liminf_{i\to\infty} \left\{S^i\right\}_{i=1}^{\infty} = \bigcup_{n=1}^{\infty}\left(\bigcap_{m=n}^{\infty} S^m\right)$$

In other words, a persistent clause is in every S^i from a certain $i > i_0$ onwards.

It is neither feasible in practice nor necessary for the Inst-saturation process to saturate the clause set with all consequences and to remove all redundant clauses. For refutational completeness in the Inst-Gen-Eq method it suffices if the saturation process eventually produces a clause set with an unsatisfiable ground abstraction if the input clause set is unsatisfiable. In order to guide the process towards an unsatisfiable ground abstraction we use a US-saturation process on selected literal closures and define the notion of conflicts on clauses considering only the persistent clauses.

Definition 4.16. Let $\{\langle S^i, I^i_\perp, \mathrm{sel}^i\rangle\}_{i=1}^{\infty}$ be an Inst-saturation process with the set S^∞ of persistent clauses and $K = \{(L_1 \vee C_1)\cdot\theta_1, \ldots, (L_n \vee C_n)\cdot\theta_n\}$ be a set of n ground instances from S^∞. Let $\mathcal{L} = \{L_1\cdot\theta'_1, \ldots, L_n\cdot\theta'_n\}$ be a set of literal closures where $\theta'_i = \theta_i|_{\mathrm{var}(L_i)}$ and each L_i is selected infinitely often: for all $1 \leq j \leq n$ and infinitely many i we have $\mathrm{sel}^i(L_j \vee C_j) = L_j$.

The pair $\langle K, \mathcal{L}\rangle$ is called a *persistent conflict* if there is a contradiction $\square$ in the US-saturation $\langle \mathcal{N}, \mathcal{D}\rangle^{\mathrm{sat}}$ (Definition 4.7 on page 51) of a partition of $\mathcal{L}$ into $\langle \mathcal{N}, \mathcal{D}\rangle$ such that $S^\infty \models (l \simeq r)\,\theta$ holds for each closure $(l \simeq r)\cdot\theta \in \mathcal{D}$.

We remark that the definition of a persistent conflict does not refer to Inst-redundancy. A conflict $\langle K, \mathcal{L}\rangle$ can contain Inst-redundant closures in K and thus not all literal closures in $\mathcal{L}$ are necessarily relevant. However, it is the intention of a fair Inst-saturation process to make persistent conflicts irrelevant by making closures in K redundant. A persistent conflict indicates where the Inst-saturation process needs to refine the ground abstraction.

Definition 4.17. An Inst-saturation process is *fair* if for every persistent conflict $\langle K, \mathcal{L}\rangle$ at least one of the closures in K is Inst-redundant in some state i of the saturation process.

Let us now take a closer look at the models of the ground abstraction in the Inst-saturation process. In each state, if the ground abstraction $S^i\perp$ is satisfiable, there can be several models $I^i_\perp$, from which we non-deterministically choose one. In practice, this choice is delegated to a black box ground solver.

Adding to the clause set S^i in order to refine the ground abstraction on a persistent conflict triggers changes in the model of the ground abstraction, which may contradict assignments in the model of the preceding state. We can have $I^i_\perp \models L\perp$ and $I^{i+1}_\perp \models \overline{L}\perp$ and therefore in particular $I^i_\perp \cup I^j_\perp$ may be inconsistent for $i \neq j$.

When the model of the ground abstraction $I^i_\perp$ changes in a successor state, this affects the selection function sel^i, which is based on $I^i_\perp$. If there is more than one choice for a selection function, we again non-deterministically choose one. In an implementation we employ heuristics for literal selection.

We are now ready to prove the main theorem about the effectiveness of a fair Inst-saturation process: from a satisfiable set of clauses a fair Inst-saturation process reaches an Inst-saturated set of clauses in the limit. Since models of the ground abstraction and thus selection functions change in the course of the saturation process, this theorem is not obvious.

For this theorem in particular we have be able to deal with the situation of a divergent selection function in the limit of the saturation process. We do not have the notion of a persistent selection and since the model of the ground abstraction is calculated by the black box ground solver, we cannot enforce a persistent model. It is possible that the model and thus the selection function alternates between two states of the saturation process, such that the selection functions in the states of the saturation process do not converge towards one selection function in the limit. To cope with this situation we do not require conditions on the selection function to hold for all states $i > i_0$ from a certain state i_0 on, but weaken it and demand that they only hold for infinitely many states. Divergent selection functions that change in every state are covered by the latter condition, but not by the former.

Theorem 4.15. *Let S^∞ be a set of persistent clauses of a fair Inst-saturation process $\left\{\left\langle S^i, I^i_\bot, \mathrm{sel}^i\right\rangle\right\}_{i=1}^\infty$ and let $S^\infty\bot$ be satisfiable. Then there is a model $I_\bot$ of $S^\infty\bot$ and a selection function* sel *based on $I_\bot$ such that S^∞ is Inst-saturated with regard to* sel *and any partition $\langle \mathcal{N}, \mathcal{D}\rangle$ of the S-relevant literal closures.*

Proof. We arrange the clauses in S^∞ in an arbitrary order as $\{C_i\}_{i=1}^\infty$ and by induction over the clauses thus arranged incrementally construct a model J^n of the ground abstraction of the first n clauses, i.e. the set $\{C_i\bot\}_{i=1}^n$, and a selection function sel_{J^n} based on J^n such that for each n the following invariants hold.

(i) J^n is a consistent interpretation and a model for $\{C_i\}_{i=1}^n$. The function sel_{J^n} is a selection function based on J^n for the clauses in $\{C_i\}_{i=1}^n$.

(ii) $J^n \subseteq J^{n+1}$ and $\mathrm{sel}_{J^{n+1}}(C) = \mathrm{sel}_{J^n}(C)$ for all clauses C in $\{C_i\}_{i=1}^n$.

(iii) For infinitely many k we have $J^n \subseteq I^k_\bot$ and $\mathrm{sel}^k(C_l) = \mathrm{sel}_{J^n}(C_l)$ for all $1 \leq l \leq n$.

For $n = 1$ we have the singleton set $\{C_1\}$ for which we have to give a model J^1 and a selection function $\mathrm{sel}_{J^1}(C_1)$ that satisfy the invariants (i)-(iii) above. Since $C_1 \in S^\infty$ and consists of a finite number of literals, there is a literal $L \in C_1$ such that $I^k_\bot \models L\bot$ and $\mathrm{sel}^k(C_1) = L$ for infinitely many k. We let $J^1 = \{L\bot\}$, leaving all other ground literals undefined, and $\mathrm{sel}_{J^1}(C_1) = L$, thus satisfying invariant (iii). Since J^1 is consistent and sel_{J^1} is a selection function for C_1 based on J^1, we also satisfy invariant (i). Assuming $J^0 = \emptyset$ and $\mathrm{dom}\left(\mathrm{sel}^0_J\right) = \emptyset$, invariant (ii) holds.

We can now assume that there is a model J^n and a selection function sel_{J^n} that satisfy (i)-(iii) for some n. Let us consider the persistent clause $C_{n+1} \in S^\infty$. There is an m_0 such that for all $m > m_0$ we have $C_{n+1} \in S^m$ and hence $I^m_\perp \models C_{n+1}\perp$. Due to invariant (iii) there are infinitely many p with $J^n \subseteq I^p_\perp$ and $\mathrm{sel}^p(C_l) = \mathrm{sel}_{J^n}(C_l)$ for all $1 \leq l \leq n$.

There are infinitely many q where $\mathrm{sel}^q(C_{n+1}) = L$ for some literal $L \in C_{n+1}$, consequently $I^q_\perp \models L\perp$, and also $J^n \subseteq I^q_\perp$. Therefore $J^{n+1} = J^n \cup \{L\perp\}$ is consistent and a model for $\{C_i\}^{n+1}_{i=1}$. Further, we can define $\mathrm{sel}_{J^{n+1}}(C_{n+1}) = L$ and $\mathrm{sel}_{J^{n+1}}(C_j) = \mathrm{sel}_{J^n}(C_j)$ for all $1 \leq l \leq n$, obtaining a selection function for the clauses $\{C_i\}^{n+1}_{i=1}$ based on J^{n+1}. By construction J^{n+1} and $\mathrm{sel}_{J^{n+1}}$ satisfy invariants (i)-(ii).

By induction the invariants hold for all n and we let $I_\perp = \bigcup^\infty_{i=1} J^i$ as well as $\mathrm{sel}(C_i) = \mathrm{sel}_{J^i}(C_i)$. Since all finite subsets of all J^i are consistent, we conclude consistency of $I_\perp$ from compactness of first-order logic. Further, sel is a selection function based on $I_\perp$.

Assume S^∞ not Inst-saturated with regard to sel and some partition $\langle \mathcal{N}, \mathcal{D}\rangle$ of the S-relevant literal closures with $S^\infty \models (l \simeq r)\,\theta$ for each $(l \simeq r) \cdot \theta \in \mathcal{D}$. There is a finite subset $\mathcal{L}$ and a partition $\langle \mathcal{N}', \mathcal{D}'\rangle$, such that $\langle \mathcal{N}', \mathcal{D}'\rangle^{\mathrm{sat}}$ contains a contradiction. Let $K = \{(L_1 \vee C_1) \cdot \theta_1, \ldots, (L_n \vee C_n) \cdot \theta_n\}$ be a set of closures of clauses from S^∞ such that for each L_i there is a substitution θ'_i with $x\theta = x\theta'$ for each $x \in \mathrm{var}(L_i)$ and $L_i \cdot \theta'_i \in \mathcal{L}(S^\infty)$.

From the construction of $I_\perp$ and invariant (iii) it follows that there are infinitely many i such that $\mathrm{sel}^i(L_l \vee C_l) = L_l$ for $1 \leq l \leq n$. Hence $\langle K, \mathcal{L}\rangle$ is a persistent conflict and as the Inst-saturation process is fair, at least one of the closures in K is Inst-redundant in $\mathcal{L}(S^\infty)$.

However, this contradicts the assumption of Inst-saturation of S^∞, whereby all closures in $\mathcal{L}$ are S^∞-relevant and cannot be produced by Inst-redundant closures. □

If the ground abstraction of the persistent clauses, that is, the limit of a saturation process, is satisfiable, then there is a selection function with respect to which the set of persistent clauses is Inst-saturated for any partition of the S-relevant literal closures. The set of persistent clauses is not necessarily Inst-saturated with respect to every selection function based on a model of the ground abstraction, but the theorem guarantees the existence of at least one such selection function.

Refutational completeness of a fair Inst-saturation process is now a consequence of the previous theorem.

Corollary 4.16. *If S^1 is unsatisfiable then in every fair Inst-saturation process starting at $\langle S^1, I^1_\perp, \mathrm{sel}^1\rangle$ there is a state i_0 such that for all states $i > i_0$ the ground abstraction $S^i\perp$ is unsatisfiable.*

Proof. Assume otherwise, then the ground abstractions $S^i\perp$ are satisfiable in some state $i > i_0$ of the saturation process $\{\langle S^i, I^i_\perp, \mathrm{sel}^i\rangle\}^\infty_{i=1}$. Therefore the limit of the saturation process $S^\infty\perp$ is satisfiable and by Theorem 4.15 there is a model $I_\perp$ for $S^\infty\perp$ and a selection function sel such that S^∞ is Inst-saturated with regard to sel and any partition of the S-relevant literal closures. But then by Theorem 4.4 on page 53 the Inst-saturated S^∞ with its satisfiable ground abstraction $S^\infty\perp$ is satisfiable. By Lemma 4.13 S^1 is also satisfiable, which is a contradiction. Hence our assumption is false and $S^i\perp$ cannot be satisfiable in every state i throughout the saturation process. □

The saturation process does not in general terminate for a satisfiable clause set S^i, because of the undecidability of the satisfiability problem in first-order logic. However, we can state the following theorem that allows us to claim satisfiability after a finite number of steps in some cases.

Corollary 4.17. *If there is a state of the saturation process i where S^i is Inst-saturated and $S^i\perp$ is satisfiable, then S^1 is satisfiable.*

Proof. By Theorem 4.4 on page 53 an Inst-saturated set S^i is satisfiable because of its satisfiable ground abstraction $S^i\perp$. Due to Lemma 4.13 S^1 is also satisfiable. □

The results about effectiveness of a fair Inst-saturation process are summarised in the following theorem.

Theorem 4.18. *Let $\{\langle S^i, I^i_\perp, \mathrm{sel}^i\rangle\}^\infty_{i=1}$ be a fair Inst-saturation process. Then either*

(i) $S^i\perp$ is unsatisfiable for at least one i and therefore S^1 is unsatisfiable or

(ii) $S^i\perp$ is satisfiable for all i and therefore S^1 is satisfiable.

Moreover, if for some i in the Inst-saturation process S^i is Inst-saturated, then we can conclude at this step the satisfiability of S^1.

In this section we have defined the Inst-saturation process and a notion of fairness, which effectively guides the saturation process along persistent conflicts. We have shown that a fair Inst-saturation process eventually arrives at an unsatisfiable ground abstraction from an unsatisfiable input clause set. An Inst-saturation process produces only satisfiable ground abstractions from a satisfiable input clause set. If in an Inst-saturation process the clause set becomes Inst-saturated, the process has proved that the input clause set is satisfiable.

4.4 Instance Generation from Conflicts

Fairness of an Inst-saturation process as in Definition 4.17 on page 61 requires closures in persistent conflicts to be made redundant. Remember that by Definition 4.9 on page 52 a closure $C \cdot \sigma\theta$ of a clause C is Inst-redundant in a set of clauses S if the set contains the proper clause instance $C\sigma$. In this section we show that it suffices to consider unit superposition inferences leading to a contradiction from literal closures in a persistent conflict: from every such unit superposition proof we can extract some proper instantiating substitution for some clause that makes at least one closure in the conflict redundant. Adding clause instances, generated from proofs of contradictions from literal closures in persistent conflicts, is thus a fair Inst-saturation process, the final step ④ in Figure 4.1 on page 44.

The US-saturation of the selected literals of the closures in a persistent conflict contains a contradiction, hence there is a finite proof of a contradiction from literal closures in the unit superposition calculus. Let us first formalise the notion of a proof in the unit superposition calculus.

Definition 4.18. A *US-proof P on literal closures* is a binary tree, drawn with the root at the bottom, consisting of persistent literal closures from $\mathcal{N}^\infty$ or $\mathcal{D}^\infty$ in a US-saturation process. Each non-leaf node $L \cdot \theta$ has exactly two children, which are the premises of an inference in the unit superposition calculus on literal closures (Definition 4.1 on page 45) resulting in $L \cdot \theta$. An exception is the root node, which may be a contradiction $\square$ and in this case has only one child node $(l \not\simeq r) \cdot \theta$ such that an equality resolution inference is applicable to it. Each leaf node is a relevant literal closure and each edge is labelled with the substitution σ in the inference from the child node to the parent.

The substitutions to instantiate clauses in order to make closures in persistent conflicts redundant are extracted from the US-proof of a contradiction of relevant

literal closures. By tracing the branches of the tree from the root to the leaves we obtain a substitution for each relevant literal closure.

Definition 4.19. Let P be a US-proof and the literal closure $L \cdot \theta$ be a leaf of P. Let $\sigma_1, \ldots, \sigma_n$ be the substitutions labelling the edges along the branch of P from the leaf $L \cdot \theta$ to the root. We call the composition $\sigma = \sigma_1 \cdots \sigma_n$ the *P-relevant instantiator* and the closure $L\sigma \cdot \rho$ the *P-relevant instance* of $L \cdot \theta$ where $L\sigma\rho = L\theta$.

We prove the following theorem relating leaf literal closures, relevant instantiators and the literal closure at the root of a proof.

Theorem 4.19. *Let P be a US-proof of the literal closure $L\cdot\theta$ or a contradiction $\Box$ (the root of P) from the literal closures $L_1 \cdot \theta_1, \ldots, L_k \cdot \theta_k$ (the leaves of P). Let $\sigma_1, \ldots, \sigma_k$ be the respective P-relevant instantiators for the leaf literal closures. Then*

$$L_1\sigma_1\rho, \ldots, L_k\sigma_k\rho \models L\rho$$

and

$$L_1\sigma_1\rho, \ldots, L_k\sigma_k\rho \models \Box,$$

respectively, for every substitution ρ grounding the literals $L_1\sigma_1, \ldots, L_k\sigma_k$.

Proof. We proceed by induction over the depth of the proof tree, that is the length n of the longest path in the US-proof P.

Case $n = 0$. $L \cdot \theta$ is a literal closure at a leaf and the relevant substitution is empty $[]$. Trivially, $L[]\rho \models L\rho$ for all substitutions ρ grounding L.

Case $n > 0$. If the root of the proof tree is the empty clause $\Box$ then there is an equality resolution inference with a literal closure $(l \not\approx r) \cdot \theta$ as premise. The depth of the US-proof of $(l \not\approx r) \cdot \theta$ is $n - 1$ and thus by the induction hypothesis

$$L_1\sigma'_1\tau, \ldots, L_k\sigma'_k\tau \models (l \not\approx r)\,\tau$$

for the instantiated literal closures $L_i\sigma'_i$ at the leaves and all grounding substitutions τ. Since $l\sigma = r\sigma$ with the substitution σ in the equality resolution inference, $(l \not\approx r)\,\sigma\rho \models \Box$ for all substitutions ρ. We choose $\tau = \sigma\rho$ and by

transitivity of $\models$ conclude

$$L_1\sigma'_1\sigma\rho, \ldots, L_k\sigma'_k\sigma\rho \models (l \not\simeq r)\,\sigma\rho \models \square,$$

where each $\sigma'_i\sigma$ is the relevant instantiator for L_i in the US-proof P.

If the literal closure at the root of the tree is $L \cdot \theta = (s[r] \simeq t)\,\sigma \cdot \rho'$ [or, respectively, $L \cdot \theta = (s[r] \not\simeq t)\,\sigma \cdot \rho'$], then it is the conclusion of a superposition inference between the literal closures $(l \simeq r) \cdot \theta_{\mathrm{l}}$ and $(s[l'] \simeq t) \cdot \theta_{\mathrm{r}}$ [$(s[l'] \not\simeq t) \cdot \theta_{\mathrm{r}}$]. Since the proof trees of these two literals are at most of depth $n-1$ we apply the induction hypothesis and conclude

$$L_1\sigma'_1\tau, \ldots, L_l\sigma'_l\tau \models (l \simeq r)\,\tau$$

and

$$L_{l+1}\sigma'_{l+1}\tau, \ldots, L_k\sigma'_k\tau \models (s[l'] \simeq t)\,\tau$$

$$[L_{l+1}\sigma'_{l+1}\tau, \ldots, L_k\sigma'_k\tau \models (s[l'] \not\simeq t)\,\tau]$$

for all grounding substitutions τ.

By Lemma 4.1 on page 47 we also have

$$(l \simeq r)\,\sigma\tau, (s[l'] \simeq t)\,\sigma\tau \models (s[r] \simeq t)\,\sigma\tau$$

$$[(l \simeq r)\,\sigma\tau, (s[l'] \not\simeq t)\,\sigma\tau \models (s[r] \not\simeq t)\,\sigma\tau]$$

for all grounding substitutions τ.

Since $\models$ is transitive, we choose $\tau = \sigma\rho$ and obtain

$$L_1\sigma'_1\sigma\rho, \ldots, L_l\sigma'_l\sigma\rho, L_{l+1}\sigma'_{l+1}\sigma\rho, \ldots, L_k\sigma'_k\sigma\rho \models L\rho$$

where $\sigma'_i\sigma$ are the relevant instantiators for L_i in the proof tree with of $L \cdot \theta$ at the root for all substitutions ρ grounding the leaf literals $L_1, \ldots, L_k$. □

We are in particular interested in US-proofs with a contradiction at the root and consider the special grounding substitution $\perp$ in the following corollary.

Corollary 4.20. *Let $L_1 \cdot \theta_1, \ldots, L_k \cdot \theta_k$ be the literals at the leaves of a US-proof with the empty clause $\square$ at the root and let $\sigma_1, \ldots, \sigma_k$ be the respective relevant instantiators for the leaf literal closures. The set $\{L_1\sigma_1\perp, \ldots, L_k\sigma_k\perp\}$ is unsatisfiable.*

Proof. We have $L_1\sigma_1\rho, \ldots, L_k\sigma_k\rho \models \square$ for every substitution ρ grounding the instantiated leaf literals $L_1\sigma_1, \ldots, L_k\sigma_k$ and in particular for $\rho = \bot$. Since the empty clause $\square$ follows from $\{L_1\sigma_1\bot, \ldots, L_k\sigma_k\bot\}$, the set is unsatisfiable. $\square$

The literals $\{L_1\bot, \ldots, L_k\bot\}$, which are the ground abstractions of the literal closures at the leaves of the proof, are true in the ground model $I_\bot$, but the set $\{L_1\sigma_1, \ldots, L_k\sigma_k\}$ of literals at the leaves of a US-proof of a contradiction, instantiated with the respective relevant instantiators, is unsatisfiable in first-order. Therefore the model of the ground abstraction $I_\bot$ has to be refined in order to prevent this situation. However, since the model of the ground abstraction is calculated by a ground solver which is a black box to us, we do not directly control the model. Instead we instantiate clauses and add them to the clause set in the Inst-saturation process such that the conflict becomes redundant, which triggers an adaption of the ground model for subsequent states of the saturation process.

Since a non-proper instance of a clause C does not make C redundant and the ground abstraction of C and a non-proper instance of it are identical, it remains to show that in every US-proof there is indeed a proper instantiator. Then, one of the closures in the conflict becomes redundant and hence adding clause instances from US-proofs makes every persistent conflict redundant and we obtain a fair Inst-saturation process.

Theorem 4.21. *Let $\langle K, \mathcal{L}\rangle$ be a persistent conflict and P be a proof of the contradiction from $\mathcal{L}$. At least one of the P-relevant instantiators is proper.*

Proof. Since $\langle K, \mathcal{L}\rangle$ is a persistent conflict, every literal L_i of the relevant literal closures $L_i \cdot \theta_i \in \mathcal{L}$ is selected infinitely often in the saturation process. Hence all $L_i\bot \in \mathcal{L}$ are true in the model $I^i_\bot$ in some state i and therefore consistent. Also, the US-saturation of some partition $\langle \mathcal{N}, \mathcal{D}\rangle$ of $\mathcal{L}$ contains the empty clause $\square$ and therefore there is a proof of $\square$ from literal closures in $\mathcal{L}$.

By Corollary 4.20 the set $\{L_1\sigma_1\bot, \ldots, L_k\sigma_k\bot\}$ where each σ_i is the respective relevant instantiator for the literal closure $L_i \cdot \theta_i$ is unsatisfiable. If all σ_i are non-proper, then $L_i\sigma_i\bot = L_i\bot$ and $\{L_1\sigma_1\bot, \ldots, L_k\sigma_k\bot\} = \mathcal{L}$ which is satisfiable. Therefore at least one σ_i must be non-proper. $\square$

Since every US-proof of the contradiction $\square$ from the relevant literals in a persistent conflict contains at least one proper instantiator, we now have a concrete fair Inst-saturation process (step ④ in Figure 4.1 on page 44). For a persistent

conflict $\langle K, \mathcal{L} \rangle$ there is a US-proof P of a contradiction $\square$ from the literal closures $\mathcal{L}$. Instantiating clauses of closures in K with the respective P-relevant instantiators makes at least one closure in K redundant as required by fairness of an Inst-saturation process.

Summary and Example

This concludes the proof of refutational completeness of the Inst-Gen-Eq method with the unit superposition calculus on literals. Let us summarise the main elements and aspects we have presented, referring again to Figure 4.1 on page 44.

In the centre we have the Inst-saturation process (Definition 4.14 on page 59) on a set of clauses S, from which a model $I_\perp$ of the ground abstraction $S\perp$ is calculated, which in turn induces a selection function sel. The selection function together with Inst-redundancy (Definition 4.9 on page 52) gives rise to a set of S-relevant literal closures $\mathcal{L}\,(S)$ (Definition 4.11 on page 53) of ground instances of S.

Exhaustively applying unit superposition inferences in a fair way (Definition 4.6 on page 51) to a partition $\langle \mathcal{N}, \mathcal{D} \rangle$ of the S-relevant literal closures and removing literal closures which are US-redundant (Definition 4.3 on page 48) leads to the US-saturation $\langle \mathcal{N}, \mathcal{D} \rangle^{\text{sat}}$ (Definition 4.7 on page 51) of the S-relevant literal closures.

If the ground abstraction $S\perp$ is unsatisfiable, the saturation process stops with the result unsatisfiable. If the ground abstraction $S\perp$ is satisfiable and the US-saturation does not contain a contradiction $\square$, a model of S exists (Theorem 4.4 on page 53) and the Inst-saturation process stops with the result satisfiable.

Otherwise, there are conflicts (Definition 4.16 on page 61) which are sets of closures and literal closures for which a US-proof (Definition 4.18 on page 65) of a contradiction in the unit superposition calculus exists. A fair Inst-saturation process (Definition 4.17 on page 61) must make persistent closures in conflicts redundant, then it is refutationally complete (Theorem 4.18 on page 64).

Since at least one relevant instantiator (Definition 4.19 on page 66) extracted from a US-proof of a contradiction is proper (Theorem 4.21 on the facing page), adding instances of the clauses with respective relevant instantiators makes closures in the conflict redundant, yielding a concrete Inst-saturation process.

Example 4.2. Let us resume Example 3.3 on page 37 in order to illustrate the

calculus on closures and literal closures. We have the following clause set with the underlined literals selected.

$$\underline{h(x) \simeq x} \ \vee \ x \not\simeq a \tag{1}$$

$$\underline{f(h(y)) \simeq g(z)} \tag{2}$$

$$\underline{f(a) \not\simeq g(u)} \tag{3}$$

We can consider the following closures, which are ground instances of (1)-(3) and not Inst-redundant.

$$\underline{h(x) \simeq x} \ \vee \ x \not\simeq a \cdot [a/x] \tag{1_a}$$

$$\underline{f(h(y)) \simeq g(z)} \cdot [a/y, a/z] \tag{2_a}$$

$$\underline{f(a) \not\simeq g(u)} \cdot [a/u] \tag{3_a}$$

With the unit superposition calculus on the literal closures L_1, L_2 and L_3, which are selected in the closures (1_a)-(3_a), we find a contradiction.

$$\dfrac{\dfrac{\dfrac{L_1 \qquad\qquad L_2}{h(x) \simeq x \cdot [a/x] \qquad f(h(y)) \simeq g(z) \cdot [a/y, a/z]}\,[x/y] \qquad\qquad L_3}{f(x) \simeq g(z) \cdot [a/x, a/z]} \qquad f(a) \not\simeq g(u) \cdot [a/u]}\,[a/x]$$

$$\dfrac{g(z) \not\simeq g(u) \cdot [a/z, a/u]}{\square}\,[z/u]$$

Hence the set of closures $K = \{(1_a), (2_a), (3_a)\}$ is a conflict which has to be made redundant. We achieve this by adding the clause instances

$$\underline{h(a) \simeq a} \ \vee \ a \not\simeq a \tag{4}$$

$$\underline{f(h(a)) \simeq g(z)} \tag{5}$$

to the input set of clauses.

There is only one ground instance of the ground clause (4), namely

$$\underline{h(a) \simeq a} \ \vee \ a \not\simeq a \cdot []. \tag{4_a}$$

Since $(1_a) \succ_{\text{cl}} (4_a)$ and $(4_a) \models (1_a)$, the closure (1_a) is Inst-redundant and therefore also the conflict K.

The ground solver finds the ground abstraction of the clause set (1)-(5) to be

unsatisfiable and thus the Inst-Gen-Eq method has proved the unsatisfiability of the initial clause set (1)-(3).

As we have proved in this section, the above procedure of adding clause instances to the clause set with instantiators from a US-proof of a conflict makes the conflict redundant. Further, if closures from each conflict are made redundant, we have a fair Inst-saturation process that guarantees finding an unsatisfiable ground abstraction for each unsatisfiable clause set.

4.5 Lifting Closures to First-Order Clauses

So far we have proved completeness of the Inst-Gen-Eq method, where an Inst-saturation process generates clause instances from ground closures of clauses. In the proof we have worked on the level of ground closures in order to allow for notions of redundancy. However, in practice it is not necessary to consider all ground closures separately, of which there may be infinitely many for every clause.

In Example 4.2 on page 69 we would have to consider in addition to closure (1_a) closures from clause (1) with every other grounding substitution. Thus there are more literal closures which are potentially S-relevant and have to be saturated under unit superposition inferences.

In the following we show that the unit superposition calculus on ground literal closures can be lifted to first-order literals. It suffices to apply the first-order unit superposition calculus as introduced in Definition 3.3 on page 37 without paying attention to individual literal closures representing the ground instances of the literal.

An Inst-saturation process lifted to first-order literals is fair if it generates clause instances in a way analogous to the Inst-saturation process on S-relevant literal closures. We are first concerned with the lifting and only in the next section consider elimination of redundancy by the way of a constraint notation.

In parallel to Definition 4.18 on page 65 we can define a proof from unit superposition inferences on literals as of Definition 3.3 on page 37.

Definition 4.20. A *US-proof P on literals* is a binary tree of literals, drawn with the root at the bottom. Each non-leaf node L has exactly two children, which are the premises of an inference in the unit superposition calculus on literals (Definition 3.3 on page 37) resulting in L. An exception is the root node, which may be a contradiction $\Box$, and in this case has only one child node $l \not\approx r$ such that

an equality resolution inference is applicable to it. Each leaf node is a selected literal and each edge is labelled with the substitution σ in the inference from the child node to the parent.

Definition 4.21. Let P be a US-proof on literals and the first-order literal L be a leaf of P. Let $\sigma_1, \ldots, \sigma_n$ be the substitutions labelling the edges along the branch of P from the leaf L to the root. We call the composition $\sigma = \sigma_1 \cdots \sigma_n$ the *P-relevant instantiator* and the literal $L\sigma$ the *P-relevant instance* of L.

Now we lift the result from Theorem 4.21 on page 68 on literal closures to first-order literals.

Theorem 4.22. *Let $\mathcal{L} = \{L_1 \cdot \theta, \ldots, L_k \cdot \theta_k\}$ be a set of literal closures such that $\{L_1\bot, \ldots, L_k\bot\}$ is satisfiable and there is a contradiction $\Box$ in the US-saturation of some partition $\langle \mathcal{N}, \mathcal{D} \rangle$ of $\mathcal{L}$. There is a US-proof of a contradiction with unit superposition inferences on the set of literals $\{L_1 \ldots, L_k\}$ such that at least one relevant instantiator is proper.*

Proof. There is a finite US-proof P of a contradiction $\Box$ from $\{L_1 \cdot \theta, \ldots, L_k \cdot \theta_k\}$. With Theorem 4.21 on page 68 at least one of the P-relevant instantiators is proper.

By inspecting the side conditions on inferences in the unit superposition calculus with ground literal closures and the unit superposition calculus with literals in Definition 4.1 on page 45 and Definition 3.3 on page 37, respectively, we find that if a unit superposition inference is applicable to literal closures $L_1 \cdot \theta_1$ and $L_2 \cdot \theta_2$ with the conclusion $L \cdot \theta$, then a superposition inference is applicable to the literals L_1 and L_2 with the conclusion L and the same substitution σ is applied in both inferences. Equally, if a contradiction $\Box$ follows from a literal closure $(l \not\approx r) \cdot \theta$, then a contradiction can be derived from the literal $l \not\approx r$ with the same substitution σ.

Thus we can simulate a unit superposition proof P of a contradiction from a set of literal closures $\{L_1 \cdot \theta, \ldots, L_k \cdot \theta_k\}$ by a unit superposition proof P' on the set of first-order literals $\{L_1 \ldots, L_k\}$. Further, if the set $\{L_1\bot, \ldots, L_k\bot\}$ is satisfiable, at least one P'-relevant instantiator is proper. $\Box$

As a corollary of this theorem we present a fair Inst-saturation process with the US-saturation lifted from literal closures to first-order literals. We have an Inst-saturation process $\left\{\left\langle S^i, I^i_\bot, \mathrm{sel}^i \right\rangle\right\}_{i=1}^{\infty}$, where persistent conflicts have to

be made redundant. For $\langle K, \mathcal{L}\rangle$ with $K = \{(L_1 \vee C_1)\cdot\theta_1, \ldots, (L_n \vee C_n)\cdot\theta_n\}$ and $\mathcal{L} = \{L_1\cdot\theta'_1, \ldots, L_n\cdot\theta'_n\}$, where $\theta'_i = \theta_i\,|_{\mathrm{var}(L_i)}$, there is a state i of the Inst-saturation process where

(i) $\mathrm{sel}^i(L_j \vee C_j) = L_j$ for all $1 \leq j \leq n$ and

(ii) $\langle \mathcal{N}, \mathcal{D}\rangle^{\mathrm{sat}}$ contains a contradiction.

The fact (i) implies the satisfiability of $\{L_1\bot, \ldots, L_n\bot\}$ with a model $I_\bot$ and by (ii) and Theorem 4.22 there is a US-proof P on literals, where at least one P-relevant instantiator is proper. Without loss of generality let σ_1 for L_1 be proper, then adding $C_1\sigma_1$ to S^i makes $C_1\cdot\theta_1$ redundant in state $i+1$ and thus also the conflict $\langle K, \mathcal{L}\rangle$.

The procedure described in Example 3.3 on page 37 and Figure 3.2 on page 36 is therefore a refutationally complete method: we saturate selected literals under inferences in the unit superposition calculus and for each contradiction that is derived, extract relevant substitutions from the proof. We continue after adding instances of clauses with the respective relevant substitutions until either the ground abstraction becomes unsatisfiable or we achieve saturation of the selected literals without contradictions.

4.6 Redundancy Elimination with Constraints

Lifting the unit superposition calculus from ground literal closures to first-order literals is an essential step, since it is not practically possible to deal with all ground instances separately. However, the straightforward lifting in the previous section is too coarse to capture the notion of Inst-redundancy and it is crucial for efficiency to employ a constraint mechanism to eliminate Inst-redundancy in the lifted unit superposition calculus.

The lifting in Theorem 4.22 on the preceding page only ensures that every US-proof on literal closures corresponds to a US-proof on literals, but does not lift Inst-redundancy. There are US-proofs on literals for which all corresponding US-proofs on literal closures are redundant. That is because in every such US-proof on literal closures at least one literal closure is not relevant. This in turn is due to Inst-redundancy of the closure the literal closure is selected in. Hence the conflict that contains the closure is already irrelevant and fairness of the Inst-saturation process (Definition 4.17 on page 61) holds without generating clause

instances from the US-proof on literals.

We need to integrate the notion of Inst-redundancy in a concrete way into the Inst-saturation process with US-proofs on first-order literals. Inst-redundancy arises when clauses are instantiated: a closure $C\cdot\sigma\theta$ of a clause $C \in S$ is redundant if there is a clause $C\sigma \in S$, since the ground clause $C\sigma\theta$ is represented twice, namely by $C \cdot \sigma\theta$ and $C\sigma \cdot \theta$. Our intention is to have the ground clause $C\sigma\theta$ only represented by the smaller closure $C\sigma \cdot \theta$ of $C\sigma$ and to block the closure $C \cdot \sigma\theta$ in the closures of C.

We introduce the following constraint notion, which has already been a key component in the non-equational Inst-Gen method [Korovin, 2009]. Dismatching constraints allow eliminating precisely the kind of redundancy occurring in an Inst-saturation process, similar constraints have appeared in Comon [1991] and Caferra and Zabel [1992].

Definition 4.22. A *dismatching pair* $\overline{s} \not\rhd \overline{t}$ consists of two variable-disjoint n-tuples of terms $\overline{s} = \langle s_1, \ldots, s_n\rangle$ and $\overline{t} = \langle t_1, \ldots, t_n\rangle$. A *dismatching constraint* is either a dismatching pair $D = \overline{s} \not\rhd \overline{t}$, an n-ary conjunction $D_1 \wedge \cdots \wedge D_n$ of dismatching constraints or an n-ary disjunction $D_1 \vee \cdots \vee D_n$ of dismatching constraints, where for each two dismatching pairs $\overline{s} \not\rhd \overline{t}$ and $\overline{s'} \not\rhd \overline{t'}$ in a dismatching constraint the tuples $\overline{s}$ and $\overline{s'}$ are variable-disjoint.

A substitution σ *satisfies* a dismatching constraint D if

(i) $D = \overline{s} \not\rhd \overline{t}$ and there is no substitution τ such that $\overline{s}\tau = \overline{t}\sigma$, in other words, the tuple $\overline{s}$ cannot be matched to $\overline{t}\sigma$ or

(ii) $D = D_1 \wedge \cdots \wedge D_n$ and σ satisfies each of the dismatching constraints D_i for $1 \leq i \leq n$ or

(iii) $D = D_1 \vee \cdots \vee D_n$ and σ satisfies at least one of the dismatching constraints D_i for $1 \leq i \leq n$.

Let σ be a substitution and D be a dismatching constraint. The σ-*instance* of D is the constraint

$$D\sigma = \begin{cases} \overline{s} \not\rhd \overline{t}\sigma & \text{if } D = \overline{s} \not\rhd \overline{t}, \\ D_1\sigma \wedge \cdots \wedge D_n\sigma & \text{if } D = D_1 \wedge \cdots \wedge D_n \text{ and} \\ D_1\sigma \vee \cdots \vee D_n\sigma & \text{if } D = D_1 \vee \cdots \vee D_n. \end{cases}$$

The *constraint D_σ of a substitution* σ, where $\mathrm{dom}(\sigma) = \{x_1, \ldots, x_n\}$, is the dismatching pair $\langle x_1\sigma, \ldots, x_n\sigma\rangle \not\triangleright \langle x_1, \ldots, x_n\rangle$.

If a dismatching constraint is the nullary conjunction, it is satisfied by any substitution. We call the nullary conjunction the *empty dismatching constraint* and denote it by $\top$.

We note that disjunctive dismatching constraints are only used in one of the labelled approaches in the next chapter in Section 5.3 on page 110.

Definition 4.23. A *constrained clause* $C \mid D$ is a clause C with a dismatching constraint D. We let $\mathrm{Cl}(C \mid D)$ denote the set of all ground closures $C \cdot \theta$ of C where θ satisfies D.

We let $\mathrm{Unc}(C \mid D)$ denote the unconstrained clause $C \mid \top$ and also write just C for the unconstrained clause $C \mid \top$.

Let us prove the property that makes dismatching constraints a useful concept in instantiation-based theorem proving.

Lemma 4.23. *If the substitution σ does not satisfy the dismatching constraint D, then neither does a more specific substitution $\sigma\rho$ for any substitution ρ.*

Proof. Let $D = \overline{s} \not\triangleright \overline{t}$ be not satisfied by σ, then there is a substitution τ such that $\overline{s}\tau = \overline{t}\sigma$. Since $\overline{s}\tau\rho = \overline{t}\sigma\rho$, we have a substitution $\tau\rho$ hence $\sigma\rho$ does not satisfy D.

We continue by induction over the structure of D.

Let $D = D_1 \wedge \cdots \wedge D_n$ be not satisfied by σ. In this case, at least one D_i is not satisfied by σ and, using the induction hypothesis, this D_i is not satisfied by any $\sigma\rho$, either. Hence, D is not satisfied by any $\sigma\rho$.

Let $D = D_1 \vee \cdots \vee D_n$ be not satisfied by σ, then D_i is not satisfied for all $1 \leq i \leq n$. Due to the induction hypothesis, no D_i is satisfied by any $\sigma\rho$ and therefore D is not satisfied by any $\sigma\rho$. □

Corollary 4.24. *A satisfiable dismatching constraint is satisfied by the empty substitution* $[]$.

Proof. We take the contraposition of the previous lemma: if the dismatching constraint D is satisfied by a substitution $\sigma\rho$, then it is satisfied by σ. Now let D be satisfiable by a substitution ρ, and choose $\sigma = []$. □

If a dismatching constraint blocks a substitution σ, then it blocks all substitutions more specific than σ. In particular the dismatching constraint D_σ of a substitution σ blocks all substitutions that are more specific than σ. We exploit this fact in the instantiation process. If a clause $C \mid D$ is instantiated to $C\sigma$, we conjunctively add the constraint D_σ of the substitution σ to the dismatching constraint D and continue with the constrained clause $C \mid D \wedge D_\sigma$. The closures $\mathrm{Cl}\,(C \mid D_\sigma)$ are the ground closures $C \cdot \theta$ for all θ, except the closures $C \cdot \sigma\rho$ for all ρ grounding $C\sigma$. The ground clauses $C\sigma\rho$ are represented by closures of the unconstrained instance $C\sigma$ instead.

Example 4.3. Let $\sigma_1 = [f(u,v)/x]$ and $\sigma_2 = [a/u, b/v]$ be two substitutions and $D_\tau = \langle f(a,v)\rangle \not\trianglerighteq \langle x\rangle$ be the constraint of the substitution $\tau = [f(a,v)/x]$.

The dismatching constraint D_τ is satisfied by the substitution $\sigma_1 = [f(u,v)/x]$, since $\langle f(a,v)\rangle\,\rho \neq \langle f(u,v)\rangle = \langle x\rangle\,\sigma_1$ for every substitution ρ.

Instantiating the constraint D_τ with the substitution σ_1, we obtain the constraint $D_\tau\sigma_1 = \langle f(a,v)\rangle \not\trianglerighteq \langle f(u,v)\rangle$. The substitution $\sigma_2 = [a/u, b/v]$ does not satisfy $D_\tau\sigma_1$, since with $\rho = [b/v]$ we have $\langle f(a,v)\rangle\,\rho = \langle f(a,b)\rangle = \langle f(u,v)\rangle\,\sigma_2$.

Moreover, the composition of the two substitutions $\sigma_1\sigma_2 = [f(a,b)/x]$ does not satisfy the constraint D_τ, since the same substitution $\rho = [b/v]$ results in $\langle f(a,v)\rangle\,\rho = \langle f(a,b)\rangle = \langle x\rangle\,\sigma_1\sigma_2$.

The dismatching constraint $D_\tau = \langle f(a,v)\rangle \not\trianglerighteq \langle x\rangle$ blocks all substitutions more specific than $\tau = [f(a,v)/x]$, in this example $\sigma_1\sigma_2$, but it does not block σ_1.

Finally, the constraint $D_\tau\sigma_1\sigma_2 = \langle f(a,v)\rangle \not\trianglerighteq \langle f(a,b)\rangle$ is unsatisfiable.

We prove the following lemma that is relevant to checking dismatching constraints in practice: instead of satisfiability of an instantiated constraint $D\sigma$ with a substitution ρ, we can consider satisfiability of the constraint D with the composed substitution $\sigma\rho$.

Lemma 4.25. *The composition of two substitutions $\sigma\rho$ satisfies the constraint D if and only if ρ satisfies the instantiated constraint $D\sigma$.*

Proof. Let $D = \overline{s} \not\trianglerighteq \overline{t}$ and $D\sigma = \overline{s} \not\trianglerighteq \overline{t}\sigma$ be two dismatching pairs. If a substitution $\sigma\rho$ satisfies D, then there exists a substitution τ, such that $\overline{s}\tau = \overline{t}\sigma\rho$. Hence, ρ satisfies $D\sigma = \overline{s} \not\trianglerighteq \overline{t}\sigma$. If a substitution ρ satisfies $D\sigma$, then there exists a substitution τ, such that $\overline{s}\tau = \overline{t}\sigma\rho$ and $D = \overline{s} \not\trianglerighteq \overline{t}$ is satisfied by $\sigma\rho$.

We continue by induction over the structure of the dismatching constraint D. If $D = D_1 \wedge \cdots \wedge D_n$, then $D\sigma = D_1\sigma \wedge \cdots \wedge D_n\sigma$. Due to the induction hypothesis, the substitution $\sigma\rho$ satisfies the constraints D_i if and only if the substitution ρ satisfies the constraints $D_i\sigma$. Hence, D is satisfied by $\sigma\rho$ if and only if $D\sigma$ is satisfied by ρ. In analogy we prove the lemma if the dismatching constraint D is a disjunction and conclude that the lemma holds for any dismatching constraint. □

Checking if a substitution σ satisfies a dismatching constraint D can be polynomially reduced to deciding if a monotone Boolean formula is satisfied by an assignment of truth values to its variables. A monotone Boolean formula contains no negations and we can decide if an assignment satisfies a Boolean formula without negations in time linear to the size of the formula.

We recursively reduce a dismatching constraint D to a Boolean formula by mapping conjunctions and disjunctions in D to Boolean conjunctions and disjunctions. We map a dismatching pair $\overline{s} \not\triangleright \overline{t}$ to the propositional constant true or false, respectively, if the constraint is satisfiable or unsatisfiable.

Deciding if a substitution σ satisfies a dismatching pair is equivalent to solving the matching problem on the n-tuple of term $\overline{s}$ and $\overline{t}$. The matching problem is a sub-problem of syntactic unification, the problem of finding a substitution ρ such that $l\rho = r\rho$ for two terms l and r. Unification and matching are well studied [Baader and Snyder, 2001] and efficient linear time and space-efficient algorithms in almost-linear time exist.

As Korovin [2009] notes, our problem of checking if a given substitution σ satisfies a dismatching constraint D can therefore be solved in polynomial time. Moreover, we can use sophisticated algorithms like term indexing [Graf, 1996] in an implementation.

We now extend the unit superposition calculus from Definition 3.3 on page 37 with dismatching constraints and eagerly check dismatching constraints in each inference. A selected literal inherits the dismatching constraint of the clause it is selected in.

Definition 4.24 (Unit Superposition with Dismatching Constraints)**.**

Unit Superposition

$$\frac{(l \simeq r) \mid D_{\mathrm{l}} \qquad (s[l'] \simeq t) \mid D_{\mathrm{r}}}{(s[r] \simeq t)\sigma \mid D_{\mathrm{l}}\sigma \wedge D_{\mathrm{r}}\sigma}\,(\sigma) \qquad \frac{(l \simeq r) \mid D_{\mathrm{l}} \qquad (s[l'] \not\simeq t) \mid D_{\mathrm{r}}}{(s[r] \not\simeq t)\,\sigma \mid D_{\mathrm{l}}\sigma \wedge D_{\mathrm{r}}\sigma}\,(\sigma)$$

where for some grounding substitution θ with $\mathrm{dom}\,(\theta) = \mathrm{var}\,(\{l\sigma, r\sigma, s[l']\sigma, t\sigma\})$

(i) $\sigma = \mathrm{mgu}(l, l')$,

(ii) l' is not a variable,

(iii) $l\sigma\theta \succ_{\mathrm{gr}} r\sigma\theta$,

(iv) $s[l']\sigma\theta \succ_{\mathrm{gr}} t\sigma\theta$,

(v) $\mathrm{var}\,(\{l, r\}) \cap \mathrm{var}\,(\{s[l'], t\}) = \emptyset$.

and σ satisfies the dismatching constraints D_{l} and D_{r}.

Equality Resolution

$$\frac{(l \not\simeq r) \mid D}{\square}\,(\sigma)$$

where $\sigma = \mathrm{mgu}(l, r)$ and σ satisfies the dismatching constraint D.

The unit superposition calculus with dismatching constraints allows to discard proofs early, which would produce redundant instances. By using dismatching constraints we reduce the number of literals to be considered in the US-saturation and in this way also avoid producing proofs which would generate redundant instances.

Example 4.4. Consider the unsatisfiable clause set

$$\underline{f(g(x), a) \not\simeq c} \;\vee\; f(x, y) \not\simeq c \tag{1}$$

$$\underline{f(g(z), z) \simeq c} \tag{2}$$

$$\underline{f(a, a) \simeq c}. \tag{3}$$

The ground abstraction is satisfiable and the first literals in each clause can

be selected. There is a proof of a contradiction as follows.

$$\dfrac{\dfrac{\overset{L_2}{f(g(z),z) \simeq c} \qquad \overset{L_1}{f(g(x),a) \not\simeq c}}{c \not\simeq c}\,[a/x, a/z]}{\square}$$

We obtain two new instances of clauses (1) and (2)

$$f(g(a),a) \not\simeq c \;\vee\; \underline{f(a,u) \not\simeq c} \tag{4}$$

$$\underline{f(g(a),a) \simeq c}. \tag{5}$$

The ground abstraction remains satisfiable, but we can only select the second literal in the new clause (4), since its first (ground) literal conflicts with the new ground unit clause (5).

We also extend the previously empty dismatching constraints of the instantiated clauses with the constraints of the respective instantiating substitution.

$$\underline{f(g(x),a) \not\simeq c} \;\vee\; f(x,y) \not\simeq c \quad \mid \langle a\rangle \not\rhd \langle x\rangle \tag{1}$$

$$\underline{f(g(z),z) \simeq c} \quad \mid \langle a\rangle \not\rhd \langle z\rangle \tag{2}$$

The ground closures of the two constrained clauses (1) and (2) do not contain the two closures blocked by their respective constraints:

$$\underline{f(g(x),a) \not\simeq c} \;\vee\; f(x,y) \not\simeq c \cdot [a/x] \tag{1_a}$$

$$\underline{f(g(z),z) \simeq c} \cdot [a/z]. \tag{2_a}$$

These ground instances are now represented by the ground closures of the clause instances (4) and (5) instead.

With the new instance (5) there is another proof of a contradiction

$$\dfrac{\dfrac{\overset{L_5}{f(g(a),a) \simeq c} \qquad \overset{L_1}{f(g(x),a) \not\simeq c \mid \langle a\rangle \not\rhd \langle x\rangle}}{c \not\simeq c}\,[a/x]}{\square}$$

that would lead to instance (4) again. However, since the substitution $[a/x]$ in the first inference step is blocked by the dismatching constraint on literal L_1 from clause (1), the proof is redundant. The dismatching constraint tells us that it is

not necessary to consider conflicts containing the literal closure $L_1 \cdot [a/x]$ selected in closure (1_a), since this closure is redundant.

Indeed, the inconsistency the blocked proof has found also holds for the instantiated clauses (4) and (5) and can already be witnessed in the ground abstraction. The first literal $f(g(a), a) \not\simeq c$ in clause (4), which is identical to the literal represented by the blocked literal closure $L_1 \cdot [a/x]$, cannot be selected and no further inferences in first-order reasoning are necessary.

We continue with the following proof of a contradiction

$$\dfrac{\dfrac{\begin{array}{cc} L_5 & L_4 \\ f(a,a) \simeq c & f(a,u) \not\simeq c \end{array}}{c \not\simeq c}}{\square}\,[a/x],$$

which leads to the clause instance

$$f(g(a), a) \not\simeq c \;\vee\; f(a, a) \not\simeq c. \tag{6}$$

Clauses (3), (5) and (6), which are identical to their ground abstractions, are unsatisfiable and thus the input clause set (1)-(3) is.

Using the unit superposition calculus with dismatching constraints therefore blocks proofs of a contradiction in conflicts, where closures are already Inst-redundant. Since the constraints are checked eagerly before each inference, literals that are blocked are not added to the saturation process.

In order to show refutational completeness of redundancy elimination with dismatching constraints in the unit superposition calculus, we need to lift the US-saturation process from literal closures to constrained literals.

In the following we consider only dismatching constraints that model Inst-redundancy in the following sense.

Definition 4.25. We extend the notions of Cl and Unc from Definition 4.23 to sets of clauses:

$$\mathrm{Cl}\,(S) = \{C \cdot \theta \in \mathrm{Cl}\,(C) \mid C \in S\}$$

and

$$\mathrm{Unc}\,(S) = \{\mathrm{Unc}\,(C) \mid C \in S\}\,.$$

A set of clauses S is *well-constrained* if

$$\mathrm{Cl}\,(S) \setminus \mathcal{R}_{\mathrm{Inst}}\,(\mathrm{Cl}\,(S)) \supseteq \mathrm{Cl}\,(\mathrm{Unc}\,(S)) \setminus \mathcal{R}_{\mathrm{Inst}}\,(\mathrm{Cl}\,(\mathrm{Unc}\,(S)))\,,$$

which means that a dismatching constraint must not block closures that are not Inst-redundant.

The closures of a set of well-constrained clauses contain all closures which are not Inst-redundant. We note that adding the constraint D_σ of a substitution σ to the dismatching constraint of the clause $C \mid D$ when adding an instance $C\sigma$ to S keeps the clause set S well-constrained.

We now lift the definitions about the US-saturation process from Section 4.1 from literal closures to constrained literals. We give the new notion of USD-redundancy that incorporates Inst-redundancy by the means of dismatching constraints, define a USD-saturation process on constrained literal closures and the USD-saturation.

Definition 4.26. A constrained clause $C \mid D$ is *Inst-redundant* if all closures in $\mathrm{Cl}\,(C \mid D)$ are Inst-redundant.

Definition 4.27. Let $\mathcal{L}D$ be a set of constrained literals. A constrained literal $L \mid D \in \mathcal{L}D$ is *USD-redundant* in $\mathcal{L}D$ if all literal closures in $\mathrm{Cl}\,(L \mid D)$ are US-redundant in $\mathrm{Cl}\,(\mathcal{L}D)$.

Let $\mathcal{R}_{\mathrm{USD}}\,(\mathcal{L}D)$ denote the set of all constrained literals in $\mathcal{L}D$, which are USD-redundant in $\mathcal{L}D$.

Definition 4.28. A *USD-saturation process* on constrained literals is an infinite sequence of pairs $\{\langle \mathcal{N}D^i, \mathcal{D}D^i \rangle\}_{i=1}^{\infty}$, where $\mathcal{N}D^i$ is a set of constrained literals and $\mathcal{D}D^i$ is a disjoint set of positive constrained literals. Each such pair $\langle \mathcal{N}D^{i+1}, \mathcal{D}D^{i+1} \rangle$ with $i > 1$, called a *successor state*, is obtained from the previous state $\langle \mathcal{N}D^i, \mathcal{D}D^i \rangle$ by either

(i) adding to $\mathcal{N}D^i$ the conclusion of a unit superposition inference with one premise in $\mathcal{N}D^i$ and the second premise in $\mathcal{N}D^i \cup \mathcal{D}D^i$, or the conclusion of an equality resolution inference with its premise in $\mathcal{N}D^i$,

(ii) adding to $\mathcal{D}D^i$ the conclusion of a unit superposition inference with both premises in $\mathcal{D}D^i$, or the conclusion of an equality resolution inference with its premise in $\mathcal{D}D^i$,

(iii) removing a constrained literal from $\mathcal{N}D^i$ which is USD-redundant in $\mathcal{N}D^i \cup \mathcal{D}D^i$ or

(iv) removing a literal from $\mathcal{D}D^i$ which is USD-redundant in $\mathcal{D}D^i$.

Definition 4.29. Let us denote by $\mathcal{N}D^\infty$ and $\mathcal{D}D^\infty$ the sets of *persistent literals* in the USD-saturation process, which are the lower limits of $\{\mathcal{N}D^i\}_{i=1}^\infty$ and $\{\mathcal{D}D^i\}_{i=1}^\infty$, respectively:

$$\mathcal{N}D^\infty = \liminf_{i\to\infty} \mathcal{N}D^i = \bigcup_{n=1}^{\infty} \left(\bigcap_{m=n}^{\infty} \mathcal{N}D^m \right)$$

$$\mathcal{D}D^\infty = \liminf_{i\to\infty} \mathcal{D}D^i = \bigcup_{n=1}^{\infty} \left(\bigcap_{m=n}^{\infty} \mathcal{D}D^m \right)$$

Definition 4.30. A USD-saturation process is *fair* if

(i) for every unit superposition inference with one premise in $\mathcal{N}D^\infty$ and the second premise in $\mathcal{N}D^\infty \cup \mathcal{D}D^\infty$, there is an i such that the conclusion is in $\mathcal{N}D^\infty$ or USD-redundant in $\mathcal{N}D^i \cup \mathcal{D}D^i$ and

(ii) for every unit superposition inference with both premises in $\mathcal{D}D^\infty$ there is a j such that the conclusion is in $\mathcal{D}D^\infty$ or USD-redundant in $\mathcal{D}D^j$.

Definition 4.31. Let $\langle \mathcal{N}D, \mathcal{D}D \rangle$ be a pair of a set of constrained literals $\mathcal{N}D$ and a disjoint set of positive constrained literals $\mathcal{D}D$. Let $\{\langle \mathcal{N}D^i, \mathcal{D}D^i \rangle\}_{i=1}^\infty$ be an arbitrary but fixed and fair USD-saturation process with $\mathcal{N}D^1 = \mathcal{N}D$ and $\mathcal{D}D^1 = \mathcal{D}D$. We call the set

$$\langle \mathcal{N}D, \mathcal{D}D \rangle^{\text{sat}} = \left(\mathcal{N}D^\infty \setminus \mathcal{R}_{\text{USD}}\left(\mathcal{N}D^\infty \cup \mathcal{D}D^\infty\right)\right) \cup \left(\mathcal{D}D^\infty \setminus \mathcal{R}_{\text{USD}}\left(\mathcal{D}D^\infty\right)\right)$$

the *USD-saturation* of $\langle \mathcal{N}D, \mathcal{D}D \rangle$.

Since we now have a lifted USD-saturation process on constrained literals, we also need to lift the notion of relevant literal closures from a set of clauses to a set of constrained clauses.

Definition 4.32. Let sel be a selection function based on a model $I_\perp$ of the ground abstraction $\text{Unc}(S) \perp$ of the set S of constrained clauses. The set of S-relevant constrained literals $\mathcal{L}\text{D}(S)$ contains all constrained literals $L \mid D$ such that

(i) $(L \vee C) \mid D \in S$,

(ii) $(L \vee C) \mid D$ is not Inst-redundant in S,

(iii) $L = \mathrm{sel}\,(L \vee C)$.

Definition 4.33. Let S be a set of constrained clauses and $\langle \mathcal{N}D, \mathcal{D}D \rangle$ be a partition of the S-relevant constrained literals $\mathcal{L}\mathrm{D}\,(S)$ such that $S \models \mathrm{Unc}\,(l \simeq r \mid D)$ holds for each constrained literal $l \simeq r \mid D \in \mathcal{D}D$. Let $\langle \mathcal{N}D, \mathcal{D}D \rangle^{\mathrm{sat}}$ be the USD-saturation of $\langle \mathcal{N}D, \mathcal{D}D \rangle$. The set of clauses S is *Inst-saturated* with regard to a selection function sel and the partition $\langle \mathcal{N}D, \mathcal{D}D \rangle$ if $\langle \mathcal{N}D, \mathcal{D}D \rangle^{\mathrm{sat}}$ does not contain a contradiction $\Box$.

Definition 4.34. A *USD-proof P on constrained literals* is a binary tree, drawn with the root at the bottom, consisting of constrained literals in $\langle \mathcal{N}D, \mathcal{D}D \rangle^{\mathrm{sat}}$. Each non-leaf node $L \mid D$ has exactly two children, which are the premises of an inference in the unit superposition calculus with dismatching constraints (Definition 4.24) resulting in $L \mid D$. An exception is the root node, which may be a contradiction $\Box$ and in this case has only one child node $l \not\simeq r \mid D$ such that an equality resolution inference is applicable to it. Each leaf node is a relevant literal closure and each edge is labelled with the substitution σ in the inference from the child node to the parent.

Lemma 4.26. *Each P-relevant instantiator from a USD-proof P of a constrained literal $L \mid D$ satisfies the dismatching constraint D if the dismatching constraints of the leaf literals in P are satisfiable.*

Proof. We additionally propose and prove that the dismatching constraint of a literal $L \mid D$ derived in a USD-proof P is the conjunction of dismatching constraints $D_1\sigma_1 \wedge \cdots \wedge D_n\sigma_n$, where each D_i is the dismatching constraint of a leaf literal $L_i \mid D_i$ in P and σ_i is the P-relevant instantiator for $L_i \mid D_i$.

Let $L \mid D$ be a literal with a satisfiable dismatching constraint. If $L \mid D$ is a leaf literal and the root of an empty USD-proof P, then there is only one P-relevant instantiator, which is the empty substitution [] and satisfies the dismatching constraint D. Further, the dismatching constraint $D = D[]$ of the root $L \mid D$ is the unary conjunction of the dismatching constraint D of the leaf literal $L \mid D$ instantiated with the P-relevant instantiator [].

Let $L \mid D \neq \Box \mid D$ be the conclusion of a unit superposition inference from the premises $l \simeq r \mid D_{\mathrm{l}}$ and $L' \mid D_{\mathrm{r}}$, derived in the USD-proofs P_{l} and P_{r}, respectively.

By the induction hypothesis the dismatching constraints $D_\mathrm{l} = D_1\sigma_1 \wedge \cdots \wedge D_l\sigma_l$ and $D_\mathrm{r} = D_{l+1}\sigma_{l+1} \wedge \cdots \wedge D_n\sigma_n$ are conjunctions of instantiated dismatching constraints, where $\sigma_1, \ldots, \sigma_l$ and $\sigma_{l+1}, \ldots, \sigma_n$ are the P_l- and P_r-relevant instantiators and $D_1, \ldots, D_l$ and $D_{l+1}, \ldots, D_n$ are the dismatching constraints of the leaf literals in P_l and P_r. Each σ_i satisfies all the dismatching constraints $D_1, \ldots, D_n$.

Let us consider the USD-proof P with $L \mid D$ at the root and the above premises $l \simeq r \mid D_\mathrm{l}$ and $L' \mid D_\mathrm{r}$ as children of the root. The substitution σ of the unit superposition inference satisfies both dismatching constraints D_l and D_r, hence each P-relevant instance $\sigma_1\sigma, \ldots, \sigma_n\sigma$ satisfies all the dismatching constraints $D_1, \ldots, D_n$. Moreover, the dismatching constraint of the conclusion is $D_\mathrm{l}\sigma \wedge D_\mathrm{r}\sigma = D_1\sigma_1\sigma \wedge \cdots \wedge D_n\sigma_n\sigma$.

We draw an analogous conclusion for the equality resolution inference, which proves the lemma. □

We can further exploit this in an implementation to avoid instantiating the constraints D_l and D_r in the conclusion of an inference: we check if the constraints of the leaf literals inherited from the clause are satisfied by the respective relevant instantiators instead.

With the previous definitions we can state and prove the following lifting lemma, where completeness follows from a corollary.

Lemma 4.27. *Let $\mathcal{L}D$ be a set of constrained literals. If the US-saturation of a partition of the literal closures $\mathrm{Cl}(\mathcal{L}D)$ into $\langle \mathcal{N}, \mathcal{D} \rangle$ contains a contradiction, then the USD-saturation of the partition $\langle \mathcal{N}D, \mathcal{D}D \rangle$ of the constrained literals $\mathcal{L}D$ contains a contradiction.*

Proof. If the set of literal closures $\langle \mathcal{N}, \mathcal{D} \rangle^{\mathrm{sat}}$ contains a contradiction, then there is a US-proof of a contradiction from a finite set of literal closures in $\langle \mathcal{N}, \mathcal{D} \rangle^{\mathrm{sat}}$. We show by induction that there is also a USD-proof of a contradiction on constrained literals.

By Theorem 4.22 on page 72 we already have that there is a US-proof on literals if there is a US-proof on literal closures. Therefore we only need to consider the additional restriction of the inference rules in Definition 4.24 introduced by the dismatching constraints. We show that a US-proof P on literal closures can be simulated by a USD-proof P' on constrained literals.

For every leaf literal closure $L \cdot \theta$ in $\mathrm{Cl}(\mathcal{L}D)$ we have that θ satisfies the dismatching constraint of $L \mid D \in \mathcal{L}D$. We therefore assume as an induction

hypothesis that for two literal closures $(l \simeq r) \cdot \theta_\mathrm{l}$ and $(s[l'] \simeq t) \cdot \theta_\mathrm{r}$ in the US-proof P the substitutions θ_l and θ_r satisfy the dismatching constraints D_l and D_r of the constrained literals $(l \simeq r) \mid D_\mathrm{l}$ and $(s[l'] \simeq t) \mid D_\mathrm{r}$ in the USD-proof P'. Because $(\theta_\mathrm{l} \cup \theta_\mathrm{r}) = \sigma\rho$, it follows that the substitution ρ in the conclusion of the unit superposition inference $(s[r] \simeq t)\,\sigma \cdot \rho$ satisfies the dismatching constraints $D_\mathrm{l}\sigma$ and $D_\mathrm{r}\sigma$. The same argument holds if we consider the literal $(s[l'] \not\simeq t) \mid D_\mathrm{r}$ as the right premise and we conclude that a unit superposition inference on constrained literals is applicable if a unit superposition inference on literal closures is.

Since for a literal $(l \not\simeq r) \cdot \theta$ the substitution θ satisfies the dismatching constraint D of the literal $(l \not\simeq r) \mid D$, the mgu σ of l and r also satisfies D and an equality resolution inference on a constrained literal is applicable if an equality resolution inference on a literal closure is applicable.

We can therefore simulate the US-proof P on literal closures from $\mathrm{Cl}\,(\mathcal{L}D)$ by a USD-proof P' on constrained literals from $\mathcal{L}D$, where in each inference step the dismatching constraint condition is satisfied. Hence every contradiction in $\langle \mathcal{N}, \mathcal{D} \rangle^{\mathrm{sat}}$ is also found in $\langle \mathcal{N}D, \mathcal{D}D \rangle^{\mathrm{sat}}$. □

Finally we can lift the model generation theorem (Theorem 4.4 on page 53).

Theorem 4.28. *If a well-constrained set of clauses S is Inst-saturated and the ground abstraction* $\mathrm{Unc}\,(S) \bot$ *is satisfiable, then* $\mathrm{Unc}\,(S)$ *is also satisfiable.*

Proof. We use the contraposition of Lemma 4.27. Since S is Inst-saturated, the USD-saturation of the S-relevant constrained literals $\mathcal{L}D\,(S)$ does not contain a contradiction and thus the US-saturation of the literal closures of the constrained S-relevant literals $\mathrm{Cl}\,(\mathcal{L}D\,(S))$ does not contain a contradiction, either. Since S is well-constrained, $\mathcal{L}D\,(S)$ contains all S-relevant literal closures and $\mathrm{Cl}\,(\mathcal{L}D\,(S)) \supseteq \mathcal{L}\,(\mathrm{Unc}\,(S))$. In other words, the set of literal closures of the S-relevant constrained literals contains all $\mathrm{Unc}\,(S)$-relevant literal closures. The model generation in Theorem 4.4 on page 53 applies and we conclude that $\mathrm{Unc}\,(S)$ is satisfiable. □

The last theorem concludes the proof of refutational completeness of the Inst-Gen-Eq method in this chapter. Let us briefly review the main characteristics of the calculus and the important role the lifting theorems.

We are working on a set of constrained clauses and a ground abstraction. Based on a model of the ground abstraction that is calculated by a ground solver

modulo equality we select one literal in each clause. Each selected literal inherits the dismatching constraint of its clause and we saturate the set of selected literals under the unit superposition calculus with dismatching constraints. If a contradiction is found, the clauses of the selected literals form a conflict and we can extract substitutions to instantiate some clauses in order to make the ground solver refine the model. For redundancy elimination we extend the dismatching constraint of an instantiated clause with the constraint of the instantiating substitution.

We have proved refutational completeness of the method by refining the superposition reasoning to ground literal closures and also conflicts to ground closures. If there is a proof of a contradiction from ground literal closures, there is at least one relevant instantiator for a clause extracted from the proof tree such that the conflict on closures becomes irrelevant when the clause instance is added to the clause set. We have lifted this proof from ground literal closures to first-order literals by showing that every proof of a contradiction from literal closures can also be obtained from literals. Moreover, in every lifted proof there is a relevant instantiator that makes the conflict on clauses irrelevant. Adhering to fairness in the sense that every conflict eventually becomes irrelevant guarantees that the ground abstraction of an unsatisfiable clause set eventually becomes unsatisfiable.

We have also shown that a model of a clause set can be generated if the ground abstraction is satisfiable and the set of relevant literal closures is saturated under unit superposition inferences and does not contain a contradiction. We have lifted the notion of Inst-redundancy from literal closures to first-order literals by introducing dismatching constraints. Using the notion of well-constrainedness we have lifted the model generation proof to show that a model of a well-constrained clause set with a satisfiable ground abstraction can be generated if the set of selected constrained literals is saturated under unit superposition inferences and does not contain a contradiction.

The two lifting theorems allow us to reason exclusively with the unit superposition calculus on constrained literals. The formalism of literal closures is a mere tool to prove completeness and not relevant in practice. In the remainder of the thesis we will encounter literal closures only once more in Chapter 6 to justify simplification inferences in the unit superposition calculus.

4.7 Incremental Instantiation

We can view the Inst-Gen-Eq method as consisting of three components we have already discussed: a black boxed ground solver, the Inst-saturation process and the USD-saturation process. The ground solver provides the Inst-saturation process with a model, which a selection function is based on. The USD-saturation process in turn is given selected constrained literals as input and by exhaustively applying inferences of the unit superposition calculus finds proofs of contradictions. From these proofs instantiating substitutions are extracted for clauses in the Inst-saturation process, such that eventually the ground solver can witness unsatisfiability on the ground abstraction.

The combination of the three components to obtain an effective and efficient system needs some consideration, since the naive approach of alternating runs of each component up to saturation will only be successful for the most simple inputs. To build a system for the Inst-Gen-Eq method we exploit the modularity of the method, which allows detaching the three processes of ground solving, Inst-saturation and USD-saturation, if fairness conditions are obeyed and the properties hold in the limit of the processes.

Running a solver for ground satisfiability modulo equality can be an expensive operation, while working with an out of date selection function does not harm soundness or completeness. Hence, it pays off to skip calling the ground solver in most states of the Inst-saturation process and to only obtain an up to date model of the ground abstraction in larger intervals.

The USD-saturation process does not necessarily terminate, since an infinite number of literals can be derived, while it is also possible to find an infinite number of contradictions. It is therefore not possible to compute the USD-saturation in each state of the Inst-saturation process.

The set of clauses in the Inst-saturation process is extended with instances of clauses, guided by conflicts on selected literals. Clauses are removed only if they are redundant, that is, follow from smaller clauses. It is therefore expected that the model of the ground abstraction changes only locally from one state of the Inst-saturation process to the next and hence the selection function can be kept constant on most clauses. This ensues that most of the relevant literals of the clauses in a successor state of the Inst-saturation process have already been relevant in the previous state. Thus, restarting the USD-saturation from the set of relevant literals in every state of the Inst-saturation process means discarding

results from the previous USD-saturation and duplicating most of the work.

In order to reign in the potentially not terminating USD-saturation process, while at the same time reusing conclusions derived in earlier states, our implementation, described in detail in Chapter 7, interleaves the Inst-saturation and the USD-saturation process in the following way. Literals are selected in a fixed number of clauses and propagated to the USD-saturation process. A fixed number of unit superposition inferences are performed and clause instances from proofs of contradictions are propagated to the Inst-saturation process. In turn, the Inst-saturation process selects literals in another batch of clauses and propagates them to the USD-saturation process.

There are two challenges in the interleaving of Inst- and USD-saturation. Since the ground abstraction in a successor state of the Inst-saturation process can require a change in the selection function, some literals lose their relevance and must be removed from the USD-saturation process. Furthermore, adding a clause instance extends the dismatching constraint of the instantiated clause, which is inherited by the selected literal. Consequently, the proofs of some literals in the USD-saturation process become blocked. Both adaptions to a changed selection function, that is, removal of leaf literals or inner literals in a USD-proof, result in conclusions of inferences becoming invalid and having to be removed. In an implementation, we can address the removal in a lazy way by checking the validity of a literal only when necessary.

Although the interleaving of Inst-saturation and USD-saturation complicates an implementation of the Inst-Gen-Eq method, it is worth the effort compared to restarting USD-saturation from scratch in each state of the Inst-saturation process. In the rest of the thesis we consider the Inst-Gen-Eq method in this incremental way with Inst- and USD-saturation interleaved. Hence, we need to develop methods to efficiently adapt a partial USD-saturation to an evolving selection function.

Chapter 5

Labelled Unit Superposition

The previous chapter has introduced the Inst-Gen-Eq method and the unit superposition calculus that serves to find inconsistent literals in the selection and to provide the clause instances that refine the ground abstraction on the conflicts. While the focus of the presentation there was on completeness and effective interleaving of the calculus, we now turn to one of the main issues in an efficient implementation of the Inst-Gen-Eq method.

We have considered how to generate instances of clauses by extracting substitutions from unit superposition proofs. The US-saturation process, where we find those proofs, frequently contains literals which are equal up to renaming. However, as we show by the way of an example, it is necessary for completeness to distinguish between certain literals even though they are equal up to renaming. Having to deal with literal variants separately instead of modulo renaming leads to duplication of efforts in the saturation process and motivates more robust handling of literal variants and more efficient extraction of substitutions in unit superposition.

In this chapter[1] we introduce labels for literals and investigate three versions of the unit superposition calculus, where labels are sets, AND/OR trees or ordered binary decision diagrams (OBDDs). In Section 5.2 we present set labelled unit superposition as the simplest structure, including redundancy elimination and adaption to an evolving selection function. This prompts the development of labels with a Boolean structure and we present tree labelled unit superposition in Section 5.3. Looking for efficient checking for equivalence of labels leads us to OBDD labelled unit superposition discussed in Section 5.4. To conclude the

[1]Parts of this chapter were published as Korovin and Sticksel [2010a].

chapter in Section 5.5 we prove that set and OBDD labelled unit superposition are decision procedures for the Bernays-Schönfinkel fragment of first-order logic modulo equality. The purpose of the first section of this chapter is to motivate our labelling approach.

5.1 Literal Variants and Proofs

Let us begin with an example, where literals equal up to renaming have to be kept distinct.

Example 5.1. Consider the following set of clauses, which is based on problem SEU150+1 from the TPTP benchmark library version 5.1.0, originally a set theoretical problem from the Mizar mathematical library.

$$\underline{f(x,y) \simeq f(y,x)} \tag{1}$$

$$\underline{f(u,v) \not\simeq g(z)} \;\vee\; u \simeq z \tag{2}$$

$$\underline{f(a,b) \simeq g(c)} \tag{3}$$

$$\underline{a \not\simeq b} \tag{4}$$

The clause set is unsatisfiable, which the Inst-Gen-Eq method can prove. There are two sets of inconsistent literals and hence two unit superposition proofs from which instances need to be generated. We choose an ordering $\succ_{\text{gr}}$ such that $f(b,a) \succ_{\text{gr}} f(a,b) \succ_{\text{gr}} g(c) \succ_{\text{gr}} b \succ_{\text{gr}} a$.

The first step in the Inst-Gen-Eq method is to construct the ground abstraction of the clause set (1)-(4) by mapping all variables to the ground term $\bot$. For clauses (1) and (2) we obtain

$$f(\bot,\bot) \simeq f(\bot,\bot) \tag{$1_\bot$}$$

$$f(\bot,\bot) \not\simeq g(\bot) \;\vee\; \bot \simeq \bot, \tag{$2_\bot$}$$

respectively. Clauses (3) and (4) are ground and therefore identical to their ground abstractions.

The ground abstraction of the input clause set consisting of clauses ($1_\bot$), ($2_\bot$), (3) and (4) is satisfiable modulo equality, which in practice is discovered by the ground solver. The solver also returns a model, which is the basis of a selection function for literals in the first-order clause set. In our case we select the first

literal in each clause (1) to (4), which we have underlined above and respectively denote with L_1 to L_4 in the following.

The selected literals L_2 and L_3 in clauses (2) and (3) are inconsistent in first-order logic. One way a contradiction can be derived is as in the following unit superposition proof.

$$\dfrac{\dfrac{\overset{L_3}{f(a,b) \simeq g(c)} \qquad \overset{L_2}{f(u,v) \not\simeq g(z)}}{g(c) \not\simeq g(z)}\,[a/u, b/v]}{\square}\,[c/z] \tag{$*$}$$

Now the ground abstraction has to be refined on the conflict between the selected first-order literals L_2 and L_3. To this end we add to the clause set the relevant instances of clauses (2) and (3), where L_2 and L_3 are respectively selected in.

In order to obtain the relevant instances for the clauses, we trace the branches of the proof tree and compose the substitutions along the paths from the root to the two leaf literals. For this proof the resulting substitution for both branches is $\sigma_1 = [a/u, b/v]\,[c/z] = [a/u, b/v, c/z]$.

We now apply the substitution σ_1 to clauses (2) and (3). Since clause (3) is ground, instantiation with σ_1 has no effect. Instantiating clause (2) with σ_1 yields

$$f(a,b) \not\simeq g(c) \;\vee\; a \simeq c, \tag{5}$$

which is added to the input clause set.

There is another conflict between the selected literals L_1, L_2 and L_3, which can be found by deriving a contradiction in the following proof.

$$\dfrac{\overset{L_3}{f(a,b) \simeq g(c)} \qquad \dfrac{\overset{L_1}{f(x,y) \simeq f(y,x)} \qquad \overset{L_2}{f(u,v) \not\simeq g(z)}}{f(v,u) \not\simeq g(z)}\,[u/x, v/y]}{\dfrac{g(c) \not\simeq g(z)}{\square}\,[c/z]}\,[b/u, a/v] \tag{$\dagger$}$$

The rightmost branch of the proof tree contains the literals $f(u,v) \not\simeq g(z)$ and $f(v,u) \not\simeq g(z)$, which are equal up to renaming. The latter literal is inferred from the former with the commutativity axiom $f(x,y) \simeq f(y,x)$. Omitting this inference we obtain proof $(*)$ above. Hence one could regard proof $(*)$ as a

subproof of (†). In proof (†) the inference steps from the second literal variant $f(v,u) \not\simeq g(z)$ and L_3 to the contradiction are identical to the inference steps in the previous proof (∗) in the sense that the premises are equal up to renaming and the inferences are on the same positions in the literals with the same substitutions.

However, the sets of relevant instances extracted from the two proofs are different and in fact both are required. The substitutions extracted from proof (†) are

$$\sigma_{21} = [b/u, a/v]\,[c/z] = [b/u, a/v, c/z]$$

for L_3 and

$$\sigma_{22} = [u/x, v/y]\,[b/u, a/v]\,[c/z] = [b/x, a/y, b/u, a/v, c/z]$$

for L_1 and L_2. We apply these substitutions to the respective clauses: instantiating ground clause (3) with σ_{21} results in the same clause while instantiating clauses (1) and (2) with σ_{22} leads to these two new clauses to be added to the input clause set.

$$f(b,a) \simeq f(a,b) \tag{6}$$
$$f(b,a) \not\simeq g(c) \;\vee\; b \simeq c. \tag{7}$$

The ground abstractions of the new ground clause instances (5)-(7) from the two proofs are the clauses themselves as there are no variables to be mapped to ⊥. Altogether we have the following ground abstraction of the input clauses and their instances.

$$f(\bot,\bot) \simeq f(\bot,\bot) \tag{$1_\bot$}$$
$$f(\bot,\bot) \not\simeq g(\bot) \;\vee\; \bot \simeq \bot \tag{$2_\bot$}$$
$$f(a,b) \simeq g(c) \tag{$3_\bot$}$$
$$a \not\simeq b \tag{$4_\bot$}$$
$$f(a,b) \not\simeq g(c) \;\vee\; a \simeq c \tag{$5_\bot$}$$
$$f(b,a) \simeq f(a,b) \tag{$6_\bot$}$$
$$f(b,a) \not\simeq g(c) \;\vee\; b \simeq c \tag{$7_\bot$}$$

The ground solver finds clauses ($3_\bot$)-($7_\bot$) to be ground unsatisfiable modulo equality: from ($3_\bot$) and ($5_\bot$) we can infer the equation $a \simeq c$, from ($3_\bot$), ($6_\bot$) and ($7_\bot$) the equation $b \simeq c$ follows and $a \simeq c$ and $b \simeq c$ together contradict the

unit clause $(4_\bot)$ $a \not\simeq b$.

If we omitted one of the proofs, the ground abstraction would not contain $(5_\bot)$ from proof $(*)$ or $(6_\bot)$ and $(7_\bot)$ from proof $(\dagger)$, respectively. Hence, either $a \simeq c$ or $b \simeq c$ would not be consequences in the ground abstraction and unsatisfiability could not be concluded.

Since $a \succ_{\text{gr}} b$ and $f(b,a) \succ_{\text{gr}} f(a,b)$, the side conditions of the inference rules prevent any other inference than those in proofs $(*)$ and $(\dagger)$. It is therefore essential to ensure that both proofs are found, which in turn requires considering $f(u,v) \not\simeq g(z)$ and its variant $f(v,u) \not\simeq g(z)$ as being different.

However, then the commutativity axiom $f(x,y) \simeq f(y,x)$ in clause (1) can produce an unbounded number of variants of $f(u,v) \not\simeq g(z)$. Since we assume the premises of a unit superposition inference to be variable disjoint, we can generate ever more variants of $f(u,v) \not\simeq g(z)$ from clause (2) in the following way: the conclusion of a unit superposition inference between $f(x,y) \simeq f(y,x)$ and $f(u,v) \not\simeq g(z)$ is $f(v,u) \not\simeq g(z)$, a variant of $f(u,v) \not\simeq g(z)$ to which we can apply another unit superposition inference with the literal $f(x,y) \simeq f(y,x)$ from the commutativity clause to obtain yet another new variant. One such variant suffices for completeness in this example, while in general any number of variants may be required.

From the example we learn that in the unit superposition calculus literals must be treated separately even if they are equal up to renaming. A practical implementation of the unit superposition calculus has to distinguish between literal variants in order to allow more than one variant of a literal on the same branch of a proof tree. However, a simple implementation along the lines suggested by the example would be inefficient for two reasons:

(i) no upper bound for the number of variants of one literal can be given and

(ii) reasoning with each literal variant separately creates significant duplication.

An inference step applicable to one literal variant can also be applied to every other literal variant resulting in one variant of the conclusion each. Since Inst-Gen-Eq must saturate the set of literals under unit superposition inferences, finding all inconsistencies, the amount of duplication of inferences and the size of the set of literals in the saturation process make this approach intractable even in cases as small as in the example.

Literal Variants and Cyclic Proofs

Our main approach is based on labelling of literals and described in the next section. In the rest of this section we discuss an alternative approach without labels that we did not follow. We believe it offers insights about the obstacles dealing with literal variants and motivates our labelling approach. Readers may safely skip to the next Section 5.2 and the rest of the chapter, where we present unit superposition calculi on labelled literals, starting with set labels and continuing with tree labels and OBDD labels.

One possible approach to literal variants that avoids duplication in the saturation process is to treat as identical all literals equal up to renaming. However, this comes at the cost of a more complicated structure of proofs, which are then not trees in every case. If in Example 5.1 on page 90 we do not distinguish between the literals $f(u,v) \not\simeq g(z)$ and $f(v,u) \not\simeq g(z)$, the two nodes of these literals in the proof tree (†) collapse into one and we obtain a cyclic graph. The edge from the right premise to the conclusion is a cycle on $f(u,v) \not\simeq g(z)$. Now extraction of substitutions is complicated by having to consider this cycle an unbounded number of times in order to generate all instances necessary for completeness as in the previous example. Techniques like iterative deepening have to be invoked in order to preserve fairness of the US-saturation process.

We notice that if the composed substitution between two literal variants on a branch is proper, the cycle can be unfolded by eagerly generating instances, thus eliminating some proofs with cycles. The following example demonstrates this idea. Unfortunately, eager instantiation does not work if the substitution is non-proper.

Example 5.2. The following unsatisfiable set of clauses comes from problem SWV268-2 from the TPTP benchmark library v5.1.0, modelling properties of a cryptographic protocol.

$$\underline{f(f(u)) \simeq f(u)} \tag{1}$$

$$\underline{g(f(x), f(y)) \simeq h(z)} \;\vee\; g(x,y) \not\simeq h(c) \tag{2}$$

$$\underline{g(f(a), f(b)) \not\simeq h(w)} \tag{3}$$

$$\underline{g(f(a), b) \simeq h(c)} \tag{4}$$

The ground abstraction of the clauses is satisfiable and a possible selection

consists of the first literal in each clause, which we have underlined and denote as L_1 to L_4. We derive a variant of literal L_2 from clause (2) in the following way.

$$\dfrac{\overset{L_1}{f(f(u)) \simeq f(u)} \qquad \overset{L_2}{g(f(x), f(y)) \simeq h(z)}}{g(f(u), f(y)) \simeq h(z)}\,[f(u)/x] \tag{$*$}$$

Since the conclusion $g(f(u), f(y)) \simeq h(z)$ is a variant of the leaf literal L_2 and the extracted substitution $\sigma_1 = [f(u)/x]$ is proper, we can discard this proof if we instantiate the clauses at the leaves.

Applying σ_1 to clause (1) does not change the clause, the instance of clause (2) with σ_1 is

$$\underline{g(f(f(x')), f(y')) \simeq h(z')} \;\vee\; g(f(x'), y') \not\simeq h(c), \tag{5}$$

where we use the fresh variables x', y' and z' to keep the clause variable disjoint from (1)-(4). The ground abstraction of clauses (1)-(5) is satisfiable and we can select the first literals in each clause.

We derive a contradiction from the inconsistent literals L_1, L_3 and L_5 in clauses (1), (3) and (5), respectively, as follows.

$$\dfrac{\dfrac{\overset{L_1}{f(f(u)) \simeq f(u)} \qquad \overset{L_5}{g(f(f(x')), f(y')) \simeq h(z')}}{g(f(x'), f(y')) \simeq h(z')}\,[x'/u] \qquad \overset{L_3}{g(f(a), f(b)) \not\simeq h(w)}}{\dfrac{h(z') \not\simeq h(w)}{\square}\,[w/z']}\,[a/x', b/y'] \tag{$\dagger$}$$

The first inference is between L_1 and L_5 from the instantiated clause (5), the conclusion $g\,(f(x'), f(y')) \simeq h(z')$ is not a variant of the premise L_5 and the substitution $[x'/u]$ in the inference is non-proper . The whole proof tree does not contain variants of any literal on the same branch.

We extract the substitutions

$$\sigma_{21} = [x'/u]\,[a/x', b/y']\,[w/z'] = [a/x', b/y', w/z', a/u]$$

and

$$\sigma_{22} = [a/x', b/y']\,[w/z'] = [a/x', b/y', w/z']$$

for the left and the right branches, respectively. Instantiating clause (3) with σ_{22} results in the same clause, the instances of clauses (1) and (5) with σ_{21} are

$$f(f(a)) \simeq f(a) \tag{6}$$
$$g(f(f(a)), f(b)) \simeq h(w) \ \vee \ g(f(a), b) \not\simeq h(c). \tag{7}$$

In proof (†) the last two inference steps on $g\,(f(x'), f(y')) \simeq h(z')$ could have been applied to its variant $g\,(f(u), f(y)) \simeq h(z)$ in order to continue the abandoned proof (∗) to the contradiction, proving inconsistency of the selected literals L_1, L_2 and L_3. Applying the relevant instantiators from this proof to the respective clauses (1), (2) and (3), we would obtain exactly the relevant instances (6) and (7) we have from proof (†).

Now the ground abstraction of clauses (1)-(7) is as follows.

$$f(f(\bot)) \simeq f(\bot) \tag{$1_\bot$}$$
$$g(f(\bot), f(\bot)) \simeq h(\bot) \ \vee \ g(\bot, \bot) \not\simeq h(c) \tag{$2_\bot$}$$
$$g(f(a), f(b)) \not\simeq h(\bot) \tag{$3_\bot$}$$
$$g(f(a), b) \simeq h(c) \tag{$4_\bot$}$$
$$g(f(f(\bot)), f(\bot)) \simeq h(\bot) \ \vee \ g(f(\bot), \bot) \not\simeq h(c) \tag{$5_\bot$}$$
$$f(f(a)) \simeq f(a) \tag{$6_\bot$}$$
$$g(f(f(a)), f(b)) \simeq h(\bot) \ \vee \ g(f(a), b) \not\simeq h(c) \tag{$7_\bot$}$$

The ground abstraction is unsatisfiable modulo equality: due to the unit clause ($4_\bot$) the second literal of ($7_\bot$) must be false. However, the first literal of ($7_\bot$) and $f\,(f(a)) \simeq f(a)$ from ($6_\bot$) contradict $g(f(a), f(b)) \not\simeq h(\bot)$ from unit clause ($3_\bot$). Therefore, no interpretation can satisfy all clauses ($3_\bot$), ($4_\bot$), ($6_\bot$) and ($7_\bot$). The ground abstraction of the instantiated clause ($5_\bot$) from the abandoned proof is not necessary to show ground unsatisfiability.

In this example we derive a variant of the literal $g\,(f(x), f(y)) \simeq h(z)$ with a proper substitution $\sigma_1 = [f(u)/x]$. We abandon the proof which would have led to a contradiction, instantiate the clauses at the leaves and find another proof of the contradiction, which does not contain multiple variants of $g\,(f(x), f(y)) \simeq h(z)$ and has the same relevant instances as the abandoned proof.

Eager instantiation from cycles as shown in the example is applicable as soon as the literal at the root of a proof is a variant of a literal in a branch and only

if the composed substitution between the literal variants is proper. Instantiating the clauses at the leaves of the proof with the extracted substitutions makes every continuation of the proof redundant as for every such proof there is a proof without the cycle, using a literal from an instantiated clause that leads to the same conclusion.

As mentioned before, the instantiation approach only eliminates cycles with proper substitutions. Let us consider the first inference in proof (†) from Example 5.1 on page 90.

$$\frac{\overset{L_1}{f(x,y) \simeq f(y,x)} \qquad \overset{L_2}{f(u,v) \not\simeq g(z)}}{f(v,u) \not\simeq g(z)}\,[u/x, v/y]$$

The conclusion of the inference is a variant of the right premise with the non-proper substitution $\sigma = [u/x, v/y]$. Applying σ to clauses (1) and (2) only creates variants of these clauses. Then variants of L_1 and L_2 are selected in the new clauses, the same proof as above is obtained and we still have a variant of the premise $f(u,v) \not\simeq g(z)$ as the conclusion.

We have not pursued the sketched approach further from here. Although it seems attractive to consider only one representative of all literals equal up to renaming in the saturation process, the downside of losing the tree structure of proofs through the introduction of cycles leaves the benefits in terms of efficiency unclear. A significant amount of bookkeeping about cycles with non-proper substitutions is required and how to integrate instantiations from cycles with proper substitutions is not immediately obvious.

The result would be a rather ad hoc approach in contrast to the robust approach we are presenting in the rest of this chapter. Labelling literals and explicit merging of variants allow to uniformly integrate literal variants into the saturation process while directly addressing the problem of duplication.

5.2 Set Labelled Unit Superposition

In order to tackle the problem of literal variants in proof trees as presented in the examples of the previous section, we have chosen a robust approach that combines multiple literal variants, avoids duplication of inferences and preserves the tree structure of proofs. The idea of our approach is to attach labels to

literals, accumulating composed substitutions from inferences in the label. The labels allow us to treat literal variants initially as distinct. Merging of literal variants is done in an explicit step with a new calculus rule that keeps track of the literal variants merged in the label. After merging two literal variants, inferences are drawn simultaneously on all literal variants merged and instead of tracing proof trees, relevant instances are extracted from the label.

The Inst-Gen-Eq method with labelled unit superposition proceeds in the same way as in the unlabelled case. We saturate a set of literals under inferences and add relevant instances to the clause set when finding a contradiction. While unlabelled unit superposition starts with the set of selected literals, we now start with the set of initially labelled selected literals.

The labels in our labelled superposition calculi contain closures $C \cdot \theta$, which are not necessarily ground, such that C is the clause a leaf literal is selected in and the substitution θ is the relevant instantiator for C. Let us introduce set labels as the first and simplest label structure.

Definition 5.1. A *set label* $\mathcal{L}$ is a set of closures $\{C_1 \cdot \theta_1, \ldots, C_n \cdot \theta_n\}$ where the closures are not necessarily ground. A *set labelled literal* $\mathcal{L}\colon L$ is a pair of a set label $\mathcal{L}$ and a literal L where $\mathcal{L}$ and L are variable disjoint, such that for each $C_i \cdot \theta_i \in \mathcal{L}$ we have $\operatorname{var}(C_i) \cap \operatorname{var}(L) = \emptyset$.

Let sel be a selection function, C be a clause and $C \cdot \theta$ be an arbitrary but fixed closure, such that C and $C\theta$ are equal up to renaming. The *initial labelling* of the selected literal L in clause C with respect to the selection function sel is the set labelled literal $\{C \cdot \theta\} : \operatorname{sel}(C)\,\theta$.

Let $\mathcal{L} = \{C_1 \cdot \theta_1, \ldots, C_n \cdot \theta_n\}$ be a set label and σ a substitution, which is variable disjoint from $\mathcal{L}$ in the following way: $\operatorname{var}(\operatorname{rng}(\sigma)) \cap \operatorname{var}(\{C_1, \ldots, C_n\}) = \emptyset$. The *$\sigma$-instance* of the set label $\mathcal{L}$ is $\mathcal{L}\sigma = \{C_1 \cdot \theta_1\sigma, \ldots, C_n \cdot \theta_n\sigma\}$.

We say two set labelled literals $\mathcal{L}\colon L$ and $\mathcal{L}'\colon L'$ are *equal up to renaming* if there is a renaming ρ such that $L = L'\rho$ and for each closure in $\mathcal{L}$ there is a closure equal up to renaming in $\mathcal{L}'\rho$ and vice versa. We usually do not distinguish between labelled literals equal up to renaming.

In a non-ground closure $C \cdot \theta$ (see Definition 2.7 on page 26) the clauses C and $C\theta$ are variable disjoint. An initially labelled selected literal $\{C \cdot \theta\} : \operatorname{sel}(C)\,\theta$ satisfies the variable disjointness property, since $\operatorname{var}(C) \cap \operatorname{var}(\operatorname{sel}(C)\,\theta) = \emptyset$. The σ-instance of a set labelled literal preserves variable disjointness since the condition $\operatorname{var}(\operatorname{rng}(\sigma)) \cap \operatorname{var}(\{C_1, \ldots, C_n\}) = \emptyset$ has to be observed.

An initial labelling $\{C \cdot \theta\} : \operatorname{sel}(C)\,\theta$ also has the property to bind the variables in the selected literal in the closure $C \cdot \theta$ in the label to the variables in the labelled literal: $\operatorname{sel}(C)\,\theta \in C\theta$. Instantiating the label and the literal with the same substitution preserves this binding.

The inference rules of set labelled unit superposition in the following definition are extensions to set labelled literals of the previously used unlabelled inference rules (see Definition 3.3 on page 37) and include a new merging rule.

Definition 5.2 (Set Labelled Unit Superposition)**.** The respective conclusions of superposition and equality resolution are new labelled literals. The conclusion of merging replaces the two premises.

Unit Superposition

$$\frac{\mathcal{L} \colon l \simeq r \qquad \mathcal{L}' \colon s[l'] \simeq t}{\mathcal{L}\sigma \cup \mathcal{L}'\sigma \colon (s[r] \simeq t)\,\sigma}\,(\sigma) \qquad\qquad \frac{\mathcal{L} \colon l \simeq r \qquad \mathcal{L}' \colon s[l'] \not\simeq t}{\mathcal{L}\sigma \cup \mathcal{L}'\sigma \colon (s[r] \not\simeq t)\,\sigma}\,(\sigma)$$

where for some grounding substitution θ with $\operatorname{dom}(\theta) = \operatorname{var}(\{l\sigma, r\sigma, s[l']\sigma, t\sigma\})$

(i) $\sigma = \operatorname{mgu}(l, l')$,

(ii) l' is not a variable,

(iii) $l\sigma\theta \succ_{\mathrm{gr}} r\sigma\theta$,

(iv) $s[l']\sigma\theta \succ_{\mathrm{gr}} t\sigma\theta$,

(v) $\operatorname{var}(\{l, r\}) \cap \operatorname{var}(\{s[l'], t\}) = \emptyset$.

and there is at least one closure $C \cdot \rho \in \mathcal{L}$ and one closure $C' \cdot \rho' \in \mathcal{L}'$ such that $\rho\sigma$ satisfies the dismatching constraint of $C \mid D \in S$ and $\rho'\sigma$ satisfies the dismatching constraint of $C' \mid D' \in S$.

Merging (Replacement rule for both premises)

$$\frac{\mathcal{L} \colon l \simeq r \qquad \mathcal{L}' \colon l' \simeq r'}{\mathcal{L} \cup \mathcal{L}'\sigma \colon l \simeq r}\,(\sigma) \qquad\qquad \frac{\mathcal{L} \colon l \not\simeq r \qquad \mathcal{L}' \colon l' \not\simeq r'}{\mathcal{L} \cup \mathcal{L}'\sigma \colon l \not\simeq r}\,(\sigma)$$

where σ is a renaming such that $l'\sigma = l$ and $r'\sigma = r$.

Equality Resolution

$$\frac{\mathcal{L} \colon l \not\simeq r}{\mathcal{L}\sigma \colon \square}\,(\sigma)$$

where $\sigma = \operatorname{mgu}(l, r)$ and there is at least one closure $C \cdot \rho \in \mathcal{L}$ such that $\rho\sigma$ satisfies the dismatching constraint of $C \mid D \in S$.

Labels do not occur in side conditions, we can therefore view labelling as an elegant way of bookkeeping about clauses where the leaves of the proofs are selected in and their relevant instantiators in proofs. Since the substitutions from inferences are applied to closures in the label and the label of a conclusion is the union of the labels of the premises, we obtain the relevant instances from a proof of a literal directly from its label. Hence extraction of substitutions from proof trees as in the unlabelled calculus is obsolete.

Definition 5.3. The *relevant instantiators* and the *relevant instances* of the set labelled literal $\{C_1 \cdot \theta_1, \ldots, C_n \cdot \theta_n\} : L$ are the substitutions $\theta_1, \ldots, \theta_n$ for the clauses $C_1, \ldots, C_n$, respectively, and the set $\{C_1\theta_1, \ldots, C_n\theta_n\}$.

Let us resume the motivating example from the beginning of the chapter, demonstrating the inference rules of set labelled unit superposition.

Example 5.3. We consider the clause set from Example 5.1 on page 90

$$\underline{f(x,y) \simeq f(y,x)} \tag{1}$$

$$\underline{f(u,v) \not\simeq g(z)} \ \vee \ u \simeq z \tag{2}$$

$$\underline{f(a,b) \simeq g(c)} \tag{3}$$

$$\underline{a \not\simeq b} \tag{4}$$

with the same literals selected.

In this example we implicitly view variables in clauses and labelled literals as disjoint. The initially labelled selected literals of clauses (1) and (2) are

$$\mathcal{L}_1 \colon L_1 = \{(1) \cdot []\} : f(x,y) \simeq f(y,x)$$

and

$$\mathcal{L}_2 \colon L_2 = \{(2) \cdot []\} : f(u,v) \not\simeq g(z).$$

Between these two labelled literals we draw the following unit superposition inference.

$$\frac{\begin{array}{cc} \mathcal{L}_1 \colon L_1 & \mathcal{L}_2 \colon L_2 \\ \{(1) \cdot []\} \colon f(x,y) \simeq f(y,x) & \{(2) \cdot []\} \colon f(u,v) \not\simeq g(z) \end{array}}{\{(1) \cdot [u/x, v/y], (2) \cdot []\} \colon f(v,u) \not\simeq g(z)} \, [u/x, v/y] \tag{$\dagger'$}$$

This step corresponds to the first inference in proof (†) in Example 5.1. In

order to preserve variable disjointness between a literal and the clauses in its label, without loss of generality we restrict the domain of the substitution $\sigma_1 = [u/x, v/y]$ from the inference to the variables occurring in the literals. When applying σ_1 to the label of L_2, we therefore obtain the empty substitution $[]$.

The literal $f(v, u) \not\simeq g(z)$ in the conclusion is a variant of the literal in the right premise $f(u, v) \not\simeq g(z)$. We use the merging rule to combine the two literal variants into one.

$$\frac{\{(2) \cdot []\} \colon f(u, v) \not\simeq g(z) \qquad \{(1) \cdot [u/x, v/y], (2) \cdot []\} \colon f(v, u) \not\simeq g(z)}{\{(2) \cdot [], (1) \cdot [v/x, u/y], (2) \cdot [v/u, u/v]\} \colon f(u, v) \not\simeq g(z)}\,[v/u, u/v] \quad (\ddagger)$$

We apply the renaming substitution $[v/u, u/v]$ to the label of the right premise only and take the union with the left premise, obtaining a set of three closures. The labelled literal

$$\{(2) \cdot [], (1) \cdot [v/x, u/y], (2) \cdot [v/u, u/v]\} : f(u, v) \not\simeq g(z)$$

in the conclusion supersedes the two literal variants that were merged and we only consider this literal from now on.

With a superposition inference between $\mathcal{L}_3 \colon L_3$, the initially labelled selected literal in clause (3), and the merged literal and subsequent equality resolution, which are the common steps in both proofs $(*)$ and $(\dagger)$ in Example 5.1, we derive a contradiction.

$$\frac{\{(3) \cdot []\} \colon f(a, b) \simeq g(c) \quad \{(2) \cdot [], (1) \cdot [v/x, u/y], (2) \cdot [v/u, u/v]\} \colon f(u, v) \not\simeq g(z)}{\dfrac{\{(3) \cdot [], (2) \cdot [a/u, b/v], (1) \cdot [b/x, a/y], (2) \cdot [b/u, a/v]\} \colon g(c) \not\simeq g(z)}{\{(3) \cdot [], (2) \cdot [a/u, b/v, c/z], (1) \cdot [b/x, a/y], (2) \cdot [b/u, a/v, c/z]\} \colon \square}\,[c/z]}\,[a/u, b/v]$$

Before applying the substitution from an inference to the label of a literal, without loss of generality, we restrict the domain of the substitution to the set of variables of the literal in order to satisfy variable disjointness of a literal and the clauses in its closures.

Because of the merging step, the label of the empty clause $\square$ contains the relevant instances from both proofs $(*)$ and $(\dagger)$ in Example 5.1. Clause (3) is ground, thus the relevant instance represented by the closure $(3) \cdot []$ is (3). The

relevant instances represented by the remaining closures in the label are

$$f(a,b) \not\simeq g(c) \ \vee \ a \simeq c \tag{5}$$

$$f(b,a) \simeq f(a,b) \tag{6}$$

$$f(b,a) \not\simeq g(c) \ \vee \ b \simeq c, \tag{7}$$

which are exactly instances (5)-(7) in Example 5.1. Ground unsatisfiability is shown on the ground abstraction of clauses (1)-(7) in the same way as there.

Set labelled unit superposition avoids duplication of inferences when the merging rule is applied. The common parts of proofs (∗) and (†) in Example 5.1 are factored out and instead of two separate proofs on the literal variants $f(u,v) \not\simeq g(z)$ and $f(v,u) \not\simeq g(z)$, set labelled unit superposition needs only one proof on a labelled literal, which has both variants merged.

As the example illustrates, set labels make available the relevant instances directly, thus obsoleting the need to trace a proof tree. Note that the proof tree of a literal cannot be reconstructed from its label as the label only contains the relevant instances and no information about the proof structure. Neither can the information be recovered, which literal variants have been merged. However, the relevant instances are all that is needed in the instantiation process of Inst-Gen-Eq.

The merging rule eliminates the literal variant in the right premise by combining it with the literal variant in the left premise. There are no two literals that are equal up to renaming if the merging rule is applied exhaustively. Hence merging reduces the size of the set of literals in the US-saturation process.

Example 5.4. Let us look at Example 5.3 again and show how labelled unit superposition deals with literal variants. Consider the labelled literal from the commutativity axiom in clause (1)

$$\mathcal{L}_1 \colon L_1 = \{(1) \cdot []\} \colon f(x,y) \simeq f(y,x)$$

and the conclusion of the merging inference (‡)

$$\mathcal{L}_2 \colon L_2 = \{(2) \cdot [], (1) \cdot [v/x, u/y], (2) \cdot [v/u, u/v]\} \colon f(u,v) \not\simeq g(z).$$

As discussed in Example 5.1, an unbounded number of variants can be generated from the unlabelled literals $f(x,y) \simeq f(y,x)$ and $f(u,v) \not\simeq g(z)$ in the

unlabelled calculus by repeated superposition with $f(x, y) \simeq f(y, x)$.

Here we draw a superposition inference between the labelled literals $\mathcal{L}_1 \colon L_1$ and $\mathcal{L}_2 \colon L_2$ as in $(\dagger')$ in Example 5.3 and obtain the labelled literal

$$\{(1) \cdot [u/x, v/y], (2) \cdot [], (1) \cdot [v/x, u/y], (2) \cdot [v/u, u/v]\} : f(v, u) \not\simeq g(z).$$

We merge it with $\mathcal{L}_2 \colon L_2$ to get

$$\mathcal{L}_2' \colon L_2 = \{(2) \cdot [], (1) \cdot [v/x, u/y], (2) \cdot [v/u, u/v], (1) \cdot [u/x, v/y]\} : f(u, v) \not\simeq g(z).$$

The label $\mathcal{L}_2'$ replaces $\mathcal{L}_2$ on L_2, where the only difference is the addition of the closure $(1) \cdot [u/x, v/y]$. A further superposition inference between $\mathcal{L}_1 \colon L_1$ and $\mathcal{L}_2' \colon L_2$ results in

$$\{(2) \cdot [], (1) \cdot [v/x, u/y], (2) \cdot [v/u, u/v], (1) \cdot [u/x, v/y]\} : f(v, u) \not\simeq g(z).$$

This is equal up to renaming with the substitution $[v/u, u/v]$ to $\mathcal{L}_2' \colon L_2$ and is therefore discarded.

The set labelled commutativity axiom $\{(1) \cdot []\} : f(x, y) \simeq f(y, x)$ can only generate a finite number of labelled literals, which are not equal up to renaming, from the initial labelling $\mathcal{L}_2 \colon L_2$. As soon as we have derived $\mathcal{L}_2' \colon L_2$, a further inference with $f(x, y) \simeq f(y, x)$ results in a literal which is equal up to renaming to $\mathcal{L}_2' \colon L_2$ and does not add new relevant instances to the label.

Another advantage of merging literal variants is that inferences with a labelled literal are simultaneously applied to all literal variants merged in the label. In this way set labelled unit superposition eliminates duplication in an unlabelled calculus where each literal variant would have to be treated separately.

Completeness

We show completeness of an Inst-saturation process with set labelled unit superposition, where clause instances are obtained from the set label of a contradiction, by lifting the completeness of unit superposition with dismatching constraints from Section 4.6.

We first define a correspondence between set labelled literals and constrained literals.

Definition 5.4. Let $C_1 \mid D_1, \ldots, C_n \mid D_n$ be constrained clauses in S and $\mathcal{L}\colon L$ be a set labelled literal with $\mathcal{L} = \{C_1 \cdot \sigma_1, \ldots, C_n \cdot \sigma_n\}$. The *set of constrained literals* $\mathrm{DLit}\,(\mathcal{L}\colon L)$ of the set labelled literal is

$$\mathrm{DLit}\,(\mathcal{L}\colon L) = \{L \mid D_1\sigma_1, \ldots, L \mid D_k\sigma_n\}\,.$$

We extend DLit from set labelled literals to sets of set labelled literals as

$$\mathrm{DLit}\,(\mathcal{L}L) = \bigcup_{\mathcal{L}\colon\, L \in \mathcal{L}L} \mathrm{DLit}\,(\mathcal{L}\colon L)\,.$$

As in the proof of completeness of unit superposition with dismatching constraints in Section 4.6, we first define redundancy on set labelled literals and subsequently lift the definitions of a saturation process to set labelled literals.

Definition 5.5. Let $\mathcal{L}L$ be a set of labelled literals. A labelled literal $\mathcal{L}\colon L \in \mathcal{L}L$ is *USL-redundant* in $\mathcal{L}L$ if

(i) every constrained literal in $\mathrm{DLit}\,(\mathcal{L}\colon L)$ is USD-redundant in $\mathrm{DLit}\,(\mathcal{L}L)$ or

(ii) there is an $\mathcal{L}'\colon L' \in \mathcal{L}L$ such that L and L' are equal up to the renaming σ and there is a renaming ρ such that $\mathcal{L}'\sigma\rho \supsetneq \mathcal{L}$. In other words, there is a set labelled literal $\mathcal{L}'\sigma\colon L'$, which is equal up to renaming to $\mathcal{L}\colon L$ and the set label $\mathcal{L}'\sigma\rho$ is a superset of or equal to $\mathcal{L}$.

Let $\mathcal{R}_{\mathrm{USL}}\,(\mathcal{L}L)$ denote the set of all set labelled literals in $\mathcal{L}L$, which are USL-redundant in $\mathcal{L}L$.

We note that the above definition implies a natural property: the set labelled literal $\emptyset\colon L$ is USL-redundant in any non-empty set of set labelled literals. It is important to define USL-redundancy with strict set inclusion $\supsetneq$, since otherwise we would have the undesirable effect that a set labelled literal would make itself redundant.

However, in contrast to previous definitions of redundancy (US-redundancy in Definition 4.3 on page 48 and USD-redundancy in Definition 4.27 on page 81) the above notion of redundancy is not well-founded in general. It is not only based on US-redundancy, where a literal closure is redundant if it follows from smaller

literal closures in the well-founded ordering $\succ_1$, but in condition (ii) there is the set inclusion $\supsetneq$, which is not necessarily well-founded in a possibly infinte set of labelled literals $\mathcal{L}L$. Nevertheless, USL-redundancy with set inclusion is a useful concept and in particular it is sufficient to justify eliminating the two premises of the merging inference rule.

Theorem 5.1. *The merging inference rule is a simplification inference, that is the conclusion makes both of its premises USL-redundant.*

Proof. Let $\mathcal{L}\colon L$ and $\mathcal{L}'\colon L'$ be the premises of a merging inference rule. Without loss of generality we assume $\mathcal{L} \neq \emptyset$ and $\mathcal{L}' \neq \emptyset$ in the premises. Otherwise, either $\mathcal{L} = \mathcal{L}' = \emptyset$ and the conclusion is USL-redundant, or $\mathcal{L} \neq \emptyset$ and $\mathcal{L}' = \emptyset$ and the conclusion $\mathcal{L} \cup \mathcal{L}'\sigma\colon L = \mathcal{L}\colon L$ is identical to the premise.

We have L and L' equal up to the renaming σ, that is $L\sigma = L'$ and $L'\sigma^{-1} = L$, and also $\mathcal{L} \cup \mathcal{L}'\sigma \supsetneq \mathcal{L}$ and $(\mathcal{L} \cup \mathcal{L}'\sigma)\,\sigma^{-1} = (\mathcal{L}\sigma^{-1} \cup \mathcal{L}') \supsetneq \mathcal{L}'$. By condition (ii) in Definition 5.5 the conclusion $\mathcal{L} \cup \mathcal{L}'\sigma\colon L$ makes both $\mathcal{L}\colon L$ and $\mathcal{L}'\colon L'$ USL-redundant. □

Hence, the effect of the merging inference rule is covered by USL-redundancy and in the following saturation process we only need to pay attention to the unit superposition rule and the equality resolution rule in the set labelled unit superposition calculus.

Definition 5.6. A *USL-saturation process* on set labelled literals is an infinite sequence of pairs $\{\langle \mathcal{N}L^i, \mathcal{D}L^i \rangle\}_{i=1}^{\infty}$, where $\mathcal{N}L^i$ is a set of set labelled literals and $\mathcal{D}L^i$ is a disjoint set of positive set labelled literals. Each pair $\langle \mathcal{N}L^{i+1}, \mathcal{D}L^{i+1} \rangle$ with $i > 1$, called a *successor state*, is obtained from the previous state $\langle \mathcal{N}L^i, \mathcal{D}L^i \rangle$ by either

(i) adding to $\mathcal{N}L^i$ the conclusion of a set labelled unit superposition inference with one premise in $\mathcal{N}L^i$ and the second premise in the union $\mathcal{N}L^i \cup \mathcal{D}L^i$, adding to $\mathcal{N}L^i$ the conclusion of an equality resolution inference with the premise in $\mathcal{N}L^i$, adding to $\mathcal{N}L^i$ the conclusion of a set labelled merging inference with both premises in $\mathcal{N}L^i$,

(ii) adding to $\mathcal{D}L^i$ the conclusion of a set labelled unit superposition inference with both premises in $\mathcal{D}L^i$, adding to $\mathcal{D}L^i$ the conclusion of an equality resolution inference with the premise in $\mathcal{N}L^i$, adding to $\mathcal{D}L^i$ the conclusion of a set labelled merging inference with both premises in $\mathcal{D}L^i$,

(iii) removing a set labelled literal from $\mathcal{N}L^i$ which is USL-redundant in the union $\mathcal{N}L^i \cup \mathcal{D}L^i$ or

(iv) removing a literal from $\mathcal{D}L^i$ which is USL-redundant in $\mathcal{D}L^i$.

Since USL-redundancy is not well-founded, it is difficult to define the notion of persistent literals in the USL-saturation process and also USL-saturation. Thus, we omit these two definitions and give a definition of fairness not based on persistent literals, which still suffices to prove completeness of the calculus. The following definition of fairness is essentially identical to fairness in the US- and USD-saturation process, but it avoids the notion of persistent literals.

Definition 5.7. A USL-saturation process is *fair* if there is a k for every i such that

(i) for every possible unit superposition inference with with one premise in $\mathcal{N}L^i$ and the second premise in $\mathcal{N}L^i \cup \mathcal{D}L^i$, for every equality resolution inference with the premise in $\mathcal{N}L^i$ and for every merging inference with both premises in $\mathcal{N}L^i$ the respective conclusion is in $\mathcal{N}L^{i+k}$ or redundant in $\mathcal{N}L^{i+k} \cup \mathcal{D}L^{i+k}$ and

(ii) for every possible unit superposition inference with both premises in $\mathcal{D}L^i$ and for every equality resolution inference with the premise in $\mathcal{D}L^i$ the respective conclusion is in $\mathcal{D}L^{i+k}$ or redundant in $\mathcal{D}L^{i+k}$

In the previous chapter we have shown that generating clause instances with relevant instantiators obtained from proofs of contradictions from constrained literals is a fair Inst-saturation process and thus refutationally complete. We use this result to justify refutational completeness with the following lifting theorem.

Theorem 5.2. *If there is a USD-proof P of a contradiction $\square$ from the constrained literals $L_1 \mid D_1, \ldots, L_n \mid D_n$ in the USD-saturation $\langle \mathcal{N}D, \mathcal{D}D \rangle^{sat}$, then there is a set labelled contradiction $\mathcal{L}\colon \square$ in some state of the USL-saturation process, such that for at least one proper P-relevant instantiator σ for the clause C a closure equal up to renaming to $C \cdot \sigma$ is in $\mathcal{L}$.*

Proof. For each S-relevant constrained literal $L \mid D$ in the first state of the USD-saturation process, there is an initially labelled literal $\{C \cdot \theta\} : L\theta$ in the first state of the USL-saturation process. Assuming all dismatching constraints

to be empty, the side conditions on the unit superposition inference and the equality resolution inference are identical in the unlabelled unit superposition calculus with dismatching constraints (Definition 4.24 on page 78) and in the set labelled unit superposition calculus (Definition 5.2). Hence, a contradiction can be derived in the USL-saturation process if a contradiction can be derived in the USD-saturation process.

Now assume that the set labelled contradiction $\mathcal{L}\colon \square$, obtained by repeating the inferences in the USD-proof P in the set labelled unit superposition calculus, is USL-redundant and removed in some state j in the USL-saturation process. We have $\mathcal{L} = \{C_1 \cdot \sigma_1, \ldots, C_n \cdot \sigma_n\}$, where $\mathrm{sel}\,(C_i) = L_i$ for each $1 \leq i \leq n$. If the set labelled contradiction $\mathcal{L}\colon \square$ is USL-redundant due to a set labelled literal $\mathcal{L}'\colon \square$ in $\langle \mathcal{N}L^j, \mathcal{D}L^j \rangle$, then there is a renaming ρ such that $\mathcal{L}'\rho \supsetneq \mathcal{L}$. Since there are no variables in $\square$, the renaming σ is the empty substitution []. We continue with $\mathcal{L}'\colon L'$. Otherwise, let $\mathcal{L}'\colon L' = \mathcal{L}\colon L$. We have $\mathcal{L}'\rho \supsetneq \{C_1 \cdot \sigma_1\rho, \ldots, C_n \cdot \sigma_n\rho\}$ and each $\sigma_i\rho$ is equal up to renaming to σ_i.

If the labelled contradiction $\mathcal{L}'\colon \square$ is USL-redundant in $\langle \mathcal{N}L^j, \mathcal{D}L^j \rangle$, then all constrained literals in $\mathrm{DLit}\,(\mathcal{L}'\colon L') \supseteq \{\square \mid D_1\sigma_1\rho, \ldots, \square \mid D_n\sigma_1\rho\}$ are USD-redundant in $\mathrm{DLit}\,(\langle \mathcal{N}L^j, \mathcal{D}L^j \rangle)$. Because $\square$ is the smallest literal and there are no smaller literals it follows from, the sets of closures $\mathrm{Cl}\,(\square \mid D_i\sigma_i\rho)$ of the constrained contradictions are empty and hence the dismatching constraints $D_i\sigma_i\rho$ are unsatisfiable for all $1 \leq i \leq n$.

In the USD-proof P the contradiction $\square \mid D$ at the root is constrained with D, which is the union of the constraints $D_1\sigma_1, \ldots, D_n\sigma_n$, where $\sigma_1, \ldots, \sigma_n$ are the respective P-relevant instantiators for $L_1 \mid D_1, \ldots, L_n \mid D_n$. In the USD-proof P each P-relevant instantiator σ_i satisfies the constraint D_i by Lemma 4.26 on page 83. However, this contradicts the assumption of USL-redundancy of $\mathcal{L}'\colon \square$ and the consequence that all dismatching constraints $D_i\sigma_i\rho$ are unsatisfiable.

We conclude that the set labelled contradiction $\mathcal{L}'\colon \square$ is not USL-redundant, therefore contained in some state of the USL-saturation process $\langle \mathcal{N}L^j, \mathcal{D}L^j \rangle$ and the set label $\mathcal{L}'$ contains a closure $C_i \cdot \sigma_i\rho$, which is equal up to renaming to a closure $C_i \cdot \sigma_i$ of a proper P-relevant instantiator σ_i for the clause C_i. $\square$

Instead of the Inst-saturation process with the USD-saturation process on constrained literals and extraction of relevant instances from USD-proofs of a contradiction, we have another fair Inst-saturation process based on the USL-saturation process on set labelled literals. Here we do not need to record proofs

or literal variants and can rely on set labels of contradictions derived in the set labelled unit superposition calculus to provide sufficient clause instances for fairness of an Inst-saturation process.

Redundancy Elimination in Set Labels

Set labels are a concise and powerful enough mechanism in many practical cases and an implementation based on set labels is efficient, as the evaluation in Chapter 8 shows.

However, set labels do not eliminate redundancy completely in the incremental process of instantiation. Set labels collect clauses at the leaves of proofs and accumulate respective relevant substitutions. Merging inferences combine superposition proofs and the conclusion contains closures from several proofs. When a leaf clause becomes redundant with the accumulated relevant substitution, every proof with the clause at a leaf is redundant and all leaf clauses in this proof can be eliminated. However, in a set label we cannot separate out all closures corresponding to the redundant proof from a set label since the structure is lost when two proofs are merged.

Example 5.5. Let us extend the running example and add dismatching constraints for redundancy elimination. We consider the clauses (1)-(4) in Examples 5.1 and 5.3 with initially empty dismatching constraints. We add to the clause set an instance of clause (2) with the substitution $[b/u, c/z]$.

$$\underline{f(x,y) \simeq f(y,x)} \tag{1}$$

$$\underline{f(u,v) \not\simeq g(z)} \vee u \simeq z \quad \mid \{\langle b,c\rangle \not\trianglerighteq \langle u,z\rangle\} \tag{2}$$

$$\underline{f(b,v) \not\simeq g(c)} \ \vee \ b \simeq c \tag{2'}$$

$$\underline{f(a,b) \simeq g(c)} \tag{3}$$

$$\underline{a \not\simeq b} \tag{4}$$

In order to block in clause (2) the redundant ground closures already represented by its instance (2′), the constraint of the instantiating substitution $[b/u, c/z]$ is added to the dismatching constraint of the instantiated clause (2).

In the unlabelled unit superposition proof (†) from Example 5.1 we obtain the relevant substitution $\sigma_{22} = [b/x, a/y, b/u, a/v, c/z]$ for the literal L_2 which is the selected literal in clause (2). The dismatching constraint $\{\langle b,c\rangle \not\trianglerighteq \langle u,z\rangle\}$ blocks

the substitution σ_{22}. Moreover, since the dismatching constraint for one clause at a leaf of the proof (†) is not satisfied, the entire proof is redundant and none of the relevant instances

$$\{(3)\,[]\,,(1)\,[b/x,a/y]\,,(2)\,[b/u,a/v,c/z]\}$$

needs to be added to the clause set.

However, proof (∗) is not redundant, since the relevant substitution σ_1 satisfies the dismatching constraints of all clauses at the leaves. In particular for clause (2) the relevant substitution $\sigma_1 = [a/x, b/y, c/z]$, is not blocked by the dismatching constraint. Hence the relevant instances

$$\{(3)\,[]\,,(2)\cdot[a/u,b/v,c/z]\}$$

are required.

In the set labelled unit superposition calculus in Example 5.3, both proofs were merged and we had

$$\{(3)\cdot[],(2)\cdot[a/u,b/v,c/z],(1)\cdot[b/x,a/y],(2)\cdot[b/u,a/v,c/z]\}$$

as the label of the contradiction.

If we do not merge literal variants in the set labelled unit superposition, we obtain two separate proofs, corresponding to the two unlabelled proofs. For the redundant unlabelled proof (†) we obtain the set label of the contradiction

$$\{(2)\cdot[b/u,a/v,c/z],(1)\cdot[b/x,a/y],(3)\cdot[]\}$$

and we want to eliminate these redundant closures from the set label of the contradiction in the merged proofs. However, the set label from the non-redundant proof (∗) is

$$\{(3)\cdot[],(2)\cdot[a/u,b/v,c/z]\}$$

and we have to retain these closures.

As the information about the proof structure cannot be recovered from a set label, we cannot eliminate all closures from the set label. In particular, we would need to know that the proofs overlap on closure $(3)\cdot[]$ and not on $(1)\cdot[b/x,a/y]$.

A similar problem arises from changes in the selection function in the incremental Inst-saturation process. The model of the ground abstraction may change in a way that a different literal has to be selected in a clause than before. In that case, all proofs with the previously selected literal at a leaf should be eliminated. Again, we want to remove a subset of the clauses in the label of a literal and it is not possible to determine if a clause has to be kept in the label as it may well be from the label of a non-redundant proof that was merged.

The cause of the problem is that we use the set union for combining labels in both the merging inference and in the superposition inference. In the next section we present a different label structure that preserves the shape of proofs by using two different operations in merging and superposition.

Let us finally note again that set labels are still a useful sound and complete mechanism. Although we cannot determine the full subset to be eliminated from a label, we can safely remove each closure from a set label which has become redundant with its substitution or where the selection has changed. A set label is then an overapproximation of the set of relevant instances of unlabelled proofs. Hence, unit superposition with set labels generates more instances of clauses than strictly necessary, while adding these instances does not harm soundness nor completeness. An instance of a clause is a sound consequence of the clause.

Set labelled unit superposition is a decision procedure for the Bernays-Schönfinkel fragment as we show in Section 5.5. In Chapter 8 we evaluate an implementation of set labels with this restricted elimination against the label structures with more powerful elimination presented next.

5.3 Tree Labelled Unit Superposition

Unlabelled proofs become redundant when dismatching constraints are extended after instantiating clauses and when leaf literals are no longer selected in a clause. In order to eliminate from a label precisely the closures corresponding to relevant instances from redundant unlabelled proofs, we need to preserve a certain Boolean structure in the labels in the unit superposition calculus to keep track of merging inferences.

To this end we regard a closure $C \cdot \theta$ as a propositional variable. A merging inference corresponds to a disjunction and a superposition to a conjunction of labels. Eliminating a redundant literal and precisely the closures from proofs

depending on the redundant literal then means replacing a propositional variable with the propositional constant false and simplifying the Boolean structure.

Definition 5.8. A *tree label* $\mathcal{T}$ is either a closure $C \cdot \theta$, which is not necessarily ground, a conjunction $\bigsqcap_{i=1}^{n} \mathcal{T}_i$ or a disjunction $\bigsqcup_{i=1}^{n} \mathcal{T}_i$ of n tree labels $\mathcal{T}_1, \ldots, \mathcal{T}_n$.

The set of closures $\bigcup(\mathcal{T})$ of the tree label $\mathcal{T}$ is recursively defined as

$$\bigcup(\mathcal{T}) = \begin{cases} C \cdot \theta & \text{if } \mathcal{T} = C \cdot \theta \\ \bigcup_{i=1}^{n} \bigcup(\mathcal{T}_i) & \text{if } \mathcal{T} = \bigsqcap_{i=1}^{n} \mathcal{T}_i \\ \bigcup_{i=1}^{n} \bigcup(\mathcal{T}_i) & \text{if } \mathcal{T} = \bigsqcup_{i=1}^{n} \mathcal{T}_i. \end{cases}$$

A *tree labelled literal* $\mathcal{T} : L$ is a pair of a tree label $\mathcal{T}$ and a literal L, where $\mathcal{T}$ and L are variable disjoint, such that $\operatorname{var}(C) \cap \operatorname{var}(L) = \emptyset$ for each $C \cdot \theta \in \bigcup(\mathcal{T})$.

Let sel be a selection function, C be a clause and $C \cdot \theta$ be an arbitrary but fixed closure, such that C and $C\theta$ are equal up to renaming. The *initial labelling* of the selected literal L in clause C with respect to the selection function sel is the tree labelled literal $C \cdot \theta : \operatorname{sel}(C)\,\theta$.

Let $\mathcal{T}$ be a tree label and σ a substitution, which is variable disjoint from $\mathcal{T}$ in the following way: $\operatorname{var}(\operatorname{rng}(\sigma)) \cap \operatorname{var}(\bigcup(\mathcal{T})) = \emptyset$. The *$\sigma$-instance* of a tree label $\mathcal{T}$ is the tree label $\mathcal{T}\sigma$ such that

$$\mathcal{T}\sigma = \begin{cases} C \cdot \theta\sigma & \text{if } \mathcal{T} = C \cdot \theta, \\ \bigsqcap_{i=1}^{n} \mathcal{T}_i\sigma & \text{if } \mathcal{T} = \bigsqcap_{i=1}^{n} \mathcal{T}_i \text{ or} \\ \bigsqcup_{i=1}^{n} \mathcal{T}_i\sigma & \text{if } \mathcal{T} = \bigsqcup_{i=1}^{n} \mathcal{T}_i. \end{cases}$$

A tree label $\mathcal{T} = C \cdot \theta$ is equal up to renaming to a tree label $\mathcal{T}' = C' \cdot \theta'$ if the closures are equal up to renaming. A tree label $\mathcal{T} = \mathcal{T}_1 \sqcup \cdots \sqcup \mathcal{T}_n$ is equal up to renaming to the tree label $\mathcal{T}' = \mathcal{T}'_1 \sqcup \cdots \sqcup \mathcal{T}'_n$ if for every $\mathcal{T}_i$ there is a $\mathcal{T}'_j$, which is equal up to renaming, and for every $\mathcal{T}'_i$ there is a $\mathcal{T}_j$ equal up to renaming. A tree label $\mathcal{T} = \mathcal{T}_1 \sqcap \cdots \sqcap \mathcal{T}_n$ is equal up to renaming to a tree label $\mathcal{T}' = \mathcal{T}'_1 \sqcap \cdots \sqcap \mathcal{T}'_n$ in the analogue way.

We say two tree labelled literals $\mathcal{T} : L$ and $\mathcal{T}' : L'$ are *equal up to renaming* if there is a renaming ρ such that $L = L'\rho$ and the tree labels $\mathcal{T}$ and $\mathcal{T}'\rho$ are equal up to renaming. We usually do not distinguish between tree labelled literals equal up to renaming.

A tree label is isomorphic to an AND/OR tree, where all non-leaf nodes are

either labelled as AND nodes or OR nodes. AND and OR nodes alternate on each level of the tree such that AND nodes only have OR nodes as successors and vice versa for OR nodes. A tree label is also isomorphic to a monotone Boolean formula in the following way.

Definition 5.9. The *formula of a tree label* $\mathcal{T}$, with respect to a set of constrained clauses S, written as $\varphi(\mathcal{T})$, is recursively defined as

$$\varphi(\mathcal{T}) = \begin{cases} \mathsf{false} & \text{if } \mathcal{T} = C \cdot \sigma \text{ and } C \mid D \in S, \text{ where } D\sigma \text{ is unsatisfiable} \\ X_{C \cdot \sigma} & \text{if } \mathcal{T} = C \cdot \sigma \text{ and } C \mid D \in S, \text{ where } D\sigma \text{ is satisfiable} \\ \bigwedge_{i=1}^{n} \varphi(\mathcal{T}_i) & \text{if } \mathcal{T} = \bigsqcap_{i=1}^{n} \mathcal{T}_i \\ \bigvee_{i=1}^{n} \varphi(\mathcal{T}_i) & \text{if } \mathcal{T} = \bigsqcup_{i=1}^{n} \mathcal{T}_i \end{cases}$$

The strength of tree labels when compared to set labels is the precise elimination of redundancy by simplifying the Boolean structure. If the closure $C \cdot \theta$ has to be eliminated from a label, we can simplify the tree label $C \cdot \theta \sqcup \mathcal{T}$ to $\mathcal{T}$ and the label $C \cdot \theta \sqcap \mathcal{T}$ to the empty label. Literals with the latter label are redundant and can be discarded.

As for set labelled literals, an initially tree labelled literal $C \cdot \theta \colon \operatorname{sel}(C)\,\theta$ satisfies the variable disjointness property $\operatorname{var}(C) \cap \operatorname{var}(\operatorname{sel}(C)\,\theta) = \emptyset$ and σ-instances of tree labels preserve this variable disjointness. Also, the binding of the variables in an initially labelled literal, where $\operatorname{sel}(C)\,\theta = L\theta$ is preserved when instantiating the tree label and the literal with the same substitution σ.

We now define a unit superposition calculus with different operators to combine labels in the merging and the superposition inference, namely $\sqcup$ and $\sqcap$.

Definition 5.10 (Tree Labelled Unit Superposition)**.** The respective conclusions of superposition and equality resolution are new labelled literals. The conclusion of merging replaces the two premises.

Unit Superposition

$$\frac{\mathcal{T}: l \simeq r \qquad \mathcal{T}': s[l'] \simeq t}{(\mathcal{T} \sqcap \mathcal{T}')\sigma: \ (s[r] \simeq t)\,\sigma}\,(\sigma) \qquad\qquad \frac{\mathcal{T}: l \simeq r \qquad \mathcal{T}': s[l'] \not\simeq t}{(\mathcal{T} \sqcap \mathcal{T}')\sigma: \ (s[r] \not\simeq t)\,\sigma}\,(\sigma)$$

where for some grounding substitution θ with $\operatorname{dom}(\theta) = \operatorname{var}(\{l\sigma, r\sigma, s[l']\sigma, t\sigma\})$

(i) $\sigma = \operatorname{mgu}(l, l')$,

(ii) l' is not a variable,

(iii) $l\sigma\theta \succ_{\text{gr}} r\sigma\theta$,

(iv) $s[l']\sigma\theta \succ_{\text{gr}} t\sigma\theta$,

(v) $\operatorname{var}(\{l, r\}) \cap \operatorname{var}(\{s[l'], t\}) = \emptyset$.

and $\varphi(\mathcal{T}\sigma)$ and $\varphi(\mathcal{T}'\sigma)$ are satisfiable.

Merging (Replacement rule for both premises)

$$\frac{\mathcal{T}: l \simeq r \qquad \mathcal{T}': l' \simeq r'}{\mathcal{T} \sqcup \mathcal{T}'\sigma: l \simeq r}\,(\sigma) \qquad\qquad \frac{\mathcal{T}: l \not\simeq r \qquad \mathcal{T}': l' \not\simeq r'}{\mathcal{T} \sqcup \mathcal{T}'\sigma: l \not\simeq r}\,(\sigma)$$

where σ is a renaming such that $l'\sigma = l$ and $r'\sigma = r$.

Equality Resolution

$$\frac{\mathcal{T}: l \not\simeq r}{\mathcal{T}\sigma: \square}\,(\sigma)$$

where $\sigma = \operatorname{mgu}(l, r)$ and $\varphi(\mathcal{T}\sigma)$ is satisfiable.

As in the set labelled unit superposition calculus, we start with the selected literals, which are initially labelled with the respective clauses they are selected in. Upon an equality resolution to a contradiction we generate the instances from all closures at the leaves of the tree.

Definition 5.11. The *relevant instantiators* and the *relevant instances* of a tree labelled literal $\mathcal{T}: L$ are the relevant instantiators and the relevant instances of the set labelled literal $\bigcup(\mathcal{T}) : L$.

Let us give an example to illustrate elimination of redundancy by simplification of the AND/OR tree.

Example 5.6. We apply tree labelled unit superposition inferences instead of set labelled unit superposition inferences on the selected literals in the clause set in Example 5.3.

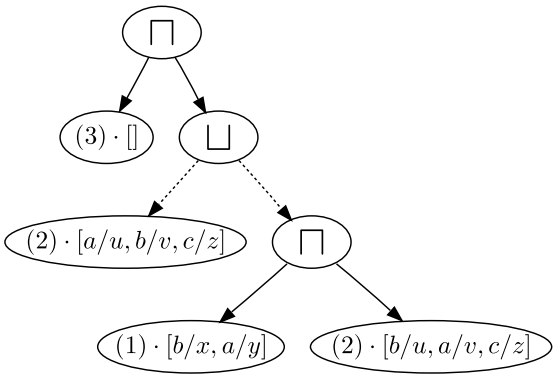


Figure 5.1: AND/OR tree of the tree labelled contradiction in Example 5.6.

The contradiction in the set labelled unit superposition proof from Example 5.3 has the set label

$$\mathcal{L} = \Big\{(3)\cdot[], (2)\cdot[a/u, b/v, c/z], (1)\cdot[b/x, a/y], (2)\cdot[b/u, a/v, c/z]\Big\}.$$

In the corresponding tree labelled unit superposition proof, we obtain the label

$$\mathcal{T} = (3)\cdot[] \sqcap \Big((2)\cdot[a/u, b/v, c/z] \sqcup \Big((1)\cdot[b/x, a/y] \sqcap (2)\cdot[b/u, a/v, c/z]\Big)\Big),$$

$$\varphi(\mathcal{T}) = X_{(3)\cdot[]} \wedge \Big(X_{(2)\cdot[a/u,b/v,c/z]} \vee \big(X_{(1)\cdot[b/x,a/y]} \wedge X_{(2)\cdot[b/u,a/v,c/z]}\big)\Big),$$

also pictured in Figure 5.1. In contrast to the set label, the tree label with the distinct operators $\sqcap$ and $\sqcup$ for superposition and merging, respectively, keeps the proof structure visible. Going back to Example 5.1, the two unlabelled proofs $(*)$ and $(\dagger)$ that the labelled calculi join in the merging inference can be told apart in the tree label, but not in the set label.

If, as in Example 5.5, we are to eliminate the closure $(2)\cdot[b/u, a/v, c/z]$ due to a dismatching constraint blocking the substitution, we consider the Boolean formula $\varphi(\mathcal{T})$ of the tree label $\mathcal{T}$. We replace the propositional variable $X_{(2)\cdot[b/u,a/v,c/z]}$ with the propositional constant false and simplify the Boolean formula by recursively rewriting the Boolean formula with the logically equivalences $A \vee \mathsf{false} \to A$ and $A \wedge \mathsf{false} \to \mathsf{false}$.

$$X_{(3)\cdot[]} \wedge \Big(X_{(2)\cdot[a/u,b/v,c/z]} \vee \big(X_{(1)\cdot[b/x,a/y]} \wedge \mathsf{false}\big)\Big) = X_{(3)\cdot[]} \wedge X_{(2)\cdot[a/u,b/v,c/z]}$$

The relevant instances from the tree label corresponding to the simplified

formula are exactly the relevant instances from the unlabelled proof (†) in Example 5.1. In the set label we can only eliminate the closure $(2) \cdot [b/u, a/v, c/z]$, since the information about where proofs are merged is not kept.

Eliminating a closure from the other proof (∗), for instance $(2) \cdot [a/u, b/v, c/z]$, from the formula of the tree label leaves us with

$$X_{(3)\cdot[]} \wedge \left(\mathsf{false} \vee \left(X_{(1)\cdot[b/x,a/y]} \wedge X_{(2)\cdot[b/u,a/v,c/z]}\right)\right) = X_{(3)\cdot[]} \wedge X_{(1)\cdot[b/x,a/y]} \wedge X_{(2)\cdot[b/u,a/v,c/z]}.$$

As before, we find three relevant instances in the simplified tree label, which are exactly the relevant instances from the unlabelled proof.

Completeness

In order to prove completeness of an Inst-saturation process with tree labelled unit superposition, we prove a lifting theorem similar to the one in the previous section for set labelled unit superposition. We proceed in a parallel way and give definitions adapted to tree labels.

Definition 5.12. Let us define an auxiliary function DC to obtain a dismatching constraint from a tree label, using disjunctive dismatching constraints from Definition 4.22 on page 74 for the first time.

$$\mathrm{DC}(\mathcal{T}) = \begin{cases} D\sigma & \text{if } \mathcal{T} = C \cdot \sigma \text{ and } C \mid D \in S \\ \bigwedge_{i=1}^{n} \mathrm{DC}(\mathcal{T}_i \colon L) & \text{if } \mathcal{T} = \prod_{i=1}^{n} \mathcal{T}_i \\ \bigvee_{i=1}^{n} \mathrm{DC}(\mathcal{T}_i \colon L) & \text{if } \mathcal{T} = \bigsqcup_{i=1}^{n} \mathcal{T}_i. \end{cases}$$

The *constrained literal* of a tree labelled literal $\mathcal{T} \colon L$ is

$$\mathrm{DLit}(\mathcal{T} \colon L) = L \mid \mathrm{DC}(\mathcal{T}).$$

We extend DLit from tree labelled literals to sets of tree labelled literals as

$$\mathrm{DLit}(\mathcal{L}T) = \{\mathrm{DLit}(\mathcal{T} \colon L) \mid \mathcal{T} \colon L \in \mathcal{L}T\}.$$

In contrast to a set labelled literal $\mathcal{L} \colon L$, where we have a set of constrained literals $\mathrm{DLit}(\mathcal{L} \colon L)$, there is only one constrained literal $\mathrm{DLit}(\mathcal{T} \colon L)$ for a tree

labelled literal $\mathcal{T}\colon L$. Further, the dismatching constraint $\mathrm{DC}(\mathcal{T})$ is isomorphic to the formula $\varphi(\mathcal{T})$ of the tree label $\mathcal{T}$.

Definition 5.13. Let $\mathcal{L}T$ be a set of tree labelled literals. A tree labelled literal $\mathcal{T}\colon L \in \mathcal{L}T$ is *UST-redundant* in $\mathcal{L}T$ if

(i) the constrained literal $\mathrm{DLit}(\mathcal{L}\colon T)$ is USD-redundant in $\mathrm{DLit}(\mathcal{L}T)$ or

(ii) there is a $\mathcal{T}'\colon L' \in \mathcal{L}T$ such that L and L' are equal up to the renaming σ and there is a renaming ρ such that $\varphi(\mathcal{T}) \models \varphi(\mathcal{T}'\sigma\rho)$ and not $\varphi(\mathcal{T}'\sigma\rho) \models \varphi(\mathcal{T})$.

Let $\mathcal{R}_{\mathrm{UST}}(\mathcal{L}T)$ denote the set of all tree labelled literals in $\mathcal{L}T$, which are UST-redundant in $\mathcal{L}T$.

Redundancy of tree labels is not defined with set inclusion as for set labels, but with entailment on the formulae of a tree label. It has analogous properties, such that a tree labelled literal $\mathcal{T}\colon L$ with $\varphi(\mathcal{T}) = \mathsf{false}$ is redundant in any non-empty set of tree labelled literals. Due to the *"ex falso quodlibet"* principle, we have the entailment $\varphi(\mathcal{T}) = \mathsf{false} \models \varphi(\mathcal{T}')$ for the formula of any tree label $\mathcal{T}'$.

If we allowed $\varphi(\mathcal{T}'\sigma\rho) \models \varphi(\mathcal{T})$, then we would have the same situation as for non-strict inclusion of set labels: $\mathcal{T}$ would make $\mathcal{T}'$ UST-redundant and vice versa.

In UST-redundancy, a more general tree label $\mathcal{T}'$ makes a more specific tree label $\mathcal{T}$ redundant, that is, if $\mathcal{T}'$ is a consequence of $\mathcal{T}$. Since there is no most general tree label, whose formula entails the formula of all other tree labels, UST-redundancy is not well-founded. As before in set labelled unit superposition, the merging inference on tree labelled literals makes both its premises UST-redundant.

Theorem 5.3. *The merging inference rule is a simplification inference, that is the conclusion makes both of its premises UST-redundant.*

Proof. The literal in the conclusion of a merging inference is equal to the literal in the left premise and equal up to the renaming σ to the literal in the right premise. The formula of the tree label of the conclusion is $\varphi(\mathcal{T}) \vee \varphi(\mathcal{T}'\sigma)$. We have both $\varphi(\mathcal{T}) \models \varphi(\mathcal{T}) \vee \varphi(\mathcal{T}'\sigma)$ and $\varphi(\mathcal{T}'\sigma) \models \varphi(\mathcal{T}) \vee \varphi(\mathcal{T}'\sigma)$, hence both premises of the merging inference are UST-redundant in presence of the conclusion. □

The elimination of the premises in the merging inference is covered by redundancy and we are working with the more general concept of UST-redundancy in the following.

We lift the definitions around the US-saturation process in close analogy to the lifting in the context of the set labelled unit superposition.

Definition 5.14. A *UST-saturation process* on tree labelled literals is an infinite sequence of pairs $\{\langle \mathcal{N}T^i, \mathcal{D}T^i \rangle\}_{i=1}^{\infty}$, where $\mathcal{N}T^i$ is a set of tree labelled literals and $\mathcal{D}T^i$ a disjoint set of positive tree labelled literals. Each pair $\langle \mathcal{N}T^{i+1}, \mathcal{D}T^{i+1} \rangle$ with $i > 1$, called a *successor state*, is obtained from the previous state $\langle \mathcal{N}T^i, \mathcal{D}T^i \rangle$ by either

(i) adding to $\mathcal{N}T^i$ the conclusion of a tree labelled unit superposition inference with one premise in $\mathcal{N}T^i$ and the second premise in the union $\mathcal{N}T^i \cup \mathcal{D}T^i$, adding to $\mathcal{N}T^i$ the conclusion of a tree labelled equality resolution inference with the premise in $\mathcal{N}T^i$, adding to $\mathcal{N}T^i$ the conclusion of a tree labelled merging inference with both premises in $\mathcal{N}T^i$,

(ii) adding to $\mathcal{D}T^i$ the conclusion of a tree labelled unit superposition inference with both premises in $\mathcal{D}T^i$, adding to $\mathcal{D}T^i$ the conclusion of a tree labelled equality resolution inference with its premise in $\mathcal{D}T^i$, adding to $\mathcal{D}T^i$ the conclusion of a tree labelled merging inference with both premises in $\mathcal{D}T^i$,

(iii) removing a tree labelled literal from $\mathcal{N}T^i$ which is UST-redundant in the union $\mathcal{N}T^i \cup \mathcal{D}T^i$ or

(iv) removing a tree labelled literal from $\mathcal{D}T^i$ which is UST-redundant in $\mathcal{D}T^i$.

Since UST-redundancy in the same way as USL-redundancy is not well-founded, we face the same problems defining persistent literals and the notion of UST-saturation. We resort to a similar definition of fairness as in the USL-saturation process, which also in this case is sufficient to prove completness.

Definition 5.15. A UST-saturation process is *fair* if there is a k for every i such that

(i) for every possible unit superposition inference with with one premise in $\mathcal{N}T^i$ and the second premise in $\mathcal{N}T^i \cup \mathcal{D}T^i$, for every equality resolution inference with the premise in $\mathcal{N}T^i$ and for every merging inference with both

premises in $\mathcal{N}T^i$ the respective conclusion is in $\mathcal{N}T^{i+k}$ or UST-redundant in $\mathcal{N}T^i \cup \mathcal{D}T^i$ and

(ii) for every possible unit superposition inference with both premises in $\mathcal{D}T^i$ and for every equality resolution inference with the premise in $\mathcal{D}T^i$ the respective conclusion is in $\mathcal{D}T^{i+k}$ or UST-redundant in $\mathcal{D}T^i$.

With the above definitions we can prove a lifting theorem that allows us to use tree labelled unit superposition in a fair Inst-saturation process.

Theorem 5.4. *If there is a USD-proof P of a contradiction $\square$ from the constrained literals $L_1 \mid D_1, \ldots, L_n \mid D_n$ in the USD-saturation $\langle \mathcal{N}D, \mathcal{D}D \rangle^{sat}$, then there is a tree labelled contradiction $\mathcal{T} \colon \square$ in some state of the UST-saturation process, such that for at least one proper P-relevant instantiator σ for the clause C a closure equal up to renaming to $C \cdot \sigma$ is in $\bigcup(\mathcal{T})$.*

Proof. We have the same premise as in the lifting of set labelled unit superposition in Theorem 5.2. For each S-relevant constrained literal $L \mid D$ in the first state of the USD-saturation process, there is an initially labelled literal $C \cdot \theta \colon L\theta$ in the first state of the UST-saturation process. If all dismatching constraints are empty, then there is a contradiction $\mathcal{T} \colon \square$ in the some state j of the UST-saturation process $\langle \mathcal{N}T^j, \mathcal{D}T^j \rangle$, which can be derived with inferences in the tree labelled unit superposition calculus, if a contradiction can be derived in the USD-saturation $\langle \mathcal{N}D, \mathcal{D}D \rangle^{\text{sat}}$ from unlabelled unit superposition inferences, since the side conditions of the inference rules are equivalent.

Let us introduce some terminology and a normal form of tree labels for the purpose of this proof, see for instance Goldsmith et al. [2005]. We call a conjunction of literals a *monomial*, which is the dual to the disjunction of literals in a clause. We also consider a monomial as a set. A Boolean formula without negations is called *monotone*. A *disjunctive normal form (DNF)* of a Boolean formula F is a disjunction of monomials that is logically equivalent to F. An *implicant* C of a formula F is a monomial such that the formula $C \to F$ is valid. An implicant is a *prime implicant* if $C \to F$ and no proper subset $C' \subsetneq C$ is an implicant. For a monotone Boolean formula F the disjunction of all its prime implicants is a logically equivalent and unique DNF, which we call the *canonical DNF* of F. For non-monotone formulae this property does not hold. Since the formula $\varphi(\mathcal{T})$ of a tree label is monotone, we take the canonical DNF as the normal form of $\varphi(\mathcal{T})$.

With tree labelled unit superposition inferences corresponding to unlabelled unit superposition inferences in the derivation in the USD-proof P we derive a tree labelled contradiction $\mathcal{T}\colon \square$, where $\mathcal{T} = C_1 \cdot \sigma_1 \sqcap \cdots \sqcap C_n \cdot \sigma_n$.

We assume that $\mathcal{T}\colon \square$ is UST-redundant in some state j of the UST-saturation process $\langle \mathcal{N}T^j, \mathcal{D}T^j \rangle$. If it is UST-redundant because $\langle \mathcal{N}T^j, \mathcal{D}T^j \rangle$ contains a tree labelled contradiction $\mathcal{T}'\colon \square$, then there is a renaming ρ such that $\varphi(\mathcal{T}) \models \varphi(\mathcal{T}'\rho)$. The renaming $\sigma = []$, since there are no variables in $\square$. Let us consider the canonical DNF of the monotone formula $\varphi(\mathcal{T}'\rho)$ and the Boolean formula $\varphi(\mathcal{T})$, which is a monomial and thus already in canonical DNF. If $\varphi(\mathcal{T}) \models \varphi(\mathcal{T}'\rho)$, then, without loss of generality, there is a prime implicant $X_{C_1 \cdot \sigma_1 \rho} \wedge \cdots \wedge X_{C_l \cdot \sigma_l \rho}$ with $l \leq n$ in the canonical DNF of $\varphi(\mathcal{T}'\rho)$. Since $\mathcal{T}' \in \langle \mathcal{N}T^j, \mathcal{D}T^j \rangle$, there is a derivation of a contradiction from the tree labelled literals $C_1 \cdot \sigma_1\rho\colon L_1, \ldots, C_l \cdot \sigma_l\rho\colon L_l$ and a corresponding USD-proof P' from constrained literals $L_1 \mid D_1, \ldots, L_l \mid D_l$. Since in every USD-proof there is at least one proper instantiator (Theorem 4.22 on page 72), there is a proper instantiator already in $\sigma_1\rho, \ldots, \sigma_l\rho$ and it suffices to consider the shorter proof P' instead of P, where the relevant instances are variants of $\sigma_1, \ldots, \sigma_l$.

If there is no tree labelled contradiction $\mathcal{T}'\colon \square \in \{\langle \mathcal{N}T^i, \mathcal{D}T^i \rangle\}_{i=1}^{\infty}$ making $\mathcal{T}\colon \square$ UST-redundant, we let $\mathcal{T}'\colon \square = \mathcal{T}\colon \square$.

Since $L_1 \mid D_1, \ldots, L_l \mid D_l \in \langle \mathcal{N}D, \mathcal{D}D \rangle^{\text{sat}}$, each of the P'-relevant instantiators $\sigma_1, \ldots, \sigma_l$ satisfies the respective dismatching constraint $D_1, \ldots D_l$. By the same argument as in the set labelled proof, $\mathcal{T}'\colon \square$ is not UST-redundant: the contradiction $\square$ is the smallest literal and does not follow from smaller literals, hence $\mathrm{Cl}(\mathrm{DLit}(\mathcal{T}'))$ must be empty, which contradicts the satisfiability of the dismatching constraints $D_i\sigma_i$ for $1 \leq i \leq n$.

We conclude that $\mathcal{T}'\colon \square$ is not UST-redundant and $\bigcup(\mathcal{T}')$ contains at least one closure $C_i \cdot \sigma_i\rho$, where σ_i is a proper P-relevant instantiator for C. $\square$

Tree labelled unit superposition therefore leads to another fair Inst-saturation process. We saturate the set of initially labelled selected literals under unit superposition inferences and generate instances from tree labelled contradictions. The stronger notion of UST-redundancy enables precise elimination of redundancy, compared to the weaker USL-redundancy for set labels.

Since UST-redundancy is defined on logical entailment between the Boolean formulae of tree labels, we can arbitrarily transform a tree label as long as logical equivalence of its formula is preserved. In particular we exploit this property to

eliminate subtrees, where the substitution of a closure $C \cdot \sigma$ does not satisfy the dismatching constraint of the clause $C \mid D$. For the same reason we may simplify tree labels where a leaf literal is no longer selected in a clause.

Tree labelled unit superposition is implemented in our system as an alternative to set labelled unit superposition and we compare the two labelling approaches in Chapter 8. However, we find that tree labels with precise elimination of redundancy are not automatically superior to set or tree labels. A tree label preserves the structure of the proofs being merged, but it is not necessarily in a normal form as a set label is. In the next section we are concerned with combining the positive features of tree and set labels, namely precise redundancy elimination and normal forms.

5.4 OBDD Labelled Unit Superposition

By Definition 5.8 the tree labelled literals $\mathcal{T}\colon L$ and $\mathcal{T}'\colon L'$ are equal up to renaming if the literals L and L' are equal up to the renaming σ and there is a renaming ρ such that the tree labels $\mathcal{T}$ and $\mathcal{T}'\sigma\rho$ are isomorphic. However, UST-redundancy (Definition 5.13) allows us to go beyond simple isomorphism of trees and to consider logical equivalence of formulae of tree labels. The tree label $\mathcal{T}\colon L$ makes the tree label $\mathcal{T}'\colon L'$ UST-redundant if the literals L and L' are equal up to the renaming σ and there is a renaming ρ such that the formulae $\varphi(\mathcal{T})$ and $\varphi(\mathcal{T}'\sigma\rho)$ are logically equivalent. We now look at with mechanisms to exploit this stronger notion of equality up to renaming.

Since there are many tree labels with logically equivalent formulae, the natural question of normal forms for tree labels arises. The sequence of merging and superposition inferences determines the shape of the tree, which makes comparing tree labels by their shape unusable except in simple cases. Maintaining tree labels in a normal form has obvious benefits such as uniform algorithms and data structures. More importantly, however, keeping tree labels in a suitable normal form makes it computationally easy to discard tree labelled literals, whose formulae are logically equivalent up to renaming to the formula of a previously derived tree labelled literal. In a UST-saturation process this is not only an important simplification step leading to greater efficiency, in certain cases it is essential for termination as we show on our running example.

Consequently, only Inst-Gen-Eq with tree labelled unit superposition, treating

equality of tree labels up to renaming based on logical equivalence of formulae, is a decision procedure for the Bernays-Schönfinkel fragment of first-order logic. Sets are unordered collections of elements and all set labels are in a natural normal form. We show that Inst-Gen-Eq with set labelled unit superposition is another decision procedure for the Bernays-Schönfinkel fragment of first-order logic.

Standard normal forms of Boolean formulae are the disjunctive and conjunctive normal forms (DNF and CNF) and we have already used the DNF in the proof of Theorem 5.4. However, in practice DNFs and CNFs are unfortunately frequently exponential in the size of the original formula. There exist now widely used approaches to computing small CNFs in first-order logic (see Nonnengart and Weidenbach [2001]) like the definitional transformation, which introduces new variables for subterms of the original formula. Therefore, it does not produce a unique normal form as needed for our purposes.

The formula of a tree label is monotone, that is, does not contain negations, but Goldsmith et al. [2005] prove that computing a unique DNF of a monotone Boolean formula is as complex as it is for arbitrary Boolean formulae. There are DNFs of exponential size for monotone Boolean formulae and checking whether a monomial is a prime implicant is a **DP**-complete problem. Languages in the problem class **DP** are the set difference between two languages in **NP**, the following inclusions hold in the polynomial complexity hierarchy: $\mathbf{NP} \subseteq \mathbf{DP} \subseteq \Sigma_2\mathbf{P}$ [Papadimitriou, 1994]. These properties make it appear unattractive to maintain tree labels in DNF.

Instead, we choose ordered binary decision diagrams (OBDDs) [Bryant, 1986], which offer particularly promising features, above all a unique normal form and checking of equivalence in constant time. An OBDD is a graph with common subtrees shared, it provides a compact and well-understood normal form of Boolean formulae. Although OBDDs can be exponential in the size of the original formula just as CNFs and DNFs, they are successfully used to encode Boolean structures in applications like formal verification [Bryant, 1995] and also in similar contexts to ours [de Moura and Bjørner, 2008a].

An OBDD is based on the Shannon decomposition of Boolean formulae, which is the logical equivalence

$$F \equiv (\overline{x} \wedge F_{\overline{x}}) \vee (x \wedge F_x)\,.$$

The formula F_x is the *positive cofactor* of F with respect to the variable x, it is

obtained by replacing the variable x in F with the propositional constant true. The *negative cofactor* $F_{\overline{x}}$ is obtained by replacing the variable x with false. We choose an arbitrary but fixed ordering on the variables of a Boolean formula F and recursively split F into positive and negative cofactors with respect to the ordering of its variables. Each node in an OBDD corresponds to a Boolean formula F, is labelled with a variable x and has two outgoing edges: one to the *high node*, corresponding to the positive cofactor F_x, and one to the *low node*, corresponding to the negative cofactor $F_{\overline{x}}$. Edges to low nodes are commonly drawn as dashed lines.

An OBDD is reduced in the sense that each formula is represented by exactly one node, hence there is exactly one node for the propositional constants true and false, which are conventionally labelled with 1 and 0, respectively. Further, an OBDD contains no redundant nodes, that is nodes where the succeeding high and low nodes are identical. The root of an OBDD represents the formula.

In order to compute the truth value of a formula given an assignment to the variables, one traverses the OBDD from the root to either the 0 or the 1 node, at each node labelled with a variable x following the high edge if x is assigned true and the low edge otherwise. All Boolean operations like conjunctions and disjunctions of OBDDs can be performed on their graph representation only, it is not necessary to switch to a different representation of the Boolean formula.

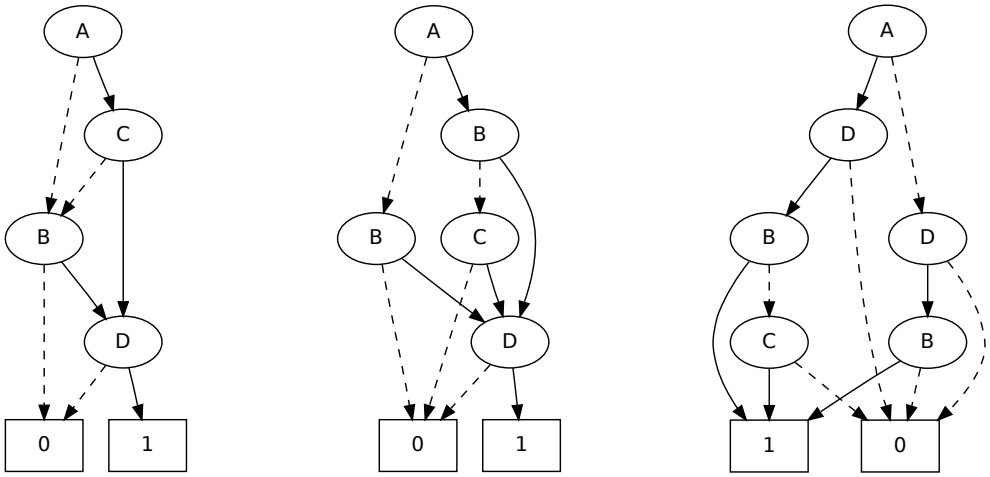
The most important property of OBDDs is their uniqueness: if two formulae are logically equivalent, their OBDDs are identical if the same variable ordering is used. Consequently, all unsatisfiable formulae are represented by the OBDD consisting of the single 0 node.

Example 5.7. Let us consider the tree label

$$\mathcal{T} = (3) \cdot [] \sqcap \Big((2) \cdot [a/u, b/v, c/z] \sqcup \big((1) \cdot [b/x, a/y] \sqcap (2) \cdot [b/u, a/v, c/z] \big) \Big),$$

depicted in Figure 5.1 on page 114 and use the following abbreviations, since we are only interested in the structure of the Boolean formula.

$$A = (1) \cdot [b/x, a/y]$$
$$B = (2) \cdot [a/u, b/v, c/z]$$
$$C = (2) \cdot [b/u, a/v, c/z]$$
$$D = (3) \cdot []$$



(a) $A > C > B > D$ (b) $A > B > C > D$ (c) $A > D > B > C$

Figure 5.2: OBDDs with three different variable orderings for the formula $D \wedge (B \vee (A \wedge C))$ of the tree label in Figure 5.1. Notice that permuting B and C in the alphabetic order of the middle OBDD (b) results in the smaller left OBDD (a). A permutation of C and D in the latter ordering leads to the larger right OBDD (c).

We have the following formula

$$\varphi(\mathcal{T}) = D \wedge \Big(B \vee \big(A \wedge C\big)\Big) = (B \wedge D) \vee (A \wedge C \wedge D) = D \wedge (A \vee B) \wedge (B \vee C)\,,$$

where the latter two formulae are in DNF and CNF.

Three OBDDs of the above formula with different variable orderings are in Figure 5.2. Depending on the variable ordering, the OBDDs have four (a), five (b) and six nodes (c).

It is well-known that the size and thus the performance of OBDDs strongly depends on the chosen variable ordering, see Bryant [1992]. Not surprisingly, finding an optimal ordering for the propositional variables to minimise the size of an OBDD is **NP**-complete as proved by Bollig and Wegener [1996], where the best published algorithm to obtain such an ordering is cited with a run time of $O(n3^n)$.

In the case of OBDD labelled unit superposition we have to deal with many OBDDs and in order to maximise sharing between the individual OBDDs, we consider the forest-like structure of a shared OBDD. A shared OBDD has multiple

roots, each corresponding to one OBDD and common subgraphs are shared across all represented OBDDs. It requires a global variable ordering and Tani et al. [1993] have proved that finding an optimal variable ordering in a shared OBDD is also **NP**-complete.

Despite the potential drawbacks, we have experimented with OBDD labelled unit superposition, since it seems more promising than the DNF as a normal form. In particular sharing nodes in OBDDs across all labels, the availability of industrial-strength OBDD libraries and the great success of OBDDs in diverse application areas make the effort seem worthwhile.

We extend tree labelled unit superposition to OBDD labelled superposition in a straightforward way. Instead of the AND/OR tree $\mathcal{T}$ (Definition 5.8) we use an OBDD $\mathcal{B}$, where each node is labelled with the propositional variable $X_{C\cdot\theta}$ of a closure $C \cdot \theta$ and $\mathcal{B}$ represents the formula $\varphi(\mathcal{T})$ of the tree label. The ordering of the variables $X_{C\cdot\theta}$ is arbitrary but fixed across the OBDDs in all labels. The set of closures $\bigcup(\mathcal{B})$ of an OBDD label is the set of all closures $C \cdot \theta$ such that there is a node labelled $X_{C\cdot\theta}$ in $\mathcal{B}$. We obtain the set of relevant instances and relevant instantiators in analogy to tree labels of an OBDD label from this set. Closures in the label of an OBDD labelled literal obey the same variable-disjointness properties as in tree labelled literals. The initial labelling of the literal $L = \mathrm{sel}(C)$ is the OBDD of the propositional variable $X_{C\cdot\theta}$.

In the σ-instance of an OBDD label we replace in each node the propositional variable $X_{C\cdot\theta}$ with the variable $X_{C\cdot\theta\sigma}$. However, this may require a reordering of the nodes in the OBDD, which can be a computationally expensive operation. Further, the structure of the OBDD changes if a substitution unifies two closures, that is, for some closures $C \cdot \theta$ and $C \cdot \theta'$ we have $C \cdot \theta\sigma = C \cdot \theta'\sigma$. Subsequently we must restore the property that in the σ-instance of the OBDD each formula is represented by at most one node.

An OBDD labelled literal $\mathcal{B}\colon L$ is equal up to renaming to $\mathcal{B}'\colon L'$ if the literals L and L' are equal up to the renaming σ and there exists a renaming ρ such that the OBDDs $\mathcal{B}$ and $\mathcal{B}\sigma\rho$ are identical.

The inference rules of OBDD labelled unit superposition are identical to tree labelled unit superposition in Definition 5.10, where tree labels are replaced with OBDD labels and merging and superposition inferences create the disjunction and conjunction of the OBDD labels of the premises, respectively.

As we have discussed, keeping a tree labels in the normal form of an OBDD is

justified by the notion of UST-redundancy, hence all other definitions for the UST-saturation process remain valid. We have also mentioned that UST-redundancy covers choosing only one representative of all OBDD labelled literals equal up to renaming. With a slightly modified proof the OBDD labelled merging inference is shown to be an instance of simplification with UST-redundancy elimination. Consequently, in particular an Inst-saturation process with OBDD labelled unit superposition is fair and a refutationally complete procedure.

Let us discuss our running example, where tree labelled unit superposition leads to non-termination, while OBDD labelled unit superposition terminates.

Example 5.8. We notice that the equation $f(x, y) \simeq f(y, x)$ is not orientable. If for the substitution $\theta = [a/x, b/y]$ and an ordering $\succ_{\mathrm{gr}}$ we have

$$f(x, y)\theta = f(a, b) \succ_{\mathrm{gr}} f(b, a) = f(y, x)\theta,$$

then with $\theta' = [b/x, a/y]$ we have

$$f(y, x)\theta' = f(a, b) \succ_{\mathrm{gr}} f(b, a) = f(x, y)\theta'.$$

Since we regard equations as unordered multisets, we must consider both literals $f(x, y) \simeq f(y, x)$ and $f(y, x) \simeq f(x, y)$. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be the initial tree labels of $L_1 = f(x, y) \simeq f(y, x)$ and $L_2 = f(u, v) \not\simeq g(z)$ from Example 5.3 on page 100.

We draw a first superposition inference between the literals with the equation in the first orientation

$$\frac{\begin{array}{cc} L_1 & L_2 \\ \mathcal{T}_1 \colon f(x, y) \simeq f(y, x) & \mathcal{T}_2 \colon f(u, v) \not\simeq g(z) \end{array}}{\mathcal{T}_1[u/x, v/y] \sqcap \mathcal{T}_2 \colon f(v, u) \not\simeq g(z)}\,[u/x, v/y] \qquad (*)$$

and merge the conclusion with the right premise

$$\frac{\begin{array}{cc} L_2 & (*) \\ \mathcal{T}_2 \colon f(u, v) \not\simeq g(z) & \mathcal{T}_1[u/x, v/y] \sqcap \mathcal{T}_2 \colon f(v, u) \not\simeq g(z) \end{array}}{\mathcal{T}_2 \sqcup \Big(\mathcal{T}_1[v/x, u/y] \sqcap \mathcal{T}_2[v/u, u/v]\Big) \colon f(u, v) \not\simeq g(z)}\,[v/u, u/v] \qquad (\ddagger)$$

The merged conclusion replaces the initially labelled literal $\mathcal{T}_2 \colon L_2$ and we use the abbreviation

$$\mathcal{T}_2' = \mathcal{T}_2 \sqcup \Big(\mathcal{T}_1[v/x, u/y] \sqcap \mathcal{T}_2[v/u, u/v]\Big).$$

A second superposition with the equation reversed

$$\frac{\begin{array}{cc} L_1 & L_2 \\ \mathcal{T}_1 \colon f(y,x) \simeq f(x,y) & \mathcal{T}_2' \colon f(u,v) \not\simeq g(z) \end{array}}{\mathcal{T}_1[v/x,u/y] \sqcap \mathcal{T}_2' \colon f(v,u) \not\simeq g(z)}\,[v/x,u/y] \qquad (\S)$$

results in the same literal in the conclusion with a different label. We merge it with the right premise

$$\frac{\begin{array}{cc} L_2 & (\S) \\ \mathcal{T}_2' \colon f(u,v) \not\simeq g(z) & \mathcal{T}_1[v/x,u/y] \sqcap \mathcal{T}_2' \colon f(v,u) \not\simeq g(z) \end{array}}{\mathcal{T}_2' \sqcup \Big(\mathcal{T}_1[u/x,v/y] \sqcap \mathcal{T}_2'[v/u,u/v]\Big) \colon f(u,v) \not\simeq g(z)}\,[v/u,u/v]. \qquad (\|)$$

We spell out the label of the conclusion of the proof as

$$\mathcal{T}_2 \sqcup \Big(\mathcal{T}_1[v/x,u/y] \sqcap \mathcal{T}_2[v/u,u/v]\Big) \sqcup \Big(\mathcal{T}_1[u/x,v/y] \sqcap \Big(\mathcal{T}_2[v/u,u/v] \sqcup \Big(\mathcal{T}_1[u/x,v/y] \sqcap \mathcal{T}_2\Big)\Big)\Big).$$

Let us abbreviate this labelled conclusion as

$$\mathcal{T}_2^{(\|)} \colon L_2 = \mathcal{T}_2^1 \sqcup \Big(\mathcal{T}_1^{-1} \sqcap \mathcal{T}_2^{-1}\Big) \sqcup \Big(\mathcal{T}_1^1 \sqcap \Big(\mathcal{T}_2^{-1} \sqcup \Big(\mathcal{T}_1^1 \sqcap \mathcal{T}_2^1\Big)\Big)\Big) \colon f(u,v) \not\simeq g(z)$$

using the superscript ^{1} to denote the substitutions [] and $[u/x,v/y]$ as well as $^{-1}$ for $[v/x,u/y]$ and $[v/u,u/v]$. In this notation the labelled literal $\mathcal{T}_2' \colon L_2$, which is the conclusion of (‡), is

$$\mathcal{T}_2' \colon L_2 = T_2^1 \sqcup \Big(T_1^{-1} \sqcap T_2^{-1}\Big) \colon f(u,v) \not\simeq g(z).$$

We can see that the label T_2' is contained in the top-level disjunction of the label $\mathcal{T}_2^{(\|)}$, hence the it is UST-redundant and a merging inference results in the labelled literal $\mathcal{T}_2^{(\|)} \colon L_2$.

The corresponding set label is

$$\Big\{\mathcal{T}_2^1, \mathcal{T}_2^{-1}, \mathcal{T}_1^1, \mathcal{T}_1^{-1}\Big\}.$$

It is now necessary to repeat inferences (∗) and (§) for the merged labelled literal $\mathcal{T}_2^{(\|)} \colon f(u,v) \not\simeq g(z)$. We note that for the substitutions applied to $\mathcal{T}_2$ in

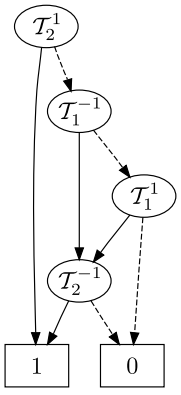
the merging, we have:

$$\begin{aligned}\mathcal{T}_1^{-1}[v/u, u/v] &= \mathcal{T}_1^1\\ \mathcal{T}_1^1[v/u, u/v] &= \mathcal{T}_1^{-1}\\ \mathcal{T}_2^{-1}[v/u, u/v] &= \mathcal{T}_2^1\\ \mathcal{T}_2^1[v/u, u/v] &= \mathcal{T}_2^{-1}.\end{aligned}$$

The label of the conclusion has the structure of $\mathcal{T}_2^{(\|)}$, where $\mathcal{T}_2^1$ is substituted by $\mathcal{T}_2^{(\|)}$ and $\mathcal{T}_2^{-1}$ by $\mathcal{T}_2^{(\|)}$ with the substitution $[v/u, u/v]$ applied.

$$\begin{aligned}\mathcal{T}_2^1 \sqcup \left(\mathcal{T}_1^{-1} \sqcap \mathcal{T}_2^{-1}\right) \sqcup \left(\mathcal{T}_1^1 \sqcap \mathcal{T}_2^{-1}\right) \sqcup\\ \left(\mathcal{T}_1^{-1} \sqcap \left(\mathcal{T}_2^{-1} \sqcup \left(\mathcal{T}_1^1 \sqcap \mathcal{T}_2^1\right) \sqcup \left(\mathcal{T}_1^{-1} \sqcap \mathcal{T}_2^1\right)\right)\right) \sqcup\\ \left(\mathcal{T}_1^1 \sqcap \left(\mathcal{T}_2^{-1} \sqcup \left(\mathcal{T}_1^1 \sqcap \mathcal{T}_2^1\right) \sqcup \left(\mathcal{T}_1^{-1} \sqcap \mathcal{T}_2^1\right)\right)\right)\end{aligned}$$

The set label does not change as no new closures are added in the tree label. We discard the conclusion, since there is already a literal equal up to renaming.
As the tree labels of the conclusion and $\mathcal{T}_2^{(\|)}$ and have a different structure from the way they were built up during the inferences, they are not equal up to renaming and we would continue with inferences, obtaining ever-growing labels in the conclusion.
If we, however, transform both labels to an OBDD using the same ordering, we obtain the relatively simple OBDD shown to the right. Just as for set labels we do not need to generate further inferences from here.



The features exhibited by the different label structures in the example and preempting results from the evaluation in a later chapter strongly suggest a hybrid approach to labelled unit superposition. The success of set and tree labelled superposition in practice is thanks to the low complexity of the operations on their labels during superposition and merging. However, both set and tree labelling do miss a significant number of problems classified as simple, respectively due to the weaker approximate elimination of redundancy in the flat set structure and the lack of a normal form in the Boolean tree structure. Since OBDD labelled

unit superposition trails behind the two simple label structures in the overall experimental results, we are looking for label structures, which are efficient in practice, with both a normal form and precise redundancy elimination.

However, we find some discouraging facts in this endeavour. The formulae of tree labels are monotone Boolean formulae and some problems in propositional logic are computationally easier if the formulae do not contain negations. Unfortunately, deciding logical equivalence of monotone Boolean formulae is as hard as if the formulae contained negations: Reith [2003] proves **coNP**-completeness. While it was conjectured there that computing DNFs of monotone Boolean formulae should be easier, we have already mentioned that Goldsmith et al. [2005] disprove the conjecture and have shown that computing DNFs of monotone Boolean formulae is as hard as computing DNFs of Boolean formulae with negations.

Unless $\mathbf{P} = \mathbf{NP}$, there is probably no generally efficient solution to decide if the formulae of two tree labels are logically equivalent. However, there are certainly a number of directions to investigate in further work, in order to obtain more efficient labels in a normal form.

For OBDD labelled unit superposition it remains to conceive good heuristics, exploiting the particular background of labels, to obtain variable orderings producing small and tractable shared OBDDs. A heuristic variable ordering has to be able to handle σ-instantiation of an OBDD in an efficient way. In a merging inference a renaming σ is applied to the label of the right premise, while in a superposition inference σ is an arbitrary substitution and applied to both premises. In both cases the variable ordering may be violated in the OBDD label of the conclusion, making a reordering necessary. Even worse, the substitution from a superposition inference can make two previously distinct variables equal, thus resulting in OBDDs which are not reduced.

A smart heuristic variable ordering would produce small OBDDs and minimise the impact of reordering the OBDDs after substitutions. However, we have not taken up the challenge to define such heuristics in the present work. Our implementation simply orders closures by the sequence they are inserted into the shared OBDD.

There have been many variants of OBDDs introduced in the literature and further work should evaluate some of these approaches, although due to results mentioned above one cannot expect to find a generally efficient procedure. Future work has to identify suitable candidates from the "alphabet soup" of acronyms

[Bryant, 1995] in the context of literal labels and to evaluate them on real problems. Notable variants of OBDDs change assumptions such as using a different function decomposition than the Shannon decomposition [Drechsler and Becker, 1998, Kebschull et al., 1992], relaxing the ordering requirement or the reduction requirement as in zero-suppressed decision diagrams (ZDDs) Minato [1993].

The representation of Boolean formulae is also a relevant topic in areas like logic synthesis and verification. One could expect to find promising approaches that are adaptable for our purposes there; it might be useful to study techniques on Boolean circuits [Abdulla et al., 2000, Andersen and Hulgaard, 2002], in particular verification and minimisation.

Another approach from model checking is described in Damm et al. [2007] and could provide an efficient algorithm for checking logical equivalence of formulae of tree labels. Boolean formulae with linear arithmetic are encoded as AND-Inverter Graphs (AIGs) and in a hierarchical method, first quick checks for equivalence are done with learnt test sets, if those are inconclusive, the non-canonical AIGs are compared and ultimately an SMT solver is invoked.

Since the problem of deciding equivalence of monotone Boolean formula can be reduced to a satisfiability problem and the Inst-Gen method modularly includes a ground solver, a final attempt could be to delegate the task to this solver.

We have motivated and presented set, tree and OBDD labelled unit superposition and we have discussed their main features on our running example. In Chapter 8 we give a quantitative comparison of the three labelling approaches. To conclude the chapter we look at the Bernays-Schönfinkel fragment of first-order logic and the decision procedure property of the labelled unit superposition calculi.

5.5 Deciding the Bernays-Schönfinkel Fragment

In the Bernays-Schönfinkel or "effectively propositional" fragment of first-order logic, all function symbols are of arity zero, in other words all terms are constants. For this reason, one also speaks of function-free clause logic and due to the lack of function symbols, every clause has only a finite number of instances.

No restriction is placed on predicate symbols P, which are translated to function symbols f_P in our pure equational approach. However, since we do not permit such translated predicate symbols f_P to occur in substitutions, only constants

remain and our pure equations nevertheless have a finite number of instances.

This property makes instantiation-based methods a natural choice and most methods decide this fragment. In this section we show that set and OBDD labelled unit superposition are decision procedures for the Bernays-Schönfinkel fragment, while tree labelled unit superposition not necessarily is.

As set and OBDD labels are in a normal form, we can show that there is only a finite number of labelled literals in a given signature, which makes the calculus terminate after at most a finite number of steps, since we consider only one representative of all labelled literals equal up to renaming. However, as demonstrated on Example 5.8, tree labels can grow ad infinitum.

Theorem 5.5. *Inst-Gen-Eq with set labelled unit superposition is a decision procedure for the Bernays-Schönfinkel fragment of first-order logic with equality.*

Proof. We have a finite signature Σ of function symbols, all of which are of arity zero. Hence, there is a finite number of terms and a finite number of substitutions, which are not equal up to renaming. Thus for a given set of clauses S the number of ground and non-ground closures, which are instances of S and not equal up to renaming, is finite. The power set of this finite set of closures is finite and hence there is a finite set of possible set labels. Since the set of selected literals is finite, the USL-saturation process $\{\langle \mathcal{N}L^i, \mathcal{D}L^i \rangle\}_{i=1}^{\infty}$ contains a finite set of set labelled literals which are not equal up to renaming. A USL-saturation process stops after a finite number of inference steps, when the USL-saturation is reached.

A fair Inst-saturation process $\left\{\left\langle S^i, I^i_\bot, \mathrm{sel}^i \right\rangle\right\}_{i=1}^{\infty}$ eventually makes all persistent conflicts in a set of closures redundant. There is only a finite number of ground closures, which are ground instances of S, and hence the number of persistent conflicts is finite, therefore in a finite number of steps in the Inst-saturation process all persistent conflicts are redundant. Since ground satisfiability modulo equality is decidable and USL-saturation can be achieved in a finite number of steps in the USL-saturation process, the fair Inst-saturation process with set labelled unit superposition stops after a finite number of steps with either the result "unsatisfiable" or "satisfiable". □

We prove a variant of this theorem for OBDD labelled unit superposition.

Theorem 5.6. *Inst-Gen-Eq with OBDD labelled unit superposition is a decision procedure for the Bernays-Schönfinkel fragment of first-order logic with equality.*

Proof. From the proof of the previous theorem we know that the number of ground and non-ground closures, which are instances of clauses in S and not equal up to renaming, is finite. There is only a finite number of Boolean functions of n variables and the number of possible OBDD labels is finite. As before, the UST-saturation process $\{\langle \mathcal{N}T^i, \mathcal{D}T^i \rangle\}_{i=1}^{\infty}$ is finite in every state if we consider equality of OBDD labelled literals up to renaming. Hence, a UST-saturation process with OBDD labelled unit superposition inferences stops after a finite number of steps.

Since Inst-saturation with OBDD labelled unit superposition is fair, by the same argument as above, this Inst-saturation process stops after a finite number of steps with a result of "unsatisfiable" or "satisfiable". □

We note that the last theorem extends to any other labelled superposition calculus that is a variant of tree labelled unit superposition such that equality up to renaming of labels is considered not on the shape of tree labels but on logical equivalence of formulae of tree labels.

Chapter 6

Exploiting Unit Clauses

The Inst-Gen-Eq calculus as presented so far is sound and complete for any set of first-order clauses. In the previous chapter we have introduced a labelling that allows us to simplify literals by merging variants. However, in the context of equational reasoning stronger simplification inferences based on rewriting are possible and necessary for efficiency in practice.

We now turn to mechanisms for simplification that are only applicable for certain literals, due to the following observation. A unit clause in the input clause set S contains exactly one literal L. For the ground abstraction $S\bot$ to be satisfiable, the ground literal $L\bot$ must be true in every model $I_\bot$. Therefore, the literal L is always selected and relevant for the saturation process regardless of the model $I_\bot$.

Taking the special status of unit clauses into account is an important step, since the vast majority of problems from applications does contain unit clauses. For instance, about 97% of the problems in CNF form in the TPTP benchmark library have at least one unit clause. Many algebraic properties like associativity and commutativity are expressed in terms of unit equations, and treating in particular those equations in a special way can provide a dramatic increase in performance.

Simplifications with unit clauses that we are presenting in this chapter are not possible with literals selected in non-unit clauses. Since in the incremental instantiation process the selection function changes, the equational model induced in the saturation process on literals can change. Hence, an equation that held once can be withdrawn later and this would require backtracking the simplification inferences with that equation. The necessary effort of undoing simplifications

with non-unit clauses makes it not worthwhile considering such inferences.

In the Section 6.1 we first review the main features from the completeness proof in Chapter 4 that enable treating literals from unit clauses specially. We then show in Section 6.2 that proofs only with literals from unit clauses subsume certain proofs from non-unit literals, thus leading to subsumption of literals by literals from unit clauses. Finally, in Section 6.3 we introduce a demodulation inference rule as in paramodulation-based reasoning to simplify literals.

6.1 Literals from Unit Clauses

The US-saturation process on literal closures $\{\langle \mathcal{N}^i, \mathcal{D}^i \rangle\}_{i=1}^{\infty}$ as introduced in Definition 4.4 on page 50 contains in each state a pair $\langle \mathcal{N}^i, \mathcal{D}^i \rangle$, where $\mathcal{N}^i$ is a set of literal closures and $\mathcal{D}^i$ is a set of positive literal closures. Let us highlight how this definition provides hooks to justify the simplification inferences to be presented in this chapter.

Inst-saturation of a set of clauses S in Definition 4.12 on page 53 means that there is no contradiction $\square$ in the US-saturation $\langle \mathcal{N}, \mathcal{D} \rangle^{\text{sat}}$ of the S-relevant literal closures (Definition 4.7 on page 51). For a literal closure $L \cdot \theta$ to be in $\mathcal{N} = \mathcal{N}^1$ it is necessary that there is a clause C with $\text{sel}(C) = L$, hence $\mathcal{N}$ depends on the selection function, which may change in the incremental Inst-saturation process, such that $\text{sel}(C) = L'$ for some $L' \neq L$.

On the other hand, a literal closure is in $\mathcal{D} = \mathcal{D}^1$ regardless of the selection function. These literal closures are required to be consequences of S, that is, every $(l \simeq r) \cdot \theta \in \mathcal{D}$ must satisfy $S \models (l \simeq r)\,\theta$. The set $\mathcal{D}^i$ in a state i of the US-saturation process is modified by adding conclusions of unit superposition inferences with both premises in $\mathcal{D}^i$ or removing literal closures US-redundant in $\mathcal{D}^i$, hence the set $\mathcal{D}^{i+1}$ in every successor state of the US-saturation process remains independent of $\mathcal{N}^i$ and the selection function. By induction we conclude that the persistent closures $\mathcal{D}^{\infty}$ (Definition 4.5 on page 50) and the set $\mathcal{D}^{\infty} \setminus \mathcal{R}_{\text{US}}(\mathcal{D}^{\infty})$, which is a subset of the US-saturation $\langle \mathcal{N}, \mathcal{D} \rangle^{\text{sat}}$, are independent of the selection function for S.

The partition of the US-saturation process into sets $\mathcal{N}$ and $\mathcal{D}$ makes it possible to simplify literal closures in $\mathcal{N}$ with literal closures from $\mathcal{D}$, such that the simplifications also hold if the selection function changes and literal closures are added to and removed from $\mathcal{N}$ in the incremental instantiation process.

Inst-saturation allows any S-relevant literal closure $(l \simeq r) \cdot \theta$ to be in $\mathcal{D}$ if $S \models (l \simeq r)\,\theta$. Since this entailment is a complex problem to decide, in practice we take $\mathcal{D}$ to contain literals for which the entailment is trivial: let $C = l \simeq r \in S$ be a unit clause, then for every ground instance $(l \simeq r) \cdot \theta$ we have $S \models (l \simeq r)\,\theta$.

We can lift the above remarks from literal closures to constrained literals and labelled literals, since the arguments also hold in the contexts of the USD-saturation process on constrained literals and the set and tree labelled USL- and UST-saturation processes. For labelled unit superposition calculi we need an initial labelling for each literal. Since the Inst-saturation process allows to add any clause that follows from the clause set S, we do not lose generality if we choose for the set and tree labelled literals $\mathcal{L}\colon l \simeq r$ and $\mathcal{T}\colon l \simeq r$ the labels $\mathcal{L} = \{(l \simeq r) \cdot \theta\}$ and $\mathcal{T} = (l \simeq r) \cdot \theta$ as the initial set and tree labels, respectively. We can implicitly add the unit clause $l \simeq r$ to S.

There is an important restriction on literals from unit clauses that the following simplification techniques impose. Neither subsumption by literals from unit clauses nor demodulation is compatible with non-empty dismatching constraints on unit clauses. We therefore assume unit clauses to be always unconstrained and do not add a constraint when instantiating. We give reasons for the incompatibility of dismatching constraints in the context of the simplification techniques.

Dismatching constraints on unit clauses do not provide the great advantage they have on non-unit clauses. In the instance $C\sigma$ of a non-unit clause C the selection function does not necessarily select the instance $L\sigma$ of the selected literal L in C. In fact, it is precisely the intention of the Inst-saturation process to trigger such adaptions of the selection function by refining the ground abstraction on persistent conflicts by generating proper instances of clauses. If the clause C is instantiated to $C\sigma$, the constraint is extended with the dismatching constraint D_σ of the instantiating substitution, such that all closures $L \cdot \sigma\theta$ are eliminated from the US-saturation process. If the selection function does not select the literal $L\sigma$ in the clause instance $C\sigma$, then the literal closures $L\sigma \cdot \theta$ are not relevant and not in the US-saturation, either. In this way no ground literal $L\sigma\theta$ is represented by any closure in the US-saturation process. Inferences requiring the grounding $L\sigma\theta$ are then redundant and blocked by the dismatching constraint D_σ on C.

The situation is different for unit clauses. Since there is only one literal to choose in the selection function, the literal instance $L\sigma$ is selected in every σ-instance of the unit clause L. Blocking by dismatching constraints as for literals

from non-unit clauses do not occur and it is expected that an implementation can efficiently handle the duplication of ground literals represented by a literal and an instance. Therefore, losing dismatching constraints on unit literals is not a great issue.

6.2 Subsumption by Literals from Unit Clauses

Unit clauses have special status, since as facts they are true in every model. Conclusions from facts also hold in every model and therefore it is not necessary to derive those conclusions again from literals selected in non-unit clauses. We formalise this idea and justify it in the framework of US-redundancy.

A state $\langle \mathcal{N}^i, \mathcal{D}^i \rangle$ of the US-saturation process can have a literal closure $L \cdot \theta$ in the set $\mathcal{D}^i$ and a literal closure $L' \cdot \theta'$ in $\mathcal{N}^i$, which represent the same ground literal $L\theta = L'\theta'$. We show that the literal closure $L \cdot \theta \in \mathcal{D}^i$ makes the literal closure $L' \cdot \theta' \in \mathcal{N}^i$ US-redundant. Therefore labelled unit superposition calculi can eliminate a set labelled or tree labelled literal in $\mathcal{N}L^i$ or $\mathcal{N}T^i$ if a variant of the literal is in $\mathcal{D}L^i$ or $\mathcal{D}T^i$, respectively. Merging inferences are only necessary between literal variants in $\mathcal{N}L^i$ or $\mathcal{N}T^i$, a literal in $\mathcal{D}L^i$ or $\mathcal{D}T^i$ subsumes its variants in $\mathcal{N}L^i$ or $\mathcal{N}T^i$, resulting a shorter labels.

By Definition 4.2 on page 47, the total ordering $\succ_{\mathrm{l}}$ on literal closures allows to arbitrarily choose either $L \cdot \theta \succ_{\mathrm{l}} L' \cdot \theta'$ or $L' \cdot \theta' \succ_{\mathrm{l}} L \cdot \theta$ if neither $L\theta \succ_{\mathrm{gr}} L'\theta'$ nor $L'\theta' \succ_{\mathrm{gr}} L\theta$. We can therefore make the literal closure $L \cdot \theta \in \mathcal{D}^i$ smaller than the literal closure $L' \cdot \theta' \in \mathcal{N}^i$ and exploit US-redundancy in Definition 4.3 on page 48, where a literal closure is redundant if it follows from smaller literal closures. Since the literals in each $\mathcal{D}^i$ are independent of the actual selection function, every literal closure $L' \cdot \theta' \in \mathcal{N}^i$ is always redundant.

Definition 6.1. Let $\langle \mathcal{N}, \mathcal{D} \rangle$ be a partition of ground literal closures. Let $\succ_{\mathrm{l}}^{\langle \mathcal{N}, \mathcal{D} \rangle}$ be an arbitrary total well-founded extension of $\succ_{\mathrm{gr}}$ from ground literals to ground literal closures such that $L \cdot \theta \succ_{\mathrm{l}}^{\langle \mathcal{N}, \mathcal{D} \rangle} M \cdot \rho$ if

(i) $L\theta \succ_{\mathrm{gr}} M\rho$ or

(ii) $L\theta = M\rho$ and $L \cdot \theta \in \mathcal{N}$ and $M \cdot \rho \in \mathcal{D}$.

Now US-redundancy can use the refined and stronger ordering $\succ_{\mathrm{l}}^{\langle \mathcal{N}, \mathcal{D} \rangle}$ on literal closures and we have the following theorem.

Theorem 6.1. *Let $(l \simeq r) \cdot \theta \in \mathcal{N}$ and $(s \simeq t) \cdot \rho \in \mathcal{D}$ be literal closures such that $l \simeq r$ and $s \simeq t$ are equal up to renaming and $(l \simeq r)\,\theta = (s \simeq t)\,\rho$. The first literal closure $(l \simeq r) \cdot \theta$ is US-redundant in $\mathcal{N} \cup \mathcal{D}$.*

Proof. Let R be a rewrite system oriented with respect to $\succ_{\mathrm{gr}}$ such that the substitution θ from the literal closure $(l \simeq r) \cdot \theta$ is irreducible. There is a renaming τ such that $l \simeq r = (s \simeq t)\,\tau$ and we have $(s \simeq t)\,\rho = (l \simeq r)\,\theta = (s \simeq t)\,\tau\theta$. The literal closures $(l \simeq r) \cdot \theta$ and $(s \simeq t) \cdot \rho$ are well-defined, hence $\mathrm{rng}\,(\tau\theta) = \mathrm{rng}\,(\rho)$ and we conclude irreducibility of ρ in R. Further, we have $(l \simeq r)\,\theta \models (s \simeq t)\,\rho$ and $(l \simeq r) \cdot \theta \succ_{\mathrm{l}}^{\langle \mathcal{N}, \mathcal{D} \rangle} (s \simeq t) \cdot \rho$, therefore

$$R \cup \mathrm{irred}_R\left(\left\{L \cdot \theta \in \mathcal{N} \cup \mathcal{D} \mid (l \simeq r) \cdot \theta \succ_{\mathrm{l}}^{\langle \mathcal{N}, \mathcal{D} \rangle} L \cdot \theta\right\}\right) \models (l \simeq r)\,\theta.$$

As a consequence $(l \simeq r) \cdot \theta$ is US-redundant in $\mathcal{N} \cup \mathcal{D}$. □

We lift the theorem from literal closures to constrained first-order literals.

Theorem 6.2. *Let $l \simeq r \mid D \in \mathcal{N}D$ and $s \simeq t \in \mathcal{D}D$ be a constrained and an unconstrained literal, where $l \simeq r$ and $s \simeq t$ are equal up to renaming. The constrained literal $l \simeq r \mid D$ is USD-redundant in $\mathcal{N}D \cup \mathcal{D}D$.*

Proof. Let $(l \simeq r) \cdot \theta \in \mathrm{Cl}\,(l \simeq r \mid D)$ be a literal closure of the constrained literal, then there is a literal closure $(s \simeq t) \cdot \rho \in \mathrm{Cl}\,(s \simeq t)$ of the unconstrained literal such that $(l \simeq r)\,\theta = (s \simeq t)\,\rho$. By Theorem 6.1 the latter literal closure makes the former US-redundant. Since every literal closure $(l \simeq r) \cdot \theta \in \mathrm{Cl}\,(l \simeq r \mid D)$ is US-redundant in this way, the constrained literal $l \simeq r \mid D$ is USD-redundant. □

As we have noted before, unit clauses must have an empty dismatching constraint. For a literal closure $(l \simeq r) \cdot \theta$ to be US-redundant we have to take into account all ground rewrite systems R that do not reduce a term in $\mathrm{rng}\,(\theta)$. Then $(l \simeq r) \cdot \theta$ is US-redundant if it follows from literal closures with irreducible substitutions in R. In order to satisfy this irreducibility requirement we take the unit clause $s \simeq t$ to be unconstrained and, as shown in the proof, the closures of the unconstrained literal are sufficient to make $l \simeq r \mid D$ USD-redundant.

Let us give an example to illustrate the benefits of subsumption by literals from unit clauses in the unlabelled unit superposition calculus.

Example 6.1. The following set of clauses is unsatisfiable.

$$\underline{f(x, f(y, z)) \simeq a \ \vee \ z \not\simeq a} \tag{1}$$

$$\underline{f(u, a) \simeq a} \tag{2}$$

$$\underline{f(a, a) \not\simeq a} \tag{3}$$

The ground abstraction is satisfiable and we select the first literals in each clause. We find a contradiction with the following unlabelled unit superposition proof from literals in unit clauses only.

$$\dfrac{\dfrac{\overset{L_2}{f(u, a) \simeq a} \qquad \overset{L_3}{f(a, a) \not\simeq a}}{a \not\simeq a}\,[a/u]}{\square} \tag{$*$}$$

After adding the ground instance of clause (2)

$$f(a, a) \simeq a \tag{4}$$

to the clause set, its ground abstraction becomes unsatisfiable.

There is another proof of a contradiction involving the non-unit clause (1).

$$\dfrac{\dfrac{\dfrac{\overset{L_2}{f(u, a) \simeq a} \qquad \overset{L_1}{f(x, f(y, z)) \simeq a}}{f(x, a) \simeq a}\,[u/y, a/z] \qquad \overset{L_3}{f(a, a) \not\simeq a}}{a \not\simeq a}\,[a/x]}{\square} \tag{$\dagger$}$$

The only relevant instance in this proof that is proper is

$$f(a, f(w, a)) \simeq a \ \vee \ a \not\simeq a. \tag{5}$$

The ground abstraction of clauses (1), (2), (3) and (5) is unsatisfiable.

Since (2) is a unit clause, its single literal $f(u, a) \simeq a$ is in $\mathcal{D}D$ and upon deriving the literal $f(x, a) \simeq a$ in proof ($\dagger$), which would be added to $\mathcal{N}D$, we have a variant of a unit literal in $\mathcal{D}D$, which is USD-redundant. We abandon proof ($\dagger$) at this point and do not need to generate clause (5).

Subsumption by literals from unit clauses finds proofs, where relevant instances are unit clauses. Such clauses are more useful for the ground solver than

instances of non-unit clauses, since they force a particular literals to be in a model of the ground abstraction.

If a contradiction is derived from literals from unit clauses only, every other contradiction is a variant and US-redundant. In this case the Inst-saturation process can terminate with the result "unsatisfiable". Since the instances of the literals at the leaves of a USD-proof are in a conflict and unsatisfiable together, we have found a set of unsatisfiable clause instances and the ground solver would fail to find a model of the ground abstraction. Therefore we can skip actually generating the instances and invoking the ground solver.

The subsumption by literals from unit clauses can be lifted to labelled calculi, where we do not eliminate proofs, but discard literals which are equal up to renaming instead of merging them.

Theorem 6.3. *Let $\mathcal{L}\colon l \simeq r \in \mathcal{N}L$ and $\{s \simeq t \cdot [] \colon s \simeq t\} \in \mathcal{D}L$ be two set labelled literals, where $l \simeq r$ and $s \simeq t$ are equal up to renaming. The first set labelled literal $\mathcal{L}\colon l \simeq r$ is USL-redundant in $\mathcal{N}L \cup \mathcal{D}L$.*

Proof. Let $l \simeq r \mid D$ be a constrained literal from $\mathrm{DLit}(\mathcal{L}\colon l \simeq r)$. Since the unit clause $s \simeq t$ is unconstrained, $\mathrm{DLit}(\{s \simeq t \cdot []\} : s \simeq t)$ contains only the unconstrained literal $s \simeq t$. By Theorem 6.2 each constrained literal $l \simeq r \mid D$ in $\mathrm{DLit}(\mathcal{L}\colon l \simeq r)$ is USD-redundant and therefore the set labelled literal $\mathcal{L}\colon l \simeq r$ is USL-redundant in $\mathcal{N}L \cup \mathcal{D}L$. □

Theorem 6.4. *Let $\mathcal{T} : l \simeq r \in \mathcal{N}T$ and $s \simeq t \cdot [] \colon s \simeq t \in \mathcal{D}T$ be two tree labelled literals, where $l \simeq r$ and $s \simeq t$ are equal up to renaming. The first tree labelled literal $\mathcal{T} : l \simeq r$ is USL-redundant in $\mathcal{N}T \cup \mathcal{D}T$.*

Proof. Let $l \simeq r \mid D$ be the constrained literal $\mathrm{DLit}(\mathcal{T} : l \simeq r)$. Since the unit clause $s \simeq t$ is unconstrained, $\mathrm{DLit}(s \simeq t \cdot [] \colon s \simeq t)$ is simply the unconstrained literal $s \simeq t$. By Theorem 6.2 the constrained literal $l \simeq r \mid D = \mathrm{DLit}(\mathcal{L}\colon l \simeq r)$ is USD-redundant and therefore the tree labelled literal $\mathcal{T} : l \simeq r$ is UST-redundant in $\mathcal{N}T \cup \mathcal{D}T$. □

While subsumption by unit literals is an efficient simplification by itself, it is most powerful when combined with the demodulation inference we describe next.

6.3 Demodulation

The concept of demodulation in paramodulation-based theorem proving was introduced as early as in Wos et al. [1967] and treated, for instance, by Bachmair and Ganzinger [1994] in a framework of redundancy elimination.

The idea of demodulation is to use certain equations to reduce literals to smaller literals in a way so that the reduced literal becomes redundant. However, equations in first-order clausal logic do not occur independently, but in clauses, which are disjunctions of literals. Hence a reduction by an equation has to be seen under the condition of the literals in a clause. Nevertheless, demodulations with equations from unit clauses hold unconditionally and are therefore particularly interesting. In practice demodulation is usually restricted to equations from unit literals.

The situation is no different for the instantiation-based Inst-Gen-Eq. We cannot use every equation in a state of the US-saturation process $\langle \mathcal{N}^i, \mathcal{D}^i \rangle$ for demodulation, since a change of the selection function can subsequently remove the equation and invalidate the reductions that have been made with it. Therefore we only consider equations in $\mathcal{D}$, since this set is independent of the selection function and thus demodulations hold unconditionally.

A demodulation inference rule with a non-proper matching substitution between the demodulating equation and the demodulated literal is already defined in the original Inst-Gen-Eq calculus of Ganzinger and Korovin [2004]. In this section we first present this non-proper demodulation and then extend it to proper demodulation, allowing for unrestricted matching substitutions. If the substitution in a demodulation inference is proper, it is necessary to generate instances of the demodulating equation and we prove completeness of proper demodulation in this way.

Let us begin with non-proper demodulation on literal closures.

Definition 6.2 (Non-proper Demodulation [Ganzinger and Korovin, 2004])**.**

Non-proper Demodulation

$$\frac{(l \simeq r) \cdot \sigma\theta \qquad (s[l'] \simeq t) \cdot \theta}{(s[r\sigma] \simeq t) \cdot \theta|_{\mathrm{var}(\{s[r\sigma],t\})}}\,(\sigma) \qquad \frac{(l \simeq r) \cdot \sigma\theta \qquad (s[l'] \not\simeq t) \cdot \theta}{(s[r\sigma] \not\simeq t) \cdot \theta|_{\mathrm{var}(\{s[r\sigma],t\})}}\,(\sigma)$$

where

(i) $l\sigma = l'$,

(ii) σ is not proper

(iii) l' is not a variable,

(iv) $l\sigma\theta \succ_{\mathrm{gr}} r\sigma\theta$,

(v) $(s[l'] \simeq t)\,\theta \succ_{\mathrm{gr}} (l \simeq r)\,\sigma\theta$,

(vi) $\mathrm{var}(r) \subseteq \mathrm{var}(l)$,

(vii) $\mathrm{var}(\{l, r\}) \cap \mathrm{var}(\{s[l'], t\}) = \emptyset$.

A demodulation inference is similar in structure to a superposition inference (Definition 4.1 on page 45) with some notable differences. As before, the left premise is a positive equation, whereas the right premise is either a positive or a negative equation. However, the substitution σ is a matching, not a unifier, between the left-hand side of the equation $l \simeq r$ in the left premise and a subterm l' of the right premise. No substitution is applied to the right premise. In this first variant of demodulation the substitution σ must not be proper, that is, all terms in the range of σ are variables. We remind ourselves that a non-proper substitution is not necessarily a renaming by considering the example of the substitution $[z/x, z/y]$.

The equation in the left premise must be ordered according to $\succ_{\mathrm{gr}}$, while in contrast to superposition inferences $t\theta \succ_{\mathrm{gr}} s[l']\theta$ is permitted in the right premise. This is a desirable property, since a simplification inference rule should be applicable as often as possible, while inferences like superposition, which add a new literal or literal closure, should be limited as much as possible.

Since a closure $C \cdot \theta$ is only well-defined if $\mathrm{var}(C) = \mathrm{dom}(\theta)$ we have to restrict the substitution θ in the conclusion to the actual variables of C. Further, no substitution is applied to the right premise and new variables must not be introduced into the conclusion, hence the condition $\mathrm{var}(r) \subseteq \mathrm{var}(l)$.

Lemma 6.5 (Ganzinger and Korovin [2004])**.** *Non-proper demodulation is a simplification rule, that is, the left premise and the conclusion make the right premise US-redundant.*

Proof. Let R be a ground rewrite system oriented with respect to $\succ_{\text{gr}}$ such that θ is irreducible in R. The substitution σ is non-proper due to (ii), therefore $\operatorname{rng}(\sigma\theta) = \operatorname{rng}(\theta)$ and the composed substitution $\sigma\theta$ is also irreducible in R. Let

$$\mathcal{L} = \left\{(l \simeq r) \cdot \sigma\theta,\ (s[r\sigma] \simeq t) \cdot \theta|_{\operatorname{var}(\{s[r\sigma],t\})}\right\}$$

consist of the left premise and the conclusion of a demodulation of a positive right premise and we have

$$\operatorname{irred}_R(\mathcal{L}) = \mathcal{L}.$$

By condition (v) on the demodulation inference $(s[l'] \simeq t) \cdot \theta \succ_{\text{l}} (l \simeq r) \cdot \sigma\theta$ and we also have $(s[l'] \simeq t) \cdot \theta \succ_{\text{l}} (s[r\sigma] \simeq t) \cdot \theta|_{\operatorname{var}(\{s[r\sigma],t\})}$, because conditions (i) and (iv) lead to $l'\theta = l\sigma\theta \succ_{\text{gr}} r\sigma\theta$. The left premise and the conclusion are therefore smaller than the positive right premise in $\succ_{\text{l}}$ and

$$\operatorname{irred}_R\left(\mathcal{L}_{(s[l'] \simeq t) \cdot \theta \succ_{\text{l}}}\right) = \mathcal{L}.$$

By condition (i) $l\sigma = l'$ holds and hence

$$\left\{(l \simeq r)\,\sigma\theta,\ (s[r\sigma] \simeq t)\,\theta|_{\operatorname{var}(\{s[r\sigma],t\})}\right\} \models (s[l'] \simeq t)\,\theta.$$

Altogether we conclude

$$R \cup \operatorname{irred}_R\left(\mathcal{L}_{(s[l'] \simeq t) \cdot \theta \succ_{\text{l}}}\right) \models (s[l'] \simeq t)\,\theta.$$

By Definition 4.3 on page 48 the positive right premise is US-redundant with the left premise and the conclusion.

A similar proof holds for the negative right premise $(s[l'] \not\simeq t) \cdot \theta$. □

We can show that naively extending demodulation to non-proper matching substitutions σ by removing side condition (ii) fails.

Example 6.2 (Modified from Ganzinger and Korovin [2004][1])**.** Let us consider

[1] In their example literal closure (2) is $(g(f(x)) \simeq c) \cdot [d/x]$ and hence

$$(2) = (g(f(x)) \simeq c) \cdot [d/x] = g(f(d)) \simeq c \not\succ_{\text{gr}} g(f(d)) \simeq c = (g(x) \simeq c) \cdot [f(d)/x] = (1).$$

A demodulation inference between the literal closures (1) and (2) is not applicable, since side condition (iv) is violated. This ordering constraint in the demodulation inference holds in our modified example.

the set of literal closures

$$(g(x) \simeq c) \cdot [f(d)/x] \quad (1)$$
$$(g(g(f(y))) \simeq g(c)) \cdot [d/y] \quad (2)$$
$$(f(d) \simeq m) \cdot [] \quad (3)$$
$$(g(g(m)) \not\simeq g(c)) \cdot [] \quad (4)$$

A contradiction follows in a US-proof in the unit superposition calculus.

$$\dfrac{\dfrac{\dfrac{(2) \quad (g(g(f(y))) \simeq g(c)) \cdot [d/y] \qquad (3) \quad (f(d) \simeq m) \cdot []}{(g(g(m)) \simeq g(c)) \cdot []}\,[d/y] \qquad (4) \quad (g(g(m)) \not\simeq g(c)) \cdot []}{(g(c) \not\simeq g(c)) \cdot []}\,[]}{\square}\,[]$$

Instead of using the above proof, let us first simplify the literal closure (2) with a demodulation inference, despite the fact that the substitution σ is proper.

$$\dfrac{(1) \quad (g(x) \simeq c) \cdot [f(d)/x] \qquad (2) \quad (g(g(f(y))) \simeq g(c)) \cdot [d/y]}{(g(c) \simeq g(c)) \cdot []}\,[f(y)/x]$$

We have $\sigma\theta = [f(d)/x] = [f(y)/x][d/y]$ and all side constraints on the demodulation except (ii) are satisfied.

We omit the tautological literal closure obtained as the conclusion and also the right premise (2), wrongly assuming its US-redundancy.

$$(g(x) \simeq c) \cdot [f(d)/d] \quad (1)$$
$$(f(d) \simeq m) \cdot [] \quad (3)$$
$$(g(g(m)) \not\simeq g(c)) \cdot [] \quad (4)$$

We now fail to derive a contradiction, in fact we cannot perform any inference between the above literal closures. Since superposition into a variable in the right premise is not allowed, no inference applies between (1) and (3). The substitution of the literal closures blocks a possible inference between (1) and (4) and, finally, no subterms of (3) and (4) can be unified.

The reason for the failure to find the contradiction is that we have removed the demodulated literal closure (2) although it is not US-redundant. While we

have

$$g(f(d)) \simeq c \models g(g(f(d))) \simeq g(c)$$

and also

$$g(g(f(d))) \simeq g(c) \succ_{\mathrm{gr}} g(f(d)) \simeq c,$$

irreducibility of the substitution $[f(d)/x]$ in the demodulating literal closure does not hold.

Consider for instance the rewrite system R that contains the rule $f(d) \to m$, which is generated from literal closure (3) if R is a candidate model of the literal closures (1), (3) and (4) by Definition 4.13 on page 54. The substitution $[d/y]$ in the demodulated literal closure (2) is irreducible in R, but the substitution in the demodulating literal closure (1) can be reduced. Hence (1) is not in $\mathrm{irred}_R(\mathcal{L})$ and (2) is not US-redundant.

As the example shows, it is essential that the matching substitution is non-proper. In the proof of Lemma 6.5 irreducibility of the substitution $\sigma\theta$ in the left premise depends on the matching substitution σ not being proper.

Although non-proper demodulation reduces redundancy in the US-saturation process by some amount, the benefits are limited. If the demodulating equation contains variables, it cannot simplify ground terms in literals. Such equations cannot be used by the ground solver, either, since in the ground abstraction variables are mapped to the distinct constant $\bot$, which does not occur in the input. We therefore need to find ways to work around the restriction to non-proper matching substitutions.

One way to turn a demodulation with a proper substitution into a non-proper demodulation is by instantiating the demodulating equation. Since the candidate equations for demodulation inferences come from the set $\mathcal{D}^i$ in the US-saturation process $\{\langle \mathcal{N}^i, \mathcal{D}^i \rangle\}_{i=1}^{\infty}$, the equation $l \simeq r$ in each literal closure $(l \simeq r) \cdot \theta \in \mathcal{D}^i$ follows from the input clause set S, that is, $S \models l \simeq r$. We also have $S \models (l \simeq r)\,\sigma$ for each instance of $l \simeq r$. The US-saturation process starts with a fixed initial state $\langle \mathcal{N}, \mathcal{D} \rangle$ (Definition 4.12 on page 53), which consists of the S-relevant literal closures, whereas in the Inst-saturation (Definition 4.14 on page 59) process we may add any set of clauses N to S, if $S \models N$. Thus, in order to effectively add an instance $(l \simeq r)\,\sigma \cdot \theta$ of a literal closure $(l \simeq r) \cdot \sigma\theta$ to $\mathcal{D}$ in the US-saturation process we add the unit clause $(l \simeq r)\,\sigma$ to S in the Inst-saturation process.

We modify the demodulation inference rule by requiring a suitable instance

of the demodulating equation to be generated and dropping the restriction to non-proper matching substitutions. The following inference rule first derives an instance $(l \simeq r)\,\sigma \cdot \theta$ of the demodulating literal closure and then performs a non-proper demodulation inference with the instantiated demodulating literal closure. The unit clause $(l \simeq r)\,\sigma$ of the instantiated literal closure is to be added to the Inst-saturation process, thus the literal closure $(l \simeq r)\,\sigma \cdot \theta$ becomes relevant in the US-saturation process.

Definition 6.3 (Demodulation with Instantiation, Literal Closures)**.**

Demodulation

$$\frac{\dfrac{(l \simeq r) \cdot \sigma\theta}{(l \simeq r)\,\sigma \cdot \theta} \qquad (s[l'] \simeq t) \cdot \theta}{(s[r\sigma] \simeq t) \cdot \theta|_{\mathrm{var}(\{s[r\sigma],t\})}}\,(\sigma) \qquad \frac{\dfrac{(l \simeq r) \cdot \sigma\theta}{(l \simeq r)\,\sigma \cdot \theta} \qquad (s[l'] \not\simeq t) \cdot \theta}{(s[r\sigma] \not\simeq t) \cdot \theta|_{\mathrm{var}(\{s[r\sigma],t\})}}\,(\sigma)$$

where

(i) $l\sigma = l'$,

(ii) l' is not a variable,

(iii) $l\sigma\theta \succ_{\mathrm{gr}} r\sigma\theta$,

(iv) $(s[l'] \simeq t)\,\theta \succ_{\mathrm{gr}} (l \simeq r)\,\sigma\theta$,

(v) $\mathrm{var}(r) \subseteq \mathrm{var}(l)$,

(vi) $\mathrm{var}(\{l, r\}) \cap \mathrm{var}(\{s[l'], t\}) = \emptyset$.

We decompose the substitution of the left premise into σ and θ in order to factor out the matching substitution σ. After instantiating the left premise with σ there is a non-proper demodulation inference possible with an empty matching substitution between the instantiated left-hand side of the equation $l\sigma$ and the subterm l', since by condition (i) $l\sigma = l'$ already holds.

The side conditions in the instantiating demodulation inference rule are identical to the side conditions of the non-proper demodulation inference rule in Definition 6.2, except for the missing (ii). We prove that simplification with proper demodulation is covered by US-redundancy.

Lemma 6.6. *Proper demodulation with instantiating the demodulating equation is a simplification rule, that is, the instantiated demodulating equation and the conclusion make the right premise US-redundant.*

Proof. We have $l\sigma = l'$ and therefore a matching substitution $\sigma' = []$ between the left-hand side $l\sigma$ of the instantiated literal closure $(l \simeq r)\,\sigma \cdot \theta$ and the subterm l' of $(s[l'] \simeq t) \cdot \theta$. Since this σ' is non-proper, the demodulation inference between $(l \simeq r)\,\sigma \cdot \theta$ and $(s[l'] \simeq t) \cdot \theta$ is a non-proper demodulation as in Definition 6.2. Due to Lemma 6.6 the conclusion $(s[r\sigma] \simeq t) \cdot \theta|_{\mathrm{var}(\{s[r\sigma],t\})}$ and the instantiated left premise $(l \simeq r)\,\sigma \cdot \theta$ make the right premise $(s[l'] \simeq t) \cdot \theta$ US-redundant. □

Example 6.3. We resume Example 6.2 and repeat the proper demodulation inference, this time instantiating the demodulating equation.

$$\dfrac{\dfrac{\overset{(1)}{(g(x) \simeq c) \cdot [f(d)/x]}}{(g(f(y)) \simeq c) \cdot [d/y]}\,[f(y)/x] \qquad \overset{(2)}{(g(g(f(y))) \simeq g(c)) \cdot [d/y]}}{(g(c) \simeq g(c)) \cdot []}\,[]$$

In the inference the substitutions are $\sigma = [f(y)/x]$ and $\theta = [d/y]$. We discard the tautological conclusion, add the instance of the demodulating equation as $(1')$ and remove the US-redundant literal closure (2).

$$(g(x) \simeq c) \cdot [f(d)/x] \tag{1}$$
$$(g(f(y)) \simeq c) \cdot [d/y] \tag{1'}$$
$$(f(d) \simeq m) \cdot [] \tag{3}$$
$$(g(g(m)) \not\simeq g(c)) \cdot [] \tag{4}$$

A contradiction can be derived with inferences from the unit superposition calculus from these literal closures.

$$\dfrac{\dfrac{\dfrac{\overset{(3)}{(f(d) \simeq m) \cdot []} \qquad \overset{(1')}{(g(f(z)) \simeq c) \cdot [d/z]}}{(g(m) \simeq c) \cdot []}\,[d/z] \qquad \overset{(4)}{(g(g(m)) \not\simeq g(c)) \cdot []}}{(g(c) \not\simeq g(c)) \cdot []}\,[]}{\square}\,[]$$

In Example 6.2 the instantiated demodulating equation $(1')$ was missing and no US-proof of a contradiction could be found. In the proof above all inferences can actually be regarded as demodulation inferences. In particular the demodulating equation $(1')$ is itself demodulated with the equation $(f(d) \simeq m) \cdot []$.

Since the substitutions in the instantiated demodulating literal closure $(1')$

and the right premise (2) are identical, the former is irreducible if the latter is. Since (2) follows from (1′), where the substitution is reducible if the substitution of (2) is, US-redundancy of (2) with (1′) follows.

We now lift the demodulation inference rule with instantiation from literal closures to first-order literals. It is important to note that demodulation of first-order literals is not compatible with dismatching constraints on demodulating equations.

As before, satisfying USD-redundancy requires an instance of the demodulating equation to be in the set $\mathcal{D}D^i$ of the USD-saturation process $\{\langle \mathcal{N}D^i, \mathcal{D}D^i \rangle\}_{i=1}^{\infty}$. For every constrained literal $l \simeq r \mid D \in \mathcal{D}D^i$ we have $S \models \mathrm{Unc}\,(l \simeq r \mid D)$ (Definition 4.33 on page 83) and hence also $S \models (l \simeq r)\,\sigma$ for any substitution σ. In analogy to the approach in the US-saturation process we can thus effectively add the required instance of the demodulating equation to $\mathcal{D}D$ in the USD-saturation process by adding the unit clause $(l \simeq r)\,\sigma$ to the set S in the Inst-saturation process.

Definition 6.4 (Proper Demodulation)**.**

Demodulation

$$\frac{\dfrac{l \simeq r}{(l \simeq r)\,\sigma} \qquad s[l'] \simeq t \mid D}{s[r\sigma] \simeq t \mid D}\,(\sigma) \qquad\qquad \frac{\dfrac{l \simeq r}{(l \simeq r)\,\sigma} \qquad s[l'] \not\simeq t \mid D}{s[r\sigma] \not\simeq t \mid D}\,(\sigma)$$

where for all grounding substitutions θ with $\mathrm{dom}\,(\theta) = \mathrm{var}\,(\{l\sigma\theta, r\sigma\theta, s[l']\theta, r\theta\})$

(i) $l\sigma = l'$,

(ii) l' is not a variable,

(iii) $l\sigma\theta \succ_{\mathrm{gr}} r\sigma\theta$,

(iv) $(s[l'] \simeq t)\,\theta \succ_{\mathrm{gr}} (l \simeq r)\,\sigma\theta$,

(v) $\mathrm{var}(r) \subseteq \mathrm{var}(l)$,

(vi) $\mathrm{var}\,(\{l, r\}) \cap \mathrm{var}\,(\{s[l'], t\}) = \emptyset$.

In contrast to the lifted unit superposition inference rule in Definition 4.24 on page 78, it is not sufficient if the equation $l \simeq r$ can be oriented for some grounding substitutions only. All ground instances of the conclusion and all ground instances of the left premise must be smaller than the right premise. Therefore for all substitutions θ grounding $(l \simeq r)\,\sigma$ and $s[l'] \simeq t$ we require conditions (iii) and (iv) to hold.

In order to prove completeness of unit superposition with the demodulation inference on first-order literals, we need to show that all literal closures of the right premise become redundant.

Theorem 6.7. *Proper demodulation is a simplification rule, that is, the instantiated demodulating equation and the conclusion make the right premise USD-redundant.*

Proof. Let $(s[l'] \simeq t) \cdot \theta$ be a literal closure of the constrained literal $s[l'] \simeq t \mid D$. The instantiated demodulating equation $(l \simeq r)\,\sigma$ is unconstrained, hence there is a literal closure $(l \simeq r)\,\sigma \cdot \theta$.

By Lemma 6.6 the literal closures $(l \simeq r)\,\sigma \cdot \theta$ and $(s[r\sigma] \simeq t) \cdot \theta$ make the literal closure $(s[l'] \simeq t) \cdot \theta$ US-redundant. We conclude that in a demodulation inference the constrained literal $s[r\sigma] \simeq t \mid D$ in the conclusion together with the unconstrained instantiated left premise $(l \simeq r)\,\sigma$ make the constrained right premise $s[l'] \simeq t \mid D$ USD-redundant (Definition 4.27 on page 81).

A similar proof holds for a negative right premise $s[l'] \not\simeq t \mid D$. □

The reason why the demodulating equation must be unconstrained lies once more in the requirement of irreducibility of the substitution of a literal closure for US-redundancy. We have to consider rewrite systems R such that the substitution θ is irreducible and have to provide literal closures in $\mathcal{D}D$ with a substitution irreducible in R, which are smaller than the demodulated literal closure and together entail the demodulated literal closure. Unless the dismatching constraint on the demodulating equation is empty, we cannot guarantee that there are such literal closures in $\mathcal{D}D$ that make the demodulated literal USD-redundant.

We now extend the demodulation inference rule to both set and tree labelled literals.

Definition 6.5 (Set Labelled Demodulation)**.**

Demodulation

$$\frac{\dfrac{l \simeq r}{(l \simeq r)\,\sigma} \qquad \mathcal{L}\colon s[l'] \simeq t}{\mathcal{L}\colon s[r\sigma] \simeq t}\,(\sigma) \qquad\qquad \frac{\dfrac{l \simeq r}{(l \simeq r)\,\sigma} \qquad \mathcal{L}\colon s[l'] \not\simeq t}{\mathcal{L}\colon s[r\sigma] \not\simeq t}\,(\sigma)$$

where for all grounding substitutions θ with $\operatorname{dom}(\theta) = \operatorname{var}(\{l\sigma\theta, r\sigma\theta, s[l']\theta, r\theta\})$

(i) $l\sigma = l'$,

(ii) l' is not a variable,

(iii) $l\sigma\theta \succ_{\text{gr}} r\sigma\theta$,

(iv) $(s[l'] \simeq t)\,\theta \succ_{\text{gr}} (l \simeq r)\,\sigma\theta$,

(v) $\operatorname{var}(r) \subseteq \operatorname{var}(l)$,

(vi) $\operatorname{var}(\{l, r\}) \cap \operatorname{var}(\{s[l'], t\}) = \emptyset$.

Definition 6.6 (Tree Labelled Demodulation)**.**

Demodulation

$$\frac{\dfrac{l \simeq r}{(l \simeq r)\,\sigma} \qquad \mathcal{T}\colon s[l'] \simeq t}{\mathcal{T}\colon s[r\sigma] \simeq t}\,(\sigma) \qquad\qquad \frac{\dfrac{l \simeq r}{(l \simeq r)\,\sigma} \qquad \mathcal{T}\colon s[l'] \not\simeq t}{\mathcal{T}\colon s[r\sigma] \not\simeq t}\,(\sigma)$$

where for all grounding substitutions θ with $\operatorname{dom}(\theta) = \operatorname{var}(\{l\sigma\theta, r\sigma\theta, s[l']\theta, r\theta\})$

(i) $l\sigma = l'$,

(ii) l' is not a variable,

(iii) $l\sigma\theta \succ_{\text{gr}} r\sigma\theta$,

(iv) $(s[l'] \simeq t)\,\theta \succ_{\text{gr}} (l \simeq r)\,\sigma\theta$,

(v) $\operatorname{var}(r) \subseteq \operatorname{var}(l)$,

(vi) $\operatorname{var}(\{l, r\}) \cap \operatorname{var}(\{s[l'], t\}) = \emptyset$.

Theorem 6.8. *Set labelled demodulation is a simplification rule, that is, the instantiated demodulating equation and the conclusion make the right premise USL-redundant.*

Proof. Let

$$\operatorname{DLit}(\mathcal{L}\colon s[l'] \simeq t) = \{s[l'] \simeq t \mid D_1, \ldots, s[l'] \simeq t \mid D_n\}$$

be the set of constrained literals of the right premise (Definition 5.4 on page 104). The label of the conclusion is the same as the label of the right premise, hence

we have

$$\mathrm{DLit}\,(\mathcal{L}\colon s[r\sigma] \simeq t) = \{s[l'] \not\simeq t \mid D_1, \ldots, s[l'] \simeq t \mid D_n\}$$

as the set of constrained literals of the conclusion.

By Theorem 6.7 the literal $(l \simeq r)\,\sigma$ and a constrained literal $s[r\sigma] \not\simeq t \mid D_i$ make the constrained literal $s[l'] \not\simeq t \mid D_i$ USD-redundant. All constrained literals in $\mathrm{DLit}\,(\mathcal{L}\colon s[l'] \simeq t)$ are USD-redundant in this way and therefore the set labelled literal $\mathcal{L}\colon s[l'] \simeq t$ is USL-redundant in the presence of the set labelled left premise and the conclusion.

A similar proof holds for a negative right premise $\mathcal{L}\colon s[l'] \not\simeq t$. □

Theorem 6.9. *Tree labelled demodulation is a simplification rule, that is, the instantiated demodulating equation and the conclusion make the right premise UST-redundant.*

Proof. We proceed in a similar way as in the previous theorem. The tree label of the right premise and the tree label of the conclusion are identical and the instantiated left premise is unconstrained. We have

$$\mathrm{DLit}\,(\mathcal{T}\colon s[l'] \simeq t) = s[l'] \simeq t \mid \mathrm{DC}\,(\mathcal{T})$$

and

$$\mathrm{DLit}\,(\mathcal{T}\colon s[r\sigma] \simeq t) = s[r\sigma] \simeq t \mid \mathrm{DC}\,(\mathcal{T})\,.$$

With the argument as in the proof of the previous theorem, by Theorem 6.7 the left premise $(l \simeq r)\,\sigma$ and the conclusion $s[r\sigma] \simeq t \mid \mathrm{DC}\,(\mathcal{T})$ make the right premise $s[l'] \simeq t \mid \mathrm{DC}\,(\mathcal{T})$ USD-redundant.

We conclude that the left premise $(l \simeq r)\,\sigma$ and the tree labelled conclusion $\mathcal{T}\colon s[r\sigma] \simeq t$ make the tree labelled right premise $\mathcal{T}\colon s[l'] \simeq t$ UST-redundant.

A similar proof holds for a negative right premise $\mathcal{T}\colon s[l'] \not\simeq t$. □

Demodulation is a simplification inference, that is, a conclusion $s[r\sigma] \simeq t$ replaces the right premise $s[l'] \simeq t$ in the US-saturation process. In order to show that fairness of the Inst-saturation process is preserved when using demodulation in the US-saturation process, we have to consider the relevant instances generated from proofs of contradictions in the US-saturation process.

We proceed in a similar way to Section 4.3, where we have introduced the concept of relevant instantiators from proofs. We are first concerned with proofs

of contradictions from literal closures, using inferences in the unit superposition calculus and demodulation inferences. The key to fairness of the Inst-saturation process is that in every proof of a contradiction from literal closures in a persistent conflict there is at least one proper instantiator. We show that this property of proofs is maintained when demodulating literal closures. We then lift proofs with demodulation to first-order literals and to set labels and tree labels.

Definition 6.7. A *US-proof with demodulation* is a binary tree drawn with the root at the bottom, where each node is labelled with a sequence of literal closures $[L_1 \cdot \theta, \ldots, L_n \cdot \theta]$ of a US-saturation process. For each $1 \leq i < n$ the literal closure $L_{i+1} \cdot \theta$ is the conclusion of a demodulation inference from the literal $L_i \cdot \theta$ and some demodulating literal closure $(l \simeq r)\,\sigma \cdot \tau \in \mathcal{D}^\infty$.

In each *leaf node* $[L_1 \cdot \theta_1, \ldots, L_n \cdot \theta_n]$ the closure $L_1 \cdot \theta_1$ is S-relevant.

Inner nodes $[L_1 \cdot \theta, \ldots, L_k \cdot \theta]$ have exactly two children $[L'_1 \cdot \theta', \ldots, L'_m \cdot \theta']$ and $[L''_1 \cdot \theta'', \ldots, L''_n \cdot \theta'']$, where $L'_m \cdot \theta'$ and $L''_n \cdot \theta''$ are the premises of a unit superposition inference with the conclusion $L_1 \cdot \theta$.

The *root node* is either derived in a unit superposition inference like an inner node or is a contradiction $\square$, in which case the root node has only one child node $[L_1 \cdot \theta, \ldots, L_n \cdot \theta]$ such that an equality resolution inference is applicable to $L_n \cdot \theta$.

Each *edge* in the tree is labelled with the substitution σ from the unit superposition or the equality resolution inference.

In US-proofs without demodulation (Definition 4.18 on page 65) each node is labelled with a literal closure, whereas in the above definition a node is labelled with a sequence of literal closures. A demodulation inference does not change the structure of a US-proof tree, the conclusion of the demodulation inference is only appended to the sequence of the root node. We think of a node as being represented by the last literal closure in its sequence.

Since US-proofs with and without demodulation are isomorphic, the definition of relevant instances from a US-proof remains virtually unchanged from Definition 4.19 on page 66.

Definition 6.8. Let P be a US-proof with demodulation with a leaf node labelled with the sequence $N = [L_1 \cdot \theta, \ldots, L_n \cdot \theta]$. Let $\sigma_1, \ldots, \sigma_n$ be the substitutions labelling the edges along the branch of P from the leaf N to the root. We call the composition $\sigma = \sigma_1, \ldots, \sigma_n$ the P-relevant instantiator and the closure $L_1\sigma \cdot \rho$ the P-relevant instance of the leaf node, where $L_1\sigma\rho = L_1\theta_1$.

We now prove a variant of Theorem 4.19 on page 66.

Theorem 6.10. *Let P be a US-proof with demodulation with a root node $\square$ or $[L'_1 \cdot \theta', \ldots, L'_k \cdot \theta']$ and the literal closures $L_1 \cdot \theta_1, \ldots, L_n \cdot \theta_n$ as the first elements in the sequences labelling the n leaf nodes. Let $L_1\sigma_1 \cdot \rho_1, \ldots, L_n\sigma_n \cdot \rho_n$ be the P-relevant instances of the leaf nodes. Let $\mathcal{D}^\infty$ be the set of persistent literals in the US-saturation process.*

For some finite subset $\{D_1 \cdot \tau_1, \ldots, D_m \cdot \tau_m\}$ of $\mathcal{D}^\infty$ and for every substitution ρ grounding the literals $L_1\sigma_1, \ldots, L_n\sigma_n$ and $D_1, \ldots, D_m$

$$D_1\rho, \ldots, D_m\rho, L_1\sigma_1\rho, \ldots, L_n\sigma_n\rho \models L'_i\rho$$

for all $1 \leq i \leq k$ and

$$D_1\rho, \ldots, D_m\rho, L_1\sigma_1\rho, \ldots, L_n\sigma_n\rho \models \square,$$

respectively.

Proof. Let us first consider a US-proof with demodulation, where each node is labelled with a singleton sequence of exactly one literal closure. This US-proof is isomorphic to a US-proof without demodulation (Definition 4.18 on page 65), where each node is labelled with the single literal of the sequence and by Theorem 4.19 on page 66 we have

$$L_1\sigma_1\rho, \ldots, L_n\sigma_n\rho \models L'_1\rho$$

and

$$L_1\sigma_1\rho, \ldots, L_n\sigma_n\rho \models \square,$$

respectively, for every substitution ρ grounding the literals $L_1\sigma_1, \ldots, L_n\sigma_n$.

We now let $\left[L'_1 \cdot \theta, \ldots, L'_k \cdot \theta, L'_{k+1} \cdot \theta\right]$ be the sequence at the root of a US-proof with demodulation and assume the theorem holds for sequences up to length k. Consequently there is a finite subset $\{D_1 \cdot \tau_1, \ldots, D_m \cdot \tau_m\}$ of $\mathcal{D}^\infty$ such that for all substitutions ρ grounding $D_1, \ldots, D_m, L_1\sigma_1, \ldots, L_n\sigma_n$

$$D_1\rho, \ldots, D_m\rho, L_1\sigma_1\rho, \ldots, L_n\sigma_n\rho \models L'_k\rho.$$

Let $L_{k+1} \cdot \theta = (s[r\sigma] \simeq t) \cdot \theta$ be the conclusion of a demodulation inference between the literal closure $L_k \cdot \theta = (s[l'] \simeq t) \cdot \theta$ and some demodulating literal closure $(l \simeq r)\,\sigma \cdot \theta \in \mathcal{D}^\infty$. We have

$$(l \simeq r)\,\sigma\rho, (s[l'] \simeq t)\,\rho \models (s[r\sigma] \simeq t)\,\rho$$

for all substitutions ρ and therefore

$$(l \simeq r)\,\sigma\rho, D_1\rho, \ldots, D_m\rho, L_1\sigma_1\rho, \ldots, L_n\sigma_n\rho \models L'_{k+1}\rho.$$

A similar argument holds for the demodulation of a negative literal closure $L_{k+1} \cdot \theta = (s[r\sigma] \not\simeq t) \cdot \theta$ and by induction we conclude that the theorem holds for every US-proof with demodulation. □

In contrast to Theorem 4.19, if the US-proof contains demodulation inferences, the implication of the literal at the root by the literals at the leaves only holds together with a subset of the demodulating inferences. Considering the special grounding substitution $\rho = \bot$ we arrive at a variant of Corollary 4.20 on page 67.

Corollary 6.11. *Let $L_1 \cdot \theta_1, \ldots, L_n \cdot \theta_n$ be the literals at the leaves of a US-proof P with redundancy elimination, where the root is a contradiction $\square$. Let $\sigma_1, \ldots, \sigma_k$ be the respective P-relevant instantiators for the leaf literal closures. With some finite subset $\{D_1 \cdot \tau_1, \ldots, D_m \cdot \tau_m\}$ of $\mathcal{D}^\infty$ the set $\{D_1\bot, \ldots, D_m\bot, L_1\sigma_1\bot, L_n\sigma_n\bot\}$ is unsatisfiable.*

Proof. Since $D_1\rho, \ldots, D_m\rho, L_1\sigma_1\rho, \ldots, L_n\sigma_n\rho \models \square$ for each substitution ρ grounding $D_1, \ldots, D_m, L_1\sigma_1, \ldots, L_n\sigma_n$, we choose $\rho = \bot$. Then, a contradiction follows from $\{D_1\bot, \ldots, D_m\bot, L_1\sigma_1\bot, L_n\sigma_n\bot\}$ and hence it is unsatisfiable. □

We use the above corollary to prove that, as required, we can obtain proper instantiators from a US-proof with demodulation.

Theorem 6.12. *Let $\langle K, \mathcal{L}\rangle$ be a persistent conflict and P be a US-proof with demodulation of a contradiction from $\mathcal{L}$. At least one of the P-relevant instantiators is proper.*

Proof. As in Theorem 4.21 on page 68 we have that all $L_i\bot \in \mathcal{L}$ are true in the model $I^i_\bot$ in all states $i \geq i_0$ from some i_0 on. The same holds for each literal closure $(l \simeq r) \cdot \theta \in \mathcal{D}$. Since the US-saturation contains a contradiction, there is a US-proof with demodulation with $\square$ at the root.

Corollary 6.11 states that the set $\{D_1\bot, \ldots, D_m\bot, L_1\sigma_1\bot, L_n\sigma_n\bot\}$ is unsatisfiable. If no relevant instantiator $\sigma_1, \ldots, \sigma_n$ were proper, then for each $1 \leq i \leq n$ we would have $L_i\sigma_i\bot = L_i\bot$ and $\{D_1\bot, \ldots, D_m\bot, L_1\sigma_1\bot, L_n\sigma_n\bot\}$ would be unsatisfiable, since a contradiction can be derived from it. However, $\mathcal{L}$ is satisfiable and each $(l \simeq r) \cdot \theta \in \mathcal{D}$ is true in the model $I^i_\bot$. Hence, there must be at least one proper instantiator in $\sigma_1, \ldots, \sigma_n$. □

With this theorem we can justify the following fair Inst-saturation process. Relevant literal closures are partitioned into $\langle \mathcal{N}, \mathcal{D} \rangle$ and this set is saturated under inferences in the unit superposition calculus in a US-saturation process. In addition, demodulation inferences are performed such that the conclusion replaces the demodulated premise, while at the same time an instance of the demodulating equation is added to the Inst-saturation process. In each US-proof with demodulation of a contradiction there is at least one proper instantiator, hence instantiating the clause of a relevant literal at the leaf of a US-proof makes the conflict irrelevant.

Lifting this US-saturation process from literal closures to first-order literals is not complicated. The proof of Theorem 4.22 on page 72 uses the fact that a US-proof on literal closures can be simulated by a US-proof on first-order literals. We have already lifted the demodulation inference to first-order literals, showing that the first-order conclusion makes each literal closure US-redundant. Together, a variant of the lifting theorem 4.22 holds and we can generate instances for a fair Inst-saturation process from US-proofs with demodulation of first-order literals.

Extending first-order clauses and literals with dismatching constraints is easy. We have to note that demodulating equations cannot be constrained with dismatching constraints. The lifting in Lemma 4.27 on page 84 holds without modification.

In labelled calculi labels take the role of proofs and relevant instances are extracted from there. It is not necessary to modify the extraction of relevant instances from labels for the proofs of completeness in Theorem 5.2 on page 106 for set labels and Theorem 5.4 on page 118 for tree labels. Both refer to USD-redundancy, which covers dismatching constraints and redundancy elimination with demodulation, hence they do not need to be adapted.

We conclude that the labelled unit superposition calculi with demodulation lead to a fair Inst-saturation process. Saturating the set of initially labelled literals under unit superposition and equality resolution inferences and adding clause

instances from labels of contradictions to the Inst-saturation process eventually makes all persistent conflicts irrelevant. In the labelled calculi the merging and demodulation inferences are available in order to eliminate redundancy, the latter is only valid if an instance of the demodulating equation is added to the Inst-saturation process.

In practice, every demodulation inference generates a new clause to be added to the Inst-saturation process, which in turn results in a new literal in the US-saturation process. Usually only few of the demodulation inferences performed contribute to US-proofs of contradictions. Since for completeness it suffices to make persistent conflicts in the Inst-saturation irrelevant, where most of the instances of demodulating equations are not required, a lazy approach to these instantiations is possible and beneficial. Moreover, in the incremental Inst-saturation process the selection function can change, thus literals previously selected are removed from the US-saturation process or dismatching constraints block inferences. As discussed, in these cases parts of a literal label, corresponding to a then redundant proof, can be eliminated and the instances from demodulation inferences produced in those proofs are not required, either.

Lazy instantiation of demodulating equations can be uniformly integrated into the instantiation process of labels of contradictions and benefit from redundancy elimination in labels.

Definition 6.9 (Set Labelled Demodulation with Lazy Instantiation)**.**

Demodulation

$$\frac{\{l \simeq r \cdot []\} : l \simeq r \qquad \mathcal{L} \colon s[l'] \simeq t}{\{l \simeq r \cdot \sigma\} \cup \mathcal{L} \colon s[r\sigma] \simeq t}\,(\sigma) \qquad \frac{\{l \simeq r \cdot []\} : l \simeq r \qquad \mathcal{L} \colon s[l'] \not\simeq t}{\{l \simeq r \cdot \sigma\} \cup \mathcal{L} \colon s[r\sigma] \not\simeq t}\,(\sigma)$$

where for all grounding substitutions θ with $\mathrm{dom}\,(\theta) = \mathrm{var}\,(\{l\sigma\theta, r\sigma\theta, s[l']\theta, r\theta\})$

(i) $l\sigma = l'$,

(ii) l' is not a variable,

(iii) $l\sigma\theta \succ_{\mathrm{gr}} r\sigma\theta$,

(iv) $(s[l'] \simeq t)\,\theta \succ_{\mathrm{gr}} (l \simeq r)\,\sigma\theta$,

(v) $\mathrm{var}(r) \subseteq \mathrm{var}(l)$,

(vi) $\mathrm{var}\,(\{l, r\}) \cap \mathrm{var}\,(\{s[l'], t\}) = \emptyset$.

Definition 6.10 (Tree Labelled Demodulation with Lazy Instantiation)**.**

Demodulation

$$\frac{l \simeq r \cdot [] : l \simeq r \qquad \mathcal{T} : s[l'] \simeq t}{l \simeq r \cdot \sigma \sqcap \mathcal{T} : s[r\sigma] \simeq t}\,(\sigma) \qquad \frac{l \simeq r \cdot [] : l \simeq r \qquad \mathcal{T} : s[l'] \not\simeq t}{l \simeq r \cdot \sigma \sqcap \mathcal{T} : s[r\sigma] \not\simeq t}\,(\sigma)$$

where for all grounding substitutions θ with $\mathrm{dom}\,(\theta) = \mathrm{var}\,(\{l\sigma\theta, r\sigma\theta, s[l']\theta, r\theta\})$

(i) $l\sigma = l'$,

(ii) l' is not a variable,

(iii) $l\sigma\theta \succ_{\mathrm{gr}} r\sigma\theta$,

(iv) $(s[l'] \simeq t)\,\theta \succ_{\mathrm{gr}} (l \simeq r)\,\sigma\theta$,

(v) $\mathrm{var}(r) \subseteq \mathrm{var}(l)$,

(vi) $\mathrm{var}\,(\{l, r\}) \cap \mathrm{var}\,(\{s[l'], t\}) = \emptyset$.

Since dismatching constraints are not compatible with demodulation, we mark the closures from an instantiation of a literal label and only check dismatching constraints in unit superposition and equality resolution inferences on unmarked closures.

In Theorem 6.10 the substitution ρ for grounding the literals at the leaves of a US-proof tree is arbitrary. The literals $L_1\sigma_1, \ldots, L_n\sigma_n$ at the leaves of the proof tree and the instances of the demodulating equations $D_1, \ldots, D_m$ are pairwise variable-disjoint. Although in Corollary 6.11 all literals are grounded with the substitution $\rho = \bot$, we can choose any other substitution $\sigma\bot$ for literals in $\mathcal{D}$ instead without losing generality of the theorem. Therefore, besides considering the dismatching constraint to be empty, literal closures from demodulation inferences do not need to be treated different from other closures in labels and it is in particular sound to instantiate the closure representing an instance of a demodulator when instantiating a label.

Having derived a contradiction in a labelled unit superposition calculus with the above demodulation inference rule, the relevant instances of the label also contain the instances of demodulators to be added to the Inst-saturation process. We therefore do not need to find separately instances of demodulating equations and adding the instances only together with the relevant instanced from the label of a contradiction prevents generating unnecessary instances of demodulating equations.

With the above inference rules we conclude this chapter. We have introduced

a demodulation inference rule for redundancy elimination, based on the non-proper demodulation inference from Ganzinger and Korovin [2004]. In order to allow proper substitutions in a demodulation inference, we need to generate an instance of the demodulating equation in the Inst-saturation process. We have lifted this approach to labelled calculi, where we can uniformly integrate the instantiation of demodulating equations into the generation of relevant instances from labels of contradictions. This lazy instantiation has the additional benefit of avoiding unnecessary instances when a derivation does not lead to a contradiction or a part of a label is eliminated.

Demodulation is only a valid simplification if the demodulating equation is a literal from a unit clause. In the framework of the US-saturation process there is a further simplification possible that is in practice particularly efficient together with demodulation. Literals derived from literals in unit clauses make variants of literals derived from non-unit literals redundant. In this way merging inferences, where one set or tree labelled literal is in $\mathcal{N}L$ or $\mathcal{N}T$ and the second literal is in $\mathcal{D}L$ or $\mathcal{D}T$, respectively, are avoided. Literals in $\mathcal{N}L$ and $\mathcal{N}T$ subsume other variants.

Chapter 7

The iProver-Eq System

In the preceding chapters we have described the contributions of the thesis to the theory of instantiation-based reasoning and the Inst-Gen-Eq calculus. All aspects presented are implemented in the iProver-Eq[1] system, which we describe in this chapter and evaluate in the next.[2]

The iProver-Eq system is an experimental branch of the iProver system [Korovin, 2008], which is separately maintained. iProver treats equations only axiomatically, iProver-Eq reasons on pure equational first-order logic with labelled unit superposition calculi as described in this work. While iProver-Eq is an extension of the iProver system and reuses and refines the basic structure and code, it does represent a significant development effort as this chapter details.

The implementation addresses the combination of three components:

(i) Deciding satisfiability of a ground abstraction by an SMT solver,

(ii) Literal selection on first-order clauses based on a model of the ground abstraction and

(iii) First-order equational reasoning on literals with a labelled unit superposition calculus and generation of instances from labelled contradictions.

Parts (ii) and (iii) represent the Inst-saturation and the US-saturation process, respectively. Both saturation processes are based on the concept of the so-called given clause algorithm. It has become the standard approach in saturation-based theorem proving since it was made popular in two variants, both named after the

[1] iProver-Eq is available from `http://www.cs.man.ac.uk/~sticksec/iprover-eq`

[2] Parts of this chapter were published as Korovin and Sticksel [2010b]

provers that first implemented them: the DISCOUNT loop by Denzinger et al. [1997] and the Otter loop by McCune and Wos [1997]. Contemporary successful theorem provers for first-order logic modulo equality employing a given clause algorithm are, for instance, Vampire by Riazanov and Voronkov [2003], E by Schulz [2002] and SPASS by Weidenbach et al. [2007].

A given clause algorithm saturates a set of clauses under inferences in the following way. The current set of clauses is partitioned into two disjoint sets, commonly called the active and passive clauses. Initially all clauses are passive and the active set is empty. In one step of the saturation procedure, for some clause from the passive set, called the given clause, all inferences with clauses in the active set are performed and the conclusions are added to the passive set. The given clause then becomes active and the process takes a new given clause from the passive set until the passive set is empty. The invariant maintained by the procedure is that the active set is saturated under inferences. If the given clauses are taken from the passive clauses in a fair way, that is, every clause becomes the given clause in some iteration of the procedure, the algorithm implements a fair saturation process in the sense that every inference between two clauses is performed eventually. It is possible to integrate redundancy elimination strategies into the given clause algorithm and this is the main aspect variants of the algorithm differ in.

The Inst-Gen-Eq method contains the nested Inst-saturation process and the US-saturation process and iProver-Eq implements this nesting with two interleaved given clause procedures we are calling the Inst-Gen-Eq-loop and the US-loop in the following. The Inst-Gen-Eq-loop maintains an Inst-active and an Inst-passive set of clauses, in the beginning the former is empty and the latter contains the entire input clause set S. In parallel the US-loop deals with a US-active and a US-passive of labelled literals, both sets are initially empty.

Let us first sketch the Inst-Gen-Eq-loop (Figure 7.1), the US-loop and the integration of ground solving before we give details about the implementation of each component.

In every step the Inst-Gen-Eq-loop takes a given clause from the Inst-passive set, selects one literal based on a model of the ground abstraction, adds the literal with an initial labelling to the US-passive set and the given clause to the Inst-active set. The invariant of the Inst-Gen-Eq-loop is that the selection function on the Inst-active clauses is based on some model of the ground abstraction and

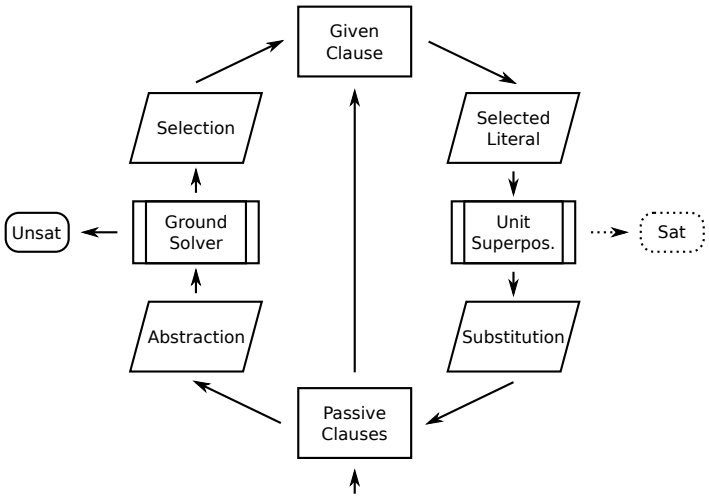


Figure 7.1: The Inst-Gen-Eq-loop

each literal selected in an Inst-active clause is US-passive or US-active.

The US-saturation process in turn takes a given literal from the set of US-passive literals, first adds the given literal to the US-active set and then performs all unit superposition inferences between the given literal and US-active literals, which now include the given literal itself. If an equality resolution inference cannot be applied to a conclusion of a unit superposition inference, the conclusion is added to the US-passive set. Otherwise, the relevant clause instances from the label of the contradiction are added to the Inst-passive set and their abstractions to the ground solver. In the US-loop the invariant is that the US-active literals are closed under unit superposition inferences and relevant instances from each contradiction resulting from an equality resolution with a literal in the US-active set are in the Inst-passive or Inst-active set. Merging and demodulation inferences for redundancy elimination are performed on the given literal, on UP-active literals and on the conclusions of unit superposition inferences.

The iProver-Eq system interleaves the three components for Inst-saturation, US-saturation and ground solving as described in Section 4.7. The input clauses are abstracted and passed to the ground solver to check satisfiability. If the ground solver finds the ground abstraction to be unsatisfiable the system terminates with the result unsatisfiable. Otherwise, the Inst-Gen-Eq-loop processes a fixed number of given clauses from the Inst-passive set, subsequently the US-loop takes a fixed number of given literals from the US-passive set and returns to the Inst-Gen-Eq-loop. After a fixed number of given clauses in the Inst-Gen-Eq-loop

the ground solver is invoked to return an updated model of the ground abstraction. If both the Inst-passive and the US-passive sets are empty and the ground abstraction is satisfiable, iProver-Eq returns with the result satisfiable.

We now separately turn to the three components, describe some details of their respective implementations and highlight the contributions of this work to the iProver-Eq system.

7.1 Ground Satisfiability Solving

The most visible change in the iProver-Eq system in the ground solving component is the use of a solver for satisfiability modulo theories (SMT) instead of a propositional satisfiability (SAT) solver. Since the ground abstraction has to be considered modulo equality, a SAT solver is not sufficient and also the integration of the solver has to be adapted.

The iProver-Eq system integrates a full SMT solver for ground equational satisfiability, although SMT solving is more powerful and solvers have more features than strictly necessary. Future work will extend the Inst-Gen-Eq calculus to first-order reasoning modulo theories as proposed in Ganzinger and Korovin [2006]. Since this requires ground solving modulo theories, the approach of this work can be extended without much effort.

A central rationale behind the Inst-Gen(-Eq) method is to employ an off-the-shelf tool for ground solving. However, only a few specialised tools for ground solving modulo equality are available (DPT by Krstić and Goel [2007] and EufDpll by Tveretina and Wesselink [2009]), while on the other hand SMT solving is a highly active field of research. Theories of interest in the SMT community are several variants of arithmetic (linear, real, integer, for example), bit vectors and data structures like arrays. All theories include the core theory of equality and speaking in SMT terminology we are interested in the theory of "unquantified formulae over uninterpreted function symbols" (QF_UF) [Barrett et al., 2010].

Stimulated by an annual competition (SMT-COMP 2005-2011, see Barrett et al. [2008]) and by demand from applications, performance and ease of integration are of prime concern to developers of SMT solvers. Several stable and mature tools have been developed and are available to solve the QF_UF theory as we require. In the first instance we have chosen the CVC3 solver [Barrett

and Tinelli, 2007], since it serves as a reference implementation for the SMT-LIB benchmark library and is freely available with an appropriate open source licence. Nevertheless, the integration is modular enough to connect other SMT solvers like OpenSMT [Bruttomesso et al., 2010], veriT [Bouton et al., 2009] or Z3 [de Moura and Bjørner, 2008b] in further work.

The ground SMT solver has to provide the following operations.

Assert(C) Add the ground clause C to the solver.

Push()/Pop() Increase the level of the assertion stack (Push) and return to the previous level (Pop), undoing all assertions made on higher stack levels.

CheckSat() Return whether the clause set is satisfiable and compute a model.

GetValue(L) Return the truth value of the ground literal L in the model computed in the last CheckSat() operation.

The SMT solver is regarded as a black box and only used with these operations, which are usually exposed in its API and also part of the command language in the SMT-LIB v2.0 standard [Barrett et al., 2010]. It is expected that the SMT solver provides a fast response to the operations Assert(C), Push(), Pop() and LiteralValue(L), while CheckSat() is computationally expensive. The next section discusses the ways the Inst-Gen-Eq-loop uses the SMT solver and how the impact of the expensive CheckSat() operation is minimised.

7.2 Literal Selection on First-Order Clauses

The main purpose of the Inst-Gen-Eq-loop is the selection of literals in clauses and the propagation of selected literals to the US-loop. A step in the procedure mainly consists of taking a given literal from the Inst-passive set, selecting a literal based on a model of the ground abstraction and adding the initially labelled selected literal to the US-passive literals.

Checking satisfiability of the ground abstraction and computing a model is an expensive operation for the SMT solver and, what is worse, a changed model of the ground abstraction induces changes to the selection function and requires a number of actions in the saturation processes. Selecting a different literal in a clause removes the previously selected literal from the US-loop and causes all inferences with this literal to become invalid. In turn new inferences become

necessary when the newly selected literal is inserted into the US-passive set. In order to ease the impact of these potentially significant consequences, the Inst-Gen-Eq-loop takes a lazy approach to checking satisfiability and changing the selection function, as long as possible both deferring the former and attempting to preserve the latter.

Invoking the satisfiability check of the ground solver is only necessary if new clauses have been added to the passive set. Since the ground solver works in an incremental way, we can expect that most of the model remains identical to the previous model and changes are only local. Further, for fairness of the Inst-saturation process it is enough if the literal selection of the Inst-active clauses is based on a model of the ground abstraction only eventually. It does not harm soundness nor completeness to perform inferences with literals not actually selected, that is, their ground abstraction is not true in the current model of the ground abstraction. This allows to arbitrarily defer ground satisfiability checks and in the Inst-Gen-Eq-loop CheckSat() in the SMT solver is only called in intervals after a fixed number of given clauses has been taken from the Inst-passive set.

For this reason, clause instances from contradictions in the US-loop are not immediately added to the passive set, but to a separate "unprocessed" set of clauses, such that no unprocessed clause is eligible to become the given clause. An unprocessed clause instance is not in the ground abstraction, not for all literals of the unprocessed clause there is a truth value defined in the model of the ground abstraction, hence, in order to be able to select a literal, a CheckSat() operation is necessary. Not inserting unprocessed clauses into the Inst-passive set and deferring updating the ground abstraction with the unprocessed clauses until the next scheduled CheckSat() can save a number of such expensive computations of a ground model, which is done en bloc without losing fairness of the Inst-saturation procedure.

When the model of the ground abstraction has changed, it is possible that for some clause in the Inst-active set the selection is not based on the new model of the ground abstraction. Nevertheless, we defer moves to the Inst-passive set necessary due to changes in the literal selection of Inst-active clauses until saturation has been achieved, that is, the Inst-passive set is empty. As before, we assume that changes in the model of the ground abstraction are limited to few literals and most of the selection remains constant.

Models of the ground abstraction are in general not unique and since the ground solver in our black box view non-deterministically chooses a model, we attempt to influence the model search in our favour through the mechanism of the Push() and Pop() operations in two ways.

Upon saturation we have to move Inst-active clauses to Inst-passive, where the selection is not based on a model of the ground abstraction. In order to minimise the number of those moves we do not rely on the model as calculated by the solver, but search for a maximal satisfiable set of literals in the following way. We increment the assertion stack level with a Push() operation and assert a selected literal. If CheckSat() finds the ground abstraction to be satisfiable, we Push() again and assert the next literal. If CheckSat() returns unsatisfiable, we decrement the stack level, backtracking the assertion of the last literal and move the clause the literal is selected in to Inst-passive. When all selected literals have been asserted, we have moved only clauses where literals were selected that cannot be in a ground abstraction together with the literals that remained in clauses kept Inst-active. To clean up we decrement the assertion stack level to the initial level, backtracking all literal assertions.

Due to the moves of Inst-active clauses back to Inst-passive, it is possible that a given clause has been Inst-active before. Instead of selecting a literal which is true in the ground abstraction, we try to influence the computed model in a similar way as described above. We increment the assertion stack, assert the literal that was previously selected and check the satisfiability of the ground abstraction with the literal. If there is a model such that the literal is true, we backtrack the assertion of the literal and keep the selection. If the ground abstraction together with the literal is unsatisfiable, we backtrack the assertion of the literal and have the solver compute the previous model once again. We then select a different literal that is true in the ground abstraction.

Invoking the ground solver only in larger intervals, keeping new clauses in a separate set of unprocessed clauses and trying to preserve the previous selection of a clause are techniques already implemented in the iProver system that had to be only slightly modified for iProver-Eq. However, in the non-equational case it is not possible that the selected literals are inconsistent upon saturation if it is ensured that the selection function does not select a literal and its complement. Checking the selection function for inconsistency upon saturation is necessary in iProver-Eq and our evaluation in the following Chapter 8 shows its efficiency.

To conclude the presentation of the Inst-Gen-Eq-loop we mention heuristics in the selection process and techniques for simplification of clauses that were already present in the iProver system and that persist in the iProver-Eq system.

Choosing the given clause from the Inst-passive set has to be fair, that is, every clause has to become given eventually. However, there are clauses that are more promising to lead to a refutation of the input clause set than others. Therefore, iProver and iProver-Eq maintain the Inst-passive clauses in two priority queues from which a given clause is alternatively chosen in a fixed ratio. The first priority queue ensures fairness of the saturation process and clauses are ordered by their age, that is, the point in time when they were generated. Initially all input clauses are in the queue and instances from conflicts are appended to the end of the queue. From clauses of the same age, the smallest clause is selected first. The second queue can follow an arbitrary heuristic, by default clauses are preferred that are closer to some distinguished conjecture clauses in the input, that contain a symbol that occurs in these conjecture clauses and that have fewer variables.

Since the clause length remains constants when instantiating, it is vital for the success of the iProver system to employ methods to obtain shorter clauses. These methods have a similar beneficial effect in the iProver-Eq system although they have not been adapted to equational reasoning. The technique of global propositional subsumption described in Korovin [2009] uses the ground solver to shorten ground and non-ground clauses. The iProver system further contains a prover using the non-equational resolution calculus, which is run in parallel to the instantiation process, and adds clauses to the ground solver that enhance the global propositional subsumption. Another reduction of clause lengths is achieved by ground splitting clauses. These techniques for simplifying clauses are applied to the given clause before selecting a literal.

7.3 Labelled Unit Superposition

We now turn to the implementation of the US-saturation process that was introduced and proved complete in Chapter 4, extended to a labelled approach in Chapter 5 and enhanced with redundancy elimination inferences in Chapter 6.

The US-loop is a variant of a given clause algorithm and maintains a set of US-active and a set of US-passive literals. Both sets are maximally reduced with respect to merging inferences, hence there is at most one literal variant of each

literal in the US-loop. In the implementation, potential duplication of storing a US-passive and a US-active variant is eliminated by labelling a literal with both an active and a passive label. When a new labelled literal is to be added to the US-passive or US-active set and a variant of the literal is in the US-loop, its label is merged with the active or passive label of the single variant of the literal in the US-loop.

The input to the US-loop are initially labelled literals selected in clauses by the Inst-Gen-Eq-loop. Demodulation and equality resolution inferences are always applied eagerly to a literal in order to detect contradictions early and to keep the number of literals small. Hence, a labelled literal propagated from the Inst-Gen-Eq-loop is only added to the US-passive set after it has been demodulated with equations from the US-active set and if no equality resolution inference can be applied. In the latter case, relevant instances are generated from the label of the contradiction and passed to the Inst-Gen-Eq-loop, where they are stored in the unprocessed clause set.

After the Inst-Gen-Eq-loop has selected literals in a fixed number of clauses and initially filled the US-passive set, control is handed over to the US-loop, where a fixed number of US-passive literals is processed. If the US-passive set becomes empty, the US-loop returns to the Inst-Gen-Eq-loop, since Inst-saturation has not been achieved unless the Inst-passive set is also empty.

As in the Inst-Gen-Eq loop, there are two priority queues, which the given literals are alternatively chosen from with a fixed ratio. Again, the first queue of literals is sorted by age and draws are resolved by considering smaller literals first. The age of a selected literal is equal to the age of the clause it is selected in, the age of a literal derived in a unit superposition inference from two premises is the age of the oldest premise incremented by one. Simplification by demodulation does not increment the age of the demodulated literal and when merging literal variants the age of the conclusion is the least age of the two variants. Therefore, the first priority queue ensures fairness of the procedure, since no literal is chosen as the given literal before all literals it is derived from have been taken.

The second priority queue prefers literals with characteristics that are expected to produce contradictions earlier and to yield smaller relevant instantiators. By default, the priority queue is ordered by a lexicographic combination of the literal sign, preferring negative literals, groundness, preferring ground literals, the number of variables and the number of symbols, preferring smaller literals in

both cases.

Literals in the US-active set are not explicitly listed, but stored in term indexes for efficient retrieval of unifiable literals and subterms, see Graf [1996] or Sekar et al. [2001]. iProver-Eq reuses and adapts the implementation of non-perfect discrimination trees in iProver. A discrimination tree is a data structure similar to a trie used for string search and the main feature is that common prefixes of terms are shared. A term index consists of one discrimination tree that contains all terms it indexes. Querying a perfect discrimination tree finds all terms stored in the tree that are unifiable with the query term. In non-perfect discrimination trees all variables are considered equal and the query finds only terms that are potentially unifiable, thus for each candidate term an additional check is necessary if it is actually unifiable with the query term. Since matching is a subproblem of unification with a small modification a term indexing discrimination tree can be used to query terms matching a given term or terms that the given term matches.

The following three term indexes are maintained:

Literal index If a positive US-active literal $l \simeq r$ is orientable, that is, $l\theta \succ_{\mathrm{gr}} r\theta$ for each grounding substitution θ, only the left-hand side l is stored in the literal index. Otherwise, there are grounding substitutions θ and ρ such that $l\theta \succ_{\mathrm{gr}} r\theta$ and $r\rho \succ_{\mathrm{gr}} l\rho$, and both sides l and r are stored in the index.

Demodulation index The demodulation index is similar to the literal index, but it only contains positive equations that can be used as demodulating equations, that is, literals from unit clauses. Since demodulating equations have to be orientable, only the greater left-hand side terms of the equations are in the index.

Subterm index For every positive literal $l \simeq r$ and every negative literal $l \not\simeq r$ every subterm of l and r is added to the index, the terms l and r themselves are also in the subterm index.

The term indexes facilitate the search for candidates for forward and backward of demodulation and unit superposition inferences. Every new literal is demodulated with US-active equations stored in the demodulation index. In this forward demodulation we try to demodulate each subterm of the new literal in a bottom-up manner starting with the deepest subterms. For each subterm l' of the literal, which is not a variable, we query the demodulation index to find a demodulating equation $l \simeq r$ where l matches l' with a substitution σ. We then perform

the demodulation inference, replacing the subterm l' with the term $r\sigma$ and continue demodulating other subterms until we ultimately attempt to demodulate the literal itself. Since we keep the demodulating equations in the US-active set maximally reduced, the demodulation of a term is unique and we can use caching to store and even quicker retrieve the demodulation of a term, without the need to query the index for each position in the term.

In order to keep the set of US-active demodulating equations interreduced, we have to employ backward demodulation if a new demodulating equation $l \simeq r$ is added to the US-active set. We extend backward demodulation and reduce not only demodulating equations but in addition each US-active literal. We query the subterm index for literals containing a subterm l' that the left-hand side l of the new demodulating equation can be matched to with a substitution σ. We then demodulate each literal, replacing the subterm l' with the term $r\sigma$.

In order to saturate the US-active set under unit superposition inferences we have to apply inferences between the given literal and literals in the US-active set forward and backward in the above sense and in a similar way. In a backward superposition the given literal is the left premise and therefore has to be a positive equation $l \simeq r$. We query the subterm index for literals containing a subterm l' unifiable with l and perform the unit superposition inference if the side conditions are satisfied. If the equation in the given literal is not orientable, that is, there are substitutions θ and ρ such that $l\theta \succ_{\mathrm{gr}} r\theta$ and $r\rho \succ_{\mathrm{gr}} l\rho$, we also search for literals with a subterm l' unifiable with the right-hand side r of the equation.

Forward unit superposition is applicable for positive and negative given literals, for each subterm l' we query the literal index for equations $l \simeq r$ unifiable with l'. Since the literal index contains the left-hand side of each US-active equation and additionally the right-hand of unorientable equations, one query is sufficient. If the side conditions of the unit superposition inference are satisfied for the given literal and the US-active equations retrieved from the index we perform the inference.

Figure 7.2 gives an overview of the complete US-loop, showing in particular where simplification by demodulation is performed.

Before an initially labelled literal that is propagated from a selection in the Inst-Gen-Eq loop is inserted into the US-passive set, it is demodulated using the demodulation index of US-active demodulating equations. If an equality resolution inference is applicable to the simplified selected literal, the relevant

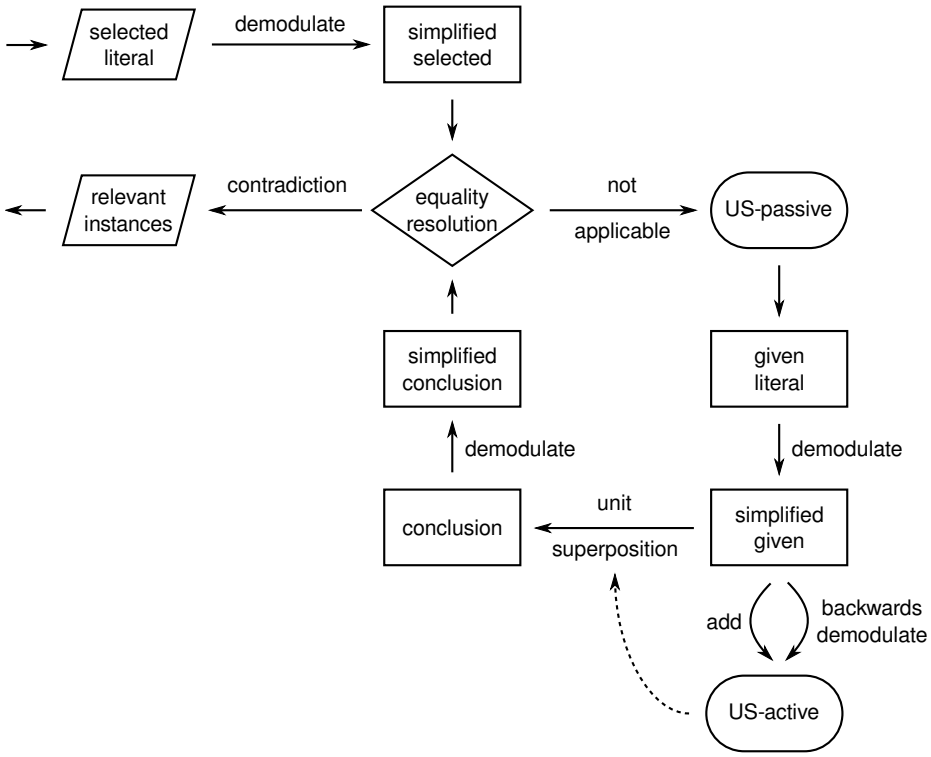


Figure 7.2: The US-loop

instances from the contradiction derived in the inference are propagated back to the unprocessed set of the Inst-Gen-Eq-loop. If a variant of the literal is US-active or US-passive, the label of the simplified selected literal is merged with the possibly empty passive label. Otherwise, the selected literal is inserted into the US-passive set and into the two priority queues the given literal is taken from.

An iteration of the US-loop takes a literal from one of the priority queues, honouring the fixed ratio to maintain fairness. Similar to the way a selected literal is treated, the given literal is demodulated as far as possible and if an equality resolution inference can be applied, the relevant instances are propagated to the Inst-Gen-Eq-loop. In this case the passive label is emptied and the next iteration of the US-loop starts.

Otherwise, the passive label is merged with the possibly empty active label. If the active label is unchanged after merging the passive label, the US-active labelled literal is equal up to renaming and subsumes the given labelled literal. Since this is redundant we continue with the next iteration of the US-loop without further inferences.

The given literal is now US-active and we perform all forward and backward

unit superposition inferences between the given literal with the passive label and US-active literals with the active label, which also enables inferences with the given literal itself. Each derived conclusion is demodulated, here not by querying the demodulation index but using the cached demodulations, since a significant number of conclusions is generated and accessing the demodulation index for each is computationally too expensive. Unless an equality resolution can be applied to the conclusion, in which case relevant instances of the label of the contradiction are propagated to the Inst-Gen-Eq-loop, the simplified conclusion is either added to the US-passive set and the priority queues or its label is merged with the passive label of an already US-passive variant. Since US-passive literals are demodulated again when they become the given literal, it is sufficient to approximate demodulation using caching. However, it is essential to demodulate conclusions to a certain degree, since this drastically reduces the number of US-passive literals.

We finish the description of the US-loop with remarks on the implementation of literal labels. Literal labels contain non-ground closures and each such closure is given a sequential integer number. Set labels are treated with a built-in implementation of sets. For tree and OBDD labels, in order to conserve memory, we maximise structure sharing with the mechanism of hash-consing described in Filliâtre and Conchon [2006], a familiar approach to structure sharing in functional programming.

Common subtrees of tree labels are shared across different labels and also within one tree label if possible. Maximal sharing of subtrees or subgraphs is a key implementation technique to achieve reduced OBDDs and we base our OBDD implementation on the one by Filliâtre and Conchon [2006].

We have implemented two additional operations with OBDDs that are necessary in our labelling approach. In order eliminate a closure from an OBDD we eliminate all nodes labelled with the contradiction by replacing the edge from the parent node with the edge to the low node and subsequently recursively reduce the OBDD.

When applying a substitution to an OBDD the ordering of the OBDD can become violated and it is also possible that a node occurs twice on a path from the root. Therefore we recurse into the OBDD, ordering the nodes in a bubblesort-like way bottom-up from the terminal nodes. We merge nodes that occur twice on a path in this process. The assumption is that applying the substitution to an

OBDD has only a local effect such that the bubblesort approach is advantageous over other sorting algorithms.

7.4 Using the System

iProver-Eq is implemented in the functional language OCaml and uses CVC3 as ground SMT solver via its C/C++ API. Given a problem that does not contain equations it falls back to the Inst-Gen calculus of the iProver version it was forked from. Since no SMT solver is required in this case, the SAT solver MiniSat is used for ground solving instead.

The input to iProver-Eq are files in TPTP format, in either first-order form (FOF) or conjunctive normal form (CNF). Clausification of FOF problems is delegated to the Vampire theorem prover.

After installation and compilation the executable file `iprover-cvc-eq-nc` is generated. The behaviour of the prover can be changed through close to 100 command line options, whose defaults are tuned to the TPTP benchmark library. We list the most relevant settings for features discussed in this work. All options can be obtained by calling `iprover-cvc-eq-nc --help`.

--time_out_real ⟨**int**⟩ Time limit for iProver-Eq in seconds

--up_literal_labels ⟨**set** | **tree** | **bdd**⟩ The type of literal labels

--up_superposition ⟨**bool**⟩ Unit superposition (**true**) or unit paramodulation (**false**), where condition (iv) $s[l']\sigma\theta \succ_{\text{gr}} t\sigma\theta$ is not enforced on the right-hand side (Definition 4.24 on page 78)

--up_orient_after_subst ⟨**bool**⟩ Orientability of the premises with the substitution σ applied (**true**) as in conditions of (iii) and (iv) in Definition 4.24 or weakening orientability (**false**) without regard to σ, that is, only $l\theta \succ_{\text{gr}} r\theta$ and $s[l']\theta \succ_{\text{gr}} t\theta$ for some grounding substitution θ.

--up_unit_lit_proof ⟨**bool**⟩ Subsumption by unit clauses (Section 6.2)

--up_use_demod ⟨**bool**⟩ Demodulation with equations from unit clauses (Section 6.3)

--up_demod_proper ⟨**bool**⟩ Proper matching substitutions in demodulation (**true**) or only non-proper demodulation (**false**)

--up_demod_concl_cached ⟨**bool**⟩ Cached demodulation for conclusions (**true**) or full demodulation with demodulation index (**false**)

--comb_up_mult ⟨**int**⟩ Number of given literals to process in the US-loop before switching to the Inst-Gen-Eq-loop

--comb_inst_mult ⟨**int**⟩ Number of given clauses to process in the Inst-Gen-Eq-loop before switching to the US-loop

--comb_res_mult ⟨**int**⟩ Number of given clauses to process in the resolution prover before switching to the Inst-Gen-Eq reasoning

--inst_solver_per_active ⟨**int**⟩ Number of given clauses in the Inst-Gen-Eq-loop before invoking the ground solver

--up_pass_queue1_mult ⟨**int**⟩ Ratio for selection of US-passive literals from the first priority queue in the US-loop

--up_pass_queue2_mult ⟨**int**⟩ Ratio for selection of US-passive literals from the second priority queue in the US-loop

--up_pass_queue1 ⟨**str**⟩ Configuration of the first priority queue in the US-loop

--up_pass_queue2 ⟨**str**⟩ Configuration of the second priority queue in the US-loop

--inst_pass_queue1_mult ⟨**int**⟩ Ratio for selection of Inst-passive clauses from the first priority queue in the Inst-Gen-Eq-loop

--inst_pass_queue2_mult ⟨**int**⟩ Ratio for selection of Inst-passive clauses from the second priority queue in the Inst-Gen-Eq-loop

--inst_pass_queue1 ⟨**str**⟩ Configuration of the first priority queue in the Inst-Gen-Eq-loop

--inst_pass_queue2 ⟨**str**⟩ Configuration of the second priority queue in the Inst-Gen-Eq-loop

--inst_loop ⟨**lazy** | **lazy_with_renewal** | **lazy_with_eager_renewal**⟩ Type of Inst-Gen-Eq-loop: no moving of clauses with inconsistently selected literals to Inst-passive (incomplete), move clauses upon saturation (default) or move clauses after each CheckSat()

--inst_lit_sel ⟨**str**⟩ Configuration of literal selection in clauses

After iProver-Eq has terminated, timed out or is interrupted, statistics are output that can be used to explain the result on the particular problem. We list the most relevant statistics for this work.

prop_solver_calls Number of calls of to CheckSat()

prop_solver_time_sum Accumulated time in seconds taken by the ground solver

inst_num_of_clauses Number of clauses in the Inst-Gen-Eq-loop

inst_num_in_active Number of Inst-active clauses

inst_num_in_passive Number of Inst-passive clauses

inst_num_in_unprocessed Number of unprocessed clauses

inst_num_of_loops Number of iterations of the Inst-Gen-Eq-loop

inst_num_of_selection_renewals Number of times the Inst-passive set became empty and clauses with inconsistently selected literals were moved to Inst-passive

inst_num_moves_active_passive Number of Inst-active clauses moved to Inst-passive due to an inconsistently selected literal

num_of_splits Number of ground splits of given clauses

prop_fo_subsumed Number of propositional subsumptions of a given clause

up_num_of_loops Number of US-loop iterations

up_num_of_literals Number of literals in the US-loop

up_num_in_active Number of US-active literals

up_num_in_passive Number of US-passive literals

up_given_demodulated Number of given literals simplified by forward demodulation

up_active_demodulated Number of active literals simplified by backward demodulation with the given literal

up_conclusions_demodulated Number of conclusions simplified by forward demodulation

up_input_demodulated Number of selected literals from the Inst-Gen-Eq-loop simplified with forward demodulation

up_inferences Number of unit superposition inferences

up_tautologies Number of tautologies derived

up_contradictions Number of equality resolution inferences

up_eq_orient_after_subst Number of inferences with left premises only orientable after application of the substitution in an inference and blocked by condition (iii) in Definition 4.24 on page 78

up_lit_orient_after_subst Number of inferences with right premises only orientable after application of the substitution in an inference and blocked by condition (iii) in Definition 4.24 on page 78

up_dismatched_inferences Number of conclusions of unit superposition inferences blocked by dismatching constraints

up_dismatched_literals Number of given literals blocked by dismatching constraints

up_dismatched_contradictions Number of equality resolutions blocked by dismatching constraints

up_closures_in_bdd_labels Number of closures in OBDD labels

up_nodes_in_bdd_labels Number of distinct nodes all OBDD labels

Chapter 8

Evaluation

In this chapter we present an empirical evaluation of the iProver-Eq system. We are in particular interested in validating the contributions of this work and therefore focus on the labelled calculi, redundancy elimination with unit clauses and the SMT solver for ground satisfiability modulo equality.

The evaluation of the features of iProver-Eq is based on the TPTP benchmark library, which we introduce first. We have used a cluster of Intel Xeon Quad Core machines with 2.33GHz and 2GB of memory limit and ran each problem for at most 120 seconds.

We also report on the results of the CADE Automated Theorem Prover System Competition (CASC) iProver-Eq recently participated in and show how the system compares with state-of-the-art systems and in particular the implementations of other instantiation-based methods.

8.1 The TPTP Benchmark Library

The *Thousands of Problems for Theorem Provers* (TPTP) problem library [Sutcliffe, 2009] is a collection of literally thousands of benchmark problems in first-order logic that has become the de facto standard set of test problems for evaluation of automated theorem proving. Since the first release in 1993 it has continually received relevant new problems from the automated reasoning community, contributed both from developers of automated reasoning systems as well as from a large variety of applications. The problems are formulated in a common language that has become a standard by itself and is recognised by most theorem provers, with iProver-Eq being no exception. The benchmark library is actively

curated and the problems form the basis of the annual *CADE Automated Theorem Prover System Competition* (CASC) [Sutcliffe and Suttner, 2006].

TPTP problems are classified in a domain structure according to their subject area, which exhibits a diversity of backgrounds, such as mathematics, including problems from algebra and geometry, software and hardware verification and knowledge representation and reasoning. Since iProver-Eq is meant to be a general purpose theorem prover and the TPTP in its largest part is made up of problems from relevant application domains for automated reasoning, we base the evaluation of iProver-Eq on the TPTP benchmark library.

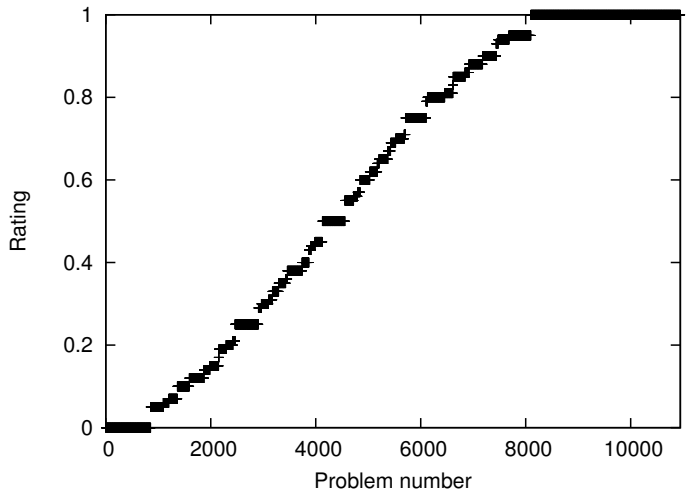
Our results were obtained in compliance with the guidelines for use of the TPTP as given in Sutcliffe [2009]. We have used the TPTP in version 5.1.0 and have not changed the problems from the distribution. iProver-Eq does not use other information than the formulae of the problems and in particular does not make use of information in the problem headers.

The TPTP library contains 14 771 problems in plain first-order logic syntax, of those there are 10 945 problems (74%) with at least one equation and 2 176 problems (15%), which consist entirely of equations. Another subclass are unit equational problems, where all clauses consist of exactly one positive or negative equation, overall 1 092 problems (7%).

For the evaluation in this chapter we focus on 10 049 equational problems, leaving out a number of extremely large problems like those from the Cyc TPTP Challenge Problem Set. Each such problem includes the entire OpenCyc knowledge base of about 3 340 000 axioms. We do not expect to solve these problems, since already parsing and loading the axioms requires a different approach and none of the problems relies on equational reasoning, although the equality predicate occurs.

Problems are presented in one of two syntactical forms: first-order form (FOF), where a problem consists of a list of quantified first-order formulae, and clausal normal form (CNF), where a problem is given as a set of clauses. The latter problems can be processed by iProver-Eq directly, FOF problems are passed to the Vampire system for clausification into CNF form. The 10 945 equational problems are almost evenly split into 5 459 FOF problems and 5 486 CNF problems.

A problem is either satisfiable or unsatisfiable, while the satisfiability of some problems that have never been solved by a theorem prover is considered to be



Rating	Problems
0.0 – 0.1	1 364
0.1 – 0.2	929
0.2 – 0.3	671
0.3 – 0.4	776
0.4 – 0.5	386
0.5 – 0.6	718
0.6 – 0.7	677
0.7 – 0.8	607
0.8 – 0.9	1 050
0.9 – 1.0	918
1.0	2 849
	10 945

Figure 8.1: Distribution of problem ratings among the 10 945 equational problems of the TPTP v5.1.0 (graph adapted from Sutcliffe [2009])

unknown. Satisfiability of the equational problems is unevenly distributed, with 9 110 of 10 945 problems (83%) being unsatisfiable and only 939 (9%) satisfiable problems. It is unknown whether the remaining 896 problems (8%) are satisfiable.

Each problem is ranked with a difficulty rating between 0 and 1. This rating is calculated by an algorithm taking into account the number of state-of-the-art theorem provers able to solve the problem, see Sutcliffe and Suttner [2001]. Generally speaking, a problem with a rating of 0 is solved by all provers and considered easy, while a 1-rated problem is not solved by any prover and considered difficult. Since a version of iProver-Eq with only minor differences to the version used in this chapter has participated in the problem rating for the TPTP version 5.1.0, it is not expected that new 1-rated problems are solved. Figure 8.1 shows the distribution of the difficulty ratings among the equational problems. Apart from the longer tails of easy and difficult problems the middle difficulty ratings are rather uniformly distributed.

8.2 Set, Tree and OBDD Labels

We have presented labelled unit superposition calculi in three variants in Chapter 5. Set labels have a simple structure and are in a natural normal form, such that equivalence up to renaming of two labelled literals can easily be decided. However, elimination from set labels is only approximate and not all redundant

Problems	solved	not solved	fastest
set	2 737	93	1 082
tree	2 643	187	1 237
OBDD	1 383	1 447	511

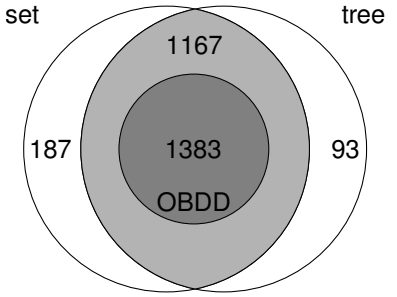


Figure 8.2: Comparing labels by the number of solved problems out of 10 049 in total: set labels solved 2 737 problems and were fastest on 1 082 problems, while 93 problems were solved only with other labels. OBDD labels solved 1 383 problems, all of which were also solved by set and tree labels, while 1 167 problems were solved by set and tree labels, not by OBDD labels.

closures can be removed from a set label. The precise elimination of redundant closures is the strength of tree labels, but since tree labels are not produced in a normal form, labelled literals equal up to renaming are detected only in simple cases. In order to combine precise redundancy elimination and normal forms, we have introduced OBDD labelled unit superposition, which has the disadvantage of higher computational complexity.

Figure 8.2 illustrates the result of running iProver-Eq on the 10 049 selected problems from the TPTP for at most 120 seconds with set, tree and OBDD labelled unit superposition, enabling simplifications by demodulation and subsumption by literals from unit clauses. In total 2 820 problems were solved within the time limit by at least one label implementation. With set labelled unit superposition 93 problems were missed and tree labelled unit superposition could not solve 187 problems. OBDD labelled unit superposition solved only a subset of the problems that both set and tree labelled unit superposition could solve.

Figure 8.3 gives a breakdown of the number of solved problems by rating. It shows that for low ratings the numbers of successfully solved problems are nearly identical in the set and tree labelled implementations. Set labels initially have a minor advantage that accumulates towards more difficult problems. However, we remark that the problems solved with set labels are not a superset of the problems solved by tree labels. As mentioned above, there is a significant number of problems solved with tree labels and not with set labels and vice versa. Looking at the relative number of problems set and tree labelled literals solved for different ratings, we find that these figures to be similar. For each problem rating the

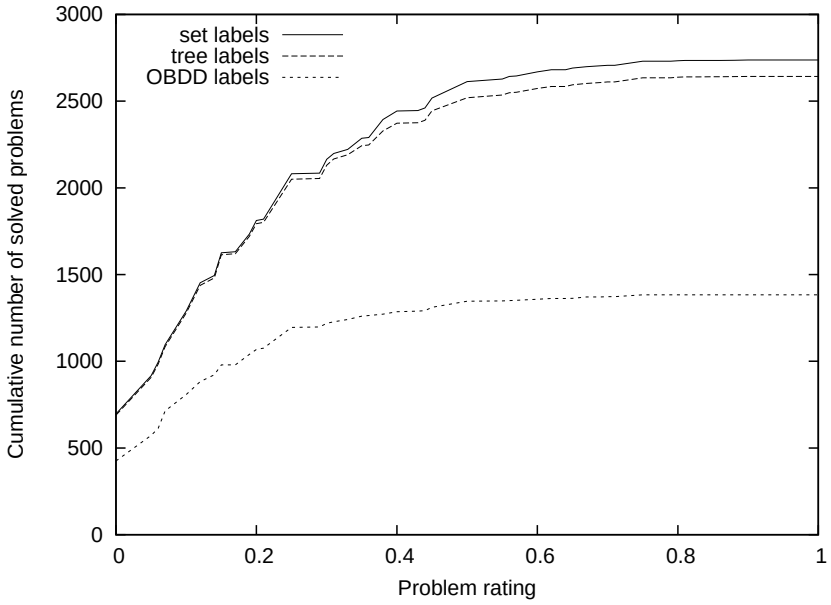


Figure 8.3: Number of problems with a rating less than or equal to a given rating solved within the time limit.

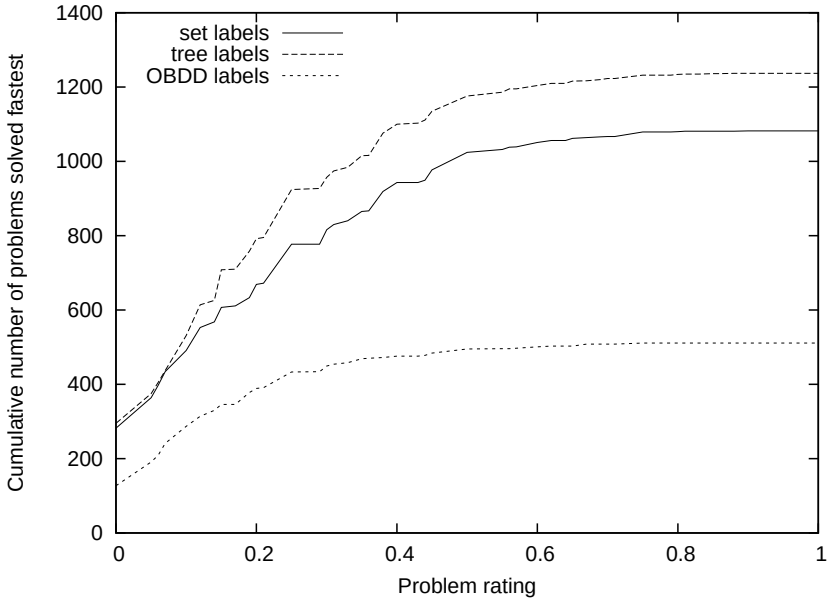


Figure 8.4: Number of problems with a rating less than or equal to a given rating solved fastest in the respective label implementation.

relative number of problems solved with set and tree labels are about equal, hence it is not the case that one of the labels had an advantage for easier or more difficult problems.

The figure also shows that OBDD labels were only successful on easier problems and not many further problems were solved for higher ratings. As we have already indicated, the problems solved with OBDD labels were in fact a subset of both the problems solved with set labels and with tree labels.

Despite the fact that OBDD labels provide a normal form, efficient checking for label equivalence and precise elimination of redundancy, in our experiments they remained considerably weaker than trees and sets. Their performance is mainly hit by the effort spent building labels, which can become rather large. In the problems that could be solved, OBDDs were well-behaved so that the number of nodes was in most cases much less than quadratic in the number of variables, that is the number of closures in labels. Problems that were not solved in the time limit mostly had either a large number of closures (up to 50 000) or the Boolean structure had to be represented with a large number of nodes (many with several millions).

The picture changes when we compare the time it takes to solve problems with different labels. We find that set labels were fastest on only 1 082 problems, whereas tree labels came first on 1 237 problems. Of the 1 383 problems that could be solved with OBDD labels, they were competitive and solved 511 problems faster than either set or tree labels. We have that 40% of the problems solved with set labels were solved faster than any other label, of the problems solved with tree labels 47% were fastest and among the problems solved with OBDD labels 37% were not solved faster by set or tree labels.

Figure 8.4 shows how the fastest runtime relates to the difficulty ratings of the problems. On easy problems there is almost a draw between set and tree labels with a slight advantage for the latter. The gap widens when considering more difficult problems. Set and tree labels were superior to OBDD labels in terms of solved problems and therefore also in terms of problems solved faster.

Let us pairwise compare the label implementations with the runtimes of problems in Figure 8.5. The plots are on a logarithmic scale, each point corresponds to a problem solved in both label implementations. A point on the diagonal is a problem with equal runtime in both labels, a point above the diagonal is solved

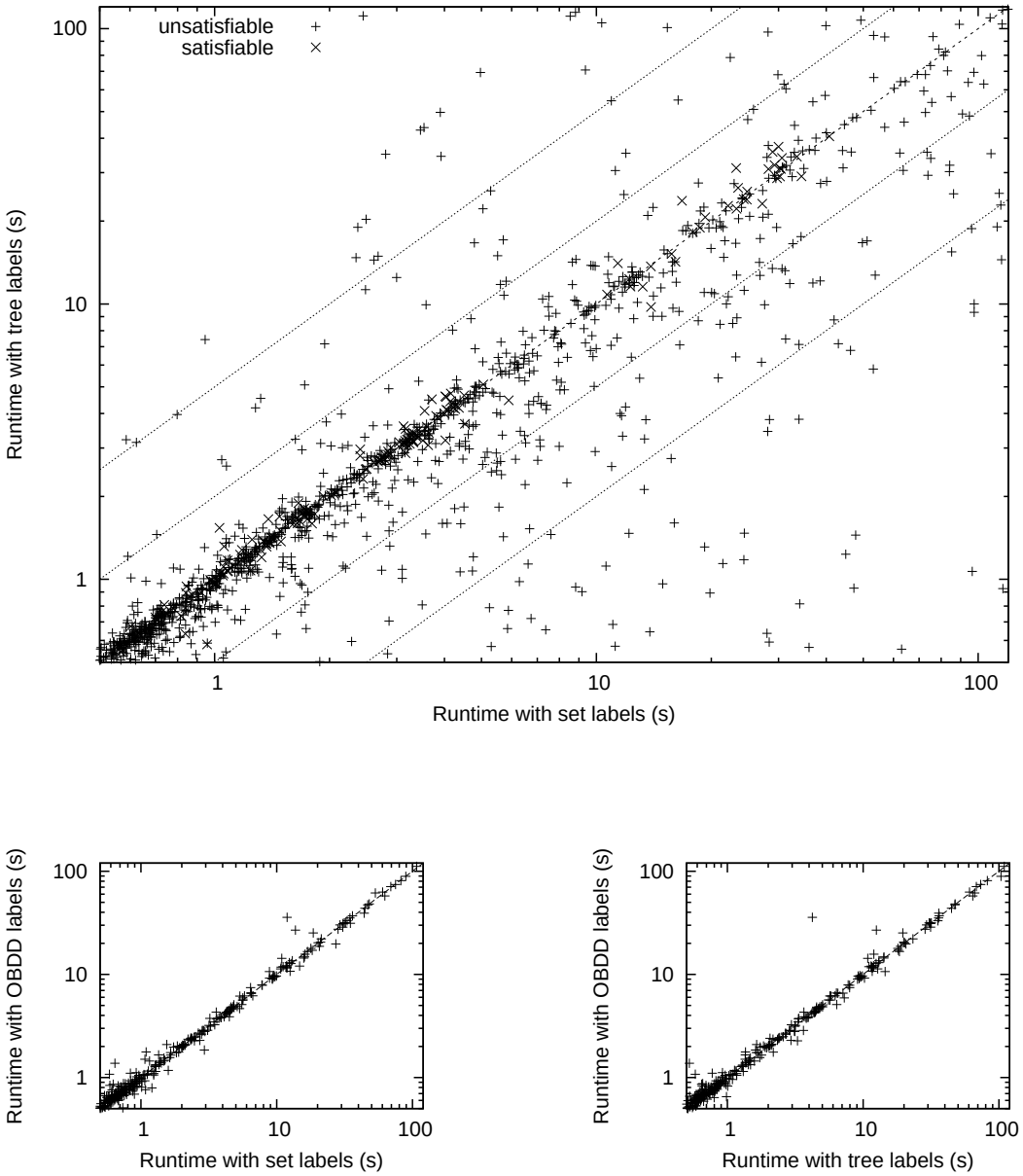


Figure 8.5: Logarithmic scatterplots for the runtimes of problems solved within the time limit of 120 seconds in both label implementations. Each data point below the diagonal line in the first plot represents a problem that was solved faster in tree labelled unit superposition than in set labelled unit superposition, greater distances from the diagonal mean a greater runtime difference. The dotted lines in parallel to the diagonal mark areas where runtimes differ by more than a factor of 2 and 5, respectively. In the top plot a plus (+) marks an unsatisfiable problem and a cross (×) stands for a satisfiable problem.

Solved faster	set	tree
all	1 133	1 417
unsatisfiable	1 007	1 299
satisfiable	126	118

Solved faster by factor	set	tree
2	60	149
5	27	58

Table 8.1: Breakdown of the 2 550 problems solved within the time limit with both tree and set labels. The table on the left shows the number of problems solved faster with set than with tree labels, and vice versa, by the satisfiability of the problem. The table on the right counts the number of problems where the runtime with tree labels is greater than the runtime with set labels by more than factors of 2 and 5 and vice versa.

faster by the label implementation denoted on the bottom X-axis, a point below the diagonal is solved faster by the label implementation denoted at the left Y-axis.

From the top plot in Figure 8.5 we can deduce that tree labels speeded up solving problems often significantly. Table 8.1 summarises the numbers of problems contained in the figure. We find tree labels to be faster than set labels in the majority of cases in total and for unsatisfiable problems. Further, even for longer runtimes there were 149 and 58 problems that tree labels solved faster by a factor of 2 and 5, respectively, while set labels had an advantage of this magnitude only for 60 and 27 problems, respectively. If we consider satisfiable problems, the picture changes and set labels are faster than tree labels on 126 problems, while tree labels solve only 118 problems in a shorter time. This can be explained by the normal form of set labels, which enables finding more literals to be equal up to renaming than tree labels, thus the set of literals to become saturated earlier. However, in absolute numbers neither set nor tree labels exhibit differences in runtimes of more than 50%.

The plots comparing OBDD labels with set and tree labels show most of the data points close to the diagonal, indicating that the advantage of OBDD labels in absolute numbers is not great. However, it has to be noted that the plots contain significantly less problems than the plot above and many problems tackled with OBDD labels fail due to the memory limit.

We report a final observation on the number of problems solved by considering satisfiable and unsatisfiable problems separately. Set labels solved 27% of the 9 110 unsatisfiable problems and 26% of the 939 satisfiable problems, while tree labels solved about 26% in both cases. Contrary to these nearly uniform success

rates across satisfiable and unsatisfiable problems, OBDD labels succeeded on 15% of the unsatisfiable problems, but this rate dropped to 3% on satisfiable problems. For a satisfiable problem the set of labelled literals has to become saturated under inferences, which in the case of OBDD labels can consume a lot of memory. OBDD labelled unit superposition reaches the memory limit before saturation and aborts.

The results so far confirm our expectations about the performance of the different label structures. Due to the natural normal form in set labels there are more problems solved overall, whereas the precise elimination of redundancy gives tree labels an advantage when runtime is considered. OBDD labels lag behind, because computing a normal form of a Boolean formula is hard. If we double the time limit from 120 seconds to 240 seconds, we observe that 51 problems are solved with OBDD labels in addition to the 1 383 problems solved before. These contain 10 problems not solved with set or tree labels within 120 seconds, of which one problem remains unsolved even after 240 seconds. In our experiments most of the unsuccessful attempts to solve a problem with OBDD labels were terminated due to the memory limit. Memory was not a big issue for set or tree labels, where the majority of unsuccessful attempts of a problem were aborted when the time limit was reached.

8.3 Simplification with Unit Clauses

In Chapter 6 we have enhanced labelled unit superposition calculi with two methods for redundancy elimination. We have defined a demodulation inference rule and subsumption of labelled literals by labelled literals derived purely from literals in unit clauses.

In this section we evaluate the two simplification techniques within set and tree labelled unit superposition, we do not consider OBDD labels any further here. We ran iProver-Eq again on the 10 049 selected TPTP problems with a time limit of 120 seconds, once with set labels and once with tree labels, each in a first round without demodulation and subsumption, afterwards with demodulation and subsumption separately enabled and finally with both demodulation and subsumption as in the previous section.

Without subsumption by literals from unit clauses, demodulation only had a mildly positive effect in both set and tree labelled unit superposition. Set labelled

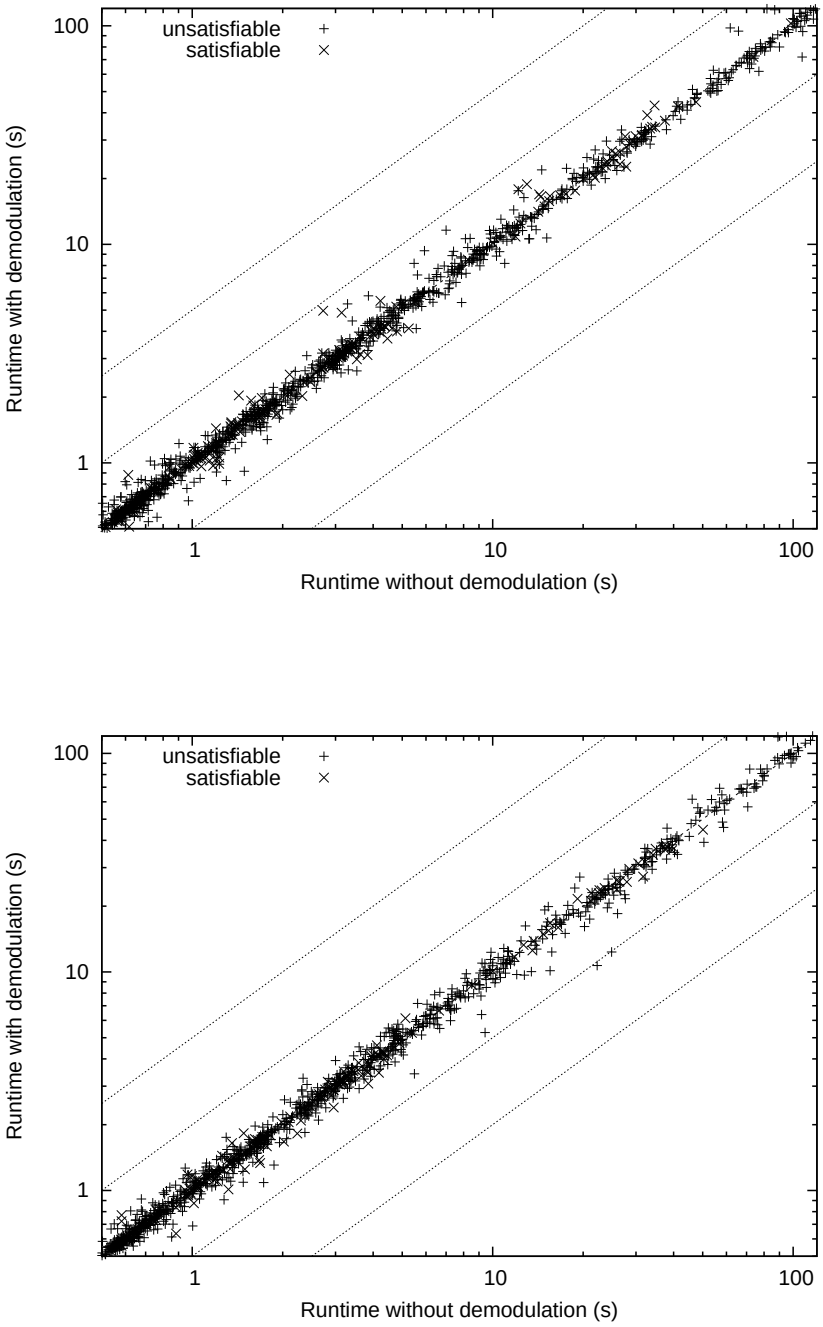


Figure 8.6: Unit superposition without subsumption by literals from unit clauses. The top graph compares the runtimes of problems solved with set labelled unit superposition with and without demodulation, the bottom graph shows the same for tree labels. The graphs are on a logarithmic scale, the dotted lines in parallel to the diagonal mark areas where runtimes differ by more than a factor of 2 and 5, respectively. A plus (+) marks an unsatisfiable problem and a cross (×) stands for a satisfiable problem.

unit superposition solved 2 356 problems both with and without demodulation, but demodulation enabled solving three more difficult problems that were not finished in the time limit without demodulation, while losing three easier problems solved before. With tree labels the situation is similar: without demodulation 2 281 problems were solved, switching on demodulation succeeded on an additional four slightly higher rated problems and lost three easier problems. The graphs in Figure 8.6 show a comparison of the runtimes with and without demodulation for set and tree labels, respectively. The performance of demodulation did not depend on the satisfiability of the problem.

A drastic improvement can be seen when enabling subsumption by literals from unit clauses. There were 435 (set labels) and 404 (tree labels) new problems solved that were not solved without subsumption, neither with nor without demodulation. On the other hand, 57 (set labels) and 46 (tree labels) problems were solved only when subsumption was disabled and 2 302 (set labels) and 2 239 (tree labels) problems were solved both with subsumption enabled and without. However, the picture is not as clear when we compare the runtimes of the problems solved in Figure 8.7. Some problems saw a speed up by a factor greater than two (62 with set labels and 57 with tree labels) while a smaller number was slowed down by factors greater than two (36 for set labels and 25 for tree labels). Overall, the runtime decreased and increased for about the same number of problems. As before, the satisfiability of the problem had no influence.

Without demodulation we find a similar situation when enabling or disabling subsumption with literals from unit clauses. In set labelled unit superposition there were an additional 203 problems solved and 59 problems missed, the numbers for tree labels were 213 and 45. The comparison of the runtimes in Figure 8.8 shows the effect seen before: a few problems are solved faster or slower by greater factors, while overall the increases and decreases in runtime occur to about the same amount.

We summarise that in the current implementation with default settings redundancy elimination has an overall positive effect on the performance of iProver-Eq, in particular solving a number of problems not solved without. Demodulation by itself does not have a great influence, the overhead of maintaining indexes and rewriting literals even leads to a certain slowdown on some of problems, although one would expect to be able to exploit as demodulators the unit clauses occurring in 97% of the TPTP problems to a greater degree. On the other hand, those unit

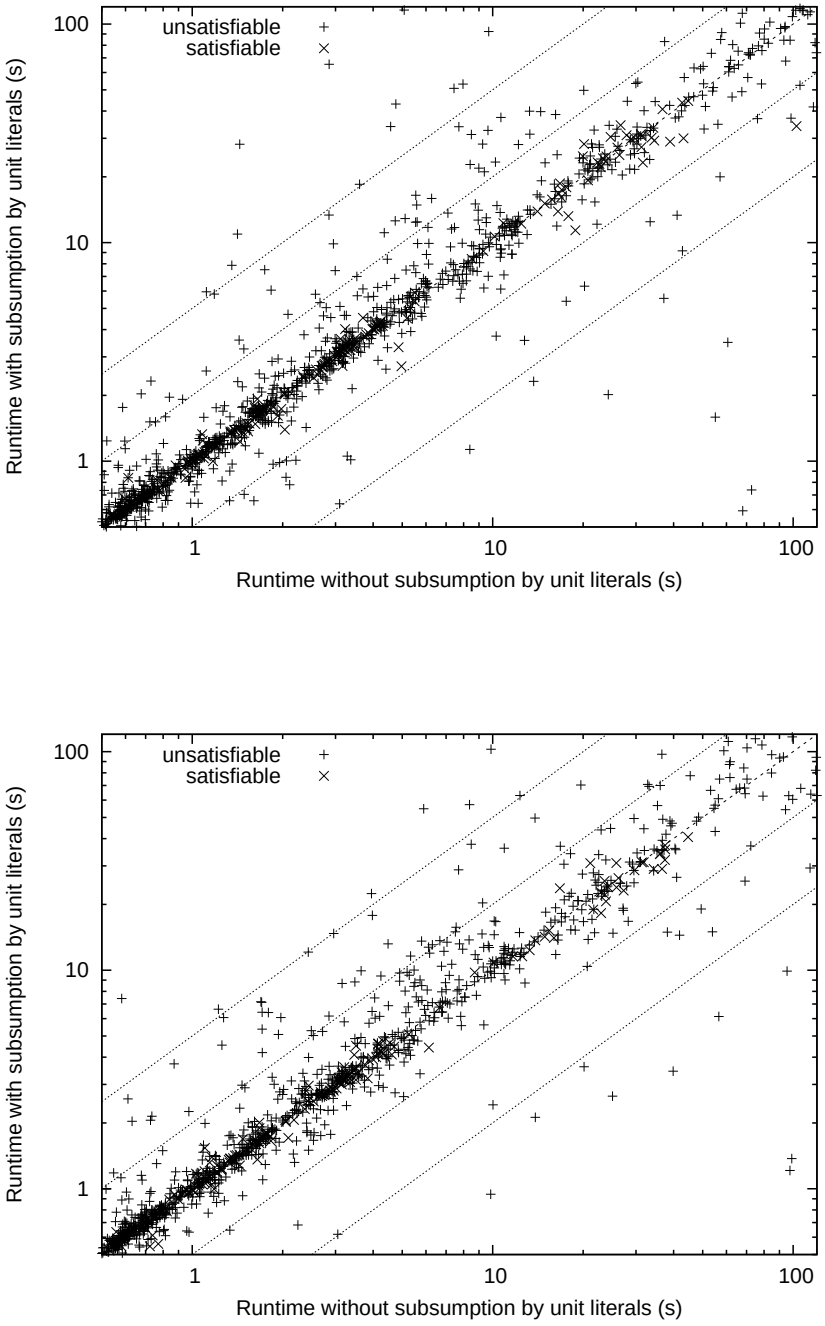


Figure 8.7: Unit superposition with demodulation and subsumption by unit literals enabled or disabled. The top graph compares the runtimes of problems solved in set labels, the bottom graph shows the same for tree labels. The graphs are on a logarithmic scale, the dotted lines in parallel to the diagonal mark areas where runtimes differ by more than a factor of 2 and 5, respectively. A plus (+) marks an unsatisfiable problem and a cross (×) stands for a satisfiable problem.

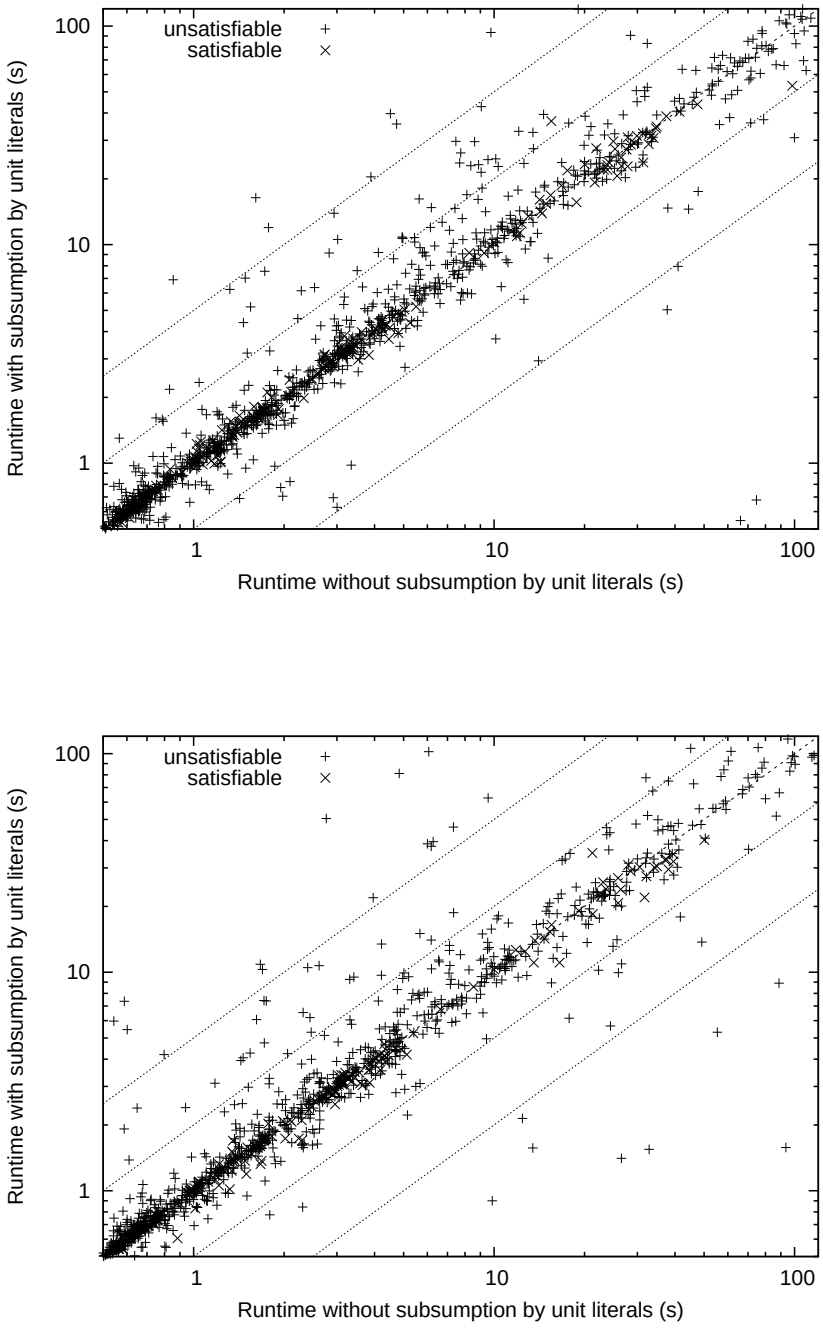


Figure 8.8: Set labelled (top) and tree labelled (bottom) unit superposition without demodulation and subsumption by unit literals enabled and disabled. The graphs are on a logarithmic scale, the dotted lines in parallel to the diagonal mark areas where runtimes differ by more than a factor of 2 and 5, respectively. A plus (+) marks an unsatisfiable problem and a cross (×) stands for a satisfiable problem.

Problems	solved	not solved	fastest
set	2 737	968	815
tree	2 643	1 062	879
Inst-Gen	2 905	800	2 011

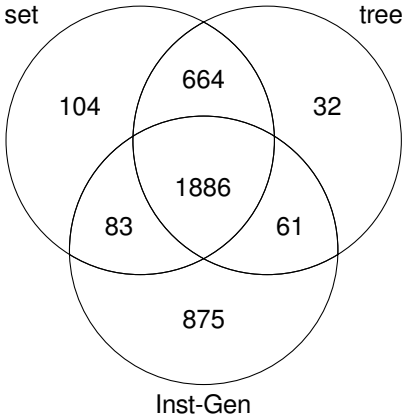


Figure 8.9: Number of solved problems in Inst-Gen-Eq with set labelled and tree labelled unit superposition and with axiomatic equational reasoning in Inst-Gen.

clauses are useful by means of subsumption that has a positive effect even without demodulation and is amplified when demodulation inferences are performed.

8.4 Equational Reasoning and Ground solving

The Inst-Gen-Eq method requires a ground solver modulo equality and we have integrated the SMT solver CVC3. In the Inst-Gen method ground solving reduces to the propositional satisfiability problem and iProver delegates the ground reasoning to the SAT solver MiniSat.

Since the implementation of the Inst-Gen calculus from the iProver system was kept in the iProver-Eq system, we can directly compare the performance of equational reasoning with labelled unit superposition to the previous approach of equational reasoning with axioms.

As one would expect, a very different set of problems is successfully solved with the Inst-Gen implementation compared to the Inst-Gen-Eq implementation with set and tree labels, see Figure 8.9. There are 1 886 problems solved by all three implementations, but Inst-Gen-Eq with set and tree labelled unit superposition and Inst-Gen with axiomatic equality miss 968, 1 062 and 800 problems, respectively.

If we compare the runtimes of solved problems, we find an extremely non-uniform picture. Overall the non-equational calculus is fastest on 2 011 problems, compared to 815 problems and 879 problems, where Inst-Gen-Eq with set and tree labelled unit superposition comes first, respectively. The graphs in Figure 8.10

comparing runtimes of set and tree labelled Inst-Gen-Eq with Inst-Gen show a distribution that seems almost random. Although for short runtimes there are some clusters of problems indicating that the non-equational approach is faster, there are many problems for which the runtimes differ by orders of magnitude. The situation is equal if we look at satisfiable and unsatisfiable problems separately.

The source of the very different performance becomes obvious when we emphasise the ground solving as one main difference of the Inst-Gen and the Inst-Gen-Eq approach. Inst-Gen-Eq has to deal with a significant overhead from invoking an SMT solver, while Inst-Gen relies on a smaller and efficient SAT solver. Figures 8.11 and 8.12 show the time spent in ground solving in relation to the total runtime of the problem for solved and unsolved problems. Propositional satisfiability can be decided quickly and in almost all problems solved in Inst-Gen not more than 5% of the runtime is taken by the MiniSat solver with a median of 1.3%. For a problem to be solved in Inst-Gen-Eq CVC3 as the ground solver has to decide ground satisfiability modulo equality, which is harder than propositional satisfiability. There are problems, where almost all of the runtime is spent in CVC3, the median of the total runtime used for ground solving is 26% for both set labels and tree labels. Problems not solved due to time or memory limits show a similar pattern, here SAT solving with MiniSat takes a median of 1.5% of the total runtime and CVC3 for ground solving modulo equality accounts for 32% and 16% of the total runtime with set labels and tree labels, respectively.

The TPTP library contains only 26 problems with equality in the Bernays-Schönfinkel fragment, all of which are classified as easy and solved by both iProver and iProver-Eq, mostly in less than one second. This does not allow a significant comparison of reasoning in this class between the axiomatic approach in iProver and the superposition approach in the iProver-Eq system. We can only report the trend that almost all satisfiable problems are solved faster with iProver by factors greater than two, whereas unsatisfiable problems see both increases and decreases in runtime by smaller factors. However, these remarks have to qualified with the short runtimes of less than one second, where other effects may overshadow the differences in the calculi.

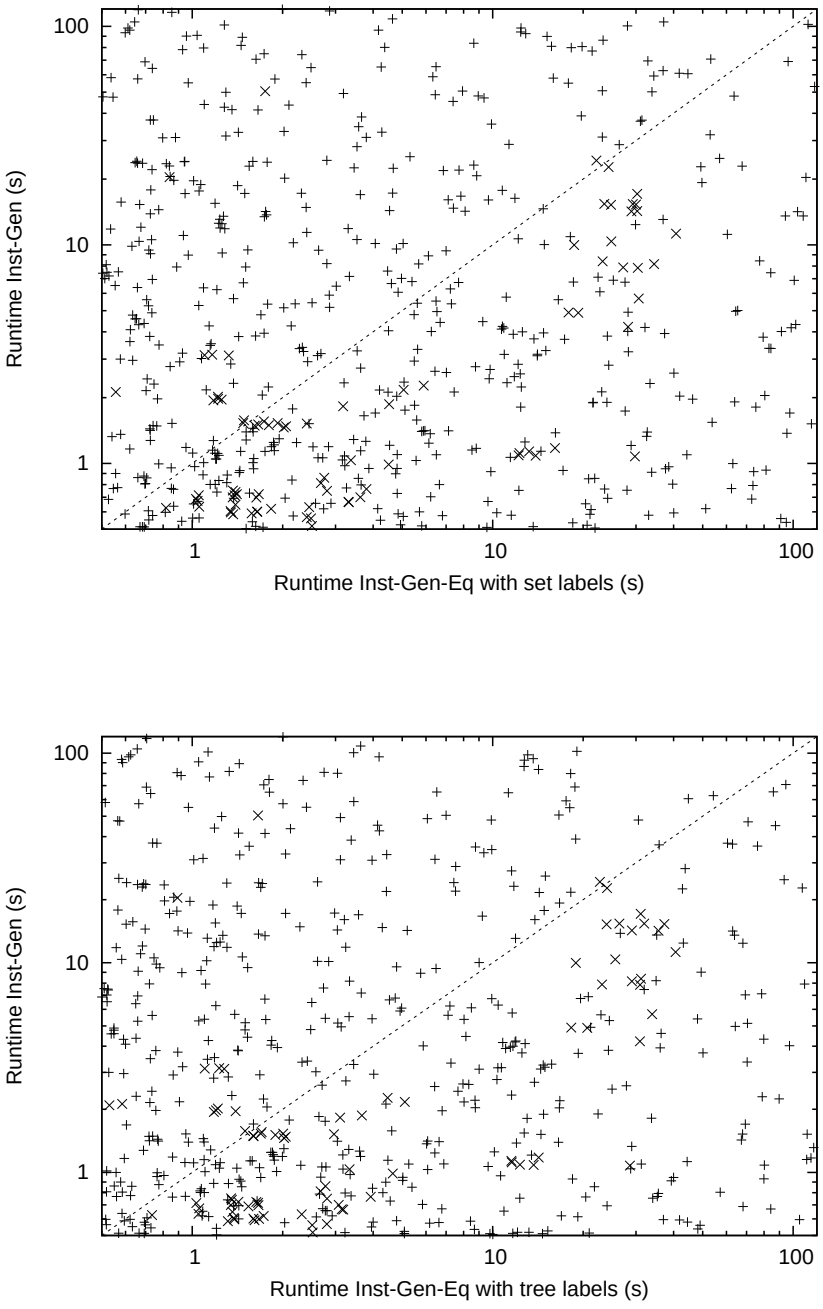


Figure 8.10: Runtimes of solved problems Inst-Gen-Eq with set labelled (top) and tree labelled (bottom) unit superposition with demodulation and subsumption by unit literals and in non-equational Inst-Gen The graphs are on a logarithmic scale, a plus (+) marks an unsatisfiable problem and a cross (×) stands for a satisfiable problem.

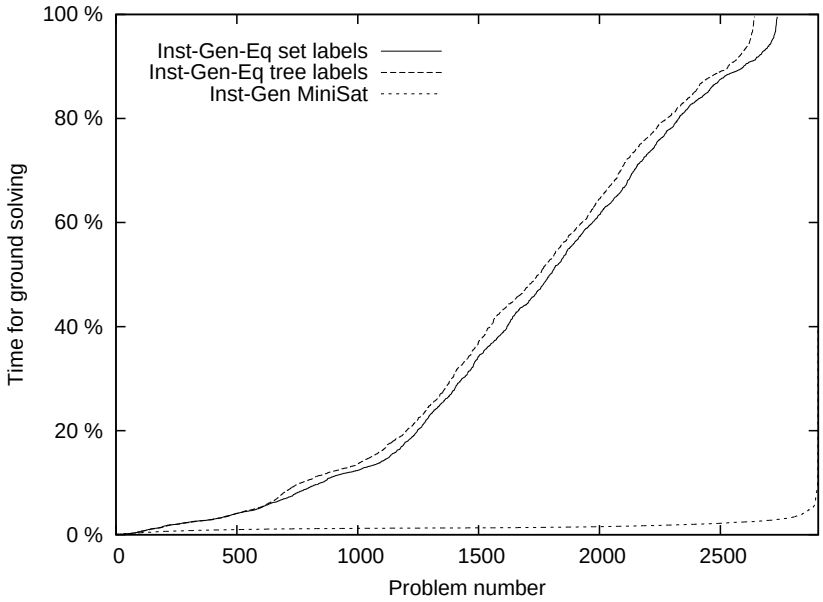


Figure 8.11: Number of solved problems where at most the given share of the total runtime is taken by the ground solver.

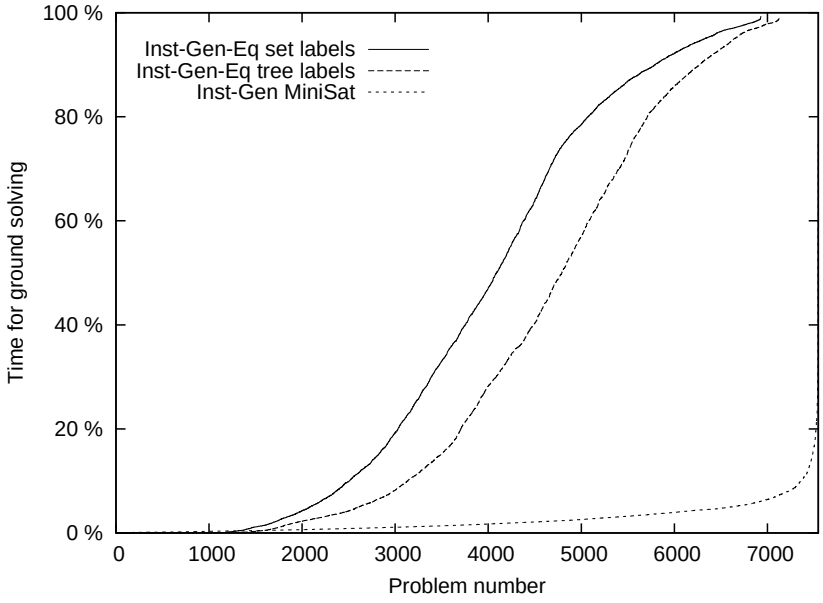


Figure 8.12: Number of not solved problems where at most the given share of the total runtime is taken by the ground solver.

8.5 The CADE ATP Systems Competition

Finally, we discuss the results of the most recent CADE ATP Systems Competition (CASC-J5), held in summer 2010 during the fifth International Joint Conference on Automated Reasoning (IJCAR). Although the CASC is by no means an exhaustive evaluation, it does provide a forum to compare state-of-the-art automated reasoning systems in a competition setting on a small subset of the TPTP. The specifics of CASC-J5 are described in Sutcliffe [2011], a more general overview is in Sutcliffe and Suttner [2006].

The selection of TPTP problems for the competition is done randomly and biased towards recent problems with a number of previously unseen problems mixed in. Only problems with a difficulty rating within certain bounds are eligible in order to get challenging but not unsolvable problems. In the competition a CPU time limit of 300 seconds was enforced and additionally no problem was allowed a wallclock runtime of twice the CPU time limit.

There are three major divisions with several sub-divisions each, based on syntactic properties of the problems. The Bernays-Schönfinkel fragment, in this context called effectively propositional (EPR), is a separate division, containing problems in CNF form with function symbols of non-zero arity only. The other two divisions are made up of FOF and CNF problems.

The FOF problems are not in clausal normal form and further divided into problems without equality (FNE), problems with equality (FEQ) and effectively propositional problems (FEP). All CNF problems contain function symbols of arities greater than zero and are thus not effectively propositional. The CNF division is split by the Horn property, that is, at most one negative literal per clause, and the content of equality, resulting in the following sub-divisions: Horn problems without equality (HNE), Horn problems with equality but not purely equational (HEQ), Non-Horn problems without equality (NNE), Non-Horn problems with equality but not purely equational (NEQ) and purely equational problems (PEQ).

The iProver-Eq system discussed here was entered as iProver-Eq 0.6 into all three divisions, competing instantiation-based systems were iProver in versions 0.7 and 0.8, Darwin, E-Darwin and E-KRHyper, see Chapter 3 for a discussion of the relation of these systems to iProver-Eq.

In the FOF and CNF divisions (Tables 8.2 and 8.3) iProver-Eq was placed in the middle alongside the E-Darwin system and below the non-equational iProver system, although there are problems iProver-Eq could solve and that were not

solved by the iProver system.

The EPR division (Table 8.4) is clearly dominated by instantiation-based systems, followed by dedicated model finding tools (Paradox and Equinox), while the E and Vampire systems that lead the FOF and CNF divisions are trailing behind. It is further notable that the equational systems iProver-Eq and E-Darwin are beaten by their non-equational counterparts iProver and Darwin. This is due to the initially mentioned fact that the equational axioms for a signature without non-zero function symbols do not introduce function symbols, thus an equational problem remains in the Bernays-Schönfinkel fragment even after the addition of the axioms. The non-equational systems can rely on their efficient procedures, while the equational systems have the overhead of superposition reasoning. In particular iProver-Eq needs to invoke the SMT solver, while in iProver the SAT solver is sufficient. As the results show, treating equality axiomatically in the Bernays-Schönfinkel fragment can be an advantage over non-axiomatic equational reasoning, although this does not generalise to full first-order logic and there are EPR problems that are solved faster with superposition reasoning than with the axiomatic approach.

8.6 Summary

Our own evaluation shows the potential of the Inst-Gen-Eq calculus and its implementation in the iProver-Eq system. As the CASC results have demonstrated, the system is already competitive with state-of-the-art systems and in particular on one level with other instantiation-based systems.

We have also exhibited weaknesses of the system, however, since we have focussed the evaluation only on the main contributions of the thesis, we have chosen rigid default values for many other influential settings. In particular the parameters for the priority queues of passive clauses in the Inst-Gen-Eq-loop and passive literals in the US-loop can be fine-tuned to particular problems. Further parameters are the ratio of the Inst-Gen-Eq-loop and the US-loop or the heuristics for selecting a clause literal. In our experience manually adjusting these parameters makes iProver-Eq solve problems significantly faster than before and enables solving problems that have failed before.

We have not done more detailed code profiling beyond recording the time taken by the ground solver as reported in Section 8.4. A detailed account of

Table 8.2: CASC-J5 results in the FOF division, excerpted from Sutcliffe [2011]. The sub-divisions are: problems without equality and not effectively propositional (FNE), problems with equality and not effectively propositional (FEQ) and effectively propositional problems (FEP). Instantiation-based systems are highlighted.

ATP System	FOF /200	Avg CPU	FNE /50	FEQ /125	FEP /25
Vampire 0.6	178	14.2	44	112	22
E 1.2pre	143	10.2	38	88	17
EP 1.2pre	143	14.8	38	88	17
Vampire 11.0	125	44.9	36	73	16
iProver 0.8	*117*	*25.6*	*39*	*63*	*15*
Equinox 5.0	108	17.4	18	66	24
iProver-Eq 0.6	*90*	*18.1*	*37*	*30*	*23*
E-Darwin 1.3	*81*	*25.3*	*23*	*37*	*21*
leanCoP 2.2	54	42.6	30	22	2
Zenon 0.6.3	46	36.0	20	7	19
LEO-II 1.2	45	45.0	19	26	0
Geo 2010C	42	22.7	17	25	0
Metis 2.2	36	32.7	17	18	1
E-KRH' 1.1.4	*32*	*24.3*	*22*	*9*	*1*
Otter 3.3	18	9.3	4	14	0
Muscadet 4.0	15	3.9	5	9	1
Ayane 2	0	–	0	0	0

Table 8.3: CASC-J5 results in the CNF division, excerpted from Sutcliffe [2011]. The sub-divisions are: Horn problems without equality (HNE), Horn problems with some (not pure) equality (HEQ), Non-Horn problems with no equality (NNE), Non-Horn problems with some (not pure equality) (NEQ) and purely equational problems (PEQ). Instantiation-based systems are highlighted.

ATP System	CNF /200	Avg CPU	HNE /30	HEQ /30	NNE /30	NEQ /70	PEQ /40
Vampire 0.6	170	15.1	29	25	28	55	33
Vampire 10.0	168	15.7	29	27	28	54	30
E 1.2pre	161	14.9	23	25	27	57	29
iProver 0.8	*92*	*16.4*	*20*	*7*	*28*	*37*	*0*
Equinox 5.0	77	26.2	8	11	20	33	5
E-Darwin 1.3	*69*	*20.8*	*13*	*11*	*9*	*28*	*8*
iProver-Eq 0.6	*66*	*19.8*	*20*	*2*	*25*	*16*	*3*
E-KRH' 1.1.4	*58*	*38.0*	*18*	*10*	*12*	*17*	*1*
LEO-II	52	54.9	11	11	6	14	10
Otter 3.3	42	11.2	10	14	8	7	3
Geo 2010C	41	30.8	8	11	16	5	1
Metis 2.2	39	27.4	3	4	10	8	14
Ayane 2	1	26.4	0	0	0	1	0

Table 8.4: CASC-J5 results in the EPR division, excerpted from Sutcliffe [2011]. The sub-divisions are: effectively propositional theorems (unsatisfiable) (EPT) and effectively propositional non-theorems (satisfiable) (EPS). Instantiation-based systems are highlighted.

ATP System	EPR /100	Avg CPU	EPS /50	EPT /50
iProver 0.8	*95*	*26.4*	*45*	*50*
iProver 0.7	*84*	*27.1*	*42*	*42*
Darwin 1.4.5	*72*	*19.4*	*22*	*50*
iProver-Eq 0.6	*59*	*44.0*	*11*	*48*
E-Darwin 1.3	*52*	*8.4*	*11*	*41*
Paradox 4.0	51	4.5	8	43
Equinox 5.0	44	24.3	8	36
E-KRH' 1.1.4	*30*	*16.5*	*3*	*27*
Geo 2010C	25	2.8	0	25
E 1.2pre	15	22.5	5	10
Vampire 0.6	13	86.7	13	0
Metis 2.2	8	157.9	3	5
Ayane 2	0	–	0	0

which parts of the system consume the most runtime will be useful for fine tuning the parameters as well as to pinpoint bottlenecks that deserve more optimisation effort.

Nevertheless, the evaluation with rigid defaults already gives hints on directions for further work and we expect that an evaluation and profiling with fine-tuning will provide more insights.

Chapter 9

Conclusion and Further Work

The work in this thesis has made significant contributions to efficient equational reasoning in the instantiation-based Inst-Gen-Eq method.

The Inst-Gen method for instantiation-based reasoning [Ganzinger and Korovin, 2003] is based on a modular combination of first-order reasoning on literals and ground satisfiability solving of an abstraction of the first-order clauses. Clause instances are generated from conflicts in first-order and propagated to the ground abstraction such that the model of the ground abstraction is refined. This process continues until either all conflicts have become irrelevant or the ground abstraction is found to be unsatisfiable, which in turn proves unsatisfiability of the input first-order clause set. The ground solver is further used by means of a selection function on clauses to guide the first-order reasoning on literals towards relevant conflicts.

The Inst-Gen-Eq method [Ganzinger and Korovin, 2004] keeps this paradigm, but we have to replace the single inference rule, which is sufficient to detect conflicts in the non-equational Inst-Gen method, with a calculus on first-order literals. Clause instances are obtained from proofs of contradictions in this calculus.

We have given a detailed completeness proof of the Inst-Gen-Eq method, extending the originally published proof by strengthening the ordering constraints in the inference rule, thus advancing from ordered unit paramodulation to unit superposition, a step that is not obvious in the context of instantiation-based methods. We have also included provisions that serve to justify simplification inferences with unit clauses.

Motivated by the need to deal with literal variants in a robust way we have presented labelled unit superposition calculi, the main contribution of the thesis.

The central feature of the labelled approach is a new merging inference rule to combine literal variants that are initially considered disjoint. The labels keep track of information about the proof structure and eagerly calculate the relevant instances from proofs such that an explicit treatment of proof trees is not necessary.

We have presented the simple label structure of sets, which have the advantage of natural normal forms, but only approximate redundancy elimination. In order to achieve precise redundancy elimination we have to consider labels with a Boolean structure such as AND/OR trees. Since these labels are not produced in a normal form, we have investigated ordered binary decision diagrams (OBDDs) as a label structure that combines precise elimination of redundancy with a normal form.

In order to prevent potential non-termination already in rather simple cases and to obtain a decision procedure for the Bernays-Schönfinkel fragment it is essential to consider normal forms of labels. Tree and OBDD labels with their Boolean structures are isomorphic to monotone Boolean formulae, but deciding logical equivalence in this class is already **coNP**-complete. Therefore, probably no efficient normal form of AND/OR tree labels exists in general. We have discussed ways to improve OBDD labelled unit superposition and hinted on possible other approaches to normal forms of AND/OR tree labels.

We have then turned to literals in unit clauses, which have a special status in the saturation process of selected literals. Since they are in every set of selected literals, they can be used for simplification inferences that persist across selection changes induced by the ground solver. We have shown that a literal from a unit clause makes all variants of it derived in a unit superposition calculus redundant, such that they can be discarded without performing the otherwise necessary merging inference. We have further investigated demodulation as a simplification inference rule based on rewriting with literals from unit clauses. Due to the interaction between the first-order reasoning on literals and the ground satisfiability solving, demodulation is only complete if additional instances are generated.

The iProver-Eq system was developed for this thesis based on the iProver system that implements the non-equational Inst-Gen calculus. All aspects of equational reasoning in Inst-Gen-Eq presented so far have been implemented in the iProver-Eq system using state-of-the-art techniques. We have discussed the main features and in particular focussed on the three constituent parts, namely,

ground solving, which is delegated to an SMT solver, and the two nested saturation processes of instantiation and unit superposition.

Using the TPTP benchmark library that contains a wide variety of up to date problems from relevant application areas, we have evaluated our contributions to equational reasoning with the Inst-Gen-Eq method. In particular we have compared set, tree and OBDD labelled unit superposition calculi, the simplification inferences based on unit clauses and the performance of ground solving by the SMT solver.

We have found that a number of problems was solved by all three labelled calculi, but that in particular set and tree labelled unit superposition show some specialisation and a significant number of problems could only be solved successfully with one of the calculi. OBDD labelled unit superposition lags behind the two simpler labelled calculi, which is to be expected from our discussion of the complexity issues of computing normal forms of tree labels. Nevertheless, OBDD labelled unit superposition is faster by some margin than both other labelled calculi on a significant number of problems.

We have entered the iProver-Eq system into the annual CASC competition, the "world championship of theorem provers", where it was on a par with other instantiation-based systems and not too far from the leading and longer established systems.

Further Work

There are a number of directions for further work based on the results in this thesis.

As we have mentioned in the discussion of the evaluation, in our experience fine-tuning of parameters can have a significant effect on the performance of the system. In particular the ratio between the priority queues for passive literals and passive clauses in the US-loop and the Inst-Gen-Eq-loop, respectively, can have a big influence. Certain settings may prevent the system from solving a problem at all, such that the approach of rigid default values is not optimal. Strategies to adapt the parameters to a current problem or changes at runtime will certainly improve the general performance of iProver-Eq.

A more interesting problem is finding the optimal labelled calculus for an input set of clauses. Our results suggest a hybrid approach to labelling, combining, for

example, set and AND/OR tree labels. In this context it is certainly instructive to investigate, which classes of problems a particular labelling approach is best suited to. We have already mentioned and discussed alternatives to OBDD labelled unit superposition, which combines the advantages of a normal form with precise redundancy elimination. However, there is the barrier of high computational complexity such that it might not be possible to find a generally efficient solution.

Our evaluation has further shown that the ground solving modulo equality with the SMT solver is a bottleneck. Since there is considerable current activity in the SMT community, a number of tools have been developed. Our implementation is modular to allow exchanging the currently used CVC3 solver for a different solver. Due to the variety of approaches to SMT solving, it would not be surprising to see certain SMT solvers being better suited than others to the kind of problems discharged from the iProver-Eq system.

The modular approach of the Inst-Gen method delegates the ground reasoning to a black boxed solver, expecting to be able to exploit its sophisticated techniques. However, it is certainly worth investigating if closer cooperation between the ground reasoning and first-order reasoning up to an opening of the black box would be beneficial. The ground reasoning influences the first-order reasoning through the selection function, which is based on a model of the ground abstraction. Selecting a literal requires performing all possible inferences with all literals in the US-saturation process. Since there are equations such as $x \simeq t$, where x is a variable and t is a term, which generate many conclusions, it is beneficial to exploit the non-determinism when choosing a ground model to make the selection function select a different literal whenever possible. As an extension of this approach, prolific literals having produced many inferences can be detected at runtime and attempts can be made to coerce the ground solver to provide a different model not containing the prolific literal.

The information in labels is at the moment used only for the purposes of redundancy elimination and extraction of relevant clause instances. However, we can take information from labels to replace an SMT solver with a simpler SAT solver by generating lemmas from labels. From a conflict on first-order literals, clause instances are generated and propagated to the ground abstraction in the solver. The ground solver must be able to reason modulo equality so that it can witness the conflict on relevant instances in the ground abstraction. The solver is then forced to refine the ground model. If we replace a ground

solver modulo equality with a propositional solver, we can achieve the same effect by explicitly generating lemmas in addition to the relevant instances from the conflict. However, since the propositional solver is agnostic about equality, it is possible that literals are selected, which are already inconsistent modulo equality in the ground abstraction. In this case the unit superposition calculus finds a contradiction, where none of the relevant instances in the label is proper and only lemmas are generated that force the ground solver to revise the selection. Thus, an approach with lemma generation and propositional satisfiability solving will find more contradictions and place more burden on the saturation process on literals than the approach of a ground solver modulo equality, where equational this equational reasoning happens in the ground solver. For tree and OBDD labels the information in the label of a contradiction is sufficient to provide the ground solver with such lemmas, there is not enough information in set labels, however. Preliminary experiments with lemma generation from tree labels show promising potential, in many cases delivering about equal performance to using an SMT solver without lemma generation.

Finally, as we have mentioned in the beginning, a natural line of research is to move beyond equational reasoning in Inst-Gen-Eq to reasoning modulo theories. Theory reasoning in the style of Inst-Gen has already been discussed by Ganzinger and Korovin [2006], where the role of the unit superposition calculus is taken by a unit calculus for the theory to be solved modulo. It is necessary to develop calculi that, similar to unit superposition, find contradictions modulo a theory on a set of selected literals and provide relevant instances for a ground solver modulo the same theory in order to refine the ground abstraction. This is certainly no easy task and it is not clear if the labelling approach presented here can be lifted to any theory.

Moreover, satisfiability of a set of ground clauses is not decidable for many relevant theories like arithmetic and we have to deal with an incomplete ground solver that does not terminate or may return only a partial model or for literal selection.

The approach of lemma generation from labels allows to use a weaker ground solver than the unit reasoning if appropriate lemmas are generated from conflicts. Therefore lemma generation could be used to replace a potentially not terminating SMT solver with a SAT solver, if it is possible to define labelled unit calculi.

Bibliography

Parosh Abdulla, Per Bjesse, and Niklas Eén. Symbolic Reachability Analysis Based on SAT-Solvers. In Susanne Graf and Michael Schwartzbach, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 6th International Conference, TACAS 2000. Proceedings*, volume 1785 of *Lecture Notes in Computer Science*, pages 411–425, Berlin / Heidelberg, 2000. Springer.

Henrik R. Andersen and Henrik Hulgaard. Boolean Expression Diagrams. *Information and Computation*, 179(2):194–212, December 2002.

Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998.

Franz Baader and Wayne Snyder. Unification Theory. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, chapter 8, pages 445–532. Elsevier and MIT Press, 2001.

Leo Bachmair and Harald Ganzinger. Rewrite-based Equational Theorem Proving with Selection and Simplification. *Journal of Logic and Computation*, 4(3): 217–247, June 1994.

Leo Bachmair and Harald Ganzinger. Equational Reasoning in Saturation-Based Theorem Proving. In Wolfgang Bibel and Peter. H. Schmitt, editors, *Automated Deduction - A Basis for Applications*, number 1 in Applied Logic Series, chapter 11, pages 353–397. Kluwer Academic Publishers, 1998.

Leo Bachmair and Harald Ganzinger. Resolution Theorem Proving. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, chapter 2, pages 19–99. Elsevier and MIT Press, 2001.

Leo Bachmair, Harald Ganzinger, Christopher Lynch, and Wayne Snyder. Basic Paramodulation. *Information and Computation*, 121(2):172–192, 1995.

Thomas Ball, Ella Bounimova, Rahul Kumar, and Vladimir Levin. SLAM2: Static Driver Verification with Under 4% False Alarms. In Roderick Bloem and Natasha Sharygina, editors, *Formal Methods in Computer-Aided Design, 10th International Conference, FMCAD 2010. Proceedings*, 2010. To appear.

Clark Barrett and Cesare Tinelli. CVC3. In Aarti Gupta and Sharad Malik, editors, *Computer Aided Verification, 20th International Conference, CAV 2007. Proceedings*, volume 4590 of *Lecture Notes in Computer Science*, pages 298–302, Berlin / Heidelberg, 2007. Springer.

Clark Barrett, Morgan Deters, Albert Oliveras, and Aaron Stump. Design and results of the 3rd annual satisfiability modulo theories competition (SMT-COMP 2007). *International Journal on Artificial Intelligence Tools*, 17(04):569–606, 2008.

Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB Standard: Version 2.0. In Aarti Gupta and Daniel Kroening, editors, *8th International Workshop on Satisfiability Modulo Theories, SMT 2010. Proceedings*, 2010.

Peter Baumgartner. FDPLL - A First Order Davis-Putnam-Longeman-Loveland Procedure. In David A. McAllester, editor, *Automated Deduction - CADE-17, 17th International Conference on Automated Deduction. Proceedings*, volume 1831 of *Lecture Notes in Computer Science*, pages 200–219, Berlin / Heidelberg, 2000. Springer.

Peter Baumgartner. Logical Engineering with Instance-Based Methods. In Frank Pfenning, editor, *Automated Deduction - CADE-21, 21st International Conference on Automated Deduction. Proceedings*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 404–409, Berlin / Heidelberg, 2007. Springer.

Peter Baumgartner and Evgenij Thorstensen. Instance Based Methods — An Overview. *KI - Künstliche Intelligenz*, 24(1):35–42, 2009.

Peter Baumgartner and Cesare Tinelli. The Model Evolution Calculus. In Franz Baader, editor, *Automated Deduction - CADE-19, 19th International Conference on Automated Deduction. Proceedings*, volume 2741 of *Lecture Notes in Computer Science*, pages 350–364, Berlin / Heidelberg, 2003. Springer.

Peter Baumgartner and Cesare Tinelli. The Model Evolution Calculus with Equality. In Robert Nieuwenhuis, editor, *Automated Deduction - CADE-20, 20th International Conference on Automated Deduction. Proceedings*, volume 3632 of *Lecture Notes in Computer Science*, pages 392–408, Berlin / Heidelberg, 2005. Springer.

Peter Baumgartner and Uwe Waldmann. Superposition and Model Evolution Combined. In Renate A. Schmidt, editor, *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction. Proceedings*, volume 5663 of *Lecture Notes in Computer Science*, pages 17–34, Berlin / Heidelberg, 2009. Springer.

Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. Implementing the Model Evolution Calculus. *International Journal on Artificial Intelligence Tools*, 15(1):21–52, 2005. Special Issue on Empirically Successful First Order Reasoning.

Peter Baumgartner, Björn Pelzer, and Cesare Tinelli. Model Evolution with Equality – Revised and Implemented. Submitted for journal publication, June 2010.

Bernhard Beckert, Tony Hoare, Reiner Hähnle, Douglas R. Smith, Cordell Green, Silvio Ranise, Cesare Tinelli, Thomas Ball, and Sriram K. Rajamani. Intelligent Systems and Formal Methods in Software Engineering. *Intelligent Systems, IEEE*, 21(6):71–81, 2006.

Dan Benanav, Deepak Kapur, and Paliath Narendran. Complexity of matching problems. In *Rewriting Techniques and Applications, First International Conference, RTA-85. Proceedings*, volume 202 of *Lecture Notes in Computer Science*, pages 417–429, Berlin / Heidelberg, 1985. Springer.

Jean-Paul Billon. The disconnection method. In *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 1996. Proceedings*, volume 1071 of *Lecture Notes in Computer Science*, pages 110–126, Berlin / Heidelberg, 1996. Springer.

Beate Bollig and Ingo Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, 45(9):993–1002, Sept 1996.

Thomas Bouton, Caminha, David Déharbe, and Pascal Fontaine. veriT: An Open, Trustable and Efficient SMT-Solver. In Renate A. Schmidt, editor, *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction. Proceedings*, volume 5663 of *Lecture Notes in Computer Science*, pages 151–156, Berlin / Heidelberg, 2009. Springer.

Roberto Bruttomesso, Edgar Pek, Natasha Sharygina, and Aliaksei Tsitovich. The OpenSMT solver. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010. Proceedings*, volume 6015 of *Lecture Notes in Computer Science*, pages 150–153, Berlin / Heidelberg, 2010. Springer.

Randal E. Bryant. Graph-Based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

Randal E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR)*, 24(3):293–318, September 1992.

Randal E. Bryant. Binary decision diagrams and beyond: enabling technologies for formal verification. In *IEEE/ACM International Conference on Computer-Aided Design, ICCAD-95. Digest of Technical Papers*, pages 236–243. IEEE/ACM, 1995.

Ricardo Caferra and Nicolas Zabel. A method for simultaneous search for refutations and models by equational constraint solving. *Journal of Symbolic Computation*, 13(6):613–641, 1992.

Hubert Comon. Disunification: a survey. In *Computational Logic: Essays in Honor of Alan Robinson*, pages 322–359, 1991.

Werner Damm, Stefan Disch, Hardi Hungar, Swen Jacobs, Jun Pang, Florian Pigorsch, Christoph Scholl, Uwe Waldmann, and Boris Wirtz. Exact state set representations in the verification of linear hybrid systems with large discrete state space. In Kedar S. Namjoshi, Tomohiro Yoneda, Teruo Higashino, and Yoshio Okamura, editors, *Automated Technology for Verification and Analysis, 5th International Symposium, ATVA 2007. Proceedings*, volume 4762 of *Lecture Notes in Computer Science*, pages 425–440, Berlin / Heidelberg, 2007. Springer.

Martin Davis and Hilary Putnam. A Computing Procedure for Quantification Theory. *Journal of the ACM*, 7(3):201–215, 1960.

Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

Leonardo de Moura and Nikolaj Bjørner. Deciding Effectively Propositional Logic Using DPLL and Substitution Sets. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning, 4th International Joint Conference, IJCAR 2008. Proceedings*, volume 5195 of *Lecture Notes in Computer Science*, pages 410–425, Berlin / Heidelberg, 2008a. Springer.

Leonardo de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340, Berlin / Heidelberg, 2008b. Springer.

Jörg Denzinger, Martin Kronenburg, and Stephan Schulz. DISCOUNT - A Distributed and Learning Equational Prover. *Journal of Automated Reasoning*, 18 (2):189–198, April 1997.

R. Drechsler and B. Becker. Ordered Kronecker functional decision diagrams-a data structure for representation and manipulation of Boolean functions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17 (10):965–973, Oct 1998.

Thomas Eiter, Wolfgang Faber, and Patrick Traxler. Testing strong equivalence of datalog programs - implementation and examples. In Chitta Baral, Gianluigi Greco, Nicola Leone, and Giorgio Terracina, editors, *Logic Programming and Nonmonotonic Reasoning, 8th International Conference, LPNMR 2005. Proceedings*, volume 3662 of *Lecture Notes in Computer Science*, pages 437–441, Berlin / Heidelberg, 2005. Springer.

Moshe Emmer, Zurab Khasidashvili, Konstantin Korovin, and Andrei Voronkov. Encoding Industrial Hardware Verification Problems into Effectively Propositional Logic. In Roderick Bloem and Natasha Sharygina, editors, *Formal Methods in Computer-Aided Design, 10th International Conference, FMCAD 2010. Proceedings*. IEEE, October 2010. URL `http://fmcad10.iaik.tugraz.at`. to appear.

Jean C. Filliâtre and Sylvain Conchon. Type-safe modular hash-consing. In Andrew Kennedy and François Pottier, editors, *Proceedings of the ACM Workshop on ML*, pages 12–19, New York, NY, USA, 2006. ACM.

Harald Ganzinger and Konstantin Korovin. New Directions in Instantiation-Based Theorem Proving. In *18th IEEE Symposium on Logic in Computer Science, LICS 2003. Proceedings*, pages 55–64. IEEE Computer Society, 2003.

Harald Ganzinger and Konstantin Korovin. Integrating Equational Reasoning into Instantiation-Based Theorem Proving. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *Computer Science Logic, 18th International Workshop, CSL 2004. Proceedings*, volume 3210 of *Lecture Notes in Computer Science*, pages 71–84, Berlin / Heidelberg, 2004. Springer.

Harald Ganzinger and Konstantin Korovin. Theory Instantiation. In Miki Hermann and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 13th International Conference. Proceedings*, number 4246 in Lecture Notes in Computer Science, pages 497–511, Berlin / Heidelberg, 2006. Springer.

P. C. Gilmore. A proof method for quantification theory: its justification and realization. *IBM Journal of Research and Development*, 4:28–35, January 1960.

Judy Goldsmith, Matthias Hagen, and Martin Mundhenk. Complexity of DNF and isomorphism of monotone formulas. In Joanna Jędrzejowicz and Andrzej Szepietowski, editors, *Mathematical Foundations of Computer Science 2005, 30th International Symposium, MFCS 2005. Proceedings*, volume 3618 of *Lecture Notes in Computer Science*, pages 410–421, Berlin / Heidelberg, 2005. Springer.

Peter Graf. *Term Indexing*, volume 1053 of *Lecture Notes in Computer Science*. Springer, Berlin / Heidelberg, 1996.

J. N Hooker, G. Rago, V. Chandru, and A. Shrivastava. Partial Instantiation Methods for Inference in First-Order Logic. *Journal of Automated Reasoning*, 28(4):371–396, 2002.

Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reducing SHIQ- Description Logic to Disjunctive Datalog Programs. In Mary-Anne Williams Didier Dubois,

Christopher A. Welty, editor, *Principles of Knowledge Representation and Reasoning, KR2004, Ninth International Conference. Proceedings*, pages 152–162. AAAI Press, 2004a.

Ulrich Hustadt, Renate A. Schmidt, and Lilia Georgieva. A Survey of Decidable First-Order Fragments and Description Logics. *Journal on Relational Methods in Computer Science*, 1:251–276, 2004b.

Swen Jacobs and Uwe Waldmann. Comparing Instance Generation Methods for Automated Reasoning. In *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2005. Proceedings*, volume 3702 of *Lecture Notes in Computer Science*, pages 153–168, Berlin / Heidelberg, 2005. Springer.

Roope Kaivola, Rajnish Ghughal, Naren Narasimhan, Amber Telfer, Jesse Whittemore, Sudhindra Pandav, Anna Slobodová, Christopher Taylor, Vladimir Frolov, Erik Reeber, and Armaghan Naik. Replacing Testing with Formal Verification in Intel CoreTM i7 Processor Execution Engine Validation. In *Computer Aided Verification, 22nd International Conference, CAV 2009. Proceedings*, pages 414–429, 2009.

U. Kebschull, E. Schubert, and W. Rosenstiel. Multilevel logic synthesis based on functional decision diagrams. In *Design Automation, 3rd European Conference. Proceedings*, pages 43–47, 1992.

Konstantin Korovin. An Invitation to Instantiation-Based Reasoning: From Theory to Practice. In A. Podelski, A. Voronkov, and R. Wilhelm, editors, *Volume in memoriam of Harald Ganzinger*, Lecture Notes in Computer Science. Springer, Berlin / Heidelberg, 2009. Invited paper. To appear.

Konstantin Korovin. iProver - An Instantiation-Based Theorem Prover for First-Order Logic (System Description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning, 4th International Joint Conference, IJCAR 2008. Proceedings*, volume 5195 of *Lecture Notes in Computer Science*, pages 292–298, Berlin / Heidelberg, 2008. Springer.

Konstantin Korovin and Christoph Sticksel. Labelled Unit Superposition Calculi for Instantiation-Based Reasoning. In Christian G. Fermüller and Andrei

Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 17th International Conference. Proceedings*, volume 6397 of *Lecture Notes in Computer Science*, pages 459–473, Berlin / Heidelberg, 2010a. Springer.

Konstantin Korovin and Christoph Sticksel. iProver-Eq: An Instantiation-Based Theorem Prover with Equality. In Jürgen Giesl and Reiner Hähnle, editors, *Automated Reasoning, 5th International Joint Conference, IJCAR 2010. Proceedings*, volume 6173 of *Lecture Notes in Computer Science*, pages 196–202, Berlin / Heidelberg, 2010b. Springer.

Sava Krstić and Amit Goel. Architecting Solvers for SAT Modulo Theories: Nelson-Oppen with DPLL. In Boris Konev and Frank Wolter, editors, *Frontiers of Combining Systems, 6th International Symposium, FroCoS 2007. Proceedings*, volume 4720 of *Lecture Notes in Computer Science*, pages 1–27, Berlin / Heidelberg, 2007. Springer.

Reinhold Letz and Gernot Stenz. DCTP - A Disconnection Calculus Theorem Prover - System Abstract. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Automated Reasoning, First International Joint Conference, IJCAR 2001. Proceedings*, volume 2083 of *Lecture Notes in Computer Science*, pages 381–385, Berlin / Heidelberg, 2001a. Springer.

Reinhold Letz and Gernot Stenz. Proof and Model Generation with Disconnection Tableaux. In Robert Nieuwenhuis and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 8th International Conference, LPAR 2001. Proceedings*, volume 2250 of *Lecture Notes in Computer Science*, pages 142–156, Berlin / Heidelberg, 2001b. Berlin / Heidelberg.

Reinhold Letz and Gernot Stenz. Integration of Equality Reasoning into the Disconnection Calculus. In *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2002. Proceedings*, pages 176–190, 2002.

William McCune and Larry Wos. Otter - The CADE-13 Competition Incarnations. *Journal of Automated Reasoning*, 18(2):211–220, April 1997.

Shin-ichi Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proceedings of the 30th Design Automation Conference*, pages 272–277, New York, New York, USA, 1993. ACM Press.

Robert Nieuwenhuis and Albert Rubio. Paramodulation-based theorem proving. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, chapter 7, pages 371–443. Elsevier and MIT Press, 2001.

Andreas Nonnengart and Christoph Weidenbach. Computing Small Clause Normal Horms. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, chapter 6, pages 335–367. Elsevier and MIT Press, 2001.

Christos M. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

Juan Antonio Navarro Pérez and Andrei Voronkov. Encodings of Bounded LTL Model Checking in Effectively Propositional Logic. In Frank Pfenning, editor, *Automated Deduction - CADE-21, 21st International Conference on Automated Deduction. Proceedings*, volume 4603 of *Lecture Notes in Computer Science*, pages 346–361, Berlin / Heidelberg, 2007. Springer.

David A. Plaisted and Yunshan Zhu. Ordered Semantic Hyper-Linking. *Journal of Automated Reasoning*, 25(3):167–217, 2000.

Steffen Reith. On the Complexity of Some Equivalence Problems for Propositional Calculi. In Branislav Rovan and Peter Vojtáš, editors, *Mathematical Foundations of Computer Science 2005, 28th International Symposium, MFCS 2003. Proceedings*, volume 2747 of *Lecture Notes in Computer Science*, pages 632–641, Berlin / Heidelberg, 2003. Springer.

Alexandre Riazanov and Andrei Voronkov. Limited resource strategy in resolution theorem proving. *Journal of Symbolic Computation*, 36(1-2):101–115, 2003.

John Alan Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM*, 12(1):23–41, 1965.

Renate A. Schmidt. Decidability by Resolution for Propositional Modal Logics. *Journal of Automated Reasoning*, 22(4):379–396, 1999.

Stephan Schulz. E – A Brainiac Theorem Prover. *Journal of AI Communications*, 15(2/3):111–126, 2002.

R. Sekar, I. V. Ramakrishnan, and Andrei Voronkov. Term Indexing. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated*

Reasoning, volume 2, chapter 26, pages 1853–1964. Elsevier and MIT Press, 2001.

Geoff Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, July 2009.

Geoff Sutcliffe. The 5th IJCAR automated theorem proving system competition – CASC-J5. *AI Communications*, 24(1):75–89, January 2011.

Geoff Sutcliffe and Christian Suttner. Evaluating general purpose automated theorem proving systems. *Artificial Intelligence*, 131(1-2):39–54, September 2001.

Geoff Sutcliffe and Christian Suttner. The State of CASC. *AI Communications*, 19(1):35–48, 2006.

Seiichiro Tani, Kiyoharu Hamaguchi, and Shuzo Yajima. The complexity of the optimal variable ordering problems of shared binary decision diagrams. In K. Ng, P. Raghavan, N. Balasubramanian, and F. Chin, editors, *Algorithms and Computation, 4th International Symposium, ISAAC '93. Proceedings*, volume 762 of *Lecture Notes in Computer Science*, pages 389–398, Berlin / Heidelberg, 1993. Springer.

Nikolai Tillmann and Jonathan de Halleux. Pex – White Box Test Generation for .NET. In Bernhard Beckert and Reiner Hähnle, editors, *Tests and Proofs, 2nd International Conference, TAP 2008. Proceedings*, volume 4966 of *Lecture Notes in Computer Science*, pages 134–153, Berlin / Heidelberg, 2008. Springer.

Olga Tveretina and Wieger Wesselink. EufDpll - a tool to check satisfiability of equality logic formulas. In *Mathematical Foundations of Computer Science and Information Technology, Irish Conference, MFCSIT 2006. Proceedings*, volume 225, pages 405–420. Elsevier, 2009.

Christoph Weidenbach, Renate Schmidt, Thomas Hillenbrand, Rostislav Rusev, and Dalibor Topic. System Description: Spass Version 3.0. In Frank Pfenning, editor, *Automated Deduction - CADE-21, 21st International Conference on Automated Deduction. Proceedings*, volume 4603 of *Lecture Notes in Computer Science*, pages 514–520, Berlin / Heidelberg, 2007. Springer.

Lawrence Wos, George A. Robinson, Daniel F. Carson, and Leon Shalla. The Concept of Demodulation in Theorem Proving. *Journal of the ACM*, 14:698–709, October 1967.

Index

Appendix A

Errata

This section lists errors that were found after submission of the thesis and are corrected in this edition. The line and page numbers refer to the original edition.

p. 25, l. 5 Corrected $[x/z, y/z]$ to $[z/x, z/y]$

p. 34, Def. 3.2 (v) Corrected $(s[l']\sigma \simeq t)\,\sigma$ to $(s[l'] \simeq t)\,\sigma$

p. 61, l. 5 Corrected "all $i > i_0$" to "infinitely many i"

p. 63, l. 3 Corrected $I^m_\bot \models C_{n+1}$ to $I^m_\bot \models C_{n+1}\bot$

p. 63, l. 4 Deleted "and $\mathrm{sel}^m(C_{n+1}) = L$ for some literal $L \in C_{n+1}$"

p. 63, l. 21 Corrected L to L_i in $x \in \mathrm{var}(L)$ and $L \cdot \theta'_i$

p. 63, l. 19 Corrected $\mathcal{L}$ to K

p. 111, l. 6 Corrected $C \cdot \sigma$ to $C \cdot \theta$