# Efficient reasoning in equational theories

Author: André Duarte

Supervisor: Konstantin Korovin

Co-supervisor: Renate Schmidt

*A thesis submitted for the degree of Doctor of Philosophy in the Faculty of Science and Engineering*

Department of Computer Science
School of Engineering
Faculty of Science and Engineering
University of Manchester
2023

*Blank page*

# Contents

**List of Figures** **6**

**List of Tables** **7**

**List of Inference Rules** **8**

**List of Algorithms** **9**

**Abstract** **10**

**Lay abstract** **11**

**Declaration and copyright statement** **12**

**Acknowledgements** **15**

**Notation** **17**

**Outline** **19**

**1 Introduction** **21**

1.1 First-order theorem proving . . . . . . . . . . . . . . . . . . . . . 24
1.2 Summary of contributions . . . . . . . . . . . . . . . . . . . . . . 32
1.3 Published work . . . . . . . . . . . . . . . . . . . . . . . . . . . 33

**2 Preliminaries** **37**

2.1 Basics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
2.2 First-order logic . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
Syntax . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39
Semantics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41

2.3 Equational completion . . . . . . 42
2.4 Superposition calculus . . . . . . 42
Properties . . . . . . 43

**3 Closure redundancy in superposition** **45**
3.1 Closures and closure orderings . . . . . . 45
Closure orderings . . . . . . 46
Examples . . . . . . 48
Properties . . . . . . 49
3.2 Closure redundancy . . . . . . 52
3.3 Model construction . . . . . . 55

**4 Simplifications** **65**
4.1 Demodulation . . . . . . 67
4.2 Joinability . . . . . . 69
Ground joinability . . . . . . 71
Connectedness . . . . . . 75
Ground connectedness . . . . . . 79
4.3 Associativity and commutativity . . . . . . 80
AC joinability . . . . . . 82
AC normalisation . . . . . . 85
AC demodulation . . . . . . 87

**5 Algorithms for theorem proving** **91**
5.1 Ground joinability . . . . . . 91
Incremental ground joinability . . . . . . 93
5.2 Saturation strategy . . . . . . 104
Flexible simplification setup . . . . . . 105

**6 Experimental results** **111**
6.1 Ground joinability and AC criteria . . . . . . 113
Experimental design . . . . . . 114
Results . . . . . . 115
Encompassment demodulation . . . . . . 124
Connectedness . . . . . . 125
6.2 iProver loop . . . . . . 125
6.3 CADE ATP System Competition . . . . . . 127

7 Conclusion and outlook 131

A iProver loop experimental setup 137

Index 141

Bibliography 143

Word count: 34 216

# List of Figures

6.1 Number of problems solved exclusively with and without ground joinability, and problems solved in both. . . . . . . . . . . . . . . . . . . 117

6.2 Number of successful ground joinability simplifications in each problem, among problems where ground joinability is attempted at least once. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 118

6.3 Problems solved with general ground joinability, with only AC joinability, and with neither. . . . . . . . . . . . . . . . . . . . . . . . . . . 119

6.4 Cumulative number of problems solved under a given time, with and without ground joinability. . . . . . . . . . . . . . . . . . . . . . . 120

6.5 Percentage of total runtime spent testing general ground joinability (excl. AC), among problems where ground joinability is attempted at least once, and with total runtime $> 1$ s. . . . . . . . . . . . . . . . 122

6.6 Percentage of total runtime spent testing AC joinability, among problems where ground joinability is attempted at least once, and with total runtime $> 1$ s. . . . . . . . . . . . . . . . . . . . . . . . . . . . 123

6.7 Number of problems solved by each combination of loops. . . . . . 128

# List of Tables

6.1 Number of problems solved. . . . . . . . . . . . . . . . . . . . . . . . . 116
6.2 Average runtime among problems solved with and without ground joinability. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 120
6.3 Number of problems solved significantly faster with or without ground joinability, among problems solved in both experiments, and with runtime $> 1$ s (schedule off). . . . . . . . . . . . . . . . . . . 120
6.4 Hard or unsolved problems in TPTP, solved by iProver with ground joinability. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 124
6.5 Total number of problems solved by each loop. . . . . . . . . . . . . 128
6.6 Average runtime among problems solved by all loops. . . . . . . . . 128

# List of Inference Rules

Unfailing completion . . . . . 42
Superposition . . . . . 43
Equality resolution . . . . . 43
Equality factoring . . . . . 43
Standard demodulation . . . . . 67
Encompassment demodulation . . . . . 67
Joinability . . . . . 70
Ground joinability . . . . . 73
Connectedness . . . . . 78
Ground connectedness . . . . . 79
AC joinability . . . . . 82
AC normalisation . . . . . 86
AC demodulation . . . . . 88

# List of Algorithms

1 Incremental ground joinability test . . . . . . . . . . . . . . . . . . 99
2 iProver given clause saturation loop. . . . . . . . . . . . . . . . . . 108

## Abstract

Problems in many theories axiomatised by unit equalities (UEQ), such as groups, loops, lattices, and other algebraic structures, are notoriously difficult for automated theorem provers to solve. Consequently, there has been considerable effort over decades in developing techniques to handle these theories, notably in the context of Knuth-Bendix completion and derivatives. The superposition calculus is a generalisation of completion to full first-order logic; however it does not carry over all the refinements that were developed for it, and is therefore not a strict generalisation. This means that (i) as of today, even state of the art superposition provers, while more general, are outperformed in UEQ by provers based on completion, and (ii) the sophisticated techniques developed for completion are not available in any problem which is not in UEQ.

In particular, this includes key simplifications such as ground joinability, which have been known for more than 30 years; however previous completeness proofs rely on proof orderings and proof reductions, which are not easily extensible to general clauses together with redundancy elimination.

In this work we introduce a novel notion of redundancy of clauses and inferences, and a proof of refutational completeness of the superposition calculus wrt. this notion of redundancy. Then, we use it to derive a plethora of simplification rules and redundancy criteria, including an extension of ground joinability, which elegantly generalises this simplification from equational completion (in UEQ) to superposition (over arbitrary clauses); and an extension of the connectedness critical pair criterion to superposition. We also study the theory of associative-commutative operators in particular, including deriving a stronger notion of normalisation modulo this theory, and showing where demodulation modulo AC can be applied while preserving refutational completeness.

Implementing these techniques efficiently is itself a non-trivial task, therefore we have proposed novel algorithms for tackling two key issues: the test for ground joinability of two terms, and the scheduling of simplifications in a given-clause loop.

We have implemented most of the techniques described herein in a theorem prover, iProver, including the aforementioned novel algorithms, and evaluated over the TPTP library with encouraging results.

## Lay abstract

Logic is a tool to model thought. By carefully laying out our premises and performing rigorous logical deductions on them, we can minimise the chances of error in our reasoning. Computers can help us by automating the process of finding or checking solutions to logical problems.

However, solving problems purely by first-principles logical deductions can be excruciatingly slow. While there exist computer programs which can — theoretically — find a solution to any problem which has one, given enough time and space, we find that they perform rather poorly on certain classes of problems, or certain theories. In this work, we showed how reasoning in equational theories can be simplified and accelerated, all the while retaining the nice theoretical guarantees that if a solution exists, it will eventually be found.

This has value beyond the academic. For instance, we know that computer bugs are a source of security issues, billions of pounds of damages annually, and even real-life hazard to people. But those computer systems themselves obey logical rules. By specifying the properties we wish a program to have (e.g. that it has no bugs of a certain type) in a suitable logical language, we can attempt to prove that a given program, as written, obeys those properties. Those proofs would be totally unfeasible to check by hand (and we would always have to trust that the human who checked them did not make any mistake), but automated reasoning software can tackle even very large tasks and either produce proofs of correctness, or else point out where the mistake is.

In this work we improve the state of the art of this problem through several novel theoretical contributions. We propose algorithms for computing those theoretical results. Finally, we also implement them in a theorem proving program, and show some experimental results.

The Author hereby declares that no portion of the work referred to in the 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.

*Blank page*

# Acknowledgements

First, I would like to thank my supervisor, Dr. Konstantin Korovin, for his ceaseless support and guidance throughout this project, since the very first day, and for the many insights and suggestions shared with me throughout the making of this project. I would also like to thank my co-supervisor, Dr. Renate Schmidt, and my two advisors, Dr. André Freitas and Dr. Louise Dennis. Furthermore, I would also like to thank my viva examiners, Dr. Maria Paola Bonacina and Dr. Uli Sattler, for their extremely thorough revision and their valuable critiques and suggestions, which were taken into account for the preparation of the final version of this manuscript.

On a more extended level, I am indebted to the wonderful people of the Department of Computer Science at Manchester, my colleagues and friends, for their companionship, support, and many productive (or not-so-productive) chats around the coffee machine. A word also to the outstanding automated reasoning community, especially to the anonymous reviewers of the papers I submitted, for their valuable comments and insights.

Finally, I would also like to extend my heartfelt thank you to my family, for the herculean task of putting up with me for over a quarter of a century, and to Margarida, my dear companion of many years, whose presence never once fails to cheer me up. Never in a million years would I have made it without their support.

This project has been funded by a University of Manchester scholarship (AA00772).

*Blank page*

This table summarises the notation used throughout this work. Unless otherwise noted, the symbols on the left — possibly super- and sub-scripted and primed — denote the objects on the right.

| Symbol | Meaning |
|---|---|
| $s, t, u, v, l, r$ | Terms |
| $p, q$ | Atoms, literals |
| $C, D$ | Clauses |
| $e$ | Expressions: term, atom, literal or clause |
| $\theta, \sigma, \rho$ | Substitutions |
| $f, g, h$ | Function symbols, with arity $> 0$ |
| $a, b, c$ | Constant symbols, with arity $= 0$ |
| $x, y, z, w$ | Variables |
| $>$ | Total order (strict) |
| $\succ$ | Partial order (strict) |
| $\succeq$ | Preorder (non-strict) |
| $s \approx t$ | Positive equality literal |
| $s \not\approx t$ | Negative equality literal |
| $s \dot{\approx} t$ | Either of the latter two |
| $\neg p$ | Negation of a literal |
| $e\theta$ | Application of substitution $\theta$ to expression $e$ |
| $e \cdot \theta$ | Closure with expression $e$ and substitution $\theta$ |
| $e[l \mapsto r]$ | Replacement of all occurrences of $l$ in $e$ by $r$ |
| $\rhd$ | Subterm relation |
| $\sqsupset$ | Encompassment relation |
| $\models$ | Semantic implication |
| $\vdash$ | Syntactic implication |

*Blank page*

# Outline

In Chapter 1 we give an introduction to the research field and how it relates to the topic of our work, discuss motivation and practical applications, and give a brief summary of the major contributions in this dissertation. In Chapter 2 we give a self-contained exposition of the necessary background material, and establish the notation, conventions, and definitions to be followed throughout the work.

In Chapter 3 we introduce the definition of closure redundancy, a novel and more powerful notion of redundancy than is employed in existing treatments of superposition. This is based on a rigorously defined family of orderings on closures. We then present a proof of refutational completeness of the superposition calculus with respect to closure redundancy. In Chapter 4 we apply the results of the previous chapter to derive numerous new simplification rules, including some stronger versions of existing rules, and some which have never before been used in superposition. These include ground joinability and connectedness, as well as specialised rules for associative-commutative operators.

In Chapter 5 we discuss some further issues related to implementation, including algorithms for performing these simplifications in practice, how they should be scheduled, how to choose a term ordering, etc. In Chapter 6 we analyse some experimental results from implementing the techniques herein described in a state-of-the-art theorem prover.

Finally, in Chapter 7 we summarise the main results and discuss future work.

*Blank page*

# Chapter 1

# Introduction

All men are mortal. Socrates was mortal.
Therefore, all men are Socrates.

Unknown

Formal logic is the study of abstract formal systems intended to model thought, reasoning, arguments, deduction, and other similar notions. As a discipline of philosophy, logic has been studied for millennia, but modern formal logic has its roots in the mid-19th-century work of Boole, Frege, and many others. Nowadays, systems of formal logic are not only of interest for their own sake, as objects of study in pure mathematics, but also find applications in many different fields. Such systems include propositional logic, predicate logic, various modal logics, and other less mainstream systems such as intuitionist logic, many-valued or fuzzy logic, non-monotone logic, etc.

First-order logic is a particularly interesting system, being expressive enough to conveniently formalise theories and problems in a wide variety of domains (being Turing-complete), but not so powerful that it loses many nice metalogical properties, such as semi-decidability. Perhaps most notably, it is the foremost language of mathematics (e.g. being suitable for formalising arithmetic and various algebraic structures, and also being the language of ZFC [Jech 2006], an axiomatic theory of sets which forms the most common foundation of mathematics). In this work, we will be almost exclusively concerned with first-order logic and its fragments.

Automated reasoning is a field of mathematics and computer science concerned with algorithms for performing logical deductions and other tasks in a given system of formal logic. While it is an old ambition to reduce human thought to mechanical manipulation of symbols,[1] it was only after the advent of modern digital computers that the theory of automated reasoning could be finally put into action. Nowadays, the theory and practice of automated reasoning has produced: efficient decision procedures for propositional logic [Silva and Sakallah 1999; Eén and Sörensson 2003], even when augmented with non-quantified real and integer arithmetic [Moura and Bjørner 2008; Barbosa et al. 2022], complete semi-decision procedures for first-order logic [Robinson and Voronkov 2001] and for higher-order logic with Henkin semantics [Bhayat and Reger 2020; Bentkamp 2021; Vukmirović et al. 2022], proof checkers and proof assistants for higher-order calculi [Nipkow, Paulson and Wenzel 2002], and many other developments. It continues to be an area of active and thriving research.

Automated reasoning can be broadly divided into two main areas:

- **Automated theorem proving**, where the focus is on algorithms and software for performing tasks in a logic system (proving conjectures, finding models, verifying properties, etc.) without any human intervention.

- **Interactive theorem proving**, where the focus is on algorithms and software to verify the correctness of a formal proof written by a human in a specific formal language.

In this work, we are concerned only with automated theorem proving, specifically for first-order logic.

## Applications

Automated theorem proving has numerous practical applications. First-order automated reasoning can be (and has been) applied in numerous fields, from software and hardware verification [Clarke, Khaira and Zhao 1996; Klein et al. 2009; Khasidashvili, Korovin and Tsarkov 2015; Georgiou, Gleiss and Kovács 2020], to natural language processing [Fellbaum 1998], explainable medical diagnoses [Hommersom, Lucas and Bommel 2005], semantic queries on large knowledge

[1]Leibniz famously suggested, centuries before computers were a reality, that if rigorous rules of reasoning are laid out and imprecise language is abstracted away, then verifying that an argument is correct could be accomplished by purely mechanical manipulation of symbols, as in an algebraic problem.

bases [Niles and Pease 2001], and solving long-standing open problems in pure mathematics.[2]

Broadly speaking, automated reasoning is useful for problems which can be formalised in a reasonably convenient fashion in a given logical language, but which are yet too complex or open-ended to have a specialised algorithm.

Software and hardware verification are especially important applications which fit this description nicely. It is feasible, for example, given the source code of a program and a formalisation of the programming language semantics, to check whether the program as written matches a certain formal specification (such as verifying it never reaches some failure state). This has been used to achieve a complete formal verification of a microkernel which today is used in millions of devices such as modems [Klein et al. 2009]. In another example, given a description of a hardware circuit and the function that it is meant to implement, automated reasoning can check whether there are bugs where the circuit does not agree with the target function. This is routinely used in modern chip design.[3]

Furthermore, a recurring theme in automated reasoning is the usage of algorithms for weaker/simpler logics as subroutines in algorithms for stronger logics. For example, Inst-Gen [Korovin 2013] is a procedure for first-order logic which is based around approximating the problem by a propositional problem (propositional satisfiability being much easier to solve than first-order satisfiability), which is then passed to a propositional solver; depending on the result, Inst-Gen can either conclude an answer to the first-order problem or keep refining the approximation. In another example, the AVATAR architecture employs a propositional solver to manage the splitting of first-order clauses, to great experimental success [Voronkov 2014].

Just as first-order provers rely on SAT solving subroutines, Isabelle/HOL, an interactive theorem prover for higher-order logic, uses first-order and propositional provers to attempt to discharge a proof or proof step without user

[2]Two famous examples are the four-color problem, which was reduced by human mathematicians to 1982 cases, which were all proven by computer SAT solving [Appel and Haken 1989], and the Robbins algebra problem, which was solved by a first-order equational prover running for 8 days straight [McCune 1997]. Both problems had eluded mathematicians for decades.

[3]In 1994, a hardware bug in the implementation of floating-point division in Intel CPUs led to $863 million in losses for the company in the subsequent recall (2022 dollars). After this widely-publicised incident, interest in formal verification of hardware took a marked uptick [Clarke, Khaira and Zhao 1996].

intervention (the “Sledgehammer” proof tactic [Blanchette et al. 2016]). This is a highly convenient way for users of interactive theorem provers to automate much of the tedious work involved in formal proofs. Sledgehammers have as much as $>$40% success rate in proving theorems in the Mizar math library [Hales 2014], saving huge amounts of manual work.

What this means is that improvements in, for example, first-order logic theorem proving benefit not only first-order theorem proving directly, but also indirectly improve higher-order theorem provers that build on the former. Hence, the results presented in this dissertation impact both the direct applications of our work, but also those of the tools which build on it.

## 1.1 First-order theorem proving

We will assume that the reader has a sufficient background in automated reasoning. In this section we will present the main problem which motivates our work; we do so by briefly covering the relevant history of automated reasoning and the current state of affairs of first-order equational theorem proving (see e.g. Bachmair and Ganzinger 1998; Bonacina 2022 for a more thorough review). We discuss the shortcomings in that state of affairs, and how our work addresses some of those issues.

Efforts to automate deduction in first-order logic have been ongoing since even before digital computers existed. In the first half of the 20th century there were numerous developments on the metamathematics of first-order logic, spurned by Hilbert’s program for a consistent, complete, and finite axiomatic foundation of mathematics [Zach 2019], including the 1929 discovery that Presburger arithmetic (natural numbers with $+$ and $=$) is decidable [Presburger 1929], meaning in principle an algorithm could determine whether any sentence is true or false in that theory, and the 1931 discovery that any sufficiently strong consistent axiomatic first-order theory (such as Peano arithmetic or ZF set theory) is necessarily incomplete, meaning some formulas are not in the theory and neither is their negation [Gödel 1931]. The seminal Herbrand’s theorem, and the notions of Herbrand universe and Herbrand interpretation, also date from that time [Herbrand 1930].

## Resolution

In 1965, Robinson formulated the resolution rule for first-order logic using syntactical unification [Robinson 1965]. Resolution had previously been described for propositional logic, as the following inference rule

$$\frac{p \lor C \quad \neg p \lor D}{C \lor D}, \tag{1.1}$$

meaning from two clauses with an opposite sign atom we can infer that conclusion. The clauses in (1.1) are composed of boolean atoms, and only used in the context of first-order logic via naïve application of Herbrand's theorem (i.e. via the explicit instantiation of first-order clauses into propositional clauses). Robinson's breakthrough approach instead formulates the resolution rule directly on first-order clauses via the notion of *most general unifier.* For two (first-order) clauses $p \lor C$ and $\neg q \lor D$, where $p$ and $q$ are two unifiable atoms, the first-order resolution rule is simply

$$\frac{p \lor C \quad \neg q \lor D}{(C \lor D)\theta}, \quad \text{where } \theta \text{ is the most general unifier of } p \text{ and } q. \tag{1.2}$$

In an intuitive sense, a first-order resolution inference, by using the most general unifier to instantiate only as far as needed to resolve the clauses, can represent a potentially infinite number of propositional resolution inferences done on ground instances of its two premises, in a single step [Bachmair and Ganzinger 2001].

Robinson showed that exhaustive application of this rule, together with factoring,[4] is refutationally complete, in the sense that it will derive a contradictory clause (the falsum) in a finite (but arbitrarily large) number of steps if one exists, and is consequently a semi-decision procedure for the validity/entailment/unsatisfiability problem in first-order logic. This was a major step forward in first-order automated reasoning, and resolution still forms the basis of many first-order theorem provers on non-equality clauses [McCune 2003; Weidenbach et al. 2009; McCune 2010; Kovács and Voronkov 2013; Duarte and Korovin 2020, etc]. Later, it was shown that ordering constraints and redundancy criteria can be used to further restrict the inferences needed, while maintaining refutational completeness [Bachmair and Ganzinger 2001].

[4]The inference $p \lor q \lor C \vdash (p \lor C)\theta$ where $\theta$ is the most general unifier of $p$ and $q$.

However, the following theory poses a particular challenge. Consider a binary symbol $\approx$, and the following axioms

$$\forall x.\, x \approx x \tag{1.3a}$$

$$\forall xy.\, x \approx y \Rightarrow y \approx x \tag{1.3b}$$

$$\forall xyz.\, x \approx y \wedge y \approx z \Rightarrow x \approx z \tag{1.3c}$$

and, for each function $f$ and predicate $p$,

$$\forall x_1 \dots x_n\, y_1 \dots y_n.\, \bigwedge_i (x_i \approx y_i) \Rightarrow f(x_1, \dots, x_n) \approx f(y_1, \dots, y_n) \tag{1.3d}$$

$$\forall x_1 \dots x_n\, y_1 \dots y_n.\, \bigwedge_i (x_i \approx y_i) \wedge p(x_1, \dots, x_n) \Rightarrow p(y_1, \dots, y_n) \tag{1.3e}$$

The theory axiomatised by (1.3) is the theory of equality, without a doubt the single most important and commonly-occurring theory in first-order logic.

Unfortunately, it is also ill-suited to resolution; its axioms are extremely prolific: at the very least (1.3b–c) unify with any occurrence of the $\approx$ predicate and (1.3d) and (1.3e) unify with any occurrence of a function and predicate symbol respectively. It is apparent that this is a significant hurdle, since a huge number of re-combinations of the axioms and other trivial facts will quickly swamp the search space of a prover applying resolution on equality problems. This lead to significant research efforts to overcome this limitation [Bachmair and Ganzinger 1998]

### Equational completion

In a different field, Knuth and Bendix proposed a groundbreaking algorithm for solving the word problem for an abstract algebra [Knuth and Bendix 1970]. The word problem simply asks: "Given two terms and a set of equations, are the terms equal modulo the equations?". Knuth-Bendix completion tries to solve this by computing from the equations an equivalent set of rewrite rules having the property that two terms are equal iff they have the same normal form wrt. those rewrite rules. The computation of this set may succeed or fail. If it succeeds, the word problem is easily decided for that set of equations: we can simply reduce two terms to their normal forms and check if they are equal.

The main ingredients of the Knuth-Bendix procedure are (i) the computation of critical pairs, and (ii) the notion of a reduction order on terms. The reduction order[5] is used to orient equations into rewrite rules for the system. Then, if there are any *critical pairs* between rewrite rules, they need to be explicitly added as new equations, as per a deduction rule

$$\frac{l \to r \quad s[u] \to t}{s\theta[l\theta \mapsto r\theta] \approx t\theta}, \quad \text{where } \theta \text{ is the most general unifier of } l \text{ and } u \text{ (a subterm of } s \text{ which is not a variable), } l \succ r \text{, and } s \succ t. \tag{1.4}$$

This procedure can be used for equational theorem proving, even when this process would be would be infinite, by simply reducing both goal terms during the completion procedure, wrt. the rewrite system obtained thus far, and stopping when the normal forms of both sides are equal. In this sense, Knuth-Bendix completion computes an *approximation* of a convergent rewrite system and successively refines it. However, if at any point it encounters an equation it cannot orient, the procedure fails.

To get around this problem, the variant of unfailing completion [Hsiang and Rusinowitch 1987; Bachmair, Dershowitz and Plaisted 1989] instead approximates *ground* convergence, by using instead *orientable instances* of the equations as rewrite rules,

$$\frac{l \approx r \quad s[u] \approx t}{s\theta[l\theta \mapsto r\theta] \approx t\theta}, \quad \text{where } \theta \text{ is the most general unifier of } l \text{ and } u \text{ (a subterm of } s \text{ which is, not a variable), } l\theta \not\preceq r\theta \text{, and } s\theta \not\preceq t\theta, \tag{1.5}$$

which is sufficient for theorem proving. Unfailing completion is a semi-decision procedure: if the terms are equal modulo the starting equations, unfailing completion will eventually obtain a sufficiently strong rewrite system to reduce both terms to the same normal form, and conversely, if it terminates without reducing the terms to the same normal form, then they are not equal modulo the starting equations.

An important point to note is that in addition to this "core" inference rule, there are also inferences for simplifying and deleting rewrite rules [Bachmair, Dershowitz and Plaisted 1989; Martin and Nipkow 1990; Bonacina and Hsiang 1995; Löchner and Hillenbrand 2002]. Just as redundancy criteria are vital for the efficiency of resolution, these simplification rules are also crucial for ensuring equational completion's practical effectiveness for purposes of theorem proving.

[5] A partial order with certain properties such as well-foundedness and monotonicity [Baader and Nipkow 1998].

### Superposition

In short, resolution provides us with a semi-decision procedure for first-order logic, while equational completion gives us a (rather efficient) semi-decision procedure for proving unit equational conjectures. Is it possible to combine these two ideas into a system which provides a semi-decision procedure for first-order logic which is *also* efficient for dealing with the theory of equality? The answer is yes, the superposition inference system [Bachmair and Ganzinger 1994] achieves exactly this, the core rule being

$$\frac{\underline{l \approx r} \vee C \quad \underline{s[u] \mathrel{\dot{\approx}} t} \vee D}{(s[u \mapsto r] \mathrel{\dot{\approx}} t \vee C \vee D)\theta}, \quad \text{where } \theta \text{ is the most general unifier of } l \text{ and } u \text{ (not a variable), } l\theta \npreceq r\theta \text{, and } s\theta \npreceq t\theta, \tag{1.6}$$

plus two more local rules to ensure refutational completeness.[6]

We can verify that

- In the <u>unit case</u>, superposition on positive clauses degenerates exactly into the critical pair rule of unfailing completion, while superposition on negative clauses degenerates into reduction of the goal wrt. oriented instances.
- In the <u>non-equality case</u> (representing non-equality atoms $p$ as equalities $p \approx \top$ where $\top$ is the minimal constant in the reduction ordering), superposition degenerates into resolution.[7]

This can be seen as a refinement of the paramodulation rule [Robinson and Wos 1969], published very nearly at the same time as Knuth-Bendix completion (as far as is understood, neither of them knew about the others' endeavours). In effect, superposition is an inference system which generalises resolution, paramodulation, and equational completion [Bonacina 2022].

### Simplifications

However, this is not the whole story. As alluded before, the *generating inferences* (needed for refutational completeness) are only part of the picture, as provers invariably employ *simplification inferences*, in which some or all of the premises

[6]Factoring and equality resolution; in Chapter 2 we present superposition in detail.
[7]As only top terms unify, $\top \approx \top$ is a tautology, and $\top \not\approx \top$ is a contradiction.

are deleted or simplified. For example, both superposition and equational completion admit

$$\cancel{s \approx t}\ , t \approx u \vdash s \approx u, \quad \text{if } s \succ t \succ u. \tag{1.7}$$

where $\cancel{s \approx t}$ denotes that that premise can be deleted after this inference is performed (or in other words, $s{\approx}t$ can be simplified to $s{\approx}u$ if $t{\approx}u$ is present); in order to retain refutational completeness, the clauses being discarded need to be known to be unnecessary for deriving a contradiction. Unfortunately, while the generating rules of superposition degenerate exactly into the generating rules of equational completion, many of the more sophisticated criteria for discarding or simplifying equations in completion do *not* carry over in full generality to superposition [Bonacina 2022]. This is a major problem with important practical consequences.

For the past 25 years, the top-scoring provers in the first-order division of the annual CASC competition have invariably been superposition provers [Sutcliffe 2016]. However, in purely unit-equality problems, equational completion provers still have the edge, sometimes by a substantial margin. For example, in 2019, Waldmeister [Hillenbrand, Buch et al. 1997] outperformed E and Vampire [Schulz 2013b; Kovács and Voronkov 2013], two state-of-the-art superposition provers under active development, despite Waldmeister not having been updated for over 10 years! This continued success is empirical evidence that the simplification and deletion criteria of equational completion might be key ingredients for performance in equational problems.

In practical terms this means both that

(i) Different tools are needed for different fragments of first-order logic, as superposition does not subsume completion, either theoretically or for practical purposes.

(ii) The sophisticated techniques developed for equational completion are not available in any problem which is not unit equality.[8]

This is an unsatisfactory state of affairs on both counts.

The reason for this is that equational completion and superposition are not treated in a single unifying theoretical framework. In particular, equational com-

[8]There are transformations which can encode Horn problems into unit-equational problems, but in the general case such a transformation can be done at best via an incomplete encoding [Claessen and Smallbone 2018].

pletion relies on proof orderings, where an equation can be deleted if there is a smaller proof via existing equations [Bachmair, Dershowitz and Hsiang 1986], while superposition relies on model construction via a well-founded ordering on clause instances, where a clause can be deleted if all of its ground instances follow from smaller ground instances of existing clauses [Nieuwenhuis and Rubio 2001]. This represents a significant hurdle as criteria based on proof orderings and proof reductions do not carry over to general clauses together with ground instance-based redundancy elimination [Bonacina and Hsiang 1995].

This work represents a major step in solving this issue, as we introduce a significantly more powerful notion of redundancy, wrt. which superposition is still refutationally complete, which enables the usage of simplifications from equational completion (and more). In particular, we are able to show that superposition on unit clauses *does* degenerate exactly to equational completion — including simplifications — and elegantly extending to the full clausal case. We are able to derive several simplification rules and criteria for the redundancy of inferences (corresponding roughly to simplification rules and critical pair criteria in equational completion, respectively).

## Equational theories

Some equational theories are particularly troublesome, even for equational completion. The archetypal example is associativity and commutativity. A function which satisfies the following two axioms

$$f(x,y) \approx f(y,x) \qquad\qquad f(x,f(y,z)) \approx f(f(x,y),z) \tag{1.8}$$

is said to be associative and commutative (AC). This theory is extremely ubiquitous in mathematics and other domains, yet it is problematic for equational completion and superposition, as there is no reduction order which can orient the commutativity axioms, and even just the two axioms produce an exponential number of consequences [Martin and Nipkow 1990].

Due to this, considerable effort has been directed towards equational reasoning in AC[9] over the decades [Huet 1980; Stickel 1981; Bachmair and Dershowitz 1987; Anantharaman and Mzali 1989; Martin and Nipkow 1990; Baader and Nipkow 1998; Avenhaus, Hillenbrand and Löchner 2003], and it remains an area

[9]And other equational theories.

of active research. Some proposals involve including the theory in the underlying logic, via specialised inference rules (in a way, similar to what is done with equality itself in superposition). However, these approaches make use of (semantic) AC-unification [Peterson and Stickel 1981; Anantharaman and Mzali 1989], rather than syntactic unification, which is known to be very expensive to calculate.[10] Therefore, provers based on this approach have generally not been able to achieve practical success.

Another proposal has been to make use of joinability and ground joinability criteria [Martin and Nipkow 1990]. In fact, the latter enable the deletion of huge numbers of redundant consequences in permutative theories such as AC (including e.g. almost *all* of the consequences of the axioms). Due to the fact that ground joinability modulo AC can be cheaply implemented [Avenhaus, Hillenbrand and Löchner 2003], this approach has had substantial experimental success in those problems. However, this is one of the techniques which *cannot* be proven correct using the standard theoretical framework for superposition [Nieuwenhuis and Rubio 2001].[11] As such, no superposition prover (as far as we are aware) implements general ground joinability criteria.

In our work, we present (i) general joinability and ground joinability simplification rules, whose application does not compromise refutational completeness subject to some well-defined constraints, and (ii) specialised results for AC (including AC normalisation), which enable simpler/weaker constraints and more efficient implementation. These subsume and elegantly extend the results in e.g. Avenhaus, Hillenbrand and Löchner [2003] to first-order logic.

Another useful family of simplifications has to do with criteria for blocking inferences, such as connectedness-related criteria [Bachmair and Dershowitz 1988]. In the equational completion framework, these are critical pair criteria; in our framework, they correspond to redundant superposition inferences.

## Algorithms

In order to turn this theoretical work into an efficient theorem prover, great care must be taken in terms of the algorithms used to implement these simplification criteria. Two of the main issues we have tackled are (i) the scheduling of

[10]AC-unification is known to be reducible to solving linear Diophantine equations over positive integers [Stickel 1981].

[11]Although one could in principle still use it, if one can make do without guarantees of refutational completeness.

simplifications in a given-clause saturation loop, and (ii) a novel algorithm for efficiently checking ground joinability of literals. These aspects are of crucial importance for the efficiency of the overall theorem proving procedure, and in this work we have provided novel contributions for both these issues.

With such a vast array of simplification rules and inference-blocking criteria, a highly non-trivial question is *how* and *when* to apply them. Intuitively, these simplifications help performance by reducing the search space and preventing unnecessary work; however, performing these simplifications and storing the necessary data has itself a non-negligible cost [Hillenbrand, Piskac et al. 2013]. Deciding how to balance this is a long-standing open problem (examples of mainstream approaches are e.g. McCune [1990], Denzinger, Kronenburg and Schulz [1997] and Jakubuv and Urban [2017]). In our work, we propose a flexible simplification setup which subsumes common architectures, and enables meta-optimisation using e.g. machine learning approaches [Holden and Korovin 2021].

As for ground joinability, it is straightforward to write a fast algorithm for checking ground joinability modulo AC, but less so for checking ground joinability modulo an arbitrary set of equations. We propose a novel method, incremental ground joinability, which efficiently searches for "proofs" of ground joinability of two terms, with the objective of detecting this with as few case splits as possible, while — crucially — being minimally impactful in the (vastly more common) case where the terms are not ground joinable.

## 1.2 Summary of contributions

The main contributions of this work are as follows:

- A novel criterion for redundancy of clauses and inferences called closure redundancy, which improves on the standard notion of redundancy used in most first-order saturation theorem provers.
- A proof of refutational completeness of the superposition calculus wrt. to this notion of closure redundancy.
- An enhanced variant of the widely-used demodulation simplification rule, named encompassment demodulation, which is applicable in more cases and also faster to check.

- Proof that superposition + encompassment demodulation is equivalent, on unit clauses, to unfailing completion.

- Proof that ground joinability criteria and critical pair criteria, long used in the context of unit-equational completion, can be applied under certain conditions in the superposition calculus, and thus in non-unit-equality problems.

- Specialised criteria for associative-commutative (AC) symbols, including AC normalisation and demodulation modulo AC.

- An efficient new algorithm for ground joinability checks, which is crucial for practical applicability of these criteria.

- A powerful architecture for scheduling simplifications in a given-clause loop in a flexible way, which enables smart application of all the aforementioned criteria.

- An implementation of all of the above in a state-of-the-art theorem prover, iProver, and an evaluation of the performance of this implementation.

## 1.3 Published work

Portions of this dissertation have been published in various peer-reviewed venues. Here we summarise the publications which include significant portions of this work, in chronological order.

In Duarte and Korovin [2020], we present (among other things) a description of the iProver superposition simplification setup, the flexible given-clause loop that underpins the superposition calculus in iProver (Chapter 5 section 5.2).

In Duarte and Korovin [2021], we introduce the notion of closure redundancy and give a definition of closure orderings parametrised by a simplification ordering on terms; we then give a proof of refutational completeness of superposition up to closure redundancy. Then, using this, we show several novel criteria related to AC reasoning, such as normalisation of AC terms (in a way not subsumed by demodulation via AC axioms) and deletion and simplification of AC-joinable literals. We also give a novel variant of the demodulation simplification rule, called encompassment demodulation (Chapter 3, Chapter 4 sections 4.1 and 4.3).

In Duarte and Korovin [2022], we further refine the closure ordering (the version presented in this work) to allow even greater improvements in the applicability of rewrite-type simplifications, namely we show that general ground joinability as well as connectedness (a type of critical pair criterion), hitherto restricted only to the context of equational completion (in unit-equational problems), can in fact be used in superposition under some limited constraints (which precisely match those of equational completion when restricting to positive unit clauses). We further improve the constraints on encompassment demodulation. Finally, we also discuss an algorithm for checking ground joinability of literals in practice, a crucial step for practical applicability of this criterion, and discuss experimental results after implementation in iProver (Chapter 3, Chapter 4 sections 4.1 and 4.2, Chapter 5 section 5.1).

*Blank page*

*Blank page*

# Chapter 2

# Preliminaries

In all matters, before beginning, a diligent preparation should be made.

Cicero, *De Officiis* (44 BCE)

In this chapter we introduce the fundamental concepts which are used throughout this work. We also define notation and terminology.

## 2.1 Basics

We carefully distinguish between "meta"-level notions of logical connections and relations over mathematical objects on one hand, and functions, predicates, or logical connectives inside a given logic on the other, whenever this is not obvious from context. For instance, mathematical equality between objects at a meta level is denoted with $=$, while $\approx$ always denotes the equality predicate inside a given logic. We also assume familiarity with the usual notions of set and multiset.

A binary relation over a set $S$ is a subset of $R \times R$. We often write in infix notation, e.g. $a \succ b$ instead of $\langle a, b \rangle \in \succ$. A relation $R$ is **total** over $X$ if $\forall x, y \in X.\ xRy \lor yRx \lor x = y$. A relation $R$ is **well-founded** (or **terminating**) if there are no infinitely-descending chains $x_1 R x_2 \land x_2 R x_3 \land \cdots$. A relation $R$ is **transitive** if $\forall xyz.\ xRy \land yRx \ \Rightarrow\ xRz$. A relation $R$ is **reflexive** if $\forall x.\ xRx$ and **irreflexive** if $\forall x.\ \neg(xRx)$.

The **transitive closure** of a relation $R$ is the smallest transitive relation that contains $R$, and is denoted by $R^+$. A **transitive reduction** of a relation

$R$ is a minimal relation whose transitive closure is $R^+$, and is denoted by $R^-$. The **reflexive-transitive** closure of a relation $R$ is the smallest reflexive and transitive relation that contains $R$, and is denoted by $R^*$.

A (strict) **partial order** $\succ$ is a binary relation which is transitive and irreflexive. For a given strict order $\succ$, the induced non-strict order $\succeq$ is $\succ \cup =$.

A (non-strict) partial **preorder** (or quasiorder) $\succeq$ is any binary relation which is transitive and reflexive. Whenever a non-strict preorder $\succeq$ is given, the induced equivalence relation $\sim$ is $\succeq \cap \preceq$, and the induced strict order $\succ$ is $\succeq \setminus \sim$.

If $x \not\succeq y$ and $x \not\preceq y$, then the two elements are said to be **incomparable** in that partial order/preorder; this is notated $x \bowtie y$.

For an ordering $\succ$ over a set $X$, its **multiset extension** $\ggcurly$ over multisets of $X$ is given by: $A \ggcurly B$ iff $A \neq B$ and $\forall x \in B.\ B(x) > A(x)\ \exists y \in A.\ y \succ x \wedge A(y) > B(y)$, where $A(x)$ is the number of occurrences of element $x$ in multiset $A$. We also use $\ggg\!\succ$ for the the multiset extension of $\ggcurly$. It is well known that the mutltiset extension of a well-founded/total order is also a well-founded/total order, respectively [Dershowitz and Manna 1979].

The **lexicographic extension** of $\succ$ over $X$ is denoted $\succ_{\text{lex}}$ over ordered tuples of $X$, and is given by $\langle x_1, \ldots, x_n \rangle \succ_{\text{lex}} \langle y_1, \ldots, y_n \rangle$ iff $\exists i.\ x_1 = y_1 \wedge \cdots \wedge x_{i-1} = y_{i-1} \wedge x_i \succ y_i$. The lexicographic extension of a well-founded/total order is also a well-founded/total order, respectively.

## 2.2 First-order logic

First-order logic — also known as predicate logic — is a formal system used in mathematics, computer science, and philosophy. First-order logic is of basic importance in mathematics (being, for instance, the language of ZFC, the foremost foundation of mathematics [Suppes 1972]), as well as being a convenient language to formalise many other mathematical theories and many problems in other domains.

Crucially, while being expressive (it is Turing-complete, and therefore undecidable), it still has nice metalogical properties, including completeness (and therefore semi-decidability of the entailment problem for countable axiom sets), and compactness.

It is a well-known fact that any formula in first-order logic can be re-written into an equisatisfiable formula which is a conjunction of universally quantified disjunctions of atoms and negated atoms, a so-called **conjunctive normal form** (CNF), or clausal form [Nonnengart and Weidenbach 2001]. We will make this definition clear in the next section. The superposition calculus that we treat in this work, which semi-decides the satisfiability problem,[1] is defined over formulas in CNF. Because of this, and since we will not make any considerations about the translation of an arbitrary first-order logic problem into CNF,[2] we will restrict ourselves only to a treatment of formulas in clausal form in this work.

Furthermore, we will also assume — without loss of generality — that all atoms are equations, and that the equality predicate is interpreted, as part of the logic (therefore, there are no other predicate symbols). This is not a problem because the semantics of translating non-equational atoms $P(t_1, \ldots, t_n)$ into $p(t_1, \ldots, t_n) \approx \top$, for a distinguished constant $\top$ and a fresh symbol $p$, preserves (un)satisfiability [Nieuwenhuis and Rubio 2001, p. 390].

Finally, we will also assume single-sorted logic for clarity of exposition, but all the results in this work generalise cleanly to the many-sorted case.

## Syntax

A first-order (unsorted) **signature** consists of a set $\Sigma$ of function symbols. Each symbol has a fixed non-negative arity, written $\operatorname{arity}(f)$ for any symbol $f \in \Sigma$. A symbol of 0-arity is also called a **constant**. We also assume that there exists a countably infinite set of **variables**, $\mathcal{X}$.

The set $\mathcal{T}(\Sigma, \mathcal{X})$ of first-order **terms** in a given signature is defined inductively as follows.

$$f(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma, \mathcal{X}) \qquad \text{if } \begin{cases} f \in \Sigma \\ \operatorname{arity}(f) = n \\ t_1, \ldots, t_n \in \mathcal{T}(\Sigma, \mathcal{X}) \end{cases} \tag{2.1a}$$

$$x \in \mathcal{T}(\Sigma, \mathcal{X}) \qquad \text{if } x \in \mathcal{X} \tag{2.1b}$$

We can see that this set is well-defined, and always non-empty (since $\mathcal{X}$ is non-empty).

[1] Or, equivalently, the validity/entailment problem.

[2] The problem is extensively discussed in the literature, e.g. Nonnengart and Weidenbach [2001] and Hoder et al. [2012].

The set defined by $\mathcal{T}(\Sigma, \emptyset)$ is the set of **ground terms**, i.e. terms without variables. This set is non-empty iff there exists at least one constant symbol.[3] Therefore, we will always assume that the signature contains at least one constant [Nieuwenhuis and Rubio 2001].

**Atoms** are unordered pairs of terms, written $s \approx t$ (or equivalently $t \approx s$) for two terms $s$ and $t$. **Literals** are atoms together with a boolean polarity, written $s \approx t$ for a positive literal and $s \not\approx t$ for a negative literal. We write $s \mathrel{\dot{\approx}} t$ to mean a literal which can be positive or negative.

A **clause** is a multiset of literals, written either with normal set notation ($\{s_1 \mathrel{\dot{\approx}} t_1, \dots\}$) or as a disjunction of literals ($s_1 \mathrel{\dot{\approx}} t_1 \vee \cdots$), alluding to its semantics. Collectively, terms, literals, and clauses will be called **expressions**.

A **substitution** is a mapping from variables to terms which is the identity for all but finitely many variables. If $e$ is an expression, we denote the application of a substitution $\sigma$ by $e\sigma$, which is obtained by replacing all variables with their image in $\sigma$, i.e.

$$C\sigma = \{(s \mathrel{\dot{\approx}} t)\sigma \mid s \mathrel{\dot{\approx}} t \in C\}\,, \tag{2.2a}$$

$$(s \mathrel{\dot{\approx}} t)\sigma = s\sigma \mathrel{\dot{\approx}} t\sigma\,, \tag{2.2b}$$

$$f(t_1, \dots, t_n)\,\sigma = f(t_1\sigma, \dots, t_n\sigma)\,, \tag{2.2c}$$

and $x\sigma$ is simply equal to the image of $x$ in $\sigma$.

An injective substitution onto variables is called a **renaming**. The identity substitution is denoted by $id$. The composition of two substitutions $\sigma$ and $\rho$ is denoted by juxtaposition and read left-to-right, that is, $\theta = \sigma\rho \;\Rightarrow\; s\theta = (s\sigma)\rho$.

We write $s[t]$ if $t$ is a **subterm** of $s$, meaning

$$s = t \text{ or } s_1[t] \text{ or } \cdots \text{ or } s_n[t], \quad \text{where } s = f(s_1, \dots, s_n). \tag{2.3}$$

If $s[t]$ but $s \neq t$, then it is a **strict subterm**. We denote these relations by $s \trianglerighteq t$ and $s \triangleright t$ respectively. We write $s[t \mapsto t']$ to denote the term obtained from $s$ by replacing *all* occurrences of $t$ by $t'$.

We also overload the notation in the previous paragraph for literals and clauses, for example $C[u]$ or $u \trianglelefteq C$ denotes $\exists s \mathrel{\dot{\approx}} t \in C.\ (s[u]$ or $t[u])$.

A substitution $\theta$ is **more general** than $\sigma$ if $\theta\rho = \sigma$ for some substitution $\rho$ which is not a renaming. If $s$ and $t$ can be **unified**, that is, if there exists $\sigma$ such that $s\sigma = t\sigma$, then there also exists the **most general unifier**, written $\mathrm{mgu}(s, t)$.

[3]But if it exists, it is infinite iff any non-constant symbol exists.

A term $s$ is said to be **more general** than $t$ if there exists a substitution $\theta$ that makes $s\theta = t$ but there is no substitution $\sigma$ such that $t\sigma = s$. Two terms $s$ and $t$ are said to be **equal modulo renaming** if there exist injective $\theta, \sigma$ such that $s\theta = t$ and $t\sigma = s$. The relations "less general than", "equal modulo renaming", and their union are represented respectively by the symbols $\sqsupset$, $\equiv$, and $\sqsupseteq$.

An inference rule is written

$$\text{Name} \qquad \frac{C_1 \ \cdots \ C_n}{D_1 \ \cdots \ D_m} \tag{2.4}$$

or $C_1, \ldots, C_n \vdash D_1, \ldots, D_m$, where $C_1, \ldots, C_n$ are the **premises** and $D_1, \ldots, D_m$ are the **conclusions**. Inferences may have side-conditions, which are denoted to the right of the inference.

## Semantics

In this work, the semantics of clauses with equality are defined in terms of term rewrite systems. We will first introduce some notions.

A binary relation $\to$ over a set of terms is a **rewrite relation** if (i) $l \to r \ \Rightarrow \ l\sigma \to r\sigma$ and (ii) $l \to r \ \Rightarrow \ s[l] \to s[l \mapsto r]$. A **term rewrite system** $R$, composed of a set of **rewrite rules** $l_i \to r_i$, *induces* the following rewrite relation, denoted $\to_R$:

$$\{s[l\sigma] \to_R s[r\sigma] \mid l \to r \in R, s \in \mathcal{T}(\Sigma, \mathcal{X}), l\sigma \in \mathcal{T}(\Sigma, \mathcal{X})\} \tag{2.5}$$

If $\to_R$ is the relation induced by the term rewrite system $R$, we say that:

- A positive literal $s \approx t$ is true in $R$ if $s \overset{*}{\leftrightarrow}_R t$, while a negative literal $s \not\approx t$ is true in $R$ if $s \overset{*}{\not\leftrightarrow}_R t$.
- A clause $C$ is true in $R$ if at least one literal $s \dot{\approx} t \in C$ is true.
- A set of clauses $S$ is true in $R$ if all clauses $C \in S$ are true.

A set of clauses is said to be **satisfiable** iff it has a **model**, i.e. there exists a term rewrite system which makes it true.

The relation of **entailment** is as expected: $S_1 \models S_2$, for two sets of clauses $S_1, S_2$, iff all term rewrite systems which are a model of all the clauses in $S_1$ are also a model of all the clauses in $S_2$.

Furthermore, two terms are **joinable** in a rewrite relation (written $s \downarrow t$) if $s \overset{*}{\to} u \overset{*}{\leftarrow} t$. An important feature of some rewrite systems is the **Church-Rosser property**, which is defined as $s \overset{*}{\leftrightarrow} t \ \Rightarrow \ s \downarrow r$. This property is equivalent to that of **confluence**, which is defined as $s \overset{*}{\leftarrow} u \overset{*}{\to} t \ \Rightarrow \ s \downarrow t$.

If a rewrite relation is also a strict ordering, then it is a **rewrite ordering**. A **reduction ordering** is a rewrite ordering which is well-founded. In this work we consider reduction orderings which are total on ground terms, such orderings are also **simplification orderings** i.e., satisfy $s \rhd t \Rightarrow s \succ t$.

Given an equation $l \approx r$ and a rewrite ordering $\succ$, we say that there is a rewrite $s \to t$ **via** $l \approx r$ if $s \to_R t$ where $\to_R$ is the rewrite system induced by $R = \{l\sigma \to r\sigma \mid l\sigma \succ r\sigma\} \cup \{r\sigma \to l\sigma \mid r\sigma \succ l\sigma\}$, i.e. the set of orientable instances of $l \approx r$. In this sense, a set of equations $E$ also *induces* the rewrite system $\{l\sigma \to r\sigma \mid l \approx r \in E, l\sigma \succ r\sigma\}$, which in turn induces a rewrite relation as per (2.5).

## 2.3 Equational completion

Equational completion was originally introduced by Knuth and Bendix [1970], as a procedure for computing a confluent and well-founded term rewrite system which is equivalent to a given set of equations. Here, we are only interested in its use for theorem proving. Specifically, we present the unfailing completion procedure as defined in Bachmair, Dershowitz and Plaisted [1989]. With an implicit orientation step, the sole inference rule of equational completion necessary for refutational completeness is, for a given rewrite ordering $\succ$,

$$\text{Critical Pair} \qquad \frac{l \approx r \quad s[u] \approx t}{s\theta[l\theta \mapsto r\theta] \approx t\theta}, \qquad \begin{array}{l}\text{where } \theta = \mathrm{mgu}(l, u),\\ l\theta \not\preceq r\theta \text{ and } s\theta \not\preceq t\theta,\\ \text{and } u \text{ not a variable,}^{4}\end{array} \tag{2.6}$$

with the property that if a goal $s \approx t$ follows from a set of equations $E$, exhaustively applying this rule on $E \cup \{eq(x, x) \approx \top,\ eq(s, t) \approx \bot\}$ will eventually produce $\top \approx \bot$ [Bachmair, Dershowitz and Plaisted 1989].

## 2.4 Superposition calculus

Merging the ideas of resolution, on the one hand, and of unfailing completion via a reduction ordering on terms, on the other (as outlined in the introduction), the superposition calculus is an inference system for first-order logic with equality, working on clausified formulas.

[4] Equivalent to the Orientation and $\text{Deduction}_2$ rules in Bachmair, Dershowitz and Plaisted [1989].

**Definition 2.1.** The superposition calculus for a simplification order $\succ$ on terms comprises the following inference rules.

$$\text{Superposition}\quad \frac{\underline{l \approx r} \vee C \quad \underline{s[u] \mathrel{\dot\approx} t} \vee D}{(s[u \mapsto r] \mathrel{\dot\approx} t \vee C \vee D)\theta}, \quad \begin{array}{l}\text{where } \theta = \mathrm{mgu}(l, u),\\ l\theta \not\preceq r\theta,\ s\theta \not\preceq t\theta,\\ \text{and } u \text{ is not a variable,}\end{array} \tag{2.7a}$$

$$\text{Eq. Resolution}\quad \frac{\underline{s \not\approx t} \vee C}{C\theta}, \quad \text{where } \theta = \mathrm{mgu}(s, t), \tag{2.7b}$$

$$\text{Eq. Factoring}\quad \frac{\underline{s \approx t} \vee \underline{s' \approx t'} \vee C}{(s \approx t \vee t \not\approx t' \vee C)\theta}, \quad \begin{array}{l}\text{where } \theta = \mathrm{mgu}(s, s'),\\ s\theta \not\preceq t\theta \text{ and } t\theta \not\preceq t'\theta,\end{array} \tag{2.7c}$$

and where the selection function (underlined) selects either at least one negative or else all maximal (wrt. $\succ$) literals in the clause.

## Properties

The key property of the superposition calculus is that it is a **refutationally complete** calculus for first-order logic with equality. This means that a set of clauses which is saturated wrt. these inference rules (i.e. there are no more valid inferences with premises in the set and conclusions not in the set), and which does not contain the empty clause, must be satisfiable. The inference rules are also **sound**. Completeness and soundness imply that fair[5] application of these rules is a semi-decision procedure for the validity problem in first-order logic with equality.

This notion can be extended to **refutational completeness up to redundancy**, where, given a suitable definition of **redundant clause** and **redundant inference**, the property becomes: that a set of clauses which is saturated up to redundancy — meaning that there are no more non-redundant inferences which have non-redundant premises in the set and a non-redundant conclusion not in the set — must be satisfiable.

Making this precise, for a definition of "clause $C$ redundant in set $S$" and "inference $C_1, \ldots, C_n \vdash D$ redundant in set $S$", the definition of "$S$ saturated up to redundancy" is:

[5] Informally, that all inferences will eventually be performed; this notion can be made precise [Bachmair and Ganzinger 2001], but here we just consider the "static" view of saturation.

**Definition 2.2** (Saturation up to redundancy)**.** A set of clauses $S$ is **saturated up to redundancy** if any inference $C_1, \ldots, C_n \vdash D$ which has $C_1, \ldots, C_n \in S$ and $D \notin S$ has either (i) one of $C_1, \ldots, C_n, D$ redundant in $S$ or (ii) $C_1, \ldots, C_n \vdash D$ redundant in $S$.

**Definition 2.3** (Refutational completeness)**.** A set of inference rules is refutationally complete iff any set of clauses which is saturated up to redundancy wrt. those inference rules is satisfiable or contains the empty clause.

The notion of saturation, and therefore of refutational completeness, is parametrised by a concrete notion of "redundant clause" and "redundant inference". In Section 3.2, we will introduce a novel notion of redundancy, and show that the superposition inference system as defined above is refutationally complete up to that notion of redundancy.

# Chapter 3

# Closure redundancy in superposition

If we have no idea why a statement is true, we can still prove it by induction.

Gian-Carlo Rota (1956)

The main result of this chapter is a novel and powerful notion of redundancy of clauses and inferences, together with a proof that the superposition calculus is refutationally complete wrt. this notion.

The work presented here was first peer-reviewed and published in Duarte and Korovin 2021 and later refined in Duarte and Korovin 2022.

## 3.1 Closures and closure orderings

As previously discussed, the notion of refutational completeness up to redundancy necessitates a specific definition of redundant clause and redundant inference. In turn, our novel notion of redundancy depends on the definitions presented in this section.

**Definition 3.1.** A **closure** is a pair of an expression and a substitution, denoted by $e \cdot \theta$.

We say that a closure $e \cdot \theta$ *represents* the expression $e\theta$. In this work, we mainly deal with **ground closures**, i.e. closures where $e\theta$ is ground.

Closures give us a more refined notion of an instance of an expression: rather than representing an instance of $e$ by an expression $e\theta$, for some substitution $\theta$, we instead represent it by a closure $e \cdot \theta$. Retaining the information about the original expression as well as the instance will be crucial to formulate some of the notions that follow.

*Example* 3.1. Consider for instance two terms $f(x, y)$ and $f(a, y)$, and consider the substitution $\theta = \{x \mapsto a, y \mapsto b\}$, which is a grounding substitution for both. The instances $f(x, y)\,\theta$ and $f(a, y)\,\theta$, as terms, are both equal to $f(a, b)$, but by representing them by a closure, the instances $f(x, y) \cdot \theta$ and $f(a, y) \cdot \theta$ can be distinguished.

Indeed, because $f(x, y)$ is more general than $f(a, y)$, it is useful to use this information, for example, in inductive proofs, via an ordering that makes instances from more general terms be smaller than the same instance from less general ones (e.g. $f(x, y) \cdot \theta$ smaller than $f(a, y) \cdot \theta$). This is the intuitive motivation for the ordering that we define in the next section.

In terms of semantics, and by abuse of notation, a literal (resp. clause) closure is said to be true in a term rewrite system $R$ iff the literal (resp. clause) it represents is true in $R$.

Finally, let us define the following.

**Definition 3.2.** Let $e$ be an expression. Then

- $\mathrm{GSubs}(e) = \{\theta \mid e\theta \text{ is ground}\}$.
- $\mathrm{GInsts}(e) = \{e\theta \mid e\theta \text{ is ground}\}$.
- $\mathrm{GClos}(e) = \{e \cdot \theta \mid e\theta \text{ is ground}\}$.

The idea of closure was first introduced by Bachmair, Ganzinger et al. [1995].

## Closure orderings

As illustrated in the example above, the main intuition is that we wish to define an ordering on closures such that terms which are *more general* tend to be *smaller*. More precisely, we start from a “base” simplification ordering on terms, and first compare — with this ordering — the terms represented by the closures. If they are equal, ties are broken by taking closures from more general terms to be smaller than closures from less general ones. This is the main idea behind our definition of closure orderings.

**Definition 3.3.** Let $\succ_t$ be a simplification ordering which is total on ground terms. The usual way to extend this to an ordering on literals and clauses is as follows. Define the multisets

$$M_l(s \approx t) = \{s, t\}\,, \tag{3.1a}$$

$$M_l(s \not\approx t) = \{s, t, s, t\}\,, \tag{3.1b}$$

then

$$p \succ_l q \qquad \text{iff} \qquad M_l(p) \succ\!\!\succ_t M_l(q)\,, \tag{3.2}$$

and

$$C \succ_c D \qquad \text{iff} \qquad C \succ\!\!\succ_l D\,. \tag{3.3}$$

That is, intuitively: the ordering on literals $\succ_l$ compares the maximal side of the (dis)equality first, then places negative equalities over positive, then compares the remaining sides; the ordering on clauses $\succ_c$ compares them as multisets.

**Definition 3.4** (Closure orderings)**.** We will now extend this to an ordering on closures as well. First, we define the order $\succ_{tc}$ on ground term closures as follows. Let

$$s \cdot \sigma \succ_{tc'} t \cdot \rho \qquad \text{iff} \qquad \begin{array}{l} \text{either } s\sigma \succ_t t\rho \\ \text{or else } s\sigma = t\rho \text{ and } s \sqsupset t\,, \end{array} \tag{3.4}$$

where $s\sigma$ and $t\rho$ are ground, and let $\succ_{tc}$ be an (arbitrary) total well-founded extension of $\succ_{tc'}$ over ground closures.[1]

Then, we define an ordering $\succ_{cc}$ on ground clause closures. Let $M_{cc}$ be defined as follows, depending on whether the clause is a positive unit, a negative unit, or non-unit clause:

$$M_{cc}(\emptyset \cdot \theta) = \emptyset\,, \tag{3.5a}$$

$$M_{cc}((s \approx t) \cdot \theta) = \{\{s \cdot \theta\}, \{t \cdot \theta\}\}\,, \tag{3.5b}$$

$$M_{cc}((s \not\approx t) \cdot \theta) = \{\{s \cdot \theta, t \cdot \theta, s\theta \cdot id, t\theta \cdot id\}\}\,, \tag{3.5c}$$

$$M_{cc}((s \doteq t \vee \cdots) \cdot \theta) = \{M_{lc}(L \cdot \theta) \mid L \in \{s \doteq t, \dots\}\}\,, \tag{3.5d}$$

where (recall that $\dot\approx$ stands for $\approx$ or $\not\approx$)

$$M_{lc}((s \approx t) \cdot \theta) = \{s\theta \cdot id, t\theta \cdot id\}\,, \tag{3.5e}$$

$$M_{lc}((s \not\approx t) \cdot \theta) = \{s\theta \cdot id, t\theta \cdot id, s\theta \cdot id, t\theta \cdot id\}\,, \tag{3.5f}$$

then $\succ_{cc}$ is defined by

$$C \cdot \sigma \succ_{cc} D \cdot \rho \qquad \text{iff} \qquad M_{cc}(C \cdot \sigma) \succ\!\!\succ\!\!\succ_{tc} M_{cc}(D \cdot \rho)\,. \tag{3.6}$$

[1] This "two-step" definition is necessary because (3.4), as defined, is not total; see Lemma 3.1.

There are two parts to this definition. First, (3.4) implements the intuition of breaking ties on the base $\succ_t$ order by making $\succ_{tc}$ place instances of more general terms lower than instances of less general ones. Then, (3.5–3.6) extends the order $\succ_{tc}$ on term closures to an order $\succ_{cc}$ on clause closures; in short: (3.5b) handles positive unit clauses, (3.5c) handles negative unit clauses, and (3.5d–f) handle non-unit clauses.

The main purpose of this definition is twofold:

(i) That when $s\theta \succ_t t\theta$ and $u$ occurs in a clause $D$, then either $s\theta \lhd u$ or $s \sqsubset s\theta = u$ imply $(s \approx t) \cdot \theta\rho \prec_{cc} D \cdot \rho$,[2] and

(ii) That when $C$ is a positive unit clause, $D$ is not, $s$ is the maximal subterm in $C\theta$ and $t$ is the maximal subterm in $D\sigma$, $s \preceq_t t$ implies $C \cdot \theta \prec_{cc} D \cdot \sigma$.[3]

Critically, these two properties will enable unconditional rewrites via oriented unit equations on positive unit clauses to succeed whenever they would also succeed in unfailing completion [Bachmair, Dershowitz and Plaisted 1989], and rewrites on negative unit and non-unit clauses to always succeed. This enables a plethora of rewrite-based simplification rules to be formulated, in such a way that their application does *not* compromise refutational completeness. These will be presented in detail, with all necessary definitions rigorously explained, in Chapter 4. For now, let us continue by visiting some examples of these orderings in action, and then by showing several important properties of the $\succ_{tc}$ and $\succ_{cc}$ orderings.

## Examples

Let $f \succ_t a \succ_t b \succ_t c$. Then

$$f(a,b) \bowtie_t f(x,b) \tag{3.7a}$$

that is, $f(a,b)$ and $f(x,b)$ are incomparable wrt. (any) $\succ_t$. However, even though the substitution $(x \mapsto a)$ makes them both equal, the corresponding closures can be ordered with $\succ_{tc}$:

$$f(a,b)\, id =_t f(x,b)\, (x \mapsto a) \tag{3.7b}$$

$$f(a,b) \cdot id \succ_{tc} f(x,b) \cdot (x \mapsto a) \tag{3.7c}$$

[2]Lemma 3.8.

[3]Lemma 3.7.

Note that some closures cannot be ordered by $\succ_{tc'}$

$$f(a,y)\cdot(y\mapsto b)\bowtie_{tc'} f(x,b)\cdot(x\mapsto a) \tag{3.7d}$$

but they are still (arbitrarily) ordered by $\succ_{tc}$.

Note, of course, that $\succ_{cc}$ is *not* an extension of $\succ_c$, therefore some ground closures are ordered *opposite* to what the ground clauses they represent would be ordered in $\succ_c$. For example,

$$f(a)\approx b\succ_c f(a)\approx c \tag{3.7e}$$
$$(f(x)\approx b)\cdot(x\mapsto a)\prec_{cc}(f(a)\approx c)\cdot id \tag{3.7f}$$

and

$$f(a)\approx b\prec_c f(a)\not\approx c \tag{3.7g}$$
$$(f(a)\approx b)\cdot id\prec_{cc}(f(a)\not\approx c)\cdot id \tag{3.7h}$$

and

$$f(a)\approx b\succ_c f(a)\approx c\lor f(b)\approx c \tag{3.7i}$$
$$(f(a)\approx b)\cdot id\prec_{cc}(f(a)\approx c\lor f(b)\approx c)\cdot id \tag{3.7j}$$

Example (3.7f) is illustrative of the difference between $\succ_c$ and $\succ_{cc}$; it holds because $(f(x)\approx b)\cdot(x\mapsto a)$ and $(f(a)\approx c)\cdot id$ are represented, in terms of Definition 3.4, by the multisets $\{\{f(x)\cdot(x\mapsto a)\},\{b\cdot(x\mapsto a)\}\}$ and $\{\{f(a)\cdot id\},\{c\cdot id\}\}$ respectively. But $f(x)\cdot(x\mapsto a)\prec_{tc} f(a)\cdot id$ (because $f(x)\,(x\mapsto a)=f(a)\,id=f(a)$, but $f(x)$ is more general than $f(a)$), and $f(x)\cdot(x\mapsto a)\prec_{tc} c\cdot id$. So the former multiset is smaller than the latter, wrt. $\ggg_{tc}$, meaning the former clause closure is smaller than the latter, wrt. $\succ_{cc}$.

## Properties

The following lemmas about the orderings will be needed throughout this work.

**Lemma 3.1.** $\succ_{tc}$ and $\succ_{cc}$ are well-founded and total on ground term closures and ground clause closures, respectively.

*Proof.* $\succ_{tc'}$ is a well-founded ordering, since $\succ_t$ and $\sqsupset$ are well-founded and the lexicographic combination of two well-founded orderings is well-founded

[Baader and Nipkow 1998]. However, it is only a partial order even on ground closures, e.g. (3.7d). Nonetheless, it is well-known that any partial well-founded order can be extended to a total well-founded order (see e.g. Bonnet and Pouzet [1982]; this follows from Zorn's lemma). Therefore we defined $\succ_{tc}$ as an *arbitrary* total well-founded extension of $\succ_{tc'}$.

Then, since $\succ_{cc}$ is a (twice) multiset extension of $\succ_{tc}$, and since the multiset extension of a total well-founded order is also a total well-founded order [Dershowitz and Manna 1979], we have that $\succ_{cc}$ is also well-founded and total on ground clause closures. □

**Lemma 3.2.** Assume $s, t$ are ground, then $s \cdot id \succ_{tc} t \cdot id \;\Leftrightarrow\; s \succ_t t$.

*Proof.*

($\Rightarrow$) Definition 3.4.

($\Leftarrow$) Definition 3.4 and the fact that $s\,id = t\,id \;\Rightarrow\; s \not\sqsupset t$. □

**Lemma 3.3.**

- $s\sigma \succ_t t\rho \;\Rightarrow\; s \cdot \sigma \succ_{tc} t \cdot \rho$
- $s\sigma =_t t\rho \;\Rightarrow\; s \cdot \sigma \succeq_{tc} t \cdot \rho$
- $s\sigma \succeq_t t\rho \;\Rightarrow\; s \cdot \sigma \succeq_{tc} t \cdot \rho$

*Proof.* Definition 3.4. □

**Lemma 3.4.** $t\rho \cdot \sigma \succeq_{tc} t \cdot \rho\sigma$. Analogously for $\succ_{cc}$. In particular, $t\sigma \cdot id \succeq_{tc} t \cdot \sigma$, and analogously for $\succ_{cc}$.

*Proof.* From Definition 3.4 and the fact that $t\rho \sqsupseteq t$. □

**Lemma 3.5.** $t \cdot \sigma \succ_{tc} s \cdot id \;\Leftrightarrow\; t\sigma \succ_t s$.[4]

*Proof.*

($\Rightarrow$) For $t \cdot \sigma \succ_{tc} s \cdot id$ to hold, either $t\sigma \succ_t s$, or else $t\sigma = s$ but then $t \sqsupset s$ cannot hold.

($\Leftarrow$) Follows from the definition. □

**Lemma 3.6.** $\succ_{tc}$ has the following property: $l \succ_t r \;\Rightarrow\; s[l] \cdot \theta \succ_{tc} s[l \mapsto r] \cdot \theta$. Analogously for $\succ_{cc}$: $l \succ_t r \;\Rightarrow\; C[l] \cdot \theta \succ_{cc} C[l \mapsto r] \cdot \theta$.

[4]But not, in general, $s \cdot id \succ_{tc} t \cdot \sigma \;\Leftrightarrow\; s \succ_t t\sigma$, e.g. $f(a) \cdot id \succ_{tc} f(x) \cdot (x \mapsto a)$.

*Proof.* For $\succ_{tc}$: let $l \succ_t r$. By the fact that $\succ_t$ is a rewrite relation, we have $l \succ_t r \Rightarrow s[l] \succ_t s[l \mapsto r] \Rightarrow s[l]\theta \succ_t s[l \mapsto r]\theta$. Then, by the definition of $\succ_{tc}$, $s[l] \cdot \theta \succ_{tc} s[l \mapsto r] \cdot \theta$.

For $\succ_{cc}$: if $C[l]$ then $l$ occurs in at least some term $s$ in $C$, so for that term we have $s[l] \cdot \theta \succ_{tc} s[l \mapsto r] \cdot \theta$. Since $C[l]$ and $C[l \mapsto r]$ have the same number of literals, and since if $C[l]$ is a positive (resp. negative) unit clause then $C[l \mapsto r]$ is also a positive (resp. negative) unit clause, then

$$M_{cc}(C[l] \cdot \theta) \ggg_{tc} M_{cc}(C[l \mapsto r] \cdot \theta) \tag{3.8}$$

by (3.5), and therefore $C[l] \cdot \theta \succ_{cc} C[l \mapsto r] \cdot \theta$ by (3.6). □

**Definition 3.5.** $\max_t(C)$ is the maximal element of $\{s \mid C[s]\}$ wrt. $\succ_t$, where $C$ is a ground clause.

**Lemma 3.7.** If $\max_t(C\sigma) \succ_t \max_t(D\rho)$, then $C \cdot \sigma \succ_{cc} D \cdot \rho$. If $\max_t(C\sigma) = \max_t(D\rho)$, then if $D\rho$ is a positive unit and $C\sigma$ is not, $C \cdot \sigma \succ_{cc} D \cdot \rho$.

*Proof.* Let $s\sigma = \max_t(C\sigma)$ and $t\rho = \max_t(D\rho)$. Assume $s\sigma \succ_t t\rho$. Then, for all $u \cdot \rho \in M \in M_{cc}(D \cdot \rho)$,

$$\begin{aligned} u \cdot \rho &\preceq_{tc} t\rho \cdot id \\ &\prec_{tc} s \cdot \sigma \\ &\preceq_{tc} s\sigma \cdot id\,, \end{aligned} \tag{3.9}$$

(using Lemmas 3.4 and 3.5) therefore, for any such $M$, we have $M \lll_{tc} N$ where $N \in M_{cc}(C \cdot \sigma)$ is a multiset which contains $s \cdot \sigma$ or $s\sigma \cdot id$. Hence, $M_{cc}(D \cdot \rho) \llll_{tc} M_{cc}(C \cdot \sigma)$, and $D \cdot \rho \prec_{cc} C \cdot \sigma$.

Now assume $s\sigma = t\rho$, $D\rho$ is a positive unit (wlog. $(t \approx v) \cdot \rho$), and $C\sigma$ is not. We have $M_{cc}(D \cdot \rho) = \{\{t \cdot \rho\}, \{v \cdot \rho\}\}$. Since $C\sigma$ is either a non-unit or a negative unit clause, there exists an $N \in M_{cc}(C\sigma)$ such that $s\sigma \cdot id \in N$ and $N$ has at least 2 elements (3.5c–f). Therefore

$$\begin{aligned} & v\rho \preceq_t t\rho =_t s\sigma \\ \Rightarrow\ & v \cdot \rho \preceq_{tc} t \cdot \rho \preceq_{tc} s \cdot \sigma \preceq_{tc} s\sigma \cdot id \\ \Rightarrow\ & \{t \cdot \rho\} \lll_{tc} N \text{ and } \{v \cdot \rho\} \lll_{tc} N \\ \Rightarrow\ & M_{cc}(D \cdot \rho) \llll_{tc} M_{cc}(C \cdot \sigma)\,, \end{aligned} \tag{3.10}$$

using Lemmas 3.3 and 3.4, and $D \cdot \rho \prec_{cc} C \cdot \sigma$. □

**Lemma 3.8.** Let $D[u]$ be a clause and $s\theta, t\theta$ be two terms such that $u \rhd s\theta$ or $u = s\theta \sqsupset s$. If $s\theta \succ_t t\theta$, then $(s \approx t) \cdot \theta\rho \prec_{cc} D \cdot \rho$, for any $\rho \in \mathrm{GSubs}(s \approx t, D)$.

*Proof.* Since $s\theta \succ_t t\theta$, then $\max_t(s\theta\rho \approx t\theta\rho) = s\theta\rho$ for any $\rho \in \mathrm{GSubs}(s \approx t, D)$. Therefore, if $s\theta \lhd u$ and $D[u]$, then $s\theta \prec_t u \Rightarrow s\theta\rho \prec_t u\rho \preceq_t \max_t(D\rho)$, implying $(s \approx t) \cdot \theta\rho \prec_{cc} D \cdot \rho$ by Lemma 3.7.

If $u = s\theta \sqsupset s$, then still if $u \prec_t \max_t(D)$ we have $s\theta\rho = u\rho \prec_t \max_t(D\rho)$, again implying $(s \approx t) \cdot \theta\rho \prec_{cc} D \cdot \rho$ by Lemma 3.7.

Otherwise, we have $\max_t(D) = u = s\theta \sqsupset s$. Consider the following cases.

If $D$ is not a positive unit, then the second part of Lemma 3.7 also implies $(s \approx t) \cdot \theta\rho \prec_{cc} D \cdot \rho$, as $\max_t(s\theta\rho \approx t\theta\rho) = s\theta\rho = u\rho = \max_t(D)$ (and $s\theta\rho \approx t\theta\rho$ is a positive unit clause).

If $D$ is a positive unit, then it is of the form $s\theta = v$ (where, recall, $s\theta \succ_t v$). But since $s\theta \sqsupset s$, then $s \cdot \theta\sigma \prec_{tc} s\theta \cdot \rho$, by (3.4). Then, by the rest of the definition of $\succ_{cc}$, we have $(s \approx t) \cdot \theta\rho \prec_{cc} (s\theta \approx v) \cdot \rho = D \cdot \rho$. □

## 3.2 Closure redundancy

In order to understand the advantages of our novel notion of redundancy, let us present the standard notion of redundancy first [Bachmair and Ganzinger 2001, p. 39], so that we may contrast them.

**Definition 3.6** (Standard redundancy criterion, clauses)**.** A clause $C$ is redundant in a set $S$ if all $C\theta \in \mathrm{GInsts}(C)$ follow from smaller ground instances in $\mathrm{GInsts}(S)$ – i.e., for all $C\theta \in \mathrm{GInsts}(C)$ there exists a set $G \subseteq \mathrm{GInsts}(S)$ such that $G \models C\theta$ and $\forall D\rho \in G.\, D\rho \prec_c C\sigma$.

**Definition 3.7** (Standard redundancy criterion, inferences)**.** An inference

$$C_1, \ldots, C_n \vdash D$$

is redundant in a set $S$ if, for all $\theta \in \mathrm{GSubs}(C_1, \ldots, C_n, D)$, the clause $D\theta$ follows from clauses in $\mathrm{GInsts}(S)$ which are smaller (wrt. $\succ_c$) than the maximal element of $\{C_1\theta, \ldots, C_n\theta\}$.

These definitions are both parametrised by an ordering on clauses, $\succ_c$. Unfortunately, this standard notion of redundancy does not cover many simplifications such as AC normalisation, ground joinability, a large class of demodulations, and others (which we discuss in Chapter 4).

*Example* 3.2. Consider unit clauses

$$S = \{f(x) \approx g(x),\ g(b) \approx b\} \tag{3.11}$$

where $f(x) \succ_t g(x) \succ_t b$. Then take the clause $C = f(b) \approx b$, which has only one ground instance (namely: itself). This clause is not redundant in $S$ (per Definition 3.6), because although it follows from $f(b) \approx g(b)$ and $g(b) \approx b$, which are both instances of clauses in $S$, the former is not smaller than $C$. Indeed, $C$ does not follow from *any* set of smaller (wrt. $\succ_c$) ground instances of clauses in $S$.

*Example* 3.3. Consider the set

$$S = \{f(x,y) \approx f(y,x)\}\,. \tag{3.12}$$

All ground instances of $f(x, f(y,z)) \approx f(f(z,y),x)$ follow from ground instances of the sole clause in $S$, however — again — not all of them follow from the set of ground instances of the clause in $S$ which are smaller than them (for instance, simply $f(a, f(b,b)) \approx f(f(b,b),a)$), and therefore the clause is not redundant in $S$.

By taking the notion of "instance" to mean closure rather than clause, and then using $\succ_{cc}$ rather than $\succ_c$ to formulate the definition, we adapt this redundancy notion to a closure-based one, which allows for such simplifications.

**Definition 3.8** (Closure redundancy criterion, clauses)**.** A clause $C$ is **closure redundant** in a set $S$ if all $C \cdot \theta \in \mathrm{GClos}(C)$ follow from smaller ground closures in $\mathrm{GClos}(S)$ — i.e., for all $C \cdot \theta \in \mathrm{GClos}(C)$ there exists a set $G \subseteq \mathrm{GClos}(S)$ such that $G \models C \cdot \theta$ and $\forall D \cdot \rho \in G.\, D \cdot \rho \prec_{cc} C \cdot \sigma$.

Although Definition 3.6 and Definition 3.8 look quite similar, let us revisit the examples.

*Example* 3.4. Consider again unit clauses $S = \{f(x) \approx g(x), g(b) \approx b\}$ where $f(x) \succ_t g(x) \succ_t b$, and let again $C = f(b) \approx b$. Now $C$ *is* closure redundant: it has one ground instance (this time represented by a *closure* $(f(b) \approx b) \cdot id$, rather than a clause),[5] which follows from closures $(f(x) \approx g(x)) \cdot (x \mapsto b)$ and

[5]Actually, there are infinitely many ground closures: $(f(b) \approx b) \cdot \theta$ for any substitution $\theta$; however, they are all representing the same clause and have the same base clause, hence we say by abuse of language that there is only one distinct ground closure.

$(g(b) \approx b) \cdot id$, which are both instances of clauses in $S$ and (crucially!) are both smaller (wrt. $\succ_{cc}$) than $(f(b) \approx b) \cdot id$. This is because

$$\begin{aligned} g(x) \cdot (x \mapsto b) \ \prec_{tc}\ f(x) \cdot (x \mapsto b) \ \prec_{tc}\ f(b) \cdot id \\ \Rightarrow\ (f(x) \approx g(x)) \cdot (x \mapsto b) \prec_{cc} (f(b) \approx b) \cdot id \end{aligned} \tag{3.13a}$$

and

$$\begin{aligned} b \cdot id \prec_{tc} g(b) \cdot id \prec_{tc} f(b) \cdot id \\ \Rightarrow\ (g(b) \approx b) \cdot id \prec_{cc} (f(b) \approx b) \cdot id \end{aligned} \tag{3.13b}$$

and the clauses represented by $(f(x) \approx g(x)) \cdot (x \mapsto b)$ and $(g(b) \approx b) \cdot id$ (namely, $f(b) \approx g(b)$ and $g(b) \approx b$) imply the clause represented by $(f(b) \approx b) \cdot id$ (the sole instance of $f(b) \approx b$).

In other words, the new redundancy criterion allowed a demodulation (from $f(b) \approx b$ to $g(b) \approx b$ via $f(x) \approx g(x)$) even when the (clause) instance of the equation we demodulated with is greater than the target literal, simply because the matching substitution was not a renaming. We will show rigorously how this is permitted in the general case in Chapter 4, and we show that this considerably simplifies the applicability condition on demodulation, and perhaps more importantly, when dealing with theories such as AC, it allows us to use AC axioms to normalise clauses where standard demodulation is not be applicable.

*Example* 3.5. Now all ground instances of $f(x, f(y, z)) \approx f(f(z, y), x)$ follow from smaller ground instances of $f(x, y) \approx f(y, x)$, hence the former is closure redundant wrt. the latter (showing this takes some calculation, with which we will concern ourselves in Section 4.3).

Among the properties of this redundancy relation, we highlight the following, which will be invoked during Chapter 4:

**Lemma 3.9** (Independence of redundant clauses)**.** If $C$ is closure redundant in $S$ and $C'$ is closure redundant in $S'$, then $C$ is closure redundant in $(S \cup S') \setminus \{C'\}$ (even if $C' \in S$ or $S = S'$).

Likewise, we extend the standard notion of redundant inference.

**Definition 3.9** (Closure redundancy criterion, inferences)**.** An inference

$$C_1, \ldots, C_n \vdash D$$

is **closure redundant** in a set $S$ if, for all $\theta \in \mathrm{GSubs}(C_1, \dots, C_n, D)$, the closure $D \cdot \theta$ follows from closures in $\mathrm{GClos}(S)$ which are smaller wrt. $\succ_{cc}$ than the maximal element of $\{C_1 \cdot \theta, \dots, C_n \cdot \theta\}$.[6]

Let us establish the following connection between closure redundant inferences and closure redundant clauses.

**Definition 3.10.** An inference $C_1, \dots, C_n \vdash D$ is **reductive** if for all $\theta \in \mathrm{GSubs}(C_1, \dots, C_n, D)$ we have $D \cdot \theta \prec_{cc} \max\{C_1 \cdot \theta, \dots, C_n \cdot \theta\}$.

**Lemma 3.10.** If the conclusion of a reductive inference is in $S$ or is closure redundant in $S$, then the inference is closure redundant in $S$.

*Proof.* If $D$ is in $S$, then all $D \cdot \theta$ are in $\mathrm{GClos}(S)$. But if the inference is reductive then $D \cdot \theta \prec_{cc} \max\{C_1 \cdot \theta, \dots, C_n \cdot \theta\}$, so it trivially follows from a closure smaller than that maximal element: itself.

If $D$ is redundant, then all $D \cdot \theta$ follow from smaller closures in $\mathrm{GClos}(S)$. But if the inference is reductive then again $D \cdot \theta \prec_{cc} \max\{C_1 \cdot \theta, \dots, C_n \cdot \theta\}$, so it also follows from closures smaller than that maximal element. $\square$

A set of clauses $S$ is **saturated up to closure redundancy** if any inference $C_1, \dots, C_n \vdash D$, whose premises $C_1, \dots, C_n$ are in $S$ and not closure redundant in $S$, is closure redundant in $S$.

In the rest of this work, we work exclusively with the new notion of closure redundancy, and therefore we will refer to it simply as "redundancy", when clear form the context.

## 3.3 Model construction

We will now prove the central result of this chapter:

**Theorem 3.11.** The superposition inference system (2.7) is refutationally complete wrt. closure redundancy, that is, if a set of clauses is saturated up to closure redundancy and does not contain the empty clause $\bot$, then it is satisfiable.

[6]Or equivalently, and more conveniently, if for all $\theta \in \mathrm{GSubs}(C_1, \dots, C_n, D)$ the closure $D \cdot \theta$ follows from closures in $\mathrm{GClos}(S)$ which are smaller wrt. $\succ_{cc}$ than some element of $\{C_1 \cdot \theta, \dots, C_n \cdot \theta\}$.

*Proof.* This is a proof by Noetherian induction on the ground closures of saturated sets, using the closure ordering defined in Definition 3.4. Let $N$ be a set of clauses such that $\bot \notin N$, and $G = \mathrm{GClos}(N)$. Let us assume $N$ is saturated up to closure redundancy. We will build a model for $G$, and hence for $N$ as follows.

As noted in page 41, a model is represented by a convergent term rewrite system (we will show convergence in Lemma 3.12), such that a closure $C \cdot \theta$ is true in a given model $R$ if at least one of its positive literals $(s \approx t) \cdot \theta$ has $s\theta \downarrow_R t\theta$, or if at least one of its negative literals $(s \not\approx t) \cdot \theta$ has $s\theta \not\downarrow_R t\theta$.

For each ground closure $C \cdot \theta \in G$, the partial model $R_{C\cdot\theta}$ and the set $\epsilon_{C\cdot\theta}$ are mutually recursively defined. $R_{C\cdot\theta}$ is the following rewrite system, defined by induction wrt. $\succ_{cc}$:

$$R_{C\cdot\theta} = \bigcup_{D\cdot\sigma \prec_{cc} C\cdot\theta} \epsilon_{D\cdot\sigma}\,, \tag{3.14}$$

while $\epsilon_{D\cdot\sigma}$ denotes a set, with 0 or 1 elements, which may or may not contain a new rewrite rule (for the partial models which include it). We define it in terms of the following conditions. If:

a. $C \cdot \theta$ is false in $R_{C\cdot\theta}$,
b. $l\theta \approx r\theta$ strictly maximal in $C\theta$,[7]
c. $l\theta \succ_t r\theta$, (3.15)
d. $C \cdot \theta \setminus \{(l \approx r) \cdot \theta\}$ is false in $R_{C\cdot\theta} \cup \{l\theta \to r\theta\}$, and
e. $l\theta$ is irreducible via $R_{C\cdot\theta}$,

then $\epsilon_{C\cdot\theta} = \{l\theta \to r\theta\}$ and $C \cdot \theta$ is called **productive**, otherwise $\epsilon_{C\cdot\theta} = \emptyset$.

The total model $R_\infty$ is thus $\bigcup_{D\cdot\sigma\in G} \epsilon_{D\cdot\sigma}$. Let also $R^{C\cdot\theta}$ be $R_{C\cdot\theta} \cup \epsilon_{C\cdot\theta}$. Our goal is to show that $R_\infty$ is a model for $G$. We will prove this by contradiction: if this is not the case, then there is a minimal (wrt. $\succ_{cc}$) closure $C \cdot \theta$ such that $R_\infty \not\models C \cdot \theta$. We will show by case analysis how the existence of this closure leads to a contradiction, if the set is saturated up to redundancy, and hence that all sets saturated by (2.7) up to redundancy have at least a model $R_\infty$.

First, some lemmas.

**Lemma 3.12.** $R_\infty$ and all $R_{C\cdot\theta}$ are convergent, i.e. terminating and confluent.

*Proof.* They are terminating since the rewrite relation is contained in $\succ_t$, which is well-founded. For confluence it is sufficient to show that left hand sides of rules in $R_\infty$ are irreducible in $R_\infty$. Assume that $l \to r$ and $l' \to r'$ are two rules

[7]Wrt. (3.2).

produced by closures $C \cdot \theta$ and $D \cdot \sigma$ respectively. By (3.15b) and (3.15c) we have $l = \max(C\theta)$ and $l' = \max(D\sigma)$. Wlog. assume $l$ is reducible by $l' \to r'$. Then $l \trianglerighteq l'$, and since $\succ_t$ is a simplification order, then $l \succeq_t l'$.

If $l \succ_t l'$, then we have $\max(C\theta) \succ_t \max(D\sigma) \Rightarrow C \cdot \theta \succ_{cc} D \cdot \sigma$ (by Lemma 3.7). But then $C \cdot \theta$ could not be productive due to (3.15e).

If $l = l'$ then both rules can reduce each other, and again due to (3.15e) whichever closure is larger would not be productive. In either case we obtain a contradiction. □

**Lemma 3.13.** If $R^{C\cdot\theta} \models C \cdot \theta$, then $R_{D\cdot\sigma} \models C \cdot \theta$ for any $D \cdot \sigma \succ_{cc} C \cdot \theta$, and $R_\infty \models C \cdot \theta$.

*Proof.* If a positive literal $s \approx t$ of $C\theta$ is true in $R^{C\cdot\theta}$, then $s \downarrow_{R^{C\cdot\theta}} t$. Since no rules are ever removed during the model construction, then $s \downarrow_{R_{D\cdot\sigma}} t$ and $s \downarrow_{R_\infty} t$.

If a negative literal $(s \not\approx t) \cdot \theta$ of $C \cdot \theta$ is true in $R^{C\cdot\theta}$, then $s\theta \not\downarrow_{R^{C\cdot\theta}} t\theta$. Wlog. assume that $s\theta \succ_t t\theta$. Consider a productive closure $D \cdot \sigma \succ_{cc} C \cdot \theta$ that produced a rule $l\sigma \to r\sigma$. Let us show that $l\sigma \to r\sigma$ cannot reduce $s\theta \not\approx t\theta$. Assume otherwise. By (3.15b) and (3.15c), $l\sigma = \max(D\sigma)$, so if $l\sigma \to r\sigma$ reduces either $t\theta$ or a strict subterm of $s\theta$, then $l\sigma \prec_t s\theta$, then $\max(D\sigma) \prec_t \max(C\theta)$, then $D \cdot \sigma \prec_{cc} C \cdot \theta$ (Lemma 3.7), which is a contradiction. If, on the other hand, $l\sigma = s\theta$, then since $C \cdot \theta$ cannot be a positive unit, $D \cdot \sigma$ and $C \cdot \theta$ are compared as follows.

- $M_{cc}(C \cdot \theta)$ contains $\{s \cdot \theta, t \cdot \theta, s\theta \cdot id, t\theta \cdot id\}$ (if unit) or $\{s\theta \cdot id, t\theta \cdot id, s\theta \cdot id, t\theta \cdot id\}$ (if non-unit).
- $M_{cc}(D \cdot \sigma)$ contains $\{l\sigma \cdot id\}$ and $\{r\sigma \cdot id\}$ (if unit) or $\{l\sigma \cdot id, r\sigma \cdot id\}$ (if non-unit).

But we have

$$\begin{aligned}
&\{s\theta \cdot id, t\theta \cdot id, s\theta \cdot id, t\theta \cdot id\}\\
\succeq_{tc}\ &\{s \cdot \theta\ \ , t \cdot \theta\ \ , s\theta \cdot id, t\theta \cdot id\}\\
\ggcurly_{tc}\ &\{l\sigma \cdot id, r\sigma \cdot id\}\\
\ggcurly_{tc}\ &\{l \cdot \sigma\} \text{ and } \ggcurly_{tc} \{r \cdot \sigma\}
\end{aligned} \tag{3.16}$$

because of Lemma 3.4, and since $s\theta = l\sigma \succ_t t\sigma$ implies $s\theta \cdot id = l\sigma \cdot id$ and $s \cdot \theta \succ_{tc} r\sigma \cdot id$. This contradicts $D \cdot \sigma \succ_{cc} C \cdot \theta$. □

**Lemma 3.14.** If $C \cdot \theta = (C' \vee l \approx r) \cdot \theta$ is productive, then $R_{D\cdot\sigma} \not\models C' \cdot \theta$ for any $D \cdot \sigma \succ_{cc} C \cdot \theta$, and $R_\infty \not\models C' \cdot \theta$.

*Proof.* All literals in $C' \cdot \theta$ are false in $R^{C\cdot\theta}$ by (3.15d). For all negative literals $(s \not\approx t) \cdot \theta$ in $C' \cdot \theta$, if they are false then $s\theta \downarrow_{R^{C\cdot\theta}} t\theta$. Since no rules are ever removed during the model construction then $s\theta \downarrow_{R_{D\cdot\sigma}} t\theta$ and $s\theta \downarrow_{R_\infty} t\theta$.

For all positive literals $(s \approx t) \cdot \theta$ in $C' \cdot \theta$, if they are false in $R^{C\cdot\theta}$ then $s\theta \not\downarrow_{R^{C\cdot\theta}} t\theta$. By (3.15b) and (3.15c) we have $l\theta \succeq_t s\theta$ and $l\theta \succeq_t t\theta$. Any closure $D \cdot \sigma$ that produces a rule $l'\sigma \to r'\sigma$ which reduces $s\theta$ or $t\theta$ must have either have $l'\sigma \prec_t l\theta$, in which case $\max(D\sigma) \prec_t \max(C\theta) \Rightarrow D \cdot \sigma \prec_{cc} C \cdot \theta$ (Lemma 3.7), or $l'\sigma = l\theta$, in which case whichever clause is bigger would not be productive due to (3.15e). ☐

We are now ready to prove the main proposition by Noetherian induction on closures, using the closure ordering $\succ_{cc}$ (see Lemma 3.1); namely that for all $C \cdot \theta \in G$ we have $R_\infty \models C \cdot \theta$.

In fact, we will show a stronger result: that for all $C \cdot \theta \in G$ we have $R^{C\cdot\theta} \models C \cdot \theta$ (the former result follows from the latter by Lemma 3.13).

If this is not the case, then there exists a minimal counterexample $C \cdot \theta \in G$ which is false in $R^{C\cdot\theta}$, i.e. $R^{C\cdot\theta} \not\models C \cdot \theta$. Therefore, by induction hypothesis all closures $D \cdot \sigma \in G$ such that $D \cdot \sigma \prec_{cc} C \cdot \theta$ have $R^{D\cdot\sigma} \models D \cdot \sigma$, so by Lemma 3.13 we have $R_{C\cdot\theta} \models D \cdot \sigma$ (and $R^{C\cdot\theta} \models D \cdot \sigma$).

Our case analysis is structured as follows: after each case is discharged, subsequent cases will assume that the negation of the previous cases holds.

**Case 1.** $C$ is redundant.

*Proof.* By definition, $C \cdot \theta$ follows from smaller closures in $G$. But if $C \cdot \theta$ is the minimal closure which is false in $R^{C\cdot\theta}$, then all smaller $D \cdot \sigma$ are true in $R^{D\cdot\sigma}$, which (as noted above) means that all smaller $D \cdot \sigma$ are true in $R_{C\cdot\theta}$, which means $C \cdot \theta$ is true in $R_{C\cdot\theta}$, which is a contradiction. ☐

**Case 2.** There is a reductive inference $C, \ldots \vdash D$ which is redundant, such that $\{C, \ldots\} \subseteq N$, $C \cdot \theta$ is maximal in $\{C \cdot \theta, \ \ldots\}$, and $D \cdot \theta \models C \cdot \theta$.

*Proof.* Then $D \cdot \theta$ is implied by closures in $G$ smaller than $C \cdot \theta$. But since those closures are true in $R^{C\cdot\theta}$, then $D \cdot \theta$ is true, and since $D \cdot \theta$ implies $C \cdot \theta$, then $C \cdot \theta$ is true in $R^{C\cdot\theta}$, which is a contradiction. ☐

**Case 3.** $C$ contains a variable $x$ such that $x\theta$ is reducible.

*Proof.* Then $R^{C\cdot\theta}$ contains a rule $x\theta \to t$. Let $\theta'$ be identical to $\theta$ except that it maps $x$ to $t$. Then $C\cdot\theta' \prec_{cc} C\cdot\theta$, and therefore $C\cdot\theta'$ is true in $R_{C\cdot\theta}$. But $C\cdot\theta'$ is true in $R^{C\cdot\theta}$ iff $C\cdot\theta$ in $R^{C\cdot\theta}$, since $x\theta \downarrow_{R^{C\cdot\theta}} t$, therefore $C\cdot\theta$ is also true in $R^{C\cdot\theta}$, which is a contradiction. □

**Case 4.** Neither of the previous cases apply, and $C$ contains a *negative* literal which is selected in the clause, i.e., $C\cdot\theta = (C' \vee s \not\approx t)\cdot\theta$ with $s \not\approx t$ selected in $C$.

*Proof.* Then either $s\theta \not\downarrow_{R^{C\cdot\theta}} t\theta$ and $C\cdot\theta$ is true and we are done, or else $s\theta \downarrow_{R^{C\cdot\theta}} t\theta$. By (3.15a) and (3.15d), $C\cdot\theta$ is only productive if $(s \not\approx t)\cdot\theta$ is false in $R_{C\cdot\theta}$ and $R^{C\cdot\theta}$, so $s\theta \downarrow_{R^{C\cdot\theta}} t\theta$ iff $s\theta \downarrow_{R_{C\cdot\theta}} t\theta$. Wlog., let us assume $s\theta \succeq_t t\theta$.

**Subcase 4.1.** $s\theta = t\theta$.

*Proof.* Then $s$ and $t$ are unifiable, meaning that there is an equality resolution inference

$$\frac{C' \vee s \not\approx t}{C'\sigma}, \quad \text{with } \sigma = \operatorname{mgu}(s,t), \tag{3.17}$$

with premise in $N$.

Take the instance $C'\sigma\cdot\rho$ of the conclusion such that $\sigma\rho = \theta$; it always exists since $\sigma = \operatorname{mgu}(s,t)$. Also, since the mgu is idempotent [Baader and Nipkow 1998] then $\sigma\theta = \sigma(\sigma\rho) = \sigma\rho = \theta$, so $C'\sigma\cdot\rho = C'\sigma\cdot\theta$.

We will show that $C\cdot\theta = (C' \vee s \not\approx t)\cdot\sigma\rho \succ_{cc} C'\sigma\cdot\rho = C'\sigma\cdot\theta$.

– If $C'$ is empty, then this is trivial.

– If $C'$ has more than 1 element, then this is also trivial ($M_{cc}(C\cdot\theta) \supset M_{cc}(C'\sigma\cdot\theta)$).

– If $C'$ has exactly 1 element, then let $C' = \{s' \approx t'\}$. We have $(s' \approx t' \vee s \not\approx t)\cdot\sigma\rho \succ_{cc} (s' \approx t')\sigma\cdot\rho$ if

$$\begin{aligned} &M_{cc}((s' \approx t' \vee s \not\approx t)\cdot\sigma\rho) \\ &= M_{cc}((s' \approx t' \vee s \not\approx t)\sigma\rho\cdot id) \\ &\ggcurly_{tc} M_{cc}((s' \approx t')\sigma\cdot\rho) \end{aligned} \tag{3.18}$$

(see definition of $\succ_{cc}$) which is true since, by Lemma 3.4, $s'\sigma\rho\cdot id \succeq_{tc} s'\sigma\cdot\rho$ and $t'\sigma\rho\cdot id \succeq_{tc} t'\sigma\cdot\rho$.

Now notice also that if $C'\sigma \cdot \rho$ is true then $(C' \vee \cdots) \cdot \sigma\rho$ must also be true. Recall that Case 2 does not apply. But we have shown that this inference is reductive, with $C \in N$, $C \cdot \theta$ trivially maximal in $\{C \cdot \theta\}$, and that the instance $C'\sigma \cdot \theta$ of the conclusion implies $C \cdot \theta$. So for Case 2 not to apply the inference must be non-redundant. Also since Case 1 doesn't apply then the premise is not redundant. This means that the set is not saturated, which is a contradiction. □

**Subcase 4.2.** $s\theta \succ t\theta$.

*Proof.* Then (recall that $s\theta \downarrow_{R_{C\cdot\theta}} t\theta$) $s\theta$ must be reducible by some rule in $R_{C\cdot\theta}$. Let us say that this rule is $l\theta \to r\theta$, produced by a closure $D \cdot \theta$ smaller than $C \cdot \theta$.[8] Therefore closure $D \cdot \theta$ must be of the form $(D' \vee l \approx r) \cdot \theta$, with $l\theta \approx r\theta$ strictly maximal in $D\theta$, and $D' \cdot \theta$ false in $R_{D\cdot\theta}$. Also note that $D$ cannot be redundant, or else $D \cdot \theta$ would follow from smaller closures, but those closures (which are smaller than $D \cdot \theta$ and therefore smaller than $C \cdot \theta$) would be true, so $D \cdot \theta$ would be also true in $R_{D\cdot\theta}$, so by (3.15a) it would not be productive. Then $l\theta = u\theta$ for some subterm $u$ of $s$, meaning $l$ is unifiable with $u$, meaning there exists a superposition inference

$$\frac{D' \vee l \approx r \quad C' \vee s[u] \not\approx t}{(D' \vee C' \vee s[u \mapsto r] \not\approx t)\sigma}, \quad \text{with } \sigma = \mathrm{mgu}(l, u), \tag{3.19}$$

Similar to what we did before, consider the instance $(D' \vee C' \vee s[u \mapsto r] \not\approx t)\sigma \cdot \rho$ with $\sigma\rho = \theta$.[9] We wish to show that this instance of the conclusion is smaller than $C \cdot \theta$ (an instance of the second premise), that is that

$$(C' \vee s \not\approx t) \cdot \sigma\rho \ \succ_{cc} \ (D' \vee C' \vee s[u \mapsto r] \not\approx t)\sigma \cdot \rho\,. \tag{3.20}$$

Several cases arise:

– $\underline{C' \neq \emptyset}$. Then both premise and conclusion are non-unit, so comparing them means comparing

$$M_{cc}((C'\theta \vee s\theta \not\approx t\theta) \cdot id) \text{ and } M_{cc}((D'\theta \vee C'\theta \vee s\theta[u\theta \mapsto r\theta] \not\approx t\theta) \cdot id)\,, \tag{3.21}$$

[8]We can use the same substitution $\theta$ on both $C$ and $D$ by simply assuming wlog. that they have no variables in common.

[9]And again note that the mgu $\sigma$ is idempotent so $(D' \vee C' \vee s[u \mapsto r] \not\approx t)\sigma \cdot \rho = (D' \vee C' \vee s[u \mapsto r] \not\approx t)\sigma \cdot \theta$.

or after removing common elements, comparing

$$\{\{s\theta \cdot id, t\theta \cdot id, s\theta \cdot id, t\theta \cdot id\}\} \text{ and } M_{cc}((D'\theta \vee s\theta[u\theta \mapsto r\theta] \not\approx t\theta) \cdot id)\,. \tag{3.22}$$

We have:

(i) $u\theta = l\theta \succ_t r\theta \;\Rightarrow\; s\theta \cdot id \succ_{tc} s\theta[u\theta \mapsto r\theta] \cdot id$,
(ii) $s\theta \succ_t t\theta \;\Rightarrow\; s\theta \cdot id \succ_{tc} t\theta \cdot id$, and
(iii) $s\theta \succeq_t l\theta \succ_t r\theta$ and $l\theta \approx r\theta$ strictly maximal in $l\theta \approx r\theta \vee D'\theta$;

these imply

$$\begin{aligned} &\{s\theta \cdot id, t\theta \cdot id, s\theta \cdot id, t\theta \cdot id\} \\ \underline{\ggcurly}_{tc}\, &\{s \cdot \theta \quad, t \cdot \theta \quad, s\theta \cdot id, t\theta \cdot id\} \\ \ggcurly_{tc}\, &M_{lc}(L\theta \cdot id), \quad \text{for all } L \in D' \end{aligned} \tag{3.23}$$

(by Lemmas 3.4 and 3.6). Therefore, $M_{cc}((C'\theta \vee s\theta \not\approx t\theta) \cdot id) \ggg_{tc} M_{cc}((D'\theta \vee C'\theta \vee s\theta[u\theta \mapsto r\theta] \not\approx t\theta) \cdot id)$.

– $\underline{C' = \emptyset \text{ and } D' \neq \emptyset}$. Then we compare $M_{cc}((s \not\approx t) \cdot \theta)$ and $M_{cc}((D' \vee s[u \mapsto r] \not\approx t)\theta \cdot id)$, i.e.

$$\{\{s \cdot \theta, t \cdot \theta, s\theta \cdot id, t\theta \cdot id\}\} \text{ and } M_{cc}((D' \vee s[u \mapsto r] \not\approx t)\theta \cdot id) \tag{3.24}$$

But due to the facts enumerated in the previous point we have already shown that the former is greater wrt. $\ggg_{tc}$ than the latter.

– $\underline{C' = \emptyset \text{ and } D' = \emptyset}$. Then simply $s\theta[u\theta] \succ_t s\theta[u\theta \mapsto r\theta]$ means $s[u] \cdot \sigma\rho \succ_{tc} s[u \mapsto r]\sigma \cdot \rho$, which since $s\theta \succ_t t\theta$, means $(s[u] \not\approx t) \cdot \sigma\rho \succ_{cc} (s[u \mapsto r] \not\approx t)\sigma \cdot \rho$.

In all these cases this instance of the conclusion is always smaller than the instance $C \cdot \theta$ of the second premise. Note also that $C \cdot \theta$ is maximal in $\{C \cdot \theta, D \cdot \theta\}$. Also, since $D' \cdot \theta$ is false in $R_{C \cdot \theta}$ (by Lemma 3.14) and $(s[u \mapsto r] \not\approx t) \cdot \theta$ is false in $R_{C \cdot \theta}$ (since $(s \not\approx t) \cdot \theta$ is in the false closure $C \cdot \theta$, $u\theta \downarrow_{R_{C \cdot \theta}} r\theta$, and the rewrite system is confluent), then in order for that instance of the conclusion to be true in $R_{C \cdot \theta}$ it must be the case that $C'\sigma \cdot \rho$ is true in $R_{C \cdot \theta}$. But if the latter is true then $C \cdot \theta = (C' \vee \cdots) \cdot \sigma\rho$ is true, in $R_{C \cdot \theta}$. In other words that instance of the conclusion implies $C \cdot \theta$. Therefore again, since Case 1 and Case 2 do not apply, we conclude that the inference is non-redundant with non-redundant premises, so the set is not saturated, which is a contradiction. □

This proves all subcases. □

**Case 5.** Neither of the previous cases apply, so *all* selected literals in $C$ are positive, i.e., $C \cdot \theta = (C' \vee s \approx t) \cdot \theta$ with $s \approx t$ selected in $C$.

*Proof.* Then, since if the selection function doesn't select a negative literal then it must select all maximal ones, wlog. one (and only one) of the selected literals $s \approx t$ maximal in $C$ must have $s\theta \approx t\theta$ maximal in $C\theta$. Then if either $C' \cdot \theta$ is true in $R_{C\cdot\theta}$, or $\epsilon_{C\cdot\theta} = \{s\theta \to t\theta\}$, or $s\theta = t\theta$, then $C \cdot \theta$ is true in $R^{C\cdot\theta}$ and we are done. Otherwise, $\epsilon_{C\cdot\theta} = \emptyset$, $C' \cdot \theta$ is false in $R_{C\cdot\theta}$, and wlog. $s\theta \succ_t t\theta$.

**Subcase 5.1.** $s\theta \approx t\theta$ maximal but not strictly maximal in $C\theta$.

*Proof.* If this is the case, then there is at least one other maximal positive literal in the clause. Let $C \cdot \theta = (C'' \vee s \approx t \vee s' \approx t') \cdot \theta$, where $s\theta = s'\theta$ and $t\theta = t'\theta$. Therefore $s$ and $s'$ are unifiable and there is an equality factoring inference:

$$\frac{C'' \vee s \approx t \vee s' \approx t'}{(C'' \vee s \approx t \vee t \not\approx t')\sigma}, \quad \text{with } \sigma = \text{mgu}(s, s'). \tag{3.25}$$

Take the instance of the conclusion $(C'' \vee s \approx t \vee t \not\approx t')\sigma \cdot \rho$ with $\sigma\rho = \theta$. This is smaller than $C \cdot \theta$, since:

- $\max(C\theta) = \max((C'' \vee s \approx t \vee t \not\approx t')\theta)$,
- Neither is a unit clause,
- $\{s'\theta, t'\theta\} \succ\!\!\succ_t \{t\theta, t'\theta, t\theta, t'\theta\}$, and
- Lemmas 3.2 and 3.7 apply.

Since $t\theta = t'\theta$ and $C''\sigma \cdot \rho$ is false in $R_{C\cdot\theta}$, this instance of the conclusion is true in $R_{C\cdot\theta}$ iff $(s\sigma \approx t\sigma) \cdot \rho$ is true in $R_{C\cdot\theta}$. But if the latter is true in $R_{C\cdot\theta}$ then $(s \approx t \vee \cdots) \cdot \sigma\rho$ also is. Therefore that instance of the conclusion implies $C \cdot \theta$. As such, and since again Cases 1 and 2 do not apply, we have a contradiction. □

**Subcase 5.2.** $s\theta \approx t\theta$ strictly maximal in $C\theta$, and $s\theta$ reducible (in $R_{C\cdot\theta}$).

*Proof.* This is similar to Subcase 4.2. If $s\theta$ is reducible, say by a rule $l\theta \to r\theta$, then (since $\epsilon_{C\cdot\theta} = \emptyset$) this is produced by some closure $D \cdot \theta$ smaller than $C \cdot \theta$, with $D \cdot \theta = (D' \vee l \approx r) \cdot \theta$, with the $l\theta \approx r\theta$ maximal in $D\theta$, $D$ not redundant, and with $D' \cdot \theta$ false in $R_{D\cdot\theta}$.

Then there is a superposition inference

$$D' \vee l \approx r \,,\, C \vee s[u] \approx t \vdash (D' \vee C' \vee s[u \mapsto r] \approx t)\sigma, \quad \sigma = \text{mgu}(l, u). \tag{3.26}$$

Again taking the instance $(D' \vee C' \vee s[u \mapsto r] \approx t)\sigma \cdot \rho$ with $\sigma\rho = \theta$, we see that it is smaller than $C \cdot \theta$ (see discussion in Subcase 4.2). Furthermore since $D' \cdot \theta$ and $C' \cdot \theta$ are false in $R_{C \cdot \theta}$, then that instance of the conclusion is true in $R_{C \cdot \theta}$ iff $(s[u \mapsto r] \approx t)\sigma \cdot \rho$ is. But since also $u\theta \downarrow_{R_{C \cdot \theta}} r\theta$, then $(s[u \mapsto r] \approx t)\sigma \cdot \rho$ implies $(s[u] \approx t)\sigma \cdot \rho$. Therefore that instance of the conclusion implies $C \cdot \theta$. Again this means we have a contradiction. □

**Subcase 5.3.** $s\theta \approx t\theta$ strictly maximal in $C\theta$, and $s\theta$ irreducible (in $R_{C \cdot \theta}$).

*Proof.* Since $C \cdot \theta$ is not productive, and at the same time all criteria in (3.15) except (3.15d) are satisfied, it must be that (3.15d) is not, that is $C' \cdot \theta$ must be true in $R^{C \cdot \theta} = R_{C \cdot \theta} \cup \{s\theta \to t\theta\}$. Then this must mean we can write $C' \cdot \theta = (C'' \vee s' \approx t') \cdot \theta$, where the latter literal is the one that becomes true with the addition of $\{s\theta \to t\theta\}$, whereas without that rule it was false.

But this means that $s'\theta \downarrow_{R^{C \cdot \theta}} t'\theta$ such that any rewrite proof needs at least one step where $s\theta \to t\theta$ is used, since $s\theta$ is irreducible by $R_{C \cdot \theta}$. Wlog. say $s'\theta \succ_t t'\theta$. Since:

(i) $\{s\theta, t\theta\} \succ\!\succ_t \{s'\theta, t'\theta\}$,
(ii) $s\theta \succ_t t\theta$, and
(iii) $s'\theta \succ_t t'\theta$,

then $s\theta \succeq_t s'\theta \succ_t t'\theta$, which implies $t'\theta \not\succeq s\theta$, which implies $s\theta \to t\theta$ cannot be used to reduce $t'\theta$, and similarly, nor to reduce $s'\theta$ if $s\theta \succ_t s'\theta$. Thus the only way it can reduce $s'\theta$ or $t'\theta$ is if $s\theta = s'\theta$. This means there is an equality factoring inference:

$$\frac{C'' \vee s' \approx t' \vee s \approx t}{(C'' \vee s' \approx t' \vee t \not\approx t')\sigma}, \quad \text{with } \sigma = \mathrm{mgu}(s, s'). \tag{3.27}$$

Taking $\theta = \sigma\rho$, we see that the instance of the conclusion $(C'' \vee t \not\approx t' \vee s \approx t)\sigma \cdot \rho$ is smaller than the instance of the premise $(C'' \vee s' \approx t' \vee s \approx t) \cdot \sigma\rho$ (see Subcase 5.1).

But we have said that $s'\theta \downarrow_{R^{C \cdot \theta}} t'\theta$, where the first rewrite step had to take place by rewriting $s'\theta = s\theta \to t\theta$, and the rest of the rewrite proof then had to use only rules from $R_{C \cdot \theta}$. In other words, this means $t\theta \downarrow_{R_{C \cdot \theta}} t'\theta$. As such, the literal $(t \not\approx t') \cdot \theta$ is false in $R_{C \cdot \theta}$, and so the conclusion is true in $R_{C \cdot \theta}$ iff rest of the closure is true in $R_{C \cdot \theta}$. But if the rest of the closure $(C'' \vee s' \approx t')\sigma \cdot \rho$ is true then so is $C \cdot \theta$, so that instance of the conclusion implies $C \cdot \theta$. Once again, this leads to a contradiction since none of Cases 1 and 2 apply and therefore the set must not be saturated. □

This proves all the subcases and the theorem. □

*Remark* 3.1. As part of this proof, we have also shown that all inferences in the superposition system are reductive, so per Lemma 3.10 one way to make inferences redundant is simply to add the conclusion.

# Chapter 4

# Simplifications

Everything must be made as simple as possible. But no simpler.

Albert Einstein (1933)

In this chapter, we will apply the results that we have just obtained to formulate several simplification rules and criteria for redundancy of inferences in the superposition calculus.

This chapter is divided into three main sections:

- First, we improve on the usual notion of demodulation via a novel rule of encompassment demodulation.
- Then, we present simplification rules based on joinability, ground joinability, and connectedness.
- Finally, we discuss specialised criteria for dealing with associative-commutative (AC) theories.

Of the work presented in this chapter, the parts on encompassment demodulation, AC normalisation, and AC joinability were peer-reviewed and published in Duarte and Korovin 2021, and the parts on general ground joinability, connectedness, and ground connectedness were peer-reviewed and published in Duarte and Korovin 2022.

Armed with Definition 3.8, the criterion for closure redundancy of clauses, we can formulate **simplification rules**, which we take to be inferences of the form

$$\frac{C_1 \quad \cdots \quad \cancel{D}}{D'} \tag{4.1}$$

where $D$ is closure redundant in $\{C_1, \ldots, D'\}$; we also require rules to be sound. We then say that $\{C_1, \ldots\}$ are used to simplify $D$, the deleted premise, into $D'$, the conclusion. In practice, this means that when the premises $C_1, \ldots, D$ have been derived, we can replace $D$ with $D'$ with no loss of completeness.

These rules include such things as removing tautologies, deleting trivial literals, deleting clauses subsumed by other clauses, etc.

To aid the statement of the constraints for rewrite-based simplification rules, we introduce the following definition, to be re-used throughout this chapter.

**Definition 4.1.** A rewrite via $l \approx r$ in clause $C[l\theta]$ is **admissible** if one of the following conditions holds:

(i) $C$ is not a positive unit, or

(ii) (let $C = s[l\theta] \approx t$ for some $\theta$)

(ii.a) $l\theta \neq s$, or

(ii.b) $l\theta \sqsupset l$, or

(ii.c) $s \prec_t t$,[1] or

(ii.d) $r\theta \prec_t t$.

In other words, given an equation $l \approx r$, where there exists a substitution $\theta$ such that $l\theta$ is a subterm in $C$, then the rewrite is said to be admissible if $C$ is not a positive unit, or if $l\theta$ occurs at a strict subterm position, or if $l\theta$ is less general than $l$ (i.e. $\theta$ is not a renaming), or if $l\theta$ occurs outside a maximal side, or if $r\theta$ is smaller than the other side.

This definition will be invoked for most of the simplification rules we will present in this chapter, as it neatly encapsulates the conditions which are sufficient for completeness of many rewrite-based simplification rules, as we shall begin to see in the next section.

[1]We note that (ii.c) is superfluous if $l\theta \succ r\theta$ (implies (ii.d)), but we include it since in practice it is easier to check, as it is local to the clause being rewritten and therefore needs to be checked at most once for each unit equational clause we attempt to rewrite, while (ii.d) needs to be checked with each rewrite attempt.

## 4.1 Demodulation

The standard demodulation rule used in virtually all state-of-the art superposition-based provers is as follows [Kovács and Voronkov 2013].

$$\text{Demodulation}\quad \frac{l \approx r \quad \cancel{C[l\theta]}}{C[l\theta \mapsto r\theta]}, \quad \begin{array}{l}\text{where } l\theta \succ_t r\theta,\\ \text{and } l\theta \approx r\theta \prec_c C.\end{array} \tag{4.2}$$

However, as discussed in Examples 3.2 and 3.4, this is not applicable when the rewrite happens at a top position of a literal and the equation happens not to be smaller than the clause being rewritten, wrt. $\prec_c$. Furthermore, there is an issue of performance, as the check $l\theta \approx r\theta \prec_c C$ may be expensive in practice (depending on the term ordering used, but usually at least linear in the size of $C$).

We present the following simplification rule, which has an entirely weaker constraint, and can therefore be applied in more instances.

$$\begin{array}{l}\text{Encompassment}\\ \text{Demodulation}\end{array}\quad \frac{l \approx r \quad \cancel{C[l\theta]}}{C[l\theta \mapsto r\theta]}, \quad \begin{array}{l}\text{where } l\theta \succ_t r\theta, \text{ and}\\ \text{rewrite via } l \approx r \text{ in } C \text{ is admissible.}\end{array} \tag{4.3}$$

In other words, demodulation is now allowed whenever $l$ (the lhs of the equation) is more general than $l\theta$ (the subterm being rewritten), and is also allowed unconditionally on clauses which are not positive unit equalities (even if $l\theta \approx r\theta \not\prec_c C$).

*Example* 4.1. Consider the equation $f(x) \approx s$, where $f(x) \succ_t s$. Then we can apply encompassment demodulation to simplify clauses

- $f(a) \approx t$ into $s \approx t$,
- $f(x) \not\approx t$ into $s \not\approx t$,
- $f(x) \approx t \vee C$ into $s \approx t \vee C$,
- $f(x) \approx t$ into $s \approx t$ (when $s \prec_t t$),

while the usual demodulation rule would require $s \prec_t t$ for any of these to go through.

The condition in (4.3) is therefore much weaker than the one given in (4.2), and, when we restrict ourselves to unit equalities, equivalent to the one in equational completion. More precisely:

*Remark* 4.1. Encompassment demodulation (4.3) subsumes the $\mathrm{Simplification}_2$, $\mathrm{Composition}_2$, and $\mathrm{Collapse}_2$ rules in the Bachmair, Dershowitz and Plaisted [1989] unfailing completion calculus, being equivalent to these on positive unit equational clauses.

We shall now prove that application of this rule does not compromise the completeness of superposition.

**Theorem 4.1.** Encompassment demodulation is a simplification rule of the superposition calculus wrt. closure redundancy.

*Proof.* This is a valid redundancy if all ground closures of $C[l\theta]$ follow from some smaller ground closures of $C[l\theta \mapsto r\theta]$ and $l \approx r$. Let $C \cdot \rho$ be an arbitrary ground closure of $C$. That $C \cdot \rho$ follows from $C[l\theta \mapsto r\theta] \cdot \rho$ and $(l \approx r) \cdot \theta\rho$, is trivial. The fact that if $l\theta \succ_t r\theta$ then, for all groundings $\rho$, $C[l\theta{\mapsto}r\theta]\cdot\rho \prec_{cc} C\cdot\rho$, also follows from Lemma 3.6. It remains to be shown that, for all $\rho$, $(l{\approx}r){\cdot}\theta\rho \prec_{cc} C \cdot \rho$.

If $C$ is not a positive unit, then since $l{\approx}r$ is a positive unit with $l\theta \preceq_t \max(C)$ (because $l\theta \trianglelefteq$ some subterm of $C$), we have $(l \approx r) \cdot \theta\rho \prec_{cc} C \cdot \rho$ by Lemma 3.7.

Otherwise, $C$ has the form $s[l\theta] \approx t$. If $l\theta \neq s$, that means $l\theta \lhd s \Rightarrow l\theta \prec_t s \Rightarrow l \cdot \theta\rho \prec_{tc} s \cdot \rho \Rightarrow (l \approx r) \cdot \theta\rho \prec_{cc} (s \approx t) \cdot \rho = C \cdot \rho$.

If $l\theta = s$, but $s$ is the minimal side of $s \approx t$ (i.e. $s \prec_t t$), then still we have $r\theta\rho \prec_t l\theta\rho = s\rho \prec_t t\rho \Rightarrow (l \approx r) \cdot \theta\rho \prec_{cc} (s \approx t) \cdot \rho = C \cdot \rho$.

If $l\theta = s \nprec_t t$, then still if $l\theta \sqsupset l$ then for all $\rho$, $l\theta \cdot \rho \succ_{tc} l \cdot \theta\rho$ (Lemma 3.4). Therefore $(l \approx r) \cdot \theta\rho \prec_{cc} (l\theta \approx t) \cdot \rho = C \cdot \rho$ for all $\rho$.

Finally, if $l\theta = l = s \nprec_t t$, we need to compare $t$ and $r\theta$. If $t \succ_t r\theta$, then, for all $\rho$, $t \cdot \rho \succ_{tc} r \cdot \theta\rho \Rightarrow (l \approx r) \cdot \theta\rho \prec_{cc} (l \approx t) \cdot \rho = C \cdot \rho$. $\square$

There are two main benefits from replacing standard demodulation with encompassment demodulation:

- The most obvious is that encompassment demodulation can be used to simplify more clauses than demodulation. This can enable (i) quick proofs to be found in situations where saturation with generating rules would take longer, and (ii) for more clauses to be pruned from the search space, accelerating performance.

- Furthermore, the test itself is cheaper. In the standard version of demodulation, a series of $\succ_t$ ordering checks (in the worst case as many as the number of literals in $C$) is necessary whenever a rewrite at a top position is attempted; in encompassment demodulation if the clause is non-unit or a negative unit clause, then the rewrite is immediately successful. Only if the clause is a unit equality do we need first to check if the instantiator $\theta$ is a renaming, which is quick in any reasonable data representation of a substitution, and only if $\theta$ is a renaming do we need to do one ordering check, wrt. the other side of the literal.

We also note that this is a "drop-in" replacement for standard demodulation, as its implementation is straightforward and requires no extra indexing, search, or traversal algorithms. Indeed, it has already been adopted by leading superposition provers [Suda 2022].

In the sequel, except when we explicitly disambiguate between "standard" demodulation and "encompassment" demodulation, we will refer to the latter by simply "demodulation".

## 4.2 Joinability

We can combine encompassment demodulation with simple criteria for the removal of $s \approx s$ and $s \not\approx s$ literals, since a clause $s \approx s \vee C$ is always redundant wrt. any set of clauses $S$, and a clause $s \not\approx s \vee C$ is always redundant wrt. $S \cup \{C\}$. By doing this, we obtain a class of joinability-based simplification criteria.

Indeed, this is already what application of standard/encompassment demodulation looks like in practice: the two sides of a literal are exhaustively demodulated, and if they are equal, the clause (if the literal is positive) or the literal (if the literal is negative) is thrown away. The interest of presenting this as a separate concept has more to do with how we can extend it further, as we shall see in a few pages.

We will first present the following definition.

**Definition 4.2** (Strong joinability)**.** Two terms are **strongly joinable** ($s \leadsto t$), **in** a clause $C$ **wrt.** a set of equations $S$, if either

(a) $s = t$, or

(b) $s \to s[l_1\sigma_1 \mapsto r_1\sigma_1] \xrightarrow{*} t$ via equations $l_i \approx r_i \in S$, where the rewrite via[2] $l_1 \approx r_1$ is admissible in $C$, or

(c) $s \to s[l_1\sigma_1 \mapsto r_1\sigma_1] \downarrow t[l_2\sigma_2 \mapsto r_2\sigma_2] \leftarrow t$ via equations $l_i \approx r_i \in S$, where the rewrites via $l_1 \approx r_1$ and $l_2 \approx r_2$ are admissible in $C$.

This is a subset of the 'plain' joinability relation (as defined in page 41), formulated directly in terms of a set of equations rather than on a set of rewrite rules, and referring to Definition 4.1 to encapsulate the conditions which will ensure completeness. Specifically, the condition of admissibility is only explicitly asserted for the first rewrite step that is undertaken on each of the terms. This is because — for all purposes for which this definition is invoked throughout this work — the admissibility of the first rewrite step on a term will imply the admissibility of the remaining ones.

We then have:

$$\text{Joinability} \qquad \frac{\cancel{s \approx t \vee C} \quad S}{}\,, \quad \begin{array}{l}\text{where } s \mathrel{\rotatebox[origin=c]{-90}{$\rightsquigarrow$}} t \\ \text{in } s \approx t \vee C \text{ wrt. } S,\end{array} \tag{4.4a}$$

$$\text{Joinability} \qquad \frac{\cancel{s \not\approx t \vee C} \quad S}{C}\,, \quad \begin{array}{l}\text{where } s \mathrel{\rotatebox[origin=c]{-90}{$\rightsquigarrow$}} t \\ \text{in } s \not\approx t \vee C \text{ wrt. } S.\end{array} \tag{4.4b}$$

**Theorem 4.2.** Joinability is a simplification rule of the superposition calculus wrt. closure redundancy.

*Proof.* If $s \mathrel{\rotatebox[origin=c]{-90}{$\rightsquigarrow$}} t$ in $s \approx t \vee C$ wrt. $S$, then one of these following points of Definition 4.2 holds:

(a), and the clause is $s \dot{\approx} s \vee C$.

(b), and there is a chain of (encompassment) demodulation inferences of the form

$$\begin{array}{llll} s \dot{\approx} t \vee C & , l_1\sigma_1 \approx r_1\sigma_1 & \vdash s_1 \dot{\approx} t \vee C \\ & \dots & \\ s_{i-1} \dot{\approx} t \vee C & , l_i\sigma_i \approx r_i\sigma_i & \vdash s_i \dot{\approx} t \vee C \\ & \dots & \\ s_{n-1} \dot{\approx} t \vee C & , l_n\sigma_n \approx r_n\sigma_n & \vdash s_n \dot{\approx} t \vee C\,, \end{array} \tag{4.5}$$

for some $n \geq 1$, and where $s_n = t$. The side-condition in (4.3), for the completeness of encompassment demodulation, is verified for each of these steps, since by Definition 4.2(b) we have that the rewrite via $l_1\sigma_1 \approx r_1\sigma_1$ is admissible (in $s \dot{\approx} t \vee C$), and because $l_i\sigma_i \trianglelefteq s_{i-1} \prec s$ (implying that $l_i\sigma_i \prec s$) we have that the rewrites via the remaining $l_i\sigma_i \approx r_i\sigma_i$ are admissible in $s \dot{\approx} t \vee C$ as well, as they satisfy Definition 4.1(ii.a).

[2]Recall what we mean by "rewrite via", on page 42.

(c), and similarly to the previous point we have a chain of encompassment demodulation inferences of the form

$$\begin{array}{lll} s \mathrel{\dot{\approx}} t \vee C & , l_1\sigma_1 \approx r_1\sigma_1 & \vdash s_1 \mathrel{\dot{\approx}} t \vee C \\ & \quad \dots & \\ s_{n-1} \mathrel{\dot{\approx}} t \vee C & , l_n\sigma_n \approx r_n\sigma_n & \vdash s_n \mathrel{\dot{\approx}} t \vee C \\ s_n \mathrel{\dot{\approx}} t \vee C & , l_{n+1}\sigma_{n+1} \approx r_{n+1}\sigma_{n+1} & \vdash s_n \mathrel{\dot{\approx}} t_1 \vee C \\ & \quad \dots & \\ s_n \mathrel{\dot{\approx}} t_{m-1} \vee C & , l_{n+m}\sigma_{n+m} \approx r_{n+m}\sigma_{n+m} & \vdash s_n \mathrel{\dot{\approx}} t_m \vee C \,, \end{array} \tag{4.6}$$

for some $n, m \geq 1$, and where $s_n = t_m$. The side-condition in (4.3) is again verified for each of these steps, since (identically to the previous point): by Definition 4.2(c) we have that the rewrites via $l_1\sigma_1 \approx r_1\sigma_1$ and $l_{m+1}\sigma_{m+1} \approx r_{m+1}\sigma_{m+1}$ are admissible, and the remaining ones are as well because $\forall i \in 2, \dots, n.\ l_i\sigma_i \trianglelefteq s_{i-1} \prec s$ and $\forall j \in 2, \dots, m.\ l_{n+j}\sigma_{n+j} \trianglelefteq t_{j-1} \prec t$, which implies that Definition 4.1(ii.a) is satisfied.

In all cases, we have that $s \mathrel{\dot{\approx}} t \vee C$ is redundant wrt. $S \cup \{u \mathrel{\dot{\approx}} u \vee C\}$, for some $u \preceq_t s$ and $\preceq_t t$. Then, by observing that

- $\{u \approx u \vee C\}$ is redundant wrt. $\emptyset$, and
- $\{u \not\approx u \vee C\}$ is redundant wrt. $\{C\}$,

simply by Lemma 3.9 we have that $s \approx t \vee C$ is redundant wrt. $S$ and $s \not\approx t \vee C$ is redundant wrt. $S \cup \{C\}$. □

## Ground joinability

However, this joinability rule is not sufficient in many situations. In particular, because each rewrite has to be oriented (that is, each step in a "proof" of joinability has to be done via an oriented instance of an equation in $S$), this precludes its usage in situations where the clause might in fact be redundant, but where there is no rewrite chain oriented with $\succ$ to prove it.

*Example* 4.2. A stereotypical example is associative-commutative operators. Let

$$S = \{f(x,y) \approx f(y,x),\ f(x, f(y,z)) \approx f(f(x,y),z)\}\,. \tag{4.7}$$

Take the clauses

$$f(x, f(y, z)) \approx f(x, f(z, y))\,, \tag{4.8a}$$
$$f(f(x, y), z) \approx f(x, f(z, y))\,. \tag{4.8b}$$

Inspection using Definition 3.8 shows that they are both redundant wrt. $S$, yet joinability cannot be applied here, and neither can any encompassment demodulation step[3] (nor the subsumption rule[4]).

The crucial observation is that, even though the clauses themselves cannot be rewritten by orientable instances of equations in $S$, *all* of their ground clauses can. That is to say, for example, for any ground closure

$$(f(x, f(y, z)) \approx f(x, f(z, y))) \cdot \theta \tag{4.9}$$

of (4.8a), we either have

- $f(y\theta, z\theta) \succ_t f(z\theta, y\theta)$, and the left-hand side $f(x, f(y, z)) \cdot \theta$ can be rewritten to $f(x, f(z, y)) \cdot \theta$ via a (smaller) instance of $f(x, y) \approx f(y, x)$, or
- $f(z\theta, y\theta) \succ_t f(y\theta, z\theta)$, and the right-hand side $f(x, f(z, y)) \cdot \theta$ can be rewritten to $f(x, f(y, z)) \cdot \theta$ via a (smaller) instance of $f(x, y) \approx f(y, x)$, or
- $f(y\theta, z\theta) = f(z\theta, y\theta)$,

which means that for all ground instances of that clause, a tautology follows from smaller ground instances of clauses in $S$, and therefore the clause is redundant wrt. $S$, a fact that we have been able to show merely by showing that all ground instances of that clause are rewritable to a tautology via ground instances of equations in $S$. We just had to ensure that the ground instances of the equations used in the rewrites were all smaller than the ground instance of the clause being rewritten.

This is an example that motivates the formulation of a ground joinability rule, analogous to the joinability rule in (4.4).

In the context of equational completion, it can be shown that ground joinable equations are *not* needed for computing critical pairs (but they may still be retained for rewriting), in that discarding those does not compromise the

[3]Note that $f(x, y) \approx f(y, x)$ cannot be oriented by *any* simplification ordering.

[4]A rule that states that $C\theta \lor D$ is redundant wrt. $\{C\}$.

refutational completeness of the procedure [Martin and Nipkow 1990]. Empirically, this appears to be an important factor for equational completion provers routinely outperforming superposition provers, often by a wide margin, in unit-equational problems [Sutcliffe 2016], and it is unsatisfactory that this technique is unavailable in superposition. In this context, the equivalent notion would be ground joinable equations being redundant in the set of clauses wrt. which they are ground joinable.

In the sequel, we will show how a similar rule to (4.4) still holds if we replace the requirement of joinability by a requirement of ground joinability. Let us write the following definition.

**Definition 4.3** (Strong ground joinability)**.** Two terms are **strongly ground joinable** ($s \mathrel{\rotatebox[origin=c]{-90}{$\leftrightsquigarrow$}\!\!\rotatebox[origin=c]{-90}{$\leftrightsquigarrow$}} t$), **in** a clause $C$ **wrt.** a set of equations $S$, if for all $\theta \in \mathrm{GSubs}(s,t)$ we have $s\theta \mathrel{\rotatebox[origin=c]{-90}{$\rightsquigarrow$}} t\theta$ in $C$ wrt. $S$.

Note that strong joinability implies strong ground joinability, but not the converse.

We then have:

$$\text{Ground joinability} \qquad \frac{\cancel{s \approx t \vee C} \quad S}{\phantom{C}}, \qquad \begin{array}{l}\text{where } s \mathrel{\rotatebox[origin=c]{-90}{$\leftrightsquigarrow$}\!\!\rotatebox[origin=c]{-90}{$\leftrightsquigarrow$}} t \\ \text{in } s \approx t \vee C \text{ wrt. } S,\end{array} \tag{4.10a}$$

$$\text{Ground joinability} \qquad \frac{\cancel{s \not\approx t \vee C} \quad S}{C}, \qquad \begin{array}{l}\text{where } s \mathrel{\rotatebox[origin=c]{-90}{$\leftrightsquigarrow$}\!\!\rotatebox[origin=c]{-90}{$\leftrightsquigarrow$}} t \\ \text{in } s \not\approx t \vee C \text{ wrt. } S.\end{array} \tag{4.10b}$$

In other words, rather than demanding a joinability "proof" for the (possibly non-ground) literal, we relax that to a requirement of only a *separate* joinability "proof" for each of the ground instances of the literal, using smaller (wrt. $\succ_{cc}$) instances of equations in $S$.

**Theorem 4.3.** Ground joinability is a simplification rule of the superposition calculus wrt. closure redundancy.

*Proof.* We will prove this for (4.10a), the positive case, first. If $s \mathrel{\rotatebox[origin=c]{-90}{$\leftrightsquigarrow$}\!\!\rotatebox[origin=c]{-90}{$\leftrightsquigarrow$}} t$, then for any instance $(s \approx t \vee C) \cdot \theta$ we either have $s\theta = t\theta$, and therefore $\emptyset \models (s \approx t) \cdot \theta$, or we have wlog. $s\theta \succ_t t\theta$, with $s\theta \downarrow t\theta$. Then $s\theta$ and $t\theta$ can be rewritten to the same normal form $u$ by $l_i\sigma_i \to r_i\sigma_i$ where $l_i \approx r_i \in S$. Since $u \prec_t s\theta$ and $u \preceq_t t\theta$, then $(s \approx t \vee C) \cdot \theta$ follows from smaller $(u \approx u \vee C) \cdot \theta$[5] (a tautology, i.e. follows from

[5]Wlog. $u\theta = u$, renaming variables in $u$ if necessary.

$\emptyset$) and from the instances of clauses in $S$ used to rewrite $s\theta \to u \leftarrow t\theta$. It only remains to show that these latter instances are also smaller than $(s \approx t \vee C) \cdot \theta$. Since we have assumed $s\theta \succ_t t\theta$, then at least one rewrite step must be done on $s\theta$. Let $l_1\sigma_1 \to r_1\sigma_1$ be the instance of the rule used for that step, with $(l_1 \approx r_1) \cdot \sigma_1$ the closure that generates it. By Definitions 4.1, 4.2 and 4.3, one of the following holds:

- $C \neq \emptyset$, therefore $(l_1 \approx r_1) \cdot \sigma_1 \prec_{cc} (s \approx t \vee C) \cdot \theta$ (Lemma 3.7), or
- $l_1\sigma_1 \lhd s\theta$, therefore $l_1\sigma_1 \prec_t s\theta \Rightarrow l_1 \cdot \sigma_1 \prec_{tc} s \cdot \theta \Rightarrow (l_1 \approx r_1) \cdot \sigma_1 \prec_{cc} (s \approx t \vee C) \cdot \theta$, or
- $l_1\sigma_1 = s\theta$ and $s \sqsupset l_1$, therefore $l_1 \cdot \sigma_1 \prec_{tc} s \cdot \theta \Rightarrow (l_1 \approx r_1) \cdot \sigma_1 \prec_{cc} (s \approx t \vee C) \cdot \theta$, or
- $l_1\sigma_1 = s\theta$ and $s \equiv l_1$ and $r_1\sigma_1 \prec_t t\theta$, therefore $r_1 \cdot \sigma_1 \prec_{tc} t \cdot \theta \Rightarrow (l_1 \approx r_1) \cdot \sigma_1 \prec_{cc} (s \approx t \vee C) \cdot \theta$.

As for the remaining steps, they are done on the smaller side $t\theta$ or on the other side after this first rewrite, which is smaller than $s\theta$. Therefore all subsequent steps done by any $l_j\sigma_j \to r_j\sigma_j$ will have

$$r_j \cdot \sigma_j \prec_{tc} l_j \cdot \sigma_j \prec_{tc} s \cdot \theta \Rightarrow (l_j \approx r_j) \cdot \sigma_j \prec_{cc} (s \approx t \vee C) \cdot \theta\,. \tag{4.11}$$

In other words, we have shown that closure $(s \approx t \vee C) \cdot \theta$ follows from smaller $(l_i \approx r_i) \cdot \sigma_i \in \mathrm{GClos}(S)$ and smaller $(u \approx u \vee C) \cdot \theta$. The latter, of course, is a tautology and redundant in any context. As such, since this holds for all ground closures $(s \approx t \vee C) \cdot \theta$, then $s \approx t \vee C$ is (closure) redundant wrt. $S$.

For (4.10b), the negative case, the proof is similar: if $s \between t$, then for any instance $(s \not\approx t \vee C) \cdot \theta$ we either have $s\theta = t\theta$, or wlog. $s\theta \succ_t t\theta$ with $s\theta \downarrow t\theta$. If the latter is true, then $s\theta$ and $t\theta$ can be rewritten to the same normal form $u$ by $l_i\sigma_i \to r_i\sigma_i$ where $l_i \approx r_i \in S$. Again, since $u \prec_t s\theta$ and $u \preceq_t t\theta$, then $(s \not\approx t \vee C) \cdot \theta$ follows from smaller $(u \not\approx u \vee C) \cdot \theta$ and from the instances of clauses in $S$ used to rewrite $s\theta \to u \leftarrow t\theta$. By the same case analysis as above, these instances $(l_i \approx r_i) \cdot \sigma_i \in \mathrm{GClos}(S)$ are smaller than $(s \not\approx t \vee C) \cdot \theta$.

As such, we will conclude that $(s \not\approx t \vee C) \cdot \theta$ follows from smaller $(l_i \approx r_i) \cdot \sigma_i \in \mathrm{GClos}(S)$ and smaller $(u \not\approx u \vee C) \cdot \theta$. But now the latter simply follows from the smaller closure $C \cdot \theta$, and therefore (by monotonicity of the redundancy relation, Lemma 3.9) $s \not\approx t \vee C$ is (closure) redundant wrt. $S \cup \{C\}$. □

Note that using closure redundancy is essential for the proof, as ground joinability would *not* be correct using the standard redundancy criterion, as evidenced e.g. by Example 3.3.

In equational completion, the ground joinability inference rule following Avenhaus, Hillenbrand and Löchner [2003] is exactly (using our notation)

$$\frac{\cancel{s \approx t} \quad S}{}, \qquad \text{where } s \Downarrow t \text{ in } s \approx t \text{ wrt. } S \tag{4.12}$$

which is simply a special case of (4.10a).[6]

Finally, this also raises the question of how to actually test ground joinability in practice. Indeed, every non-ground formula has an infinite number of ground instances, whenever the signature contains at least one non-constant function symbol.[7] This means that a practical test with any hope of even terminating in the general case, let alone being efficient for use in a simplification rule for theorem proving, will have to employ an efficient algorithm, which potentially only *approximates* the $\Downarrow$ relation.[8] We will explore this in detail in Section 5.1.

## Connectedness

The simplification rules for joinability and ground joinability require that each step in proving $\downarrow$ / $\Downarrow$ is done via an oriented instance of an equation in the set. However, this may be unecessarily restrictive. In this section, we show that we can formulate a criterion for the redundancy of *inferences*, rather than clauses, which uses a notion of *connectedness* which is weaker than the notion of joinability.

Informally, in a joinability proof we have e.g.

$$s \to u_1 \to \cdots \to u_k \leftarrow \cdots \leftarrow u_n \leftarrow t \tag{4.13}$$

where $\to \subseteq \succ_t$, therefore all the $u_i$ are smaller wrt. $\succ_t$ than $s$ or $t$. Together with the extra conditions in Definition 4.2, this is what ensures that the criterion for redundant clause (Definition 3.8) is satisfied: all instances of the original clause follow from smaller instances of the equations used in the rewrite plus the resulting clause.

As a generalisation, we may have a *connectedness* proof e.g.

$$s \leftrightarrow u_1 \leftrightarrow \cdots \leftrightarrow u_i \leftrightarrow \cdots \leftrightarrow u_n \leftrightarrow t \tag{4.14}$$

[6]And at the same time a sufficient criterion for both (4.10a) and (4.10b), as both easily follow from (4.12).

[7]That is, the signature is not in the EPR/Bernays-Schönfinkel class.

[8]Of course, $\Downarrow$ is undecidable in the general case.

where each step may decrease *or increase* the term (wrt. $\succ_t$, or even go to an incomparable term), but all the $u_i$ stay smaller than terms in a certain set.

In particular, in the context of equational completion one may write **critical pair criteria** [Bachmair and Dershowitz 1988; Bonacina and Hsiang 1995], i.e. sets of critical pairs $s \leftarrow u \rightarrow t$ such that not adding equation $s \approx t$ to the saturation set yields the same rewrite system in the limit (in other words, critical pairs in critical pair criteria may be ignored by a completion procedure without compromising refutational completeness). The analogue of this notion, in the context of the superposition calculus, would be that of a redundant superposition (2.7a) inference.

One of the more useful families of critical pair criteria in equational completion is precisely connectedness-related criteria: in equational completion, a critical pair $s \leftarrow u \rightarrow t$ where (4.14) holds with all $u_i \prec_t u$ can be ignored in saturation [Bachmair and Dershowitz 1988; Smallbone 2021].

Here, we introduce a connectedness-based criterion for superposition which (i) is the first time such a criterion was formulated for a full clausal first-order logic calculus, and also (ii) slightly generalises the criterion in Bachmair and Dershowitz [1988] even in the unit-equality case.

Let us make our previous intuition rigorous, by giving the following definition.

**Definition 4.4** (Connectedness)**.** Terms $s$ and $t$ are **connected** under clauses $N$ and unifier $\rho$ wrt. a set of equations $S$ if there exist terms $v_1, \ldots, v_n$, equations $l_1 \approx r_1, \ldots, l_{n-1} \approx r_{n-1}$, and substitutions $\sigma_1, \ldots, \sigma_{n-1}$ such that:

(i) $v_1 = s$ and $v_n = t$, and
(ii) for all $i \in 1, \ldots, n-1$, either $v_{i+1} = v_i[l_i\sigma_i \mapsto r_i\sigma_i]$ or $v_i = v_{i+1}[l_i\sigma_i \mapsto r_i\sigma_i]$, with $l_i \approx r_i \in S$, and
(iii) for all $i \in 1, \ldots, n-1$ and for all $u_i \in \{l_i, r_i\}$, there exists $C \in N$ and $w \in \bigcup_{p \dot{\approx} q \in C} \{p, q\}$[9] such that either (a) $u_i\sigma_i \prec_t w\rho$, or (b) $u_i\sigma_i = w\rho$ and either $u_i \sqsubset w$ or $C$ is not a positive unit.

*Example* 4.3. Consider the following set of clauses (a real example from a prob-

[9] That is, in the set of top-level terms of literals of $C$.

lem in Boolean algebras).

$$S = \begin{cases} \dots \\ x \cdot ((x+y) \cdot z) \approx x \cdot z \\ x + (y \cdot (x \cdot z)) \approx x \\ x + (x \cdot y) \approx x \\ \dots \end{cases} \tag{4.15}$$

Assuming $\cdot \succ_t +$, the following superposition inference (with the intermediate instances after unification written out for clarity)

$$\cfrac{\cfrac{\cfrac{x_1 \cdot ((x_1+y_1) \cdot z_1) \approx x_1 \cdot z_1}{x \cdot ((x+y) \cdot z) \approx x \cdot z} \qquad \cfrac{x_2 + (y_2 \cdot (x_2 \cdot z_2)) \approx x_2}{(x+y) + (x \cdot ((x+y) \cdot z)) \approx (x+y)}}{(x+y) + (x \cdot z) \approx (x+y)}}{x + (y + (x \cdot z)) \approx x + y} \tag{4.16}$$

has $x + (y + (x \cdot z))$ and $x + y$ connected under the premises wrt. $S$, because

$$\begin{aligned} & x + (y + (x \cdot z)) && && (4.17a) \\ \leftrightarrow\ & y + (x + (x \cdot z)) && \text{by } x + (y + (x \cdot z)) \approx y + (x + (x \cdot z)) && (4.17b) \\ \leftrightarrow\ & y + x && \text{by } x + (x \cdot z) \approx x && (4.17c) \\ \leftrightarrow\ & x + y && \text{by } x + y \approx y + x && (4.17d) \end{aligned}$$

with the intermediate steps obeying the conditions in Definition 4.4(ii–iii), under clauses $N = \{x_1 \cdot ((x_1+y_1) \cdot z_1) \approx x_1 \cdot z_1,\ x_2 + (y_2 \cdot (x_2 \cdot z_2)) \approx x_2\}$ and unifier $\rho = x_1 \mapsto x, y_1 \mapsto y, z_1 \mapsto z, x_2 \mapsto x + y, y_2 \mapsto x, z_2 \mapsto z$.

As criteria for redundancy of a clause, finding either joinability or ground joinability of a literal in the clause means that the clause can be deleted or the literal removed from the clause (in case of a positive or negative literal, resp.) in any context, that is, we can for example add them to a set of deleted clauses, and for any new clause, if it appears in that set, then immediately remove it since we already saw that it is redundant.

The criterion of connectedness which we will present, however, is a criterion for redundancy of *inferences*. This means that a conclusion simplified by this criterion can be deleted (or rather, not added), but in that context only; if it ever comes up again as a conclusion of a different inference, then it is not necessarily also redundant. This is important when it comes to efficient implementation, and may pose issues depending on the architecture of the saturation loop and underlying data structures (see Section 5.2).

The statement of the connectedness criterion in superposition is as follows.

**Theorem 4.4.** Superposition inferences of the form

$$\frac{\underline{l \approx r} \vee C \quad \underline{s[u] \approx t} \vee D}{(s[u \mapsto r] \approx t \vee C \vee D)\rho}, \quad \begin{array}{l}\text{where } \rho = \mathrm{mgu}(l, u),\\ l\rho \not\preceq_t r\rho,\, s\rho \not\preceq_t t\rho,\\ \text{and } u \text{ not a variable,}\end{array} \tag{4.18}$$

where $s[u \mapsto r]\rho$ and $t\rho$ are connected under $\{l \approx r \vee C,\ s \approx t \vee D\}$ and unifier $\rho$ wrt. some set of clauses $S$, are redundant inferences wrt. $S$.

*Proof.* Let us denote $s' = s[u \mapsto r]$. Let also $U = \{l \approx r \vee C,\ s \approx t \vee D\}$ and $M = \bigcup_{C \in U} \bigcup_{p \dot{\approx} q \in C} \{p, q\}$. We will show that if $s'\rho$ and $t\rho$ are connected under $U$ and $\rho$, wrt. in $S$, then every instance of that inference obeys the condition for closure redundancy of an inference (Definition 3.9), wrt. $S$.

Consider any $(s' \approx t \vee C \vee D)\rho \cdot \theta$ where $\theta \in \mathrm{GSubs}(U\rho)$. Either $s'\rho\theta = t\rho\theta$, and we are done (it follows from $\emptyset$), or $s'\rho\theta \succ_t t\rho\theta$, or $s'\rho\theta \prec_t t\rho\theta$.

Consider the case $s'\rho\theta \succ_t t\rho\theta$. For all $i \in 1, \dots, n-1$, there exists a $C' \in U$ and a $w \in C'$ such that either

- $l_i\sigma_i\theta \prec_t w\rho\theta$ (iii.a), or
- $l_i\sigma_i\theta = w\rho\theta$ and $l_i \sqsubset v$ (iii.b), or
- $l_i\sigma_i\theta = w\rho\theta$ and $C'$ is not a positive unit (iii.b).

Likewise for $r_i$.

Therefore, for all $i \in 1, \dots, n-1$, there exists a $C' \in U$ such that

$$(l_i \approx r_i) \cdot \sigma_i\theta \prec_{cc} C' \cdot \rho\theta\,. \tag{4.19}$$

Since $(t \approx t \vee \cdots)\rho \cdot \theta$ is also smaller than $(s' \approx t \vee \cdots)\rho \cdot \theta$ and a tautology, the instance $(s' \approx t \vee \cdots)\rho \cdot \theta$ of the conclusion follows from closures in $\mathrm{GClos}(S)$ such that each is smaller than one of $(l \approx r \vee C) \cdot \rho\theta$, $(s \approx t \vee D) \cdot \rho\theta$.

In the case that $s'\rho\theta \prec_t t\rho\theta$, the same idea applies, only now it is instead $(s' \approx s' \vee \cdots)\rho \cdot \theta$ which is smaller than $(s' \approx t \vee \cdots)\rho \cdot \theta$ and is a tautology.

Therefore, we have shown that for all $\theta \in \mathrm{GSubs}((l \approx r \vee C)\rho,\ (s \approx t \vee D)\rho)$, the instance $(s' \approx t \vee C \vee D)\rho \cdot \theta$ of the conclusion follows from closures in $\mathrm{GClos}(S)$ which are all smaller than one of $(l \approx r \vee C) \cdot \rho\theta$ or $(s \approx t \vee D) \cdot \rho\theta$. Since any valid superposition inference with ground clauses has to have $l = u$, then any $\theta' \in \mathrm{GSubs}(l \approx r \vee C,\ s \approx t \vee D,\ (s' \approx t \vee C \vee D)\rho)$ such that the inference

$$(l \approx r \vee C)\theta'\,,\ (s \approx t \vee D)\theta' \vdash (s' \approx t \vee C \vee D)\rho\theta' \tag{4.20}$$

is valid must have $\theta' = \rho\theta''$, since $\rho$ is the most general unifier. Therefore, we have shown that for all $\theta' \in \mathrm{GSubs}(l \approx r \vee C,\ s \approx t \vee D,\ (s' \approx t \vee C \vee D)\rho)$ for which (4.20) is a valid superposition inference, the instance $(s' \approx t \vee C \vee D)\rho \cdot \theta'$ of the conclusion follows from closures in $\mathrm{GClos}(S)$ which are all smaller than one of $(l \approx r \vee C) \cdot \theta',\ (s \approx t \vee D) \cdot \theta'$, so the inference is redundant. □

**Theorem 4.5.** Superposition inferences of the form

$$\frac{\underline{l \approx r} \vee C \quad \underline{s[u] \not\approx t} \vee D}{(s[u \mapsto r] \not\approx t \vee C \vee D)\rho}, \quad \begin{array}{l}\text{where } \rho = \mathrm{mgu}(l, u),\\ l\rho \not\preceq_t r\rho,\, s\rho \not\preceq_t t\rho,\\ \text{and } u \text{ not a variable,}\end{array} \tag{4.21}$$

where $s[u \mapsto r]\rho$ and $t\rho$ are connected under $\{l \approx r \vee C,\ s \not\approx t \vee D\}$ and unifier $\rho$ wrt. some set of clauses $S$, are redundant inferences wrt. $S \cup \{(C \vee D)\rho\}$.

*Proof.* Analogously to the previous proof, we find that for all instances of the inference, the closure $(s' \not\approx t \vee \cdots)\rho \cdot \theta$ follows from smaller closure $(t \not\approx t \vee \cdots)\rho \cdot \theta$ or $(s' \not\approx s' \vee \cdots)\rho \cdot \theta$ and closures $(l_i \approx r_i) \cdot \sigma_i\theta$ smaller than $\max\{(l \approx r \vee C) \cdot \theta,\ (s \not\approx t \vee D) \cdot \theta,\ (s' \not\approx t \vee C \vee D)\rho \cdot \theta\}$. But $(t \not\approx t \vee C \vee D)\rho \cdot \theta$ and $(s' \not\approx s' \vee C \vee D)\rho \cdot \theta$ both follow from smaller $(C \vee D)\rho \cdot \theta$, therefore the inference is redundant wrt. $S \cup \{(C \vee D)\rho\}$. □

## Ground connectedness

Just as joinability can be generalised to ground joinability, so can connectedness be generalised to ground connectedness.

**Definition 4.5** (Ground connectedness)**.** Terms $s, t$ are **ground connected** under $U$ and $\rho$ wrt. $S$ if, for all $\theta \in \mathrm{GSubs}(s, t)$, $s\theta$ and $t\theta$ are connected under $D$ and $\rho$ wrt. $S$.

**Theorem 4.6.** Superposition inferences of the form

$$\frac{\underline{l \approx r} \vee C \quad \underline{s[u] \approx t} \vee D}{(s[u \mapsto r] \approx t \vee C \vee D)\rho}, \quad \begin{array}{l}\text{where } \rho = \mathrm{mgu}(l, u),\\ l\rho \not\preceq_t r\rho,\, s\rho \not\preceq_t t\rho,\\ \text{and } u \text{ not a variable,}\end{array} \tag{4.22}$$

where $s[u \mapsto r]\rho$ and $t\rho$ are ground connected under $\{l \approx r \vee C,\ s \approx t \vee D\}$ and unifier $\rho$ wrt. some set of clauses $S$, are redundant inferences wrt. $S$.

**Theorem 4.7.** Superposition inferences of the form

$$\frac{\underline{l \approx r} \vee C \quad \underline{s[u] \not\approx t} \vee D}{(s[u \mapsto r] \not\approx t \vee C \vee D)\rho}, \quad \begin{array}{l}\text{where } \rho = \mathrm{mgu}(l, u),\\ l\rho \not\preceq_t r\rho,\, s\rho \not\preceq_t t\rho,\\ \text{and } u \text{ not a variable,}\end{array} \tag{4.23}$$

where $s[u \mapsto r]\rho$ and $t\rho$ are ground connected under $\{l \approx r \vee C,\ s \not\approx t \vee D\}$ and unifier $\rho$ wrt. some set of clauses $S$, are redundant inferences wrt. $S \cup \{(C \vee D)\rho\}$.

*Proof.* The proofs of Theorems 4.6 and 4.7 is analogous to those of Theorems 4.4 and 4.5. The weakening of connectedness to ground connectedness only means that the proof of connectedness (e.g. the $v_i$, $l_i \approx r_i$, $\sigma_i$) may be different for different ground instances . For all the steps in the proof to hold we only need that for all the instances $\theta \in \mathrm{GSubs}(l \approx r \vee C,\ s \dot{\approx} t \vee D,\ (s[u \mapsto r] \dot{\approx} t \vee C \vee D)\rho)$ of the inference, $\theta = \sigma\theta'$ with $\sigma \in \mathrm{GSubs}(s[u \mapsto r]\rho, t\rho)$, which is true. □

## 4.3 Associativity and commutativity

Associative-commutative symbols (AC) are a ubiquitous feature of problems in a wide variety of domains. It is the theory axiomatised by the following equations:

$$f(x, y) \approx f(y, x)\,, \tag{4.24a}$$

$$f(x, f(y, z)) \approx f(f(x, y), z)\,. \tag{4.24b}$$

AC shows up contained in the axioms of several important algebraic structures in mathematics, in common-sense reasoning, and in software verification, to give some examples [Sutcliffe 2017]; this makes it a particularly interesting theory to study (as noted in the introduction, AC has long been an object of active research in automated reasoning).

Unfortunately, it poses particular problems to superposition-based reasoning.

**Theorem 4.8.** The superposition-saturated set of (4.24) is infinite, and the number of consequences with $n$ occurrences of $f$ is $n!\big(\frac{(2n-2)!}{(n-1)!n!}\big)^2 \sim \mathcal{O}(e^n)$.[10]

[10]The Catalan number $C_n = \frac{(2n)!}{(n+1)!n!}$ is the number of different ways that balanced parentheses can be placed to group pairs of terms in an expression with $n+1$ terms [Sloane 2022a], and $n! \sim \sqrt{2\pi n}(n/e)^n$ is an assymptotic expression for $n!$ (Stirling's formula).

Indeed, this combinatorial explosion is enough to make even apparently simple problems impossibly difficult to solve, as without special treatment this deluge of new clauses will very quickly overwhelm the search space, drown out other useful inferences, and generally grind the prover to a halt.

*Example* 4.4. State-of-the-art superposition provers which do not incorporate any AC-specialised reasoning have a hard time proving even the following very simple conjecture.

$$\begin{aligned} x + y &\approx y + x \qquad & x + (y + z) &\approx (x + y) + z \\ x + (-x) &\approx 0 \qquad & x + 0 &\approx x \\ \Rightarrow a_1 + (a_2 + (\cdots + (a_n &+ (-a_1)))) \approx a_2 + (\cdots + a_n) \end{aligned} \tag{4.25}$$

For example in Vampire 4.6, a top-performing superposition prover,[11] we measured the time taken to prove this problem. It appears to be exponential in $n$, with e.g. the $n = 3$ problem taking 1.8 s after 18 212 generated clauses but $n = 4$ taking 43 s and 246 049 generated clauses. This is illustrative of the problem facing superposition provers when dealing with AC axioms and other permutative theories. Note also the extreme dependency on the specific term ordering: if $a_1$ were the maximal element in $\{a_i\}$, any size instance of this problem could be trivially solved with encompassment demodulation.

However, ground joinability is enough to conclude for instance that *all but one* of the consequences of (4.24) are redundant. Since AC is such an important and frequently-occurring theory, it is relevant to investigate simplification rules for AC in particular, in addition to the results presented in the preceding sections of this chapter, which are fully general and applicable for any equational theory. It is also relevant to specialise algorithms for general simplification rules — such as ground joinability — to the AC case.

Let us first investigate the application of ground joinability to the AC axioms and their consequences.

[11] Using the default $\succ_t$, which is a Knuth-Bendix order with $a_1 \prec_t a_2 \prec_t \cdots$.

## AC joinability

Let $AC_f$ be

$$f(x,y) \approx f(y,x)\,, \tag{4.26a}$$
$$f(x,f(y,z)) \approx f(f(x,y),z)\,, \tag{4.26b}$$
$$f(x,f(y,z)) \approx f(y,f(x,z))\,. \tag{4.26c}$$

The first two (4.26a–b) are the AC axioms for $f$. The third equation (4.26c) follows from those two and will be used to avoid any inferences between these axioms, and more generally to justify the AC joinability simplifications defined next.

We define the two following rules:

$$\text{AC joinability} \qquad \frac{\cancel{s \approx t \lor C} \quad AC_f}{}\,, \quad \text{where } s =_{AC_f} t\text{, and } (s \approx t \lor C) \notin AC_f, \tag{4.27a}$$

$$\text{AC joinability} \qquad \frac{\cancel{s \not\approx t \lor C} \quad AC_f}{C}\,, \quad \text{where } s =_{AC_f} t. \tag{4.27b}$$

Contrast (4.27) and (4.10). $=_{AC_f}$ is of course much easier to check than $\between$; because there are no completeness side-conditions to check or keep track of, it can be implemented by simply collecting and sorting $f$-subterms (according to any arbitrary total order, unrelated to $\succ_t$) and comparing for equality.[12]

**Theorem 4.9.** AC joinability is a simplification rule of the superposition calculus wrt. closure redundancy.

*Proof.* Having proven Theorem 4.3, it suffices to show that '$s =_{AC_f} t$' implies '$s \between t$ in $s \dot{\approx} t \lor C$ wrt. $AC_f$', except in the case that $s \dot{\approx} t \lor C$ is itself one of the three members of $AC_f$. However, $AC_f$ is ground confluent [Martin and Nipkow 1990], therefore $s =_{AC_f} t$ implies $s\theta \downarrow_{AC_f} t\theta$ for all $\theta \in \mathrm{GSubs}(s,t)$. All that is left is ensuring the completeness conditions in Definitions 4.2 and 4.3.

Let $(s \dot{\approx} t \lor C) \cdot \theta$ be an arbitrary ground instance of $s \dot{\approx} t \lor C$. If $s\theta \downarrow_{AC_f} t\theta$ then there is a $u$ such that: $s\theta = u$ or there is a chain $s\theta \to \cdots \to u$, and $t\theta = u$ or there is a chain $t\theta \to \cdots \to u$, such that each rewrite step is made via an equation in $AC_f$.

[12]The algorithm for $\between$ is much more involved, as discussed in Section 5.1.

Wlog., consider the first rewrite on $s\theta$,[13] and let $l \approx r$ be the equation in $AC_f$ used (i.e. such that $s\theta[l\sigma\theta]$). Let also $\mathrm{subterms}_f$ be a function that collects all "consecutive" $f$-subterms into a multiset, that is

$$\mathrm{subterms}_f(u) = \begin{cases} \mathrm{subterms}_f(s) \cup \mathrm{subterms}_f(t) & \text{if } u = f(s,t) \\ u & \text{otherwise} \end{cases} \tag{4.28a}$$

so for example $\mathrm{subterms}_f(f(a, f(f(b,c),d))) = \{a,b,c,d\}$.

The following cases arise:

- If $|\,\mathrm{subterms}_f(s)| \geq 4$, then $l\sigma\theta \lhd s\theta$ or $\sigma$ cannot be a renaming. This is evident because all terms $l$ occurring in $AC_f$ have $2 \leq |\,\mathrm{subterms}_f(l)| \leq 3$, where $\mathrm{subterms}_f(l)$ contains only variables, so it is impossible to have an injection $\sigma$ from variables in $l$ onto variables in $s$ while still having $s = l\sigma$. Either way, the rewrite is admissible (Definition 4.1), and the constraints for strong joinability are satisfied (Definition 4.2).

- If $|\,\mathrm{subterms}_f(s)| \leq 3$, then if at least one $l \in \mathrm{subterms}_f(s)$ is not a variable, then $\sigma$ also cannot be a renaming, and the rewrite is also admissible.

- If $|\,\mathrm{subterms}_f(s)| \leq 3$ and all $l \in \mathrm{subterms}_f(s)$ are variables, ad-hoc proofs are needed. Fortunately, there are not many cases (modulo renaming), so exhaustively checking all possibilities can be easily done:

For conciseness, let us denote $f(a,b)$ by $ab$ in the sequel. We will assume that the term ordering has following properties: if $s \succ_t t$ then $st \succ_t ts$ and $s(tu) \succ_t t(su)$, and also that $(xy)z \succ_t x(yz)$. These conditions hold for most commonly used families of orderings, such as KBO or LPO [Baader and Nipkow 1998].[14]

| | | |
|---|---|---|
| $x \approx x$ | Tautology | (4.29a) |
| $xy \approx xy$ | Tautology | (4.29b) |
| $xy \approx yx$ | Instance of (4.26a) | (4.29c) |

[13] Literals are equal modulo flipping the equality sides, so the whole proof carries unchanged to the $t$ side.

[14] But not for all orderings; for example a KBO or LPO variant where the "lexicographical" step is backwards, starting from the right-most argument. However, it is mechanically trivial to modify the proof where the converse of any of those relations holds, so we will proceed by assuming those conditions.

| | | |
|---|---|---|
| $x(yz) \approx x(yz)$ | Tautology | (4.29d) |
| $x(yz) \approx y(zx)$ | If $x\theta \prec z\theta$, rewrite $zx \to xz$ to get an instance of (4.26c). If $z\theta \prec x\theta$ and $z\theta \prec y\theta$, rewrite $y(zx) \to z(xy)$ to get an instance of (4.29i ). If $y\theta \prec z\theta \prec x\theta$, rewrite $x(yz) \to z(yx)$ – using smaller (4.29i ) – to get an instance of (4.26c). | (4.29e) |
| $x(yz) \approx z(xy)$ | If $y\theta \prec x\theta$, rewrite $xy \to yx$ to get an instance of (4.29i ). If $z\theta \prec y\theta$, rewrite $yz \to zy$ to get an instance of (4.29i ). If $x\theta \prec y\theta \prec z\theta$, rewrite $z(xy) \to y(xz)$ – using smaller (4.29i ) – to get an instance of (4.26c). | (4.29f ) |
| $x(yz) \approx y(xz)$ | Instance of (4.26c) | (4.29g) |
| $x(yz) \approx x(zy)$ | If $y\theta \prec z\theta$, rewrite $zy \to yz$. If $z\theta \prec y\theta$, rewrite $yz \to zy$. In both cases, we reach a tautology. | (4.29h) |
| $x(yz) \approx z(yx)$ | If $z\theta \prec y\theta$ and $x\theta \prec y\theta$, rewrite $yz \to zy$ and $yx \to xy$ to get an instance of (4.26c). If $y\theta \prec z\theta$ and $y\theta \prec x\theta$, rewrite $z(yx) \to y(zx)$ to get an instance of (4.29i ). If $x\theta \prec y\theta \prec z\theta$, rewrite on the right: $yx \to xy$, then $z(xy) \to x(zy)$, then $zy \to yz$ to obtain a tautology. If $z\theta \prec y\theta \prec x\theta$, rewrite on the left: $yz \to zy$, then $x(zy) \to z(xy)$, then $xy \to yx$ to obtain a tautology. | (4.29i ) |
| $(xy)z \approx x(yz)$ | Instance of (4.26b) | (4.29j ) |
| $(xy)z \approx y(zx)$ | If $x\theta \prec y\theta$, rewrite $(xy)z \to x(yz)$ to get an instance of (4.29e). If $y\theta \prec x\theta$, rewrite $xy \to yx$ to get $(yx)z \approx y(zx)$, rewrite $(yx)z \to y(xz)$ to get an instance of (4.29h). | (4.29k) |
| $(xy)z \approx z(xy)$ | Instance of (4.26a) | (4.29l ) |
| $(xy)z \approx y(xz)$ | If $x\theta \prec y\theta$, rewrite $y(xz) \to x(yz)$. If $y\theta \prec x\theta$, rewrite $xy \to yx$. In both cases, we reach an instance of (4.26b). | (4.29m) |

| | | |
|---|---|---|
| $(xy)z \approx x(zy)$ | If $y\theta \prec z\theta$, rewrite $zy \to yz$ to get an instance of (4.26b). If $z\theta \prec y\theta$, rewrite $(xy)z \to z(xy)$ (via a proper instance of $xy \approx yx$, that is) to get an instance of (4.26c). | (4.29n) |
| $(xy)z \approx z(yx)$ | If $x\theta \prec y\theta$, rewrite $yx \to xy$. If $y\theta \prec x\theta$, rewrite $xy \to yx$. In both cases, we reach an instance of (4.26a). | (4.29o) |

This proves all the cases.

Recapitulating, we have shown that if $(s \dot{\approx} t \vee C) \notin AC_f$, then $s =_{AC_f} t \Rightarrow s \between t$ in $s \dot{\approx} t \vee C$ wrt. $AC_f$. Therefore by the proof of Theorem 4.3, AC joinability is also a valid simplification rule. □

This immediately implies the following simple but very powerful corollary.

**Corollary 4.10.** All the conclusions of superposition inferences among clauses in $AC_f$ are redundant wrt. $AC_f$.

*Proof.* Applying the superposition rule on (4.26) yields the following set of consequences (excluding tautologies):

$$f(x, f(y, z)) \approx f(z, f(x, y)) \quad \text{by 4.26b,a,} \tag{4.30a}$$
$$f(x, f(y, z)) \approx f(y, f(z, x)) \quad \text{by 4.26c,a,} \tag{4.30b}$$
$$f(x, f(y, z)) \approx f(f(y, x), z) \quad \text{by 4.26b,a,} \tag{4.30c}$$
$$f(x, f(y, z)) \approx f(f(x, z), y) \quad \text{by 4.26c,a,} \tag{4.30d}$$
$$f(x, f(y, f(z, w))) \approx f(f(y, z), f(x, w)) \quad \text{by 4.26c,b,} \tag{4.30e}$$
$$f(x, f(y, f(z, w))) \approx f(y, f(z, f(x, w))) \quad \text{by 4.26c,} \tag{4.30f}$$
$$f(x, f(f(y, z), w)) \approx f(y, f(z, f(x, w))) \quad \text{by 4.26c,b,} \tag{4.30g}$$
$$f(x, f(f(y, z), w)) \approx f(f(y, f(x, z)), w) \quad \text{by 4.26b,c,} \tag{4.30h}$$
$$f(f(x, y), f(z, w)) \approx f(f(x, f(y, z)), w) \quad \text{by 4.26b,b;} \tag{4.30i}$$

each of these is always redundant wrt. (4.26) by application of rule (4.27a). □

## AC normalisation

We will now show some examples to motivate another simplification rule.

*Example* 4.5. Assume $a \prec_t b \prec_t c$. The encompassment demodulation rule already enables us to rewrite any occurrence of, for instance, $b(ca)$, or $(ac)b$, or any other such permutation, to $a(bc)$, This is independent of where these occur, be it at a subterm or top position, on a maximal or minimal side of a positive or negative literal, and in a unit or non-unit clause.

However, take the term $b(xa)$. It cannot be simplified by demodulation (via any of the $AC_f$). Yet it is easy to see that in any instance of a clause where it appears, it can be rewritten to a smaller $a(xb)$ via smaller instances of clauses in $AC_f$.

Such cases motivate the following simplification rule.[15]

$$\text{AC norm.} \quad \frac{\cancel{C[t_1(\cdots t_n)]} \quad AC_f}{C[t'_1(\cdots t'_n)]}, \quad \begin{array}{l}\text{where } t_1,\dots,t_n \succ_{\text{lex}} t'_1,\dots,t'_n \\ \text{and } \{t_1,\dots,t_n\} = \{t'_1,\dots,t'_n\}^{16}\end{array} \tag{4.31}$$

Some examples (assume $(\dots)^{-1} \succ_t b \succ_t a$):

$$b(xa) \to a(xb) \tag{4.32a}$$
$$b(xa) \to a(bx) \tag{4.32b}$$
$$x(ba) \to x(ab) \tag{4.32c}$$
$$x^{-1}(ax) \to a(xx^{-1}) \tag{4.32d}$$
$$(bx)^{-1}(ba)^{-1} \to (ab)^{-1}(bx)^{-1} \tag{4.32e}$$

In practice, this criterion can be easily implemented using topological sorting. Note, however, that the result may not be unique, e.g. (4.32a/b).

**Theorem 4.11.** AC normalisation is a simplification rule of the superposition calculus wrt. closure redundancy.

*Proof.* Consider each instance $C \cdot \theta$. The conclusion $C[t_1(\cdots t_n) \mapsto t'_1(\cdots t'_n)]$ of (4.31) is smaller than the premise $C$, since $t_1(\cdots t_n) \succ_t t'_1(\cdots t'_n)$, therefore all instances $C[t_1(\cdots t_n) \mapsto t'_1(\cdots t'_n)] \cdot \theta$ of the conclusion are smaller than $C \cdot \theta$ (Lemma 3.6). It is also trivial to see that the conclusion follows from the premises: $\{t_1,\dots,t_n\} = \{t'_1,\dots,t'_n\}$ means $t_1(\cdots t_n)$ and $t'_1(\cdots t'_n)$ are equal modulo $AC_f$.

[15] Note we trivially assume all $AC_f$ terms are right associative, since $(xy)z \to x(yz)$ is always oriented; we are still assuming the conditions for $\succ_t$ in page 83).

[15] As multisets.

Now, let $m$ be a bijection over integers in $[1, n]$ such that $t_i = t'_{m(i)}$. Then consider the following clause: $D = x_1(\cdots x_i(\cdots x_n)) \approx x_{m(1)}(\cdots x_{m(i)}(\cdots x_{m(n)}))$. We have

$$AC_f \models D \tag{4.33a}$$

$$C, D \models C[t_1(\cdots t_n) \mapsto t'_1(\cdots t'_n)] \tag{4.33b}$$

Let $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$. It is easy to see that for all instances $C \cdot \theta$ we have $D \cdot \sigma\theta \prec_{cc} C \cdot \theta$, since $x_1(\cdots x_n) \sqsubset t_1(\cdots t_n) \trianglelefteq C$ (the only way it would not be so is if all $t_i$ were distinct variables, but then $t_1, \ldots, t_n \succ_{\text{lex}} t'_1, \ldots, t'_n$ would not hold), and $x_1(\cdots x_n)\theta \succ x_{m(1)}(\cdots x_{m(n)})\theta$. Therefore we conclude that all $C \cdot \theta$ follow from the smaller $C[t_1(\cdots t_n) \mapsto t'_1(\cdots t'_n)] \cdot \theta$ and $D \cdot \sigma\theta$, and therefore $C$ is redundant in $\{C[t_1(\cdots t_n) \mapsto t'_1(\cdots t'_n)],\ D\}$.

Finally, since $D$ is redundant in $AC_f$ (Theorem 4.9), then by monotonicity (Lemma 3.9), $C$ is redundant in $\{C[t_1(\cdots t_n) \mapsto t'_1(\cdots t'_n)]\} \cup AC_f$. □

The main advantages of applying this simplification rule are as follows.

- Strictly more redundant clauses found. For example, in the set $\{a(bx), a(xb), x(ab), b(xa), b(ax), x(ba)\}$, the last three are redundant, instead of only the last one.

- Faster implementation. Even for simplifications that were already allowed by demodulation, we avoid the work of searching in indices and instantiating the axioms to perform the rewrites. In fact, we can avoid storing $AC_f$ in the demodulation indices entirely, as AC normalisation is strictly more general than demodulation via $AC_f$. Since (4.26a) matches with all $f$-terms, and (4.26c) with all $f$-terms with 3 or more elements, this makes all queries on those indices faster.

## AC demodulation

Consider again situations like Example 4.4. We can see clearly that whenever we have, for instance, $s_1 s_n \approx t$ and $s_1(s_2(\cdots s_n))$, the latter can be rewritten to $s_2(\cdots(s_{n-1}t))$, In fact, more generally, if we have $t_1 t_n \approx t$ with $t_1\theta = s_1$ and $t_n\theta = s_n$, we can rewrite $s_1(s_2(\cdots s_n))$ to $s_2(\cdots(s_{n-1}\, t\theta))$. However, this process:

(i) may involve several superposition and demodulation steps (where each intermediate superposition step generates a clause that needs to be tracked and stored), and

(ii) is highly sensitive to the term ordering, for example: if $s_1 \prec_t s_n \prec_t \cdots$,[17] an AC normalisation inference suffices to put it into a form where encompassment demodulation via $t_1(t_n x) \approx xt$ can be used to perform the rewrite, but if $s_1 \prec_t \cdots \prec_t s_n$, as in Example 4.4, this is only possible after generating $t_1(x_2(\cdots(x_{n-1}t_n))) \approx x_2(\cdots x_{n-1})$.

More formally, we know that

$$\begin{cases} AC_f \\ l_1(\cdots l_k) \approx r \\ C[t_1(\cdots t_n)] \end{cases} \quad \text{implies} \quad C' = C[t_1(\cdots t_n) \mapsto t_1'(\cdots(t_m' \, r\theta))] \,, \tag{4.34}$$

if $\{l_1\theta, \ldots, l_k\theta\} \subseteq \{t_1, \ldots, t_n\}$ and $\{t_1', \ldots, t_m'\} = \{t_1, \ldots, t_n\} \setminus \{l_1\theta, \ldots, l_k\theta\}$, i.e. a sort of “rewrite modulo AC”. We show how in some positions this rewrite does not compromise completeness; we call this “AC demodulation” and show that it is a simplification rule of the superposition calculus. Furthermore, it has very desirable properties from an implementation point of view, as we can effectively re-use the same indexing structures of clause subsumption.

Let

$$\text{AC demodulation} \qquad \frac{\cancel{C[t_1(\cdots t_n)]} \quad l_1(\cdots l_k) \approx r \quad AC_f}{C[t_1'(\cdots(t_m' \, r\theta))]} \tag{4.35}$$

where

- $\{l_1\theta, \ldots, l_k\theta\} \subseteq \{t_1, \ldots, t_n\}$,
- $\{t_1', \ldots, t_m'\} = \{t_1, \ldots, t_n\} \setminus \{l_1\theta, \ldots, l_k\theta\}$,
- $t_1(\cdots t_n) \lhd C \succ_c C[t_1'(\cdots(t_m' \, r\theta))]$,

**Theorem 4.12.** AC demodulation is a simplification rule of the superposition calculus wrt. closure redundancy.

*Proof.* Let us assume, again, that the clauses are normalised wrt. $(xy)z \to x(yz)$. Let $C' = C[t_1(\cdots t_n) \mapsto t_1'(\cdots(t_m' \, r\theta))]$ (the conclusion). Consider an arbitrary ground closure $C' \cdot \sigma$.

[17] Where ‘$\cdots$’ stands for the remaining $s_i$.

Let us also make the correspondence between the $l_i$ and the $t_i$ explicit: $\lambda$ is an injective mapping from $\{l_i\}$ such that $\lambda(l_i)\,\theta = t_i$.

If $t_1(\cdots t_n)$ occurs at a subterm position of $C$, then consider the equation $D = x_1(\cdots x_n) \approx x_{m(1)}(\cdots x_{m(n)})$ and the substitution $\rho$ such that $x_i\rho = t_i$.[18] We choose a permutation where $x_{m(n-k+i)}\rho = \lambda(l_i)$ — informally, which shuffles the terms into the correct order so that $l_1(\cdots l_k)$ matches syntactically on the rightmost positions.

It is easy to see that $C$ together with $D\rho$ and $l_1(\cdots l_k)\theta \approx r\theta$ imply $C'$, therefore $C \cdot \sigma$ together with $D \cdot \rho\sigma$ and $(l_1(\cdots l_k) \approx r) \cdot \theta\sigma$ imply $C' \cdot \sigma$. We know that $C' \cdot \sigma$ is smaller than $C \cdot \sigma$ by construction; $D \cdot \rho\sigma$ and $(l_1(\cdots l_k) \approx r) \cdot \theta\sigma$ are also smaller than $C \cdot \sigma$, since their maximal terms are subterms of $C\sigma$ (by Lemma 3.7). Therefore, as the above holds for any $\sigma$, $C$ is redundant wrt. $\{C',\ l_1(\cdots l_k) \approx r,\ D\}$, and since $D$ is redundant wrt. $AC_f$ (Theorem 4.9), $C$ is redundant wrt. $\{C',\ l_1(\cdots l_k) \approx r\} \cup AC_f$ (Lemma 3.9). $\square$

The application of this simplification rule in saturation is however restricted by the condition that the occurrence must happen at a subterm position. In particular, we cannot prevent, for an equation such as e.g. $xx^{-1} \approx u$, an infinite number of recombinations through superposition inferences with $AC_f$,which produce $x(x^{-1}y) \approx uy$, $x(yx^{-1}) \approx uy$, $x(y(x^{-1}z)) \approx u(yz)$, etc.,[19] even though in practice many of these consequences can be proven redundant via a combination of AC normalisation, AC joinability, and connectedness. Nonetheless, AC demodulation at top positions can of course still be employed whenever refutational completeness is not a constraint, including during preprocessing phases.

### Implementation

One important advantage of AC demodulation is that the task of finding candidates for it, either in the forward or backward direction,[20] consists of finding multisets of terms (from AC-subterms) such that

$$\{l_1\theta, \ldots, l_k\theta\} \subseteq \{t_1, \ldots, t_n\}, \tag{4.36}$$

[18]See proof of Theorem 4.11.

[19]Although there are particular term orderings where this is the case, see e.g. Martin and Nipkow [1990].

[20]I.e. using an existing set of equations to simplify one clause, and using an equation to simplify an existing set of clauses, respectively.

and is therefore almost exactly equivalent to the task of finding subsumed/subsuming clauses.[21] There is extensive literature on this subject [Graf 1995; Riazanov 2003; Schulz 2013a], as subsumption is a very important simplification rule where efficient indexing has a major impact on performance, which means that implementing AC demodulation is simply a matter of recycling the algorithms and data structures used for subsumption (indexing terms in an AC subterm, rather than literals in a clause), and adding checks for the completeness conditions in (4.35).

[21]Subsumption has the added complication of needing to consider the flipping of equalities.

# Chapter 5

# Algorithms for theorem proving

Beware of bugs in the above code; I have only proved it correct, not tried it.

Donald Knuth (1977)

In this chapter we explore several topics connected to algorithms related to the preceding topics, including a flexible setup for a given-clause saturation loop, and a novel algorithm for checking strong ground joinability of two terms.

Of the material presented in this chapter, the section on ground joinability has been peer-reviewed and published in Duarte and Korovin 2022, while the section on the given clause loop and simplification setup has been peer-reviewed and published in Duarte and Korovin 2020.

## 5.1 Ground joinability

The general criterion for simplification of clauses with ground joinable literals (4.10) is extremely powerful, but it raises the question of how to test, in practice, whether $s \mathrel{\rotatebox[origin=c]{-90}{$\rightsquigarrow$}} t$ in a clause $s \mathrel{\dot{\approx}} t \lor C$. Several such algorithms have been proposed [Martin and Nipkow 1990; Avenhaus, Hillenbrand and Löchner 2003; Smallbone 2021].

All of these are based on the following observation: for a given simplification ordering on terms $\succ_t$, if we consider all total preorders $\succeq_v$ over $\mathrm{Vars}(s, t)$,[1] and for all of them show strong joinability with a modified ordering — which we

[1]Where $\mathrm{Vars}(s, t) = \{x \mid s[x] \text{ or } t[x]\}$.

denote $\succ_{t[v]}$ — which extends $\succ_t$ and $\succeq_v$, then we have shown strong *ground* joinability in the order $\succ_t$ [Martin and Nipkow 1990].

Let us define this "extension" of a term ordering with a variable preorder as follows.

**Definition 5.1.** A simplification order on terms $\succ_t$ **extended with** a preorder on variables $\succeq_v$, denoted $\succeq_{t[v]}$, is a simplification preorder (i.e. its strict part satisfies all the relevant properties of a simplification order) such that $\succeq_{t[v]} \supseteq \succ_t \cup \succeq_v$.

*Example* 5.1. If $x \succ_v y$, then e.g.:

- $g(x) \succ_{t[v]} g(y)$,
- $g(x) \succ_{t[v]} y$,
- $f(x, y) \succ_{t[v]} f(y, x)$[2],

Of course, if $\succeq_v$ is total over $\mathrm{Vars}(s, t)$, then $\succ_{t[v]}$ is total over all terms (ground or not).

The simplest algorithm based on this approach would be to enumerate all possible total preorders $\succeq_v$ over $\mathrm{Vars}(s, t)$, and exhaustively reduce both sides via equations in $S$ orientable by $\succ_{t[v]}$, checking if the terms can be reduced to the same normal form for all total preorders. If yes, then $s$ and $t$ are ground joinable. This is very inefficient since there are $\mathcal{O}(n!e^n)$ such total preorders [Barthelemy 1980], where $n$ is the cardinality of $\mathrm{Vars}(s, t)$.

Let us use $\Downarrow_{\succ}$ for the strong joinability relation defined in Definition 4.2 with an explicitly given term ordering $\succ_t = \succ$. Another approach is to consider only a smaller number of partial preorders, based on the obvious observation that

$$s \Downarrow_{\succ_{t[v]}} t \Rightarrow \forall \succeq'_v \supseteq \succeq_v .\ s \Downarrow_{\succ_{t[v']}} t\,, \tag{5.1}$$

that is: strong joinability under $\succ_t$ extended with some partial variable preorder implies strong joinability under $\succ_t$ extended with any variable preorder which contains it. This means that joinability under a smaller number of partial preorders can imply joinability under all the total preorders, necessary to prove ground joinability.

However, this raises the question of how to choose which partial preorders to check. Intuitively, for performance, we would like the following:

[2]If $s \succ_t t \Rightarrow f(s, t) \succ_t f(t, s)$, of course.

- Whenever the two terms are **not** ground joinable, that some total preorder where they are not joinable is found as early as possible; and that
- Whenever the two terms **are** ground joinable, that all total preorders needed to conclude so are covered in as few partial preorders as possible.

*Example* 5.2. Let $S = \{f(x, f(y, z)) \approx f(y, f(x, z))\}$. Then

$$f(x, f(y, f(z, f(w, u)))) \approx f(x, f(y, f(w, f(z, u)))) \tag{5.2}$$

can be shown to be (strongly) ground joinable wrt. $S$ by checking just three cases: $\succeq_v \in \{z{\succ}w\ ,\ z{\sim}w\ ,\ z{\prec}w\}$, even though there are 6942 possible preorders [Sloane 2022b].

Approaches to this problem vary. Waldmeister (a well-established and successful unfailing completion prover) first selects two variables among $\mathrm{Vars}(s, t)$ and tries to show joinability under all preorders over those. If this fails it tries another two variables, and if there are no more pairs of variables it moves on to selecting *three* variables from among $\mathrm{Vars}(s, t)$, etc., until success (by showing joinability under all preorders over a subset of $\mathrm{Vars}(s, t)$), failure (by trying a total preorder and failing to join) or reaching a predefined limit of attempts [Avenhaus, Hillenbrand and Löchner 2003].

Twee (a newer prover with outstanding experimental success) has a somewhat opposite direction: first it tries an arbitrary total preorder on variables, then tries to weaken it until $s$ and $t$ are no longer joinable, then repeats this process until all total preorders have been thus covered, or until the arbitrary total preorder chosen does not join the two terms [Smallbone 2021].

We propose a novel algorithm — incremental ground joinability — whose main improvement is *guiding* the process of picking which preorders to check by finding, during the process of searching for rewrites on subterms of the terms we are attempting to join, minimal extensions of the term order with a variable preorder which allow the rewrite to be done in the $\succ$ direction.

## Incremental ground joinability

Our algorithm is summarised as follows (we shall present a more formal description in a few pages).

- We start with an empty queue of variable preorders, $V$, initially containing only the empty preorder.

- Then, while $V$ is not empty, we pop a preorder $\succeq_v$ from the queue, and attempt to perform a rewrite on the terms we wish to join via an equation which is newly orientable *using some extension* $\succeq_v'$ of $\succeq_v$. That is, during the process of finding generalisations of a subterm of $s$ or $t$ among left-hand sides of candidate unoriented unit equations $l \approx r$, when we check that the instance $l\theta \approx r\theta$ used to rewrite is oriented, we try to force this to be true under some minimal extension $\succ_{t[v']}$ of $\succ_{t[v]}$, if possible.[3]
    - If no such rewrite exists, or if we simply cannot extend $\succ_{t[v]}$ further because $\succeq_v$ is already total, then the two terms are not strongly joinable under $\succ_{t[v]}$ or any extension, and so are not strongly ground joinable and we are done with a **negative** result.
    - If it exists, we continue the process.
- Now, we exhaustively rewrite the terms using $\succ_{t[v']}$, and check if we obtain the same normal form.
    - If we do not obtain it yet, we repeat the process of searching for rewrites via equations orientable by further extensions of $\succ_{t[v']}$.
    - But if we do, then we have proven joinability in the extended preorder $\succ_{t[v']}$; now we must add back to the queue a set of preorders $O$ such that: all the total variable preorders which are $\supseteq \succeq_v$ (initially popped from the queue) but not $\supseteq \succeq_v'$ (minimal extension under which we have proven joinability) are $\supseteq$ of some $\succeq_v'' \in O$ (pushed back into the queue to be checked).

      Obtaining this $O$ and ensuring it is as minimal as possible and doesn't overlap with the preorders which have already been checked is an important part of our algorithm; it is implemented by $\mathrm{order\_diff}(\succeq_v, \succeq_v')$ as defined below.
- Whenever there are no more preorders in the queue to check, then we have checked that the terms are strongly joinable under all possible total preorders, and we are done with a **positive** result.

Together with this, some book-keeping for keeping track of completeness conditions is necessary. We know that for completeness to be guaranteed, the

[3]We show how to compute this for Knuth-Bendix orders at the end of this section.

conditions in Definition 4.1 must hold. They automatically do if $C$ is not a positive unit or if a rewrite happens on a strict subterm. We also know that after a term has been rewritten at least once, rewrites on that side are always complete (since it was rewritten to a smaller term). Therefore we store in the queue, together with the preorder, a flag in $\mathcal{P}(\{\mathsf{L},\mathsf{R}\})$ indicating on which sides does a top rewrite need to be checked for completeness. Consider a literal $s \mathrel{\dot{\approx}} t$ in some clause. Initially the flag is $\{\mathsf{L}\}$ if $s \succ_t t$, $\{\mathsf{R}\}$ if $s \prec_t t$, $\{\mathsf{L},\mathsf{R}\}$ if $s$ and $t$ are incomparable, and $\{\}$ if the clause is not a positive unit. When a rewrite at the top is attempted (say, $l \approx r$ used to rewrite $s = l\theta$ with $t$ being the other side), if the flag for that side is set, then we check if $l\theta \sqsupset l$ or $r\theta \prec t$. If this fails, the rewrite is rejected. Finally, whenever a side is rewritten (at any position), the flag for that side is cleared.

**Order diff**

Intuitively, $\mathrm{order\_diff}(\succeq_1, \succeq_2)$ computes the following set: given a preorder $\succeq_1$ (the preorder which we popped from the queue, and under which therefore we needed to verify ground joinability), and a preorder $\succeq_2$ (the preorder under which we *did* manage to verify ground joinability), a set such that all preorders which contain $\succeq_1$ but do not contain $\succeq_2$, contain one (and only one!) preorder in that set. This is what enables us to figure out which variable preorders we need to add to the queue, in each incremental ground joinability iteration.

The definition of order_diff is as follows. Let a transitive reduction of a preorder be represented by a set of edges of the form $x{\succ}y$ / $x{\sim}y$. Then[4]

$$\mathrm{order\_diff}(\succeq_1, \succeq_2) = \{{\succeq^+} \mid {\succeq} \in \mathrm{order\_diff}'(\succeq_1, {\succeq_2}^-)\}\,, \tag{5.3a}$$

$$\mathrm{order\_diff}'(\succeq_1, \succeq_2^-) = \tag{5.3b}$$

$$\begin{cases} \succeq_2^- = \{x{\succ}y\} \uplus {\succeq_2^-}' \Rightarrow \begin{cases} x \succ_1 y \Rightarrow \mathrm{order\_diff}'(\succeq_1, {\succeq_2^-}') \\ x \nsucc_1 y \Rightarrow \begin{array}{l} \{\succeq_1 \cup \{y{\succ}x\}\,,\ \succeq_1 \cup \{x{\sim}y\}\} \\ \cup\, \mathrm{order\_diff}'(\succeq_1 \cup \{x{\succ}y\}, {\succeq_2^-}') \end{array} \end{cases} \\ \succeq_2^- = \{x{\sim}y\} \uplus {\succeq_2^-}' \Rightarrow \begin{cases} x \sim_1 y \Rightarrow \mathrm{order\_diff}'(\succeq_1, {\succeq_2^-}') \\ x \nsim_1 y \Rightarrow \begin{array}{l} \{\succeq_1 \cup \{x{\succ}y\}\,,\ \succeq_1 \cup \{y{\succ}x\}\} \\ \cup\, \mathrm{order\_diff}'(\succeq_1 \cup \{x{\sim}y\}, {\succeq_2^-}') \end{array} \end{cases} \\ \succeq_2^- = \emptyset \qquad\qquad\quad \Rightarrow \emptyset\,. \end{cases}$$

[4]Recall $\succeq^-$ and $\succeq^+$ denote a transitive reduction and the transitive closure, respectively.

where $\succeq_1 \subseteq \succeq_2$. Informally: we take a transitive reduction of $\succeq_2$, and for all edges $\ell$ (of the form $x{\succ}y$ or $x{\sim}y$) in the set that represents that reduction, if $\ell \notin \succeq_1$, return the preorders $\succeq_1$ augmented with the reverse of $\ell$,[5] and recurse with $\succeq_1 = \succeq_1 \cup \ell$. Note that $\mathrm{order\_diff}(\succeq_1, \succeq_2)$ is not unique, and depends on the exact transitive reduction used, and on the order of visiting its edges.

*Example* 5.3.

$$\begin{matrix} \succeq_1 = x \succ y \\ \succeq_2 = x \succ y \succ z \succ w \end{matrix} \quad \Rightarrow \ \mathrm{order\_diff}(\succeq_1, \succeq_2) = \begin{cases} x \succ y \sim z \\ x \succ y \prec z \\ x \succ y \succ z \sim w \\ x \succ y \succ z \prec w \end{cases} \tag{5.4}$$

*Example* 5.4.

$$\begin{matrix} \succeq_1 = y \prec x \succ z \\ \succeq_2 = x \succ y \succ z \end{matrix} \quad \Rightarrow \ \mathrm{order\_diff}(\succeq_1, \succeq_2) = \begin{cases} x \succ y \sim z \\ x \succ z \succ y \end{cases} \tag{5.5}$$

As mentioned before, it is important that there is as little overlap as possible (while of course ensuring every relevant preorder is covered); this definition has properties which ensure that is the case. It it also easy to compute, given only algorithms for computing transitive reductions and closures, and a suitable data structure to represent a (partial) preorder.

**Theorem 5.1.** For all total $\succeq_v^T \supseteq \succeq_1$, there exists one and only one $\succeq_i \in \{\succeq_2\} \cup \mathrm{order\_diff}(\succeq_1, \succeq_2)$ such that $\succeq_v^T \supseteq \succeq_i$. For all $\succeq_v^T \not\supseteq \succeq_1$, there is no $\succeq_i \in \{\succeq_2\} \cup \mathrm{order\_diff}(\succeq_1, \succeq_2)$ such that $\succeq_v^T \supseteq \succeq_i$.

*Proof.* It is easy to see that all $\succeq \in \{\succeq_2\} \cup \mathrm{order\_diff}(\succeq_1, \succeq_2)$ have $\succeq_1 \subseteq \succeq$, therefore if $\succeq^T \not\supseteq \succeq_1$ then there exists no $\succeq$ in that set such that $\succeq^T \supseteq \succeq$. Consider a total $\succeq^T \supseteq \succeq_1$. We will show recursively on the definition of $\mathrm{order\_diff}'$ that there either exists one and only one $\succeq^- \in \mathrm{order\_diff}'(\succeq_1, \succeq_2^-)$ such that $\succeq^T \supseteq \succeq^-$, or else there exists none and then $\succeq^T \supseteq \succeq_2$.

Consider a call to $\mathrm{order\_diff}'(\succeq_a, \succeq_b)$. Let us denote $\mathrm{order\_diff}''(\succeq_a, \succeq_b) = \{\succeq_2\} \cup \mathrm{order\_diff}'(\succeq_a, \succeq_b)$. We will show that $\succeq^T \supseteq \succeq^- \in \mathrm{order\_diff}''(\succeq_a, \succeq_b)$ by either showing an explicit $\succeq^- \in \mathrm{order\_diff}'(\succeq_a, \succeq_b)$ or recursing with a $\succeq_b'$

[5] I.e., $\succeq_1 \cup \{y{\succ}x\}$ , $\succeq_1 \cup \{x{\sim}y\}$ if $\ell$ is of the form $x{\succ}y$, and $\succeq_1 \cup \{x{\succ}y\}$ , $\succeq_1 \cup \{y{\succ}x\}$ if $\ell$ is of the form $x{\sim}y$.

which is a strict subset of $\succeq_b$. Since the initial $\succeq_b = \succeq_2$ is finite this terminates. Note also the following invariants are maintained throughout: $\succeq^T \supseteq \succeq_a$, and $(\succeq_a \cup \succeq_b)^+ = \succeq_2$ (they are trivially true in the initial call $\mathrm{order\_diff}'(\succeq_1, \succeq_2^-)$).

Consider all possible cases following the definition of $\mathrm{order\_diff}'$ (see (5.3b)).

1. $\succeq_b = \{x \succ y\} \uplus \succeq'_b$.

1.1. If $x \succ_a y$, then $x \succ^T_v y$ (by hypothesis $\succeq_a \subseteq \succeq^T$), by definition of $\mathrm{order\_diff}'$, in this case $\mathrm{order\_diff}'(\succeq_a, \succeq_b) = \mathrm{order\_diff}'(\succeq_a, \succeq'_b)$, then $\succeq^T \supseteq \succeq^- \in \mathrm{order\_diff}''(\succeq_a, \succeq_b)$ iff $\succeq^T \supseteq \succeq^- \in \mathrm{order\_diff}''(\succeq_a, \succeq'_b)$. The invariants $\succeq^T \supseteq \succeq_a$, and $(\succeq_a \cup \succeq'_b)^+ = \succeq_2$ are maintained.

1.2. Otherwise, $x \not\succ_a y$, so $\succeq^T$ may have $x \succ^T y$, $x \sim^T y$, or $x \prec^T y$.

1.2.1. If $x \prec^T y$, then $\succ^T \supseteq \succeq_a \cup \{x \prec y\}$, so there is (by definition of $\mathrm{order\_diff}'$) a $\succeq^- = \succeq_a \cup \{x \prec y\} \in \mathrm{order\_diff}'(\succeq_a, \succeq_b)$ such that $\succeq^T \supseteq \succeq^-$. Any other $\succeq^{-\prime} \in \mathrm{order\_diff}'(\succeq_a, \succeq_b) \setminus \{\succeq^-\}$ have (again by definition) $x \succ' y$ or $x \sim' y$, so $\succeq^T \not\supseteq \succeq'^-$, so that $\succeq^-$ is unique. So we are done.

1.2.2. Similarly, if $x \sim^T y$, then $\succ^T \supseteq \succeq_a \cup \{x \sim y\}$, so there is (by definition of $\mathrm{order\_diff}'$) a $\succeq^- = \succeq_a \cup \{x \sim y\} \in \mathrm{order\_diff}'(\succeq_a, \succeq_b)$ such that $\succeq^T \supseteq \succeq^-$. Any other $\succeq^{-\prime} \in \mathrm{order\_diff}'(\succeq_a, \succeq_b) \setminus \{\succeq^-\}$ have (again by definition) $x \succ' y$ or $x \prec' y$, so $\succeq^T \not\supseteq \succeq'^-$, so that $\succeq^-$ is unique. So we are done.

1.2.3. If $x \succ^T y$, by hypothesis $\succeq_a \subseteq \succeq^T$, and therefore $\succeq^T \supseteq \succeq_a \cup \{x \succ y\}$. By definition of $\mathrm{order\_diff}'$, in this case $\mathrm{order\_diff}'(\succeq_a, \succeq_b) = \mathrm{order\_diff}'(\succeq_a \cup \{x \succ y\}, \succeq'_b)$, therefore we have $\succeq^T \supseteq \succeq^- \in \mathrm{order\_diff}''(\succeq_a, \succeq_b)$ iff $\succeq^T \supseteq \succeq^- \in \mathrm{order\_diff}''(\succeq_a \cup \{x \succ y\}, \succeq'_b)$, and the invariants are maintained: $\succeq^T \supseteq \succeq_a \cup \{x \succ y\}$, and $(\succeq_a \cup \{x \succ y\} \cup \succeq'_b)^+ = \succeq_2$.

2. $\succeq_b = \{x \sim y\} \uplus \succeq'_b$. This case is similar to the previous.

2.1. If $x \sim_a y$, then $x \sim^T y$ (by hypothesis $\succeq_a \subseteq \succeq^T$), by definition of $\mathrm{order\_diff}'$, in this case $\mathrm{order\_diff}'(\succeq_a, \succeq_b) = \mathrm{order\_diff}'(\succeq_a, \succeq'_b)$, then $\succeq^T \supseteq \succeq^- \in \mathrm{order\_diff}''(\succeq_a, \succeq_b)$ iff $\succeq^T \supseteq \succeq^- \in \mathrm{order\_diff}''(\succeq_a, \succeq'_b)$. The invariants $\succeq^T \supseteq \succeq_a$, and $(\succeq_a \cup \succeq_b)^+ = \succeq_2$ are maintained.

2.2. Otherwise, $x \nsim_a y$, so $\succeq^T$ may have $x \succ^T y$, $x \sim^T y$, or $x \prec^T y$.

2.2.1. If $x \succ^T y$, then $\succ^T \supseteq \succeq_a \cup \{x \succ y\}$, so there is (by definition of $\mathrm{order\_diff}'$) a $\succeq^- = \succeq_a \cup \{x \succ y\} \in \mathrm{order\_diff}'(\succeq_a, \succeq_b)$ such that $\succeq^T \supseteq \succeq^-$. Any other $\succeq^{-\prime} \in \mathrm{order\_diff}'(\succeq_a, \succeq_b) \setminus \{\succeq^-\}$ have (again by definition) $x \sim' y$ or $x \prec' y$, so $\succeq^T \not\supseteq \succeq'^-$, so that $\succeq^-$ is unique.

2.2.2. Similarly, if $x \prec^T y$, then $\succ^T \supseteq \succeq_a \cup \{x \prec y\}$, so there is (by definition of $\mathrm{order\_diff}'$) a $\succeq^- = \succeq_a \cup \{x \prec y\} \in \mathrm{order\_diff}'(\succeq_a, \succeq_b)$ such that $\succeq^T \supseteq \succeq^-$. Any other $\succeq^{-\prime} \in \mathrm{order\_diff}'(\succeq_a, \succeq_b) \setminus \{\succeq^-\}$ have (again by definition) $x \succ' y$ or $x \sim' y$, so $\succeq^T \not\supseteq \succeq^{\prime -}$, so that $\succeq^-$ is unique.

2.2.3. If $x \sim^T y$, by hypothesis $\succeq_a \subseteq \succeq^T$, and therefore $\succeq^T \supseteq \succeq_a \cup \{x \sim y\}$. By definition of $\mathrm{order\_diff}'$, in this case $\mathrm{order\_diff}'(\succeq_a, \succeq_b) = \mathrm{order\_diff}'(\succeq_a \cup \{x \sim y\}, \succeq_b')$, therefore we have $\succeq^T \supseteq \succeq^- \in \mathrm{order\_diff}''(\succeq_a, \succeq_b)$ iff $\succeq^T \supseteq \succeq^- \in \mathrm{order\_diff}''(\succeq_a \cup \{x \sim y\}, \succeq_b')$, and the invariants are maintained: $\succeq^T \supseteq \succeq_a \cup \{x \sim y\}$, and $(\succeq_a \cup \{x \sim y\} \cup \succeq_b')^+ = \succeq_2$.

3. $\succeq_b = \emptyset$. Then $\mathrm{order\_diff}'(\succeq_a, \succeq_b) = \emptyset$, so there is no $\succeq \in \mathrm{order\_diff}'(\succeq_a, \succeq_b)$ such that $\succeq^T \supseteq \succeq$. But since we have $(\succeq_a \cup \emptyset)^+ = \succ_2$ and $\succeq^T \supseteq \succeq_a$, then $\succeq^T \supseteq \succeq_2$. □

Note that this theorem gives strong guarantees which are very important for efficiency of the algorithm: that all total preorders $\succeq_v^T \supseteq \succeq_1$ are covered by one of the preorders in the set $\{\succeq_2\} \cup \mathrm{order\_diff}(\succeq_1, \succeq_2)$ (ensuring correctness) but also by *only one* of them (ensuring no overlap and duplicated work), and that all preorders $\succeq_v^T \not\supseteq \succeq_1$ are covered by none of them (ensuring no useless work).

An algorithm based on searching for rewrites in minimal extensions of a variable preorder (starting with minimal extensions of the bare term ordering, $\succ_{t[\emptyset]}$), has several advantages. The main benefit of this approach is that, instead of imposing an a priori ordering on variables and then checking joinability under that ordering, we instead build a minimal ordering *while* searching for candidate unit equations to rewrite subterms of $s, t$. For instance, if two terms are *not* ground joinable, or not even rewritable in any $\succ_{t[v]}$ where it was not rewritable in $\succ_t$, then an approach such as the one used in Avenhaus, Hillenbrand and Löchner [2003] cannot detect this until it has extended the preorder arbitrarily to a total ordering, while our incremental algorithm immediately realises this. We should note that empirically this is what happens in most cases: most of the literals we check during a run are *not* ground joinable, so for practical performance it is essential to optimise this case.

We now give a proof that incremental ground joinability implements Definition 4.3.

**Algorithm 1:** Incremental ground joinability test

```
Input: literal $s \dot{\approx} t \in C$; set of unoriented equations $S$
Output: whether $s \between t$ in $C$ wrt. $S$
begin
    $c \leftarrow \emptyset$ if $C$ not pos. unit, $\{\mathsf{L}\}$ if $s \succ t$, $\{\mathsf{R}\}$ if $s \prec t$, $\{\mathsf{L}, \mathsf{R}\}$ otherwise
    $V \leftarrow \{\langle \emptyset, s, t, c \rangle\}$
    while $V$ is not empty do
        $\langle \succeq_v, s, t, c \rangle \leftarrow$ pop from $V$
        $s, t, c \leftarrow$ normalise $s, t$ wrt. $\succ_{t[v]}$, with completeness flag $c$
        if $s \sim_{t[v]} t$ then
            continue
        else
            $s', t', c' \leftarrow s, t, c$
            while exists rewrite in $s', t'$ via $S$ wrt. extension of $\succeq_{s[v]}$
              with completeness flag $c$ do
                $s', t', c' \leftarrow$ normalise $s', t'$ wrt. $\succ_{t[v']}$, with compl. flag $c$
                if $s' \sim_{t[v']} t'$ then
                    push $\{\langle \succeq''_v, s, t, c \rangle \mid \succeq''_v \in \mathrm{order\_diff}(\succeq_v, \succeq'_v)\}$ to $V$
                    break
                end
                $\succeq_v \leftarrow \succeq'_v$
            else
                return Fail
            end
        end
    else
        return Success
    end
end
```

**Algorithm 1 (cont.):** Incremental ground joinability test

```
function normalise $s, t$ via $S$ wrt. $\succeq$ with completeness flag $c$
    if exists $l \approx r \in S$ and $\sigma$ such that
        • $l\sigma \trianglelefteq s$ or $l\sigma \trianglelefteq t$, and
        • $l\sigma \succ r\sigma$, and
        • rewrite $s, t$ via $l \approx r$ wrt. $\succeq$ with complet. flag $c$ is admissible
    then
        $s \leftarrow s[l\sigma \mapsto r\sigma]$
        $t \leftarrow t[l\sigma \mapsto r\sigma]$
        if $s$ was changed then $c \leftarrow c \setminus \{\mathsf{L}\}$
        if $t$ was changed then $c \leftarrow c \setminus \{\mathsf{R}\}$
        return normalise $s, t$ via $S$ wrt. $\succeq$ with completeness flag $c$
    else
        return $\langle s, t, c \rangle$
    end
end
function rewrite $s, t$ via $l \approx r$ wrt. $\succeq$ with completeness flag $c$ is
  admissible
    return
        • $u$ is a strict subterm of $s$ or $t$, or
        • $u = s$ with $\mathsf{L} \notin c$, or
        • $u = t$ with $\mathsf{R} \notin c$, or
        • $l \sqsubset u$, or
        • $u = s$ with $r\sigma \prec t$, or
        • $u = t$ with $r\sigma \prec s$.
end
function exists rewrite in $s, t$ via $S$ wrt. extension of $\succeq_{t[v]}$ with
  completeness flag $c$
    return exists $l \approx r \in S$, $\sigma$, and $\succeq'_v \supset \succeq_v$ such that
        • $l\sigma \trianglelefteq s$ or $l\sigma \trianglelefteq t$, and
        • $l\sigma \succ_{t[v']} r\sigma$, and
        • rewrite $s, t$ via $l \approx r$ wrt. $\succeq_{t[v']}$ with complet. flag $c$ is admissible
end
```

**Theorem 5.2.** Algorithm 1 returns "Success" only if $s \Downarrow t$ in $C$ wrt. $S$.[6]

*Proof.* We will show that Algorithm 1 returns "Success" if and only if $s \downarrow_{\succ_{t[v^T]}} t$ for all total $\succeq_v^T$ over $\mathrm{Vars}(s,t)$, which implies $s \Downarrow_{\succ_t} t$.

When $\langle \succeq_v, s, t, c \rangle$ is popped from $V$, we exhaustively reduce $s, t$ via equations in $S$ oriented wrt. $\succ_{t[v]}$, obtaining $s^r, t^r$. If $s^r \sim_{t[v]} t^r$, then $s \downarrow_{\succ_{t[v]}} t$, and so $s \downarrow_{\succ_{t[v^T]}} t$ for all total $\succeq_v^T \supseteq \succeq_v$. If $s^r \nsim_{t[v]} t^r$, we will attempt to rewrite one of $s^r, t^r$ using *some* extended $\succ_{t[v']}$ where $\succeq_v' \supset \succeq_v$. If this is impossible, then $s \not\downarrow_{\succ_{t[v']}} t$ for any $\succeq_v' \supseteq \succeq_v$, and therefore there exists at least one total $\succeq_v^T$ such that $s \not\downarrow_{\succeq_v^T} t$, and we return "Fail".

If this is possible, then we repeat the process: we exhaustively reduce wrt. $\succ_{t[v']}$, obtaining $s', t'$. If $s' \nsim_{t[v']} t'$, then we start again the process from the step where we attempt to rewrite via an extension of $\succeq_v'$: we either find a rewrite with some $\succ_{t[v'']}$ with $\succeq_v'' \supset \succeq_v'$, and exhaustively normalise wrt. $\succ_{t[v'']}$ obtaining $s'', t''$, etc., or we fail to do so and return "Fail".

If in any such step (after exhaustively normalising wrt. $\succ_{t[v']}$) we find $s' \sim_{t[v']} t'$, then $s \downarrow_{\succ_{t[v']}} t$, and so $s \downarrow_{\succ_{t[v^T]}} t$ for all total $\succeq_v^T \supseteq \succeq_v'$. Now at this point we must add back to the queue a set of preorders $\succeq_{v\,i}''$ such that: for all total $\succeq_v^T \supseteq \succeq_v$, either $\succeq_v^T \supseteq \succeq_v'$ (proven to be $\downarrow$) or $\succeq_v^T \supseteq$ some $\succeq_{v\,i}''$ (added to $V$ to be checked). For efficiency, we would also like for there to be no overlap: no total $\succeq_v^T \supseteq \succeq_v$ is an extension of more than one of $\{\succeq_v', \succeq_{v\,1}'', \ldots\}$.

This is true because of Theorem 5.1. So we add $\{\langle \succeq_{v\,i}'', s^r, t^r, c^r \rangle \mid \succeq_{v\,i}'' \in \mathrm{order\_diff}(\succeq_v, \succeq_v')\}$ to $V$, where $c^r = c \setminus (\text{if } s^r \neq s \text{ then } \{\mathsf{L}\} \text{ else } \{\}) \setminus (\text{if } t^r \neq t$ then $\{\mathsf{R}\}$ else $\{\})$. Note also that $s \downarrow_{\succ_{t[v]}} s^r$ and $t \downarrow_{\succ_{t[v]}} t^r$, therefore also $s \downarrow_{\succ_{t[vi'']}} s^r$ and $t \downarrow_{\succ_{t[vi'']}} t^r$ if $\succeq_{v\,i}'' \supset \succeq_v$.

During this whole process, any rewrites must pass a completeness test mentioned previously, such that the conditions in the definition of $\downarrow$ hold. Let $s_0, t_0$ be the original terms and $s, t$ be the ones being rewritten and $c$ the completeness flag. If the rewrite is at a strict subterm position, it succeeds by Definitions 4.2 and 4.3. If the rewrite is at the top, then we check $c$. If $\mathsf{L}$ is unset ($\mathsf{L} \notin c$), then either $s \preceq s_0 \prec t_0$ or $s \prec s_0$ or the clause is not a positive unit, so we allow a rewrite at the top of $s$, again by Definitions 4.2 and 4.3. If $\mathsf{L}$ is set ($\mathsf{L} \in c$), then an explicit check must be done: we allow a rewrite at the top of $s$ ($= s_0$) iff it is

[6]Note that the other direction may not always hold, there are strongly ground joinable terms which are not detected by this method of analysing all preorders between variables, e.g. $f(x, g(y)) \Downarrow f(g(y), x)$ wrt. $S = \{f(x,y) \approx f(y,x)\}$.

done by $l\sigma \to r\sigma$ with $l\sigma \sqsupset l$ or $r\sigma \prec t_0$. Respectively for R, with the roles of $s$ and $t$ swapped.

In short, we have shown that if $\langle \succeq_v, s', t', c' \rangle$ is popped from $V$, then $V$ is only ever empty, and so the algorithm only terminates with “Success”, if $s' \downarrow_{\succ_{t[v^T]}} t'$ for all total $\succeq_v^T \supseteq \succeq_v$. Since $V$ is initialised with $\langle \emptyset, s, t, c \rangle$, then the algorithm only returns “Success” if $s \downarrow_{\succ_{t[v^T]}} t$ for all total $\succeq_v^T$. □

## Orienting via extension of variable ordering

In order to apply the ground joinability algorithm we also need a way to check, for a given $\succ_t$ and $\succeq_v$ and some $s, t$, whether there exists a $\succeq_v' \supset \succeq_v$ such that $s \succ_{t[v']} t$. Here we show how to do this when $\succ_t$ is a Knuth-Bendix ordering (KBO) [Knuth and Bendix 1970].

Recall the definition of KBO. Let $\succ_s$ be a partial order on symbols, $w$ be an $\mathbb{N}$-valued weight function on symbols and variables, with the property that $w(x) = m$ for all variables $x$, $w(c) \geq m$ for all constants $c$, and there may only exist one unary symbol $f$ with $w(f) = 0$ and in this case $f \succ_s g$ for all other symbols $g$. For terms, their weight is $w(f(s_1, \dots)) = w(f) + w(s_1) + \cdots$. Let also $|s|_x$ be the number of occurrences of $x$ in $s$. Then

$$f(s_1, \dots) \succ_{\text{KBO}} g(t_1, \dots) \quad \text{iff} \quad \begin{cases} \text{either } w(f(s_1, \dots)) > w(g(t_1, \dots)), \\ \text{or } w(f(s_1, \dots)) = w(g(t_1, \dots)) \\ \quad \text{and } f \succ_s g, \\ \text{or } w(f(s_1, \dots)) = w(g(t_1, \dots)) \\ \quad \text{and } f = g, \\ \quad \text{and } s_1, \dots \succ_{\text{KBOlex}} t_1, \dots; \\ \text{and } \forall x \in \mathcal{X}.\ |f(\dots)|_x \geq |g(\dots)|_x. \end{cases} \tag{5.6a}$$

$$f(s_1, \dots) \succ_{\text{KBO}} x \quad \text{iff} \quad |f(s_1, \dots)|_x \geq 1\,. \tag{5.6b}$$

$$x \succ_{\text{KBO}} y \quad \text{iff} \quad \bot\,. \tag{5.6c}$$

The conditions on variable occurrences ensure that $s \succ_{\text{KBO}} t \Rightarrow \forall \theta.\ s\theta \succ_{\text{KBO}} t\theta$.

When we extend the order $\succ_{\text{KBO}}$ with a variable preorder $\succeq_v$, the starting point is that $x \succ_v y \Rightarrow x \succ_{\text{KBO}[v]} y$ and $x \sim_v y \Rightarrow x \sim_{\text{KBO}[v]} y$. Then, to ensure that all the properties of a simplification order (including the one mentioned above) hold, we arrive at the following definition (similar to [Avenhaus, Hillenbrand and Löchner 2003]).

$$f(s_1,\ldots) \succ_{\mathrm{KBO}[v]} g(t_1,\ldots) \quad \text{iff} \quad \begin{cases} \text{either } w(f(\ldots)) > w(g(\ldots)), \\ \text{or } w(f(s_1,\ldots)) = w(g(t_1,\ldots)) \\ \quad \text{and } f \succ_s g, \\ \text{or } w(f(s_1,\ldots)) = w(g(t_1,\ldots)) \\ \quad \text{and } f = g, \\ \quad \text{and } s_1,\ldots \succ_{\mathrm{KBO}[v]_{\mathrm{lex}}} t_1,\ldots; \\ \text{and } \forall x \in \mathcal{X}.\ \sum_{y \succeq_v x} |f(\ldots)|_y \\ \quad \geq \sum_{y \succeq_v x} |g(\ldots)|_y. \end{cases} \tag{5.7a}$$

$$f(s_1,\ldots) \succ_{\mathrm{KBO}[v]} x \quad \text{iff} \quad \exists y \succeq_v x.\ |f(s_1,\ldots)|_y \geq 1\,. \tag{5.7b}$$

$$x \succ_{\mathrm{KBO}[v]} y \quad \text{iff} \quad x \succ_v y\,. \tag{5.7c}$$

To check whether there exists a $\succeq'_v \supset \succeq_v$ such that $s \succ_{\mathrm{KBO}[v']} t$, we need to check whether there are some $x{\succ}y$ or $x{=}y$ relations that we can add to $\succeq_v$ such that all the conditions above hold (and such that it still remains a valid preorder). Let us denote "there exists a $\succeq'_v \supset \succeq_v$ such that $s \succ_{\mathrm{KBO}[v']} t$" by $s \succ_{\mathrm{KBO}[v,v']} t$. Then the definition is

$$f(s_1,\ldots) \succ_{\mathrm{KBO}[v,v']} g(t_1,\ldots) \quad \text{iff} \quad \begin{cases} \text{either } w(f(\ldots)) > w(g(\ldots)), \\ \text{or } w(f(s_1,\ldots)) = w(g(t_1,\ldots)) \\ \quad \text{and } f \succ_s g, \\ \text{or } w(f(s_1,\ldots)) = w(g(t_1,\ldots)) \\ \quad \text{and } f = g, \\ \quad \text{and } s_1,\ldots \succ_{\mathrm{KBOlex}} t_1,\ldots; \\ \text{and } \exists x_1, y_1, \ldots \\ \quad \succeq'_v = (\succeq_v \cup \{\langle x_1, y_1\rangle, \ldots\})^+ \text{ is a preorder} \\ \quad \text{such that } \forall x \in \mathcal{X}.\ \sum_{y \succeq'_v x} |f(\ldots)|_y \\ \quad \geq \sum_{y \succeq'_v x} |g(\ldots)|_y. \end{cases} \tag{5.8a}$$

$$f(s_1,\ldots) \succ_{\mathrm{KBO}[v,v']} x \quad \text{iff} \quad \begin{cases} \exists y \not\prec_v x.\ |f(s_1,\ldots)|_y \geq 1\,, \\ \quad \text{with } \succeq'_v = \succeq_v \cup \{x{\succ}y\} \\ \quad \text{or } \succeq'_v = \succeq_v \cup \{x{=}y\}\,. \end{cases} \tag{5.8b}$$

$$x \succ_{\mathrm{KBO}[v,v']} y \quad \text{iff} \quad \begin{cases} x \not\prec_v y \\ \quad \text{with } \succeq'_v = \succeq_v \cup \{x{\succ}y\}\,. \end{cases} \tag{5.8c}$$

This check can be used in Algorithm 1 for finding extensions of variable orderings that orient equations for rewriting.

## 5.2 Saturation strategy

We have explored a large amount of simplification rules in Chapter 4. Together with many more described in the ample literature published on the superposition calculus over the decades, we have an abundance of simplification rules and redundancy criteria at our disposal for usage in superposition theorem proving. While only the three generating rules in (2.7) are necessary for refutational completeness, simplification rules are crucial for practical performance; intuitively, simplification rules are beneficial to taming the growth of the search space as more clauses get generated.

It could seem that exhaustive application of every simplifcation rule at hand would be the right option, but naturally the computation of those simplifications *itself* takes a non-negligible amount of time,[7] so while these will often reduce the number of clauses to be considered, being too eager in applying them will also grind the prover to a halt. It is an open problem what the optimal strategy to balance these conflicting requirements is, and although there is a huge amount of flexibility in how to perform simplifications,[8] most provers are rather restrictive on this matter. In Duarte and Korovin [2020] we developed a flexible simplification setup that subsumes and generalises most common given-clause loop architectures.

Recall the algorithm for saturation using a standard given-clause loop [Wos, Robinson and Carson 1965; McCune 1990]. This algorithm is near-universally employed in theorem proving using saturation, from resolution to superposition to equational completion [Denzinger, Kronenburg and Schulz 1997; Hillenbrand, Buch et al. 1997; Schulz 2002; McCune 2003; Weidenbach et al. 2009; Korovin 2013; Waldmann et al. 2020; Smallbone 2021].

The clause set is split into an **active set**, where all generating inferences among clauses have been performed, and a **passive set**, of clauses waiting to participate in inferences. Clauses are initially added to the passive set, then in each iteration one given clause is picked from the passive set (according to some clause selection criterion), all inferences between the given clause and the active set are performed, and the given clause is added to the active set. Newly derived clauses are pushed into the passive set, and the process is repeated. The loop finishes when all clauses have been moved to the active set with no passive

[7]Empirically, a majority of time in fact.

[8]See e.g. Waldmann et al. [2020] for an abstract treatment of saturation theorem proving.

clauses remaining, meaning the active set is saturated (and therefore the initial set is satisfiable), or when a contradiction is derived, meaning the initial set is unsatisfiable.

## Flexible simplification setup

Since the choice of how the simplification rules and the redundancy criteria are performed can *dramatically* impact the performance of the solver, care is needed in making these choices, and tuning this part of the solver can pay off significantly. There is a significant amount of choice in how to perform simplifications. We can choose

- which simplifications to perform,
- at what times,
- and with respect to which clauses;

additionally, some of these simplifications require auxiliary data structures (here referred to generally as "indices") to be performed efficiently, and some indices support several simplification rules.[9] Therefore we also need to choose which clauses to add to which indices at which stages.

For example, Otter-style loops [McCune 1990] perform simplifications on clauses before adding them to the passive set. The problem with this is that the passive set grows very fast, and is often orders of magnitude larger than the active set, therefore (i) performance will degrade significantly as this set grows, as modification and retrieval operations on indices become slower the more items are stored, and (ii) the system will spend much of its time performing simplifications on clauses that may not even end up being activated.

On the other hand, DISCOUNT-style loops [Denzinger, Kronenburg and Schulz 1997][10] perform simplifications *only* with clauses that have been added to the active set, the reasoning being that removing or simplifying clauses currently participating in generating inferences is much more critical than doing so on clauses awaiting in a passive set. This has the benefit of reducing the time spent in simplifications, at the cost of potentially missing many valuable sim-

[9]For instance, if subsumption is implemented using a feature-vector or fingerprint index (or some such data structure) of literals in clauses, then subsumption-resolution (sometimes called reduction or simplify-reflect in the literature) can also be implemented for free on the same index [Graf 1995; Schulz 2013a].

[10]As used in several high-profile provers since, e.g. E [Schulz 2013b] or Twee [Smallbone 2021].

plifications wrt. passive clauses, some of which could shorten the proof search process considerably. It is not clear where the "sweet spot" is, in terms of these setups, or even if there is one (for all problems).

However, it is possible, for example, to choose to apply only "cheap" simplifications to the full active + passive set (e.g. subset subsumption[11] or light normalisation[12]), and use more expensive ones only on the much smaller active set (e.g. full subsumption or ground joinability), where simplifications are also much more valuable as they prevent generating inferences from taking place. This offers a better compromise than either of the standard approaches listed above, and is the main idea of our novel algorithm, presented in Algorithm 2.

## Immediate simplification set

We introduce into this process the element of **immediate simplification**. The intuition is as follows.

- Clauses that are derived in each loop are "related" to each other. Often, for example, some of the conclusions are subsumed by others. It may be beneficial to keep the set of immediate conclusions inter-simplified or inter-reduced with respect to a certain set of inference rules.

- Also, throughout the execution of the program, the set of generated clauses in each loop remains small compared to the set of passive or active clauses. Therefore, we can get away with applying more expensive rules that we do not necessarily want to apply on the set of all clauses (e.g. only "cheap" simplifications between newly derived clauses and passive clauses, but more expensive "full" simplifications among newly derived clauses themselves).

- Finally, during this process, it is possible that the given clause itself becomes redundant (e.g. subsumed by one of its children). If this happens, we can add *only* the clauses responsible for making it redundant to the

[11]A variant of subsumption where substitutions are not considered; it becomes equivalent to string prefix searching, quickly computable in e.g. in $\mathcal{O}(\text{length of clause})$ with a trie, in the forward and backward directions.

[12]A variant of demodulation where only a limited fixed number of instances is considered for matching, efficiently implementable in the forward direction with a hash-table, with application becoming $\mathcal{O}(1)$ regardless of the number of equations in the set [Duarte and Korovin 2020].

passive set, then remove the given clause from the active set, and throw away the rest of the iteration's newly generated clauses, abort the rest of the iteration, and proceed to the next given clause. In some problems, a significant number of iterations may be aborted, which means that fewer new clauses are added to the passive queue, and that we avoid the work of computing those inferences. This can be seen as a variant of orphan elimination [Schulz 2002]. Even when the given clause is not eliminated it is often beneficial to extensively inter-simplify immediate descendants of the given clause.

### Algorithm

We present the iProver given-clause saturation loop for superposition, a novel algorithm which incorporates the ideas of a flexible simplification setup and of immediate simplification, described above. It is presented in Algorithm 2, in terms of the following.

A **simplification set** consists of a collection of indices, each of which supports one or more simplification rules, either in the **forward** direction, meaning a clause is simplified/deleted wrt. an existing set of clauses, and/or in the **backward** direction, meaning clauses in an existing set are simplified/deleted wrt. another clause.

In our given clause loop we have three such sets:

- $S_{\text{passive}}$, the passive set,
- $S_{\text{active}}$, the active set, and
- $S_{\text{immed}}$, the newly derived clauses (this set is cleared at the end of every given-clause iteration, and the non-redundant clauses added to the passive queue).

Each set supports the following operations:

- "Add clause $C$ to set $S$ in indices $R$", which adds a clause to some given indices in a specific set,
- "Forward simplify clause $C$ wrt. set $S$ via simplification rules/criteria $R$", which applies the rules in specification $R$, using clauses indexed in the corresponding indices in $S$, to attempt to simplify clause $C$, and

**Algorithm 2:** iProver given clause saturation loop.

```
Input: $I$: set of input clauses
$S_{\text{passive}} \leftarrow \emptyset$
for $C$ in $I$ do
    $C \leftarrow$ simplify $C$ wrt. $S_{\text{passive}}$ via $R^{\text{fw}}_{\text{input}}$
    if $C$ is deleted then continue
    $S_{\text{passive}} \leftarrow$ simplify $S_{\text{passive}}$ wrt. $C$ via $R^{\text{bw}}_{\text{input}}$
    $S_{\text{passive}} \leftarrow \{C\} \cup S_{\text{passive}}$ on indices $R^{\text{index}}_{\text{input}}$
end
$S_{\text{active}} \leftarrow \emptyset$
loop                                   ▷ until $S_{\text{passive}}$ empty or contradiction found
    ${S_{\text{passive}}}' = \{G\} \uplus S_{\text{passive}}$        ▷ $G$ chosen by clause selection heuristic
    $S_{\text{passive}} \leftarrow {S_{\text{passive}}}'$
    $G \leftarrow$ simplify $G$ wrt. $S_{\text{active}} \cup S_{\text{passive}}$ via $R^{\text{fw}}_{\text{active}}$
    if $G$ is deleted then continue
    $S_{\text{active}} \leftarrow$ simplify $S_{\text{active}}$ wrt. $G$ via $R^{\text{bw}}_{\text{active}}$
    $S_{\text{immed}} \leftarrow \{G\}$
    for $C$ in generating inferences between $G$ and $S_{\text{active}}$ do
        $C \leftarrow$ simplify $C$ wrt. $S_{\text{immed}}$ via $R^{\text{fw}}_{\text{immed}}$
        $C \leftarrow$ simplify $C$ wrt. $S_{\text{active}} \cup S_{\text{passive}}$ via $R^{\text{fw}}_{\text{passive}}$
        if $C$ is deleted then continue
        $S_{\text{immed}} \leftarrow$ simplify $S_{\text{immed}}$ wrt. $G$ via $R^{\text{bw}}_{\text{immed}}$
        $S_{\text{passive}}, S_{\text{active}} \leftarrow$ simplify $S_{\text{active}} \cup S_{\text{passive}}$ wrt. $G$ via $R^{\text{bw}}_{\text{passive}}$
        if $G \notin S_{\text{immed}}$ (simplified by clauses $U$) then
            $S_{\text{passive}} \leftarrow U \cup S_{\text{passive}}$
            continue
        end
        $S_{\text{immed}} \leftarrow \{C\} \cup S_{\text{immed}}$ on indices $R^{\text{index}}_{\text{immed}}$
    end
    $S_{\text{passive}} \leftarrow S_{\text{immed}} \cup S_{\text{passive}}$ on indices $R^{\text{index}}_{\text{passive}}$
    $S_{\text{active}} \leftarrow \{G\} \cup S_{\text{active}}$ on indices $R^{\text{index}}_{\text{active}}$
end
```

- "Backward simplify set $S$ wrt. clause $C$ via simplification rules/criteria $R$", which applies the rules in $R$, using $C$, to attempt to simplify clauses indexed in $S$.

These are called at several points in the loop (see the pseudocode in Algorithm 2), and the user can configure which indices/rules (the $R$'s) are involved in each operation.

Furthermore, the possibility of manual tuning by the user is not the only objective. In particular, iProver includes an auto-schedule, which has been tuned via machine learning [Holden and Korovin 2021].[13] The flexibility afforded by this novel given-clause algorithm enabled the machine learning process to discover many interesting parameter sets, and combinations thereof, that (ideally) maximise the number of problems solved.

Note also that generally, when during immediate simplification a parent clause of a newly derived clause is made redundant, we can remove all the children of that clause from the immediate set (and thus avoid adding them to the passive queues), except for the ones which caused it to be redundant. Currently, we restrict this feature to the given clause rather than to all the parent clauses, therefore, this simplifies to checking whether the given clause is made redundant in $S_{\text{immed}}$, and if so abort the loop, add only the clauses that made it redundant to the passive set, and remove the given clause.

## Implementation

Currently iProver uses non-perfect discrimination trees for implementing unification for generating inferences [Graf 1995], perfect discrimination trees for matching for demodulation, ground joinability, and unit subsumption [Graf 1995], hash-tables for light normalisation, feature vector indices for subsumption and subsumption resolution [Schulz 2013a], tries for subset subsumption [Graf 1995], the MiniSat solver [Eén and Sörensson 2003] for global propositional subsumption, and the Z3 solver [Moura and Bjørner 2008] for canonicalisation and instantiation of arithmetic terms.

As alluded several times before, choosing an appropriate term ordering is sometimes the difference between finishing in a fraction of a second, or running

[13]This approach uses a mixture of dynamic clustering (to classify problems, since different problems require drastically different heuristics), and hyperparameter optimisation (to optimise the heuristics and schedules thereof, for a given cluster).

out of time/memory before finding a solution. We also know that certain theories have a “natural” ordering which matches the intuitive direction in which rewrites should tend to be performed. For example, problems in the theory of rings often benefit from an LPO ordering where $0 \prec 1 \prec + \prec \times \prec -$, which orders axioms in a way that matches the intuitive direction that a human would most often choose to apply them, for instance rewriting terms into sums of monomials (e.g. $-(a \times (b+c)) \to -(a \times b) + -(a \times c)$). This is despite the fact that, overall, KBO outperforms LPO most often than not [Schulz 2002; Löchner 2007].

Therefore we have also incorporated a simple scheme for theory detection in iProver for many common algebraic theories [Hillenbrand, Jaeger and Löchner 1999; Cruanes 2015]. In summary: first, for a given theory, exhaustive testing is used to establish empirically an ordering (or combination of orderings) that results in the highest success rate, among a library of pre-existing problems in that theory; these orderings are recorded in a database embedded in iProver. Then, in the preprocessing phase of iProver, syntactic detection of algebraic theories in the input problem takes place, which triggers the usage of the appropriate ordering (and/or other configurations).

# Chapter 6

# Experimental results

*Co'um saber só d'experiencias feyto*
*Tais palauras tirou do experto peito:*

Luís de Camões, *Os Lusíadas* (1572)

In this chapter, we discuss experimental results obtained after implementing the research described in the preceding chapters in a state-of-the-art theorem prover.

## iProver

iProver[1] [Korovin 2008; 2013; Duarte and Korovin 2020] is an automated theorem prover for first-order logic. It supports unsorted and many-sorted first-order logic, as well as interpreted integer, rational, and real arithmetic. Originally based on the Inst-Gen calculus, it now has support for running any number of Inst-Gen, superposition, and resolution components, in sequence or concurrently. These calculi can be interleaved in a time-sliced fashion, and clauses and other information can be shared between components.[2] It supports

[1]Available at `http://www.cs.man.ac.uk/~korovink/iprover/`.

[2]For example, superposition can easily derive lemmas which Inst-Gen has a hard time obtaining, and vice-versa, so both calculi can, for instance, commit clauses to a shared propositional solver, which can be accessed by any other component for usage in global subsumption (see Korovin 2013 for an example of this); or a superposition component can query a previous superposition component for the clauses which were more useful for simplifications during the former run (and in the contrary, those which were more prolific for generation and less suitable for simplification), and adjust their position in the clause selection heuristic accordingly.

sophisticated preprocessing, but depends on external tools for clausification of non-clausal form problems into CNF. It also implements a general abstraction-refinement framework [López-Hernández and Korovin 2018].

There are hundreds of options that control the behaviour of the prover, from search parameters, term orderings, to the simplification and inference rules, or the scheduling of different calculi. The auto-schedule depends on the input problem, and has been tuned via machine learning [Holden and Korovin 2021].

iProver is written in OCaml, with few external dependencies (some of them optional). It is free software under GNU GPL v3.

## Computational resources

The experiments in this chapter were performed on a computational cluster composed of 25 nodes, each with an AMD Opteron 4334 CPU, with 6 cores (12 virtual) at 3.1 GHz, and 128 GB of RAM. Each problem is allocated one (physical) core.

## Problem set

We have chosen the TPTP (Thousands of Problems for Theorem Provers) library [Sutcliffe 2017] as a source of test problems for our evaluation, as it is by far the most comprehensive freely-available source of first-order logic problems in a standard input format; it incorporates problems from various domains (from algebra, analysis, and geometry, to software verification and Sledgehammer[3] dumps) and various difficulty levels (from trivial to open).

Our test set is comprised of the CNF and FOF[4] portions of the TPTP v8.1.2 library, the latest available version at the time of writing. These amount to 17 436 problems.

TPTP also includes a database of solutions for each problem (called TSTP), by actively developed as well as legacy theorem provers, including superposition, resolution, equational completion, tableaux, SMT, higher-order, and other types of provers. This enables us to easily evaluate our contributions in comparison to existing provers (especially superposition and equational completion ones), including identifying problems which have no prior recorded solution. Of the 17 436 problems in our test set, 2349 have no solution in TSTP.

[3]Blanchette et al. 2016.

[4]First-order, untyped, no arithmetic.

## 6.1 Ground joinability and AC criteria

In this section we evaluate the impact of ground joinability criteria on overall problem-solving performance, after implementation in iProver. This is, to our knowledge, the first implementation of general ground joinability in a superposition prover, and also features our novel incremental ground joinability algorithm for testing this criterion.

Specifically, we are interested in testing the following aspects.

**Problems solved** We investigate whether ground joinability increases the overall number of problems solved, by measuring the impact of enabling this simplification, as compared to an otherwise identical parameter setup where this simplification is disabled. Furthermore, we also investigate which *new* problems are solved (as there naturally will be problems solved only when ground joinability is disabled and only where ground joinability is enabled, even if the overall number of problems solved increases), especially by consulting the TSTP library of solutions to identify problems which have no previous recorded solution at all, but which are solved using superposition with ground joinability, in iProver.

**Runtime** While solving more problems is usually taken to be the most important goal of theorem proving, minimising runtime for successful solutions is also an important goal (of critical importance in many "online" applications, such as embedding into higher-order provers or compilers, less so in "offline" tasks like chip verification). Therefore, we will also measure how enabling ground joinability affects the time to obtain a solution, among problems which are solved regardless of whether it is enabled or not. On one hand, ground joinability can simplify more clauses, on the other hand, it takes time to check even when it is unsuccessful; this is why it is not clear *a priori* if the impact on runtime will be positive or negative.

**Incremental ground joinability cost** Ground joinability is a very powerful, but also very heavy simplification rule. Because of this, we have spent significant effort in designing the incremental ground joinability algorithm (Section 5.1) for maximum performance and minimal runtime impact. To measure the degree to which we have been successful, we have also performed measurements of the percentage of runtime spent testing ground joinability, compared to the overall runtime of the prover.

**AC simplifications** In our treatment in Chapter 4, we have distinguished between "general" ground joinability, and the specialised AC joinability rule, which can be implemented in a more efficient way (a simple sort and compare of AC subterms, versus the intricacies of Algorithm 1). Therefore, in our experiments, we will also measure the impact of these techniques separately.

## Experimental design

As noted, iProver is currently a sophisticated system which supports running several "components" concurrently, i.e. different calculi with different strategies and options, which may also share information between them. We have performed experiments using manually defined configurations where only one superposition (and preprocessing) components are enabled. This reduces the "noise" caused by the complex (and rather inscrutable) interplay between the different components, which would be a significant confounding factor as the subtle and non-trivial interactions between the components of the auto-schedule would prevent us from drawing meaningful conclusions in terms of the impact of ground joinability on superposition performance. Furthermore, the auto-schedule is aggressively tuned to the available parameter space [Holden and Korovin 2021]; restricting this by forcing ground joinability to be on/off will distort the results. Evaluating the interplay between ground joinability and the Inst-Gen calculus is, therefore, outside the scope of the present discussion.

Therefore, in this section we have disabled the auto-schedule and ran iProver with a manual schedule of preprocessing and superposition, whose components vary the search heuristics, the simplification setup, term ordering, clause selection heuristic, among other parameters. This aims to ensure that (i) problems solved incidently by other means (in superposition, e.g. by varying the clause selection heuristic, the term ordering, or other simplification rules) will be solved by both experiments, with and without ground joinability, thereby ensuring that the differing problems will tend to be more representative of problems benefiting from ground joinability in a realistic schedule, but not solved by existing techniques, and (ii) to effectively measure the impact of applying ground joinability in a wider variety of parameter configurations.

This experiment was repeated with the following variations (with all other options being kept equal between them):

- Ground joinability simplifications enabled,
- Ground joinability simplifications disabled,
- General ground joinability disabled but AC joinability enabled.

Then, we also ran another set of experiments (for each of the three variants above) with a straight schedule consisting of a single saturation loop for the entire duration of the run (preceded by preprocessing). We call the former set of experiments "schedule on" and the latter "schedule off". The former has the advantage of being more realistic and avoiding spurious results,[5] while the latter has the advantage of introducing less noise and enabling a straight comparison between runs with and without ground joinability.

Finally, we ran a series of experiments with several different parameters and schedules — always with ground joinability enabled — in an attempt to find solutions for hard (or unsolved) problems.

All experiments ran with a time limit of 300 s per problem.

## Results

Of the 17 436 eligible problems, ground joinability was not even attempted once in 7735 of them, either because the problem does not include equality, or because it is immediately solved through very simple steps or in preprocessing. These problems are not interesting for the purposes of this analysis, and therefore we will only consider the universe of the other 9701 problems for the remainder of this section.

Of these, ground joinability is used to eliminate clauses in 4075 of them (42%, Figure 6.2a) in the "schedule on" experiment, and 3988 of them (41%, Figure 6.2b) in the "schedule off" experiment, indicating that ground joinability is applicable in many classes of problems, including in non-unit problems where it previously had never been used.

[5]As discussed, in terms of ensuring that problems which are solvable by other means, such as by slightly changing the term ordering or the clause selection heuristic, are solved in the experiment which does not include ground joinability, and do not spuriously show up in analysis as being solved by ground joinability.

Table 6.1: Number of problems solved.

| Ground joinability | Exclusive problems | Total problems |
|---|---|---|
| On | 133 | 9841 |
| Off | 27 | 9735 |

(a) Schedule on.

| Ground joinability | Exclusive problems | Total problems |
|---|---|---|
| On | 117 | 9496 |
| Off | 29 | 9408 |

(b) Schedule off.

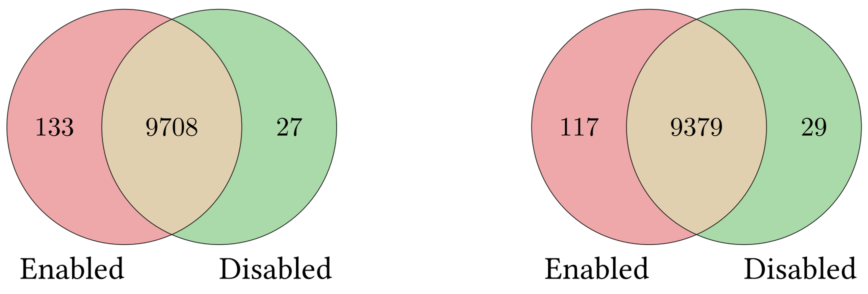
**Problems solved**

In Table 6.1, we summarise the number of problems solved in each of the experiments. We observed that turning on ground joinability enabled the solution of 133 problems which it did not solve with an identical schedule without ground joinability, for a net gain of +106 overall problems (as illustrated in Figure 6.1a). In the "schedule off" experiment, the gains were slightly more modest, as 117 more problems were solved for a net gain of +88 problems (Figure 6.1b).

We can further distinguish, from among the problems solved with ground joinability, those that are also solved only with AC joinability (and normalisation) without the need for general ground joinability. This is summarised in Figure 6.3, and indicates that while the lighter[6] AC joinability rule suffices to achieve a moderate increase in performance (mostly in problems where AC reasoning is the most significant challenge), general ground joinability is still essential for solving other types of problems, and enabling it yields the best overall performance.

Overall, and in terms of the number of problems solved, these experimental results enable us to conclude that ground joinability increases the performance of the superposition calculus, while not being a universal improvement, in the sense of solving a superset of problems compared to runs without ground

[6]In terms of implementation effort and of runtime.



(a) Schedule on. (b) Schedule off.

Figure 6.1: Number of problems solved exclusively with and without ground joinability, and problems solved in both.

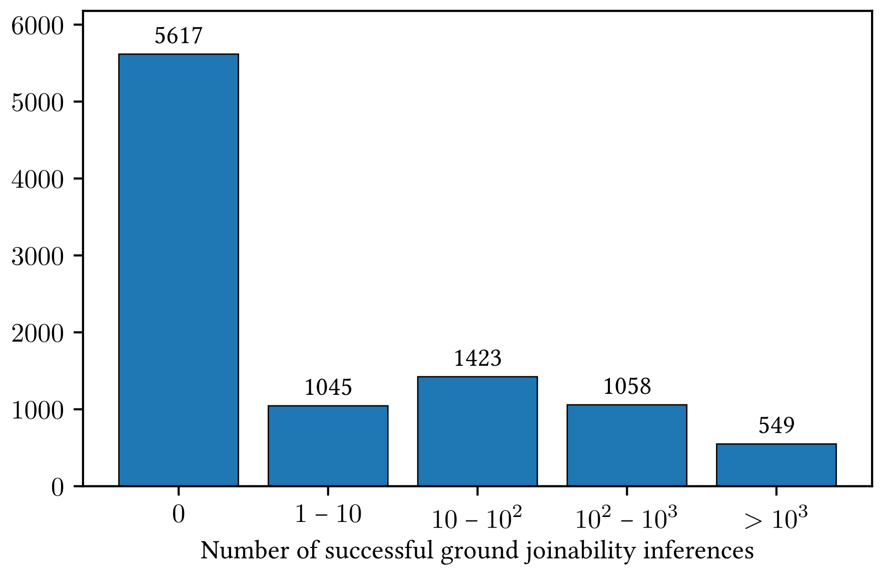
joinability. This suggests that these approaches complement each other, and that a suitable schedule is needed to maximise the number of problems solved, including components where ground joinability is both enabled and disabled.

As to the specific impact of the concrete implementation of Algorithm 1 for testing ground joinability, we cannot draw conclusions as we do not have two reference implementations to compare. Therefore, in this analysis, we cannot distinguish how much is owed to the application of ground joinability redundancy elimination in abstract, and how much is owed to this novel algorithm for testing it.

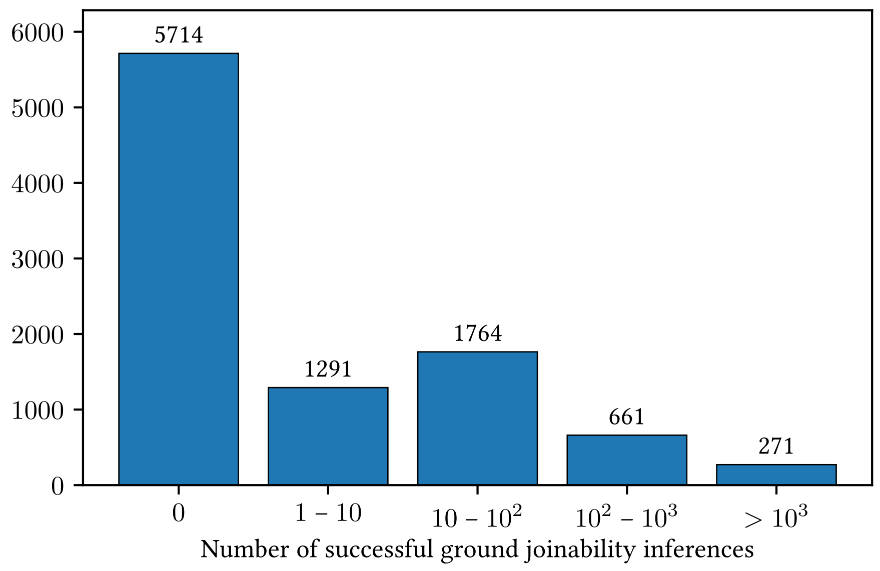
### Runtime

As mentioned, while solving problems is the priority, we have also measured runtime among the problems successfully solved by both approaches, to understand if enabling ground joinability helps reduce the time to a successful solution, even where the problem is still solved without ground joinability. We have found (Table 6.2) that there is only a very modest improvement, as the weight of ground joinability (being applied everywhere) erases — on an aggregate level — almost all of the gains from the deletion of ground joinable literals.

Note that this is an aggregate result; we can see that there are pathological problems where there is substantial speedup (or substantial slowdown). This is summarised in Table 6.3, in which we can see that there are far more problems

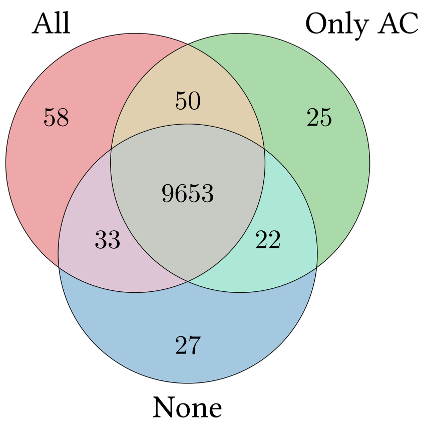


(a) Schedule on.

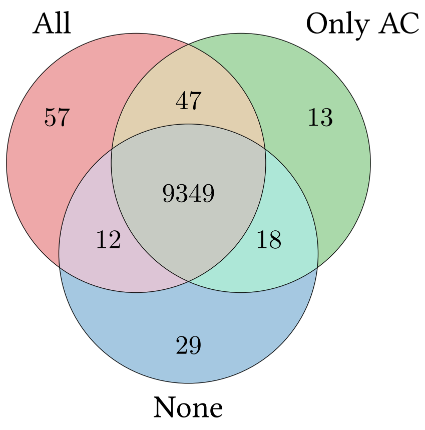


(b) Schedule off.

Figure 6.2: Number of successful ground joinability simplifications in each problem, among problems where ground joinability is attempted at least once.



(a) Schedule on.



(b) Schedule off.

Figure 6.3: Problems solved with general ground joinability, with only AC joinability, and with neither.

which are solved significantly faster with ground joinability enabled than vice-versa, even though the overall number is small (around 90% of problems solved by both approaches finished within ±30% time of each other).

Nonetheless, we consider that the fact that there is no measured slowdown among problems solved irrespectively of ground joinability is already an encouraging result, as it means we can reap the benefits (more and harder problems solved) without an unacceptable cost (significant slowdown in other problems).

A commonly-used visualisation of both runtime and number of problems solved is the graph of the cumulative number of problems solved under a given time, provided in Figure 6.4.

In terms of the runtime performance impact of enabling ground joinability, we measured the amount of time spent on ground joinability as a percentage of total runtime, for each given problem. Discarding problems which finish in under 1 s, we have plotted the results in Figure 6.5. We observe that, in the more realistic "schedule on" experiment, in 6613 out of 8281 problems (80%), the total time spent on ground joinability algorithms and data structures is less than 1% of total prover time, exceeding 20% only on 93 (1.1%) problems. In the "schedule off" experiment, the result was slightly better, with 7516 out of 8257 problems (91%) not exceeding 1% of time spent on ground joinability, and only

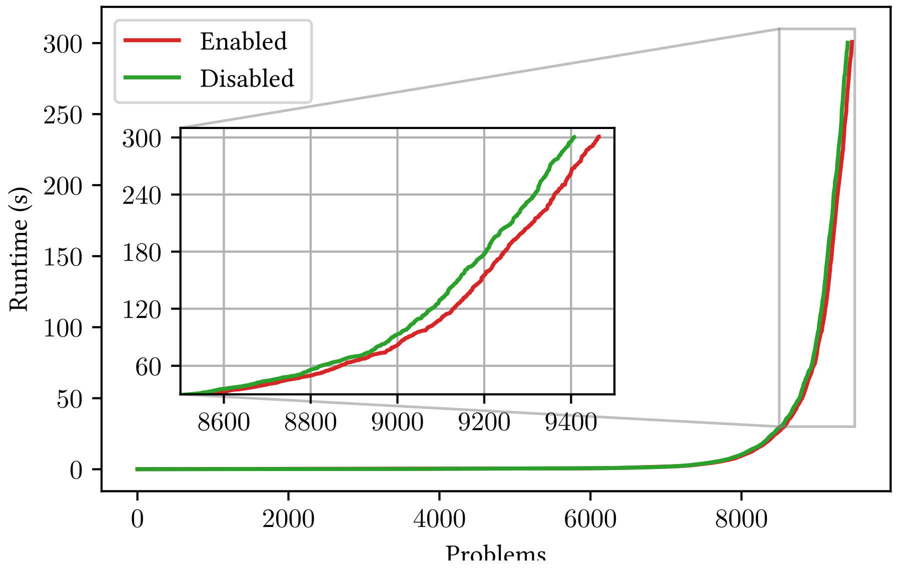


Figure 6.4: Cumulative number of problems solved under a given time, with and without ground joinability.

Table 6.2: Average runtime among problems solved with and without ground joinability.

| | Gr. join. enabled | Gr. join. disabled |
|---|---|---|
| Schedule on | 15.482 s **(−0.350 s)** | 15.832 s |
| Schedule off | 11.078 s **(−0.436 s)** | 11.514 s |

Table 6.3: Number of problems solved significantly faster with or without ground joinability, among problems solved in both experiments, and with runtime $> 1$ s (schedule off).

| | Gr. join. enabled | Gr. join. disabled |
|---|---|---|
| >30% faster | 206 (7.1%) | 87 (3.0%) |
| >60% faster | 150 (5.1%) | 29 (1.0%) |
| >100% faster | 122 (4.2%) | 19 (0.7%) |

88 (1.1%) problems going over 20%. We attribute this in part to the success of our incremental ground joinability algorithm in finishing quickly in the vast majority of cases where the two terms are *not* ground joinable.

For AC joinability, the results are even more dramatic (Figure 6.6), as the specialised test is extremely fast and does not even exceed 0.1% of total runtime in 87.2% of the problems (both experiments), for the combined application of AC joinability and AC normalisation.[7] We conclude that it pays off to specialise the ground joinability test on AC terms, especially as associative-commutative operators appear quite often in real problems (1268 in the present problem set), and that AC joinability/normalisation can almost always be enabled without fear of significant negative performance impact.

## Hard and unsolved problems

Let us also search for hard problems which are newly solved by iProver using ground joinability. TPTP classifies problems by rating in $[0, 1]$, problems with rating $\gtrsim 0.9$ being usually considered to be very challenging. We also search for problems with no recorded solution in TSTP.

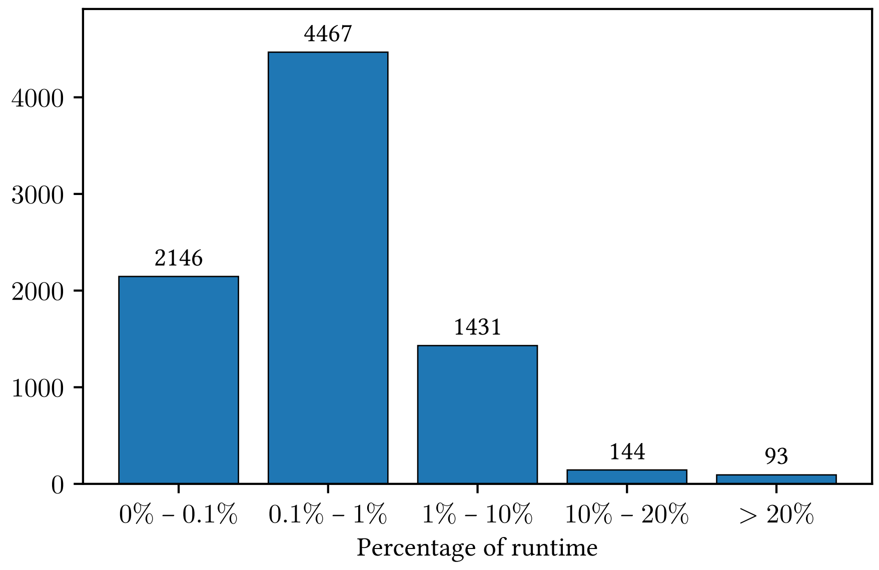
Some interesting problems solved are detailed in Table 6.4. In particular, these include problems previously solved only by unit equational provers (e.g. `LAT140-1`, `ROB018-10`, `REL045-1`, `GRP196-1`), and problems hitherto unsolved by automated theorem provers (e.g. `RNG010-1`, `RNG029-2`, `CSR205+1`).

The problems in Table 6.4 broadly include:

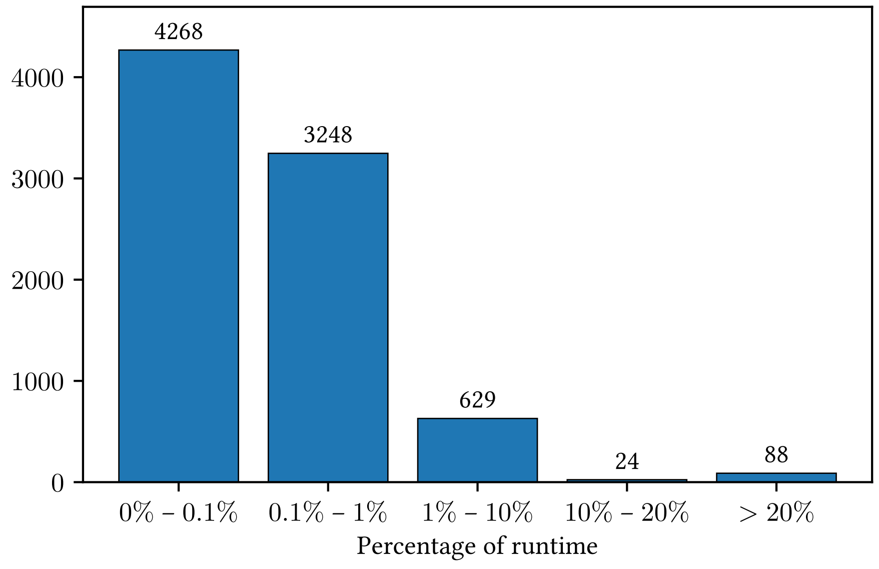
- Problems in abstract algebra, such as the problems in `RNG`, `LCL`, or `ROB` in the table, which include not only AC but also other permutative axioms,[8] at which ground joinability excels. The problem `RNG029-2`, for instance, which asks to show that one of the Moufang identities holds in any ring, was solved in only 50 s, with X successful applications of the ground joinability simplification rule, while no other prover in TSTP has a recorded solution. This is an example of what we mentioned before as motivation for our work: as it is not a unit problem, it is outside the scope of equational completion provers, but at the same time, no other first-order prover implements ground joinability.

[7] It is much faster to combine these in one pass, hence why they are counted together.

[8] Other axioms of the form $l \approx r$ where $l$ and $r$ are renamings of each other, like the commutativity axiom.

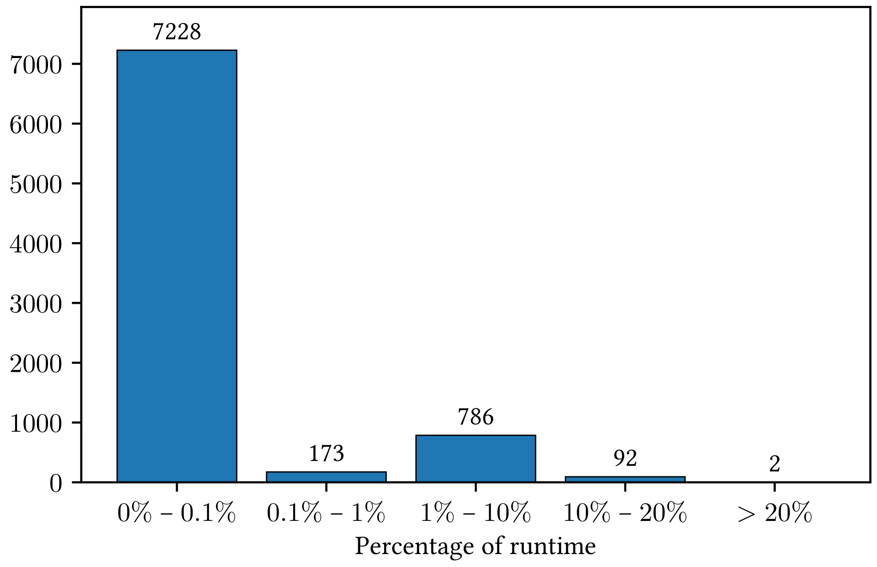


(a) Schedule on.

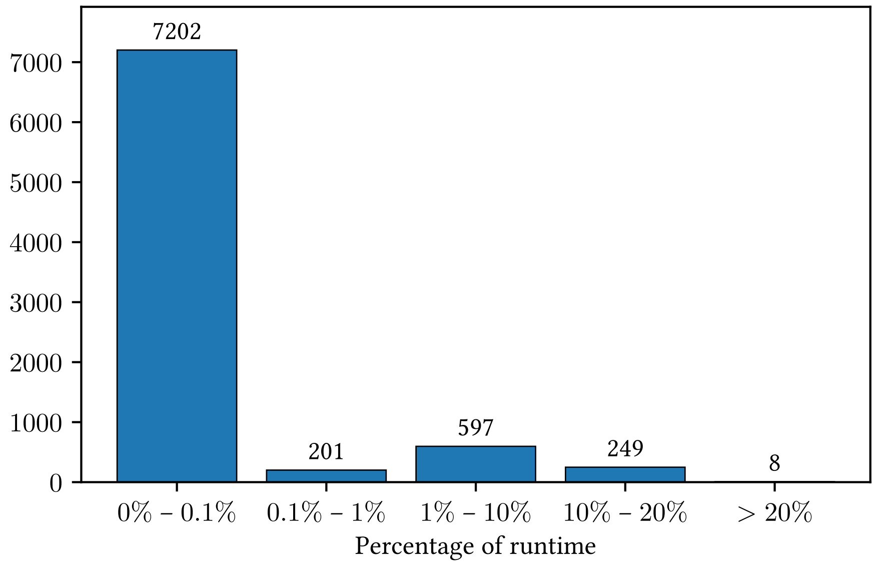


(b) Schedule off.

Figure 6.5: Percentage of total runtime spent testing general ground joinability (excl. AC), among problems where ground joinability is attempted at least once, and with total runtime $> 1$ s.



(a) Schedule on.



(b) Schedule off.

Figure 6.6: Percentage of total runtime spent testing AC joinability, among problems where ground joinability is attempted at least once, and with total runtime $> 1$ s.

Table 6.4: Hard or unsolved problems in TPTP, solved by iProver with ground joinability.

| Name | Rating | Name | Rating |
|---|---|---|---|
| LAT140-1 | 0.90 | ROB018-10 | 0.95 |
| REL045-1 | 0.90 | LCL477+1 | 0.97 |
| LCL557+1 | 0.92 | LCL478+1 | 1.00 |
| LCL563+1 | 0.92 | CSR039+6 | 1.00 |
| GRP196-1 | 0.92 | CSR040+6 | 1.00 |
| LCL474+1 | 0.94 | CSR205+1 | 1.00 |
| RNG028-2 | 0.94 | RNG010-1 | 1.00 |
| ROB018-1 | 0.95 | RNG029-2 | 1.00 |

- Problems with a large number of axioms, such as the CSR problems (which feature from thousands to millions of input clauses), all of which contain one conjecture which has a proof involving a relatively small number of those axioms. In these, the elimination of redundant formulae is critical to keep the search space manageable and thus to find those elusive proofs. However, we cannot rule out that these are "spurious" results in the sense that the application of ground joinability only coincidentally steered the proof search into the right direction.

Overall, we can conclude that ground joinability in superposition (i) increases overall performance, (ii) finds solutions to previously unsolved problems, and (iii) does not significantly impact runtime on the majority of cases.

## Encompassment demodulation

Although ground joinability is one of the main contributions of this work, the strengthening of standard demodulation into encompassment demodulation is also an important contribution to the theory and practice of the superposition calculus. However, it is difficult to measure the impact of this rule in isolation with the rest of the techniques that were implemented; this is due to the fact that (i) one of the advantages of encompassment demodulation is that indexing and application of the completeness condition is considerably simplified (and faster) — and so having an option to switch between standard and encompassment demodulation implies an inefficient implementation which negates this advantage — and (ii) the other simplification rules such as ground joinability subsume some applications of encompassment demodulation, which makes it impossible to actually measure the impact of the latter without turning all other rewrite-based rules off.

Therefore we do not have as of yet experimental data that measures the impact of encompassment demodulation and ground joinability independently of each other. This would require some amount of implementation effort in “backporting” the standard demodulation conditions onto the remaining rewrite-based simplification rules, and maintaining two separate implementations of every algorithm and data structure used in either variant. Having this data in the future may help us establish the actual practical impact of replacing standard demodulation with encompassment demodulation.

### Connectedness

Likewise, we do not have as of yet a satisfactory implementation of connectedness or ground connectedness. Implementation of the former (in its equational completion version) is known to be very challenging [Bachmair and Dershowitz 1988; Smallbone 2021], and the latter is an entirely novel contribution of this work. Intuitively, we expect that connectedness and ground connectedness complement ground joinability, but we do not currently have experimental data to verify this intuition.

## 6.2 iProver loop

In this section, we evaluate the implementation of our novel iProver given-clause loop. We compare it with two existing and widely used algorithms, as first implemented in Otter [McCune 1990] and as first implemented in Discount [Denzinger, Kronenburg and Schulz 1997];[9] in Section 5.2, we discussed these algorithms and how they compare to our novel given-clause loop. Due to being used in many very successful theorem provers, we have chosen them to benchmark our own contribution against.

[9]These algorithms have been used in many theorem provers besides; we name them thus for the first provers which implemented them.

The experimental setup is as follows. We run iProver with three distinct sets of options, identical except for the simplification setup.

- In the *Otter* experiments, we do all simplifications at clause creation time, with respect to all other clauses, in the forward and backward directions.
- In the *Discount* experiments, we do the same simplifications in the same order, but only among *activated* clauses.
- In the *iProver* loop, we perform "light" simplifications (subset subsumption, light normalisation [Duarte and Korovin 2020], etc.) among all clauses, at clause derivation time, and "heavy" simplifications (demodulation, ground joinability, subsumption resolution [Schulz 2002; Kovács and Voronkov 2013], etc.) among active clauses, at clause activation time; we also apply immediate simplification (see page 106) using subsumption and subsumption resolution, as well as attempting backward demodulation on the given clause via the derived clauses. We detail this in the appendix.

In all three experiments, the remaining options are identical. The auto-schedule is disabled and we run only one superposition calculus; i.e. we only run a single saturation loop for 300 s with the given options. Again, note that this is significantly less powerful than using a complex schedule of many components, but it ensures we are only measuring the impact of the superposition saturation loop setup, rather than having the results being confounded by the many subtle and non-trivial interactions between the components of the auto-schedule.

The results for the number of solved problems are summarised in Table 6.5. We can see that "Discount" leads "Otter",[10] but crucially, our novel simplification setup leads both of them, improving on Discount (as measured) about as much as Discount improves on Otter. The "iProver" loop solves 202 more problems than Discount and 397 more problems than Otter. An interesting fact is that these problems are mostly of increased difficulty: the average rating of problems solved *exclusively* by iProver is 0.546, compared to the average rating of 0.147 for problems solved by all three.

The distribution of problems solved by each of these approaches is also summarised in Figure 6.7. Note that, as expected, the majority of problems are solved by all three approaches, if given 300 s to complete.

[10] This is in line with "folklore", see e.g. Schulz [2002, 2009].

As mentioned before, the most important metric that can be used to judge a theorem prover's success is usually taken to be solving more problems. However, we also measured the average time taken per problem solved[11] (Table 6.6). We can observe that our novel simplification loop also succeeded in reducing the average time for a successful solution, from 8.111 s in Otter and 6.144 s in Discount to 4.802 s in our approach (−22% compared to Discount).

## 6.3 CADE ATP System Competition

Finally, we will note that iProver runs in the annual CASC competition of automated theorem provers [Sutcliffe 2016]. In 2021, after implementation of AC joinability and normalisation, of the iProver given clause loop, and of meta-optimisation of the auto-schedule [Holden and Korovin 2021], iProver came in 2nd place on the main FOF division[12] of the competition, narrowly edging out E [Schulz 2013b], a theorem prover with over 25 years of continuous and active development. In 2022, iProver came in 3rd place, after E re-gained the lead (the Vampire theorem prover [Kovács and Voronkov 2013] won the division in both years).

[11] Among all problems solved in common, since problems solved exclusively in one or two experiments tend to be unrepresentatively difficult.

[12] First-order theorems.

| Saturation loop | Solved problems | |
|---|---|---|
| Otter | 9068 | **(+0)** |
| Discount | 9263 | **(+195)** |
| iProver | 9465 | **(+397)** |

Table 6.5: Total number of problems solved by each loop.

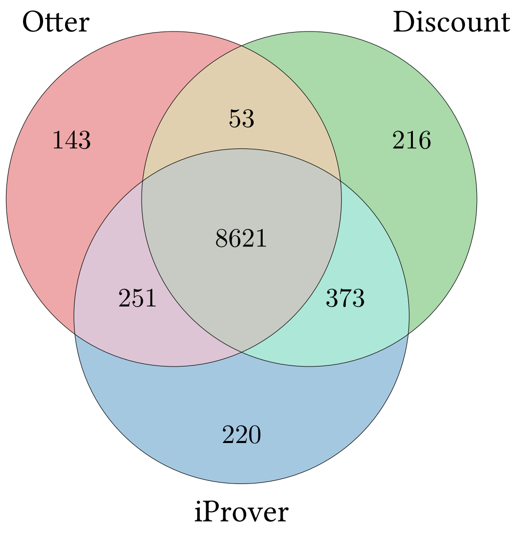


Figure 6.7: Number of problems solved by each combination of loops.

| Saturation loop | Avg. runtime (s) | |
|---|---|---|
| Otter | 8.111 | **(−0)** |
| Discount | 6.144 | **(−1.967)** |
| iProver | 4.802 | **(−3.309)** |

Table 6.6: Average runtime among problems solved by all loops.

*Blank page*

*Blank page*

# Chapter 7

# Conclusion and outlook

‘Don’t adventures ever have an end?’ Bilbo said, ‘I suppose not. Someone else always has to carry on the story.’

J.R.R. Tolkien, *The Lord of the Rings* (1954)

In this work, we have extended the state of the art of theoretical treatments of the superposition calculus, by means of a novel notion of redundancy involving closures and of a definition of closure orderings. These theoretical advances yielded a plethora of further theoretical and practical results, namely several simplification rules for the superposition calculus, such as encompassment demodulation, ground joinability, connectedness, AC normalisation, and others. We also contributed two novel algorithms which address relevant issues vis-à-vis this work, namely checking ground joinability of two terms, and controlling the search and the application of simplifications in a saturation procedure. Finally, we have implemented most of our work in iProver, and analysed its practical performance.

In particular, we have shown that simplification by rewriting with oriented instances of equations, used in unfailing completion, can coincide *exactly* with the equivalent notion in superposition, rather than the latter being more restrictive — and in a similar fashion, simplification rules like ground joinability and critical pair criteria like connectedness *can* also carry over to superposition unchanged — thereby bridging a frustrating gap that stood for decades, and unifying the theory of equational completion and superposition. This is a task that can be said to have begun in the 1970s with paramodulation and Knuth-

Bendix completion [Bonacina 2022], whose development eventually produced the superposition calculus [Bachmair and Ganzinger 1994], which as noted *did* completely generalise the generating rule of unfailing equational completion to the non-unit case, but *did not* generalise its notion of simplification or redundancy.

Furthermore, our formulation of simplification rules such as encompassment demodulation or ground joinability not only coincides with equational completion on unit equations, but also extends cleanly to the non-unit, non-equality, and negative equality cases. For example, encompassment demodulation allows rewriting with an oriented unit equation to succeed on other unit equations exactly whenever it would succeed in equational completion, and to succeed on negative or non-unit or non-equational clauses at all times.

This opens the door to bringing the most successful techniques from equational completion to bear fruit on the superposition calculus, including — for the first time — on non-unit problems. As discussed, techniques like ground joinability are part of the reason why equational completion outperforms superposition on unit-equational problems, and with this work they are now available in superposition provers, for application on non-unit-equational problems. Indeed, we identified already some hard problems, with no recorded solutions by automated theorem provers, which were solved by our implementation. Furthermore, we are aware of state-of-the-art projects that have already adopted some of the techniques in this dissertation, and implemented them in their own theorem provers.

We also studied key aspects related to algorithms necessary for a practical realisation of these results, contributing algorithms for ground joinability, and for performing simplifications in a given-clause loop. The former algorithm aims to improve the efficiency of this very costly simplification rule, by optimising the most common case, and attempting to reduce the number of rewrite steps performed (which involve expensive matching, ordering checks, etc.). The latter algorithm subsumes some of the most widely-used given-clause loop setups into a more flexible algorithm that outperforms them.

Further to this, we have also implemented the most significant practical contributions presented in this dissertation into a robust and mature theorem prover for first-order logic, iProver. This has enabled us to demonstrate empirically some of the practical performance gains we claimed as a result of this work, namely that ground joinability, AC joinability, AC normalisation, and encompassment demodulation help improve the overall number of problems solved, that they unlock solutions for very hard or hitherto unsolved problems, and that runtime efficiency is not affected negatively. We have also shown that our novel given-clause loop outperformed the two given-clause algorithms which are most widely used in superposition provers.

## Future work

As mentioned, the theoretical unification embodied in this work opens the door to further work which builds on top of it. There are also many theoretical and practical questions that remain to be addressed.

For instance, while we have contributed a novel algorithm for ground joinability, connectedness is an equally challenging rule to implement. Current approaches are limited and based on rough heuristics on a best-effort basis, and we believe that there is significant room for improvement on this front. Namely, a unified algorithm for treating all rewrite-based simplification rules — from demodulation to joinability and ground joinability to connectedness and ground connectedness — could introduce significant benefits as intermediate results are cached and repeated work is avoided.

Backward ground joinability is another aspect which is challenging to get right. While we contributed an algorithm for checking ground joinability of two terms under a fixed set of equations, whose natural application is in checking whether two sides of a literal of a new clause are ground joinable wrt. a kept set of equations (i.e. a forward simplification), checking whether a new equation makes two sides of a literal in a kept clause ground joinable (i.e. a backward simplification) is significantly more difficult. Without any further improvement, a naïve approach would require prohibitive amounts of memory for storing the intermediate state of all failed ground joinability attempts (see Algorithm 1), or simply re-doing all the work on all kept clauses for each backward simplification attempt, which is also highly inefficient. Further work on this front is needed, as this is a significant weakness in our implementation: since only one of the directions is available, success in ground joinability may be heavily dependant on the order in which clauses are derived/activated (and thus considered for simplification by ground joinability).

As remarked in the previous chapter, our measurements were done with an ad-hoc configuration of either a manual schedule of several superposition components, or a single saturation loop for the duration of the run. But iProver supports a considerably more powerful method of controlling its vast number of parameters: an auto mode which takes into account the syntactic features of the input problem (and the available time) to spawn a strategy which is most likely to solve it. After extensive machine-learning optimisation, this auto mode has drastically improved the performance of iProver as compared to its previous manually-written configuration. However, at present this does not include ground joinability (in the contrary, it is agressively tuned to a hyperparameter space that does not include it).

Crucial to the success of the auto mode is the fact that different configurations or techniques "complement" each other, and hence a schedule with several different components is more likely to succeed at solving problems than a schedule that only includes one, or a few similar ones. It is also very unintuitive to a human how to best combine hundreds of options across two saturation calculi and dozens of preprocessing steps, hence the poor performance of manually tuned schedules versus automatically tuned ones. Hence: there is significant interest in having the auto schedule make use of ground joinability, encompassment demodulation, and AC normalisation and joinability (despite the fact that the machine-learning optimisation procedure is very time-consuming). Not only will this (i) serve as a better benchmark of the practical performance of those techniques in a production-ready theorem prover, but also (ii) help us understand to which degree, and in what types of problems, are these techniques complementary to existing ones. It is our hope that this can be available in the mainline iProver version in the near future.

Furthermore, we have shown that specialising algorithms for specific theories of interest (as was done in this work for AC) offers advantages both in theoretical terms (e.g. AC normalisation being stronger than demodulation) as well as in practical terms (e.g. AC joinability being much faster than "manually" applying general ground joinability with the AC axioms).

This leads us to believe that investigating further particular theories is a very promising avenue of research, as (for instance) abelian groups, idempotence axioms, partial orders, lattices, etc. are all examples of theories which are ubiquitous in wide-ranging domains — from abstract algebra to software verification — but currently very challenging for superposition. This idea has been extensively researched in the domain of unit-equational completion [Peterson and Stickel 1981; Martin and Nipkow 1990; Baader and Nipkow 1998; Hillenbrand, Jaeger and Löchner 1999, to name a few], and now, with the theoretical framework presented in this work, the results can be extended and generalised to full clausal first-order logic, in superposition.

Moreover, we have noticed that sensitivity to term order is, empirically, a major hurdle to applicability of the techniques described (as the term order underpins all rewrite-based simplification rules). Investigating this issue could be fruitful to improve performance, and enable application in more problems. Some preliminary work has been done on this front (page 109), but it needs to be further systematised.

It is our hope that the theoretical tools presented in this work will prove useful in a wide range of fields, and serve as the basis for further theoretical developments in the future. Usage of these techniques, either in iProver or in other superposition provers which adopt them, may prove the key for solving presently open problems in abstract algebra, or other domains of mathematics, just as developments in SAT solving or equational theorem proving proved the key in solving other, long-standing open problems [Appel and Haken 1989; McCune 1997; Heule, Kullmann and Marek 2016].

*Blank page*

# Appendix A

# iProver loop experimental setup

Here we detail the contents of $R_{\text{passive}}$, $R_{\text{active}}$, $R_{\text{immed}}$, and $R_{\text{input}}$,[1] for each of the three experiments in Section 6.2.

**Otter**

- Simplify new clause wrt. passive clauses
  - Fw: demodulation, subsumption, subsumption resolution, AC normalisation, ground joinability, global subsumption.
  - Bw: demodulation, subsumption, subsumption resolution, ground joinability.
- Simplify new clause wrt. active clauses
  - Fw: demodulation, subsumption, subsumption resolution, AC normalisation, ground joinability, global subsumption.
  - Bw: demodulation, subsumption, subsumption resolution, ground joinability.
- Simplify given clause wrt. passive clauses
  - Fw: none.
  - Bw: none.
- Simplify given clause wrt. active clauses
  - Fw: none.
  - Bw: none.

[1]See Section 5.2.

- Simplify new clause wrt. immediate set
  - Fw: none.
  - Bw: none.
- Simplify input clause
  - Fw: demodulation, subsumption, subsumption resolution, AC normalisation, ground joinability, global subsumption.
  - Bw: demodulation, subsumption, subsumption resolution, ground joinability.

**Discount**

- Simplify new clause wrt. passive clauses
  - Fw: none.
  - Bw: none.
- Simplify new clause wrt. active clauses
  - Fw: none.
  - Bw: none.
- Simplify given clause wrt. passive clauses
  - Fw: demodulation, subsumption, subsumption resolution, AC normalisation, ground joinability, global subsumption.
  - Bw: demodulation, subsumption, subsumption resolution, ground joinability.
- Simplify given clause wrt. active clauses
  - Fw: demodulation, subsumption, subsumption resolution, AC normalisation, ground joinability, global subsumption.
  - Bw: demodulation, subsumption, subsumption resolution, ground joinability.
- Simplify new clause wrt. immediate set
  - Fw: none.
  - Bw: none.
- Simplify input clause

  - Fw: none.
  - Bw: none.

**iProver**

- Simplify new clause wrt. passive clauses
  - Fw: none.
  - Bw: none.
- Simplify new clause wrt. active clauses
  - Fw: AC normalisation, unit subsumption.
  - Bw: demodulation, unit subsumption.
- Simplify given clause wrt. passive clauses
  - Fw: demodulation, AC normalisation.
  - Bw: none.
- Simplify given clause wrt. active clauses
  - Fw: demodulation, subsumption, subsumption resolution, AC normalisation, ground joinability, global subsumption.
  - Bw: demodulation, subsumption, subsumption resolution, ground joinability.
- Simplify new clause wrt. immediate set
  - Fw: subsumption, subsumption resolution, AC normalisation.
  - Bw: subsumption, subsumption resolution.
- Simplify input clause
  - Fw: demodulation, subsumption, subsumption resolution, AC normalisation, ground joinability, global subsumption.
  - Bw: demodulation, subsumption, subsumption resolution, ground joinability.

*Blank page*

# Index

AC demodulation, 88
AC joinability, 82
AC normalisation, 86
active set, 104
admissible, 66
associative-commutative, 80
atom, 40
automated theorem proving, 22

backward simplification, 107

Church-Rosser property, 41
clause, 40
closure, 45
closure ordering, 47
closure redundancy, 53, 55
conclusion, 41
confluence, 41, 56
confluent, 56
conjunctive normal form, 39
connectedness, 76
constant, 39
critical pair criteria, 76

demodulation, 67
Discount loop, 105

edge, 95
encompassment demodulation, 67
entailment, 41
equal modulo renaming, 41
equational completion, 42
expression, 40

forward simplification, 107

given-clause loop, 104
ground, 40
ground closures, 45
ground connectedness, 79
ground joinability, 73

immediate simplification, 106
incomparable, 38
incremental ground joinability, 93
indexing, 105
interactive theorem proving, 22
irreflexive, 37

joinability, 41, 70

KBO, 102

lexicographic extension, 38
literal, 40

model, 41
model construction, 56
more general, 40, 41

most general unifier, 40
multiset extension, 38

order extension, 92
Otter loop, 105

partial order, 38
passive set, 104
premise, 41
preorder, 38
productive, 56

reduction ordering, 42
reductive, 55
redundant clause, 43
redundant inference, 43
reflexive, 37
reflexive-transitive, 38
refutational completeness, 43
renaming, 40
resolution, 25
rewrite ordering, 42
rewrite relation, 41
rewrite rules, 41
rewrite via, 42

satisfiability, 41
saturation, 44, 55
signature, 39
simplification orderings, 42
simplification rule, 66
simplification set, 107
soundness, 43
strict subterm, 40
strong ground joinability, 73
strong joinability, 69
substitution, 40
subterm, 40
superposition, 43

term, 39
term rewrite system, 41
terminating, 37
total, 37
transitive, 37
transitive closure, 37
transitive reduction, 37, 95

unfailing completion, 42
unification, 40

variable, 39

well-founded, 37

# Bibliography

Anantharaman and Mzali 1989

Siva Anantharaman and Jalel Mzali (1989). *Unfailing Completion modulo a set of equations*. Technical report. LRI, Université de Paris Sud (cit. on pp. 30, 31).

Appel and Haken 1989

Kenneth Appel and Wolfgang Haken (1989). "Every Planar Map is Four Colorable". In: *Contemporary Mathematics* 98. DOI: 10.1090/conm/098 (cit. on pp. 23, 135).

Avenhaus, Hillenbrand and Löchner 2003

Jürgen Avenhaus, Thomas Hillenbrand and Bernd Löchner (2003). "On using ground joinable equations in equational theorem proving". In: *Journal of Symbolic Computation* 36.1, pp. 217–233. DOI: 10.1016/S0747-7171(03)00024-5 (cit. on pp. 30, 31, 75, 91, 93, 98, 102).

Baader and Nipkow 1998

Franz Baader and Tobias Nipkow (1998). *Term Rewriting and All That*. Cambridge University Press. ISBN: 978-052177920-3. DOI: 10.1017/cbo9781139172752 (cit. on pp. 27, 30, 50, 59, 83, 134).

Bachmair and Dershowitz 1987

Leo Bachmair and Nachum Dershowitz (1987). "Completion for rewriting modulo a congruence". In: *Rewriting Techniques and Applications*. Ed. by Pierre Lescanne. Berlin, Heidelberg: Springer, pp. 192–203. ISBN: 978-3-540-47421-0. DOI: 10.1007/3-540-17220-3_17 (cit. on p. 30).

Bachmair and Dershowitz 1988

Leo Bachmair and Nachum Dershowitz (1988). "Critical Pair Criteria for

Completion”. In: *Journal of Symbolic Computation* 6.1, pp. 1–18. doi: 10.1016/S0747-7171(88)80018-X (cit. on pp. 31, 76, 125).

Bachmair, Dershowitz and Hsiang 1986
Leo Bachmair, Nachum Dershowitz and Jieh Hsiang (1986). “Orderings for Equational Proofs”. In: *Proceedings of the Symposium on Logic in Computer Science (LICS ’86), Cambridge, Massachusetts, USA, June 16-18, 1986*. IEEE Computer Society, pp. 346–357 (cit. on p. 30).

Bachmair, Dershowitz and Plaisted 1989
Leo Bachmair, Nachum Dershowitz and David A. Plaisted (1989). “Completion without failure”. In: *Resolution of Equations in Algebraic Structures, Vol. II: Rewriting Techniques*. Ed. by Hassan Aït-Kaci and Maurice Nivat. Academic Press, pp. 1–30. isbn: 978-0-12-046371-8. doi: 10.1016/B978-0-12-046371-8.50007-9 (cit. on pp. 27, 42, 48, 68).

Bachmair and Ganzinger 1994
Leo Bachmair and Harald Ganzinger (1994). “Rewrite-based Equational Theorem Proving with Selection and Simplification”. In: *Journal of Logic and Computation* 4, pp. 217–247 (cit. on pp. 28, 132).

Bachmair and Ganzinger 1998
Leo Bachmair and Harald Ganzinger (1998). “Equational Reasoning in Saturation-Based Theorem Proving”. In: *Automated Deduction: A Basis for Applications*. Ed. by Wolfgang Bibel and Peter H. Schmitt. Vol. I. Dordrecht, The Netherlands: Kluwer. Chap. 11, pp. 353–397. isbn: 0-7923-5129-0 (cit. on pp. 24, 26).

Bachmair and Ganzinger 2001
Leo Bachmair and Harald Ganzinger (2001). “Resolution Theorem Proving”. In: *Handbook of Automated Reasoning (in 2 volumes)*. Ed. by J. Alan Robinson and Andrei Voronkov. Elsevier and MIT Press, pp. 19–99. doi: 10.1016/B978-044450813-3/50004-7 (cit. on pp. 25, 43, 52).

Bachmair, Ganzinger et al. 1995
Leo Bachmair, Harald Ganzinger, Christopher A. Lynch and Wayne Snyder (1995). “Basic Paramodulation”. In: *Information and Computation* 121.2, pp. 172–192. issn: 0890-5401. doi: 10.1006/inco.1995.1131 (cit. on p. 46).

Barbosa et al. 2022
Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli and Yoni Zohar (2022). "cvc5: A Versatile and Industrial-Strength SMT Solver". In: *Tools and Algorithms for the Construction and Analysis of Systems — 28th International Conference, TACAS 2022, held as part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2–7, 2022, Proceedings, Part I*. Ed. by Dana Fisman and Grigore Rosu. Vol. 13243. Lecture Notes in Computer Science. Springer, pp. 415–442. doi: 10.1007/978-3-030-99524-9_24 (cit. on p. 22).

Barthelemy 1980
Jean Pierre Barthelemy (1980). "An asymptotic equivalent for the number of total preorders on a finite set". In: *Discrete Mathematics* 29.3, pp. 311–313. doi: 10.1016/0012-365x(80)90159-4 (cit. on p. 92).

Bentkamp 2021
Alexander Bentkamp (2021). "Superposition for Higher-Order Logic". English. PhD thesis. Vrije Universiteit Amsterdam (cit. on p. 22).

Bhayat and Reger 2020
Ahmed Bhayat and Giles Reger (2020). "A Combinator-Based Superposition Calculus for Higher-Order Logic". In: *Automated Reasoning — 10th International Joint Conference, IJCAR 2020, Paris, France, July 1–4, 2020, Proceedings, Part I*. Ed. by Nicolas Peltier and Viorica Sofronie-Stokkermans. Vol. 12166. Lecture Notes in Computer Science. Springer, pp. 278–296. doi: 10.1007/978-3-030-51074-9_16 (cit. on p. 22).

Blanchette et al. 2016
Jasmin C. Blanchette, Cezary Kaliszyk, Lawrence C. Paulson and Josef Urban (2016). "Hammering towards QED". In: *Journal of Formalized Reasoning* 9.1, pp. 101–148. doi: 10.6092/issn.1972-5787/4593 (cit. on pp. 24, 112).

Bonacina 2022
Maria Paola Bonacina (2022). "Set of Support, Demodulation, Paramodulation: A Historical Perspective". In: *Journal of Automated Reasoning* 66.4,

pp. 463–497. doi: 10.1007/s10817-022-09628-0 (cit. on pp. 24, 28, 29, 132).

Bonacina and Hsiang 1995
Maria Paola Bonacina and Jieh Hsiang (1995). “Towards a foundation of completion procedures as semidecision procedures”. In: *Theoretical Computer Science* 146.1, pp. 199–242. issn: 0304-3975. doi: https://doi.org/10.1016/0304-3975(94)00187-N (cit. on pp. 27, 30, 76).

Bonnet and Pouzet 1982
Robert Bonnet and Maurice Pouzet (1982). “Linear Extensions of Ordered Sets”. In: *Ordered Sets, NATO Advanced Study Institutes Series.* Ed. by Ivan Rival. Vol. 83. Dordrecht: Springer Netherlands, pp. 125–170 (cit. on p. 50).

Claessen and Smallbone 2018
Koen Claessen and Nicholas Smallbone (2018). “Efficient Encodings of First-Order Horn Formulas in Equational Logic”. In: *Automated Reasoning — 9th International Joint Conference, IJCAR 2018, held as part of the Federated Logic Conference, FLoC 2018, Oxford, UK, July 14–17, 2018, Proceedings.* Ed. by Didier Galmiche, Stephan Schulz and Roberto Sebastiani. Vol. 10900. Lecture Notes in Computer Science. Springer, pp. 388–404. doi: 10.1007/978-3-319-94205-6_26 (cit. on p. 29).

Clarke, Khaira and Zhao 1996
Edmund Melson Clarke, Manpreet Khaira and Xudong Zhao (1996). “Word Level Model Checking – Avoiding the Pentium FDIV Error”. In: *Proceedings of the 33rd Annual Design Automation Conference.* DAC ’96. Las Vegas, Nevada, USA: Association for Computing Machinery, pp. 645–648. isbn: 0897917790. doi: 10.1145/240518.240640 (cit. on pp. 22, 23).

Cruanes 2015
Simon Cruanes (2015). “Extending Superposition with Integer Arithmetic, Structural Induction, and Beyond”. PhD thesis. École Polytechnique — Université Paris-Saclay. url: https://hal.archives-ouvertes.fr/tel-01223502 (cit. on p. 110).

Denzinger, Kronenburg and Schulz 1997
Jörg Denzinger, Martin Kronenburg and Stephan Schulz (1997). “DISCOUNT — A Distributed and Learning Equational Prover”. In: *Journal of Automated*

*Reasoning* 18.2, pp. 189–198. DOI: 10.1023/A:1005879229581 (cit. on pp. 32, 104, 105, 125).

Dershowitz and Manna 1979
Nachum Dershowitz and Zohar Manna (1979). “Proving Termination with Multiset Orderings”. In: *Commun. ACM* 22.8, pp. 465–476. DOI: 10.1145/359138.359142 (cit. on pp. 38, 50).

Duarte and Korovin 2020
André Duarte and Konstantin Korovin (2020). “Implementing Superposition in iProver (System Description)”. In: *Automated Reasoning — 10th International Joint Conference, IJCAR 2020, Paris, France, July 1–4, 2020, Proceedings, Part II*. Ed. by Nicolas Peltier and Viorica Sofronie-Stokkermans. Vol. 12167. Lecture Notes in Computer Science. Springer, pp. 388–397. DOI: 10.1007/978-3-030-51054-1_24 (cit. on pp. 25, 33, 91, 104, 106, 111, 126).

Duarte and Korovin 2021
André Duarte and Konstantin Korovin (2021). “AC Simplifications and Closure Redundancies in the Superposition Calculus”. In: *Automated Reasoning with Analytic Tableaux and Related Methods — 30th International Conference, TABLEAUX 2021, Birmingham, UK, September 6–9, 2021, Proceedings*. Ed. by Anupam Das and Sara Negri. Vol. 12842. Lecture Notes in Computer Science. Springer, pp. 200–217. DOI: 10.1007/978-3-030-86059-2_12. arXiv: 2107.08409 [cs.LO] (cit. on pp. 33, 45, 65).

Duarte and Korovin 2022
André Duarte and Konstantin Korovin (2022). “Ground Joinability and Connectedness in the Superposition Calculus”. In: *Automated Reasoning — 11th International Joint Conference, IJCAR 2022, Haifa, Israel, August 8–10, 2022, Proceedings*. Ed. by Jasmin Blanchette, Laura Kovács and Dirk Pattinson. Vol. 13385. Lecture Notes in Computer Science. Springer, pp. 169–187. DOI: 10.1007/978-3-031-10769-6_11 (cit. on pp. 34, 45, 65, 91).

Eén and Sörensson 2003
Niklas Eén and Niklas Sörensson (2003). “An Extensible SAT-solver”. In: *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, pp. 502–518. DOI: 10.1007/978-3-540-24605-3_37 (cit. on pp. 22, 109).

Fellbaum 1998

Christiane Fellbaum, ed. (1998). *WordNet: An Electronic Lexical Database*. ISBN: 9780262061971. URL: https://wordnet.princeton.edu/ (cit. on p. 22).

Georgiou, Gleiss and Kovács 2020

Pamina Georgiou, Bernhard Gleiss and Laura Kovács (2020). “Trace Logic for Inductive Loop Reasoning”. In: *2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21–24, 2020*. IEEE, pp. 255–263. DOI: 10.34727/2020/isbn.978-3-85448-042-6_33. arXiv: 2008.01387 (cit. on p. 22).

Gödel 1931

Kurt Gödel (1931). “Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme”. German. In: *Monatshefte für Mathematik und Physik* 38.1, pp. 173–198. DOI: doi:10.1007/BF01700692 (cit. on p. 24).

Graf 1995

Peter Graf (1995). *Term Indexing*. Vol. 1053. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. 284 pp. ISBN: 978-3-540-61040-3. DOI: 10.1007/3-540-61040-5 (cit. on pp. 90, 105, 109).

Hales 2014

Thomas C. Hales (2014). “Developments in Formal Proofs”. In: *Computing Research Repository*. arXiv: 1408.6474 (cit. on p. 24).

Herbrand 1930

Jacques Herbrand (1930). “Recherches sur la théorie de la démonstration”. French. In: *Travaux de la société des Sciences et des Lettres de Varsovie* 33 (cit. on p. 24).

Heule, Kullmann and Marek 2016

Marijn J. H. Heule, Oliver Kullmann and Victor W. Marek (2016). “Solving and Verifying the Boolean Pythagorean Triples Problem via Cube-and-Conquer”. In: *Theory and Applications of Satisfiability Testing — SAT 2016*. Ed. by Nadia Creignou and Daniel Le Berre. Springer, pp. 228–245. ISBN: 978-3-319-40970-2. DOI: 10.1007/978-3-319-40970-2_15. arXiv: 1605.00723 (cit. on p. 135).

Hillenbrand, Buch et al. 1997

Thomas Hillenbrand, Arnim Buch, Roland Vogt and Bernd Löchner (1997). "Waldmeister — High-performance equational deduction". In: *Journal of Automated Reasoning* 18.2, pp. 265–270. DOI: 10.1023/A:1005872405899 (cit. on pp. 29, 104).

Hillenbrand, Jaeger and Löchner 1999

Thomas Hillenbrand, Andreas Jaeger and Bernd Löchner (1999). "System Description: Waldmeister — Improvements in Performance and Ease of Use". en. In: *Automated Deduction — CADE-16*. Ed. by Harald Ganzinger. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 232–236. ISBN: 978-3-540-48660-2. DOI: 10.1007/3-540-48660-7_20 (cit. on pp. 110, 134).

Hillenbrand, Piskac et al. 2013

Thomas Hillenbrand, Ruzica Piskac, Uwe Waldmann and Christoph Weidenbach (2013). "From Search to Computation: Redundancy Criteria and Simplification at Work". In: *Programming Logics: Essays in Memory of Harald Ganzinger*. Ed. by Andrei Voronkov and Christoph Weidenbach. Berlin, Heidelberg: Springer, pp. 169–193. ISBN: 978-3-642-37651-1. DOI: 10.1007/978-3-642-37651-1_7 (cit. on p. 32).

Hoder et al. 2012

Krystof Hoder, Zurab Khasidashvili, Konstantin Korovin and Andrei Voronkov (2012). "Preprocessing techniques for first-order clausification". In: *Formal Methods in Computer-Aided Design, FMCAD 2012, Cambridge, UK, October 22-25, 2012*. Ed. by Gianpiero Cabodi and Satnam Singh. IEEE, pp. 44–51. URL: https://ieeexplore.ieee.org/document/6462554/ (cit. on p. 39).

Holden and Korovin 2021

Edvard K. Holden and Konstantin Korovin (2021). "Heterogeneous Heuristic Optimisation and Scheduling for First-Order Theorem Proving". In: *Intelligent Computer Mathematics — 14th International Conference, CICM 2021, Timisoara, Romania, July 26–31, 2021, Proceedings*. Ed. by Fairouz Kamareddine and Claudio Sacerdoti Coen. Vol. 12833. Lecture Notes in Computer Science. Springer, pp. 107–123. DOI: 10.1007/978-3-030-81097-9_8 (cit. on pp. 32, 109, 112, 114, 127).

Hommersom, Lucas and Bommel 2005

Arjen Hommersom, Peter J. Lucas and Patrick Bommel (2005). "Automated

Theorem Proving for Quality-checking Medical Guidelines". In: *Proceedings of the Workshop on Empirically Successful Classical Automated Reasoning, 20th International Conference on Automated Deduction*. Ed. by Geoff Sutcliffe, B. Fischer and Stefan Schulz (cit. on p. 22).

Hsiang and Rusinowitch 1987
Jieh Hsiang and Michael Rusinowitch (1987). "On word problems in equational theories". In: *Automata, Languages and Programming*. Ed. by Thomas Ottmann. Berlin, Heidelberg: Springer, pp. 54–71. ISBN: 978-3-540-47747-1 (cit. on p. 27).

Huet 1980
Gérard Huet (1980). "Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems: Abstract Properties and Applications to Term Rewriting Systems". In: *Journal of the ACM* 27.4, pp. 797–821. ISSN: 0004-5411. DOI: 10.1145/322217.322230 (cit. on p. 30).

Jakubuv and Urban 2017
Jan Jakubuv and Josef Urban (2017). "ENIGMA: Efficient Learning-Based Inference Guiding Machine". In: *Intelligent Computer Mathematics — 10th International Conference, CICM 2017, Edinburgh, UK, July 17–21, 2017, Proceedings*. Ed. by Herman Geuvers, Matthew England, Osman Hasan, Florian Rabe and Olaf Teschke. Vol. 10383. Lecture Notes in Computer Science. Springer, pp. 292–302. DOI: 10.1007/978-3-319-62075-6_20 (cit. on p. 32).

Jech 2006
Thomas Jech (2006). *Set Theory*. 3rd ed. Vol. 79. Pure and Applied Mathematics. Academic Press. ISBN: 9780123819505 (cit. on p. 21).

Khasidashvili, Korovin and Tsarkov 2015
Zurab Khasidashvili, Konstantin Korovin and Dmitry Tsarkov (2015). "EPR-based $k$-induction with Counterexample Guided Abstraction Refinement". In: *Global Conference on Artificial Intelligence, GCAI 2015, Tbilisi, Georgia, October 16-19, 2015*. Ed. by Georg Gottlob, Geoff Sutcliffe and Andrei Voronkov. Vol. 36. EPiC Series in Computing. EasyChair, pp. 137–150. DOI: 10.29007/scv7 (cit. on p. 22).

Klein et al. 2009
Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David

Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch and Simon Winwood (2009). "SeL4: Formal Verification of an OS Kernel". In: *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles.* Big Sky, Montana, USA: Association for Computing Machinery, pp. 207–220. ISBN: 978-160558752-3. DOI: 10.1145/1629575.1629596 (cit. on pp. 22, 23).

Knuth and Bendix 1970
Donald E. Knuth and Peter Bendix (1970). "Simple Word Problems in Universal Algebras". In: *Computational Problems in Abstract Algebra.* Ed. by John Leech. Pergamon, pp. 263–297. DOI: https://doi.org/10.1016/B978-0-08-012975-4.50028-X (cit. on pp. 26, 42, 102).

Korovin 2008
Konstantin Korovin (2008). "iProver — An Instantiation-Based Theorem Prover for First-Order Logic (System Description)". In: *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Proceedings.* Ed. by Alessandro Armando, Peter Baumgartner and Gilles Dowek. Vol. 5195. Lecture Notes in Computer Science. Springer, pp. 292–298. DOI: 10.1007/978-3-540-71070-7_24 (cit. on p. 111).

Korovin 2013
Konstantin Korovin (2013). "Inst-Gen — A Modular Approach to Instantiation-Based Automated Reasoning". In: *Programming Logics.* Ed. by Andrei Voronkov and Christoph Weidenbach. Vol. 7797. Berlin, Heidelberg: Springer, pp. 239–270. DOI: 10.1007/978-3-642-37651-1_10 (cit. on pp. 23, 104, 111).

Kovács and Voronkov 2013
Laura Kovács and Andrei Voronkov (2013). "First-Order Theorem Proving and Vampire". In: *Computer Aided Verification — 25th International Conference, CAV 2013. Proceedings.* Ed. by Natasha Sharygina and Helmut Veith. Vol. 8044. Lecture Notes in Computer Science. Springer, pp. 1–35. DOI: 10.1007/978-3-642-39799-8_1 (cit. on pp. 25, 29, 67, 126, 127).

Löchner 2007
Bernd Löchner (2007). "Things to Know when Implementing KBO". In: *Journal of Automated Reasoning* 36.4, pp. 289–310. ISSN: 0168-7433, 1573-0670. DOI: 10.1007/s10817-006-9031-4. (Visited on 19/03/2019) (cit. on p. 110).

Löchner and Hillenbrand 2002

Bernd Löchner and Thomas Hillenbrand (2002). "A phytography of WALDMEISTER". In: *AI Communications* 15.2,3, pp. 127–133. URL: http://content.iospress.com/articles/ai-communications/aic261 (cit. on p. 27).

López-Hernández and Korovin 2018

Julio César López-Hernández and Konstantin Korovin (2018). "An Abstraction-Refinement Framework for Reasoning with Large Theories". In: *Automated Reasoning — 9th International Joint Conference, IJCAR 2018, Proceedings*. Ed. by Didier Galmiche, Stephan Schulz and Roberto Sebastiani. Vol. 10900. Lecture Notes in Computer Science. Springer, pp. 663–679. DOI: 10.1007/978-3-319-94205-6_43 (cit. on p. 112).

Martin and Nipkow 1990

Ursula Martin and Tobias Nipkow (1990). "Ordered Rewriting and Confluence". In: *Automated Deduction — 10th International Conference, Kaiserslautern, FRG, July 24–27, 1990, Proceedings*. Ed. by Mark E. Stickel. Vol. 449. Lecture Notes in Computer Science. Springer, pp. 366–380. DOI: 10.1007/3-540-52885-7_100 (cit. on pp. 27, 30, 31, 73, 82, 89, 91, 92, 134).

McCune 1990

William McCune (1990). "Otter 2.0". In: *10th International Conference on Automated Deduction*. Ed. by Mark E. Stickel. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 663–664. ISBN: 978-3-540-47171-4. DOI: 10.1007/3-540-52885-7_131 (cit. on pp. 32, 104, 105, 125).

McCune 1997

William McCune (1997). "Solution of the Robbins Problem". In: *Journal of Automated Reasoning* 19.3, pp. 263–276. DOI: 10.1023/a:1005843212881 (cit. on pp. 23, 135).

McCune 2003

William McCune (2003). "OTTER 3.3 Reference Manual". In: *Computing Research Repository*. arXiv: cs.SC/0310056 (cit. on pp. 25, 104).

McCune 2010

William McCune (2010). "Prover9 and Mace4". URL: http://www.cs.unm.edu/~mccune/prover9/ (cit. on p. 25).

Moura and Bjørner 2008
Leonardo de Moura and Nikolaj Bjørner (2008). “Z3: An Efficient SMT Solver”. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by C. R. Ramakrishnan and Jakob Rehof. Berlin, Heidelberg: Springer, pp. 337–340. ISBN: 978-3-540-78800-3. DOI: 10.1007/978-3-540-78800-3_24 (cit. on pp. 22, 109).

Nieuwenhuis and Rubio 2001
Robert Nieuwenhuis and Albert Rubio (2001). “Paramodulation-Based Theorem Proving”. In: *Handbook of Automated Reasoning (in 2 volumes)*. Ed. by J. Alan Robinson and Andrei Voronkov. Elsevier and MIT Press, pp. 371–443. DOI: 10.1016/B978-044450813-3/50009-6 (cit. on pp. 30, 31, 39, 40).

Niles and Pease 2001
Ian Niles and Adam Pease (2001). “Towards a standard upper ontology”. In: *Proceedings of the International Conference on Formal Ontology in Information Systems*, pp. 2–9 (cit. on p. 23).

Nipkow, Paulson and Wenzel 2002
Tobias Nipkow, Lawrence C. Paulson and Markus Wenzel (2002). *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Vol. 2283. Lecture Notes in Computer Science. Springer. 216 pp. ISBN: 978-3-540-43376-7. DOI: 10.1007/3-540-45949-9 (cit. on p. 22).

Nonnengart and Weidenbach 2001
Andreas Nonnengart and Christoph Weidenbach (2001). “Computing Small Clause Normal Forms”. In: *Handbook of Automated Reasoning (in 2 volumes)*. Ed. by John Alan Robinson and Andrei Voronkov. Elsevier and MIT Press, pp. 335–367. DOI: 10.1016/B978-044450813-3/50008-4 (cit. on p. 39).

Peterson and Stickel 1981
Gerald E. Peterson and Mark E. Stickel (1981). “Complete Sets of Reductions for Some Equational Theories”. In: *Journal of the ACM* 28.2, pp. 233–264. ISSN: 0004-5411. DOI: 10.1145/322248.322251 (cit. on pp. 31, 134).

Presburger 1929
Mojżesz Presburger (1929). “Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt”. Trans. German by Ryan Stansifer. In: *Comptes Ren-*

*dus du I congrès de Mathématiciens des Pays Slaves, Warszawa*, pp. 92–101 (cit. on p. 24).

Riazanov 2003
Alexandre Riazanov (2003). “Implementing an efficient theorem prover”. PhD thesis. University of Manchester. URL: https://www.freewebs.com/riazanov/Riazanov_PhD_thesis.pdf (cit. on p. 90).

Robinson and Wos 1969
George Robinson and Lawrence Wos (1969). “Paramodulation and Theorem-Proving in First-Order Theories with Equality”. In: *Fourth Annual Machine Intelligence Workshop*. Ed. by D. Michie and R. Meltzer. Edinburgh, Scotland, pp. 135–150. DOI: 10.1007/978-3-642-81955-1_19 (cit. on p. 28).

Robinson and Voronkov 2001
J. Alan Robinson and Andrei Voronkov, eds. (2001). *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press. ISBN: 978-0-444-50813-3 (cit. on p. 22).

Robinson 1965
John Alan Robinson (1965). “A Machine-Oriented Logic Based on the Resolution Principle”. In: *Journal of the ACM* 12.1, pp. 23–41. DOI: 10.1145/321250.321253 (cit. on p. 25).

Schulz 2002
Stephan Schulz (2002). “E — A Brainiac Theorem Prover”. In: *AI Communications* 15.2,3, pp. 111–126 (cit. on pp. 104, 107, 110, 126).

Schulz 2009
Stephan Schulz (2009). “Implementation of First-Order Theorem Provers”. URL: https://resources.mpi-inf.mpg.de/departments/rg1/conferences/vtsa09/slides/schulz.pdf (visited on 17/09/2022) (cit. on p. 126).

Schulz 2013a
Stephan Schulz (2013). “Simple and Efficient Clause Subsumption with Feature Vector Indexing”. In: *Automated Reasoning and Mathematics — Essays in Memory of William W. McCune*. Ed. by Maria Paola Bonacina and Mark E. Stickel. Vol. 7788. Lecture Notes in Computer Science. Springer, pp. 45–67. DOI: 10.1007/978-3-642-36675-8_3 (cit. on pp. 90, 105, 109).

Schulz 2013b

Stephan Schulz (2013). “System Description: E 1.8”. In: *Logic for Programming, Artificial Intelligence, and Reasoning — 19th International Conference, LPAR-19, Proceedings*. Ed. by Kenneth L. McMillan, Aart Middeldorp and Andrei Voronkov. Vol. 8312. Lecture Notes in Computer Science. Springer, pp. 735–743. DOI: 10.1007/978-3-642-45221-5_49 (cit. on pp. 29, 105, 127).

Silva and Sakallah 1999

João P. Marques Silva and Karem A. Sakallah (1999). “GRASP: A Search Algorithm for Propositional Satisfiability”. In: *IEEE Trans. Computers* 48.5, pp. 506–521. DOI: 10.1109/12.769433 (cit. on p. 22).

Sloane 2022a

N. J. A. Sloane, ed. (2022). *Entry A000108: Catalan numbers.* The On-Line Encyclopedia of Integer Sequences. URL: https://oeis.org/A000798 (cit. on p. 80).

Sloane 2022b

N. J. A. Sloane, ed. (2022). *Entry A000798: Number of different quasi-orders (or topologies, or transitive digraphs) with n labeled elements.* The On-Line Encyclopedia of Integer Sequences. URL: https://oeis.org/A000798 (cit. on p. 93).

Smallbone 2021

Nicholas Smallbone (2021). “Twee: An Equational Theorem Prover”. In: *Automated Deduction — 28th International Conference, Proceedings*. Ed. by André Platzer and Geoff Sutcliffe. Vol. 12699. Lecture Notes in Computer Science. Springer, pp. 602–613. DOI: 10.1007/978-3-030-79876-5_35 (cit. on pp. 76, 91, 93, 104, 105, 125).

Stickel 1981

Mark E. Stickel (1981). “A Unification Algorithm for Associative-Commutative Functions”. In: *Journal of the ACM* 28.3, pp. 423–434. ISSN: 0004-5411. DOI: 10.1145/322261.322262 (cit. on pp. 30, 31).

Suda 2022

Martin Suda (2022). “vprover/vampire – eb5108d: encompassment for Forward”. URL: https://github.com/vprover/vampire/commit/eb5108df06 (cit. on p. 69).

Suppes 1972
Patrick Suppes (1972). *Axiomatic Set Theory*. The University Series in Undergraduate Mathematics. Dover Publications. ISBN: 978-048661630-8 (cit. on p. 38).

Sutcliffe 2016
Geoff Sutcliffe (2016). “The CADE ATP System Competition — CASC”. In: *AI Mag.* 37.2, pp. 99–101. DOI: 10.1609/aimag.v37i2.2620 (cit. on pp. 29, 73, 127).

Sutcliffe 2017
Geoff Sutcliffe (2017). “The TPTP Problem Library and Associated Infrastructure — From CNF to TH0, TPTP v6.4.0”. In: *Journal of Automated Reasoning* 59.4, pp. 483–502. DOI: 10.1007/s10817-017-9407-7 (cit. on pp. 80, 112).

Voronkov 2014
Andrei Voronkov (2014). “AVATAR: The Architecture for First-Order Theorem Provers”. In: *Computer Aided Verification*. Ed. by Armin Biere and Roderick Bloem. Vol. 8559. Lecture Notes in Computer Science. Springer, pp. 696–710. DOI: 10.1007/978-3-319-08867-9_46 (cit. on p. 23).

Vukmirović et al. 2022
Petar Vukmirović, Alexander Bentkamp, Jasmin Blanchette, Simon Cruanes, Visa Nummelin and Sophie Tourret (2022). “Making Higher-Order Superposition Work”. In: *Journal of Automated Reasoning* 66.4, pp. 541–564. DOI: 10.1007/s10817-021-09613-z (cit. on p. 22).

Waldmann et al. 2020
Uwe Waldmann, Sophie Tourret, Simon Robillard and Jasmin Blanchette (2020). “A Comprehensive Framework for Saturation Theorem Proving”. In: *Automated Reasoning — 10th International Joint Conference, IJCAR 2020, Paris, France, July 1–4, 2020, Proceedings, Part I*. Ed. by Nicolas Peltier and Viorica Sofronie-Stokkermans. Vol. 12166. Lecture Notes in Computer Science. Springer, pp. 316–334. DOI: 10.1007/978-3-030-51074-9_18 (cit. on p. 104).

Weidenbach et al. 2009

Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda and Patrick Wischnewski (2009). “SPASS Version 3.5”. In: *Automated Deduction — 22nd International Conference, CADE-22, Montreal, Canada, August 2–7, 2009, Proceedings*. Ed. by Renate A. Schmidt. Berlin, Heidelberg: Springer, pp. 140–145. ISBN: 978-3-642-02959-2 (cit. on pp. 25, 104).

Wos, Robinson and Carson 1965

Lawrence Wos, George A. Robinson and Daniel F. Carson (1965). “Efficiency and Completeness of the Set of Support Strategy in Theorem Proving”. In: *Journal of the ACM* 12.4, pp. 536–541. ISSN: 0004-5411. DOI: 10.1145/321296.321302 (cit. on p. 104).

Zach 2019

Richard Zach (2019). “Hilbert’s Program. The Stanford Encyclopedia of Philosophy”. Ed. by Edward N. Zalta. URL: https://plato.stanford.edu/archives/fall2019/entries/hilbert-program (cit. on p. 24).

*Blank page*