Alexander Bentkamp

Superposition for Higher-Order Logic

VRIJE UNIVERSITEIT

Superposition for Higher-Order Logic

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor of Philosophy
aan de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. V. Subramaniam,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de Faculteit der Bètawetenschappen
op maandag 10 mei 2021 om 11.45 uur
in de online bijeenkomst van de universiteit,
De Boelelaan 1105

door

Alexander Bentkamp

geboren te Hannover, Duitsland

promotor:	prof.dr. W. J. Fokkink
copromotoren:	dr. J. C. Blanchette
	dr. U. Waldmann

Contents

Summary vii

Samenvatting ix

Acknowledgements xi

1 Introduction 1
- 1.1 Motivation . . . 2
- 1.2 Contributions . . . 3
- 1.3 Related Work . . . 4
- 1.4 Implementations . . . 5
- 1.5 Publications . . . 6
- 1.6 Structure of This Thesis . . . 7

2 Preliminaries 9
- 2.1 First-Order Logic . . . 10
- 2.2 The Superposition Calculus . . . 11
  - 2.2.1 Clausal Normal Form . . . 12
  - 2.2.2 The Term Order . . . 12
  - 2.2.3 The Inference Rules . . . 14
  - 2.2.4 Redundancy and Simplification . . . 16
- 2.3 Term Rewriting . . . 16

3 Superposition for Lambda-Free Higher-Order Logic 19
- 3.1 Introduction . . . 20
- 3.2 Logic . . . 21
  - 3.2.1 Syntax . . . 22
  - 3.2.2 Semantics . . . 23
- 3.3 The Calculi . . . 23
  - 3.3.1 The Inference Rules . . . 23
  - 3.3.2 Rationale for the Inference Rules . . . 28
  - 3.3.3 Soundness . . . 31
  - 3.3.4 The Redundancy Criterion . . . 33
  - 3.3.5 Simplification Rules . . . 36
- 3.4 Refutational Completeness . . . 37
  - 3.4.1 Outline of the Proof . . . 38
  - 3.4.2 The Ground First-Order Level . . . 39
  - 3.4.3 The Ground Higher-Order Level . . . 40
  - 3.4.4 The Nonground Higher-Order Level . . . 43

3.5 Implementation . . . 51
3.6 Evaluation . . . 52
3.7 Discussion and Related Work . . . 56
3.8 Conclusion . . . 56

4 The Embedding Path Order for Lambda-Free Higher-Order Terms 57
4.1 Introduction . . . 58
4.2 Preliminaries . . . 59
4.3 Extension Operators . . . 59
4.4 The Order . . . 60
4.5 Properties of the Order . . . 62
4.6 Examples . . . 67
4.7 Implementation . . . 68
4.8 Evaluation . . . 71
4.9 Conclusion . . . 73

5 Superposition with Lambdas 75
5.1 Introduction . . . 76
5.2 Logic . . . 77
5.3 The Calculus . . . 80
5.3.1 The Core Inference Rules . . . 81
5.3.2 Rationale for the Rules . . . 83
5.3.3 Soundness . . . 86
5.3.4 The Redundancy Criterion . . . 87
5.3.5 A Derived Term Order . . . 91
5.4 Refutational Completeness . . . 92
5.4.1 Outline of the Proof . . . 93
5.4.2 The Ground Higher-Order Level . . . 94
5.4.3 The Nonground Higher-Order Level . . . 101
5.5 Extensions . . . 107
5.6 Implementation . . . 112
5.7 Evaluation . . . 114
5.8 Conclusion . . . 117

6 Superposition with Interpreted Booleans 119
6.1 Introduction . . . 120
6.2 Logic . . . 121
6.3 The Calculus . . . 123
6.3.1 Parameters of Our Calculus . . . 123
6.3.2 The Inference Rules . . . 124
6.4 Refutational Completeness . . . 126
6.4.1 Viewing Term Rewriting Systems as Interpretations . . . 126
6.4.2 Construction of the Candidate Model . . . 129
6.4.3 Reduction of Counterexamples . . . 131
6.5 Conclusion . . . 137

7 Superposition for Full Higher-Order Logic **139**
7.1 Introduction . . . 140
7.2 Logic . . . 141
7.3 The Calculus . . . 144
7.3.1 Preprocessing . . . 144
7.3.2 Term Orders and Selection Functions . . . 148
7.3.3 The Core Inference Rules . . . 149
7.3.4 Rationale for the Rules . . . 152
7.3.5 Soundness . . . 155
7.3.6 The Redundancy Criterion . . . 156
7.3.7 Simplification Rules . . . 161
7.3.8 A Concrete Term Order . . . 161
7.4 Refutational Completeness . . . 165
7.4.1 Outline of the Proof . . . 165
7.4.2 The Ground First-Order Level . . . 165
7.4.3 The Ground Higher-Order Level . . . 166
7.4.4 The Nonground Higher-Order Level . . . 174
7.5 Clausification . . . 183
7.6 Implementation . . . 185
7.7 Evaluation . . . 186
7.8 Conclusion . . . 188

8 Conclusion **189**
8.1 Results and Impact . . . 190
8.2 Future Work . . . 191

References **193**

Summary

This thesis presents an extension of the superposition calculus to higher-order logic (also called simple type theory) and its implementation and empirical evaluation in an automated theorem prover. The standard first-order superposition calculus is extended step by step to richer logics in three major milestones, culminating with full higher-order logic.

The first milestone takes the form of four calculi for λ-free higher-order logic. They closely resemble Bachmair and Ganzinger's first-order superposition calculus, but support partial applications and applied variables. Crucially, they also support nonmonotone term orders, an essential feature in preparation for higher-order logic.

The second milestone extends one of these calculi further to a clausal fragment of higher-order logic that includes anonymous functions but excludes Booleans. It uses higher-order unification and can cope with $\beta\eta$-conversion of λ-terms.

The third milestone is the calculus for full higher-order logic. It adds support for an interpreted Boolean type with the familiar connectives and quantifiers, as well as a choice operator.

As a secondary contribution, the thesis introduces the embedding path order, a term order for λ-free higher order terms that resembles the first-order recursive path order. It is a ground-total simplification order; in particular it is monotone and thus avoids the complications dealt with in our first milestone, at the cost of being less efficiently computable.

Each milestone has been implemented in the Zipperposition prover and evaluated on TPTP and Sledgehammer benchmarks. Based on this implementation, Zipperposition outperformed all other provers in the higher-order category of the 2020 edition of the CASC prover competition.

Samenvatting

Dit proefschrift presenteert een uitbreiding van de superpositie-calculus naar hogere-orde logica (ook wel eenvoudige typentheorie genoemd) en de implementatie en empirische evaluatie ervan in een automatische stellingbewijzer. De standaard eerste-orde superpositie-calculus wordt stap voor stap uitgebreid naar rijkere logica in drie mijlpalen, aan het eind waarvan er een calculus is voor hogere-orde logica.

De eerste mijlpaal bestaat uit vier calculi voor λ-vrije hogere-orde logica. Ze lijken sterk op Bachmair en Ganzingers eerste-orde superpositie-calculus, maar ondersteunen partiële toepassingen en toegepaste variabelen. Cruciaal is dat ze ook niet-monotone termordes ondersteunen, een essentiële eigenschap ter voorbereiding op hogere-orde logica.

De tweede mijlpaal breidt een van deze calculi verder uit naar een fragment van hogere-orde logica die anonieme functies bevat, maar Booleans voorbij de clausules uitsluit. Het maakt gebruik van hogere-orde unificatie en kan omgaan met de $\beta\eta$-conversie van λ-termen.

De derde mijlpaal is de calculus voor volledige hogere-orde logica. Het voegt ondersteuning toe voor een geïnterpreteerd Bools type met de bekende connectieven en kwantoren, evenals een keuzeoperator.

Als secundaire bijdrage introduceert het proefschrift de embedding path order, een termorde voor λ-vrije termen van een hogere orde die lijkt op de recursive path order voor de eerste orde. Het is een simplification order; in het bijzonder is het monotoon en vermijdt zo de complicaties die in onze eerste mijlpaal opdoemen, ten koste van een minder efficiënte berekenbaarheid.

Elke mijlpaal is geïmplementeerd in de Zipperposition prover en geëvalueerd op TPTP en Sledgehammer benchmarks. Gebaseerd op deze implementatie, presteerde Zipperposition beter dan alle andere provers in de hogere-orde categorie van de 2020-editie van de CASC bewijzerscompetitie.

Acknowledgements

I want to thank my supervisors Jasmin Blanchette, Uwe Waldmann, and Wan Fokkink for the time and effort they invested in guiding and supporting me. Jasmin came up with the plan for this project and it has worked out perfectly. Uwe's deep understanding of the superposition calculus and his profound knowledge of the associated literature has been invaluable. Wan has not been involved in the research itself but has been extremely supportive in organizational, social, and career matters. I am also grateful to my other coauthors Simon Cruanes, Sophie Tourret, Petar Vukmirović, and Visa Nummelin for the excellent work. Without them, I would not have been able to complete this endeavor.

I would like to thank Herman Geuvers, Laura Kovács, Nicolas Peltier, Giles Reger, and Stefan Schlobach for being on the doctorate board and for their comments on my manuscript. I would like to thank Jaap Heringa for agreeing to be the chair of my defense. I thank Ahmed Bhayat and Alexander Steen for the interesting discussions on higher-order logic reasoning and the insights they shared with me. I am grateful to the maintainers of StarExec for letting us use their service and especially to Geoff Sutcliffe for his help with StarExec and the TPTP.

I also want to thank: Haniel Barbosa, Christoph Benzmüller, Maria Paola Bonacina, Sander Dahmen, Martin Desharnais, Daniel El Ouraoui, Mathias Fleury, Pascal Fontaine, Carsten Fuhs, Jürgen Giesl, Johannes Hölzl, Rob Lewis, Tomer Libal, Andrei Popescu, Femke van Raamsdonk, Anders Schlichtkrull, Stephan Schulz, Hans-Jörg Schurr, Mark Summerfield, Enrico Tassi, Dmitriy Traytel, Andrei Voronkov, Daniel Wand, Christoph Weidenbach, and anonymous reviewers of the papers I submitted.

The research presented in this thesis has benefited from funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 713999, Matryoshka), from the Netherlands Organization for Scientific Research (NWO) Incidental Financial Support scheme, and from the NWO Vidi program (project No. 016.Vidi.189.037, Lean Forward).

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

1

Introduction

The field of automated theorem proving studies automatic procedures to find mathematical proofs. These accept a mathematical conjecture as input and—if they terminate—determine either that the conjecture is a theorem or that a counterexample exists. The conjecture is stated in a formal logic. Since the 1960s, first-order logic (initially without, later with equality) has established itself as the most widely used input logic because it strikes a good balance between simplicity and expressiveness. Higher-order logic (also called simple type theory) offers a richer language to state mathematical conjectures, providing quantification over functions, λ-expressions, and a first-class Boolean type, allowing to naturally express concepts such as summations, integrals, and induction principles. However, this comes at the cost of more complex syntax and semantics, which is why the research community has mostly focussed on first-order logic in the past.

Superposition [9] is widely regarded as the calculus par excellence for first-order logic with equality. It emerged in the early 1990s from the resolution calculus and ideas from Knuth–Bendix completion. Given a set of axioms and a conjecture, superposition provers negate the conjecture and compute a clausal normal form of the formulas, on which the core calculus can operate. From the resulting initial set of clauses, the core calculus incrementally derives more clauses according to a small set of inference rules. To curb the explosive derivation of new clauses, a term order restricts the inferences that must be applied. If the conjecture is true, given enough time and resources, the procedure will eventually derive the false clause $\bot$, refuting the negated conjecture and thus proving the conjecture. Since the yearly CASC prover competition was established in 1996, the winner in the first-order category has always been based on superposition. Provers based on superposition are successfully used as a backend in proof assistants [5], software verifiers [57], and higher-order provers [126], demonstrating the efficiency of this calculus.

This thesis presents an extension of the superposition calculus to higher-order logic and to intermediate logics between first-order and higher-order logic.

1

1.1. Motivation

Our main motivation to extend the superposition calculus to higher-order logic is to provide better automation to proof assistants. In contrast to automated provers, proof assistants require interaction with a user to prove theorems but allow the user to prove sophisticated mathematical theorems that are out of reach for fully automated systems. Proof assistants are used to verify hardware and software and to formalize mathematics. Some of the biggest success stories in formalization of mathematics are the formalization of the four color theorem [65], the odd order theorem [66], and the Kepler conjecture [70]. In software verification, major achievements include the formally verified C compiler CompCert [98] and the seL4 microkernel [86].

However, much time and effort was necessary to achieve these successes. For instance, the development of the microkernel seL4 itself took about two person-years, whereas the formal verification of it took 20 person-years. Good automation within proof assistants is crucial to speed up the formalization process. For many projects, users even develop automation procedures for tasks specific to their project to avoid repeating the same proof steps over and over.

For general purpose automation, *hammers* such as Sledgehammer [111] and HOLyHammer [80] are an enormous time saver. Hammers are tools in proof assistants that allow the user to find proofs automatically by invoking external automated provers. Thomas Hales [69], who led the Flyspeck project formalizing the Kepler conjecture, writes:

> Sledgehammers and machine learning algorithms have led to visible success. Fully automated procedures can prove 40% of the theorems in the Mizar math library, 47% of the HOL Light/Flyspeck libraries, with comparable rates in Isabelle. These automation rates represent an enormous savings in human labor.

In spite of the success of hammers, they have the potential to achieve even more because the translation between different logics suppresses their true capabilities. The idea behind hammers is to send a conjecture from the proof assistant to external automated provers and—if a proof is found—reconstruct the proof in a language the proof assistant can verify. Since automated provers typically operate on first-order logic and proof assistants typically operate on some higher-order logic, the problem needs to be encoded into first-order logic before it can be handed to the automated prover [84, 115]. Such encodings are incomplete, bloat the problem, or both. For instance,

- to support partial applications and applied variables, applications $\mathsf{f}\,\mathsf{a}$ are encoded as $\mathsf{app}(\mathsf{f},\mathsf{a})$;
- to support λ-expressions, they must be encoded using $\mathsf{S}\,\mathsf{K}$ combinators—e.g., $\lambda x.\, x$ as $\mathsf{S}\,\mathsf{K}\,\mathsf{K}$;
- to support reasoning with Boolean terms, Boolean operators must be axiomatized.

Due to these encodings, a seemingly simple higher-order problem can become a surprisingly hard first-order problem.

This issue can be avoided by using automated provers operating on higher-order logic, such as Satallax [42], which is based on tableau calculus, and Leo-III [126], which is based on paramodulation. However, while these provers work well on small problems that require sophisticated higher-order reasoning, they easily fail on problems coming from proof assistants, which are large and often require only shallow higher-order reasoning [128].

This is our motivation to extend superposition, which is extremely successful on first-order problems, to operate on higher-order problems directly. Eliminating the need for encodings allows us to develop more powerful and targeted heuristics because the automated prover can assess the problem in its original form.

One of our primary design goals is a *graceful* generalization, meaning that the higher-order calculus should essentially behave as first-order superposition when given a first-order problem and smoothly generalize to problems with increasing amounts of higher-order components. This matters because problems originating from proof assistants are typically of a mostly first-order nature with only minor higher-order elements.

A second design goal is refutational completeness. Intuitively, this is the property that given a provable conjecture, with enough resources, a prover implementing the calculus will eventually find a proof. Even though it is possible to achieve high success rates with incomplete provers, refutational completeness can act as a guide to specific side conditions of calculus rules that yield an efficient procedure.

1.2. Contributions

The main contribution of this thesis is the development of a sound and refutationally complete superposition calculus for higher-order logic, implemented and evaluated in the Zipperposition prover.

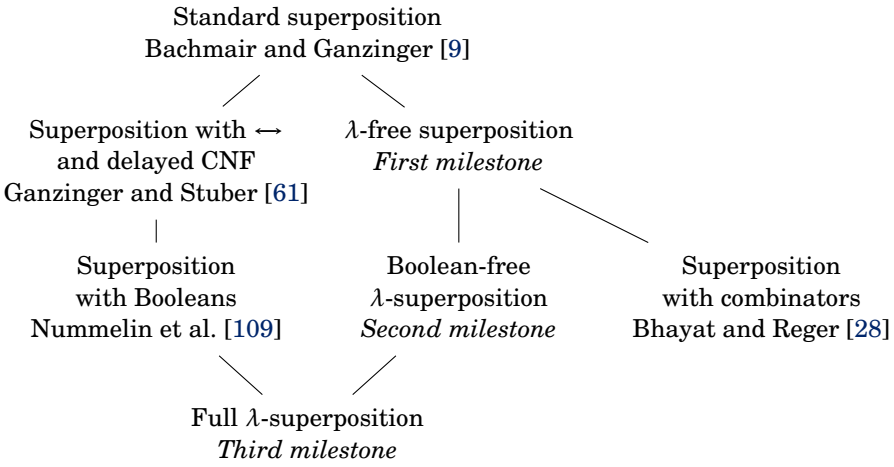
To face the different challenges of higher-order logic one by one, we partitioned the task in three major milestones. Each milestone is a sound and refutationally complete calculus for some logic that lies between first-order and higher-order logic.

- The first milestone operates on λ-free higher-order logic—a logic that is essentially first-order logic, but allows for applied variables and partial applications. We designed four different calculi and compared them empirically. The main challenge of this milestone was to cope with term orders that are not entirely monotone. In particular, we aimed to support a λ-free higher-order variant of the recursive path order (RPO) that has previously been designed by Blanchette et al. [34].
- The second milestone operates on clausal higher-order logic—a logic that additionally allows for λ-expressions, but does not have an interpreted Boolean type. The main challenges were to deal with higher-order unification and with $\beta\eta$-conversion of λ-terms. Although we could not reuse the completeness theorem of the first milestone directly, many of its proof ideas reappear.
- The third milestone finally operates on full higher-order logic. The main challenge was to add support for an interpreted Boolean type, including the familiar connectives and quantifiers, as well as a choice operator. This last

milestone is based on the second milestone, but also on an ongoing project of Nummelin, Tourret, Vukmirović, and myself, in which we are developing a calculus for first-order logic with an interpreted Boolean type. This project in turn is based on Ganzinger and Stuber's work on delayed clausification for first-order superposition [61].

The order in which we add new features to the intermediate logics is arbitrary, but it seemed convenient to us. The partition into these intermediate logics not only keeps each milestone manageable, but also allows to backtrack and explore different design decisions. For instance, Bhayat and Reger [28] used our first milestone as a basis for a calculus for clausal higher-order logic based on SK combinators.

Overall, the pedigree of the family of higher-order superposition calculi is as follows:



We have implemented each of the milestones in the Zipperposition prover and evaluated them on TPTP [130] and Sledgehammer benchmarks. The evaluation results have been promising. As further confirmation, Zipperposition won in the higher-order category at the CASC-J10 prover competition in 2020, by a large margin.

As an additional contribution, I have developed the embedding path order (EPO), a term order that resembles the recursive path order (RPO) but is a ground-total simplification order on lambda-free higher-order terms. Unlike the Knuth–Bendix order (KBO), the natural generalization of RPO to lambda-free higher-order terms is not monotone [34], causing the complications investigated in our first milestone. EPO is monotone and thus answers a research question that emerged from our work on the first milestone.

1.3. Related Work

Our calculi join the family of proof systems for higher-order logic. It is related to Andrews's higher-order resolution [1], Huet's constrained resolution [73], Jensen and

Pietrzykowski's ω-resolution [76], Snyder's higher-order E-resolution [122], Benzmüller and Kohlhase's extensional higher-order resolution [20], Benzmüller's higher-order unordered paramodulation and RUE resolution [19], and Bhayat and Reger's combinatory superposition [28]. A noteworthy variant of higher-order unordered paramodulation is Steen and Benzmüller's higher-order ordered paramodulation [126], whose order restrictions undermine refutational completeness but yield better empirical results. Other approaches are based on analytic tableaux [12, 88, 89, 114], connections [2], sequents [101], and satisfiability modulo theories (SMT) [13]. Andrews [3] and Benzmüller and Miller [21] provide excellent surveys of higher-order automation.

Our implementation in Zipperposition joins the league of automated provers for higher-order logic. Its rivals are—among others—the following provers. TPS [4] is based on the connection method and expansion proofs. LEO [20] and LEO-II [24] implement variants of RUE resolution. Leo-III [126] is based on higher-order paramodulation. Satallax [42] implements a higher-order tableau calculus guided by a SAT solver. LEO-II, Leo-III, and Satallax integrate first-order provers as terminal procedures. AgsyHOL [101] is based on a focused sequent calculus guided by narrowing. The Isabelle proof assistant [108] (which includes a tableau reasoner and a rewriting engine) and its Sledgehammer subsystem also participate in the higher-order division of the CADE ATP System Competition [129]. The SMT solvers CVC4 and veriT have recently been extended to higher-order logic [13]. Vampire now implements both combinatory superposition and a version of standard superposition with first-order unification replaced by restricted combinatory unification [27].

Many researchers have proposed or used encodings of higher-order logic constructs into first-order logic, including Robinson [115], Kerber [84], Dougherty [52], Dowek et al. [54], Hurd [75], Meng and Paulson [106], Obermeyer [110], and Czajka [48]. Encodings of types, such as those by Bobot and Paskevich [37] and Blanchette et al. [30], are also crucial to obtain a sound encoding of higher-order logic. These ideas are implemented in proof assistant in the form of hammers such as Sledgehammer [111], MizAR [134], HOLyHammer [81], and CoqHammer [49]. The translation must eliminate the λ-expressions, typically using SK combinators or λ-lifting, and encode typing information.

1.4. Implementations

For our empirical evaluations of the various calculi and the term order EPO, we have implemented them in the Zipperposition prover. Zipperposition [46, 47] is an open source superposition prover written in OCaml.[1] In contrast to highly-optimized C or C++ provers such as E, Zipperposition allows us to prototype quickly and to experiment with rules and heuristics more flexibly.

Zipperposition's architecture is a modular saturation loop with an extensible set of rules for inferences and simplifications. Based on a unit superposition prover used in the proof assistant Matita [6], its calculus and main loop where inspired by the E prover [117]. It was initially designed for polymorphic first-order logic with

[1] https://github.com/sneeuwballen/zipperposition

equality, as embodied by TPTP TF1 [32]. Later, Cruanes extended the prover with a pragmatic higher-order mode with support for λ-abstractions and extensionality, without any completeness guarantees. Our implementations builds on Cruanes's work.

We have been developing and extending Zipperposition continuously. Each evaluation section of this thesis points to supplementary material containing the relevant development version of Zipperposition, the benchmarks, and the raw evaluation results.

1.5. Publications

This is a cumulative thesis, meaning that its contents are a compilation of publications and publication drafts that have been or are to be published at conferences and in journals. The thesis contains contents from the following publications and publication drafts, with the consent of my coauthors:

1. A. Bentkamp, J. C. Blanchette, S. Cruanes, and U. Waldmann. Superposition for Lambda-Free Higher-Order Logic. In D. Galmiche, S. Schulz, R. Sebastiani (eds.) International Joint Conference on Automated Reasoning (IJCAR 2018), LNCS 10900, pp. 28–46, Springer, 2018.
2. A. Bentkamp, J. Blanchette, S. Cruanes, and U. Waldmann. Superposition for Lambda-Free Higher-Order Logic. Accepted in Logical Methods in Computer Science.
3. A. Bentkamp. The Embedding Path Order for Lambda-Free Higher-Order Terms. To be submitted.
4. A. Bentkamp, J. Blanchette, S. Tourret, P. Vukmirović, and U. Waldmann. Superposition with Lambdas. In P. Fontaine (ed.) Conference on Automated Deduction (CADE-27), LNCS 11716, pp. 55–73, Springer, 2019.
5. A. Bentkamp, J. Blanchette, S. Tourret, P. Vukmirović, and U. Waldmann. Superposition with Lambdas. Accepted in Journal of Automated Reasoning.
6. V. Nummelin, A. Bentkamp, S. Tourret, and P. Vukmirović. Superposition with First-Class Booleans and Inprocessing Clausification. Submitted.
7. A. Bentkamp, J. Blanchette, S. Tourret, and P. Vukmirović. Superposition for Full Higher-Order Logic. Submitted.

During my time as a PhD candidate, I have also worked on the following papers, which are not covered in this thesis:

8. A. Bentkamp, J. Blanchette, and D. Klakow. An Isabelle Formalization of the Expressiveness of Deep Learning (Extended Abstract). In T. C. Hales, C. Kaliszyk, S. Schulz, J. Urban (eds.) Conference on Artificial Intelligence and Theorem Proving (AITP 2017), pp. 22–23.
9. A. Bentkamp, J. C. Blanchette, and D. Klakow. A Formal Proof of the Expressiveness of Deep Learning. In M. Ayala-Rincón, C. A. Muños (eds.) Conference on Interactive Theorem Proving (ITP 2017), LNCS 10499, pp. 46–64, Springer,

2017.

10. A. Bentkamp, J. C. Blanchette, and D. Klakow. A Formal Proof of the Expressiveness of Deep Learning. Journal of Automated Reasoning 63(2), pp. 347-368, 2019.
11. P. Vukmirović, A. Bentkamp, and V. Nummelin. Efficient Full Higher-Order Unification. In Z. M. Ariola (ed.), International Conference on Formal Structures for Computation and Deduction (FSCD 2020), LIPIcs 167, pp. 5:1–5:17, Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2020.
12. P. Vukmirović, A. Bentkamp, and V. Nummelin. Efficient Full Higher-Order Unification. Extended journal version submitted.
13. P. Vukmirović, A. Bentkamp, J. Blanchette, S. Cruanes, S. Tourret, and V. Nummelin. Making Higher-Order Superposition Work. Submitted.

1.6. Structure of This Thesis

This thesis is composed of one chapter for each of the three milestones, interspersed with a chapter on the embedding path order and a chapter introducing parts of my ongoing work with Nummelin and other colleagues. I present them in the following order:

- Chapter 3 describes our calculi for lambda-free higher-order logic, our first milestone.
- Chapter 4 describes the term order EPO, which can be used as an replacement for RPO that avoids the complications described in the previous chapter.
- Chapter 5 describes our calculus for clausal higher-order logic, our second milestone.
- Chapter 6 describes a calculus for first-order logic with interpreted Booleans, which serves as a basis for the following chapter.
- Chapter 7 describes our calculus for full higher-order logic, the final milestone and main contribution of this thesis.
- Chapter 8 summarizes our results and plans for future work.

2

Preliminaries

This chapter introduces basic concepts that are a prerequisite for the rest of the thesis. I discuss the syntax and semantics of monomorphic first-order logic. I introduce the two most commonly used term orders used for superposition, the lexicographic path order and the Knuth–Bendix order. I present the first-order superposition calculus and illustrate it with an example derivation. Finally, I go over basic concepts of term rewriting.

2.1. First-Order Logic

Bachmair and Ganzinger's original version of superposition operates on untyped first-order logic with equality. However, the calculus and completeness proof apply essentially verbatim on the monomorphic variant of the logic as well. This is the variant that we introduce below.

Throughout the thesis, we use the following notation for tuples: We write $\bar{a}_n$ or $\bar{a}$ for a tuple $(a_1,\ldots,a_n)$ or a product $a_1 \times \cdots \times a_n$, where $n \geq 0$. Abusing notation, we sometimes write $f(\bar{a})$ for the tuple $(f(a_1),\ldots,f(a_n))$. We write () or ε for the empty tuple, t for the singleton tuple (t), and $\bar{s} \cdot \bar{t}$ for the concatenation of the tuples $\bar{s}$ and $\bar{t}$.

Syntax We fix a set Σ_{ty} of type constructors with associated arities. A first-order type is inductively defined to be of the form $\kappa(\bar{\tau}_n)$ for an n-ary type constructor $\kappa \in \Sigma_{\mathsf{ty}}$ and types $\bar{\tau}_n$. We write κ for $\kappa()$. A type declaration is an expression of the form $\bar{\tau}_n \Rightarrow \upsilon$ for types $\bar{\tau}_n$ and υ. We call $\bar{\tau}_n$ the argument types and υ the return type. If $n = 0$, we simply write υ for $() \Rightarrow \upsilon$.

We fix a set Ω of (function) symbols f, each associated with a type declaration $\bar{\tau}_n \Rightarrow \upsilon$, written as $\mathsf{f} : \bar{\tau}_n \Rightarrow \upsilon$ or f. We fix a set Π of predicates p, each associated with a tuple of argument types $\bar{\tau}_n$, written as $\mathsf{p} : \bar{\tau}_n$ or p. We require that Π contains an equality predicate $\approx_\tau : \tau \times \tau$ for each type τ. We usually write $\approx$ for $\approx_\tau$. Moreover, we fix a countably infinite set $\mathcal{V}$ of variables with associated types, written as $x : \tau$ or x. The notation $t : \tau$ will also be used to indicate the type of arbitrary terms t.

The sets Ω and Π form the term signature $\Sigma = (\Omega, \Pi)$. The term signature and the type signature form the signature $(\Sigma_{\mathsf{ty}}, \Sigma)$ of first-order logic. The set of first-order *terms* is inductively defined as follows. Every $x : \tau \in \mathcal{V}$ is a term of type τ. If $\mathsf{f} : \bar{\tau}_n \Rightarrow \upsilon \in \Omega$ and $\bar{t}_n : \bar{\tau}_n$ is a tuple of terms, then the application $\mathsf{f}(\bar{t}_n)$ (or simply f if $n = 0$) is a term of type υ. A term is ground if it contains no variables.

The set of first-order *formulas* is inductively defined as follows. If $\mathsf{p} \in \Pi$ has argument types $\bar{\tau}_n$ and $\bar{t}_n : \bar{\tau}_n$ is a tuple of terms, then the $\mathsf{p}(\bar{t}_n)$ is a formula. For the case of $\mathsf{p} = \approx$, we write $t \approx s$ for $\approx(t, s)$. Moreover, the expressions $\top$, $\bot$, $\neg\phi$, $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \rightarrow \psi$, $\phi \leftrightarrow \psi$, $\forall x.\, \phi$, and $\exists x.\, \phi$ are formulas if ϕ and ψ are formulas.

Subterms and positions are inductively defined as follows. A position in a term is a tuple of natural numbers. For any term t, the empty tuple ε is a position of t, and t is the subterm of t at position ε. If t is the subterm of u_i at position p, then $i.p$ is a position of $\mathsf{f}(\bar{u})$, and t is the subterm of $\mathsf{f}(\bar{u})$ at position $i.p$. We write $s|_p$ to denote the subterm at position p in s. We write $s[u]_p$ to denote a term s with the subterm u at position p and call $s[\ ]_p$ a *context*; the position p may be omitted in this notation.

A *substitution* σ, ρ, θ is a map from variables to terms that maps all but finitely many variables to themselves. Each variable must be mapped to a term of the same type. The image of a variable x under a substitution ρ is denoted $x\rho$. We write a substitution as $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ with the convention that the substitution maps all variables that are not listed to themselves. The application of a substitution ρ to a term t results in a term $t\rho$, recursively defined as $t\rho = x\rho$ if t is a variable x and $t\rho = \mathsf{f}(\bar{t}\rho)$ if $t = \mathsf{f}(\bar{t})$. We write $\rho\sigma$ for the concatenation $x \mapsto (x\rho)\sigma$ of two substitutions. We write $\rho[x \mapsto t]$ for the substitution that maps x to t and behaves like ρ on all other variables. A substitution θ is grounding for a term t if $t\theta$ is ground.

A substitution ρ is a unifier of two terms s and t if $s\rho = t\rho$. A most general unifier $\mathrm{mgu}(s,t)$ is a unifier σ of s and t such that for any other unifier θ, there exists a ρ such that $\sigma\rho = \theta$. If two terms are unifiable, a most general unifier exists and is unique up to variable renaming.

Semantics A first-order interpretation $\mathcal{I} = (\mathcal{U}, \mathcal{J})$ is a pair, consisting of a universe $\mathcal{U}_\tau$ for each type τ and an *interpretation function* $\mathcal{J}$, which associates with each symbol $\mathsf{f} : \bar{\tau} \Rightarrow \upsilon$ and universe elements $\bar{a} \in \mathcal{U}_{\bar{\tau}}$ a universe element $\mathcal{J}(\mathsf{f})(\bar{a}) \in \mathcal{U}_\upsilon$ and which associates with each predicate $\mathsf{p} : \bar{\tau}$ and universe elements $\bar{a} \in \mathcal{U}_{\bar{\tau}}$ a number $\mathcal{J}(\mathsf{p})(\bar{a}) \in \{0, 1\}$. We require that $\mathcal{J}(\approx)(a, b) = 1$ if $a = b$ and 0 otherwise.

A *valuation* is a function assigning an element $\xi(x) \in \mathcal{U}_\tau$ to each variable $x : \tau$. For an interpretation $\mathcal{I}$ and a valuation ξ, the denotation of a term is inductively defined as $[\![x]\!]^\xi_\mathcal{I} = \xi(x)$ for a variable $x \in \mathcal{V}$ and $[\![\mathsf{f}(\bar{t})]\!]^\xi_\mathcal{I} = \mathcal{J}(\mathsf{f})([\![\bar{t}]\!]^\xi_\mathcal{I})$ for a symbol $\mathsf{f} \in \Omega$ and appropriately typed terms $\bar{t}$.

Given an interpretation $\mathcal{I}$ and a valuation ξ, the denotation of a formula is inductively defined as

$$
\begin{aligned}
[\![\top]\!]^\xi_\mathcal{I} &= 1 & [\![\mathsf{p}(\bar{t})]\!]^\xi_\mathcal{I} &= \mathcal{J}(\mathsf{p})(\bar{a}) \\
[\![\bot]\!]^\xi_\mathcal{I} &= 0 & [\![\phi \to \psi]\!]^\xi_\mathcal{I} &= \max(1 - [\![\phi]\!]^\xi_\mathcal{I}, [\![\psi]\!]^\xi_\mathcal{I}) \\
[\![\neg\phi]\!]^\xi_\mathcal{I} &= 1 - [\![\phi]\!]^\xi_\mathcal{I} & [\![\phi \leftrightarrow \psi]\!]^\xi_\mathcal{I} &= \text{if } [\![\phi]\!]^\xi_\mathcal{I} = [\![\phi]\!]^\xi_\mathcal{I} \text{ then 1 else 0} \\
[\![\phi \wedge \psi]\!]^\xi_\mathcal{I} &= \min([\![\phi]\!]^\xi_\mathcal{I}, [\![\psi]\!]^\xi_\mathcal{I}) & [\![\forall x.\, \phi]\!]^\xi_\mathcal{I} &= \min\{[\![\phi]\!]^{\xi[x \mapsto a]}_\mathcal{I} \mid a \in \mathcal{U}_\tau\} \\
[\![\phi \vee \psi]\!]^\xi_\mathcal{I} &= \max([\![\phi]\!]^\xi_\mathcal{I}, [\![\psi]\!]^\xi_\mathcal{I}) & [\![\exists x.\, \phi]\!]^\xi_\mathcal{I} &= \max\{[\![\phi]\!]^{\xi[x \mapsto a]}_\mathcal{I} \mid a \in \mathcal{U}_\tau\}
\end{aligned}
$$

for predicates $\mathsf{p} : \bar{\tau}_n \in \Pi$, terms $\bar{t}_n : \bar{\tau}_n$, formulas ϕ and ψ, and variables $x : \tau$.

A formula ϕ is true in an interpretation $\mathcal{I}$ under a valuation ξ if $[\![\phi]\!]^\xi_\mathcal{I} = 1$. An interpretation $\mathcal{I}$ is a model of ϕ, written $\mathcal{I} \models \phi$, if ϕ is true in $\mathcal{I}$ under all valuations ξ. A set of formulas N entails a set of formulas M, written $N \models M$, if every model of all formulas in N is also a model of all formulas in M.

2.2. The Superposition Calculus

Superposition is a procedure to determine whether a given set of formulas is contradictory. By contradictory, we mean that these formulas entail $\bot$, i.e., that there exists no model for these formulas. First-order logic is undecidable, meaning that there exists no procedure to decide (in finite time) whether a set of formulas is contradictory. It is, however, semi-decidable, meaning that there exist procedures such as superposition that can find a contradiction for any formula set that is actually contradictory but do not always terminate for other formula sets that are not. This property of a procedure is refutational completeness.

Superposition can also be used to prove entailments—i.e, to determine whether a given set of formulas, which we call the assumptions, entails another formula, which we call the conjecture. We negate the conjecture and let superposition determine whether the assumptions together with the negated conjecture are contradictory. If they are, this implies that that the conjecture is entailed by the assumptions.

Superposition is a saturation-based procedure. The core idea is that, starting from an initial set of formulas, a set of inference rules derives more and more

formulas until it derives $\bot$, indicating a contradiction, or runs out of new formulas to derive, indicating that a model exists. Due to the undecidability of first-order logic, a third possibility is that the procedure does not terminate.

In the following, we assume that the equality predicates $\approx$ are the only predicates. This assumption simplifies both theory and implementation, but does not limit generality because predicates $\mathsf{p}(\bar{t})$ can be encoded as $\mathsf{f}_\mathsf{p}(\bar{t}) \approx \mathsf{T}$ where f_p is a function symbol with an auxiliary return type o and $\mathsf{T} : o$ is an auxiliary constant symbol. For this reason, in the remainder of the thesis, our logics will not even include a set of predicates.

2.2.1. Clausal Normal Form

The superposition calculus operates on a normal form of first-order formulas, the clausal normal form (CNF). A formula is in CNF, if it is of the form $\forall x_1. \cdots \forall x_k. C_1 \wedge \cdots \wedge C_m$ where each C_i is of the form $L_{i,1} \vee \cdots \vee L_{i,n}$ and each $L_{i,j}$ is either of the form $s_{i,j} \approx t_{i,j}$ or of the form $\neg\, s_{i,j} \approx t_{i,j}$, which we abbreviate as $s_{i,j} \not\approx t_{i,j}$. The C_i are called clauses and the $L_{i,j}$ are called literals. Since all variables are universally quantified at the top level, the quantifiers are usually left implicit. Since the order of clauses, of literals, and of sides of a literal does not influence the truth of the formula, we view literals as unordered term pairs with a sign (positive or negative), clauses as multisets of literals, and the entire CNF formula as a set of clauses. Every formula can be converted into an equisatisfiable CNF formula, meaning that the original formula has a model if and only if the CNF formula has a model.

2.2.2. The Term Order

Superposition is parameterized by a term order $\succ$ that allows us to restrict the search space. A major issue with resolution, paramodulation, and other similar calculi that predate superposition is that any symmetries in the given formulas exponentially increase the search space. The term order breaks many of such symmetries by determining on which part of a clause the calculus should work first.

The completeness proof of superposition requires the term order to be a ground-total and well-founded simplification order. Given a binary relation $>$, we write $<$ for its converse (i.e., $a < b \Leftrightarrow b > a$) and $\geq$ for its reflexive closure (i.e., $b \geq a \Leftrightarrow b > a \vee b = a$). A binary relation $>$ on terms is a simplification order if it is irreflexive (i.e., $t \not> t$), is transitive (i.e., $u > t > s \Rightarrow u > s$), is compatible with contexts (i.e., if $t_i > s_i$ for all i, then $\mathsf{f}(\bar{t}) > \mathsf{f}(\bar{s})$), is stable under substitutions (i.e., $t > s$ implies $t\sigma > s\sigma$), and has the subterm property (i.e., $t \geq s$ if s is a subterm of t). It is *ground-total* if for all distinct ground terms s and t either $t > s$ or $t < s$. It is *well founded* if there is no infinite descending chain $t_1 > t_2 > \cdots$.

To extend the term order to literals and clauses, we employ the multiset order [50]. Given a partial order $>$ on a set S, the multiset order induced by $>$ is a partial order on multisets of elements from N, defined as follows. A multiset M is larger than a multiset N if there exist multisets X and Y such that $\emptyset \neq X \subseteq M$, $N = (M - X) \cup Y$, and for every $y \in Y$ there exists an $x \in X$ with $x > y$. Using this definition, we extend the term order to a literal order, viewing a positive literal $s \approx t$ as the multiset $\{s, t\}$ and a negative literal $s \not\approx t$ as the multiset $\{s, s, t, t\}$. This order is then extended from

literals to clauses, again using the multiset order. We reuse the symbol $\succ$ for the literal and clause order induced by $\succ$.

Lexicographic Path Order The lexicographic path order [8, 82] is commonly used for superposition. It is defined as follows:

Definition 2.1 (Lexicographic path order)**.** Let the precedence $>$ be a well-founded strict order on the signature Σ. The *lexicographic path order* $>_{\mathsf{lp}}$ on first-order terms induced by $>$ is recursively defined as follows: Let $t >_{\mathsf{lp}} s$ if

1. s is a variable occurring in t and $t \neq s$; or
2. $t = \mathsf{g}(\bar{t}_n)$, $s = \mathsf{f}(\bar{s}_m)$, and one of the following conditions holds:
   (a) $t_j \geq_{\mathsf{lp}} s$ for some j;
   (b) $\mathsf{g} > \mathsf{f}$ and $t >_{\mathsf{lp}} s_i$ for all i; or
   (c) $\mathsf{g} = \mathsf{f}$, $t >_{\mathsf{lp}} s_i$ for all i, and $\bar{t}_{j-1} = \bar{s}_{j-1}$, $s_j >_{\mathsf{lp}} t_j$ for some j.

The main idea behind this term order is to compare the heads of terms, i.e., the leftmost function symbols (condition 2b). If they are equal, the order recursively compares the arguments (condition 2c). Condition 2a ensures the subterm property. Accordingly, conditions 2b and 2c must check that condition 2a cannot be applied in the opposite direction. Nonground terms are rarely comparable, but condition 1 allows us to compare some of them while preserving stability under substitutions.

For example, given the precedence $\mathsf{h} > \mathsf{g} > \mathsf{f} > \mathsf{b} > \mathsf{a}$, we have

$$\mathsf{g}(\mathsf{a}) >_{\mathsf{lp}} \mathsf{f}(\mathsf{b}) \quad \mathsf{h}(\mathsf{a},\mathsf{b},\mathsf{a}) >_{\mathsf{lp}} \mathsf{h}(\mathsf{a},\mathsf{a},\mathsf{b}) \quad \mathsf{f}(\mathsf{g}(\mathsf{b})) >_{\mathsf{lp}} \mathsf{g}(\mathsf{a}) \quad \mathsf{h}(x,\mathsf{f}(y),z) >_{\mathsf{lp}} \mathsf{h}(x,y,z)$$

The terms $\mathsf{h}(x,\mathsf{f}(y),z)$ and $\mathsf{h}(z,y,x)$, however, are incomparable.

The name of the lexicographic path order stems from condition 2c, which compares the arguments lexicographically. The *recursive path order* generalizes this condition and allows for various ways to compare the arguments recursively.

Knuth–Bendix Order The Knuth–Bendix order [8, 87] is another term order that fulfills the requirements imposed by superposition.

In addition to a precedence $>$ on the signature Σ, the Knuth–Bendix order is parameterized by a weight function $\mathcal{W} : \Sigma \to \mathbb{R}_{>0}$. The weight function is extended to variables x by assigning all of them the same weight $\mathcal{W}(x) = w_0$. It is extended to terms $t = \mathsf{f}(\bar{t})$ via the recursive equation

$$\mathcal{W}(t) = \mathcal{W}(\mathsf{f}) + \sum_i \mathcal{W}(t_i)$$

Definition 2.2. Given a precedence $>$ and a weight function $\mathcal{W}$, the *Knuth–Bendix order* $>_{\mathsf{kb}}$ on first-order terms induced by $>$ and $\mathcal{W}$ is defined as follows: Let $t >_{\mathsf{kb}} s$ if

1. $|t|_x \geq |s|_x$ for all $x \in \mathcal{V}$ and $\mathcal{W}(t) > \mathcal{W}(s)$; or
2. $|t|_x \geq |s|_x$ for all $x \in \mathcal{V}$, $\mathcal{W}(t) = \mathcal{W}(s)$, and one of the following conditions holds:
   (a) $t = \mathsf{g}(\bar{t}_n)$, $s = \mathsf{f}(\bar{s}_m)$, and $\mathsf{g} > \mathsf{f}$; or
   (b) $t = \mathsf{g}(\bar{t}_n)$, $s = \mathsf{g}(\bar{s}_n)$, and $\bar{t}_{j-1} = \bar{s}_{j-1}$, $s_j >_{\mathsf{kb}} t_j$ for some j.

Here, $|t|_x$ denotes the number of times a variable x occurs in t.

For example, given the precedence $\mathsf{h} > \mathsf{g} > \mathsf{f} > \mathsf{c} > \mathsf{b} > \mathsf{a}$, where all weights are 1 except for $\mathcal{W}(\mathsf{a}) = 5$, we have

$$\mathsf{a} >_{\mathsf{kb}} \mathsf{g}(\mathsf{f}(\mathsf{b})) \quad \mathsf{g}(\mathsf{h}(\mathsf{b},\mathsf{b},\mathsf{b})) >_{\mathsf{kb}} \mathsf{a} \quad \mathsf{h}(\mathsf{a},\mathsf{b},\mathsf{c}) >_{\mathsf{kb}} \mathsf{h}(\mathsf{c},\mathsf{b},\mathsf{a}) \quad \mathsf{h}(x,\mathsf{f}(y),z) >_{\mathsf{kb}} \mathsf{h}(z,y,x)$$

whereas a and $\mathsf{g}(\mathsf{f}(x))$ are incomparable.

2.2.3. The Inference Rules

The core of superposition is an inference system—i.e, rules stating how to derive new clauses from existing clauses. These rules are superposition (Sup), equality resolution (ERes), and equality factoring (EFact), as stated below.

Besides the term order, superposition is parameterized by a literal selection function that restricts the search space even further. This selection function maps each clause C to a subclause of C consisting of negative literals. These literals are called *selected* in C. A literal is *maximal* in a clause C if it is greater than or equal to every literal in C. It is *strictly maximal* if it is maximal and occurs only once in C. A literal L is (*strictly*) *eligible* w.r.t. a substitution σ in C if it is selected in C or there are no selected literals in C and $L\sigma$ is (strictly) maximal in $C\sigma$.

We write inference rules using the standard notation placing the premises above and the conclusion below a horizontal bar. As a general convention, we assume variables from different clauses to be different. In implementations, this is usually achieved by renaming them apart before performing an inference.

$$\frac{\overbrace{D' \vee t \approx t'}^{D} \quad \overbrace{C' \vee s[u] \mathrel{\dot{\approx}} s'}^{C}}{(D' \vee C' \vee s[t'] \mathrel{\dot{\approx}} s')\sigma}\ \textsc{Sup}$$

where $\dot{\approx}$ denotes $\approx$ or $\not\approx$ and the following conditions hold:

1. $\sigma = \mathrm{mgu}(t,u)$;
2. $t\sigma \not\preceq t'\sigma$;
3. $s[u]\sigma \not\preceq s'\sigma$;
4. $t \approx t'$ is strictly eligible w.r.t. σ in D;
5. $C\sigma \not\preceq D\sigma$;
6. $s[u] \mathrel{\dot{\approx}} s'$ is eligible w.r.t. σ in C and, if positive, even strictly eligible; and
7. $u \notin \mathcal{V}$.

Some authors distinguish between a positive Sup rule where $s[u] \mathrel{\dot{\approx}} s'$ is positive and a negative Sup rule where $s[u] \mathrel{\dot{\approx}} s'$ is negative.

$$\frac{\overbrace{C' \vee s \not\approx s'}^{C}}{C'\sigma}\ \textsc{ERes}$$

where

1. $\sigma = \mathrm{mgu}(s,s')$;
2. $s \not\approx s'$ is eligible w.r.t. σ in C.

$$\frac{\overbrace{C' \vee s' \approx t' \vee s \approx t}^{C}}{(C' \vee t \not\approx t' \vee s \approx t')\sigma}\ \textsc{EFact}$$

where

1. $\sigma = \mathrm{mgu}(s, s')$; 2. $s\sigma \not\preceq t\sigma$; 3. $s \approx t$ is eligible w.r.t. σ in C.

These inference rules must be applied to the CNF of the assumptions, the CNF of the negated conjecture, and all clauses emerging in this way until either the empty clause, written as $\bot$, is derived or the clause set is saturated—i.e., all possible inferences have been performed. Refutational completeness guarantees that superposition will eventually derive $\bot$ if the conjecture is entailed by the assumptions. If it is not entailed, the process either saturates after finitely many inference steps or does not terminate at all.

Example 2.3. The following example illustrates the calculus. Given some properties about real numbers, we want to show that

$$\frac{x}{y} + 1 = \frac{x+y}{y} \quad \text{if } y \neq 0$$

In first-order logic, we can express this conjecture as

$$\forall x.\, \forall y.\, y \not\approx 0 \rightarrow \mathsf{add}(\mathsf{div}(x, y), 1) \approx \mathsf{div}(\mathsf{add}(x, y), y)$$

where add, div, 0, and 1 are uninterpreted symbols. Negating this conjecture and bringing it into CNF yields the two clauses $\mathsf{sk_y} \not\approx 0$ and $C_{\mathsf{conj}} = \mathsf{add}(\mathsf{div}(\mathsf{sk_x}, \mathsf{sk_y}), 1) \not\approx \mathsf{div}(\mathsf{add}(\mathsf{sk_x}, \mathsf{sk_y}), \mathsf{sk_y})$ where $\mathsf{sk_x}$ and $\mathsf{sk_y}$ are fresh constants, known as Skolem constants. To make the conjecture provable, we assume that we are given some properties of the real numbers that can be clausified into the following clauses: $C_{\mathsf{inv}} = \mathsf{mul}(x, \mathsf{inv}(x)) \approx 1 \lor x \approx 0$, $C_{\mathsf{div}} = \mathsf{mul}(x, \mathsf{inv}(y)) \approx \mathsf{div}(x, y) \lor y \approx 0$, and $C_{\mathsf{distr}} = \mathsf{add}(\mathsf{mul}(x, z), \mathsf{mul}(y, z)) = \mathsf{mul}(\mathsf{add}(x, y), z)$.

Applying the inference rules above, we can then derive the empty clause $\bot$ as follows, using a selection function that selects all negative literals of a clause and using the Knuth–Bendix order with weights $\mathcal{W}(\mathsf{c}) = \mathcal{W}(x) = 1$ for all $\mathsf{c} \in \Sigma$, $x \in \mathcal{V}$ and with a precedence such that $\mathsf{add} > \mathsf{div}$ and $\mathsf{sk_y} > 0$.

$$\frac{C_{\mathsf{div}} \quad C_{\mathsf{distr}}}{\mathsf{add}(\mathsf{div}(x, w), \mathsf{mul}(y, \mathsf{inv}(w))) \approx \mathsf{mul}(\mathsf{add}(x, y), \mathsf{inv}(w)) \lor w \approx 0}\ \textsc{Sup}$$

$$\frac{C_{\mathsf{inv}} \quad \mathsf{add}(\mathsf{div}(x, w), \mathsf{mul}(y, \mathsf{inv}(w))) \approx \mathsf{mul}(\mathsf{add}(x, y), \mathsf{inv}(w)) \lor w \approx 0}{\mathsf{add}(\mathsf{div}(x, y), 1) \approx \mathsf{mul}(\mathsf{add}(x, y), \mathsf{inv}(y)) \lor y \approx 0 \lor y \approx 0}\ \textsc{Sup}$$

$$\frac{C_{\mathsf{div}} \quad \mathsf{add}(\mathsf{div}(x, y), 1) \approx \mathsf{mul}(\mathsf{add}(x, y), \mathsf{inv}(y)) \lor y \approx 0 \lor y \approx 0}{\mathsf{add}(\mathsf{div}(x, y), 1) \approx \mathsf{div}(\mathsf{add}(x, y), y) \lor y \approx 0 \lor y \approx 0 \lor y \approx 0}\ \textsc{Sup}$$

$$\frac{C_{\mathsf{conj}} \quad \mathsf{add}(\mathsf{div}(x, y), 1) \approx \mathsf{div}(\mathsf{add}(x, y), y) \lor y \approx 0 \lor y \approx 0 \lor y \approx 0}{\mathsf{div}(\mathsf{add}(\mathsf{sk_x}, \mathsf{sk_y}), \mathsf{sk_y}) \not\approx \mathsf{div}(\mathsf{add}(\mathsf{sk_x}, \mathsf{sk_y}), \mathsf{sk_y}) \lor \mathsf{sk_y} \approx 0 \lor \mathsf{sk_y} \approx 0 \lor \mathsf{sk_y} \approx 0}\ \textsc{Sup}$$

$$\frac{\mathsf{div}(\mathsf{add}(\mathsf{sk_x}, \mathsf{sk_y}), \mathsf{sk_y}) \not\approx \mathsf{div}(\mathsf{add}(\mathsf{sk_x}, \mathsf{sk_y}), \mathsf{sk_y}) \lor \mathsf{sk_y} \approx 0 \lor \mathsf{sk_y} \approx 0 \lor \mathsf{sk_y} \approx 0}{\mathsf{sk_y} \approx 0 \lor \mathsf{sk_y} \approx 0 \lor \mathsf{sk_y} \approx 0}\ \textsc{ERes}$$

$$\frac{\mathsf{sk_y} \not\approx 0 \quad \mathsf{sk_y} \approx 0 \lor \mathsf{sk_y} \approx 0 \lor \mathsf{sk_y} \approx 0}{0 \not\approx 0 \lor \mathsf{sk_y} \approx 0 \lor \mathsf{sk_y} \approx 0}\ \textsc{Sup}$$

$$\frac{0 \not\approx 0 \lor \mathsf{sk_y} \approx 0 \lor \mathsf{sk_y} \approx 0}{\mathsf{sk_y} \approx 0 \lor \mathsf{sk_y} \approx 0}\ \textsc{ERes}$$

$$\frac{\mathsf{sk_y} \not\approx 0 \quad \mathsf{sk_y} \approx 0 \lor \mathsf{sk_y} \approx 0}{0 \not\approx 0 \lor \mathsf{sk_y} \approx 0}\ \textsc{Sup}$$

$$\frac{0 \not\approx 0 \lor \mathsf{sk_y} \approx 0}{\mathsf{sk_y} \approx 0}\ \textsc{ERes}$$

$$\frac{\mathsf{sk_y} \not\approx 0 \quad \mathsf{sk_y} \approx 0}{0 \not\approx 0}\ \textsc{Sup}$$

$$\frac{0 \not\approx 0}{\bot}\ \textsc{ERes}$$

Only the part of the derivation here that eventually leads to the empty clause is displayed here. In practice, many more clauses would be derived.

2.2.4. Redundancy and Simplification

In addition to these core inference rules, superposition has a redundancy criterion that allows us to add simplification rules that preserve refutational completeness. Simplification rules are crucial for the performance of superposition provers because they replace undirected proof search by computation.

A ground clause C is redundant w.r.t. a set of ground clauses N if it is entailed by the clauses in N that are smaller than C. A (possibly nonground) clause C is redundant w.r.t. a set of clauses N if all of its ground instances are redundant w.r.t. the ground instances of clauses in N. By ground instance of a clause C, we mean a ground clause of the form $C\theta$ for some substitution θ.

Some basic examples of simplification rules are deletion of duplicated literals DD and deletion of resolved literals DR [117]:

$$\frac{s \mathrel{\dot{\approx}} t \vee s \mathrel{\dot{\approx}} t \vee C}{s \mathrel{\dot{\approx}} t \vee C}\,\mathrm{DD} \qquad \frac{s \not\approx s \vee C}{C}\,\mathrm{DR}$$

The double bar indicates that these are simplification rules—i.e., that the conclusion makes the premise redundant and can replace them. Clearly, these rules can substantially shorten the bottom part of the derivation in Example 2.3.

2.3. Term Rewriting

Term rewriting systems [8,133] are a core component of the refutational completeness proof of superposition. They are an instance of the more general concept of abstract reduction systems.

Abstract Reduction Systems An *abstract reduction system* is a pair $(A, \longrightarrow)$, where A is an arbitrary set and the reduction relation $\longrightarrow$ is a binary relation on A. Formally, we view a binary relation on A as a subset of $A \times A$, but we write $a \longrightarrow b$ for $(a,b) \in \longrightarrow$. The *converse* $\longleftarrow$ of a relation $\longrightarrow$ is the relation such that $b \longleftarrow a$ if $a \longrightarrow b$. The *symmetric closure* of a relation $\longrightarrow$ is defined as $\longleftrightarrow = (\longrightarrow \cup \longleftarrow)$. The *reflexive transitive closure* $\longrightarrow^*$ of a relation $\longrightarrow$ is the relation such that $a \longrightarrow^* b$ if there exists a tuple $\bar{c}_n$ with $n > 0$ such that $a = c_1$, $b = c_n$, and $c_i \longrightarrow c_{i+1}$ for all i. The *transitive closure* $\longrightarrow^+$ additionally requires that $n > 1$. We write $\longleftrightarrow^*$ for the *reflexive transitive symmetric closure* $(\longleftrightarrow)^*$ and $\longleftrightarrow^+$ for the *transitive symmetric closure* $(\longleftrightarrow)^+$. We write chained reductions $a \longrightarrow_1 b \longrightarrow_2 c$ to stand for $a \longrightarrow_1 b$ and $b \longrightarrow_2 c$. We define the *composition* of two relations $\longrightarrow_1$ and $\longrightarrow_2$ as $\longrightarrow_1 \circ \longrightarrow_2 = \{(a,c) \in A \times A \mid a \longrightarrow_1 b \longrightarrow_2 c \text{ for some } b\}$.

We call an element a *reducible* if there exists a b such that $a \longrightarrow b$. It is *in normal form* (or irreducible) otherwise. An element b is a *normal form* of a if $a \longrightarrow^* b$ and b is in normal form. If the normal form of a is unique, we write $a{\downarrow}$ for that normal form. A reduction relation $\longrightarrow$ is *terminating* if there exists no infinite descending chain $a_1 \longrightarrow a_2 \longrightarrow \cdots$. A reduction relation $\longrightarrow$ is *confluent* if for all a, b, and b' such that $b \longleftarrow^* a \longrightarrow^* b'$, there exists a c such that $b \longrightarrow^* c \longleftarrow^* b$.

An important property of abstract reduction systems is the following: If $\longrightarrow$ is

terminating and confluent, then every element a has a unique normal form $a\downarrow$ and moreover $a \leftrightarrow^* b$ if and only if $a\downarrow = b\downarrow$ [8, Section 2.1.2].

Term Rewriting Systems Term rewriting systems are a type of reduction system that operates on (first-order) terms. A *term rewriting system* is a set of rewrite rules. A *rewrite rule* is a pair of terms t and s, which we write as $t \longrightarrow s$, such that t is not a variable and all variables in s also occur in t.

Given a term rewriting system R, we define the *reduction relation* $\longrightarrow_R$ as follows: Let $s \longrightarrow_R t$ if there exist a rule $u \longrightarrow v \in R$, a position p in s, and a substitution ρ such that $s|_p = u\rho$ and $t = s[v\rho]_p$. We say that R is terminating, confluent, etc. when $\longrightarrow_R$ is.

A *critical pair* of a term rewriting system R is a pair of two terms s and t such that there exist rules $u \longrightarrow v \in R$ and $u' \longrightarrow v' \in R$, a position p in u, and a unifier $\sigma = \mathrm{mgu}(u|_p, u')$ where $u|_p$ is not a variable, $s = v\sigma$, and $t = (u\sigma)[v'\sigma]_p$.

The critical pair theorem for terminating term rewriting systems [8, Corollary 6.2.5] states that a terminating term rewriting system R is confluent if for each critical pair s and t, there exists a term u such that $s \longrightarrow^* u \longleftarrow^* t$. In particular, if there are no critical pairs, R is confluent.

3

Superposition for Lambda-Free Higher-Order Logic

Joint work with
Jasmin Blanchette, Simon Cruanes, and Uwe Waldmann

We introduce refutationally complete superposition calculi for intentional and extensional clausal λ-free higher-order logic, two formalisms that allow partial application and applied variables. The calculi are parameterized by a term order that need not be fully monotonic, making it possible to employ the λ-free higher-order lexicographic path and Knuth–Bendix orders. We implemented the calculi in the Zipperposition prover and evaluated them on Isabelle/HOL and TPTP benchmarks.

My contributions to this chapter are the design of the four calculi, the redundancy criterion, the completeness proof, the implementation, and the evaluation.

Parts of this chapter have been published at the International Joint Conference on Automated Reasoning (IJCAR 2018), LNCS 10900, pp. 28–46, Springer, 2018. This chapter has been accepted to be published in the journal Logical Methods in Computer Science.

3.1. Introduction

As a first milestone towards full higher-order logic, in this chapter we restrict our attention to a clausal λ-free fragment of polymorphic higher-order logic that supports partial application and application of variables (Section 3.2). This formalism is expressive enough to permit the axiomatization of higher-order combinators such as $\mathsf{pow} : \Pi\alpha.\, nat \to (\alpha \to \alpha) \to \alpha \to \alpha$ (intended to denote the iterated application $h^n\, x$):

$$\mathsf{pow}\langle\alpha\rangle\, \mathsf{Zero}\, h \approx \mathsf{id}\langle\alpha\rangle \qquad \mathsf{pow}\langle\alpha\rangle\, (\mathsf{Succ}\, n)\, h\, x \approx h\, (\mathsf{pow}\langle\alpha\rangle\, n\, h\, x)$$

Conventionally, functions are applied without parentheses and commas, and variables are italicized. Notice the variable number of arguments to $\mathsf{pow}\langle\alpha\rangle$ and the application of h. The expressiveness of full higher-order logic can be recovered by introducing SK combinators to represent λ-abstractions and proxies for the logical symbols [84, 115].

A widespread technique to support partial application and application of variables in first-order logic is to make all symbols nullary and to represent application of functions by a distinguished binary symbol $\mathsf{app} : \Pi\alpha, \beta.\, \mathit{fun}(\alpha, \beta) \times \alpha \to \beta$, where *fun* is an uninterpreted binary type constructor. Following this scheme, the higher-order term $\mathsf{f}\,(h\,\mathsf{f})$, where $\mathsf{f} : \kappa \to \kappa'$, is translated to $\mathsf{app}(\mathsf{f}, \mathsf{app}(h, \mathsf{f}))$—or rather $\mathsf{app}\langle\kappa, \kappa'\rangle(\mathsf{f}, \mathsf{app}\langle \mathit{fun}(\kappa, \kappa'), \kappa\rangle(h, \mathsf{f}))$ if we specify the type arguments. We call this the *applicative encoding*. The existence of such a reduction to first-order logic explains why λ-free higher-order terms are also called "applicative first-order terms." Unlike for full higher-order logic, most general unifiers are unique for our λ-free fragment, just as they are for applicatively encoded first-order terms.

Although the applicative encoding is complete [84] and is employed fruitfully in tools such as HOLyHammer and Sledgehammer [31], it suffers from a number of weaknesses, all related to its gracelessness. Transforming all the function symbols into constants considerably restricts what can be achieved with term orders; for example, argument tuples cannot easily be compared using different methods for different symbols [90, Section 2.3.1]. In a prover, the encoding also clutters the data structures, slows down the algorithms, and neutralizes the heuristics that look at the terms' root symbols. But our chief objection is the sheer clumsiness of encodings and their poor integration with interpreted symbols. And they quickly accumulate; for example, using the traditional encoding of polymorphism relying on a distinguished binary function symbol t [30, Section 3.3] in conjunction with the applicative encoding, the term $\mathsf{Succ}\, x$ becomes $\mathsf{t}(\mathsf{nat}, \mathsf{app}(\mathsf{t}(\mathsf{fun}(\mathsf{nat}, \mathsf{nat}), \mathsf{Succ}), \mathsf{t}(\mathsf{nat}, x)))$. The term's simple structure is lost in translation.

Hybrid schemes have been proposed to strengthen the applicative encoding: If a given symbol always occurs with at least k arguments, these can be passed directly [106]. However, this relies on a closed-world assumption: that all terms that will ever be compared arise in the initial problem. This noncompositionality conflicts with the need for complete higher-order calculi to synthesize arbitrary terms during proof search [21]. As a result, hybrid encodings are not an ideal basis for higher-order automated reasoning.

Instead, we propose to generalize the superposition calculus to *intensional* and *extensional* clausal λ-free higher-order logic. For the extensional version of the logic,

the property $(\forall x.\, h\, x \approx k\, x) \longrightarrow h \approx k$ holds for all functions h, k of the same type. For each logic, we present two calculi (Section 3.3). The intentional calculi perfectly coincide with standard superposition on first-order clauses; the extensional calculi depend on an extra axiom.

Superposition is parameterized by a term order, which is used to prune the search space. If we assume that the term order is a simplification order enjoying totality on ground terms (i.e., terms containing no term or type variables), the standard calculus rules and completeness proof can be lifted verbatim. The only necessary changes concern the basic definitions of terms and substitutions. However, there is one monotonicity property that is hard to obtain unconditionally: *compatibility with arguments*. It states that $s' \succ s$ implies $s'\, t \succ s\, t$ for all terms s, s', t such that $s\, t$ and $s'\, t$ are well typed. Blanchette, Waldmann, and colleagues recently introduced graceful generalizations of the lexicographic path order (LPO) [34] and the Knuth–Bendix order (KBO) [16] with argument coefficients, but they both lack this property. For example, given a KBO with $\mathsf{g} \succ \mathsf{f}$, it may well be that $\mathsf{g}\,\mathsf{a} \prec \mathsf{f}\,\mathsf{a}$ if f has a large enough multiplier on its argument. This lack of compatibility with arguments makes the standard superposition rules incomplete—e.g., no inferences are applicable to the unsatisfiable clause set $\{\mathsf{g} \approx \mathsf{f},\, \mathsf{g}\,\mathsf{a} \not\approx \mathsf{f}\,\mathsf{a}\}$ [26, Example 3].

Although there exist fully monotonic orders for λ-free higher-order terms, such as KBO without argument coefficients, we study nonmonotonic orders in anticipation of λ-expressions and β-reduction in subsequent chapters where compatibility with arguments is impossible in conjunction with totality on ground (i.e., closed) terms. If, for instance, $\lambda x y.\, y \succ \lambda x y.\, x$ and $\mathsf{b} \succ \mathsf{a}$ then $(\lambda x y.\, y)\,\mathsf{b}\,\mathsf{a} =_\beta \mathsf{a} \prec \mathsf{b} =_\beta (\lambda x y.\, x)\,\mathsf{b}\,\mathsf{a}$, violating compatibility with arguments. If instead $\lambda x y.\, y \prec \lambda x y.\, x$, then compatibility with arguments is violated by $(\lambda x y.\, y)\,\mathsf{a}\,\mathsf{b} =_\beta \mathsf{b} \succ \mathsf{a} =_\beta (\lambda x y.\, x)\,\mathsf{a}\,\mathsf{b}$.

Our superposition calculi are designed to be refutationally complete for orders lacking compatibility with arguments (Section 3.4). To achieve this, they include an inference rule for argument congruence, which derives $C \vee s\, x \approx t\, x$ from $C \vee s \approx t$. The redundancy criterion is defined in such a way that the larger, derived clause is not subsumed by the premise. In the completeness proof, the most difficult case is the one that normally excludes superposition at or below variables using the induction hypothesis. With nonmonotonicity, this approach no longer works, and we propose two alternatives: Either perform some superposition inferences into higher-order variables or "purify" the clauses to circumvent the issue. We refer to the corresponding calculi as *nonpurifying* and *purifying*.

The calculi are implemented in the Zipperposition prover [47] (Section 3.5). We evaluate them on first- and higher-order Isabelle/HOL [40] and TPTP benchmarks [131, 132] and compare them with the applicative encoding (Section 3.6). We find that there is a substantial cost associated with the applicative encoding, that the nonmonotonicity is not particularly expensive, and that the nonpurifying calculi outperform the purifying calculi.

3.2. Logic

Our logic is intended as an intermediate step on the way towards full higher-order logic [45, 67]. Refutational completeness of calculi for higher-order logic is usually

stated in terms of Henkin semantics [21,71], in which the universes used to interpret functions need only contain the functions that can be expressed as terms. Since the terms of λ-free higher-order logic exclude λ-abstractions, in "λ-free Henkin semantics" the universes interpreting functions can be even smaller. In that sense, our semantics resemble Henkin prestructures [97, Section 5.4]. In contrast to other higher-order logics [135], there are no comprehension principles, and we disallow nesting of Boolean formulas inside terms.

3.2.1. Syntax

We fix a set Σ_{ty} of type constructors with arities and a set $\mathcal{V}_{\mathsf{ty}}$ of type variables. We require at least one nullary type constructor and a binary type constructor $\to$ to be present in Σ_{ty}. We inductively define a λ-free higher-order type to be either a type variable $\alpha \in \mathcal{V}_{\mathsf{ty}}$ or of the form $\kappa(\bar{\tau}_n)$ for an n-ary type constructor $\kappa \in \Sigma_{\mathsf{ty}}$ and types $\bar{\tau}_n$. We write κ for $\kappa()$ and $\tau \to \upsilon$ for $\to(\tau, \upsilon)$. A type declaration is an expression of the form $\Pi\bar{\alpha}_m.\ \tau$ (or simply τ if $m = 0$), where all type variables occurring in τ belong to $\bar{\alpha}_m$.

We fix a set Σ of symbols with type declarations, written as $\mathsf{f} : \Pi\bar{\alpha}_m.\ \tau$ or f, and a set $\mathcal{V}$ of typed variables, written as $x : \tau$ or x. We require Σ to contain a symbol with type declaration $\Pi\alpha.\ \alpha$, to ensure that the (Herbrand) domain of every type is nonempty. The sets $(\Sigma_{\mathsf{ty}}, \Sigma)$ form the logic's signature. We reserve the letters s, t, u, υ, w for terms and x, y, z for variables and write $: \tau$ to indicate their type. The set of λ-free higher-order terms is defined inductively as follows. Every variable $x : \tau \in \mathcal{V}$ is a term. If $\mathsf{f} : \Pi\bar{\alpha}_m.\ \tau$ is a symbol and $\bar{\upsilon}_m$ are types, then $\mathsf{f}\langle\bar{\upsilon}_m\rangle : \tau\{\bar{\alpha}_m \mapsto \bar{\upsilon}_m\}$ is a term. If $t : \tau \to \upsilon$ and $u : \tau$, then $t\ u : \upsilon$ is a term, called an *application*. Nonapplication terms are called *heads*. Application is left-associative, and correspondingly the function type constructor $\to$ is right-associative. Using the spine notation [44], terms can be decomposed in a unique way as a head t applied to zero or more arguments: $t\, s_1 \ldots s_n$ or $t\, \bar{s}_n$ (abusing notation). A term is *ground* if it is built without using type or term variables.

Substitution and unification are generalized in the obvious way, without the difficulties caused by λ-abstractions. A substitution has the form $\{\bar{\alpha}_m, \bar{x}_n \mapsto \bar{\upsilon}_m, \bar{s}_n\}$, where each x_j has type τ_j and each s_j has type $\tau_j\{\bar{\alpha}_m \mapsto \bar{\upsilon}_m\}$, mapping m type variables to m types and n term variables to n terms. A unifier of two terms s and t is a substitution ρ such that $s\rho = t\rho$. A most general unifier $\mathrm{mgu}(s, t)$ of two terms s and t is a unifier σ of s and t such that for every other unifier θ, there exists a substitution ρ such that $\alpha\theta = \alpha\sigma\rho$ and $x\theta = x\sigma\rho$ for all $\alpha \in \mathcal{V}_{\mathsf{ty}}$ and all $x \in \mathcal{V}$. As in first-order logic, the most general unifier is unique up to variable renaming. For example, $\mathrm{mgu}(x\,\mathsf{b}\,z, \mathsf{f}\,\mathsf{a}\,y\,\mathsf{c}) = \{x \mapsto \mathsf{f}\,\mathsf{a}, y \mapsto \mathsf{b}, z \mapsto \mathsf{c}\}$, and $\mathrm{mgu}(y\,(\mathsf{f}\,\mathsf{a}), \mathsf{f}\,(y\,\mathsf{a})) = \{y \mapsto \mathsf{f}\}$, assuming that the types of the unified subterms are equal.

An equation $s \approx t$ is formally an unordered pair of terms s and t of the same type. A literal is an equation or a negated equation, written $s \not\approx t$. A clause $L_1 \vee \cdots \vee L_n$ is a finite multiset of literals L_j. The empty clause is written as $\bot$.

3.2.2. Semantics

A *type interpretation* $\mathcal{I}_{\mathsf{ty}} = (\mathcal{U}, \mathcal{J}_{\mathsf{ty}})$ is defined as follows. The set $\mathcal{U}$ is a nonempty collection of nonempty sets, called *universes*. The function $\mathcal{J}_{\mathsf{ty}}$ associates a function $\mathcal{J}_{\mathsf{ty}}(\kappa) : \mathcal{U}^n \to \mathcal{U}$ with each n-ary type constructor κ. A *type valuation* ξ is a function that maps every type variable to a universe. The *denotation* of a type for a type interpretation $\mathcal{I}_{\mathsf{ty}}$ and a type valuation ξ is defined by $[\![\alpha]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}} = \xi(\alpha)$ and $[\![\kappa(\bar{\tau})]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}} = \mathcal{J}_{\mathsf{ty}}(\kappa)([\![\bar{\tau}]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}})$. Here and elsewhere, we abuse notation by applying an operation on a tuple when it must be applied elementwise; thus, $[\![\bar{\tau}_n]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}$ stands for $[\![\tau_1]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}, \ldots, [\![\tau_n]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}$.

A type valuation ξ can be extended to be a *valuation* by additionally assigning an element $\xi(x) \in [\![\tau]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}$ to each variable $x : \tau$. An *interpretation function* $\mathcal{J}$ for a type interpretation $\mathcal{I}_{\mathsf{ty}}$ associates with each symbol $\mathsf{f} : \Pi\bar{\alpha}_m.\ \tau$ and universe tuple $\bar{U}_m \in \mathcal{U}^m$ a value $\mathcal{J}(\mathsf{f}, \bar{U}_m) \in [\![\tau]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}$, where ξ is the type valuation that maps each α_i to U_i. Loosely following Fitting [59, Section 2.5], an *extension function* $\mathcal{E}$ associates to any pair of universes $U_1, U_2 \in \mathcal{U}$ a function $\mathcal{E}_{U_1,U_2} : \mathcal{J}_{\mathsf{ty}}(\to)(U_1, U_2) \to (U_1 \to U_2)$. Together, a type interpretation, an interpretation function, and an extension function form an *interpretation* $\mathcal{I} = (\mathcal{U}, \mathcal{J}_{\mathsf{ty}}, \mathcal{J}, \mathcal{E})$.

An interpretation is *extensional* if $\mathcal{E}_{U_1,U_2}$ is injective for all U_1, U_2. Both intensional and extensional logics are widely used for interactive theorem proving; for example, Coq's calculus of inductive constructions is intensional [25], whereas Isabelle/HOL is extensional [108]. The semantics is *standard* if $\mathcal{E}_{U_1,U_2}$ is bijective for all U_1, U_2.

For an interpretation $\mathcal{I} = (\mathcal{U}, \mathcal{J}_{\mathsf{ty}}, \mathcal{J}, \mathcal{E})$ and a valuation ξ, let the denotation of a term be defined as follows: For variables x, let $[\![x]\!]^{\xi}_{\mathcal{I}} = \xi(x)$. For symbols f, let $[\![\mathsf{f}\langle\bar{\tau}\rangle]\!]^{\xi}_{\mathcal{I}} = \mathcal{J}(\mathsf{f}, [\![\bar{\tau}]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}})$. For applications $s\ t$ of a term $s : \tau \to \upsilon$ to a term $t : \tau$, let $U_1 = [\![\tau]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}$, $U_2 = [\![\upsilon]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}$, and $[\![s\ t]\!]^{\xi}_{\mathcal{I}} = \mathcal{E}_{U_1,U_2}([\![s]\!]^{\xi}_{\mathcal{I}})([\![t]\!]^{\xi}_{\mathcal{I}})$. If t is a ground term, we also write $[\![t]\!]_{\mathcal{I}}$ for the denotation of t because it does not depend on the valuation.

An equation $s \approx t$ is true in $\mathcal{I}$ for ξ if $[\![s]\!]^{\xi}_{\mathcal{I}} = [\![t]\!]^{\xi}_{\mathcal{I}}$; otherwise, it is false. A disequation $s \not\approx t$ is true if $s \approx t$ is false. A clause is true if at least one of its literals is true. The interpretation $\mathcal{I}$ is a model of a clause C, written $\mathcal{I} \models C$, if C is true in $\mathcal{I}$ for all valuations ξ. It is a model of a set of clauses if it is a model of all contained clauses.

For example, given the signature $(\{\kappa, \to\}, \{\mathsf{a} : \kappa\})$ and a variable $h : \kappa \to \kappa$, the clause $h\ \mathsf{a} \not\approx \mathsf{a}$ has an extensional model with $\mathcal{U} = \{U_1, U_2\}$, $U_1 = \{a, b\}$ $(a \neq b)$, $U_2 = \{f\}$, $\mathcal{J}_{\mathsf{ty}}(\kappa) = U_1$, $\mathcal{J}_{\mathsf{ty}}(\to)(U_1, U_1) = U_2$, $\mathcal{J}(\mathsf{a}) = a$, $\mathcal{E}_{U_1,U_1}(f)(a) = \mathcal{E}_{U_1,U_1}(f)(b) = b$.

3.3. The Calculi

We introduce four versions of the *Boolean-free λ-free higher-order superposition calculus*, articulated along two axes: intentional versus extensional, and nonpurifying versus purifying. To avoid repetitions, our presentation unifies them into a single framework.

3.3.1. The Inference Rules

To support nonmonotonic term orders, we restrict superposition inferences to *green subterms*, which are defined inductively as follows:

Definition 3.1 (Green subterms and contexts)**.** A term t' is a *green subterm* of t if $t = t'$ or if $t = s\ \bar{u}$ and t' is a green subterm of u_i for some i. We write $s\langle u\rangle$ to indicate that the subterm u of $s[u]$ is a green subterm. Correspondingly, we call the context $s\langle\,\rangle$ around a green subterm a *green context*.

By this definition, f and f a are subterms of f a b, but not green subterms. The green subterms of f a b are a, b, and f a b. Thus, [ ] a b and [ ] b are not green contexts, but f [ ] b, f a [ ], and [ ] are.

The calculi are parameterized by a partial order $\succ$ on terms that

- is well founded on ground terms;
- is total on ground terms;
- has the subterm property on ground terms;
- is *compatible with green contexts* on ground terms: $t' \succ t$ implies $s\langle t'\rangle \succ s\langle t\rangle$;
- is stable under grounding substitutions: $t \succ s$ implies $t\theta \succ s\theta$ for all substitutions θ grounding t and s.

The order need not be *compatible with arguments*: $s' \succ s$ need not imply $s'\,t \succ s\,t$, even on ground terms. The literal and clause orders are defined from $\succ$ as multiset extensions in the standard way [9]. Despite their names, the term, literal, and clause orders need not be transitive on nonground entities.

The λ-free higher-order generalizations of LPO [34] and KBO [16] as well as EPO [18] fulfill these requirements, with the caveat that they are defined on untyped terms. To use them on polymorphic terms, we can encode type arguments as term arguments and ignore the remaining type information.

Literal selection is supported. The selection function maps each clause C to a subclause of C consisting of negative literals. A literal L is (*strictly*) *eligible* w.r.t. a substitution σ in C if it is selected in C or there are no selected literals in C and $L\sigma$ is (strictly) maximal in $C\sigma$. If σ is the identity substitution, we leave it implicit.

The following four rules are common to all four calculi. We regard positive and negative superposition as two cases of the same rule

$$\frac{\overbrace{D' \vee t \approx t'}^{D} \qquad \overbrace{C' \vee s\langle u\rangle \mathrel{\dot{\approx}} s'}^{C}}{(D' \vee C' \vee s\langle t'\rangle \mathrel{\dot{\approx}} s')\sigma}\ \textsc{Sup}$$

where $\dot{\approx}$ denotes $\approx$ or $\not\approx$ and the following conditions are fulfilled:

1. $\sigma = \mathrm{mgu}(t, u)$;
2. $t\sigma \not\preceq t'\sigma$;
3. $s\langle u\rangle\sigma \not\preceq s'\sigma$;
4. $t \approx t'$ is strictly eligible w.r.t. σ in D;
5. $C\sigma \not\preceq D\sigma$;
6. $s\langle u\rangle \mathrel{\dot{\approx}} s'$ is eligible w.r.t. σ in C and, if positive, even strictly eligible;
7. the *variable condition* must hold, which is specified individually for each calculus below.

In each calculus, we will define the variable condition to coincide with the condition "$u \notin \mathcal{V}$" if the premises are first-order.

The equality resolution and equality factoring rules are almost identical to their

standard counterparts:

$$\frac{\overbrace{C' \vee s \not\approx s'}^{C}}{C'\sigma}\ \textsc{ERes}$$

where

1. $\sigma = \mathrm{mgu}(s, s')$;
2. $s \not\approx s'$ is eligible w.r.t. σ in C.

$$\frac{\overbrace{C' \vee s' \approx t' \vee s \approx t}^{C}}{(C' \vee t \not\approx t' \vee s \approx t')\sigma}\ \textsc{EFact}$$

where

1. $\sigma = \mathrm{mgu}(s, s')$;
2. $s\sigma \not\approx t\sigma$;
3. $s \approx t$ is eligible w.r.t. σ in C.

The following *argument congruence* rule compensates for the limitation that the superposition rule applies only to green subterms:

$$\frac{\overbrace{C' \vee s \approx s'}^{C}}{C'\sigma \vee s\sigma\, \bar{x} \approx s'\sigma\, \bar{x}}\ \textsc{ArgCong}$$

where

1. $s \approx s'$ is strictly eligible w.r.t. σ in C;
2. $\bar{x}$ is a nonempty tuple of distinct fresh variables;
3. σ is the most general type substitution that ensures well-typedness of the conclusion.

In particular, if s takes m arguments, there are m ArgCong conclusions for this literal, for which σ is the identity and $\bar{x}$ is a tuple of $1, \ldots, m-1$, or m variables. If the result type of s is a type variable, we have in addition infinitely many ArgCong conclusions, for which σ instantiates the type variable in the result type of s with $\bar{\alpha}_k \to \beta$ for some $k > 0$ and fresh type variables $\bar{\alpha}_k$ and β and for which $\bar{x}$ is a tuple of $m + k$ variables. In practice, the enumeration of the infinitely many conclusions must be interleaved with other inferences via some form of dovetailing.

For the **intensional nonpurifying** calculus, the variable condition of the Sup rule is as follows:

> Either $u \notin \mathcal{V}$ or there exists a grounding substitution θ with $t\sigma\theta \succ t'\sigma\theta$ and $C\sigma\theta \prec C\{u \mapsto t'\}\sigma\theta$.

This condition generalizes the standard condition that $u \notin \mathcal{V}$. The two coincide if C is first-order or if the term order is monotonic. In some cases involving nonmonotonicity, the variable condition effectively mandates Sup inferences at variable positions of the right premise, but never below. We will call theses inferences *at variables*.

For the **extensional nonpurifying** calculus, the variable condition uses the following definition.

Definition 3.2. A term of the form $x\,\bar{s}_n$, for $n \geq 0$, *jells* with a literal $t \approx t' \in D$ if $t = \tilde{t}\,\bar{y}_n$ and $t' = \tilde{t}'\,\bar{y}_n$ for some terms $\tilde{t}$, $\tilde{t}'$ and distinct variables $\bar{y}_n$ that do not occur elsewhere in D.

Using the naming convention from Definition 3.2 for $\tilde{t}'$, the variable condition can be stated as follows:

> If u has a variable head x and jells with the literal $t \approx t' \in D$, there must exist a grounding substitution θ with $t\sigma\theta \succ t'\sigma\theta$ and $C\sigma\theta \prec C''\sigma\theta$, where $C'' = C\{x \mapsto \tilde{t}'\}$.

If C is first-order, this amounts to $u \notin \mathcal{V}$. Since the order is compatible with green contexts, the substitution θ can exist only if x occurs applied in C.

Moreover, the extensional nonpurifying calculus has one additional rule, the positive extensionality rule, and one axiom, the extensionality axiom. The rule is

$$\frac{C' \vee s\,\bar{x} \approx s'\,\bar{x}}{C' \vee s \approx s'}\ \textsc{PosExt}$$

where

1. $\bar{x}$ is a tuple of distinct variables that do not occur in C', s, or s'
2. $s\,\bar{x} \approx s'\,\bar{x}$ is strictly eligible in the premise.

The extensionality axiom uses a Skolem symbol $\mathsf{diff} : \Pi\alpha, \beta.\, (\alpha \to \beta)^2 \to \alpha$ characterized by the axiom

$$x\,(\mathsf{diff}\langle\alpha,\beta\rangle\,x\,y) \not\approx y\,(\mathsf{diff}\langle\alpha,\beta\rangle\,x\,y) \vee x \approx y \qquad \textsc{(Ext)}$$

Unlike the nonpurifying calculi, the purifying calculi never perform superposition at variables. Instead, they rely on purification [41, 51, 116, 124] (also called abstraction) to circumvent nonmonotonicity. The idea is to rename apart problematic occurrences of a variable x in a clause to $x_1, \ldots, x_n$ and to add *purification literals* $x_1 \not\approx x$, …, $x_n \not\approx x$ to connect the new variables to x. We must then ensure that all clauses are purified, by processing the initial clause set and the conclusion of every inference or simplification.

In the **intensional purifying** calculus, the purification $\mathit{pure}(C)$ of clause C is defined as the result of the following procedure. Choose a variable x that occurs applied in C and also unapplied in a literal of C that is not of the form $x \not\approx y$. If no such variable exists, terminate. Otherwise, replace all unapplied occurrences of x in C by a fresh variable x' and add the purification literal $x' \not\approx x$. Then repeat these steps with another variable. The procedure terminates because the number of variables that can be chosen reduces with each step. For example,

$$\mathit{pure}(x\,\mathsf{a} \approx x\,\mathsf{b} \vee \mathsf{f}\,x \approx \mathsf{g}\,x) = x\,\mathsf{a} \approx x\,\mathsf{b} \vee \mathsf{f}\,x' \approx \mathsf{g}\,x' \vee x \not\approx x'$$

The variable condition is standard:

> The term u is not a variable.

The conclusion C of ArgCong is changed to $\mathit{pure}(C)$; the other rules preserve purity of their premises.

In the **extensional purifying** calculus, $\mathit{pure}(C)$ is defined as follows. Choose a variable x occurring in green subterms $x\,\bar{u}$ and $x\,\bar{v}$ in literals of C that are not of the form $x \not\approx y$, where $\bar{u}$ and $\bar{v}$ are distinct (possibly empty) term tuples. If no such variable exists, terminate. Otherwise, replace all green subterms $x\,\bar{v}$ with $x'\,\bar{v}$, where x' is fresh, and add the purification literal $x' \not\approx x$. Then repeat the procedure until no variable fulfilling the requirements is left. The procedure terminates because the number of variables fulfilling the requirements does not increase and with each step, the number of distinct tuples $\bar{u}$ and $\bar{v}$ fulfilling the requirements decreases for the chosen variable x. For example,

$$\mathit{pure}(x\,\mathsf{a} \approx x\,\mathsf{b} \vee \mathsf{f}\,x \approx \mathsf{g}\,x) = x\,\mathsf{a} \approx x'\,\mathsf{b} \vee \mathsf{f}\,x'' \approx \mathsf{g}\,x'' \vee x' \not\approx x \vee x'' \not\approx x$$

Like the extensional nonpurifying calculus, this calculus also contains the PosExt rule and axiom (Ext) introduced above. The variable condition is as follows:

> Either u has a nonvariable head or u does not jell with the literal $t \approx t' \in D$.

The conclusion E of each rule is changed to $\mathit{pure}(E)$, except for PosExt, which preserves purity.

Finally, we impose further restrictions on literal selection. In the nonpurifying calculi, a literal must not be selected if $x\,\bar{u}$ is a maximal term of the clause and the literal contains a green subterm $x\,\bar{v}$ with $\bar{v} \neq \bar{u}$. In the purifying calculi, a literal must not be selected if it contains a variable of functional type. These restrictions are needed in our completeness proof to show that superposition inferences below variables are redundant. For the purifying calculi, the restriction is also needed to avoid inferences whose conclusion is identical to their premise, such as

$$\frac{z\,\mathsf{a} \approx \mathsf{b} \vee z \not\approx x \vee \mathsf{f} \not\approx x}{z\,\mathsf{a} \approx \mathsf{b} \vee z' \not\approx \mathsf{f} \vee z' \not\approx z}\ \textsc{Eres}$$

where $\mathsf{f} \not\approx x$ is selected. For the nonpurifying calculi, it might be possible to avoid the restriction at the cost of a more elaborate argument.

Example 3.3. We illustrate the different variable condition with a few examples. Consider a left-to-right LPO [34] instance with precedence $\mathsf{h} \succ \mathsf{g} \succ \mathsf{f} \succ \mathsf{c} \succ \mathsf{b} \succ \mathsf{a}$. Given the clauses D and C of a superposition inference in which the grayed subterms are unified, we list below whether each calculus's variable condition is fulfilled (✓) or not.

D	C	intensional nonpurif.	extensional nonpurif.	intensional purifying	extensional purifying
$\mathsf{h} \approx \mathsf{g}$	$\mathsf{f}\,y \approx \mathsf{c}$				
$\mathsf{f}(\mathsf{h}\,\mathsf{a}) \approx \mathsf{h}$	$\mathsf{g}\,y \approx y\,\mathsf{b}$	✓	✓		
$\mathsf{h}\,x \approx \mathsf{f}\,x$	$\mathsf{g}(y\,\mathsf{b})\,y \approx \mathsf{a}$	✓		✓	
$x \approx \mathsf{c} \vee \mathsf{h}\,x \approx \mathsf{f}\,x$	$\mathsf{g}(y\,\mathsf{b})\,y \approx \mathsf{a}$	✓	✓	✓	✓

For the purifying calculi, the clauses would actually undergo purification first, but this has no impact on the variable conditions. In the last row, the term $y\,\mathsf{b}$ does not jell with $\mathsf{h}\,x \approx \mathsf{f}\,x \in D$ because x occurs also in the first literal of D.

Remark 3.4. In descriptions of first-order logic with equality, the property $y \approx y' \longrightarrow \mathsf{f}(\bar{x}, y, \bar{z}) \approx \mathsf{f}(\bar{x}, y', \bar{z})$ is often referred to as "function congruence." It seems natural to use the same label for the higher-order version $t \approx t' \longrightarrow s\,t \approx s\,t'$ and to call the companion property $s \approx s' \longrightarrow s\,t \approx s'\,t$ "argument congruence," whence the name ArgCong for our inference rule. This nomenclature is far from universal; for example, the Isabelle/HOL theorem *fun_cong* captures argument congruence and *arg_cong* captures function congruence.

3

3.3.2. Rationale for the Inference Rules

A key restriction of all four calculi is that they superpose only at green subterms, mirroring the term order's compatibility with green contexts. The ArgCong rule then makes it possible to simulate superposition at nongreen subterms:

Example 3.5. The clause $\mathsf{g} \approx \mathsf{f}$ cannot superpose into $\mathsf{g}\,\mathsf{a}\,\mathsf{b} \not\approx \mathsf{f}\,\mathsf{a}\,\mathsf{b}$ because g occurs in a nongreen context. Instead, we refute these two clauses as follows:

$$\textsc{ERes}\,\frac{\textsc{Sup}\,\dfrac{\textsc{ArgCong}\,\dfrac{\mathsf{g} \approx \mathsf{f}}{\mathsf{g}\,x_1\,x_2 \approx \mathsf{f}\,x_1\,x_2} \qquad \mathsf{g}\,\mathsf{a}\,\mathsf{b} \not\approx \mathsf{f}\,\mathsf{a}\,\mathsf{b}}{\mathsf{f}\,\mathsf{a}\,\mathsf{b} \not\approx \mathsf{f}\,\mathsf{a}\,\mathsf{b}}}{\bot}$$

The ArgCong inference adds two arguments to g, yielding the term $\mathsf{g}\,x_1\,x_2$, which is unifiable with the green subterm $\mathsf{g}\,\mathsf{a}\,\mathsf{b}$. Thus we can apply Sup to the resulting clause.

However, in conjunction with the Sup rules, ArgCong can exhibit an unpleasant behavior, which we call *argument congruence explosion*:

$$\textsc{Sup}\,\frac{\textsc{ArgCong}\,\dfrac{\mathsf{g} \approx \mathsf{f}}{\mathsf{g}\,x \approx \mathsf{f}\,x} \qquad h\,\mathsf{a} \not\approx \mathsf{b}}{\mathsf{f}\,\mathsf{a} \not\approx \mathsf{b}} \qquad\qquad \textsc{Sup}\,\frac{\textsc{ArgCong}\,\dfrac{\mathsf{g} \approx \mathsf{f}}{\mathsf{g}\,x\,y\,z \approx \mathsf{f}\,x\,y\,z} \qquad h\,\mathsf{a} \not\approx \mathsf{b}}{\mathsf{f}\,x\,y\,\mathsf{a} \not\approx \mathsf{b}}$$

In both derivation trees, the higher-order variable h is effectively the target of a Sup inference. Such derivations essentially amount to superposition at variable positions (as shown on the left) or even superposition below variable positions (as shown on the right), both of which can be extremely prolific. In standard superposition, the explosion is averted by the condition on the Sup rule that $u \notin \mathcal{V}$. In the extensional purifying calculus, the variable condition tests that either u has a nonvariable head or u does not jell with the literal $t \approx t' \in D$, which prevents derivations such as the above. In the corresponding nonpurifying calculus, some such derivations may need to be performed when the term order exhibits nonmonotonicity for the terms of interest.

In the intensional calculi, the explosion can arise because the variable conditions are weaker. The following example shows that the intensional nonpurifying calculus would be incomplete if we used the variable condition of the extensional nonpurifying calculus.

Example 3.6. Consider a left-to-right LPO [34] instance with precedence $\mathsf{h} \succ \mathsf{g} \succ \mathsf{f} \succ \mathsf{b} \succ \mathsf{a}$, and consider the following unsatisfiable clause set:

$$\mathsf{h}\, x \approx \mathsf{f}\, x \qquad \mathsf{g}\,(x\, \mathsf{b})\, x \approx \mathsf{a} \qquad \mathsf{g}\,(\mathsf{f}\, \mathsf{b})\, \mathsf{h} \not\approx \mathsf{a}$$

The only possible inference is a SUP inference of the first into the second clause, allowing the intensional nonpurifying calculus to refute the clause set as follows:

$$\textsc{Sup}\,\frac{\textsc{Sup}\,\dfrac{\mathsf{h}\, x \approx \mathsf{f}\, x \quad \mathsf{g}\,(x\, \mathsf{b})\, x \approx \mathsf{a}}{\mathsf{g}\,(\mathsf{f}\, \mathsf{b})\, \mathsf{h} \approx \mathsf{a}} \qquad \mathsf{g}\,(\mathsf{f}\, \mathsf{b})\, \mathsf{h} \not\approx \mathsf{a}}{\textsc{ERes}\,\dfrac{\mathsf{a} \not\approx \mathsf{a}}{\bot}}$$

The variable condition of the extensional nonpurifying calculus, however, is not met for the first SUP inference because $x\, \mathsf{b}$ jells with $\mathsf{h}\, x \approx \mathsf{f}\, x$.

It is unclear whether the variable condition of the intensional purifying calculus could be strengthened, but our completeness proof suggests that it cannot.

The variable conditions in the extensional calculi are designed to prevent the argument congruence explosion shown above, but since they consider only the shape of the clauses, they might also block SUP inferences whose side premises do not originate from ARGCONG. This is why we need the POSEXT rule.

Example 3.7. In the following unsatisfiable clause set, the only possible inference from these clauses in the extensional nonpurifying calculus is POSEXT, showing its necessity:

$$\mathsf{g}\, x \approx \mathsf{f}\, x \qquad \mathsf{g} \not\approx \mathsf{f} \qquad x\,(\mathsf{diff}\langle \alpha, \beta\rangle\, x\, y) \not\approx y\,(\mathsf{diff}\langle \alpha, \beta\rangle\, x\, y) \vee x \approx y$$

The same argument applies for the purifying calculus with the difference that the third clause must be purified.

Due to nonmonotonicity, for refutational completeness we need either to purify the clauses or to allow some superposition at variable positions, as mandated by the respective variable conditions. Without either of these measures, at least the extensional calculi and presumably also the intensional calculi would be incomplete, as the next example demonstrates.

Example 3.8. Consider the following clause set:

$$\mathsf{k}\,(\mathsf{g}\, x) \approx \mathsf{k}\,(x\, \mathsf{b}) \quad \mathsf{k}\,(\mathsf{f}\,(\mathsf{h}\, \mathsf{a})\, \mathsf{b}) \not\approx \mathsf{k}\,(\mathsf{g}\, \mathsf{h}) \quad \mathsf{f}\,(\mathsf{h}\, \mathsf{a}) \approx \mathsf{h} \quad \mathsf{f}\,(\mathsf{h}\, \mathsf{a})\, x \approx \mathsf{h}\, x$$
$$x\,(\mathsf{diff}\langle \alpha, \beta\rangle\, x\, y) \not\approx y\,(\mathsf{diff}\langle \alpha, \beta\rangle\, x\, y) \vee x \approx y$$

Using a left-to-right LPO [34] instance with precedence $\mathsf{k} \succ \mathsf{h} \succ \mathsf{g} \succ \mathsf{f} \succ \mathsf{b} \succ \mathsf{a}$, this clause set is saturated w.r.t. the extensional purifying calculus when omitting purification. It also quickly saturates using the extensional nonpurifying calculus when omitting SUP inferences at variables. By contrast, the intensional calculi derive $\bot$, even without purification and without SUP inferences at variables, because of the less restrictive variable conditions.

This raises the question as to whether the intensional calculi actually need to purify or to perform SUP inferences at variables. We conjecture that omitting purification and SUP inferences at variables in the intensional calculi is complete when redundant clauses are kept but that it is incomplete in general.

We initially considered inference rules instead of axiom (EXT). However, we did not find a set of inference rules that is complete and leads to fewer inferences than (EXT). We considered the POSEXT rule described above in combination with the following rule:

$$\frac{C \vee s \not\approx t}{C \vee s\,(\mathsf{sk}\langle \bar{\alpha} \rangle \bar{x}_n) \not\approx t\,(\mathsf{sk}\langle \bar{\alpha} \rangle \bar{x}_n)}\ \textsc{NegExt}$$

where sk is a fresh Skolem symbol and $\bar{\alpha}$ and $\bar{x}_n$ are the type and term variables occurring free in the literal $s \not\approx t$. However, these two rules do not suffice for a refutationally complete calculus, as the following example demonstrates:

Example 3.9. Consider the clause set

$$\mathsf{f}\,x \approx \mathsf{a} \qquad \mathsf{g}\,x \approx \mathsf{a} \qquad \mathsf{h}\,\mathsf{f} \approx \mathsf{b} \qquad \mathsf{h}\,\mathsf{g} \not\approx \mathsf{b}$$

Assuming that all four equations are oriented from left to right, this set is saturated w.r.t. the extensional calculi if (EXT) is replaced by NEGEXT; yet it is unsatisfiable in an extensional logic.

Example 3.10. A significant advantage of our calculi over the use of standard superposition on applicatively encoded problems is the flexibility they offer in orienting equations. The following equations provide two definitions of addition on Peano numbers:

$$\mathsf{add}_\mathsf{L}\ \mathsf{Zero}\ y \approx y \qquad \mathsf{add}_\mathsf{R}\ x\ \mathsf{Zero} \approx x$$
$$\mathsf{add}_\mathsf{L}\ (\mathsf{Succ}\ x)\ y \approx \mathsf{add}_\mathsf{L}\ x\ (\mathsf{Succ}\ y) \qquad \mathsf{add}_\mathsf{R}\ x\ (\mathsf{Succ}\ y) \approx \mathsf{add}_\mathsf{R}\ (\mathsf{Succ}\ x)\ y$$

Let $\mathsf{add}_\mathsf{L}\ (\mathsf{Succ}^{100}\ \mathsf{Zero})\ \mathsf{n} \not\approx \mathsf{add}_\mathsf{R}\ \mathsf{n}\ (\mathsf{Succ}^{100}\ \mathsf{Zero})$ be the negated conjecture. With LPO, we can use a left-to-right comparison for add_L's arguments and a right-to-left comparison for add_R's arguments to orient all four equations from left to right. Then the negated conjecture can be simplified to $\mathsf{Succ}^{100}\ \mathsf{n} \not\approx \mathsf{Succ}^{100}\ \mathsf{n}$ by simplification (demodulation), and $\bot$ can be derived with a single inference. If we use the applicative encoding instead, there is no instance of LPO or KBO that can orient both recursive equations from left to right. For at least one of the two sides of the negated conjecture, simplification is replaced by 100 SUP inferences, which is much less efficient, especially in the presence of additional axioms.

3.3.3. Soundness

To show the inferences' soundness, we need the substitution lemma for our logic:

Lemma 3.11 (Substitution lemma)**.** *Let* $\mathcal{I} = (\mathcal{U}, \mathcal{J}_{\mathrm{ty}}, \mathcal{J}, \mathcal{E})$ *be a* λ*-free higher-order interpretation. Then*

$$[\![\tau\rho]\!]^{\xi}_{\mathcal{I}_{\mathrm{ty}}} = [\![\tau]\!]^{\xi'}_{\mathcal{I}_{\mathrm{ty}}} \text{ and } [\![t\rho]\!]^{\xi}_{\mathcal{I}} = [\![t]\!]^{\xi'}_{\mathcal{I}}$$

for all terms t*, all types* τ*, and all substitutions* ρ*, where* $\xi'(\alpha) = [\![\alpha\rho]\!]^{\xi}_{\mathcal{I}_{\mathrm{ty}}}$ *for all type variables* α *and* $\xi'(x) = [\![x\rho]\!]^{\xi}_{\mathcal{I}}$ *for all term variables* x*.*

Proof. First, we prove that $[\![\tau\rho]\!]^{\xi}_{\mathcal{I}_{\mathrm{ty}}} = [\![\tau]\!]^{\xi'}_{\mathcal{I}_{\mathrm{ty}}}$ by induction on the structure of τ. If $\tau = \alpha$ is a type variable,

$$[\![\alpha\rho]\!]^{\xi}_{\mathcal{I}_{\mathrm{ty}}} = \xi'(\alpha) = [\![\alpha]\!]^{\xi'}_{\mathcal{I}_{\mathrm{ty}}}$$

If $\tau = \kappa(\bar{\upsilon})$ for some type constructor κ and types $\bar{\upsilon}$,

$$[\![\kappa(\bar{\upsilon})\rho]\!]^{\xi}_{\mathcal{I}_{\mathrm{ty}}} = \mathcal{J}_{\mathrm{ty}}(\kappa)([\![\bar{\upsilon}\rho]\!]^{\xi}_{\mathcal{I}_{\mathrm{ty}}}) \overset{\mathrm{IH}}{=} \mathcal{J}_{\mathrm{ty}}(\kappa)([\![\bar{\upsilon}]\!]^{\xi'}_{\mathcal{I}_{\mathrm{ty}}}) = [\![\kappa(\bar{\upsilon})]\!]^{\xi'}_{\mathcal{I}_{\mathrm{ty}}}$$

Next, we prove $[\![t\rho]\!]^{\xi}_{\mathcal{I}} = [\![t]\!]^{\xi'}_{\mathcal{I}}$ by structural induction on t. If $t = y$, then by the definition of the denotation of a variable

$$[\![y\rho]\!]^{\xi}_{\mathcal{I}} = \xi'(y) = [\![y]\!]^{\xi'}_{\mathcal{I}}$$

If $t = \mathsf{f}\langle\bar{\tau}\rangle$, then by the definition of the term denotation

$$[\![\mathsf{f}\langle\bar{\tau}\rangle\rho]\!]^{\xi}_{\mathcal{I}} = \mathcal{J}(\mathsf{f}, [\![\bar{\tau}\rho]\!]^{\xi}_{\mathcal{I}_{\mathrm{ty}}}) = \mathcal{J}(\mathsf{f}, [\![\bar{\tau}]\!]^{\xi'}_{\mathcal{I}_{\mathrm{ty}}}) = [\![\mathsf{f}\langle\bar{\tau}\rangle]\!]^{\xi'}_{\mathcal{I}}$$

If $t = u\,v$, then by the definition of the term denotation

$$[\![(u\,v)\rho]\!]^{\xi}_{\mathcal{I}} = \mathcal{E}_{U_1,U_2}([\![u\rho]\!]^{\xi}_{\mathcal{I}})([\![v\rho]\!]^{\xi}_{\mathcal{I}}) \overset{\mathrm{IH}}{=} \mathcal{E}_{U_1,U_2}([\![u]\!]^{\xi'}_{\mathcal{I}})([\![v]\!]^{\xi'}_{\mathcal{I}}) = [\![u\,v]\!]^{\xi'}_{\mathcal{I}}$$

where u is of type $\tau \to \upsilon$, $U_1 = [\![\tau\rho]\!]^{\xi}_{\mathcal{I}_{\mathrm{ty}}} = [\![\tau]\!]^{\xi'}_{\mathcal{I}_{\mathrm{ty}}}$, and $U_2 = [\![\upsilon\rho]\!]^{\xi}_{\mathcal{I}_{\mathrm{ty}}} = [\![\upsilon]\!]^{\xi'}_{\mathcal{I}_{\mathrm{ty}}}$. □

Lemma 3.12. *If* $\mathcal{I} \models C$ *for some interpretation* $\mathcal{I}$ *and some clause* C*, then* $\mathcal{I} \models C\rho$ *for all substitutions* ρ*.*

Proof. We need to show that $C\rho$ is true in $\mathcal{I}$ for all valuations ξ. Given a valuation ξ, define ξ' as in Lemma 3.11. Then, by Lemma 3.11, a literal in $C\rho$ is true in $\mathcal{I}$ for ξ if and only if the corresponding literal in C is true in $\mathcal{I}$ for ξ'. There must be at least one such literal because $\mathcal{I} \models C$ and hence C is in particular true in $\mathcal{I}$ for ξ'. Therefore, $C\rho$ is true in $\mathcal{I}$ for ξ. □

Theorem 3.13 (Soundness of the intensional calculi)**.** *The inference rules* Sup*,* ERes*,* EFact*, and* ArgCong *are sound* (*even without the variable condition and the side conditions on order and eligibility*)*.*

Proof. We fix an inference and an interpretation $\mathcal{I}$ that is a model of the premises. We need to show that it is also a model of the conclusion.

From the definition of the denotation of a term, it is obvious that congruence holds at all subterms in our logic. By Lemma 3.12, $\mathcal{I}$ is a model of the σ-instances of the premises as well, where σ is the substitution used for the inference. Fix a valuation ξ. By making case distinctions on the truth in $\mathcal{I}$ under ξ of the literals of the σ-instances of the premises, using the conditions that σ is a unifier, and applying congruence, it follows that the conclusion is also true in $\mathcal{I}$ under ξ. □

Theorem 3.14 (Soundness of the extensional calculi)**.** *The inference rules* Sup, ERes, EFact, ArgCong, *and* PosExt *are sound w.r.t. extensional interpretations* (*even without the variable condition and the side conditions on order and eligibility*).

Proof. We only need to prove PosExt sound. For the other rules, we can proceed as in Theorem 3.13. By induction on the length of $\bar{x}$, it suffices to prove PosExt sound for one variable x instead of a tuple $\bar{x}$. We fix an inference and an extensional interpretation $\mathcal{I}$ that is a model of the premise $C' \vee s\,x \approx s'\,x$. We need to show that it is also a model of the conclusion $C' \vee s \approx s'$.

Let ξ be a valuation. If C' is true in $\mathcal{I}$ under ξ, the conclusion is clearly true as well. Otherwise C' is false in $\mathcal{I}$ under ξ, and also under $\xi[x \mapsto a]$ for all a because x does not occur in C'. Since the premise is true in $\mathcal{I}$, $s\,x = s'\,x$ must be true in $\mathcal{I}$ under $\xi[x \mapsto a]$ for all a. Hence, for appropriate universes U_1, U_2, we have

$$\mathcal{E}_{U_1,U_2}([\![s]\!]_{\mathcal{I}}^{\xi[x\mapsto a]})(a) = [\![s\,x]\!]_{\mathcal{I}}^{\xi[x\mapsto a]} = [\![s'\,x]\!]_{\mathcal{I}}^{\xi[x\mapsto a]} = \mathcal{E}_{U_1,U_2}([\![s']\!]_{\mathcal{I}}^{\xi[x\mapsto a]})(a)$$

Since s and s' do not contain x, $[\![s]\!]_{\mathcal{I}}^{\xi[x\mapsto a]}$ and $[\![s']\!]_{\mathcal{I}}^{\xi[x\mapsto a]}$ do not depend on a. Thus, $\mathcal{E}_{U_1,U_2}([\![s]\!]_{\mathcal{I}}^{\xi}) = \mathcal{E}_{U_1,U_2}([\![s']\!]_{\mathcal{I}}^{\xi})$. Since $\mathcal{I}$ is extensional, $\mathcal{E}_{U_1,U_2}$ is injective and it follows that $[\![s]\!]_{\mathcal{I}}^{\xi} = [\![s']\!]_{\mathcal{I}}^{\xi}$. It follows that $s \approx s'$ is true in $\mathcal{I}$ under ξ, and so is the entire conclusion of the inference. □

A problem expressed in higher-order logic must be transformed into clausal normal form before the calculi can be applied. This process works as in the first-order case, except for skolemization. The issue is that skolemization, when performed naively, is unsound for higher-order logic with a Henkin semantics [107, Section 6], because it introduces new functions that can be used to instantiate variables.

The core of this chapter is not affected by this because the problems are given in clausal form. For the implementation, we claim soundness only w.r.t. models that satisfy the axiom of choice, which is the semantics mandated by the TPTP THF format [131]. By contrast, refutational completeness holds w.r.t. arbitrary models as defined above. Alternatively, skolemization can be made sound by introducing mandatory arguments as described by Miller [107, Section 6].

This issue also affects axiom (Ext) because it contains the Skolem symbol diff. As a consequence, (Ext) does not hold in all extensional interpretations. The extensional calculi are thus only sound w.r.t. interpretations in which (Ext) holds. However, we can prove that (Ext) is compatible with our logic:

Theorem 3.15. *Axiom* (Ext) *is satisfiable.*

Proof. For a given signature, let $(\mathcal{U}, \mathcal{J}_{\mathsf{ty}}, \mathcal{J}, \mathcal{E})$ be an Herbrand interpretation. That is, we define $\mathcal{U}$ to contain the set $\mathcal{U}_\tau$ of all terms of type τ for each ground type τ, we define $\mathcal{J}_{\mathsf{ty}}$ by $\mathcal{J}_{\mathsf{ty}}(\kappa)(\bar{\tau}) = \kappa(\bar{\tau})$, we define $\mathcal{J}$ by $\mathcal{J}(\mathsf{f}, \mathcal{U}_{\bar{\tau}}) = \mathsf{f}\langle\bar{\tau}\rangle$, and we define $\mathcal{E}$ by $\mathcal{E}_{U_\tau,U_\upsilon}(\mathsf{f}(\mathsf{a}) = \mathsf{f}\,\mathsf{a}$. Then $\mathcal{E}_{U_\tau,U_\upsilon}$ is clearly injective and hence $\mathcal{I}$ is extensional. To show that $\mathcal{I} \models$ (Ext), we need to show that (Ext) is true under all valuations. Let ξ be a valuation. If $x \approx y$ is true under ξ, (Ext) is also true. Otherwise $x \approx y$ is false under ξ, and hence $\xi(x) \neq \xi(y)$. Then we have

$$\begin{aligned}[\![x\,(\mathsf{diff}\langle\alpha,\beta\rangle\,x\,y)]\!]_{\mathcal{I}}^{\xi} &= (\xi(x))\,(\mathsf{diff}\langle\alpha,\beta\rangle\,(\xi(x))\,(\xi(y))) \\ &\neq (\xi(y))\,(\mathsf{diff}\langle\alpha,\beta\rangle\,(\xi(x))\,(\xi(y))) = [\![y\,(\mathsf{diff}\langle\alpha,\beta\rangle\,x\,y)]\!]_{\mathcal{I}}^{\xi}\end{aligned}$$

Therefore, $x\,(\mathsf{diff}\langle\alpha,\beta\rangle\,x\,y) \not\approx y\,(\mathsf{diff}\langle\alpha,\beta\rangle\,x\,y)$ is true in $\mathcal{I}$ under ξ and so is (EXT). □

3.3.4. The Redundancy Criterion

For our calculi, a redundant clause cannot simply be defined as a clause whose ground instances are entailed by smaller ($\prec$) ground instances of existing clauses, because this would make all ARGCONG inferences redundant. Our solution is to base the redundancy criterion on a weaker ground logic—ground monomorphic first-order logic—in which argument congruence does not hold. This logic also plays a central role in our refutational completeness proof.

We employ an encoding $\mathcal{F}$ to translate ground λ-free higher-order terms into ground first-order terms. It indexes each symbol occurrence with its type arguments and its term argument count. Thus, $\mathcal{F}(\mathsf{f}) = \mathsf{f}_0$, $\mathcal{F}(\mathsf{f}\,\mathsf{a}) = \mathsf{f}_1(\mathsf{a}_0)$, and $\mathcal{F}(\mathsf{g}\langle\kappa\rangle) = \mathsf{g}_0^\kappa$. This is enough to disable argument congruence; for example, $\{\mathsf{f} \approx \mathsf{h}, \mathsf{f}\,\mathsf{a} \not\approx \mathsf{h}\,\mathsf{a}\}$ is unsatisfiable, whereas its encoding $\{\mathsf{f}_0 \approx \mathsf{h}_0, \mathsf{f}_1(\mathsf{a}_0) \not\approx \mathsf{h}_1(\mathsf{a}_0)\}$ is satisfiable. For clauses built from fully applied ground terms, the two logics are isomorphic, as we would expect from a graceful generalization.

Given a λ-free higher-order signature $(\Sigma_{\mathsf{ty}}, \Sigma)$, we define a first-order signature $(\Sigma_{\mathsf{ty}}, \Sigma_{\mathsf{GF}})$ as follows. The type constructors Σ_{ty} are the same in both signatures, but $\to$ is uninterpreted in first-order logic. For each symbol $\mathsf{f} : \Pi\bar{\alpha}_m.\, \tau_1 \to \cdots \to \tau_n \to \tau$ in Σ, where τ is not functional, we introduce symbols $\mathsf{f}_l^{\bar{\upsilon}_m} \in \Sigma_{\mathsf{GF}}$ with argument types $\bar{\tau}_l\sigma$ and return type $(\tau_{l+1} \to \cdots \to \tau_n \to \tau)\sigma$, where $\sigma = \{\bar{\alpha}_m \mapsto \bar{\upsilon}_m\}$, for each tuple of ground types $\bar{\upsilon}_m$ and each $l \in \{0, \ldots, n\}$.

For example, let $\Sigma = \{\mathsf{a} : \kappa, \mathsf{g} : \kappa \to \kappa \to \kappa\}$. The corresponding first-order signature is $\Sigma_{\mathsf{GF}} = \{\mathsf{a}_0 : \kappa, \mathsf{g}_0 : \kappa \to \kappa \to \kappa, \mathsf{g}_1 : \kappa \Rightarrow \kappa \to \kappa, \mathsf{g}_2 : \kappa^2 \Rightarrow \kappa\}$ where $\mathsf{f} : \bar{\tau} \Rightarrow \upsilon$ denotes a first-order function symbol f with argument types $\bar{\tau}$ and return type υ, and $\to$ is an uninterpreted binary type constructor. The term $\mathcal{F}(\mathsf{g}\,\mathsf{a}\,\mathsf{a}) = \mathsf{g}_2(\mathsf{a}_0, \mathsf{a}_0)$ has type κ, and $\mathcal{F}(\mathsf{g}\,\mathsf{a}) = \mathsf{g}_1(\mathsf{a}_0)$ has type $\kappa \to \kappa$.

Thus, we consider three levels of logics: the λ-free higher-order level H over a given signature $(\Sigma_{\mathsf{ty}}, \Sigma)$, the ground λ-free higher-order level GH, corresponding to H's ground fragment, and the ground monomorphic first-order level GF over the signature $(\Sigma_{\mathsf{ty}}, \Sigma_{\mathsf{GF}})$ defined above. We use $\mathcal{T}_{\mathrm{H}}$, $\mathcal{T}_{\mathrm{GH}}$, and $\mathcal{T}_{\mathrm{GF}}$ to denote the respective sets of terms, $\mathcal{T}\!y_{\mathrm{H}}$, $\mathcal{T}\!y_{\mathrm{GH}}$, and $\mathcal{T}\!y_{\mathrm{GF}}$ to denote the respective sets of types, and $\mathcal{C}_{\mathrm{H}}$, $\mathcal{C}_{\mathrm{GH}}$, and $\mathcal{C}_{\mathrm{GF}}$ to denote the respective sets of clauses. In the purifying calculi, we exceptionally let $\mathcal{C}_{\mathrm{H}}$ denote the set of purified clauses. Each of the three levels has an entailment relation $\models$. A clause set N_1 entails a clause set N_2, denoted $N_1 \models N_2$, if any model of N_1 is also a model of N_2. On H and GH, we use λ-free higher-order models for the intensional calculi and extensional λ-free higher-order models for the extensional calculi; on GF, we use first-order models. This machinery may seem excessive, but it is essential to define redundancy of clauses and inferences properly, and it will play an important role in the refutational completeness proof (Section 3.4).

The three levels are connected by two functions, $\mathcal{G}$ and $\mathcal{F}$:

Definition 3.16 (Grounding function $\mathcal{G}$ on terms and clauses)**.** The grounding function $\mathcal{G}$ maps terms $t \in \mathcal{T}_{\mathrm{H}}$ to the set of their ground instances—i.e., the set of all $t\theta \in \mathcal{T}_{\mathrm{GH}}$ where θ is a substitution. It also maps clauses $C \in \mathcal{C}_{\mathrm{H}}$ to the set of their ground instances—i.e., the set of all $C\theta \in \mathcal{C}_{\mathrm{GH}}$ where θ is a substitution.

Definition 3.17 (Encoding $\mathcal{F}$ on terms and clauses)**.** The encoding $\mathcal{F} : \mathcal{T}_{\mathrm{GH}} \to \mathcal{T}_{\mathrm{GF}}$ is recursively defined as $\mathcal{F}(\mathsf{f}\langle \bar{v}_m \rangle\, \bar{u}_l) = \mathsf{f}_l^{\bar{v}_m}(\mathcal{F}(\bar{u}_l))$. The encoding $\mathcal{F}$ is extended to map from $\mathcal{C}_{\mathrm{GH}}$ to $\mathcal{C}_{\mathrm{GF}}$ by mapping each literal and each side of a literal individually.

The encoding $\mathcal{F}$ is bijective with inverse $\mathcal{F}^{-1}$. Using $\mathcal{F}^{-1}$, the clause order $\succ$ on $\mathcal{T}_{\mathrm{GH}}$ can be transferred to $\mathcal{T}_{\mathrm{GF}}$ by defining $t \succ s$ as equivalent to $\mathcal{F}^{-1}(t) \succ \mathcal{F}^{-1}(s)$. The property that $\succ$ on clauses is the multiset extension of $\succ$ on literals, which in turn is the multiset extension of $\succ$ on terms, is maintained because $\mathcal{F}^{-1}$ maps the multiset representations elementwise.

3

Schematically, the three levels are connected as follows:

$$\underset{\text{higher-order}}{\mathrm{H}} \xrightarrow{\ \mathcal{G}\ } \underset{\text{ground higher-order}}{\mathrm{GH}} \xrightarrow{\ \mathcal{F}\ } \underset{\text{ground first-order}}{\mathrm{GF}}$$

Crucially, green subterms in $\mathcal{T}_{\mathrm{GH}}$ correspond to subterms in $\mathcal{T}_{\mathrm{GF}}$ (Lemma 3.18), whereas nongreen subterms in $\mathcal{T}_{\mathrm{GH}}$ are not subterms at all in $\mathcal{T}_{\mathrm{GF}}$. For example, a is a green subterm of $\mathsf{f}\,\mathsf{a}$, and correspondingly $\mathcal{F}(\mathsf{a}) = \mathsf{a}_0$ is a subterm of $\mathcal{F}(\mathsf{f}\,\mathsf{a}) = \mathsf{f}_1(\mathsf{a}_0)$. On the other hand, f is not a green subterm of $\mathsf{f}\,\mathsf{a}$, and correspondingly $\mathcal{F}(\mathsf{f}) = \mathsf{f}_0$ is not a subterm of $\mathcal{F}(\mathsf{f}\,\mathsf{a}) = \mathsf{f}_1(\mathsf{a}_0)$.

To state the correspondence between green subterms in $\mathcal{T}_{\mathrm{GH}}$ and subterms in $\mathcal{T}_{\mathrm{GF}}$ explicitly, we define positions of green subterms as follows. A term $s \in \mathcal{T}_{\mathrm{H}}$ is a green subterm at position ϵ of s. If a term $s \in \mathcal{T}_{\mathrm{H}}$ is a green subterm at position p of u_i for some $1 \le i \le n$, then s is a green subterm at position $i.p$ of $\mathsf{f}\langle \bar{\tau} \rangle\, \bar{u}_n$ and of $x\, \bar{u}_n$. For $\mathcal{T}_{\mathrm{GF}}$, positions are defined as usual in first-order logic.

Lemma 3.18. *Let $s, t \in \mathcal{T}_{\mathrm{GH}}$. We have $\mathcal{F}(t\langle s \rangle_p) = \mathcal{F}(t)[\mathcal{F}(s)]_p$. In other words, s is a green subterm of t at position p if and only if $\mathcal{F}(s)$ is a subterm of $\mathcal{F}(t)$ at position p.*

Proof. By induction on p. If $p = \varepsilon$, then $s = t[s]_p$. Hence $\mathcal{F}(t[s]_p) = \mathcal{F}(s) = \mathcal{F}(t)\langle \mathcal{F}(s) \rangle_p$. If $p = i.p'$, then $t[s]_p = \mathsf{f}\langle \bar{\tau} \rangle\, \bar{u}_n$ with $u_i = u_i[s]_{p'}$. Applying $\mathcal{F}$, we obtain by the induction hypothesis that $\mathcal{F}(u_i) = \mathcal{F}(u_i)\langle \mathcal{F}(s) \rangle_{p'}$. Therefore, $\mathcal{F}(t[s]_p) = \mathsf{f}_n^{\bar{\tau}}(\mathcal{F}(u_1), \ldots, \mathcal{F}(u_{i-1}), \mathcal{F}(u_i)\langle \mathcal{F}(s) \rangle_{p'}, \mathcal{F}(u_{i+1}), \ldots, \mathcal{F}(u_n))$. It follows that $\mathcal{F}(t[s]_p) = \mathcal{F}(t)\langle \mathcal{F}(s) \rangle_p$. □

Corollary 3.19. *Given $s, t \in \mathcal{T}_{\mathrm{GF}}$, we have $\mathcal{F}^{-1}(t[s]_p) = \mathcal{F}^{-1}(t)\langle \mathcal{F}^{-1}(s) \rangle_p$.*

Lemma 3.20. *Well-foundedness, totality, compatibility with contexts, and the subterm property hold for $\succ$ on $\mathcal{T}_{\mathrm{GF}}$.*

Proof. COMPATIBILITY WITH CONTEXTS: We must show that $s \succ s'$ implies $t[s]_p \succ t[s']_p$ for terms $t, s, s' \in \mathcal{T}_{\mathrm{GF}}$. Assuming $s \succ s'$, we have $\mathcal{F}^{-1}(s) \succ \mathcal{F}^{-1}(s')$. By compatibility with green contexts on $\mathcal{T}_{\mathrm{GH}}$, we have $\mathcal{F}^{-1}(t)\langle \mathcal{F}^{-1}(s) \rangle_p \succ \mathcal{F}^{-1}(t)\langle \mathcal{F}^{-1}(s') \rangle_p$. By Corollary 3.19, we have $t[s]_p \succ t[s']_p$.

WELL-FOUNDEDNESS: Assume that there exists an infinite descending chain $t_1 \succ t_2 \succ \cdots$ in $\mathcal{T}_{\mathrm{GF}}$. By applying $\mathcal{F}^{-1}$, we then obtain the chain $\mathcal{F}^{-1}(t_1) \succ \mathcal{F}^{-1}(t_2) \succ \cdots$ in $\mathcal{T}_{\mathrm{GH}}$, contradicting well-foundedness on $\mathcal{T}_{\mathrm{GH}}$.

TOTALITY: Let $s, t \in \mathcal{T}_{\mathrm{GF}}$. Then $\mathcal{F}^{-1}(t)$ and $\mathcal{F}^{-1}(s)$ must be comparable by totality in $\mathcal{T}_{\mathrm{GH}}$. Hence, t and s are comparable.

SUBTERM PROPERTY: By Corollary 3.19 and the subterm property on $\mathcal{T}_{\mathrm{GH}}$, we have $\mathcal{F}^{-1}(t[s]_p) = \mathcal{F}^{-1}(t)\langle\mathcal{F}^{-1}(s)\rangle_p \succ \mathcal{F}^{-1}(s)$. Hence, $t[s]_p \succ s$. □

In standard superposition, redundancy relies on the entailment relation $\models$ on ground clauses. We will define redundancy on GH and H in the same way, but using GF's entailment relation. This notion of redundancy gracefully generalizes the first-order notion, without making all ARGCONG inferences redundant.

The standard redundancy criterion for standard superposition cannot justify subsumption deletion. Following Waldmann et al. [140], we incorporate subsumption into the redundancy criterion. A clause C *subsumes* D if there exists a substitution σ such that $C\sigma \subseteq D$. A clause C *strictly subsumes* D if C subsumes D but D does not subsume C. Let $\sqsupset$ stand for "is strictly subsumed by". Using the applicative encoding, it is easy to show that $\sqsupset$ is well founded because strict subsumption is well founded in first-order logic.

We define the sets of redundant clauses w.r.t. a given clause set as follows:

- Given $C \in \mathcal{C}_{\mathrm{GF}}$ and $N \subseteq \mathcal{C}_{\mathrm{GF}}$, let $C \in \mathit{GFRed}_{\mathrm{C}}(N)$ if $\{D \in N \mid D \prec C\} \models C$.
- Given $C \in \mathcal{C}_{\mathrm{GH}}$ and $N \subseteq \mathcal{C}_{\mathrm{GH}}$, let $C \in \mathit{GHRed}_{\mathrm{C}}(N)$ if $\mathcal{F}(C) \in \mathit{GFRed}_{\mathrm{C}}(\mathcal{F}(N))$.
- Given $C \in \mathcal{C}_{\mathrm{H}}$ and $N \subseteq \mathcal{C}_{\mathrm{H}}$, let $C \in \mathit{HRed}_{\mathrm{C}}(N)$ if for every $D \in \mathcal{G}(C)$, we have $D \in \mathit{GHRed}_{\mathrm{C}}(\mathcal{G}(N))$ or there exists $C' \in N$ such that $C \sqsupset C'$ and $D \in \mathcal{G}(C')$.

For example, $x\,(\mathsf{h}\,\mathsf{a}) \approx x\,(\mathsf{g}\,\mathsf{a})$ is redundant w.r.t. $\{\mathsf{h}\,\mathsf{a} \approx \mathsf{g}\,\mathsf{a}\}$, but not redundant w.r.t. $\{\mathsf{h} \approx \mathsf{g}\}$ because $\mathcal{F}$ encodes the unapplied occurrences of h and g as h_0 and g_0 and the applied occurrences as h_1 and g_1.

Along with the three levels of logics, we consider three inference systems: *HInf*, *GHInf* and *GFInf*. *HInf* is one of the four variants of the inference system described in Section 3.3.1. For uniformity, we regard axiom (EXT) as a premise-free inference rule EXT whose conclusion is (EXT). In the purifying calculi, the conclusion of EXT must be purified. *GHInf* consists of all SUP, ERES, and EFACT inferences from *HInf* whose premises and conclusion are ground, a premise-free rule GEXT whose infinitely many conclusions are the ground instances of (EXT), and the following ground variant of ARGCONG:

$$\frac{C' \vee s \approx s'}{C' \vee s\,\bar{u}_n \approx s'\,\bar{u}_n}\ \textsc{GArgCong}$$

where $s \approx s'$ is strictly eligible in the premise and $\bar{u}_n$ is a nonempty tuple of ground terms. *GFInf* contains all SUP, ERES, and EFACT inferences from *GHInf* translated by $\mathcal{F}$. It coincides exactly with standard first-order superposition. Given a SUP, ERES, or EFACT inference $\iota \in \mathit{GHInf}$, let $\mathcal{F}(\iota)$ denote the corresponding inference in *GFInf*.

Each of the three inference systems is parameterized by a selection function. For *HInf*, we globally fix a selection function *HSel*. For *GHInf* and *GFInf*, we need to consider different selection functions *GHSel* and *GFSel*. We write $\mathit{GHInf}^{\mathit{GHSel}}$ for *GHInf* and $\mathit{GFInf}^{\mathit{GFSel}}$ for *GFInf* to make the dependency on the respective selection functions explicit. Let $\mathcal{G}(\mathit{HSel})$ denote the set of all selection functions on $\mathcal{C}_{\mathrm{GH}}$ such that for each clause in $C \in \mathcal{C}_{\mathrm{GH}}$, there exists a clause $D \in \mathcal{C}_{\mathrm{H}}$ with $C \in \mathcal{G}(D)$ and corresponding selected literals. For each selection function *GHSel* on $\mathcal{C}_{\mathrm{GH}}$, via the

bijection $\mathcal{F}$, we obtain a corresponding selection function on $\mathcal{C}_{\mathrm{GF}}$, which we denote by $\mathcal{F}(GHSel)$.

Notation 3.21. Given an inference ι, we write $prems(\iota)$ for the tuple of premises, $mprem(\iota)$ for the main (i.e., rightmost) premise, $preconcl(\iota)$ for the conclusion before purification, and $concl(\iota)$ for the conclusion after purification. For the nonpurifying calculi, $preconcl(\iota) = concl(\iota)$ simply denotes the conclusion.

Definition 3.22 (Encoding $\mathcal{F}$ on inferences)**.** Given a SUP, ERES, or EFACT inference $\iota \in GHInf$, let $\mathcal{F}(\iota) \in GFInf$ denote the inference defined by $prems(\mathcal{F}(\iota)) = \mathcal{F}(prems(\iota))$ and $concl(\mathcal{F}(\iota)) = \mathcal{F}(concl(\iota))$.

Definition 3.23 (Grounding function $\mathcal{G}$ on inferences)**.** Given a selection function $GHSel \in \mathcal{G}(HSel)$, and a non-POSEXT inference $\iota \in HInf$, we define the set $\mathcal{G}^{GHSel}(\iota)$ of ground instances of ι to be all inferences $\iota' \in GHInf^{GHSel}$ such that $prems(\iota') = prems(\iota)\theta$ and $concl(\iota') = preconcl(\iota)\theta$ for some grounding substitution θ. For POSEXT inferences ι, which cannot be grounded, we let $\mathcal{G}^{GHSel}(\iota) = undef$.

This will map SUP to SUP, EFACT to EFACT, ERES to ERES, EXT to GEXT, and ARGCONG to GARGCONG inferences, but it is also possible that $\mathcal{G}^{GHSel}(\iota)$ is the empty set for some inferences ι.

We define the sets of redundant inferences w.r.t. a given clause set as follows:

- Given $\iota \in GFInf^{GFSel}$ and $N \subseteq \mathcal{C}_{\mathrm{GF}}$, let $\iota \in GFRed_{\mathrm{I}}^{GFSel}(N)$ if we have $prems(\iota) \cap GFRed_{\mathrm{C}}(N) \neq \varnothing$ or $\{D \in N \mid D \prec mprem(\iota)\} \models concl(\iota)$.
- Given $\iota \in GHInf^{GHSel}$ and $N \subseteq \mathcal{C}_{\mathrm{GH}}$, let $\iota \in GHRed_{\mathrm{I}}^{GHSel}(N)$ if
  - ι is not a GARGCONG or GEXT inference and $\mathcal{F}(\iota) \in GFRed_{\mathrm{I}}^{\mathcal{F}(GHSel)}(\mathcal{F}(N))$; or
  - the calculus is nonpurifying and ι is a GARGCONG or GEXT inference and $concl(\iota) \in N \cup GHRed_{\mathrm{C}}(N)$; or
  - the calculus is purifying and ι is a GARGCONG or GEXT inference and $\mathcal{F}(N) \models \mathcal{F}(concl(\iota))$.
- Given $\iota \in HInf$ and $N \subseteq \mathcal{C}_{\mathrm{H}}$, let $\iota \in HRed_{\mathrm{I}}(N)$ if
  - ι is not a POSEXT inference and $\mathcal{G}^{GHSel}(\iota) \subseteq GHRed_{\mathrm{I}}(\mathcal{G}(N))$ for all $GHSel \in \mathcal{G}(HSel)$; or
  - ι is a POSEXT inference and $\mathcal{G}(concl(\iota)) \subseteq \mathcal{G}(N) \cup GHRed_{\mathrm{C}}(\mathcal{G}(N))$.

Occasionally, we omit the selection function in the notation when it is irrelevant.

A clause set N is *saturated* w.r.t. an inference system and a redundancy criterion $(Red_{\mathrm{I}}, Red_{\mathrm{C}})$ if every inference from clauses in N is in $Red_{\mathrm{I}}(N)$. The most straightforward way to saturate a given clause set is to repeatedly add all conclusions of inferences from the given clause set. By the above definition, every inference is redundant w.r.t. its own conclusion, and hence the limit of this process is a saturated clause set.

3.3.5. Simplification Rules

The redundancy criterion $(HRed_{\mathrm{I}}, HRed_{\mathrm{C}})$ is strong enough to support most of the simplification rules implemented in Schulz's first-order prover E [117, Sections 2.3.1 and 2.3.2], some only with minor adaptions. Deletion of duplicated literals,

deletion of resolved literals, syntactic tautology deletion, negative simplify-reflect, and clause subsumption adhere to our redundancy criterion.

Positive simplify-reflect and equality subsumption are supported by our criterion if they are applied in green contexts. Semantic tautology deletion can be applied as well, but we must use the entailment relation of the GF level—i.e., only rewriting in green contexts can be used to establish the entailment. Similarly, rewriting of positive and negative literals (demodulation) can only be applied in green contexts. Moreover, for positive literals, the rewriting clause must be smaller than the rewritten clause—a condition that is also necessary with the standard first-order redundancy criterion but not always fulfilled by Schulz's rule. As for destructive equality resolution, even in first-order logic the rule cannot be justified with the standard redundancy criterion, and it is unclear whether it preserves refutational completeness.

As a representative example, we show how demodulation into green contexts can be justified. The other simplification rules can be justified similarly.

Lemma 3.24. *Demodulation into green contexts is a simplification:*

$$\frac{t \approx t' \quad \overbrace{s\langle t\sigma\rangle \mathrel{\dot\approx} s' \vee C'}^{C}}{t \approx t' \quad s\langle t'\sigma\rangle \mathrel{\dot\approx} s' \vee C'}\ \textsc{Demod}$$

where $t\sigma \succ t'\sigma$ and $C \succ (t \approx t')\sigma$. It adheres to our redundancy criterion—i.e., the deleted premise C is redundant w.r.t. the conclusions.

Proof. Let N be the set consisting of the two conclusions. We must show that $C \in HRed_{\mathrm{C}}(N)$. Let $C\theta$ be a ground instance of C. By definition of $HRed_{\mathrm{C}}$, it suffices to show that $C\theta \in GHRed_{\mathrm{C}}(\mathcal{G}(N))$. By definition of $GHRed_{\mathrm{C}}$, we must thus show that $\mathcal{F}(C\theta) \in GFRed_{\mathrm{C}}(\mathcal{F}(\mathcal{G}(N)))$. By definition of $GFRed_{\mathrm{C}}$, this is equivalent to proving that the clauses in $\mathcal{F}(\mathcal{G}(N))$ that are smaller than $\mathcal{F}(C\theta)$ entail $\mathcal{F}(C\theta)$.

By compatibility with green contexts and stability under substitutions of $\succ$, the condition $t\sigma \succ t'\sigma$ implies that $D = \mathcal{F}((s\langle t'\sigma\rangle \mathrel{\dot\approx} s' \vee C')\theta)$ is a clause in $\mathcal{F}(\mathcal{G}(N))$ that is smaller than $\mathcal{F}(C\theta)$. By stability under substitutions, $C \succ (t \approx t')\sigma$ implies that $E = \mathcal{F}((t \approx t')\sigma\theta)$ is another clause in $\mathcal{F}(\mathcal{G}(N))$ that is smaller than $\mathcal{F}(C\theta)$. By Lemma 3.18, green subterms on the GH level correspond to subterms on the GF level. Thus, $\{D, E\} \models \mathcal{F}(C\theta)$ by congruence. □

3.4. Refutational Completeness

Besides soundness, the most important property of the four calculi introduced in Section 3.3.1 is refutational completeness. We will prove the static and dynamic refutational completeness of *HInf* w.r.t. $(HRed_{\mathrm{I}}, HRed_{\mathrm{C}})$, which is defined as follows:

Definition 3.25 (Static refutational completeness)**.** Let *Inf* be an inference system and let $(Red_{\mathrm{I}}, Red_{\mathrm{C}})$ be a redundancy criterion. The inference system *Inf* is called *statically refutationally complete* w.r.t. $(Red_{\mathrm{I}}, Red_{\mathrm{C}})$ if we have $N \models \bot$ if and only if $\bot \in N$ for every clause set N that is saturated w.r.t. *Inf* and Red_{I}.

Definition 3.26 (Dynamic refutational completeness)**.** Let *Inf* be an inference system and let (Red_I,Red_C) be a redundancy criterion. Let $(N_i)_i$ be a finite or infinite sequence over sets of clauses. Such a sequence is called a *derivation* if $N_i \setminus N_{i+1} \subseteq Red_\mathrm{C}(N_{i+1})$ for all i. It is called *fair* if all *Inf*-inferences from clauses in the limit inferior $\bigcup_i \bigcap_{j \geq i} N_j$ are contained in $\bigcup_i Red_\mathrm{I}(N_i)$. The inference system *Inf* is called *dynamically refutationally complete* w.r.t. (Red_C,Red_I) if for every fair derivation $(N_i)_i$ such that $N_0 \models \bot$, we have $\bot \in N_i$ for some i.

3

3.4.1. Outline of the Proof

To circumvent the term order's potential nonmonotonicity, our SUP inference rule only considers green subterms. This is reflected in our proof by the reliance on the GF level introduced in Section 3.3.4. The equation $\mathsf{g}_0 \approx \mathsf{f}_0 \in \mathcal{C}_\mathrm{GF}$, which corresponds to the equation $\mathsf{g} \approx \mathsf{f} \in \mathcal{C}_\mathrm{GH}$, cannot be used directly to rewrite the clause $\mathsf{g}_1(\mathsf{a}_0) \not\approx \mathsf{f}_1(\mathsf{a}_0) \in \mathcal{C}_\mathrm{GF}$, which corresponds to $\mathsf{g}\ \mathsf{a} \not\approx \mathsf{f}\ \mathsf{a} \in \mathcal{C}_\mathrm{GH}$. Instead, we first need to apply ARGCONG to derive $\mathsf{g}\, x \approx \mathsf{f}\, x$, which after grounding and transfer to GF yields $\mathsf{g}_1(\mathsf{a}_0) \approx \mathsf{f}_1(\mathsf{a}_0)$. The GF level is a device that enables us to reuse the refutational completeness result for standard (first-order) superposition.

The proof proceeds in three steps, corresponding to the three levels GF, GH, and H introduced in Section 3.3.4:

1. We use Bachmair and Ganzinger's work on the refutational completeness of standard superposition [9] to prove the static refutational completeness of *GFInf*.
2. From the first-order model constructed in Bachmair and Ganzinger's proof, we derive a λ-free higher-order model to prove the static refutational completeness of *GHInf*.
3. We use the saturation framework of Waldmann et al. [140] to lift the static refutational completeness of *GHInf* to static and dynamic refutational completeness of *HInf*.

In the first step, since the inference system *GFInf* is standard ground superposition, we only need to work around minor differences between Bachmair and Ganzinger's definitions and ours. Given a saturated clause set $N \subseteq \mathcal{C}_\mathrm{GF}$ with $\bot \notin N$, Bachmair and Ganzinger prove refutational completeness by constructing a term rewriting system R_N and showing that it can be viewed as an interpretation that is a model of N. This step is exclusively concerned with ground first-order clauses.

In the second step, we derive refutational completeness of *GHInf*. Given a saturated clause set $N \subseteq \mathcal{C}_\mathrm{GH}$ with $\bot \notin N$, we use the first-order model $R_{\mathcal{F}(N)}$ of $\mathcal{F}(N)$ constructed in step (1) to derive a clausal higher-order interpretation that is a model of N. Thanks to saturation w.r.t. GARGCONG, the higher-order interpretation can conflate the interpretations of the members $\mathsf{f}_0^{\bar{\upsilon}}, \ldots, \mathsf{f}_n^{\bar{\upsilon}}$ of a same symbol family. In the extensional calculi, saturation w.r.t. GEXT can be used to show that the constructed interpretation is extensional.

In the third step, we employ the saturation framework by Waldmann et al. [140], which is largely based on Bachmair and Ganzinger's [10], to prove *HInf* refutationally complete. Like Bachmair and Ganzinger's, the saturation framework allows us to prove the static and dynamic refutational completeness of our calculus on the

nonground level. On top of that, it allows us to use the redundancy criterion defined in Section 3.3.4, which supports deletion of subsumed formulas. Moreover, the saturation framework provides completeness theorems for prover architectures, such as the DISCOUNT loop. The main proof obligation the saturation framework leaves to us is that there exist inferences in *HInf* corresponding to all nonredundant inferences in *GHInf*. For monotone term orders, we can avoid SUP inferences into variables x by exploiting the clause order's compatibility with contexts: If $t' \prec t$, we have $C\{x \mapsto t'\} \prec C\{x \mapsto t\}$, which allows us to show that SUP inferences into variables are redundant. This technique fails for variables x that occur applied in C, because the order lacks compatibility with arguments. This is why the calculi must either purify clauses to make this line of reasoning work again or perform some SUP inferences into variables.

3.4.2. The Ground First-Order Level

We use Bachmair and Ganzinger's results on standard superposition [9] to prove GF refutationally complete. In the subsequent steps, we will also make use of specific properties of Bachmair and Ganzinger's model.

Bachmair and Ganzinger's logic and inference system differ in some details from GF:

- Bachmair and Ganzinger use untyped first-order logic, whereas GF's logic is typed. Bachmair and Ganzinger's proof works verbatim for monomorphic first-order logic as well, but we need to require that the order $\succ$ has the subterm property to show that there exist no critical pairs in the term rewriting system, as observed by Wand [141, Section 3.2.1].
- In their redundancy criterion for clauses, Bachmair and Ganzinger require that a finite subset of $\{D \in N \mid D \prec C\}$ entails C, whereas we require that $\{D \in N \mid D \prec C\}$ entails C. By compactness of first-order logic, the two criteria are equivalent.

Bachmair and Ganzinger prove refutational completeness for nonground clause sets, but we only require the ground result here.

The basis of Bachmair and Ganzinger's proof is that a term rewriting system R defines an interpretation $\mathcal{T}_{\mathrm{GF}}/R$ such that for every ground equation $s \approx t$, we have $\mathcal{T}_{\mathrm{GF}}/R \models s \approx t$ if and only if $s \leftrightarrow_R^* t$. Formally, $\mathcal{T}_{\mathrm{GF}}/R$ denotes the monomorphic first-order interpretation whose universes U_τ consist of the R-equivalence classes over $\mathcal{T}_{\mathrm{GF}}$ containing terms of type τ. The interpretation $\mathcal{T}_{\mathrm{GF}}/R$ is term-generated—that is, for every element a of the universe of this interpretation and for any valuation ξ, there exists a ground term t such that $[\![t]\!]^{\xi}_{\mathcal{T}_{\mathrm{GF}}/R} = a$. To lighten notation, we will write R to refer to both the term rewriting system R and the interpretation $\mathcal{T}_{\mathrm{GF}}/R$.

The term rewriting system is constructed as follows. Let $N \subseteq \mathcal{C}_{\mathrm{GF}}$. We first define sets of rewrite rules E_N^C and R_N^C for all $C \in N$ by induction on the clause order. Assume that E_N^D has already been defined for all $D \in N$ such that $D \prec C$. Then $R_N^C = \bigcup_{D \prec C} E_N^D$. Let $E_N^C = \{s \longrightarrow t\}$ if the following conditions are met:

(a) $C = C' \vee s \approx t$;
(b) $s \approx t$ is strictly maximal in C;
(c) $s \succ t$;

(d) C' is false in R_N^C;
(e) s is irreducible w.r.t. R_N^C.

Then C is said to *produce* $s \longrightarrow t$ or to be *productive*. Otherwise, $E_N^C = \emptyset$. Finally, $R_N = \bigcup_D E_N^D$.

We call an inference $\iota \in \mathit{GFInf}$ *B&G-redundant* if some $C \in \mathit{prems}(\iota)$ is true in R_N^C or $\mathit{concl}(\iota)$ is true in $R_N^{\mathit{mprem}(\iota)}$. We call a set $N \subseteq \mathcal{C}_{\mathrm{GF}}$ *B&G-saturated* if all inferences from N are B&G-redundant.

3

Lemma 3.27. *If $\bot \notin N$ and $N \subseteq \mathcal{C}_{\mathrm{GF}}$ is saturated w.r.t. GFInf and $\mathit{GFRed}_{\mathrm{I}}$, then N is B&G-saturated.*

Proof. Let $N \subseteq \mathcal{C}_{\mathrm{GF}}$ be saturated w.r.t. *GFInf* and $\mathit{GFRed}_{\mathrm{I}}$. To show that N is B&G-saturated, let ι be an inference from N. We need to show that ι is B&G-redundant w.r.t. N. We proceed by well-founded induction on $\mathit{mprem}(\iota)$ w.r.t. $\succ$. By the induction hypothesis, for all inferences ι' with $\mathit{concl}(\iota') \prec \mathit{mprem}(\iota)$, ι' is B&G-redundant w.r.t. N. By Lemma 5.5 of Bachmair and Ganzinger, ι is B&G-redundant w.r.t. N. □

Lemma 3.28. *Let $\bot \notin N$ and $N \subseteq \mathcal{C}_{\mathrm{GF}}$ be saturated w.r.t. GFInf and $\mathit{GFRed}_{\mathrm{I}}$. If $C = C' \vee s \approx t \in N$ produces $s \longrightarrow t$, then $s \approx t$ is strictly eligible in C and C' is false in R_N.*

Proof. By Lemma 3.27, N is also B&G-saturated. By condition (d), C' is false in R_N^C. Since $s \succ t$ by condition (c) and s is irreducible w.r.t. R_N^C by condition (e), $s \approx t$ is also false in R_N^C. Hence, C is false in R_N^C. Using this and conditions (a), (b), (c), and (e), we can apply Lemma 4.11 of Bachmair and Ganzinger using N for N and for N', C for C and for D, s for s, and t for t. Part (ii) of that lemma shows that $s \approx t$ is strictly eligible in C, and part (iv) shows that C' is false in R_N. □

Theorem 3.29 (Ground first-order static refutational completeness)**.** *The inference system GFInf is statically refutationally complete w.r.t. ($\mathit{GFRed}_{\mathrm{I}}$,$\mathit{GFRed}_{\mathrm{C}}$). More precisely, if $N \subseteq \mathcal{C}_{\mathrm{GF}}$ is a clause set saturated w.r.t. GFInf and $\mathit{GFRed}_{\mathrm{I}}$ such that $\bot \notin N$, then R_N is a model of N.*

Proof. By Lemma 3.27, N is also B&G-saturated. It follows that R_N is a model of N, as shown in the proof of Theorem 4.14 of Bachmair and Ganzinger. □

3.4.3. The Ground Higher-Order Level

In this subsection, let *GHSel* be a selection function on $\mathcal{C}_{\mathrm{GH}}$, and let $N \subseteq \mathcal{C}_{\mathrm{GH}}$ with $\bot \notin N$ be a clause set saturated w.r.t. $\mathit{GHInf}^{\mathit{GHSel}}$ and $\mathit{GHRed}_{\mathrm{I}}^{\mathit{GHSel}}$. Clearly, $\mathcal{F}(N)$ is then saturated w.r.t. $\mathit{GFInf}^{\mathcal{F}(\mathit{GHSel})}$ and $\mathit{GFRed}_{\mathrm{I}}^{\mathcal{F}(\mathit{GHSel})}$.

We abbreviate $R_{\mathcal{F}(N)}$ as R. From R, we construct a model $\mathcal{I}^{\mathrm{GH}}$ of N. The key properties enabling us to perform this construction are that R is term-generated and that the interpretations of the members $\mathsf{f}_0^{\bar{\nu}}, \ldots, \mathsf{f}_n^{\bar{\nu}}$ of a same symbol family behave in the same way thanks to the ArgCong rule.

Lemma 3.30 (Argument congruence)**.** *For terms $s, t, u \in \mathcal{T}_{\mathrm{GH}}$, if $[\![\mathcal{F}(s)]\!]_R = [\![\mathcal{F}(t)]\!]_R$, then $[\![\mathcal{F}(s\,u)]\!]_R = [\![\mathcal{F}(t\,u)]\!]_R$.*

Proof. What we want to show is equivalent to

$$\mathcal{F}(s) \leftrightarrow_R^* \mathcal{F}(t) \text{ implies } \mathcal{F}(s\,u) \leftrightarrow_R^* \mathcal{F}(t\,u)$$

By induction on the number of rewrite steps and due to symmetry, it suffices to show that

$$\mathcal{F}(s) \rightarrow_R \mathcal{F}(t) \text{ implies } \mathcal{F}(s\,u) \leftrightarrow_R^* \mathcal{F}(t\,u)$$

If the rewrite step from $\mathcal{F}(s)$ is below the top level, this is obvious because there is a corresponding rewrite step from $\mathcal{F}(s\,u)$. If it is at the top level, $\mathcal{F}(s) \rightarrow \mathcal{F}(t)$ must be rule of R. This rule must originate from a productive clause of the form $\mathcal{F}(C) = \mathcal{F}(C' \vee s \approx t)$. By Lemma 3.28, $\mathcal{F}(s \approx t)$ is strictly eligible in $\mathcal{F}(C)$ w.r.t. $\mathcal{F}(GHSel)$, and hence $s \approx t$ is strictly eligible in C w.r.t. $GHSel$. Moreover, s and t have functional type. Thus, the following GArgCong inference ι is applicable:

$$\frac{C' \vee s \approx t}{C' \vee s\,u \approx t\,u}\,\text{GArgCong}$$

By saturation, ι is redundant w.r.t. N—i.e., we have $concl(\iota) \in N \cup GHRed_C(N)$ (for the nonpurifying calculi) or $\mathcal{F}(N) \models concl(\iota)$ (for the purifying calculi). In both cases, by Theorem 3.29, $\mathcal{F}(concl(\iota))$ is then true in R. By Lemma 3.28, $\mathcal{F}(C')$ is false in R. Therefore, $\mathcal{F}(s\,u \approx t\,u)$ must be true in R. □

Lemma 3.31. *For terms $s,t,u,v \in \mathcal{T}_{GH}$, if $[\![\mathcal{F}(s)]\!]_R = [\![\mathcal{F}(t)]\!]_R$ and $[\![\mathcal{F}(u)]\!]_R = [\![\mathcal{F}(v)]\!]_R$, then $[\![\mathcal{F}(s\,u)]\!]_R = [\![\mathcal{F}(t\,v)]\!]_R$.*

Proof. By Lemma 3.30, we have $[\![\mathcal{F}(s\,u)]\!]_R = [\![\mathcal{F}(t\,u)]\!]_R$. It remains to show that $[\![\mathcal{F}(t\,u)]\!]_R = [\![\mathcal{F}(t\,v)]\!]_R$. Since t is ground, it must be of the form $\mathsf{f}\langle\bar{v}_m\rangle\bar{t}_n$. The interpretation R defined above is an interpretation $(U,\mathcal{J})$ in monomorphic first-order logic. Then

$$[\![\mathcal{F}(t\,u)]\!]_R = \mathcal{J}(\mathsf{f}_{n+1}^{\bar{v}_m})([\![\mathcal{F}(\bar{t}_n)]\!]_R, [\![\mathcal{F}(u)]\!]_R) = \mathcal{J}(\mathsf{f}_{n+1}^{\bar{v}_m})([\![\mathcal{F}(\bar{t}_n)]\!]_R, [\![\mathcal{F}(v)]\!]_R) = [\![\mathcal{F}(t\,v)]\!]_R \qquad \square$$

Definition 3.32. Define a higher-order interpretation $\mathcal{I}^{GH} = (\mathcal{U}^{GH}, \mathcal{J}_{ty}^{GH}, \mathcal{J}^{GH}, \mathcal{E}^{GH})$ as follows. The interpretation R defined above is an interpretation $(U,\mathcal{J})$ in monomorphic first-order logic, where U_τ is its universe for type τ, and $\mathcal{J}$ is its interpretation function. Let $\mathcal{U}^{GH} = \{U_\tau \mid \tau \text{ is a ground type}\}$. Let $\mathcal{J}_{ty}^{GH}(\kappa)(U_{\bar{\tau}}) = U_{\kappa(\bar{\tau})}$ for all type constructors κ and type tuples $\bar{\tau}$. Let $\mathcal{J}^{GH}(\mathsf{f}, U_{\bar{\tau}}) = \mathcal{J}(\mathsf{f}_0^{\bar{\tau}})$.

Since R is term-generated, for every $a \in U_{\tau\to\upsilon}$ and $b \in U_\tau$, there exist ground terms $s : \tau \to \upsilon$ and $u : \tau$ such that $[\![\mathcal{F}(s)]\!]_R = a$ and $[\![\mathcal{F}(u)]\!]_R = b$. We define $\mathcal{E}^{GH}$ by $\mathcal{E}^{GH}_{U_\tau,U_\upsilon}(a)(b) = [\![\mathcal{F}(s\,u)]\!]_R$ for any term u. By Lemma 3.31, this definition is independent of the choice of s and u.

Lemma 3.33 (Model transfer to GH)**.** *$\mathcal{I}^{GH}$ is a model of N. In the extensional calculi, $\mathcal{I}^{GH}$ is an extensional model of N.*

Proof. We first prove by induction on terms $t \in \mathcal{T}_{GH}$ that $[\![t]\!]_{\mathcal{I}^{GH}} = [\![\mathcal{F}(t)]\!]_R$. Let $t \in \mathcal{T}_{GH}$, and assume as the induction hypothesis that $[\![u]\!]_{\mathcal{I}^{GH}} = [\![\mathcal{F}(u)]\!]_R$ for all subterms u of t. If t is of the form $\mathsf{f}\langle\bar{v}\rangle$, then

$$[\![t]\!]_{\mathcal{I}^{GH}} = \mathcal{J}^{GH}(\mathsf{f}, U_{\bar{v}}) = \mathcal{J}(\mathsf{f}_0^{\bar{v}}) = [\![\mathsf{f}_0^{\bar{v}}]\!]_R = [\![\mathcal{F}(\mathsf{f}\langle\bar{v}\rangle)]\!]_R = [\![\mathcal{F}(t)]\!]_R$$

If t is an application $t = t_1\, t_2$, where t_1 is of type $\tau \to \upsilon$, then, using the definition of the term denotation and of $\mathcal{E}^{\mathrm{GH}}$, we have

$$[\![t_1\, t_2]\!]_{\mathcal{I}^{\mathrm{GH}}} = \mathcal{E}^{\mathrm{GH}}_{U_\tau, U_\upsilon}([\![t_1]\!]_{\mathcal{I}^{\mathrm{GH}}})([\![t_2]\!]_{\mathcal{I}^{\mathrm{GH}}}) \overset{\mathrm{IH}}{=} \mathcal{E}^{\mathrm{GH}}_{U_\tau, U_\upsilon}([\![\mathcal{F}(t_1)]\!]_R)([\![\mathcal{F}(t_2)]\!]_R) = [\![\mathcal{F}(t_1\, t_2)]\!]_R$$

So we have shown that $[\![t]\!]_{\mathcal{I}^{\mathrm{GH}}} = [\![\mathcal{F}(t)]\!]_R$ for all terms t. It follows that a ground equation $s \approx t$ is true in $\mathcal{I}^{\mathrm{GH}}$ if and only if $\mathcal{F}(s \approx t)$ is true in R. Hence a ground clause C is true in $\mathcal{I}^{\mathrm{GH}}$ if and only if $\mathcal{F}(C)$ is true in R. By Theorem 3.29, R is a model of $\mathcal{F}(N)$. Thus, $\mathcal{I}^{\mathrm{GH}}$ is a model of N.

3

For the extensional calculi, it remains to show that $\mathcal{I}^{\mathrm{GH}}$ is extensional—i.e., we have to show that for all τ and υ and all $a, b \in U_{\tau\to\upsilon}$, if $a \neq b$, then $\mathcal{E}^{\mathrm{GH}}(a) \neq \mathcal{E}^{\mathrm{GH}}(b)$. Since R is term-generated, there are terms $s, t \in \mathcal{T}_{\mathrm{GF}}$ such that $[\![s]\!]_R = a$ and $[\![t]\!]_R = b$. By what we have shown above, it follows that $[\![s']\!]_{\mathcal{I}^{\mathrm{GH}}} = a$ and $[\![t']\!]_{\mathcal{I}^{\mathrm{GH}}} = b$ for $s' = \mathcal{F}^{-1}(s)$ and $t' = \mathcal{F}^{-1}(t)$.

Since N is saturated, the GExt inference that generates the clause

$$C \;=\; s'\,(\mathsf{diff}\langle\tau,\upsilon\rangle\, s'\, t') \not\approx t'\,(\mathsf{diff}\langle\tau,\upsilon\rangle\, s'\, t') \vee s' \approx t'$$

is redundant—i.e., $C \in N \cup \mathit{GHRed}_{\mathrm{C}}(N)$ (in the nonpurifying calculi) or $\mathcal{F}(N) \models \mathcal{F}(C)$ (in the purifying calculi). In both cases, it follows that $R \models \mathcal{F}(C)$ by Theorem 3.29 and thus $\mathcal{I}^{\mathrm{GH}} \models C$ by what we have shown above. The second literal of C is false in $\mathcal{I}^{\mathrm{GH}}$ because $[\![s']\!]_{\mathcal{I}^{\mathrm{GH}}} = a \neq b = [\![t']\!]_{\mathcal{I}^{\mathrm{GH}}}$. So the first literal of C must be true in $\mathcal{I}^{\mathrm{GH}}$ and thus

$$\begin{aligned}\mathcal{E}^{\mathrm{GH}}(a)([\![\mathsf{diff}\langle\tau,\upsilon\rangle\, s'\, t']\!]_{\mathcal{I}^{\mathrm{GH}}}) &= [\![s'\,(\mathsf{diff}\langle\tau,\upsilon\rangle\, s'\, t')]\!]_{\mathcal{I}^{\mathrm{GH}}} \\ &\neq [\![t'\,(\mathsf{diff}\langle\tau,\upsilon\rangle\, s'\, t')]\!]_{\mathcal{I}^{\mathrm{GH}}} = \mathcal{E}^{\mathrm{GH}}(b)([\![\mathsf{diff}\langle\tau,\upsilon\rangle\, s'\, t']\!]_{\mathcal{I}^{\mathrm{GH}}})\end{aligned}$$

It follows that $\mathcal{E}^{\mathrm{GH}}(a) \neq \mathcal{E}^{\mathrm{GH}}(b)$. □

We summarize the results of this subsection in the following theorem:

Theorem 3.34 (Ground static refutational completeness)**.** *Let GHSel be a selection function on $\mathcal{C}_{\mathrm{GH}}$. Then the inference system $\mathit{GHInf}^{\mathit{GHSel}}$ is statically refutationally complete w.r.t. $(\mathit{GHRed}_{\mathrm{I}}, \mathit{GHRed}_{\mathrm{C}})$. That means, if $N \subseteq \mathcal{C}_{\mathrm{GH}}$ is saturated w.r.t. $\mathit{GHInf}^{\mathit{GHSel}}$ and $\mathit{GHRed}_{\mathrm{I}}^{\mathit{GHSel}}$, then $N \models \bot$ if and only if $\bot \in N$.*

The construction of $\mathcal{I}^{\mathrm{GH}}$ relies on the specific properties of R. It would not work with an arbitrary first-order interpretation. Transforming a λ-free higher-order interpretation into a first-order interpretation is easier:

Lemma 3.35. *Given an interpretation $\mathcal{I}$ on* GH*, there exists an interpretation $\mathcal{I}^{\mathrm{GF}}$ on* GF *such that for any clause $C \in \mathcal{C}_{\mathrm{GH}}$ the truth values of C in $\mathcal{I}$ and of $\mathcal{F}(C)$ in $\mathcal{I}^{\mathrm{GF}}$ coincide.*

Proof. Let $\mathcal{I} = (\mathcal{U}, \mathcal{J}_{\mathsf{ty}}, \mathcal{J}, \mathcal{E})$ be a λ-free higher-order interpretation. Let $\mathcal{U}^{\mathrm{GF}}_\tau = [\![\tau]\!]_{\mathcal{I}_{\mathsf{ty}}}$ be the first-order type universe for the ground type τ. For a symbol $\mathsf{f}_l^{\bar\upsilon_m} \in \Sigma_{\mathrm{GF}}$ and universe elements $\bar a_l$, let $\mathcal{J}^{\mathrm{GF}}(\mathsf{f}_l^{\bar\upsilon_m})(\bar a_l) = [\![\mathsf{f}\langle\bar\upsilon_m\rangle\, \bar x_l]\!]_{\mathcal{I}}^{\{\bar x_l \mapsto \bar a\}}$. This defines an interpretation $\mathcal{I}^{\mathrm{GF}} = (\mathcal{U}^{\mathrm{GF}}, \mathcal{J}^{\mathrm{GF}})$ on GF.

We need to show that for any $C \in \mathcal{C}_{\mathrm{GH}}$, $\mathcal{I} \models C$ if and only if $\mathcal{I}^{\mathrm{GF}} \models \mathcal{F}(C)$. It suffices to show that $[\![t]\!]_{\mathcal{I}} = [\![\mathcal{F}(t)]\!]_{\mathcal{I}^{\mathrm{GF}}}$ for all terms $t \in \mathcal{T}_{\mathrm{GH}}$. We prove this by induction on t. Since t is ground, it must be of the form $\mathrm{f}\langle \bar{v}_m \rangle \bar{s}_l$. Then $\mathcal{F}(t) = \mathrm{f}_l^{\bar{v}_m}(\mathcal{F}(\bar{s}_l))$ and hence

$$[\![\mathcal{F}(t)]\!]_{\mathcal{I}^{\mathrm{GF}}} = \mathcal{J}^{\mathrm{GF}}(\mathrm{f}_l^{\bar{v}_m})([\![\mathcal{F}(\bar{s}_l)]\!]_{\mathcal{I}^{\mathrm{GF}}}) \overset{\mathrm{IH}}{=} \mathcal{J}^{\mathrm{GF}}(\mathrm{f}_l^{\bar{v}_m})([\![\bar{s}_l]\!]_{\mathcal{I}}) = [\![\mathrm{f}\langle \bar{v}_m \rangle \bar{s}_l]\!]_{\mathcal{I}} = [\![t]\!]_{\mathcal{I}}$$

using the definition of $\mathcal{J}^{\mathrm{GF}}$ and Lemma 3.11 for the third step. □

3.4.4. The Nonground Higher-Order Level

To lift the result to the nonground level, we employ the saturation framework of Waldmann et al. [140]. It is easy to see that the entailment relation $\models$ on GH is a consequence relation in the sense of the framework. It remains to show that our redundancy criterion on GH is a redundancy criterion in the sense of the framework and that $\mathcal{G}$ is a grounding function in the sense of the framework:

Lemma 3.36. *The redundancy criterion for* GH *is a redundancy criterion in the sense of the framework.*

Proof. We must prove the conditions (R1) to (R4) defined by Waldmann et al., which, adapted to our context, state the following for all clause sets $N, N' \subseteq \mathcal{C}_{\mathrm{GH}}$:

(R1) if $N \models \bot$, then $N \setminus \mathit{GHRed}_{\mathrm{C}}(N) \models \bot$;
(R2) if $N \subseteq N'$, then $\mathit{GHRed}_{\mathrm{C}}(N) \subseteq \mathit{GHRed}_{\mathrm{C}}(N')$ and $\mathit{GHRed}_{\mathrm{I}}(N) \subseteq \mathit{GHRed}_{\mathrm{I}}(N')$;
(R3) if $N' \subseteq \mathit{GHRed}_{\mathrm{C}}(N)$, then $\mathit{GHRed}_{\mathrm{C}}(N) \subseteq \mathit{GHRed}_{\mathrm{C}}(N \setminus N')$ and $\mathit{GHRed}_{\mathrm{I}}(N) \subseteq \mathit{GHRed}_{\mathrm{I}}(N \setminus N')$;
(R4) if $\iota \in \mathit{GHInf}$ and $\mathit{concl}(\iota) \in N$, then $\iota \in \mathit{GHRed}_{\mathrm{I}}(N)$.

(R1) It suffices to show that $N \setminus \mathit{GHRed}_{\mathrm{C}}(N) \models N$ for $N \subseteq \mathcal{C}_{\mathrm{GH}}$. Let $\mathcal{I}$ be a model of $N \setminus \mathit{GHRed}_{\mathrm{C}}(N)$. In the extensional calculi, let $\mathcal{I}$ be extensional. Then by Lemma 3.35, there exists a model $\mathcal{I}^{\mathrm{GF}}$ of $\mathcal{F}(N \setminus \mathit{GHRed}_{\mathrm{C}}(N)) = \mathcal{F}(N) \setminus \mathit{GFRed}_{\mathrm{C}}(\mathcal{F}(N))$. By Lemma 5.2 of Bachmair and Ganzinger, this is also a model of $\mathcal{F}(N)$. By Lemma 3.35, it follows that $\mathcal{I} \models N$.

(R2) By Lemma 5.6(i) of Bachmair and Ganzinger, this holds on GF. For clauses and all inferences except GArgCong and GExt, this implies that it holds on GH as well because $\mathcal{F}$ is a redundancy-preserving bijection between $\mathcal{C}_{\mathrm{GH}}$ and $\mathcal{C}_{\mathrm{GF}}$ and between these inferences. For GArgCong and GExt inferences, it holds because it holds on clauses.

(R3) The proof is analogous to (R2), with Lemma 5.6(ii) of Bachmair and Ganzinger instead of Lemma 5.6(i).

(R4) We must show that for all inferences with $\mathit{concl}(\iota) \in N$, we have $\iota \in \mathit{GHRed}_{\mathrm{I}}(N)$. Since $\mathit{concl}(\iota) \prec \mathit{mprem}(\iota)$ for all $\iota \in \mathit{GFInf}$, this holds on GF. For all inferences except GArgCong and GExt, since $\mathcal{F}$ is a bijection preserving redundancy, it follows that it also holds also on GH. For GArgCong and GExt inferences, it holds by definition. □

Lemma 3.37. *The grounding functions $\mathcal{G}^{\mathit{GHSel}}$ for $\mathit{GHSel} \in \mathcal{G}(\mathit{HSel})$ are grounding functions in the sense of the framework.*

Proof. We must prove the conditions (G1) to (G3) defined by Waldmann et al., which, adapted to our context, state the following:

(G1) $\mathcal{G}(\bot) = \{\bot\}$;

(G2) for every $C \in \mathcal{C}_{\mathrm{GH}}$, if $\bot \in \mathcal{G}(C)$, then $C = \bot$;

(G3) for every inference $\iota \in \mathit{HInf}$, if $\mathcal{G}^{\mathit{GHSel}}(\iota) \neq \mathit{undef}$, then we have $\mathcal{G}^{\mathit{GHSel}}(\iota) \subseteq \mathit{GHRed}_{\mathrm{I}}^{\mathit{GHSel}}(\mathcal{G}(\mathit{concl}(\iota)))$.

Clearly, $C = \bot$ if and only if $\bot \in \mathcal{G}(C)$ if and only if $\mathcal{G}(C) = \{\bot\}$, proving (G1) and (G2). For (G3), we have to show for all non-PosExt inferences $\iota \in \mathit{HInf}$ that $\mathcal{G}^{\mathit{GHSel}}(\iota) \subseteq \mathit{GHRed}_{\mathrm{I}}^{\mathit{GHSel}}(\mathcal{G}(\mathit{concl}(\iota)))$. Let $\iota \in \mathit{HInf}$ and $\iota' \in \mathcal{G}^{\mathit{GHSel}}(\iota)$. By the definition of $\mathcal{G}^{\mathit{GHSel}}$, there exists a grounding substitution θ such that $\mathit{prems}(\iota') = \mathit{prems}(\iota)\theta$ and $\mathit{concl}(\iota') = \mathit{preconcl}(\iota)\theta$. We want to show that $\iota' \in \mathit{GHRed}_{\mathrm{I}}^{\mathit{GHSel}}(\mathcal{G}(\mathit{concl}(\iota)))$.

If ι' is not an GArgCong or GExt inference, by the definition of inference redundancy, it suffices to show that $\{D \in \mathcal{F}(\mathcal{G}(\mathit{concl}(\iota))) \mid D \prec \mathit{mprem}(\mathcal{F}(\iota'))\} \models \mathit{concl}(\mathcal{F}(\iota'))$. We define a substitution θ' that extends θ to all variables in $\mathit{concl}(\iota)$. Due to purification, the clause $\mathit{concl}(\iota)$ may contain variables not present in $\mathit{preconcl}(\iota)$. For each such variable x', let x be the variable in $\mathit{preconcl}(\iota)$ that x' stems from and define $x'\theta' = x\theta$. Then the clause $\mathcal{F}(\mathit{concl}(\iota)\theta')$ differs from the clause $\mathcal{F}(\mathit{concl}(\iota')) = \mathcal{F}(\mathit{preconcl}(\iota)\theta')$ only in some additional grounded purification literals, which all have the form $t \not\approx t$ and are thus trivially false in any interpretation. Hence, $\mathcal{F}(\mathit{concl}(\iota)\theta') \models \mathcal{F}(\mathit{concl}(\iota'))$. Since one of the variables of a purification literal must appear applied in the clause, for each grounded purification literal $t \not\approx t$ the term t must be smaller than the maximal term of the clause $\mathcal{F}(\mathit{concl}(\iota'))$.

If no literals are selected in $\mathit{mprem}(\mathcal{F}(\iota'))$, inspection of the rules in *GFInf* shows that $\mathcal{F}(\mathit{concl}(\iota)\theta') \prec \mathit{mprem}(\mathcal{F}(\iota'))$. Otherwise, ι' can only be an ERes inference or a Sup inference into a negative literal. If it is an ERes inference, due to the selection restrictions, the substitution σ used in ι is the identity for all variables of functional type. Therefore, applying σ cannot trigger any purification and hence $\mathcal{F}(\mathit{concl}(\iota)\theta') = \mathcal{F}(\mathit{preconcl}(\iota)\theta') \prec \mathit{mprem}(\mathcal{F}(\iota'))$. If ι' is a Sup inference into a negative literal, due to the selection restrictions, the substitution $\sigma = \mathrm{mgu}(t, u)$ used in ι is the identity for all variables of functional type that stem from the main premise. Therefore the variables from the main premise C need not be purified. The variables from the side premise D might need to be purified, yielding purification literals of the form $x \not\approx y$ where $x\theta' = y\theta'$. Then x or y must appear applied in D and hence $x\theta'$ is smaller than $t\theta'$. Again, it follows that $\mathcal{F}(\mathit{concl}(\iota)\theta') \prec \mathit{mprem}(\mathcal{F}(\iota'))$.

This proves $\{D \in \mathcal{F}(\mathcal{G}(\mathit{concl}(\iota))) \mid D \prec \mathit{mprem}(\mathcal{F}(\iota'))\} \models \mathit{concl}(\mathcal{F}(\iota'))$.

In the nonpurifying calculi, if ι' is an GArgCong or GExt inference, it suffices to show that $\mathit{concl}(\iota') \in \mathcal{G}(\mathit{concl}(\iota))$. This holds because $\mathit{concl}(\iota') = \mathit{preconcl}(\iota)\theta = \mathit{concl}(\iota)\theta$. In the purifying calculi, if ι' is an GArgCong or GExt inference, we must show that $\mathcal{F}(\mathcal{G}(\mathit{concl}(\iota))) \models \mathcal{F}(\mathit{concl}(\iota'))$. Defining θ' as above, we have $\mathcal{F}(\mathit{concl}(\iota)\theta') \models \mathcal{F}(\mathit{concl}(\iota'))$, as desired. □

To lift the completeness result of the previous section to the nonground calculus *HInf*, we employ Theorem 14 of Waldmann et al., which, adapted to our context, is stated as follows. The theorem uses the notation $\mathit{Inf}(N)$ to denote the set of *Inf*-inferences whose premises are in N, for an inference system *Inf* and a clause set N. Moreover, it uses the Herbrand entailment $\models_{\mathcal{G}}$ on $\mathcal{C}_{\mathrm{H}}$, which is defined as

$N_1 \models_{\mathcal{G}} N_2$ if $\mathcal{G}(N_1) \models \mathcal{G}(N_2)$.

Theorem 3.38 (Lifting theorem)**.** *If $\mathit{GHInf}^{\mathit{GHSel}}$ is statically refutationally complete w.r.t. $(\mathit{GHRed}_{\mathrm{I}}^{\mathit{GHSel}}, \mathit{GHRed}_{\mathrm{C}})$ for every $\mathit{GHSel} \in \mathcal{G}(\mathit{HSel})$, and if for every $N \subseteq \mathcal{C}_{\mathrm{H}}$ that is saturated w.r.t. HInf and $\mathit{HRed}_{\mathrm{I}}$ there exists a $\mathit{GHSel} \in \mathcal{G}(\mathit{HSel})$ such that $\mathit{GHInf}^{\mathit{GHSel}}(\mathcal{G}(N)) \subseteq \mathcal{G}^{\mathit{GHSel}}(\mathit{HInf}(N)) \cup \mathit{GHRed}_{\mathrm{I}}^{\mathit{GHSel}}(\mathcal{G}(N))$, then HInf is statically refutationally complete w.r.t. $(\mathit{HRed}_{\mathrm{I}}, \mathit{HRed}_{\mathrm{C}})$ and $\models_{\mathcal{G}}$.*

Proof. This is essentially Theorem 14 of Waldmann et al. We take H for **F**, GH for **G**, and $\mathcal{G}(\mathit{HSel})$ for $\boldsymbol{Q}$. It is easy to see that the entailment relation $\models$ on GH is a consequence relation in the sense of the framework. By Lemma 3.36 and 3.37, $(\mathit{GHRed}_{\mathrm{I}}^{\mathit{GHSel}}, \mathit{GHRed}_{\mathrm{C}})$ is a redundancy criterion in the sense of the framework, and $\mathcal{G}^{\mathit{GHSel}}$ are grounding functions in the sense of the framework, for all $\mathit{GHSel} \in \mathcal{G}(\mathit{HSel})$. The redundancy criterion $(\mathit{HRed}_{\mathrm{I}}, \mathit{HRed}_{\mathrm{C}})$ matches exactly the intersected lifted redundancy criterion $\mathit{Red}^{\cap \mathcal{G}, \sqsupset}$ of Waldmann et al. Theorem 14 of Waldmann et al. states the theorem only for $\sqsupset = \varnothing$. By Lemma 16 of Waldmann et al., it also holds if $\sqsupset \neq \varnothing$. □

Let $N \subseteq \mathcal{C}_{\mathrm{H}}$ be a clause set saturated w.r.t. HInf and $\mathit{HRed}_{\mathrm{I}}$. We assume that HSel fulfills the selection restrictions introduced in Section 3.3.1. For the above theorem to apply, we need to show that there exists a selection function $\mathit{GHSel} \in \mathcal{G}(\mathit{HSel})$ such that all inferences $\iota \in \mathit{GHInf}^{\mathit{GHSel}}$ with $\mathit{prems}(\iota) \in \mathcal{G}(N)$ are liftable or redundant. Here, by *liftable*, we mean that ι is a $\mathcal{G}^{\mathit{GHSel}}$-ground instance of a HInf-inference from N; by *redundant*, we mean that $\iota \in \mathit{GHRed}_{\mathrm{I}}^{\mathit{GHSel}}(\mathcal{G}(N))$.

To choose the right selection function $\mathit{GHSel} \in \mathcal{G}(\mathit{HSel})$, we observe that each ground clause $C \in \mathcal{G}(N)$ must have at least one corresponding clause $D \in N$ such that C is a ground instance of D. We choose one of them for each $C \in \mathcal{G}(N)$, which we denote by $\mathcal{G}^{-1}(C)$. Then let GHSel select those literals in C that correspond to the literals selected by HSel in $\mathcal{G}^{-1}(C)$. Given this selection function GHSel, we can show that all inferences from $\mathcal{G}(N)$ are liftable or redundant.

All non-SUP inferences in GHInf are liftable (Lemma 3.40). For SUP, some inferences are liftable (Lemma 3.41) and some are redundant (Lemma 3.42). As in standard superposition, SUP inferences into positions below variables are redundant. The variable condition of each of the four calculi is designed to cover the nonredundant SUP inferences into positions of variable-headed terms, which makes these inferences liftable.

Lemma 3.39. *Let σ be the most general unifier of s and s'. Let θ be an arbitrary unifier of s and s'. Then $\sigma\theta = \theta$.*

Proof. Like in first-order logic, we can assume that σ is idempotent without loss of generality [58, Corollary 7.2.11]. Since σ is most general, there exists a substitution ρ such that $\sigma\rho = \theta$. Therefore, by idempotence, $\sigma\theta = \sigma\sigma\rho = \sigma\rho = \theta$. □

Lemma 3.40 (Lifting of ERES, EFACT, GARGCONG, and GEXT)**.** *All* ERES*,* EFACT*,* GARGCONG*, and* GEXT *inferences are liftable.*

Proof. ERES: Let $\iota \in \mathit{GHInf}^{\mathit{GHSel}}$ be an ERES inference with $\mathit{prems}(\iota) \in \mathcal{G}(N)$. Then ι is of the form

$$\frac{C\theta \;=\; C'\theta \vee s\theta \not\approx s'\theta}{C'\theta}\,\textsc{ERes}$$

where $\mathcal{G}^{-1}(C\theta) = C = C' \vee s \not\approx s'$ and the literal $s\theta \not\approx s'\theta$ is eligible w.r.t. GHSel. Let $\sigma = \mathrm{mgu}(s, s')$. It follows that $s \not\approx s'$ is eligible in C w.r.t. σ and HSel. Moreover, $s\theta$ and $s'\theta$ are unifiable and ground, and therefore $s\theta = s'\theta$. Thus, the following inference $\iota' \in \mathit{HInf}$ is applicable:

$$\frac{C' \vee s \not\approx s'}{\mathit{pure}(C'\sigma)}\,\textsc{ERes}$$

(where *pure* is the identity in the nonpurifying calculi). By Lemma 3.39, we have $C'\sigma\theta = C'\theta$. Therefore, ι is the θ-ground instance of ι' and is therefore liftable.

EFACT: Analogously, if $\iota \in \mathit{GHInf}^{\mathit{GHSel}}$ is an EFACT inference with $\mathit{prems}(\iota) \in \mathcal{G}(N)$, then ι is of the form

$$\frac{C\theta \;=\; C'\theta \vee s'\theta \approx t'\theta \vee s\theta \approx t\theta}{C'\theta \vee t\theta \not\approx t'\theta \vee s\theta \approx t'\theta}\,\textsc{EFact}$$

where $\mathcal{G}^{-1}(C\theta) = C = C' \vee s' \approx t' \vee s \approx t$, the literal $s\theta \approx t\theta$ is eligible in C w.r.t. GHSel, and $s\theta \not\prec t\theta$. Let $\sigma = \mathrm{mgu}(s, s')$. Hence, $s \approx t$ is eligible in C w.r.t. σ and HSel. We have $s \not\prec t$. Moreover, $s\theta$ and $s'\theta$ are unifiable and ground. Hence, $s\theta = s'\theta$. Thus, the following inference $\iota' \in \mathit{HInf}$ is applicable:

$$\frac{C' \vee s' \approx t' \vee s \approx t}{\mathit{pure}((C' \vee t \not\approx t' \vee s \approx t')\sigma)}\,\textsc{EFact}$$

By Lemma 3.39, we have $\mathit{preconcl}(\iota')\theta = \mathit{concl}(\iota)$. Hence, ι is the θ-ground instance of ι' and is therefore liftable.

GARGCONG: Let $\iota \in \mathit{GHInf}^{\mathit{GHSel}}$ be a GARGCONG inference with $\mathit{prems}(\iota) \in \mathcal{G}(N)$. Then ι is of the form

$$\frac{C\theta \;=\; C'\theta \vee s\theta \approx s'\theta}{C'\theta \vee s\theta\,\bar{u}_n \approx s'\theta\,\bar{u}_n}\,\textsc{GArgCong}$$

where $\mathcal{G}^{-1}(C\theta) = C = C' \vee s \approx s'$, the literal $s\theta \approx s'\theta$ is strictly eligible w.r.t. GHSel, and $s\theta$ and $s'\theta$ are of functional type. It follows that s and s' have either a functional or a polymorphic type. Let σ be the most general substitution such that $s\sigma$ and $s'\sigma$ take n arguments. Then $s \not\approx s'$ is eligible in C w.r.t. σ and HSel. Hence the following inference $\iota' \in \mathit{HInf}$ is applicable:

$$\frac{C' \vee s \approx s'}{\mathit{pure}(C'\sigma \vee s\sigma\,\bar{x}_n \approx s'\sigma\,\bar{x}_n)}\,\textsc{ArgCong}$$

Then ι is a ground instance of ι' and is therefore liftable.

GExt: The conclusion of a GExt inference in *GHInf* is by definition a ground instance of the conclusion of the Ext inference in *HInf* before purification. Hence, the GExt inference is a ground instance of the Ext inference. Therefore it is liftable. □

Lemma 3.41 (Lifting of Sup)**.** *Let $\iota \in GHInf^{GHSel}$ be a* Sup *inference*

$$\frac{D'\theta \vee t\theta \approx t'\theta \quad C'\theta \vee s\theta\langle t\theta\rangle_p \mathrel{\dot\approx} s'\theta}{D'\theta \vee C'\theta \vee s\theta\langle t'\theta\rangle_p \mathrel{\dot\approx} s'\theta}\,\text{Sup}$$

where $\mathcal{G}^{-1}(D\theta) = D = D' \vee t \approx t'$ and $\mathcal{G}^{-1}(C\theta) = C = C' \vee s \mathrel{\dot\approx} s'$. Suppose that the position p exists as a green subterm in s. Let u be the green subterm of s at that position and $\sigma = \mathrm{mgu}(t,u)$ (which exists since θ is a unifier). If the variable condition holds for C, t, t', u, and σ, then ι is liftable.

Proof. The inference conditions of ι can be lifted to D and C. That $t\theta \approx t'\theta$ is strictly eligible in $D\theta$ w.r.t. *GHSel* implies that $t \approx t'$ is strictly eligible in D w.r.t. σ and *HSel*. If $s\theta \mathrel{\dot\approx} s'\theta$ is (strictly) eligible in $C\theta$ w.r.t. *GHSel*, then $s \mathrel{\dot\approx} s'$ is (strictly) eligible in C w.r.t. σ and *HSel*. Moreover, $D\theta \not\succeq C\theta$ implies $D \not\succeq C$, $t\theta \not\prec t'\theta$ implies $t \not\prec t'$, and $s\theta \not\prec s'\theta$ implies $s \not\prec s'$.

By assumption, p is a position of s and the variable condition holds. Thus, the following inference $\iota' \in HInf$ is applicable:

$$\frac{D' \vee t \approx t' \quad C' \vee s\langle u\rangle_p \mathrel{\dot\approx} s'}{\mathit{pure}((D' \vee C' \vee s\langle t'\rangle_p \mathrel{\dot\approx} s')\sigma)}\,\text{Sup}$$

By Lemma 3.39, we have $(\mathit{preconcl}(\iota'))\theta = \mathit{concl}(\iota)$. Hence, ι is the θ-ground instance of ι' and is therefore liftable. □

The other Sup inferences might not be liftable, but they are redundant:

Lemma 3.42. *Let $\iota \in GHInf^{GHSel}$ be a* Sup *inference*

$$\frac{D'\theta \vee t\theta \approx t'\theta \quad C'\theta \vee s\theta\langle t\theta\rangle_p \mathrel{\dot\approx} s'\theta}{D'\theta \vee C'\theta \vee s\theta\langle t'\theta\rangle_p \mathrel{\dot\approx} s'\theta}\,\text{Sup}$$

where $\mathcal{G}^{-1}(D\theta) = D = D' \vee t \approx t'$ and $\mathcal{G}^{-1}(C\theta) = C = C' \vee s \mathrel{\dot\approx} s'$. Suppose that Lemma 3.41 does not apply. This could be either because the position p is below a variable in s or because the variable condition does not hold. Then $\iota \in GHRed_{\mathrm{I}}^{GHSel}(\mathcal{G}(N))$.

Proof. By the definition of $GHRed_{\mathrm{I}}$, to show $\iota \in GHRed_{\mathrm{I}}^{GHSel}(\mathcal{G}(N))$, it suffices to prove that $\{E \in \mathcal{F}(\mathcal{G}(N)) \mid E \prec \mathcal{F}(C\theta)\} \models \mathcal{F}(\mathit{concl}(\iota))$. Let $\mathcal{I}$ be a first-order model of all $E \in \mathcal{F}(\mathcal{G}(N))$ with $E \prec \mathcal{F}(C\theta)$. We must show that $\mathcal{I} \models \mathcal{F}(\mathit{concl}(\iota))$. If $\mathcal{I} \models \mathcal{F}(D'\theta)$, this is obvious. So we further assume that $\mathcal{I} \not\models \mathcal{F}(D'\theta)$. Since $D\theta \prec C\theta$ by the Sup inference conditions, it follows that $\mathcal{I} \models \mathcal{F}(t\theta \approx t'\theta)$. By congruence, it suffices to show $\mathcal{I} \models \mathcal{F}(C\theta)$. We proceed by a case distinction on the two possible reasons why Lemma 3.41 does not apply:

CASE 1: The position p is below a variable in s. Then $t\theta$ is a proper green subterm of $x\theta$ and hence a green subterm of $x\theta\,\bar{w}$ for any arguments $\bar{w}$. Let v be the term that we obtain by replacing $t\theta$ by $t'\theta$ in $x\theta$ at the relevant position. It follows from our assumptions about $\mathcal{I}$ that $\mathcal{I} \models \mathcal{F}(t\theta \approx t'\theta)$, and by congruence, $\mathcal{I} \models \mathcal{F}(x\theta\,\bar{w} \approx v\,\bar{w})$ for any arguments $\bar{w}$. Hence, $\mathcal{I} \models \mathcal{F}(C\theta)$ if and only if $\mathcal{I} \models \mathcal{F}(C\{x \mapsto v\}\theta)$. By the inference conditions we have $t\theta \succ t'\theta$, which implies $\mathcal{F}(C\theta) \succ \mathcal{F}(C\{x \mapsto v\}\theta)$ by compatibility with green contexts. Therefore, we have $\mathcal{I} \models \mathcal{F}(C\{x \mapsto v\}\theta)$ and hence $\mathcal{I} \models \mathcal{F}(C\theta)$.

CASE 2: The variable condition does not hold. In the extensional calculi, it follows that u has a variable head and jells with $t \approx t'$. By Definition 3.2, this means that u, t, and t' have the following form: $u = x\,\bar{v}_n$ for some variable x and a tuple of terms $\bar{v}_n$ of length $n \geq 0$; $t = \tilde{t}\,\bar{x}_n$ and $t' = \tilde{t}'\,\bar{x}_n$, where $\bar{x}_n$ are variables that do not occur elsewhere in D.

For the intensional calculi, we have $u \in \mathcal{V}$. Thus, u, t, and t' can be written in the same form as described above for the extensional calculi, with $n = 0$.

CASE 2.1 (PURIFYING CALCULI): First, we assume that x occurs only with arguments $\bar{v}_n$ in C. For the intensional calculus, this must be the case because $n = 0$ and hence x can only occur without arguments by the definition of *pure* and the literal selection restriction. Define a substitution θ' by $x\theta' = \tilde{t}'\theta$ and $y\theta' = y\theta$ for other variables y. Since $t\theta \succ t'\theta$ by the inference conditions, we have $C\theta \succ C\theta'$. Moreover, $C\theta' \in \mathcal{G}(N)$. Then $\mathcal{I} \models \mathcal{F}(C\theta)$ by congruence, because $\mathcal{I} \models \mathcal{F}(C\theta')$ and $\mathcal{I} \models \mathcal{F}(t\theta \approx t'\theta)$.

Now we assume that x occurs with arguments other than $\bar{v}_n$ in C. This can only happen in the extensional calculus and by the selection restrictions, $s\theta \mathrel{\dot{\approx}} s'\theta$ must not be selected in $C\theta$. Therefore, $s\theta$ is the maximal term in $C\theta$. Then $s \neq x$ and hence $\bar{v}_n \neq \varepsilon$ because otherwise $s\theta = x\theta$ would be smaller than the applied occurrence of $x\theta$ in $C\theta$.

Define a substitution θ'' such that $x\theta'' = \tilde{t}'\theta$, $y\theta'' = \tilde{t}'\theta$ for other variables y if $y\theta = s\theta$ and C contains the literal $x \not\approx y$, and $y\theta'' = y\theta$ otherwise.

We show that $C\theta \succ C\theta''$ by proving that no literal of $C\theta''$ is larger than the maximal literal $s\theta \mathrel{\dot{\approx}} s'\theta$ of $C\theta$ and that $s\theta \mathrel{\dot{\approx}} s'\theta$ appears more often in $C\theta$ than in $C\theta''$.

For a literal of the form $x \not\approx y$, we have $x\theta'' \prec s\theta$ and $y\theta'' \prec s\theta$. For literals that are not of this form, by the definition of *pure* in the extensional calculus, x occurs always with arguments $\bar{v}_n$. Hence these literals are equal or smaller in $C\theta''$ than in $C\theta$, because $x\theta''\,\bar{v}_n \prec x\theta\,\bar{v}_n$ and $y\theta'' \preceq y\theta$. Therefore, no literal of $C\theta''$ is larger than the maximal literal $s\theta \mathrel{\dot{\approx}} s'\theta$ of $C\theta$. Moreover, these inequalities show that every occurrence of $s\theta \mathrel{\dot{\approx}} s'\theta$ in $C\theta''$ corresponds to an occurrence of $s\theta \mathrel{\dot{\approx}} s'\theta$ in $C\theta$ that corresponds to a literal in C without the variable x. Since at least one occurrence of $s\theta \mathrel{\dot{\approx}} s'\theta$ in $C\theta$ corresponds to a literal in C containing x, $s\theta \mathrel{\dot{\approx}} s'\theta$ appears more often in $C\theta$ than in $C\theta''$. This concludes the argument that $C\theta \succ C\theta''$. It follows that $\mathcal{I} \models \mathcal{F}(C\theta'')$.

We need to show that $\mathcal{I} \models \mathcal{F}(C\theta)$. There is a POSEXT inference from D to $D' \vee \tilde{t} \approx \tilde{t}'$. This inference is in $HRed_{\mathrm{I}}(N)$ because N is saturated. Therefore, $D'\theta \vee \tilde{t}\theta \approx \tilde{t}'\theta$ is in $\mathcal{G}(N) \cup GHRed_{\mathrm{C}}(\mathcal{G}(N))$. It follows that $\mathcal{I} \models \mathcal{F}(D'\theta \vee \tilde{t}\theta \approx \tilde{t}'\theta)$ because this clause is smaller than $\mathcal{F}(D'\theta)$ and hence smaller than $\mathcal{F}(C\theta)$. Since $\mathcal{F}(D'\theta)$ is false in $\mathcal{I}$, we have $\mathcal{I} \models \mathcal{F}(\tilde{t}\theta \approx \tilde{t}'\theta)$.

For every literal of the form $x \not\approx y$, where $y\theta = s\theta$, the variable y can only occur without arguments in C because of the maximality of $s\theta$. We distinguish two cases. If for every literal of the form $x \not\approx y$ where $y\theta = s\theta$, we have $\mathcal{I} \models \mathcal{F}(y\theta'' \approx y\theta)$, then $\mathcal{I} \models \mathcal{F}(C\theta)$ by congruence. If for some literal of the form $x \not\approx y$ where $y\theta = s\theta$, we have $\mathcal{I} \models \mathcal{F}(y\theta'' \not\approx y\theta)$, then $\mathcal{I} \models \mathcal{F}(y\theta \not\approx x\theta)$ because $y\theta'' = \tilde{t}'\theta$, $\mathcal{I} \models \mathcal{F}(\tilde{t}'\theta \approx \tilde{t}\theta)$, and $\tilde{t}\theta = x\theta$. Hence a literal of $\mathcal{F}(C\theta)$ is true in $\mathcal{I}$ and therefore $\mathcal{I} \models C\theta$.

Case 2.2 (Nonpurifying calculi): Since the variable condition does not hold, we have $C\theta \succeq C''\theta$, where $C'' = C\{x \mapsto \tilde{t}'\}$. We cannot have $C\theta = C''\theta$ because $x\theta = \tilde{t}\theta \neq \tilde{t}'\theta$ and x occurs in C. Hence, we have $C\theta \succ C''\theta$.

By the definition of $\mathcal{I}$, $C\theta \succ C''\theta$ implies $\mathcal{I} \models \mathcal{F}(C''\theta)$. We will use equalities that are true in $\mathcal{I}$ to rewrite $\mathcal{F}(C\theta)$ into $\mathcal{F}(C''\theta)$, which implies $\mathcal{I} \models \mathcal{F}(C\theta)$ by congruence.

By saturation of N, for any well-typed m-tuple of fresh variables $\bar{z}$, we can use a PosExt with premise D (if $n > m$) or ArgCong inference with premise D (if $n < m$) or using D itself (if $n = m$) to show that $\mathcal{G}(D' \vee \tilde{t}\,\bar{z} \approx \tilde{t}'\,\bar{z}) \subseteq \mathcal{G}(N) \cup GHRed_{\mathrm{C}}(\mathcal{G}(N))$. Hence, $D'\theta \vee \tilde{t}\theta\,\bar{u} \approx \tilde{t}'\theta\,\bar{u}$ is in $\mathcal{G}(N) \cup GHRed_{\mathrm{C}}(\mathcal{G}(N))$ for any ground arguments $\bar{u}$.

We observe that whenever $\tilde{t}\theta\,\bar{u}$ and $\tilde{t}'\theta\,\bar{u}$ are smaller than the maximal term of $C\theta$ for some arguments $\bar{u}$, we have

$$\mathcal{I} \models \mathcal{F}(\tilde{t}\theta\,\bar{u}) \approx \mathcal{F}(\tilde{t}'\theta\,\bar{u}) \tag{$\dagger$}$$

To show this, we assume that $\tilde{t}\theta\,\bar{u}$ and $\tilde{t}'\theta\,\bar{u}$ are smaller than the maximal term of $C\theta$ and we distinguish two cases: If $t\theta$ is smaller than the maximal term of $C\theta$, all terms in $D'\theta$ are smaller than the maximal term of $C\theta$ and hence $D'\theta \vee \tilde{t}\theta\,\bar{u} \approx \tilde{t}'\theta\,\bar{u} \prec C\theta$. If, on the other hand, $t\theta$ is equal to the maximal term of $C\theta$, $\tilde{t}\theta\,\bar{u}$ and $\tilde{t}'\theta\,\bar{u}$ are smaller than $t\theta$. Hence $\tilde{t}\theta\,\bar{u} \approx \tilde{t}'\theta\,\bar{u} \prec t\theta \approx t'\theta$ and $D'\theta \vee \tilde{t}\theta\,\bar{u} \approx \tilde{t}'\theta\,\bar{u} \prec D\theta \prec C\theta$. In both cases, since $\mathcal{F}(D'\theta)$ is false in $\mathcal{I}$ by assumption, $\mathcal{I} \models \mathcal{F}(\tilde{t}\theta\,\bar{u}) \approx \mathcal{F}(\tilde{t}'\theta\,\bar{u})$.

We proceed by a case distinction on whether $s\theta$ appears in a selected or in a maximal literal of $C\theta$. In both cases we provide an algorithm that establishes the equivalence of $C\theta$ and $C''\theta$ via rewriting using ($\dagger$). This might seem trivial at first sight, but we can only use the equations ($\dagger$) if $\tilde{t}\theta\,\bar{u}$ and $\tilde{t}'\theta\,\bar{u}$ are smaller than the maximal term of $C\theta$. Moreover, $\bar{u}$ might itself contain positions where we want to rewrite, so the order of rewriting matters.

Case 2.2.1: $s\theta$ is the maximal side of a maximal literal of $C\theta$. Then, since $C\theta \succ C''\theta$, every term in $C\theta$ and in $C''\theta$ is smaller than or equal to $s\theta$. Let C_0 and $\tilde{C}_0$ be the clauses resulting from rewriting $\mathcal{F}(t\theta) \longrightarrow \mathcal{F}(t'\theta)$ wherever possible in $\mathcal{F}(C\theta)$ and $\mathcal{F}(C''\theta)$, respectively. Since $\mathcal{F}(t\theta)$ is a subterm of $\mathcal{F}(s\theta)$, now every term in C_0 and $\tilde{C}_0$ is strictly smaller than $\mathcal{F}(s\theta)$.

We define $C_1, C_2, \ldots$ inductively as follows: Given C_i, choose a subterm of the form $\mathcal{F}(\tilde{t}\theta\,\bar{u})$ where $\tilde{t}\theta\,\bar{u} \succ \tilde{t}'\theta\,\bar{u}$ or of the form $\mathcal{F}(\tilde{t}'\theta\,\bar{u})$ where $\tilde{t}'\theta\,\bar{u} \succ \tilde{t}\theta\,\bar{u}$. Let C_{i+1} be the clause resulting from rewriting that subterm $\mathcal{F}(\tilde{t}\theta\,\bar{u})$ to $\mathcal{F}(\tilde{t}'\theta\,\bar{u})$ or that subterm $\mathcal{F}(\tilde{t}'\theta\,\bar{u})$ to $\mathcal{F}(\tilde{t}\theta\,\bar{u})$ in C_i, depending on which term was chosen. Analogously, we define $\tilde{C}_1, \tilde{C}_2, \ldots$ by applying the same algorithm to $\tilde{C}_0$. In both cases, the process terminates because $\succ$ is compatible with green contexts and well founded. Let C_* and $\tilde{C}_*$ be the respective final clauses.

The algorithm preserves the invariant that every term in C_i and $\tilde{C}_i$ is strictly smaller than $s\theta$. By congruence via ($\dagger$), applied at every step of the algorithm, we

know that C_* and $\mathcal{F}(C\theta)$ are equivalent in $\mathcal{I}$ and that $\tilde{C}_*$ and $\mathcal{F}(C''\theta)$ are equivalent in $\mathcal{I}$ as well.

We show that $C_* = \tilde{C}_*$. Assume that $C_* \neq \tilde{C}_*$. The algorithm preserves a second invariant, namely that $\mathcal{F}^{-1}(C_i)$ and $\mathcal{F}^{-1}(\tilde{C}_j)$ are equal except for positions where one contains $\tilde{t}\theta$ and the other one contains $\tilde{t}'\theta$. Consider a deepest position where $\mathcal{F}^{-1}(C_*)$ and $\mathcal{F}^{-1}(\tilde{C}_*)$ are different. The respective position in C_* and $\tilde{C}_*$ then contains $\mathcal{F}(\tilde{t}\theta\ \bar{u})$ and $\mathcal{F}(\tilde{t}'\theta\ \bar{u})$ or vice versa. The arguments $\bar{u}$ must be equal because we consider a deepest position. But then $\tilde{t}\theta\ \bar{u} \succ \tilde{t}'\theta\ \bar{u}$ or $\tilde{t}\theta\ \bar{u} \prec \tilde{t}'\theta\ \bar{u}$, which is impossible since the algorithm terminated in C_* and $\tilde{C}_*$. This shows that $C_* = \tilde{C}_*$. Hence $\mathcal{F}(C\theta)$ and $\mathcal{F}(C''\theta)$ are equivalent, which proves $\mathcal{I} \models \mathcal{F}(C\theta)$.

3

Case 2.2.2: $s\theta$ is the maximal side of a selected literal of $C\theta$. Then, by the selection restrictions, x cannot be the head of a maximal literal of C.

At every position where $x\ \bar{u}$ occurs in C with some (or no) arguments $\bar{u}$, we rewrite $(\tilde{t}\ \bar{u})\theta$ to $(\tilde{t}'\ \bar{u})\theta$ in $C\theta$ if $(\tilde{t}\ \bar{u})\theta \succ (\tilde{t}'\ \bar{u})\theta$. We start with the innermost occurrences of x, so that the order of the two terms at one step does not change by later rewriting. Analogously, at every position where $x\ \bar{u}$ occurs in C with some (or no) arguments $\bar{u}$, we rewrite $(\tilde{t}'\ \bar{u})\theta$ to $(\tilde{t}\ \bar{u})\theta$ in $C''\theta$ if $(\tilde{t}'\ \bar{u})\theta \succ (\tilde{t}\ \bar{u})\theta$, again starting with the innermost occurrences.

We never rewrite at the top level of the maximal term of $C\theta$ or $C''\theta$ because x cannot be the head of a maximal literal of C. The two resulting clauses are identical because $C\theta$ and $C''\theta$ only differ at positions where x occurs in C. The rewritten terms are all smaller than the maximal term of $C\theta$. With (†), this implies that $\mathcal{I} \models \mathcal{F}(C\theta)$ by congruence. □

With these properties of our inference systems in place, the saturation framework's lifting theorem (Theorem 3.38) guarantees static and dynamic refutational completeness of *HInf* w.r.t. $HRed_{\mathrm{I}}$. However, this theorem gives us refutational completeness w.r.t. the Herbrand entailment $\models_{\mathcal{G}}$, defined as $N_1 \models_{\mathcal{G}} N_2$ if $\mathcal{G}(N_1) \models \mathcal{G}(N_2)$, whereas our semantics is Tarski entailment $\models$, defined as $N_1 \models N_2$ if any model of N_1 is a model of N_2. The following lemma repairs this mismatch:

Lemma 3.43. *For $N \subseteq \mathcal{C}_{\mathrm{H}}$, we have $N \models_{\mathcal{G}} \bot$ if and only if $N \models \bot$.*

Proof. By Lemma 3.12, any model of N is also a model of $\mathcal{G}(N)$—i.e., $N \not\models \bot$ implies $N \not\models_{\mathcal{G}} \bot$. For the other direction, we need to show that $N \not\models_{\mathcal{G}} \bot$ implies $N \not\models \bot$. Assume that $N \not\models_{\mathcal{G}} \bot$—i.e., $\mathcal{G}(N) \not\models \bot$. Then there is a model $\mathcal{I}$ of $\mathcal{G}(N)$. We must show that there exists a model of N—i.e., $N \not\models \bot$. Let $\mathcal{I}'$ be an interpretation derived from $\mathcal{I}$ by removing all universes that are not the denotation of a type in $\mathcal{T}y_{\mathrm{GH}}$ and removing all domain elements that are not the denotation of a term in $\mathcal{T}_{\mathrm{GH}}$, making $\mathcal{I}'$ term-generated. Clearly, in our clausal logic, this leaves the denotations of terms and the truth of ground clauses unchanged. Thus, $\mathcal{I}' \models \mathcal{G}(N)$. We will show that $\mathcal{I}' \models N$. Let $C \in N$. We want to show that C is true in $\mathcal{I}'$ for all valuations ξ. Fix a valuation ξ. By construction, for each variable x, there exists a ground term s_x such that $[\![s_x]\!]_{\mathcal{I}'} = \xi(x)$. Let ρ be the substitution that maps every free variable x in C to s_x. Then $\xi(x) = [\![s_x]\!]_{\mathcal{I}'} = [\![x\rho]\!]_{\mathcal{I}'}$ for all x. By treating the type variables of C in the same way, we can also achieve that $\xi(\alpha) = [\![\alpha\rho]\!]_{\mathcal{I}'}$ for all x. By Lemma 3.11, $[\![t\rho]\!]_{\mathcal{I}'} = [\![t]\!]^{\xi}_{\mathcal{I}'}$ for all terms t and $[\![\tau\rho]\!]_{\mathcal{I}'} = [\![\tau]\!]^{\xi}_{\mathcal{I}'}$ for all types τ. Hence, $C\rho$ and C have

the same truth value in $\mathcal{I}'$ for ξ. Since $\mathcal{I}' \models \mathcal{G}(N)$, $C\rho$ is true in $\mathcal{I}'$ and thus C is true in $\mathcal{I}'$ as well. □

Theorem 3.44 (Static refutational completeness)**.** *The inference system HInf is statically refutationally complete w.r.t.* ($HRed_{\mathrm{I}}$,$HRed_{\mathrm{C}}$)*. That means, if $N \subseteq \mathcal{C}_{\mathrm{H}}$ is a clause set saturated w.r.t. HInf and $HRed_{\mathrm{I}}$, then we have $N \models \bot$ if and only if $\bot \in N$.*

Proof. We apply Theorem 3.38. By Theorem 3.34, $GHInf^{GHSel}$ is statically refutationally complete for all $GHSel \in \mathcal{G}(HSel)$. By Lemmas 3.40, 3.41, and 3.42, for every saturated $N \subseteq \mathcal{C}_{\mathrm{H}}$, there exists a selection function $GHSel \in \mathcal{G}(HSel)$ such that all inferences $\iota \in GHInf^{GHSel}$ with $prems(\iota) \in \mathcal{G}(N)$ either are $\mathcal{G}^{GHSel}$-ground instances of $HInf$-inferences from N or belong to $GHRed_{\mathrm{I}}^{GHSel}(\mathcal{G}(N))$.

Theorem 3.38 implies that if $N \subseteq \mathcal{C}_{\mathrm{H}}$ is a clause set saturated w.r.t. $HInf$ and $HRed_{\mathrm{I}}$, then $N \models_{\mathcal{G}} \bot$ if and only if $\bot \in N$. By Lemma 3.43, this also holds for the Tarski entailment $\models$. That is, if $N \subseteq \mathcal{C}_{\mathrm{H}}$ is a clause set saturated w.r.t. $HInf$ and $HRed_{\mathrm{I}}$, then $N \models \bot$ if and only if $\bot \in N$. □

From static refutational completeness, we can easily derive dynamic refutational completeness.

Theorem 3.45 (Dynamic refutational completeness)**.** *The inference system HInf is dynamically refutationally complete w.r.t.* ($HRed_{\mathrm{I}}$,$HRed_{\mathrm{C}}$)*, as defined in Definition 3.26.*

Proof. By Theorem 17 of Waldmann et al., this follows from Theorem 3.44 and Lemma 3.43. □

3.5. Implementation

We have implemented our four calculi in the Zipperposition prover. In previous work, Cruanes had already implemented a mode for first-order logic and a pragmatic higher-order mode with support for λ-abstractions and extensionality, but without any completeness guarantees.

The pragmatic higher-order mode provided a convenient basis to implement our calculi. It employs higher-order term and type representations and orders. Its ad hoc calculus extensions are similar to our calculi. They include an ArgCong-like rule and a PosExt-like rule, and Sup inferences are performed only at green subterms. One of the bugs we found during our implementation work occurred because argument positions shift when applied variables are instantiated. We resolved this by numbering argument positions in terms from right to left.

To implement the λ-free higher-order mode, we restricted the unification algorithm to non-λ-abstractions. To satisfy the requirements on selection, we avoid selecting literals that contain higher-order variables. To comply with our redundancy notion, we disabled rewriting of nongreen subterms. To improve term indexing of higher-order terms, we replaced the imperfect discrimination trees by fingerprint indices [118]. To speed up the computation of the Sup conditions, we omit the condition $C\sigma \not\preceq D\sigma$ in the implementation, at the cost of performing some additional inferences.

For the purifying calculi, we implemented purification as a simplification rule. This ensures that it is applied aggressively on all clauses, whether initial clauses from the problem or clauses produced during saturation, before any inferences are performed.

For the nonpurifying calculi, we added the possibility to perform SUP inferences at variable positions. This means that variables must be indexed as well. In addition, we modified the variable condition. Depending on the term ordering, it may be expensive or even impossible to decide whether there exists a grounding substitution θ with $t\sigma\theta \succ t'\sigma\theta$ and $C\sigma\theta \prec C''\sigma\theta$. We overapproximate the condition as follows: (1) check whether x appears with different arguments in the clause C; (2) use a term-order-specific algorithm to determine whether there might exist a grounding substitution θ and terms $\bar{u}$ such that $t\sigma\theta \succ t'\sigma\theta$ and $t\sigma\theta\,\bar{u} \prec t'\sigma\theta\,\bar{u}$; and (3) check whether $C\sigma \not\geq C''\sigma$. If these three conditions apply, we conclude that there might exist a ground substitution θ witnessing nonmonotonicity.

For the extensional calculi, we add axiom (EXT) to the clause set. To curb the explosion associated with extensionality, this axiom and all clauses derived from it are penalized by the clause selection heuristic. We also added the NEGEXT rule described in Section 3.3.2, which resembles Vampire's extensionality resolution rule [68].

The ARGCONG rule can have infinitely many conclusions on polymorphic clauses. To capture this in the implementation, we store these infinite sequences of conclusions in the form of finite instructions of how to obtain the actual clauses. In addition to the usual active and passive set of the DISCOUNT loop architecture [7], we use a set of scheduled inferences that stores these instructions. We visit the scheduled inferences in this additional set and the clauses in the passive set fairly to achieve dynamic completeness of our prover architecture. Waldmann et al. [140, Example 34] describe this architecture in more detail.

Using Zipperposition, we can quantify the disadvantage of the applicative encoding on Example 3.10. A well-chosen KBO instance with argument coefficients allows Zipperposition to derive $\bot$ in 4 iterations of the prover's main loop and 0.03 s. KBO or LPO with default settings needs 203 iterations and 0.4 s, whereas KBO or LPO on the applicatively encoded problem needs 203 iterations and more than 1 s due to the larger terms.

3.6. Evaluation

We evaluated Zipperposition's implementation of our calculi on Sledgehammer-generated Isabelle/HOL benchmarks [40] and on benchmarks from the TPTP library [131, 132]. Our experimental data is available online.[1]

The Sledgehammer benchmarks, corresponding to Isabelle's Judgment Day suite, were regenerated to target clausal λ-free higher-order logic. They comprise 2506 problems in total, divided in two groups based on the number of Isabelle facts (lemmas, definitions, etc.) selected for inclusion in each problem: either 16 facts (SH16) or 256 facts (SH256). The problems were generated by encoding λ-expressions

[1] https://doi.org/10.5281/zenodo.3992618

as λ-lifted supercombinators [106].

From the TPTP library, we collected 708 first-order problems in TFF format and 717 higher-order problems in THF format, both groups containing both monomorphic and polymorphic problems. We excluded all problems that contain interpreted arithmetic symbols, the symbols `(@@+)`, `(@@-)`, `(@+)`, `(@-)`, `(&)`, or tuples, as well as the `SYN000` problems, which are only intended to test the parser, and problems whose clausal normal form takes longer than 15 s to compute or falls outside the λ-free fragment described in Section 3.2.

We want to answer the following questions:

1. What is the overhead of our calculi on first-order benchmarks?
2. Do the calculi outperform the applicative encoding?
3. Do the purifying or the nonpurifying calculi achieve better results?
4. What is the cost of nonmonotonicity?

Since the calculus described in this chapter is only a stepping stone towards a prover for full higher-order logic, it would be misleading to compare this prototype with state-of-the-art higher-order provers that support a stronger logic. Many of the higher-order problems in the TPTP library are in fact satisfiable for our λ-free logic, even though they may be unsatisfiable for full higher-order logic and labeled as such in the TPTP.

To answer question (1) and (3), we ran Zipperposition's first-order mode on the first-order benchmarks and the purifying and nonpurifying modes on all benchmarks. To answer question (2), we implemented an applicative encoding mode in Zipperposition, which performs the applicative encoding after the clausal normal form transformation and then proceeds with Zipperposition's first-order mode. The encoding makes all function symbols nullary and replaces all applications with a polymorphic binary app symbol.

We instantiated all four calculi with three term orders: LPO [34], KBO [16] (without argument coefficients), and EPO. Among these, LPO is the only nonmonotonic order and therefore the most relevant option to evaluate our calculi, which are designed to cope with nonmonotonicity. To answer question (4), we also include the monotone orders KBO and EPO. EPO is an order designed to resemble LPO while fulfilling the requirements of a ground-total simplification order on λ-free terms. We will discuss it in more detail in Chapter 4. KBO and EPO serve as a yardstick to assess the cost of nonmonotonicity. With these monotone orders, no superposition inferences at variables are necessary and thus the nonpurifying calculi become a straightforward generalization of the standard superposition calculus with the caveat that it may be more efficient to superpose at nongreen subterms directly instead of relying on the ArgCong rule. On first-order benchmarks and in the applicative encoding mode, all three orders are monotone because they are monotone on first-order terms.

Figure 3.1 summarizes, for the intensional calculi, the number of solved satisfiable and unsatisfiable problems within 180 s, and the time taken to show unsatisfiability. Figure 3.2 presents the corresponding data for the extensional calculi. The average time is computed over the problems that all configurations for the respective benchmark set and term order found to be unsatisfiable within the time limit. For each combination of benchmark set and term order, the best result is highlighted

in bold. The evaluation was carried out on StarExec Iowa [127] using Intel Xeon E5-2609 0 CPUs clocked at 2.40 GHz.

The experimental results on the TFF part of the TPTP library confirm that our calculi handle the vast majority of problems that are solvable in first-order mode gracefully, and thus that the overhead is minimal, answering question (1). On first-order problems, the calculi are occasionally at variance with the first-order mode, due to the interaction of ARGCONG with polymorphic types and due to the extensionality axiom (EXT). In contrast, the applicative encoding is comparatively inefficient on problems that are already first-order. For LPO, the success rate decreases by around 15%, and the average time to show unsatisfiability triples.

The SH16 benchmarks consist mostly of small higher-order problems. The small number of axioms benefits the applicative encoding enough to outperform the purifying calculi but not the nonpurifying ones. The SH256 benchmarks are also higher-order but much larger. Such problems are underrepresented in the TPTP library. On these, our calculi clearly outperform the applicative encoding, answering question (2) decisively. This is hardly surprising given that the proving effort is dominated by first-order reasoning, which they can perform gracefully.

The THF benchmarks generally require more sophisticated higher-order reasoning than the Sledgehammer benchmarks, as observed by Sultana, Blanchette, and Paulson [128, Section 5]. On these benchmarks, the empirical results are less clear; the applicative encoding and our calculi are roughly neck-and-neck. The nonpurifying calculi detect unsatisfiability slightly more frequently, whereas the applicative encoding tends to find more saturations. It seems that, due to the large amount of higher-order reasoning necessary to solve TPTP problems, the advantage of our calculi on the first-order parts of the derivation is not a decisive factor on these benchmarks.

Concerning question (3), the nonpurifying calculi outperform their purifying relatives across all benchmarks. The raw data show that on most benchmark sets, the problems solved by the nonpurifying calculi are almost a superset of the problems solved by the purifying calculi. Only on the SH256 benchmarks, the purifying calculi can solve a fair number of problems that the nonpurifying calculi cannot solve (11 problems for the intensional calculi with LPO and 9 problems for the extensional calculi with LPO).

KBO tends to have a slight advantage over LPO on all benchmark sets. But the gap between KBO and LPO is not larger on the higher-order benchmarks than on TFF. Since LPO is monotonic on first-order terms but nonmonotonic on higher-order terms, whereas KBO is monotonic on both, the best answer we can give to question (4) is that no substantial cost seems to be associated with nonmonotonicity. In particular, for the nonpurifying calculi, the additional superposition inferences at variables necessary with LPO do not have a negative impact on the overall performance. This indicates that our approach is an excellent foundation for higher-order logics with λ-expressions, for which no suitable monotonic orders exist. EPO generally performs worse than the other two orders, with the exception of the nonpurifying calculus on SH16 benchmarks, where it is roughly neck-and-neck with LPO. This suggests that for small, mildly higher-order problems, EPO can be a viable LPO-like complement to KBO if one considers the effort to implement our calculi too high.

		# sat			# unsat			⌀ time		
		LPO	KBO	EPO	LPO	KBO	EPO	LPO	KBO	EPO
SH16	applicative encoding	111	**189**	65	373	382	157	0.9	1.2	10.7
	nonpurifying calculus	**136**	165	**133**	**383**	**385**	**381**	**0.4**	**0.3**	**0.0**
	purifying calculus	82	98	82	363	363	355	1.3	2.0	**0.0**
SH256	applicative encoding	**1**	**1**	**1**	471	488	36	9.4	8.7	63.8
	nonpurifying calculus	**1**	**1**	**1**	**543**	**554**	**498**	**2.3**	**2.3**	**0.1**
	purifying calculus	**1**	**1**	**1**	523	528	484	2.6	3.4	0.5
TFF	first-order mode	0	0	0	**212**	**229**	**107**	**1.9**	**2.3**	**1.5**
	applicative encoding	0	0	0	180	205	21	7.0	10.0	4.6
	nonpurifying calculus	0	0	0	210	**229**	105	**1.9**	2.4	**1.5**
	purifying calculus	0	0	0	211	**229**	105	2.1	2.6	1.6
THF	applicative encoding	**127**	**115**	111	523	522	428	0.9	0.6	0.8
	nonpurifying calculus	111	114	**112**	**529**	**527**	**516**	**0.3**	**0.3**	**0.0**
	purifying calculus	108	109	108	528	526	514	**0.3**	0.5	**0.0**

Figure 3.1: Evaluation of the intensional calculi

		# sat			# unsat			⌀ time		
		LPO	KBO	EPO	LPO	KBO	EPO	LPO	KBO	EPO
SH16	applicative encoding	79	**152**	48	379	386	157	1.2	1.3	11.4
	nonpurifying calculus	**103**	131	**95**	**386**	**393**	**387**	**0.4**	**0.1**	**0.0**
	purifying calculus	32	57	32	367	365	363	2.0	1.7	**0.0**
SH256	applicative encoding	**1**	**1**	**1**	462	486	36	7.5	9.4	63.8
	nonpurifying calculus	**1**	**1**	**1**	**548**	**572**	**504**	**1.9**	**2.1**	**0.1**
	purifying calculus	**1**	**1**	**1**	512	529	482	2.2	5.0	**0.1**
TFF	first-order mode	0	0	0	**212**	**229**	**107**	**1.9**	**2.5**	**1.5**
	applicative encoding	0	0	0	178	202	21	7.9	11.7	4.7
	nonpurifying calculus	0	0	0	207	**229**	106	2.1	3.0	**1.5**
	purifying calculus	0	0	0	210	**229**	105	2.2	3.2	1.6
THF	applicative encoding	**108**	**109**	105	526	527	436	0.9	0.6	1.1
	nonpurifying calculus	106	108	**107**	**539**	**535**	**526**	**0.3**	**0.3**	**0.0**
	purifying calculus	96	97	96	530	529	519	**0.3**	0.6	**0.0**

Figure 3.2: Evaluation of the extensional calculi

3.7. Discussion and Related Work

Our calculi join a long list of extensions and refinements of superposition. Among the most closely related is Peltier's [112] Isabelle/HOL formalization of the refutational completeness of a superposition calculus that operates on λ-free higher-order terms and that is parameterized by a monotonic term order. Extensions with polymorphism and induction, independently developed by Cruanes [46, 47] and Wand [141], contribute to increasing the power of automated provers. Detection of inconsistencies in axioms, as suggested by Schulz et al. [121], is important for large axiomatizations.

Also of interest is Bofill and Rubio's [39] integration of nonmonotonic orders in ordered paramodulation, a precursor of superposition. Their work is a veritable tour de force, but it is also highly complicated and restricted to ordered paramodulation. Lack of compatibility with arguments being a mild form of nonmonotonicity, it seemed preferable to start with superposition, enrich it with an ArgCong rule, and tune the side conditions until we obtained a complete calculus.

Most complications can be avoided by using a monotonic order such as KBO without argument coefficients. However, coefficients can be useful to help achieve compatibility with β-reduction. For example, the term $\lambda x.\, x + x$ could be treated as a constant with a coefficient of 2 on its argument and a heavy weight to ensure $(\lambda x.\, x + x)\; y \succ y + y$. Although they do not use argument coefficients, the recently developed combinatory superposition calculus by Bhayat and Reger [28] needs a nonmonotonic order to cope with β-reduction. Their approach is modeled after our intensional nonpurifying calculus.

In the term rewriting community, λ-free higher-order logic is known as applicative first-order logic. First-order rewrite techniques can be applied to this logic via the applicative encoding. However, there are similar drawbacks as in theorem proving to having app as the only nonnullary symbol. Hirokawa et al. [72] propose a technique that resembles our mapping $\mathcal{F}$ to avoid these drawbacks.

Another line of research has focused on the development of automated proof procedures for higher-order logic. We discussed such procedures already in Section 1.3.

3.8. Conclusion

We presented four superposition calculi for intensional and extensional clausal λ-free higher-order logic and proved them refutationally complete. The calculi nicely generalize standard superposition and are compatible with our λ-free higher-order LPO and KBO. Especially on large problems, our experiments confirm what one would naturally expect: that native support for partial application and applied variables outperforms the applicative encoding.

The new calculi reduce the gap between proof assistants based on higher-order logic and superposition provers. We can use them to reason about arbitrary higher-order problems by axiomatizing suitable combinators. But perhaps more importantly, many ideas developed in this chapter form the basis of our richer higher-order calculi.

4

The Embedding Path Order for Lambda-Free Higher-Order Terms

The embedding path order (EPO) is a variant of the recursive path order (RPO) for untyped λ-free higher-order terms (also called applicative first-order terms). Unlike other higher-order variants of RPO, it is a ground-total simplification order, making it suitable for superposition. This property makes it impossible to coincide with RPO on first-order terms, but EPO strongly resembles RPO in the principle to compare terms by their heads according to a symbol precedence. I formally proved the order's theoretical properties in Isabelle/HOL and evaluated the order using the Zipperposition prover.

4.1. Introduction

To restrict the search space, the superposition calculus uses a term order, which in practice is usually the Knuth–Bendix order (KBO) or the recursive path order (RPO). Extending the calculus to higher-order logic requires suitable higher-order term orders. The proof of refutational completeness of the first-order superposition calculus relies on a ground-total and well-founded simplification order. For a higher-order superposition calculus, a simplification order for higher-order terms modulo β-conversion would be desirable but does not exist, as demonstrated by $\mathsf{a} =_\beta (\lambda x.\ \mathsf{a})\ \mathsf{a} > \mathsf{a}$, where the inequality follows from the subterm property.

However, for λ-free higher-order terms, also known as applicative first-order terms, ground-total simplification orders do exist, for instance the generalization of KBO developed by Becker et al. [16] and the term order presented in this chapter. Such term orders can be put to use in superposition variants for λ-free higher-order logic, as implemented in Ehoh [139], and also for richer higher-order logics, as in recent work by Bhayat and Reger [28].

In contrast to KBO, a λ-free higher-order variant of RPO that is a simplification order and coincides with first-order RPO on the first-order fragment of lambda-free higher-order logic is impossible, which the following example shows: If $\mathsf{g} > \mathsf{f} > \mathsf{b} > \mathsf{a}$, then $\mathsf{g}\ \mathsf{b} > \mathsf{f}\ (\mathsf{g}\ \mathsf{a})\ \mathsf{b}$ by coincidence with first-order RPO, corresponding to $\mathsf{g}(\mathsf{b}) > \mathsf{f}(\mathsf{g}(\mathsf{a}), \mathsf{b})$ in first-order syntax, but $\mathsf{g} < \mathsf{f}\ (\mathsf{g}\ \mathsf{a})$ by the subterm property and hence $\mathsf{g}\ \mathsf{b} < \mathsf{f}\ (\mathsf{g}\ \mathsf{a})\ \mathsf{b}$ by compatibility with contexts, yielding a contradiction.

Does there exist a ground-total simplification order on λ-free higher-order terms that resembles RPO but does not coincide with RPO on first-order terms? One candidate is the applicative RPO, which is obtained by encoding λ-free higher-order terms applicatively into first-order logic via a binary symbol app representing application and then using first-order RPO. However, the symbol app becomes pervasive, which undermines RPO's principle of comparing the precedence of different symbols. Moreover, it is impossible to assign different extension orders such as the lexicographic or multiset extension to different function symbols because the only applied function symbol in the encoding is app.

The embedding path order (EPO[1]) presented here allows different extension operators for different function symbols (Section 4.3). The main difference to RPO lies in using the notion of embeddings where RPO uses the notion of direct subterms (Section 4.4). EPO is a ground-total simplification order and I have formally proved this property in Isabelle/HOL (Section 4.5). I illustrate the strengths and weaknesses of EPO on several examples (Section 4.6). I have implemented EPO in the superposition prover Zipperposition (Section 4.7) and evaluated it on TPTP [129] and Sledgehammer [111] benchmarks (Section 4.8).

In the literature, there are several other variants of RPO for higher-order terms. Blanchette et al.'s RPO for λ-free higher-order terms [34] resembles EPO the most, but it sacrifices compatibility with contexts to be able to coincide with first-order RPO. Superposition with such nonmonotonic orders is possible but compromises theoretical simplicity and to some degree practical efficiency, as demonstrated by Bofill and Rubio [39] and in the previous chapter. Although targeting the more difficult

[1] Beware that the unrelated exptime path order [55] has the same abbreviation.

problem of providing useful orders for full higher-order terms with λ-abstractions, the following RPO variants are vaguely related to the present work: Lifantsev and Bachmair's lexicographic path-order on λ-free higher-order terms [100], Jouannaud and Rubio's higher-order RPO (HORPO) [78], Kop and Van Raamsdonk's iterative HORPO [91], the HORPO extension with polynomial interpretation orders by Bofill et al. [38], and the computability path order by Blanqui et al. [36]. However, these orders all lack ground-totality and, except for Lifantsev and Bachmair's order, the subterm property for terms of different types.

4.2. Preliminaries

We fix a set of variables $\mathcal{V}$ and a nonempty (possibly infinite) set of symbols Σ. We reserve the names x, y, z for variables and $\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{f}, \mathsf{g}, \mathsf{h}$ for symbols.

In untyped λ-free higher-order logic, a term is defined inductively as being either a variable, a symbol, or an application $s\,t$, where s and t are terms.

We reserve the names t, s, v, u for terms and use $\mathcal{T}$ to denote the set of all terms. Application is left-associative, i.e., $s\,t\,u = (s\,t)\,u$. These terms are isomorphic to applicative terms [83]. Any term can be written as $\zeta\,\bar{t}_n$ using spine notation [44], where ζ is a nonapplication term, called *head*, and $\bar{t}_n$ is a tuple of terms, called *arguments*. It represents the term $\zeta\,t_1\,\ldots\,t_n$.

The *size* $|t|$ of a term t is inductively defined as 1 if $t \in \mathcal{V} \cup \Sigma$ and as $|t_1| + |t_2|$ if t is an application $t_1\,t_2$. A *subterm* of a term t is inductively defined as being either t itself or, if t is an application $t_1\,t_2$, a subterm of t_1 or of t_2. The *positions* of a term are tuples containing the elements left and right and are defined as follows: If the given term is a head, its only position is (); if the given term is an application $t\,s$, the tuple $\mathsf{left} \cdot p$ is a position of $t\,s$ for each position p of t and $\mathsf{right} \cdot q$ is a position of $t\,s$ for each position q of s.

The embedding step relation $\longrightarrow_{\mathsf{emb}}$ is inductively defined as follows: For any terms s and t, there is a left-embedding step $t\,s \longrightarrow_{\mathsf{emb}} t$ at position () and a right-embedding step $t\,s \longrightarrow_{\mathsf{emb}} s$ at position (). If for $X = \mathsf{left}$ or $X = \mathsf{right}$ there is an X-embedding step $t \longrightarrow_{\mathsf{emb}} t'$ at some position p, then there is an X-embedding step $t\,s \longrightarrow_{\mathsf{emb}} t'\,s$ at position $\mathsf{left} \cdot p$ and an X-embedding step $s\,t \longrightarrow_{\mathsf{emb}} s\,t'$ at position $\mathsf{right} \cdot p$. Let the embedding relation $\trianglerighteq_{\mathsf{emb}}$ be the reflexive transitive closure of $\longrightarrow_{\mathsf{emb}}$. For example, $\mathsf{f\,a\,b\,c\,d} \longrightarrow_{\mathsf{emb}} \mathsf{a\,b\,c\,d}$ is a right-embedding step at position $(\mathsf{left}, \mathsf{left}, \mathsf{left})$; $\mathsf{f\,a\,b\,c\,d} \longrightarrow_{\mathsf{emb}} \mathsf{f\,a\,c\,d}$ is a left-embedding step at position $(\mathsf{left}, \mathsf{left})$; and $\mathsf{f\,(g\,(h\,a)\,b)\,c} \longrightarrow_{\mathsf{emb}} \mathsf{f\,(g\,h\,b)\,c}$ is a left-embedding step at position $(\mathsf{left}, \mathsf{right}, \mathsf{left}, \mathsf{right})$.

For a term $\xi\,\bar{t}_n$ with $n > 0$, we define $\mathit{chop}(\xi\,\bar{t}_n)$ as the term resulting from applying t_1 to the remaining arguments, i.e., $\mathit{chop}(\xi\,\bar{t}_n) = t_1\,t_2\,\ldots\,t_n$. For example, $\mathit{chop}(\mathsf{f\,(g\,a)\,(h\,b)}) = \mathsf{g\,a\,(h\,b)}$.

4.3. Extension Operators

In the spirit of RPO, EPO compares the heads of terms and, in case of equality, proceeds to compare the argument tuples. There is a variety of ways to extend a binary relation $>$ on an arbitrary set A to a binary relation $\gg$ on tuples A^*, which we call extension operators. We define extension operators on binary relations, not

on partial orders, because they are used in the definition of EPO at a point where we have not shown EPO to be a partial order yet.

Definition 4.1. We define the following properties of extension operators $> \mapsto \gg$, which are required for EPO to be a ground-total and well-founded simplification order. Here, given a function $h : A \to A$, we write $h(\bar{a})$ for the componentwise application of h to $\bar{a}$.

- **X1. Monotonicity:** $\bar{b} \gg_1 \bar{a}$ implies $\bar{b} \gg_2 \bar{a}$ if for all $a, b \in A$, $b >_1 a$ implies $b >_2 a$
- **X2. Preservation of stability:** $\bar{b} \gg \bar{a}$ implies $h(\bar{b}) \gg h(\bar{a})$ if for all $a, b \in \bar{a} \cup \bar{b}$, $b > a$ implies $h(b) > h(a)$
- **X3. Preservation of transitivity:** $\gg$ is transitive if $>$ is transitive
- **X4. Preservation of irreflexivity:** $\gg$ is irreflexive if $>$ is irreflexive and transitive
- **X5. Preservation of well-foundedness:** $\gg$ is well founded if $>$ is well founded
- **X6. Compatibility with tuple contexts:** $b > a$ implies $\bar{c} \cdot b \cdot \bar{d} \gg \bar{c} \cdot a \cdot \bar{d}$
- **X7. Preservation of totality:** $\gg$ is total if $>$ is total
- **X8. Compatibility with prepending:** $\bar{b} \gg \bar{a}$ implies $c \cdot \bar{b} \gg c \cdot \bar{a}$
- **X9. Compatibility with appending:** $\bar{b} \gg \bar{a}$ implies $\bar{b} \cdot c \gg \bar{a} \cdot c$
- **X10. Minimality of the empty tuple:** $a \gg ()$ for all $a \in A$

The length-lexicographic extension operator, left-to-right or right-to-left, fulfills all these properties:

Definition 4.2. The *left-to-right length-lexicographic extension operator* $> \mapsto \gg^{\mathrm{ltr}}$ is defined inductively as follows: $\bar{a}_m \gg^{\mathrm{ltr}} \bar{b}_n$ if $m > n$; or $m = n > 0$ and $a_1 > b_1$; or $m = n > 0$, $a_1 = b_1$, and $(a_2, \ldots, a_m) \gg^{\mathrm{ltr}} (b_2, \ldots, b_n)$. The *right-to-left length-lexicographic extension operator* $> \mapsto \gg^{\mathrm{rtl}}$ is defined inductively as follows: $\bar{a}_m \gg^{\mathrm{rtl}} \bar{b}_n$ if $m > n$; or $m = n > 0$ and $a_m > b_n$; or $m = n > 0$, $a_m = b_n$, and $(a_1, \ldots, a_{m-1}) \gg^{\mathrm{rtl}} (b_1, \ldots, b_{n-1})$.

The multiset extension operator fulfills all properties except X7, but if combined with a lexicographic comparison as a tie-breaker, it fulfills all properties as well:

Definition 4.3. The *multiset extension operator with tie-breaker* $> \mapsto \gg^{\mathrm{ms}}$ is defined as follows: $\bar{a} \gg^{\mathrm{ms}} \bar{b}$ if the multiset containing the elements of $\bar{a}$ is larger than the multiset containing the elements of $\bar{b}$ with respect to Dershowitz and Manna's multiset order [50]; or if the two multisets are equal and $\bar{a} \gg^{\mathrm{ltr}} \bar{b}$.

Blanchette et al. [34] give a more detailed account of different extension operators. Their list of properties is identical with the one above, except for X2, which they originally formulated differently but corrected in their technical report [33].

4.4. The Order

Any simplification order has the embedding property, i.e., the property that $t \trianglerighteq_{\mathsf{emb}} s$ implies $t \succeq s$ [8, Lemma 5.4.7]. The fundamental idea of EPO is to enforce the embedding property by replacing the notion of subterms used in the definition of RPO by the notion of embeddings. Performed naively, this causes issues with stability under substitution and with the time complexity of the order computation due to

the large number of possible embedding steps. Both of these issues are addressed by EPO.

Definition 4.4 (EPO)**.** Let $\succ$ be a well-founded total order on Σ. For each $\mathsf{f} \in \Sigma$, let $\succ \mapsto \gg^{\mathsf{f}}$ be an extension operator satisfying the properties of Definition 4.1. The induced *embedding path order* $>_{\mathsf{ep}}$ is inductively defined as follows: $t >_{\mathsf{ep}} s$ if any of the following conditions is met, where $t = \xi\, \bar{t}_n$ and $s = \zeta\, \bar{s}_m$:

E1. $n > 0$ and $\mathit{chop}(t) \geq_{\mathsf{ep}} s$
E2. $\xi, \zeta \in \Sigma$, $\xi \succ \zeta$, and we have either $m = 0$ or $t >_{\mathsf{ep}} \mathit{chop}(s)$
E3. $\xi, \zeta \in \Sigma$, $\xi = \zeta$, $\bar{t}_n \gg^{\zeta}_{\mathsf{ep}} \bar{s}_m$, and we have either $m = 0$ or $t >_{\mathsf{ep}} \mathit{chop}(s)$
E4. $\xi, \zeta \in \mathcal{V}$, $\xi = \zeta$, $\bar{t}_n \gg^{\mathsf{f}}_{\mathsf{ep}} \bar{s}_m$ for all $\mathsf{f} \in \Sigma$, $n > 0$, and we have either $m = 0$ or $\mathit{chop}(t) >_{\mathsf{ep}} \mathit{chop}(s)$

The following examples illustrate the differences between RPO and EPO on first-order terms. We use the precedence $\mathsf{g} \succ \mathsf{f} \succ \mathsf{c} \succ \mathsf{b} \succ \mathsf{a}$ and the left-to-right length-lexicographic extension for both orders.

$$\begin{array}{lll} \mathsf{f}\,(\mathsf{g}\,\mathsf{a})\,\mathsf{b} <_{\mathsf{rp}} \mathsf{g}\,\mathsf{b} & \mathsf{f}\,(\mathsf{g}\,\mathsf{a})\,\mathsf{c} <_{\mathsf{rp}} \mathsf{g}\,\mathsf{b} & \mathsf{g}\,x\,y >_{\mathsf{rp}} \mathsf{f}\,y\,y \\ \mathsf{f}\,(\mathsf{g}\,\mathsf{a})\,\mathsf{b} >_{\mathsf{ep}} \mathsf{g}\,\mathsf{b} & \mathsf{f}\,(\mathsf{g}\,\mathsf{a})\,\mathsf{c} >_{\mathsf{ep}} \mathsf{g}\,\mathsf{b} & \mathsf{g}\,x\,y \not>_{\mathsf{ep}} \mathsf{f}\,y\,y \end{array}$$

The first term pair illustrates that RPO does not have the embedding property, whereas EPO does. The relation $\mathsf{f}\,(\mathsf{g}\,\mathsf{a})\,\mathsf{b} >_{\mathsf{ep}} \mathsf{g}\,\mathsf{b}$ can be shown by applying E1. E1 requires $\mathsf{g}\,\mathsf{a}\,\mathsf{b} >_{\mathsf{ep}} \mathsf{g}\,\mathsf{b}$, which holds by E3. Finally we need E2 to show $\mathsf{g}\,\mathsf{a}\,\mathsf{b} >_{\mathsf{ep}} \mathsf{b}$. The second term pair shows that there are further disagreements between the two orders, even if one term is not an embedding of the other. As above, $\mathsf{f}\,(\mathsf{g}\,\mathsf{a})\,\mathsf{c} >_{\mathsf{ep}} \mathsf{g}\,\mathsf{b}$ can be established by applying E1, followed by E3 and E2. The third term pair is comparable with RPO but incomparable with EPO. In general, EPO cannot judge a term to be smaller if it contains more occurrences of a variable. I conjecture that there are no first-order terms comparable with EPO but incomparable with RPO. In this sense, EPO is weaker than RPO on first-order terms.

The remainder of this section justifies some of the design decisions in the definition of EPO and explains how they contribute to make EPO a ground-total simplification order that can be computed fairly efficiently.

Condition E1 enforces the embedding property in a similar way as RPO enforces the subterm property. This underlying idea gives EPO its name. A naive approach would be to test all embedding steps to enforce the embedding property, but it is sufficient to test only the embedding step *chop*, yielding a better computational complexity. The remaining conditions follow a similar structure as RPO, but contain subconditions on *chop* where RPO has subconditions on subterms.

To achieve stability under substitutions, it is essential to demand $\mathit{chop}(t) >_{\mathsf{ep}} \mathit{chop}(s)$ instead of $t >_{\mathsf{ep}} \mathit{chop}(s)$ in E4, as the following examples show. If $>'_{\mathsf{ep}}$ is the relation obtained from $>_{\mathsf{ep}}$ by replacing '$\mathit{chop}(t)$' by 't' in E4, then we have

$$x\,\mathsf{f}\,\mathsf{f} >'_{\mathsf{ep}} x\,x, \text{ but } \mathsf{f}\,y\,\mathsf{f}\,\mathsf{f} \not>'_{\mathsf{ep}} \mathsf{f}\,y\,(\mathsf{f}\,y)$$
$$x\,\mathsf{f}\,x >'_{\mathsf{ep}} x\,(x\,\mathsf{f}), \text{ but } y\,\mathsf{f}\,\mathsf{f}\,(y\,\mathsf{f}) \not>'_{\mathsf{ep}} y\,\mathsf{f}\,(y\,\mathsf{f}\,\mathsf{f})$$

Using $>_{\mathsf{ep}}$, all of these pairs are incomparable.

In condition E4, it is crucial to check $\bar{t}_n \gg^{\mathsf{f}}_{\mathsf{ep}} \bar{s}_m$ for all $\mathsf{f} \in \Sigma$. In contrast, λ-free KBO [16] and λ-free RPO [34] allow us to use a map ghd from variables to possible ground heads that might occur when a variable is instantiated. The corresponding condition in these orders then states '$\bar{t}_n \gg^{\mathsf{f}}_{\mathsf{ep}} \bar{s}_m$ for all $\mathsf{f} \in \mathit{ghd}(\zeta)$'. For EPO, this approach cannot be used. For example, assume $\mathsf{b} \succ \mathsf{a}$, $\mathit{ghd}(x) = \{\mathsf{f}\}$, and that f uses the left-to-right length-lexicographic extension. Then we would have $x\ \mathsf{b}\ \mathsf{a} > x\ \mathsf{a}\ \mathsf{b}$ if we checked only the extension orders for $\mathit{ghd}(x)$. This contradicts stability under substitutions because, if g uses the right-to-left length-lexicographic extension, $y\ \mathsf{g}\ \mathsf{b}\ \mathsf{a}$ and $y\ \mathsf{g}\ \mathsf{a}\ \mathsf{b}$ are incomparable, assuming $\mathit{ghd}(y) = \{\mathsf{f}\}$.

EPO is not a simplification order when (nonlength-)lexicographic extensions are used. With the left-to-right lexicographic extension, it lacks compatibility with contexts because for $\mathsf{g} \succ \mathsf{f} \succ \mathsf{b} \succ \mathsf{a}$, we have $\mathsf{f}\ (\mathsf{g}\ \mathsf{a}) >_{\mathsf{ep}} \mathsf{g}$ but $\mathsf{f}\ (\mathsf{g}\ \mathsf{a})\ \mathsf{b} <_{\mathsf{ep}} \mathsf{g}\ \mathsf{b}$. With the right-to-left lexicographic extension, it lacks stability under substitutions because $x\ \mathsf{f} > x$ but $\mathsf{f}\ y\ \mathsf{f} \not\succ \mathsf{f}\ y$. With the right-to-left lexicographic extension, it also lacks well-foundedness because for $\mathsf{f} \succ \mathsf{b} \succ \mathsf{a}$, we have $\mathsf{f}\ \mathsf{b} >_{\mathsf{ep}} \mathsf{f}\ \mathsf{b}\ \mathsf{a} >_{\mathsf{ep}} \mathsf{f}\ \mathsf{b}\ \mathsf{a}\ \mathsf{a} >_{\mathsf{ep}} \cdots$.

4.5. Properties of the Order

EPO fulfills all the properties of a ground-total simplification order. The proofs in this section have been developed in Isabelle/HOL and published in the Archive of Formal Proofs [18]. They are inspired by the corresponding proofs about λ-free RPO [34], which in turn were adapted from Baader and Nipkow [8] and Zantema [143].

Theorem 4.5 (Transitivity)**.** *$u >_{\mathsf{ep}} t$ and $t >_{\mathsf{ep}} s$ implies $u >_{\mathsf{ep}} s$.*

Proof. By well-founded induction on the multiset $\{|u|, |t|, |s|\}$ with respect to the multiset extension [50] of $>$ on $\mathbb{N}$. Let $u = \psi\ \bar{u}_r$, $t = \xi\ \bar{t}_n$ and $s = \zeta\ \bar{s}_m$.

If $u >_{\mathsf{ep}} t$ is derived by E1, then $r > 0$ and $\mathit{chop}(u) \geq_{\mathsf{ep}} t$. Applying the induction hypothesis to $\mathit{chop}(u)$, t, s, it follows that $\mathit{chop}(u) >_{\mathsf{ep}} s$ and hence $u >_{\mathsf{ep}} s$ by E1.

If $u >_{\mathsf{ep}} t$ is derived by E2 or E3 and $t >_{\mathsf{ep}} s$ is derived by E1, then $n > 0$ and $u >_{\mathsf{ep}} \mathit{chop}(t) \geq_{\mathsf{ep}} s$. Applying the induction hypothesis to u, $\mathit{chop}(t)$, s, it follows that $u >_{\mathsf{ep}} s$.

If $u >_{\mathsf{ep}} t$ is derived by E4 and $t >_{\mathsf{ep}} s$ is derived by E1, then $r > 0$, $n > 0$, and $\mathit{chop}(u) >_{\mathsf{ep}} \mathit{chop}(t) \geq_{\mathsf{ep}} s$. By applying the induction hypothesis to $\mathit{chop}(u)$, $\mathit{chop}(t)$, s, we get $\mathit{chop}(u) >_{\mathsf{ep}} s$. By E1, it follows that $u >_{\mathsf{ep}} s$.

If $u >_{\mathsf{ep}} t$ and $t >_{\mathsf{ep}} s$ are derived by E2 and E2, by E2 and E3, or by E3 and E2, respectively, then $\psi \succ \zeta$ and $t >_{\mathsf{ep}} \mathit{chop}(s)$. If $m = 0$, we can apply E2 directly to obtain $u >_{\mathsf{ep}} s$. If $m > 0$, by the induction hypothesis for u, t, $\mathit{chop}(s)$, it follows from $u >_{\mathsf{ep}} t$ and $t >_{\mathsf{ep}} \mathit{chop}(s)$ that $u >_{\mathsf{ep}} \mathit{chop}(s)$. Then we can apply E2 to obtain $u >_{\mathsf{ep}} s$.

If $u >_{\mathsf{ep}} t$ and $t >_{\mathsf{ep}} s$ are both derived by E3, then $\psi = \xi = \zeta \in \Sigma$, $\bar{u} \gg^{\xi}_{\mathsf{ep}} \bar{t}$, $\bar{t} \gg^{\zeta}_{\mathsf{ep}} \bar{s}$, and either $m = 0$ or $t >_{\mathsf{ep}} \mathit{chop}(s)$. By the induction hypothesis and by preservation of transitivity (property X3) on the set consisting of the elements of $\bar{u}$, $\bar{t}$ and $\bar{s}$, it follows that $\bar{u} \gg^{\zeta}_{\mathsf{ep}} \bar{s}$. If $m = 0$, we obtain $u >_{\mathsf{ep}} s$ directly by E3. If $m > 0$, we have $t >_{\mathsf{ep}} \mathit{chop}(s)$ and by applying the induction hypothesis to u, t, $\mathit{chop}(s)$, it follows that $u >_{\mathsf{ep}} \mathit{chop}(s)$. By E3, we have $u >_{\mathsf{ep}} s$.

If $u >_{\mathsf{ep}} t$ and $t >_{\mathsf{ep}} s$ are both derived by E4, then $\psi = \xi = \zeta \in \Sigma$, $\bar{u} \gg^{\mathsf{f}}_{\mathsf{ep}} \bar{t}$, $\bar{t} \gg^{\mathsf{f}}_{\mathsf{ep}} \bar{s}$ for all $\mathsf{f} \in \Sigma$, $r > 0$, $n > 0$, $\mathit{chop}(u) >_{\mathsf{ep}} \mathit{chop}(t)$, and either $m = 0$ or $\mathit{chop}(t) >_{\mathsf{ep}} \mathit{chop}(s)$.

As above, by the induction hypothesis and by preservation of transitivity (property X3) on the set consisting of the elements of $\bar{u}$, $\bar{t}$ and $\bar{s}$, it follows that $\bar{u} \gg^{\mathsf{f}}_{\mathsf{ep}} \bar{s}$ for all $\mathsf{f} \in \Sigma$. If $m = 0$, we obtain $u >_{\mathsf{ep}} s$ directly by E4. If $m > 0$, we have $\mathit{chop}(u) >_{\mathsf{ep}} \mathit{chop}(t) >_{\mathsf{ep}} \mathit{chop}(s)$. By applying the induction hypothesis to $\mathit{chop}(u)$, $\mathit{chop}(t)$, $\mathit{chop}(s)$, it follows that $\mathit{chop}(u) >_{\mathsf{ep}} \mathit{chop}(s)$. By E4, we have $u >_{\mathsf{ep}} s$.

If one of the inequalities $u >_{\mathsf{ep}} t$ and $t >_{\mathsf{ep}} s$ is derived by E2 or E3, the other cannot be derived by E4 because ξ must be either a variable or a symbol. □

Theorem 4.6 (Irreflexivity)**.** *$s \not>_{\mathsf{ep}} s$.*

Proof. By strong induction on $|s|$. We suppose that $s >_{\mathsf{ep}} s$ and derive a contradiction. Let $s = \zeta\ \bar{s}_m$.

If $s >_{\mathsf{ep}} s$ is derived by E1, then $m > 0$ and $\mathit{chop}(s) \geq_{\mathsf{ep}} s$. From the definition of *chop*, it is clear that $\mathit{chop}(s) \neq s$. Hence, $\mathit{chop}(s) >_{\mathsf{ep}} s$. By E1, we have $s >_{\mathsf{ep}} \mathit{chop}(s)$. By transitivity (Theorem 4.5), it follows that $\mathit{chop}(s) >_{\mathsf{ep}} \mathit{chop}(s)$, which contradicts the induction hypothesis.

If $s >_{\mathsf{ep}} s$ is derived by E2, we have $\zeta > \zeta$, in contradiction to $>$ being a total order.

If $s >_{\mathsf{ep}} s$ is derived by E3 or E4, we have $\bar{s} \gg^{\mathsf{f}}_{\mathsf{ep}} \bar{s}$ for some $\mathsf{f} \in \Sigma$. By preservation of irreflexivity (property X4) on the set consisting of the elements of $\bar{s}$ and by transitivity of $>_{\mathsf{ep}}$ (Theorem 4.5), it follows that $s' >_{\mathsf{ep}} s'$ for some $s' \in \bar{s}$. This contradicts the induction hypothesis. □

Lemma 4.7 (Embedding Step Property)**.** *$t \longrightarrow_{\mathsf{emb}} s$ implies $t >_{\mathsf{ep}} s$.*

Proof. By strong induction on $|s| + |t|$. Let $s = \zeta\ \bar{s}_m$ and $t = \xi\ \bar{t}_n$. We distinguish the following three cases, depending on the position of the embedding $t \longrightarrow_{\mathsf{emb}} s$.

- *Generalized Chop:* right-embedding at a position left^j for some $0 \leq j < n$, i.e., $s = t_i\ t_{i+1}\ \ldots\ t_n$ where $i = n - j$.
- *Remove Argument:* left-embedding at a position left^j for some $0 \leq j < n$, i.e., $s = \xi\ t_1\ \ldots\ t_{i-1}\ t_{i+1}\ \ldots\ t_n$ where $i = n - j$.
- *Reduce Argument:* embeddings at other positions, i.e., $s = \xi\ t_1\ \ldots\ t_{i-1}\ u\ t_{i+1}\ \ldots\ t_n$ for some i and some u such that $t_i \longrightarrow_{\mathsf{emb}} u$.

Case 1 (Generalized Chop): Then $s = t_i\ t_{i+1}\ \ldots\ t_n$ for some i. If $i = 1$, then $s = \mathit{chop}(t)$, which implies $t >_{\mathsf{ep}} s$ by E1. If $i > 1$, there is a right-embedding at position left^{n-i} from $\mathit{chop}(t)$ to s. By the induction hypothesis, $\mathit{chop}(t) >_{\mathsf{ep}} s$ and hence $t >_{\mathsf{ep}} s$ by E1.

Case 2 (Remove Argument): Then $\zeta = \xi$ and $\bar{s}_m = t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n$. Depending on whether $\zeta \in \mathcal{V}$ or $\zeta \in \Sigma$, we will use E3 or E4 to show $t >_{\mathsf{ep}} s$. To apply either condition, we show $\bar{t}_n \gg^{\mathsf{f}}_{\mathsf{ep}} \bar{s}_m$ for all symbols $\mathsf{f} \in \Sigma$. Since $\bar{s}_m = t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n$, this follows from properties X8, X9, and X10. If $m = 0$, we can apply E3 or E4 directly to obtain $t >_{\mathsf{ep}} s$.

Otherwise, both m and n are nonzero and we show that $\mathit{chop}(t) \longrightarrow_{\mathsf{emb}} \mathit{chop}(s)$ by a case distinction on whether $i = 1$ or $i > 1$ as follows. If $i = 1$, then $\mathit{chop}(t) = t_1\ t_2\ \ldots\ t_n$ and $\mathit{chop}(s) = t_2\ t_3\ \ldots\ t_n$. Hence, there is a right-embedding step at position left^{n-2} from $\mathit{chop}(t)$ to $\mathit{chop}(s)$. If $i > 1$, then $\mathit{chop}(t) = t_1\ t_2\ \ldots\ t_n$ and $\mathit{chop}(s) = t_1\ t_2\ \ldots\ t_{i-1}\ t_{i+1}\ \ldots\ t_n$. Hence, there is a left-embedding step at position left^{n-i} from $\mathit{chop}(t)$ to $\mathit{chop}(s)$.

By the induction hypothesis, $\mathit{chop}(t) >_{\mathsf{ep}} \mathit{chop}(s)$. If $\zeta \in \mathcal{V}$, we can then apply E4 to obtain $t >_{\mathsf{ep}} s$. Otherwise, $\zeta \in \Sigma$, and we apply E1 to obtain $t >_{\mathsf{ep}} \mathit{chop}(s)$ and E3 to obtain $t >_{\mathsf{ep}} s$.

CASE 3 (REDUCE ARGUMENT): Then $\zeta = \xi$ and $\bar{s} = t_1, \ldots, t_{i-1}, u, t_{i+1}, \ldots, t_n$ for some i and some u such that $t_i \longrightarrow_{\mathsf{emb}} u$. As above, depending on whether $\zeta \in \mathcal{V}$ or $\zeta \in \Sigma$, we will use E3 or E4 to show $t >_{\mathsf{ep}} s$.

To apply either condition, we show $\bar{t}_n \gg^{\mathsf{f}}_{\mathsf{ep}} \bar{s}_m$ for all symbols $\mathsf{f} \in \Sigma$. By the induction hypothesis, $t_i \longrightarrow_{\mathsf{emb}} u$ implies $t_i >_{\mathsf{ep}} u$. By property X6, we have $\bar{t}_n \gg^{\mathsf{f}}_{\mathsf{ep}} \bar{s}_m$ for all $\mathsf{f} \in \Sigma$.

Both m and n are nonzero because $0 < i \leq m = n$. We observe that $\mathit{chop}(t) \longrightarrow_{\mathsf{emb}} \mathit{chop}(s)$ because the only difference between $\mathit{chop}(t)$ and $\mathit{chop}(s)$ is that $\mathit{chop}(t)$ has the subterm t_i where $\mathit{chop}(s)$ has the subterm u and we have $t_i \longrightarrow_{\mathsf{emb}} u$. By the induction hypothesis, it follows that $\mathit{chop}(t) >_{\mathsf{ep}} \mathit{chop}(s)$. If $\zeta \in \mathcal{V}$, we apply E4 to obtain $t >_{\mathsf{ep}} s$. Otherwise, $\zeta \in \Sigma$, and we apply E1 to obtain $t >_{\mathsf{ep}} \mathit{chop}(s)$ and E3 to obtain $t >_{\mathsf{ep}} s$. □

Theorem 4.8 (Embedding Property)**.** *$t \trianglerighteq_{\mathsf{emb}} s$ implies $t \geq_{\mathsf{ep}} s$.*

Proof. Follows by induction on $t \trianglerighteq_{\mathsf{emb}} s$ from Lemma 4.7 and Theorem 4.5. □

Theorem 4.9 (Subterm Property)**.** *For all subterms s of a term t, we have $t \geq_{\mathsf{ep}} s$.*

Proof. Follows directly from Theorem 4.8. □

Lemma 4.10 (Compatibility with Functions)**.** *If $v >_{\mathsf{ep}} u$, then $s\, v >_{\mathsf{ep}} s\, u$.*

Proof. By induction on $|s|$.

Let $s = \zeta\, \bar{s}$. Depending on whether $\zeta \in \Sigma$ or $\zeta \in \mathcal{V}$, we show $s\, v >_{\mathsf{ep}} s\, u$ by applying E3 or E4. By compatibility with tuple contexts (property X6), $v >_{\mathsf{ep}} u$ implies $\bar{s} \cdot v \gg^{\mathsf{f}}_{\mathsf{ep}} \bar{s} \cdot u$ for all $\mathsf{f} \in \Sigma$. Obviously, the tuples $\bar{s} \cdot v$ and $\bar{s} \cdot u$ are not empty. So it remains to show $s\, v >_{\mathsf{ep}} \mathit{chop}(s\, u)$ if $\zeta \in \Sigma$ or $\mathit{chop}(s\, v) >_{\mathsf{ep}} \mathit{chop}(s\, u)$ if $\zeta \in \mathcal{V}$. By E1, it suffices to show $\mathit{chop}(s\, v) >_{\mathsf{ep}} \mathit{chop}(s\, u)$ in both cases.

If $\bar{s} = ()$, then $\mathit{chop}(s\, v) = v >_{\mathsf{ep}} u = \mathit{chop}(s\, u)$ by assumption. Otherwise, $\mathit{chop}(s\, v) = \mathit{chop}(s)\, v >_{\mathsf{ep}} \mathit{chop}(s)\, u = \mathit{chop}(s\, u)$ by the induction hypothesis. □

Lemma 4.11. *If $t >_{\mathsf{ep}} s$ and $v \geq_{\mathsf{ep}} u$, then $t\, v >_{\mathsf{ep}} s\, u$.*

Proof. By induction on $|t| + |s|$ and a case distinction on how $t >_{\mathsf{ep}} s$ is derived. Let $t = \xi\, \bar{t}_n$ and $s = \zeta\, \bar{s}_m$.

If $t >_{\mathsf{ep}} s$ is derived by E1, then $\mathit{chop}(t) \geq_{\mathsf{ep}} s$. By E1, $t\, v >_{\mathsf{ep}} \mathit{chop}(t\, v) = \mathit{chop}(t)\, v$. So it suffices to show $\mathit{chop}(t)\, v \geq_{\mathsf{ep}} s\, u$. If $\mathit{chop}(t) = s$, this follows from Lemma 4.10. Otherwise, we have $\mathit{chop}(t) >_{\mathsf{ep}} s$ and hence $\mathit{chop}(t)\, v >_{\mathsf{ep}} s\, u$ holds by the induction hypothesis.

If $t >_{\mathsf{ep}} s$ is derived by E2, then $\xi > \zeta$ and either $m = 0$ or $t >_{\mathsf{ep}} \mathit{chop}(s)$. To derive $t\, v >_{\mathsf{ep}} s\, u$ using E2, it remains to show $t\, v >_{\mathsf{ep}} \mathit{chop}(s\, u)$. If $m = 0$, then $\mathit{chop}(s\, u) = u$. Therefore, by the subterm property (Theorem 4.9), $t\, v >_{\mathsf{ep}} v \geq_{\mathsf{ep}} u = \mathit{chop}(s\, u)$. If $m > 0$, then $t >_{\mathsf{ep}} \mathit{chop}(s)$, and hence by the induction hypothesis, $t\, v >_{\mathsf{ep}} \mathit{chop}(s)\, u = \mathit{chop}(s\, u)$.

If $t >_{\mathsf{ep}} s$ is derived by E3 or E4, we need to show that $\bar{t}_n \gg^{\mathsf{f}}_{\mathsf{ep}} \bar{s}_m$ implies $\bar{t}_n \cdot v \gg^{\mathsf{f}}_{\mathsf{ep}} \bar{s}_m \cdot u$ for all $\mathsf{f} \in \Sigma$. We have $\bar{t}_n \cdot v \gg^{\mathsf{f}}_{\mathsf{ep}} \bar{s}_m \cdot v$ by compatibility with appending (property X9). If $v = u$, we are done. Otherwise, since $\bar{s}_m \cdot v \gg^{\mathsf{f}}_{\mathsf{ep}} \bar{s}_m \cdot u$ by compatibility with tuple contexts (property X6), it follows that $\bar{t}_n \cdot v \gg^{\mathsf{f}}_{\mathsf{ep}} \bar{s}_m \cdot u$ by preservation of transitivity (property X3) and transitivity of $>_{\mathsf{ep}}$ (Theorem 4.5).

If $t >_{\mathsf{ep}} s$ is derived by E3, we can apply E3 to derive $t\, v >_{\mathsf{ep}} s\, u$. The condition $t\, v >_{\mathsf{ep}} \mathit{chop}(s\, u)$ can be shown as we did for E2 above.

If $t >_{\mathsf{ep}} s$ is derived by E4, we can apply E4 to derive $t\, v >_{\mathsf{ep}} s\, u$. The proof for the condition $\mathit{chop}(t\, v) >_{\mathsf{ep}} \mathit{chop}(s\, u)$ is similar to the argument made for E2 above. □

Theorem 4.12 (Compatibility with Contexts)**.** *If $t >_{\mathsf{ep}} s$, then $u\,(t\,\bar{v}) >_{\mathsf{ep}} u\,(s\,\bar{v})$.*

Proof. By repeatedly applying Lemma 4.11 and finally Lemma 4.10. □

Theorem 4.13 (Stability under Substitutions)**.** *If $t >_{\mathsf{ep}} s$, then $t\sigma >_{\mathsf{ep}} s\sigma$.*

Proof. By well-founded induction on the multiset $\{|t|, |s|\}$ with respect to the multiset extension [50] of $>$ on $\mathbb{N}$, followed by a case distinction on how $t >_{\mathsf{ep}} s$ is derived. Let $t = \xi\,\bar{t}_n$ and $s = \zeta\,\bar{s}_m$.

If $t >_{\mathsf{ep}} s$ is derived by E1, then $\mathit{chop}(t) \geq_{\mathsf{ep}} s$. By the induction hypothesis, $\mathit{chop}(t)\sigma \geq_{\mathsf{ep}} s\sigma$. Since $t\sigma \longrightarrow_{\mathsf{emb}} \mathit{chop}(t)\sigma$, we have $t\sigma >_{\mathsf{ep}} \mathit{chop}(t)\sigma$ by the embedding property (Theorem 4.8). Hence, by transitivity $t\sigma >_{\mathsf{ep}} s\sigma$.

If $t >_{\mathsf{ep}} s$ is derived by E2, then $\xi, \zeta \in \Sigma$, $\xi > \zeta$, and either $m = 0$ or $t >_{\mathsf{ep}} \mathit{chop}(s)$. We show $t\sigma >_{\mathsf{ep}} s\sigma$ by applying E2. Since $\xi, \zeta \in \Sigma$, the head of $t\sigma$ is ξ, the head of $s\sigma$ is ζ, and the number of arguments of $s\sigma$ is m. Hence, it only remains to show that $t >_{\mathsf{ep}} \mathit{chop}(s)$ implies $t\sigma >_{\mathsf{ep}} \mathit{chop}(s\sigma)$, which follows from the induction hypothesis and from $\mathit{chop}(s)\sigma = \mathit{chop}(s\sigma)$.

If $t >_{\mathsf{ep}} s$ is derived by E3, then $\xi = \zeta \in \Sigma$, $\bar{t}_n \gg^{\zeta}_{\mathsf{ep}} \bar{s}_m$, and either $m = 0$ or $t >_{\mathsf{ep}} \mathit{chop}(s)$. Since $\xi, \zeta \in \Sigma$, the head of $t\sigma$ is ξ, the head of $s\sigma$ is ζ, and $\bar{t}_n\sigma$ and $\bar{s}_m\sigma$ are the respective argument tuples of $t\sigma$ and $s\sigma$. By the induction hypothesis and preservation of stability (property X2) on the set of elements of $\bar{t}_n$ and $\bar{s}_m$, we have $\bar{t}_n\sigma \gg^{\zeta}_{\mathsf{ep}} \bar{s}_m\sigma$. We apply E3 to show $t\sigma >_{\mathsf{ep}} s\sigma$. It remains to show that $t >_{\mathsf{ep}} \mathit{chop}(s)$ implies $t\sigma >_{\mathsf{ep}} \mathit{chop}(s\sigma)$, which follows from the induction hypothesis and from $\mathit{chop}(s)\sigma = \mathit{chop}(s\sigma)$.

If $t >_{\mathsf{ep}} s$ is derived by E4, then $\xi = \zeta \in \mathcal{V}$, $\bar{t}_n \gg^{\mathsf{f}}_{\mathsf{ep}} \bar{s}_m$ for all $\mathsf{f} \in \Sigma$, $n > 0$, and either $m = 0$ or $\mathit{chop}(t) >_{\mathsf{ep}} \mathit{chop}(s)$. We will show that $u\,(\bar{t}_n\sigma) >_{\mathsf{ep}} u\,(\bar{s}_m\sigma)$ for all u with $|u| \leq |\zeta\sigma|$. For $u = \zeta\sigma$, it then follows that $t\sigma >_{\mathsf{ep}} s\sigma$. We show this by induction on $|u|$. We will refer to this induction as the inner induction and to the induction on the multiset $\{|t|, |s|\}$ as the outer induction.

We have to show $u\,(\bar{t}_n\sigma) >_{\mathsf{ep}} u\,(\bar{s}_m\sigma)$. We apply E3 or E4 to do so, depending on whether the head of u is a symbol or a variable. We write $u = \psi\,\bar{u}_r$.

First, we show that $\bar{u}_r \cdot (\bar{t}_n\sigma) \gg^{\mathsf{f}}_{\mathsf{ep}} \bar{u}_r \cdot (\bar{s}_m\sigma)$ for all $\mathsf{f} \in \Sigma$. As above, by the outer induction hypothesis and preservation of stability (property X2) on the set of elements of $\bar{t}_n$ and $\bar{s}_m$, we have $\bar{t}_n\sigma \gg^{\mathsf{f}}_{\mathsf{ep}} \bar{s}_m\sigma$. Then $\bar{u}_r \cdot (\bar{t}_n\sigma) \gg^{\mathsf{f}}_{\mathsf{ep}} \bar{u}_r \cdot (\bar{s}_m\sigma)$ follows by compatibility with prepending (property X8).

If $m = 0$ and $r = 0$, we can apply E3 or E4 directly to show $u\,(\bar{t}_n\sigma) >_{\mathsf{ep}} u\,(\bar{s}_m\sigma)$.

If $r > 0$, then $\mathit{chop}(u\ (\bar{t}_n\sigma)) = \mathit{chop}(u)\ (\bar{t}_n\sigma) >_{\mathsf{ep}} \mathit{chop}(u)\ (\bar{s}_m\sigma) = \mathit{chop}(u\ (\bar{s}_m\sigma))$ by the inner induction hypothesis. If $\psi \in \mathcal{V}$, we can then apply E4 to obtain $u\ (\bar{t}_n\sigma) >_{\mathsf{ep}} u\ (\bar{s}_m\sigma)$. Otherwise, $\psi \in \Sigma$, and we can apply E1 to obtain $u\ (\bar{t}_n\sigma) >_{\mathsf{ep}} \mathit{chop}(u\ (\bar{s}_m\sigma))$ and then E3 to obtain $u\ (\bar{t}_n\sigma) >_{\mathsf{ep}} u\ (\bar{s}_m\sigma)$.

If $m > 0$ and $r = 0$, then we have $\mathit{chop}(t) >_{\mathsf{ep}} \mathit{chop}(s)$, $\mathit{chop}(u\ (\bar{t}_n\sigma)) = \mathit{chop}(t)\sigma$, and $\mathit{chop}(u\ (\bar{s}_m\sigma)) = \mathit{chop}(s)\sigma$. By the outer induction hypothesis, $\mathit{chop}(t)\sigma >_{\mathsf{ep}} \mathit{chop}(s)\sigma$, i.e., $\mathit{chop}(u\ (\bar{t}_n\sigma)) >_{\mathsf{ep}} \mathit{chop}(u\ (\bar{s}_m\sigma))$. As above, if $\psi \in \mathcal{V}$, we can then apply E4 to obtain $u\ (\bar{t}_n\sigma) >_{\mathsf{ep}} u\ (\bar{s}_m\sigma)$. Otherwise, $\psi \in \Sigma$, and we can apply E1 to obtain $u\ (\bar{t}_n\sigma) >_{\mathsf{ep}} \mathit{chop}(u\ (\bar{s}_m\sigma))$ and then E3 to obtain $u\ (\bar{t}_n\sigma) >_{\mathsf{ep}} u\ (\bar{s}_m\sigma)$.

This concludes the inner and the outer induction. □

Theorem 4.14 (Ground Totality)**.** *For ground terms t and s, we have $t <_{\mathsf{ep}} s$, $t = s$, or $t >_{\mathsf{ep}} s$.*

Proof. By well-founded induction on the multiset $\{|t|, |s|\}$ with respect to the multiset extension [50] of $>$ on $\mathbb{N}$. Let $t = \xi\ \bar{t}_n$ and $s = \zeta\ \bar{s}_m$. Then $\xi, \zeta \in \Sigma$ because t and s are ground.

If $n > 0$ and $\mathit{chop}(t) \not\geq_{\mathsf{ep}} s$, then by the induction hypothesis $\mathit{chop}(t) \geq_{\mathsf{ep}} s$ and hence $t >_{\mathsf{ep}} s$ by E1. Thus we can assume that either $n = 0$ or $s >_{\mathsf{ep}} \mathit{chop}(t)$. Analogously, we can assume that either $m = 0$ or $t >_{\mathsf{ep}} \mathit{chop}(s)$.

If $\xi > \zeta$ or $\xi \prec \zeta$, we have $t >_{\mathsf{ep}} s$ or $t <_{\mathsf{ep}} s$ by E2. Otherwise, we have $\xi = \zeta$ by totality of $\succ$. If either $\bar{t} \gg^{\zeta}_{\mathsf{ep}} \bar{s}$ or $\bar{t} \ll^{\zeta}_{\mathsf{ep}} \bar{s}$, then we have $t >_{\mathsf{ep}} s$ or $t <_{\mathsf{ep}} s$ by E3. By the induction hypothesis and preservation of totality (property X7) on the set of elements of $\bar{s}$ and $\bar{t}$, if $\bar{t} \not\gg^{\zeta}_{\mathsf{ep}} \bar{s}$ and $\bar{t} \not\ll^{\zeta}_{\mathsf{ep}} \bar{s}$, then $\bar{t} = \bar{s}$ and hence $t = s$. □

Theorem 4.15 (Well-Foundedness)**.** *The order $>_{\mathsf{ep}}$ is well founded.*

Proof. We suppose that there exists an infinite descending chain $s_0 >_{\mathsf{ep}} s_1 >_{\mathsf{ep}} \cdots$ and derive a contradiction. We use a minimal counterexample argument [56].

A term s is *bad* if there is an infinite descending $>_{\mathsf{ep}}$-chain from s. Other terms are *good*. Without loss of generality, we assume that s_0 has minimal size among all bad terms and that s_{i+1} has minimal size among all bad terms u with $s_i >_{\mathsf{ep}} u$.

For each i, let $U_i = \{u \mid s_i \rhd_{\mathsf{emb}} u\}$, where $\rhd_{\mathsf{emb}}$ is the irreflexive counterpart of $\trianglerighteq_{\mathsf{emb}}$. Let $U = \bigcup_i U_i$. All terms in U are good: If there existed a bad $u \in U_0$, then $|s_0| > |u|$, contradicting the minimality of s_0. If there existed a bad $u \in U_{i+1}$ for some i, then $s_i >_{\mathsf{ep}} s_{i+1} >_{\mathsf{ep}} u$ by the embedding property (Theorem 4.8), contradicting the minimality of s_{i+1}.

Only conditions E2, E3, and E4 can have been used to derive $s_i >_{\mathsf{ep}} s_{i+1}$. If E1 was used, then $\mathit{chop}(s_i) \geq_{\mathsf{ep}} s_{i+1} >_{\mathsf{ep}} s_{i+2}$. But then there would be an infinite descending chain $\mathit{chop}(s_i) >_{\mathsf{ep}} s_{i+2} >_{\mathsf{ep}} s_{i+3} >_{\mathsf{ep}} \cdots$ from $\mathit{chop}(s_i)$, contradicting the goodness of $\mathit{chop}(s_i) \in U$.

E2 can have been used only finitely many times in the chain since E3 and E4 preserve the head and E2 makes the head smaller with respect to the well-founded relation $\succ$. Hence, there is a number k such that the entire chain $s_k >_{\mathsf{ep}} s_{k+1} >_{\mathsf{ep}} \cdots$ has been derived by E3 and E4. Let $s_i = \zeta\ \bar{u}_i$ (where contrary to our usual convention the indices of $\bar{u}_i$ identify the tuple and do not denote its length). Then we have an infinite chain $\bar{u}_k \gg^{\mathsf{f}}_{\mathsf{ep}} \bar{u}_{k+1} \gg^{\mathsf{f}}_{\mathsf{ep}} \cdots$ for some f. All elements of these tuples are in U because

each element of $\bar{u}_i$ is embedded in s_i. However, since all elements of U are good, $>_{\mathsf{ep}}$ is well founded on U. By preservation of well-foundedness (property X5), $\gg^{\mathsf{f}}_{\mathsf{ep}}$ is well founded on U^*, which contradicts the existence of the above $\gg^{\mathsf{f}}_{\mathsf{ep}}$-chain. □

4.6. Examples

The following examples illustrate the benefits of EPO for term rewriting and superposition.

Example 4.16. Consider the following term rewriting system:

$$\mathsf{f}\ x\ \mathsf{Nil} \xrightarrow{1} x \qquad \mathsf{f}\ x\ (\mathsf{A}\ y) \xrightarrow{2} \mathsf{f}\ (\mathsf{A}\ (\mathsf{B}\ x))\ y \qquad \mathsf{f}\ x\ (\mathsf{B}\ y) \xrightarrow{3} \mathsf{f}\ (\mathsf{B}\ (\mathsf{A}\ x))\ y$$

This rewriting system can be interpreted as a definition of a function on strings. In this interpretation, Nil represents the empty string, and chains of applications of the functions A and B to Nil represent strings over the alphabet {A, B}; thus, A (B (B Nil)) represents the string ABB. The function f takes two such strings, reverses the second string, replaces in the resulting string each A by AB and each B by BA, and finally appends the first string.

All three rules are orientable by EPO with the right-to-left length-lexicographic extension for f and precedence $\mathsf{f} > \mathsf{A}, \mathsf{B}$. Rule 1 can clearly be oriented because of the subterm property (Theorem 4.9). To show that rule 2 can be oriented, we apply E3. To do so, we need to prove $(x, \mathsf{A}\ y) \gg^{\mathsf{f}}_{\mathsf{ep}} ((\mathsf{A}\ (\mathsf{B}\ x)), y)$ and $\mathsf{f}\ x\ (\mathsf{A}\ y) >_{\mathsf{ep}} \mathsf{A}\ (\mathsf{B}\ x)\ y$. The former holds by the definition of the right-to-left length-lexicographic extension and by E1. For the latter, we apply E2. To show $\mathsf{f}\ x\ (\mathsf{A}\ y) >_{\mathsf{ep}} \mathsf{B}\ x\ y$, we apply E2 again. To show $\mathsf{f}\ x\ (\mathsf{A}\ y) >_{\mathsf{ep}} x\ y$, we apply E1. To show $x\ (\mathsf{A}\ y) >_{\mathsf{ep}} x\ y$, we apply E4. Finally, $\mathsf{A}\ y >_{\mathsf{ep}} y$ holds by E1. The proof for rule 3 is analogous.

To my knowledge, the literature contains no other ground-total simplification order for λ-free higher-order terms that can orient all three of these rules. Rules 2 and 3 are not orientable by applicative KBO or applicative RPO. With applicative KBO, the weight of the right-hand sides is always too large. With applicative RPO, too many heads are the application symbol app, preventing us from finding an appropriate precedence. With λ-free KBO [16], one of the two rules 2 and 3 can be oriented by assigning either A or B zero weight, but the system as a whole is not orientable with this order either. With λ-free RPO [34], we can orient all three rules, but λ-free RPO is not a simplification order.

This example suggests that EPO with a right-to-left length-lexicographic extension is generally stronger than left-to-right. If the two arguments of f were swapped, one would intuitively attempt to use the left-to-right extension for f, but fail because $\mathsf{f}\ (\mathsf{A}\ y)\ x \not>_{\mathsf{ep}} y\ (\mathsf{A}\ (\mathsf{B}\ x))$. For this system with the arguments of f swapped, applicative RPO can orient all three rules. However, swapping arguments cannot be used as a general approach to orient rewriting systems if the affected function appears unapplied.

The term order's ability to orient equations in the right way can have considerable effects on the performance of superposition provers. The observations made in Example 4.16 have implications for the efficiency of the superposition calculus:

Example 4.17. Consider the rewrite rules from Example 4.16, recast as equations, and the negated conjecture given below, for some $k \in \mathbb{N}$:

$$\mathsf{f}\, x\, \mathsf{Nil} \approx x \qquad \mathsf{f}\, x\, (\mathsf{A}\, y) \approx \mathsf{f}\, (\mathsf{A}\, (\mathsf{B}\, x))\, y \qquad \mathsf{f}\, x\, (\mathsf{B}\, y) \approx \mathsf{f}\, (\mathsf{B}\, (\mathsf{A}\, x))\, y$$
$$\mathsf{f}\, \mathsf{c}\, (\mathsf{AB})^{k+1} \not\approx \mathsf{B}\, (\mathsf{A}\, (\mathsf{f}\, \mathsf{c}\, ((\mathsf{AB})^k\, \mathsf{A})))$$

where $(\mathsf{AB})^{k+1}$ stands for $\mathsf{A}\, (\mathsf{B}\, \ldots (\mathsf{A}\, (\mathsf{B}\, \mathsf{Nil}))\ldots)$ and $(\mathsf{AB})^k \mathsf{A}$ for $\mathsf{A}\, (\mathsf{B}\, \ldots (\mathsf{A}\, \mathsf{Nil})\ldots)$. Using the EPO from Example 4.16 that can orient the equations left to right, superposition provers can solve this problem by simplification rules only. Simplification rules are much more efficient than inference rules because simplifications replace clauses and do not add new ones. Using an order that can orient only the first equation from left to right, we would need at least k inferences; using an order that can orient the first equation and only one of the other two, we would need at least $k/2$ inferences.

4.7. Implementation

I implemented EPO in the Zipperposition prover. To evaluate EPO, we use the nonpurifying intensional calculus presented in the previous chapter, which is the best-performing one among the four. It is designed to deal with orders that do not have compatibility with arguments, such as λ-free RPO, but falls back to a simpler calculus with orders that have full compatibility with contexts, such as λ-free KBO or EPO.

The pseudocode of the implementation of EPO is given in Figure 4.1. As usual in superposition provers, the procedure compares two terms in both directions, yielding one of the answers GreaterThan, Equal, LessThan, or Incomparable. When the pseudocode refers to $>_{\mathsf{ep}}$, $\geq_{\mathsf{ep}}$, and $\gg^{\mathsf{f}}_{\mathsf{ep}}$, this is to be interpreted in terms of the function epo. The syntax '$\xi\ \bar{t}_n$ as t' in the arguments of function definitions means that t denotes the entire term, ξ denotes its head, and $\bar{t}_n$ denotes its arguments. Zipperposition's terms use hash consing, allowing for fast equality checks of terms.

It is crucial to the performance of this implementation to use memoization in the form of a cache on the function epo. For example, to compute that $\mathsf{f}^m\, x \not\geq_{\mathsf{ep}} \mathsf{f}^n\, y$ for $m \leq n$, we need at least 4^m calls to epo if the cache is inactive. With a cache however, only $(m+1)(n+1)$ of these calls to epo have to be computed; the other return values can be found in the cache. More generally, the following lemma holds:

Lemma 4.18. *To calculate the order of two terms t and s, the pseudocode in Figure 4.1 needs at most* $\mathrm{depth}(t) \cdot \mathrm{depth}(s) \cdot |t| \cdot |s|$ *distinct calls to* epo. *Here, the depth of a term $\zeta\ \bar{u}_m$ is* 1 *if $m = 0$ and* $\max_{u \in \bar{u}}(\mathrm{depth}(u)) + 1$ *otherwise.*

Proof. We define a set S_t that overapproximates the set of all embeddings of t that may be involved in computing the order of t with some other term.

To this end, let $\rhd_{\mathsf{arg}}$ be the relation defined by $\zeta\ \bar{u}_n \rhd_{\mathsf{arg}} u_i$ for all terms $\zeta\ \bar{u}_n$ and all i. Let $\rhd_{\mathsf{chop}}$ be the relation defined by $\zeta\ \bar{u}_n \rhd_{\mathsf{chop}} \mathit{chop}(\zeta\ \bar{u}_n)$ for all terms $\zeta\ \bar{u}_n$ with $n > 0$. Finally, let S_t be the set of all terms u such that $t\, (\rhd_{\mathsf{arg}} \cup \rhd_{\mathsf{chop}})^*\, u$. In other words, S_t is inductively defined as follows: Let $t \in S_t$. For any term $\zeta\ \bar{u}_n \in S_t$, let $\mathit{chop}(\zeta\ \bar{u}_n) \in S_t$ and $u_i \in S_t$ for all i.

epo($\xi\,\bar{t}_n$ as t, $\zeta\,\bar{s}_m$ as s) =
  if $t = s$ then Equal
  elif $t \in \mathcal{V}$ and $s \in \mathcal{V}$ then Incomparable
  elif $t \in \mathcal{V}$ then (if t occurs in s then LessThan else Incomparable)
  elif $s \in \mathcal{V}$ then (if s occurs in t then GreaterThan else Incomparable)
  else
    if $\xi > \zeta$ then $\mathsf{check}_{\mathsf{E2,E3}}(t,s)$
    elif $\xi < \zeta$ then $\mathsf{check}^{\mathsf{inv}}_{\mathsf{E2,E3}}(t,s)$
    elif $\xi = \zeta$ and $\zeta \in \Sigma$ then
      if $\bar{t}_n \gg^{\zeta}_{\mathsf{ep}} \bar{s}_m$ then $\mathsf{check}_{\mathsf{E2,E3}}(t,s)$
      elif $\bar{t}_n \ll^{\zeta}_{\mathsf{ep}} \bar{s}_m$ then $\mathsf{check}^{\mathsf{inv}}_{\mathsf{E2,E3}}(t,s)$
      else $\mathsf{check}_{\mathsf{E1}}(t,s)$
    elif $\xi = \zeta$ and $\zeta \in \mathcal{V}$ then
      if $\bar{t}_n \gg^{\mathsf{f}}_{\mathsf{ep}} \bar{s}_m$ for all $\mathsf{f} \in \Sigma$ and $n > 0$ then $\mathsf{check}_{\mathsf{E4}}(t,s)$
      elif $\bar{t}_n \ll^{\mathsf{f}}_{\mathsf{ep}} \bar{s}_m$ for all $\mathsf{f} \in \Sigma$ and $m > 0$ then $\mathsf{check}^{\mathsf{inv}}_{\mathsf{E4}}(t,s)$
      else $\mathsf{check}_{\mathsf{E1}}(t,s)$
    else $\mathsf{check}_{\mathsf{E1}}(t,s)$

$\mathsf{check}_{\mathsf{E1}}(\xi\,\bar{t}_n$ as t, $\zeta\,\bar{s}_m$ as $s)$ =
  if $n > 0$ and $\mathit{chop}(t) \geq_{\mathsf{ep}} s$ then GreaterThan
  elif $m > 0$ and $t \leq_{\mathsf{ep}} \mathit{chop}(s)$ then LessThan
  else Incomparable

$\mathsf{check}_{\mathsf{E2,E3}}(\xi\,\bar{t}_n$ as t, $\zeta\,\bar{s}_m$ as $s)$ =
  if $m = 0$ or $t >_{\mathsf{ep}} \mathit{chop}(s)$ then GreaterThan else $\mathsf{check}_{\mathsf{E1}}(t,s)$

$\mathsf{check}^{\mathsf{inv}}_{\mathsf{E2,E3}}(\xi\,\bar{t}_n$ as t, $\zeta\,\bar{s}_m$ as $s)$ =
  if $n = 0$ or $\mathit{chop}(t) <_{\mathsf{ep}} s$ then LessThan else $\mathsf{check}_{\mathsf{E1}}(t,s)$

$\mathsf{check}_{\mathsf{E4}}(\xi\,\bar{t}_n$ as t, $\zeta\,\bar{s}_m$ as $s)$ =
  if $m = 0$ or $\mathit{chop}(t) >_{\mathsf{ep}} \mathit{chop}(s)$ then GreaterThan else $\mathsf{check}_{\mathsf{E1}}(t,s)$

$\mathsf{check}^{\mathsf{inv}}_{\mathsf{E4}}(\xi\,\bar{t}_n$ as t, $\zeta\,\bar{s}_m$ as $s)$ =
  if $n = 0$ or $\mathit{chop}(t) <_{\mathsf{ep}} \mathit{chop}(s)$ then LessThan else $\mathsf{check}_{\mathsf{E1}}(t,s)$

Figure 4.1: Pseudocode of the EPO implementation

Inspecting the pseudocode, it is obvious that S_t and S_s together overapproximate all terms that are involved in computing the order for the two terms t and s.

In a derivation of $(\rhd_{\mathsf{arg}} \cup \rhd_{\mathsf{chop}})^*$, any $\rhd_{\mathsf{chop}}$ step before a $\rhd_{\mathsf{arg}}$ step can be eliminated. More precisely, we show that $(\rhd_{\mathsf{arg}} \cup \rhd_{\mathsf{chop}})^* = (\rhd_{\mathsf{arg}}^* \circ \rhd_{\mathsf{chop}}^*)$ by proving that $(\rhd_{\mathsf{chop}} \circ \rhd_{\mathsf{arg}}) \subseteq (\rhd_{\mathsf{arg}}^*)$. We assume that $w \rhd_{\mathsf{chop}} v \rhd_{\mathsf{arg}} u$ for some terms w, v, and u. Let $w = \zeta\, \bar{w}_n$. Then $v = \mathit{chop}(\zeta\, \bar{w}_n) = w_1\, w_2 \ldots w_n$. Let $w_1 = \xi\, \bar{v}_n$. Then $v = \xi\, \bar{v}_n\, w_2 \ldots w_n$. Hence $u \in \bar{v}_n$ or $u \in \{w_2, \ldots, w_n\}$. In the first case, we have $w \rhd_{\mathsf{arg}} w_1 \rhd_{\mathsf{arg}} u$; In the second case $w \rhd_{\mathsf{arg}} u$. Either way, $w\ (\rhd_{\mathsf{arg}}^*)\ u$, which is what we needed to show.

Hence, S_t is the set of all terms v such that $t\, (\rhd_{\mathsf{arg}}^* \circ \rhd_{\mathsf{chop}}^*)\, v$. Therefore, we can overapproximate the size of S_t as follows:

$$|S_t| \leq \sum_{u \in \mathcal{T},\, t \rhd_{\mathsf{arg}}^* u} |\{v \mid u \rhd_{\mathsf{chop}}^* v\}| \leq \sum_{u \in \mathcal{T},\, t \rhd_{\mathsf{arg}}^* u} |u| \leq \mathrm{depth}(t) \cdot |t|$$

The last inequality holds because for any number of steps k,

$$\sum_{u \in \mathcal{T},\, t \rhd_{\mathsf{arg}}^k u} |u| \leq |t|$$

and the number of $\rhd_{\mathsf{arg}}$ steps from t is bounded by $\mathrm{depth}(t)$.

Since S_t and S_s together overapproximate all terms that are involved in computing the order for the two terms t and s, we can overapproximate the number of distinct calls to epo by $|S_t \times S_s| = |S_t| \cdot |S_s| \leq \mathrm{depth}(t) \cdot |t| \cdot \mathrm{depth}(s) \cdot |s|$. □

We can use this lemma to derive the computational complexity of epo. The following theorem is stated only for the length-lexicographic extension operators since other extension operators may have a higher computational complexity.

Theorem 4.19. *For each* $\mathsf{f} \in \Sigma$*, let* $> \mapsto >>^{\mathsf{f}}$ *be either the left-to-right or the right-to-left length-lexicographic extension operator. For terms* t *and* s*, the computational complexity of* $\mathrm{epo}(t,s)$ *as given in Figure 4.1 is* $\mathrm{O}(\mathrm{depth}(t) \cdot \mathrm{depth}(s) \cdot |t| \cdot |s| \cdot \max(|t|, |s|))$ *if recursive calls are cached.*

Proof. Let $R(t,s)$ be the set of term pairs (v,u), for which $\mathrm{epo}(t,s)$ triggers directly or indirectly a call to $\mathrm{epo}(v,u)$. Let $C(v,u)$ be the complexity of $\mathrm{epo}(u,v)$ assuming constant time for all recursive calls. Then the computational complexity of $\mathrm{epo}(t,s)$ is

$$\mathrm{O}\left(\sum_{(v,u) \in R(t,s)} C(v,u)\right) \qquad (*)$$

We assume constant time for the recursive calls in the definition of $C(v,u)$ because each recursive call is either the first one for this argument pair and therefore counted by another summand of the sum above, or it is not the first one for this argument pair and can therefore be retrieved from the cache in constant time.

To determine $C(v,u)$, we analyze the implementation in Figure 4.1, assuming that all recursive calls are O(1). Searching for occurrences of a given variable in a term, computing *chop*, counting the number of arguments of a term, and iterating through the arguments for the length-lexicographic comparison are $\mathrm{O}(\max(|v|, |u|))$.

All other operations are O(1). Hence, $C(v,u)$ is O(max(|v|,|u|)). Since the term sizes do not increase in recursive calls, $C(v,u)$ is also O(max(|t|,|s|)) for all $(v,u) \in R(t,s)$. By Lemma 4.18, $|R(t,s)| \leq \mathrm{depth}(t) \cdot \mathrm{depth}(s) \cdot |t| \cdot |s|$. Hence, by (*), the computational complexity of $\mathrm{epo}(t,s)$ is O(depth(t) · depth(s) · |t| · |s| · max(|t|,|s|)). □

Compared with first-order KBO or RPO, this is rather slow. Löchner [102, 103] showed that, with a lexicographic extension, KBO can be computed in O(|t| + |s|) and RPO in O(|t| · |s|). RPO can be implemented so efficiently because the computation of the lexicographic order of the arguments, i.e., computing $\bar{t}_n \gg^{\zeta}_{\mathsf{ep}} \bar{s}_m$, can be merged with testing other conditions, i.e., the condition corresponding to $\mathsf{check}_{\mathsf{E2,E3}}(t,s)$. It is an open question whether a similar optimization is possible for EPO, although it is definitely not as straightforward as for RPO.

4.8. Evaluation

The following evaluation compares the implementation of EPO with other orders in Zipperposition. It was performed with a CPU time limit of 300 s on StarExec nodes equipped with Intel Xeon E5-2609 0 CPUs clocked at 2.40 GHz. The raw evaluation results are available online and reproducible.[2]

From the TPTP [129], 665 higher-order problems in THF format were used, containing both monomorphic and polymorphic problems and excluding problems that contain arithmetic, tuples, the `$distinct` predicate, or the `$ite` symbol, as well as problems whose clausal normal form falls outside the λ-free fragment.

The Sledgehammer (SH) benchmarks, corresponding to Isabelle's Judgment Day suite [40], were regenerated to target λ-free higher-order logic, encoding λ-expressions as λ-lifted supercombinators [106]. The SH benchmarks comprise 1253 problems, each including 256 Isabelle facts.

Besides EPO, I evaluated RPO, KBO, and their applicative counterparts (appRPO, appKBO). Each of the orders were evaluated twice, once using the left-to-right length-lexicographic extension (LTR) and once using the right-to-left length-lexicographic extension (RTL) for all symbols. In principle, EPO also allows for different extension operators for different symbols, but it is unclear how to design appropriate heuristics. The calculus used for EPO, RPO, and KBO is the intensional nonpurifying variant of the calculus described in the previous chapter. For the monotonic orders EPO and KBO, the calculus degrades to essentially first-order superposition, with the addition of an argument congruence rule that adds arguments of partially applied functions. In the case of the nonmonotonic order RPO, the calculus performs additional superposition inferences into variable positions to remain complete. These inferences are known to harm performance, which is why we would generally expect a better performance with monotonic orders. To evaluate the applicative counterparts appKBO and appRPO, I apply the applicative encoding to the given problem directly after the clausal normal form transformation and use first-order KBO and RPO, respectively, on the resulting problem. The results for these last two orders are therefore to be interpreted with care because the applicative encoding also influences various unrelated heuristics in Zipperposition.

[2] https://doi.org/10.5281/zenodo.3992684

		LTR					RTL				
		#sat	#uns	∅tim	%ord	∅cla	#sat	#uns	∅tim	%ord	∅cla
TPTP	EPO	120	463	1.3	6.9	2155	120	462	1.2	6.8	2163
	RPO	119	472	0.3	0.9	1196	119	471	0.3	1.0	1171
	KBO	121	474	0.1	1.6	430	121	473	0.2	1.6	600
	appRPO	138	472	0.6	1.1	749	123	472	1.6	2.0	1489
	appKBO	122	476	0.1	1.9	306	122	476	0.3	2.0	462
SH	EPO	1	509	2.6	23.5	6356	1	505	3.1	23.2	6251
	RPO	1	550	1.6	4.7	7130	1	549	2.4	4.8	8612
	KBO	1	594	1.6	8.8	9206	1	590	1.3	8.7	6949
	appRPO	1	481	13.3	8.1	26346	1	462	17.9	16.3	28897
	appKBO	1	502	10.6	11.3	25236	1	502	10.9	11.6	26202

Figure 4.2: Evaluation

Figure 4.2 displays the number of problems found to be satisfiable (#sat), the number of problems found to be unsatisfiable (#uns), the average CPU time per problem (∅tim), the average percentage of the CPU time used to compute order comparisons (%ord), and the average number of clauses produced during a run (∅cla). When computing the three averages, satisfiable problems and problems that at least one of the ten configurations failed to solve within the time limit were excluded.

From first-order provers, it is well known that KBO generally outperforms RPO. In the #uns columns, we observe the same effect. In the present setting, the advantage of KBO is possibly even greater because the calculus performs inferences into variable positions with RPO. Although these additional superposition inferences are not performed when using EPO, the #uns results for EPO are worse than RPO and KBO. The %ord columns reveal that this is probably because EPO takes considerably more time to compute. I hypothesized that a second reason could be that generally more term pairs are incomparable under EPO and thus more inferences need to be performed and more clauses are produced. Although the numbers in the ∅cla column on the TPTP benchmark set confirm this hypothesis, the corresponding numbers on the SH benchmark set contradict it because on those benchmarks, EPO is actually producing the least amount of clauses.

The raw data show that despite the poor performance of RPO and EPO these orders can be put to good use in a portfolio prover. The RPO configurations can solve 16 problems that neither of the KBO configurations can solve. The EPO configurations can solve 11 problems that neither of the RPO configurations can solve, 12 problems that neither of the KBO configurations can solve, 51 problems that neither of the appRPO configurations can solve, 66 problems that neither of the appKBO configurations can solve, and 4 problems that no other configuration can solve. Most of the problems where EPO outperforms other orders are in the SH benchmark set. Overall, RPO is preferable over EPO if one is willing to face the complications of a nonmonotonic order in theory and in implementation.

The direction (LTR or RTL) of the length-lexicographic extension does not have a large impact. For KBO and appKBO, this is to be expected since the lexicographic

comparison comes into play only when weights are equal. For EPO, the advantage of RTL that Example 4.16 suggests is not corroborated by the evaluation. Only with appRPO, LTR performs better than RTL. This might be because LTR tends to put more importance to the symbols that were at the heads of terms before the applicative encoding, yielding a better measure of the complexity of a term.

4.9. Conclusion

I presented a ground-total simplification order for λ-free higher-order terms resembling RPO. In first-order logic, KBO generally outperforms RPO, but RPO with well-chosen parameters behaves better than KBO on many examples. In λ-free higher-order logic, the situation appears to be similar. However, RPO cannot be easily used for superposition in this logic if we want the calculus to remain complete because the natural generalization [34] lacks compatibility with contexts. EPO seems to be a good replacement to fill the role of RPO in λ-free higher-order logic if one wants to avoid the complications of nonmonotonic orders. Otherwise, calculi specialized to deal with nonmonotonic orders are the better choice.

To explore different candidate definitions for EPO, I formalized my ideas early on in Isabelle/HOL [108]. This allowed me to keep track of changes in the definition and how they influence the properties and their proofs more easily. To find examples explaining why certain properties do not hold for some tentative definitions of EPO, Lazy SmallCheck [113] was of great help. For instance, it was Lazy SmallCheck that found the example $x\ \mathsf{f}\ \mathsf{f} >'_{\mathsf{ep}} x\ x$ versus $\mathsf{f}\ y\ \mathsf{f}\ \mathsf{f} \not>'_{\mathsf{ep}} \mathsf{f}\ y\ (\mathsf{f}\ y)$ mentioned in Section 4.4.

In future work, I would like to investigate whether the computation of EPO can be optimized further. To put EPO to use in practice, implementing it in E prover [120] would be a good target because E's λ-free higher-order mode is designed for ground-total simplification orders and its calculus is more efficient for those than Zipperposition's by circumventing the argument congruence rule.

Another application of EPO could lie in termination of λ-free higher-order term rewriting. Together with λ-free KBO [16], it could serve as a basis for a generalization of the dependency pair framework [64] to λ-free higher-order terms, which would compete with dependency pair methods for full higher-order terms [35, 60, 95, 96]. Such a generalized framework could be used to prove termination of applicative first-order benchmarks from the TPDB library [63] and similar problems.

5

Superposition with Lambdas

Joint work with
Jasmin Blanchette, Sophie Tourret,
Petar Vukmirović, and Uwe Waldmann

Based on the ideas developed in Chapter 3, we designed a superposition calculus for a clausal fragment of extensional polymorphic higher-order logic that includes anonymous functions but excludes Booleans. The inference rules work on $\beta\eta$-equivalence classes of λ-terms and rely on higher-order unification to achieve refutational completeness. We implemented the calculus in the Zipperposition prover and evaluated it on TPTP and Isabelle benchmarks.

My contributions to this chapter are the design of the core calculus, the redundancy criterion, the ground completeness proof, the soundness and satisfiability preservation proofs of the extensions, and parts of the implementation.

Parts of this chapter have been published at the Conference on Automated Deduction (CADE-27), LNCS 11716, pp. 55–73, Springer, 2019. This chapter has been accepted to be published in the Journal of Automated Reasoning.

5.1. Introduction

Designing a superposition calculus with λ-expressions poses the following three main challenges:

1. Standard superposition is parameterized by a ground-total simplification order $\succ$, but such orders do not exist for λ-terms equal up to β-conversion. The relations designed for proving termination of higher-order term rewriting systems, such as HORPO [78] and CPO [36], lack many of the desired properties (e.g., transitivity, stability under grounding substitutions).

2. Higher-order unification is undecidable and may give rise to an infinite set of incomparable unifiers. For example, the constraint $\mathsf{f}\,(y\,\mathsf{a}) \stackrel{?}{=} y\,(\mathsf{f}\,\mathsf{a})$ admits infinitely many independent solutions of the form $\{y \mapsto \lambda x.\, \mathsf{f}^n\, x\}$.

3. In first-order logic, to rewrite into a term s using an oriented equation $t \approx t'$, it suffices to find a subterm of s that is unifiable with t. In higher-order logic, this is insufficient. Consider superposition from $\mathsf{f}\,\mathsf{c} \approx \mathsf{a}$ into $y\,\mathsf{c} \not\approx y\,\mathsf{b}$. The left-hand sides can obviously be unified by $\{y \mapsto \mathsf{f}\}$, but the more general $\{y \mapsto \lambda x\, z\, x\,(\mathsf{f}\, x)\}$ also gives rise to a subterm $\mathsf{f}\,\mathsf{c}$ after β-reduction. The corresponding inference generates the clause $z\,\mathsf{c}\,\mathsf{a} \not\approx z\,\mathsf{b}\,(\mathsf{f}\,\mathsf{b})$.

To address the first challenge, we adopt the η-short β-normal form to represent $\beta\eta$-equivalence classes of λ-terms. In the spirit of Jouannaud and Rubio's early joint work [77], we state requirements on the term order only for ground terms (i.e., closed monomorphic $\beta\eta$-equivalence classes); the nonground case is connected to the ground case via stability under grounding substitutions. Even on ground terms, we cannot obtain all desirable properties. We sacrifice compatibility with arguments (the property that $s' \succ s$ implies $s'\,t \succ s\,t$), compensating with an *argument congruence* rule (ArgCong), as in Chapter 3.

For the second challenge, we accept that there might be infinitely many incomparable unifiers and enumerate a complete set (including the notorious flex–flex pairs [74]), relying on heuristics to postpone the combinatorial explosion. The saturation loop must also be adapted to interleave this enumeration with the theorem prover's other activities (Section 5.6). Despite its reputation for explosiveness, higher-order unification is a conceptual improvement over SK combinators, because it can often *compute* the right unifier. Consider the conjecture $\exists z.\, \forall x\, y.\, z\, x\, y \approx \mathsf{f}\, y\, x$. After negation, clausification, and skolemization (which are as for first-order logic), the formula becomes $z\,(\mathsf{sk}_{\mathsf{x}}\, z)\,(\mathsf{sk}_{\mathsf{y}}\, z) \not\approx \mathsf{f}\,(\mathsf{sk}_{\mathsf{y}}\, z)\,(\mathsf{sk}_{\mathsf{x}}\, z)$. Higher-order unification quickly computes the unique unifier: $\{z \mapsto \lambda x\, y.\, \mathsf{f}\, y\, x\}$. In contrast, an encoding approach based on combinators, similar to the one implemented in Sledgehammer [106], would blindly enumerate all possible SK terms for z until the right one, $\mathsf{S}(\mathsf{K}(\mathsf{S}\,\mathsf{f}))\mathsf{K}$, is found. Given the definitions $\mathsf{S}\, z\, y\, x \approx z\, x\,(y\, x)$ and $\mathsf{K}\, x\, y \approx x$, the E prover [120] in *auto* mode needs to perform 3757 inferences to derive the empty clause.

For the third challenge, the idea is that, when applying $t \approx t'$ to perform rewriting inside a higher-order term s, we can encode an arbitrary context as a fresh higher-order variable z, unifying s with $z\, t$; the result is $(z\, t')\sigma$, for some unifier σ. This is performed by a dedicated *fluid subterm superposition* rule (FluidSup).

Functional extensionality is also considered a quintessential higher-order chal-

lenge [20], although similar difficulties arise with first-order sets and arrays [68]. Our approach is to add extensionality as an axiom and provide optional rules as optimizations (Section 5.5). With this axiom, our calculus is refutationally complete w.r.t. extensional Henkin semantics (Section 5.4).

We implemented the calculus in the Zipperposition prover [47] (Section 5.6). Our empirical evaluation includes benchmarks from the TPTP [130] and interactive verification problems exported from Isabelle/HOL [40] (Section 5.7). The results clearly demonstrate the calculus's potential.

5.2. Logic

Our extensional polymorphic clausal higher-order logic is a restriction of full TPTP THF [22] to rank-1 (top-level) polymorphism, as in TH1 [79]. In keeping with standard superposition, we consider only formulas in clausal normal form, without explicit quantifiers or Boolean type. We use Henkin semantics [21,59,71], as opposed to the standard semantics that is commonly considered the foundation of the HOL systems [67]. However, both of these semantics are compatible with the notion of provability employed by the HOL systems. By admitting nonstandard models, Henkin semantics is not subject to Gödel's first incompleteness theorem, allowing us to claim not only soundness but also refutational completeness of our calculus.

Syntax We fix a set Σ_{ty} of type constructors with arities and a set $\mathcal{V}_{\mathsf{ty}}$ of type variables. We require at least one nullary type constructor and a binary function type constructor $\to$ to be present in Σ_{ty}. A type τ, υ is either a type variable $\alpha \in \mathcal{V}_{\mathsf{ty}}$ or has the form $\kappa(\bar{\tau}_n)$ for an n-ary type constructor $\kappa \in \Sigma_{\mathsf{ty}}$ and types $\bar{\tau}_n$. We write κ for $\kappa()$ and $\tau \to \upsilon$ for $\to(\tau, \upsilon)$. Type declarations have the form $\Pi\bar{\alpha}_m.\, \tau$ (or simply τ if $m = 0$), where all type variables occurring in τ belong to $\bar{\alpha}_m$.

We fix a set Σ of (function) symbols $\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{f}, \mathsf{g}, \mathsf{h}, \ldots$, with type declarations, written as $\mathsf{f} : \Pi\bar{\alpha}_m.\, \tau$ or f, and a set $\mathcal{V}$ of term variables with associated types, written as $x : \tau$ or x. The notation $t : \tau$ will also be used to indicate the type of arbitrary terms t. We require the presence of a symbol of type $\Pi\alpha.\, \alpha$ and of a symbol $\mathsf{diff} : \Pi\alpha, \beta.\, (\alpha \to \beta) \to (\alpha \to \beta) \to \alpha$ in Σ. We use diff to express the polymorphic functional extensionality axiom. A signature is a pair $(\Sigma_{\mathsf{ty}}, \Sigma)$.

In the following, we will define terms in three layers of abstraction: raw λ-terms, λ-terms, and terms; where λ-terms will be α-equivalence classes of raw λ-terms and terms will be $\beta\eta$-equivalence classes of λ-terms.

The *raw λ-terms* over a given signature and their associated types are defined inductively as follows. Every $x : \tau \in \mathcal{V}$ is a raw λ-term of type τ. If $\mathsf{f} : \Pi\bar{\alpha}_m.\, \tau \in \Sigma$ and $\bar{\upsilon}_m$ is a tuple of types, called *type arguments*, then $\mathsf{f}\langle\bar{\upsilon}_m\rangle$ (or f if $m = 0$) is a raw λ-term of type $\tau\{\bar{\alpha}_m \mapsto \bar{\upsilon}_m\}$. If $x : \tau$ and $t : \upsilon$, then the *λ-expression* $\lambda x.\, t$ is a raw λ-term of type $\tau \to \upsilon$. If $s : \tau \to \upsilon$ and $t : \tau$, then the *application* $s\, t$ is a raw λ-term of type υ.

The function type constructor $\to$ is right-associative; application is left-associative. Using the spine notation [44], raw λ-terms can be decomposed in a unique way as a nonapplication *head* t applied to zero or more arguments: $t\, s_1 \ldots s_n$ or $t\, \bar{s}_n$ (abusing notation).

A raw λ-term s is a *subterm* of a raw λ-term t, written $t = t[s]$, if $t = s$, if $t = (\lambda x.\, u[s])$, if $t = (u[s])\, v$, or if $t = u\,(v[s])$ for some raw λ-terms u and v. A *proper* subterm of a raw λ-term t is any subterm of t that is distinct from t itself.

A variable occurrence is *free* in a raw λ-term if it is not bound by a λ-expression. A raw λ-term is *ground* if it is built without using type variables and contains no free term variables.

The α-renaming rule is defined as $(\lambda x.\, t) \longrightarrow_\alpha (\lambda y.\, t\{x \mapsto y\})$, where y does not occur free in t and is not captured by a λ-binder in t. Raw λ-terms form equivalence classes modulo α-renaming, called *λ-terms*. We lift the above notions on raw λ-terms to λ-terms.

A substitution ρ is a function from type variables to types and from term variables to λ-terms such that it maps all but finitely many variables to themselves. We require that it is type-correct—i.e., for each $x : \tau \in \mathcal{V}$, $x\rho$ is of type $\tau\rho$. The letters $\theta, \pi, \rho, \sigma$ are reserved for substitutions. Substitutions α-rename λ-terms to avoid capture; for example, $(\lambda x.\, y)\{y \mapsto x\} = (\lambda x'.\, x)$. The composition $\rho\sigma$ applies ρ first: $t\rho\sigma = (t\rho)\sigma$. The notation $\sigma[\bar{x}_n \mapsto \bar{s}_n]$ denotes the substitution that replaces each x_i by s_i and that otherwise coincides with σ.

5

The β- and η-reduction rules are specified on λ-terms as $(\lambda x.\, t)u \longrightarrow_\beta t\{x \mapsto u\}$ and $(\lambda x.\, t\, x) \longrightarrow_\eta t$. For β, bound variables in t are implicitly renamed to avoid capture; for η, the variable x must not occur free in t. The λ-terms form equivalence classes modulo $\beta\eta$-reduction, called *$\beta\eta$-equivalence classes* or simply *terms*.

Convention 5.1. When defining operations that need to analyze the structure of terms, we will use the η-short β-normal form $t\downarrow_{\beta\eta}$, obtained by applying $\longrightarrow_\beta$ and $\longrightarrow_\eta$ exhaustively, as a representative of the equivalence class t. In particular, we lift the notions of subterms and occurrences of variables to $\beta\eta$-equivalence classes via their η-short β-normal representative.

Many authors prefer the η-long β-normal form [74, 77, 105], but in a polymorphic setting it has the drawback that instantiating a type variable with a functional type can lead to η-expansion. We reserve the letters s, t, u, v for terms and x, y, z for variables.

An equation $s \approx t$ is formally an unordered pair of terms s and t. A literal is an equation or a negated equation, written $\neg\, s \approx t$ or $s \not\approx t$. A clause $L_1 \vee \cdots \vee L_n$ is a finite multiset of literals L_j. The empty clause is written as $\bot$.

A *complete set of unifiers* on a set X of variables for two terms s and t is a set U of unifiers of s and t such that for every unifier θ of s and t there exists a member $\sigma \in U$ and a substitution ρ such that $x\sigma\rho = x\theta$ for all $x \in X$. We let $\mathrm{CSU}_X(s, t)$ denote an arbitrary (preferably minimal) complete set of unifiers on X for s and t. We assume that all $\sigma \in \mathrm{CSU}_X(s, t)$ are idempotent on X—i.e., $x\sigma\sigma = x\sigma$ for all $x \in X$. The set X will consist of the free variables of the clauses in which s and t occur and will be left implicit.

Given a substitution σ, the σ-instance of a term t or clause C is the term $t\sigma$ or the clause $C\sigma$, respectively. If $t\sigma$ or $C\sigma$ is ground, we call it a σ-ground instance.

Semantics A *type interpretation* $\mathcal{I}_{\mathsf{ty}} = (\mathcal{U}, \mathcal{J}_{\mathsf{ty}})$ is defined as follows. The *universe* $\mathcal{U}$ is a nonempty collection of nonempty sets, called *domains*. The function $\mathcal{J}_{\mathsf{ty}}$

associates a function $\mathcal{J}_{\mathsf{ty}}(\kappa) : \mathcal{U}^n \to \mathcal{U}$ with each n-ary type constructor κ, such that for all domains $\mathcal{D}_1, \mathcal{D}_2 \in \mathcal{U}$, the set $\mathcal{J}_{\mathsf{ty}}(\to)(\mathcal{D}_1, \mathcal{D}_2)$ is a subset of the function space from $\mathcal{D}_1$ to $\mathcal{D}_2$. The semantics is *standard* if $\mathcal{J}_{\mathsf{ty}}(\to)(\mathcal{D}_1, \mathcal{D}_2)$ is the entire function space for all $\mathcal{D}_1, \mathcal{D}_2$.

A *type valuation* ξ is a function that maps every type variable to a domain. The *denotation* of a type for a type interpretation $\mathcal{I}_{\mathsf{ty}}$ and a type valuation ξ is defined by $[\![\alpha]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}} = \xi(\alpha)$ and $[\![\kappa(\bar{\tau})]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}} = \mathcal{J}_{\mathsf{ty}}(\kappa)([\![\bar{\tau}]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}})$. We abuse notation by applying an operation on a tuple when it must be applied elementwise; thus, $[\![\bar{\tau}_n]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}$ stands for $[\![\tau_1]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}, \ldots, [\![\tau_n]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}$. A type valuation ξ can be extended to be a *valuation* by additionally assigning an element $\xi(x) \in [\![\tau]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}$ to each variable $x : \tau$. An *interpretation function* $\mathcal{J}$ for a type interpretation $\mathcal{I}_{\mathsf{ty}}$ associates with each symbol $\mathsf{f} : \Pi\bar{\alpha}_m.\, \tau$ and domain tuple $\bar{\mathcal{D}}_m \in \mathcal{U}^m$ a value $\mathcal{J}(\mathsf{f}, \bar{\mathcal{D}}_m) \in [\![\tau]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}$, where ξ is the type valuation that maps each α_i to $\mathcal{D}_i$.

The comprehension principle states that every function designated by a λ-expression is contained in the corresponding domain. Loosely following Fitting [59, Section 2.4], we initially allow λ-expressions to designate arbitrary elements of the domain, to be able to define the denotation of a term. We impose restrictions afterwards using the notion of a proper interpretation. A *λ-designation function* $\mathcal{L}$ for a type interpretation $\mathcal{I}_{\mathsf{ty}}$ is a function that maps a valuation ξ and a λ-expression of type τ to elements of $[\![\tau]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}$. A type interpretation, an interpretation function, and a λ-designation function form an (*extensional*) *interpretation* $\mathcal{I} = (\mathcal{I}_{\mathsf{ty}}, \mathcal{J}, \mathcal{L})$. For an interpretation $\mathcal{I}$ and a valuation ξ, the denotation of a term is defined as $[\![x]\!]^{\xi}_{\mathcal{I}} = \xi(x)$, $[\![\mathsf{f}\langle\bar{\tau}_m\rangle]\!]^{\xi}_{\mathcal{I}} = \mathcal{J}(\mathsf{f}, [\![\bar{\tau}_m]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}})$, $[\![s\, t]\!]^{\xi}_{\mathcal{I}} = [\![s]\!]^{\xi}_{\mathcal{I}}([\![t]\!]^{\xi}_{\mathcal{I}})$, and $[\![\lambda x.\, t]\!]^{\xi}_{\mathcal{I}} = \mathcal{L}(\xi, \lambda x.\, t)$. For ground terms t, the denotation does not depend on the choice of the valuation ξ, which is why we sometimes write $[\![t]\!]_{\mathcal{I}}$ for $[\![t]\!]^{\xi}_{\mathcal{I}}$.

An interpretation $\mathcal{I}$ is *proper* if $[\![\lambda x.\, t]\!]^{\xi}_{\mathcal{I}}(a) = [\![t]\!]^{\xi[x \mapsto a]}_{\mathcal{I}}$ for all λ-expressions $\lambda x.\, t$, all valuations ξ, and all a. If a type interpretation $\mathcal{I}_{\mathsf{ty}}$ and an interpretation function $\mathcal{J}$ can be extended by a λ-designation function $\mathcal{L}$ to a proper interpretation $(\mathcal{I}_{\mathsf{ty}}, \mathcal{J}, \mathcal{L})$, then this $\mathcal{L}$ is unique [59, Proposition 2.18]. Given an interpretation $\mathcal{I}$ and a valuation ξ, an equation $s \approx t$ is true if $[\![s]\!]^{\xi}_{\mathcal{I}}$ and $[\![t]\!]^{\xi}_{\mathcal{I}}$ are equal and it is false otherwise. A disequation $s \not\approx t$ is true if $s \approx t$ is false. A clause is true if at least one of its literals is true. A clause set is true if all its clauses are true. A proper interpretation $\mathcal{I}$ is a *model* of a clause set N, written $\mathcal{I} \models N$, if N is true in $\mathcal{I}$ for all valuations ξ.

Axiomatization of Booleans Our clausal logic lacks a Boolean type, but it can easily be axiomatized as follows. We extend the signature with a nullary type constructor $\mathit{bool} \in \Sigma_{\mathsf{ty}}$ equipped with the proxy constants $\mathsf{t}, \mathsf{f} : \mathit{bool}$, $\mathsf{not} : \mathit{bool} \to \mathit{bool}$, $\mathsf{and}, \mathsf{or}, \mathsf{impl}, \mathsf{equiv} : \mathit{bool} \to \mathit{bool} \to \mathit{bool}$, $\mathsf{forall}, \mathsf{exists} : \Pi\alpha.\, (\alpha \to \mathit{bool}) \to \mathit{bool}$, $\mathsf{eq} : \Pi\alpha.\, \alpha \to \alpha \to \mathit{bool}$, and $\mathsf{choice} : \Pi\alpha.\, (\alpha \to \mathit{bool}) \to \alpha$, characterized by the axioms

$$
\begin{array}{ccc}
\mathsf{t} \not\approx \mathsf{f} & \mathsf{or}\,\mathsf{t}\,x \approx \mathsf{t} & \mathsf{equiv}\,x\,y \approx \mathsf{and}\,(\mathsf{impl}\,x\,y)\,(\mathsf{impl}\,y\,x) \\
x \approx \mathsf{t} \vee x \approx \mathsf{f} & \mathsf{or}\,\mathsf{f}\,x \approx x & \mathsf{forall}\langle\alpha\rangle\,(\lambda x.\, \mathsf{t}) \approx \mathsf{t} \\
\mathsf{not}\,\mathsf{t} \approx \mathsf{f} & \mathsf{impl}\,\mathsf{t}\,x \approx x & y \approx (\lambda x.\, \mathsf{t}) \vee \mathsf{forall}\langle\alpha\rangle\,y \approx \mathsf{f} \\
\mathsf{not}\,\mathsf{f} \approx \mathsf{t} & \mathsf{impl}\,\mathsf{f}\,x \approx \mathsf{t} & \mathsf{exists}\langle\alpha\rangle\,y \approx \mathsf{not}\,(\mathsf{forall}\langle\alpha\rangle\,(\lambda x.\, \mathsf{not}\,(y\,x))) \\
\mathsf{and}\,\mathsf{t}\,x \approx x & x \not\approx y \vee \mathsf{eq}\langle\alpha\rangle\,x\,y \approx \mathsf{t} & y\,x \approx \mathsf{f} \vee y\,(\mathsf{choice}\langle\alpha\rangle\,y) \approx \mathsf{t} \\
\mathsf{and}\,\mathsf{f}\,x \approx \mathsf{f} & x \approx y \vee \mathsf{eq}\langle\alpha\rangle\,x\,y \approx \mathsf{f} &
\end{array}
$$

This axiomatization of Booleans can be used in a prover to support full higher-order logic with or without Hilbert choice, corresponding to the TPTP THF format variants TH0 (monomorphic) [131] and TH1 (polymorphic) [79]. The prover's clausifier would transform the outer first-order skeleton of a formula into a clause and use the axiomatized Booleans within the terms. It would also add the proxy axioms to the clausal problem. As an alternative to this complete axiomatization, Vukmirović and Nummelin [138] present a possibly refutationally incomplete calculus extension with dedicated rules to support Booleans. This approach works better in practice and contributed to Zipperposition's victory at CASC 2020.

5.3. The Calculus

Our *Boolean-free λ-superposition calculus* presented here is inspired by the extensional nonpurifying Boolean-free λ-free higher-order superposition calculus described in Chapter 3. The central idea is that superposition inferences are restricted to *unapplied* subterms occurring in the first-order outer skeleton of clauses—that is, outside λ-expressions and outside the arguments of applied variables. Thus, $\mathsf{g} \approx (\lambda x.\, \mathsf{f}\, x\, x)$ cannot be used directly to rewrite $\mathsf{g}\, \mathsf{a}$ to $\mathsf{f}\, \mathsf{a}\, \mathsf{a}$, because g is applied in $\mathsf{g}\, \mathsf{a}$. A separate inference rule, ArgCong, takes care of deriving $\mathsf{g}\, x \approx \mathsf{f}\, x\, x$, which can be oriented independently of its parent clause and used to rewrite $\mathsf{g}\, \mathsf{a}$ or $\mathsf{f}\, \mathsf{a}\, \mathsf{a}$.

As in Chapter 3, we call the subterms that superposition inferences are restricted to "green subterms." However, the notion must be adapted to this chapter's notion of terms. We will no longer consider arguments of variables to be green because in this chapter's logic they can vanish when instantiating variables due to β-reduction. Subterms inside λ-expressions will not be considered green either.

Definition 5.2 (Green positions and subterms)**.** The *green positions* and *green subterms* of a term (i.e., a βη-equivalence class) are defined inductively as follows. A green position is a tuple of natural numbers. For any term t, the empty tuple ε is a green position of t, and t is the green subterm of t at position ε. For all symbols $\mathsf{f} \in \Sigma$, types $\bar{\tau}$, and terms $\bar{u}$, if t is a green subterm of u_i at some position p for some i, then $i.p$ is a green position of $\mathsf{f}\langle\bar{\tau}\rangle\, \bar{u}$, and t is the green subterm of $\mathsf{f}\langle\bar{\tau}\rangle\, \bar{u}$ at position $i.p$. We denote the green subterm of s at the green position p by $s|_p$.

In $\mathsf{f}(\mathsf{g}\, \mathsf{a})(y\, \mathsf{b})(\lambda x.\, \mathsf{h}\, \mathsf{c}(\mathsf{g}\, x))$, the proper green subterms are a, $\mathsf{g}\, \mathsf{a}$, $y\, \mathsf{b}$, and $\lambda x.\, \mathsf{h}\, \mathsf{c}(\mathsf{g}\, x)$. The last two of these do not look like first-order terms and hence their subterms are not green.

Definition 5.3 (Green contexts)**.** We write $t = s\langle u\rangle_p$ to express that u is a green subterm of t at the green position p and call $s\langle\,\rangle_p$ a *green context*. We omit the subscript p if there are no ambiguities.

In a βη-normal representative of a green context, the hole never occurs applied. Therefore, inserting a βη-normal term into the context produces another βη-normal term.

Another key notion is that of a fluid term:

Definition 5.4 (Fluid terms)**.** A term t is called *fluid* if (1) $t\downarrow_{\beta\eta}$ is of the form $y\,\bar{u}_n$ where $n \geq 1$, or (2) $t\downarrow_{\beta\eta}$ is a λ-expression and there exists a substitution σ such that $t\sigma\downarrow_{\beta\eta}$ is not a λ-expression (due to η-reduction).

Case (2) can arise only if t contains an applied variable. Intuitively, fluid terms are terms whose η-short β-normal form can change radically as a result of instantiation. For example, $\lambda x.\, y\,\mathsf{a}\,(z\,x)$ is fluid because applying $\{z \mapsto \lambda x.\, x\}$ makes the λ vanish: $(\lambda x.\, y\,\mathsf{a}\,x) = y\,\mathsf{a}$. Similarly, $\lambda x.\, \mathsf{f}\,(y\,x)\,x$ is fluid because $(\lambda x.\, \mathsf{f}\,(y\,x)\,x)\{y \mapsto \lambda x.\, \mathsf{a}\} = (\lambda x.\, \mathsf{f}\,\mathsf{a}\,x) = \mathsf{f}\,\mathsf{a}$.

5.3.1. The Core Inference Rules

The calculus is parameterized by a strict and a nonstrict term order and a selection function. These concepts are defined below.

Definition 5.5 (Strict ground term order)**.** A *strict ground term order* is a well-founded strict total order $\succ$ on ground terms satisfying the following criteria, where $\succeq$ denotes the reflexive closure of $\succ$:

- *green subterm property*: $t\langle s\rangle \succeq s$;
- *compatibility with green contexts*: $s' \succ s$ implies $t\langle s'\rangle \succ t\langle s\rangle$.

Given a strict ground term order, we extend it to literals and clauses via the multiset extensions in the standard way [9, Section 2.4].

Two properties that are not required are *compatibility with λ-expressions* ($s' \succ s$ implies $(\lambda x.\, s') \succ (\lambda x.\, s)$) and *compatibility with arguments* ($s' \succ s$ implies $s'\,t \succ s\,t$). The latter would even be inconsistent with totality. To see why, consider the symbols $\mathsf{c} \succ \mathsf{b} \succ \mathsf{a}$ and the terms $\lambda x.\, \mathsf{b}$ and $\lambda x.\, x$. Owing to totality, one of the terms must be larger than the other, say, $(\lambda x.\, \mathsf{b}) \succ (\lambda x.\, x)$. By compatibility with arguments, we get $(\lambda x.\, \mathsf{b})\,\mathsf{c} \succ (\lambda x.\, x)\,\mathsf{c}$, i.e., $\mathsf{b} \succ \mathsf{c}$, a contradiction. A similar line of reasoning applies if $(\lambda x.\, \mathsf{b}) \prec (\lambda x.\, x)$, using a instead of c.

Definition 5.6 (Strict term order)**.** A *strict term order* is a relation $\succ$ on terms, literals, and clauses such that the restriction to ground entities is a strict ground term order and such that it is stable under grounding substitutions (i.e., $t \succ s$ implies $t\theta \succ s\theta$ for all substitutions θ grounding the entities t and s).

Definition 5.7 (Nonstrict term order)**.** Given a strict term order $\succ$ and its reflexive closure $\succeq$, a *nonstrict term order* is a relation $\succsim$ on terms, literals, and clauses such that $t \succsim s$ implies $t\theta \succeq s\theta$ for all θ grounding the entities t and s.

Although we call them orders, a strict term order $\succ$ is not required to be transitive on nonground entities, and a nonstrict term order $\succsim$ does not need to be transitive at all. Normally, $t \succeq s$ should imply $t \succsim s$, but this is not required either. A nonstrict term order $\succsim$ allows us to be more precise than the reflexive closure $\succeq$ of $\succ$. For example, we cannot have $y\,\mathsf{b} \succeq y\,\mathsf{a}$, because $y\,\mathsf{b} \neq y\,\mathsf{a}$ and $y\,\mathsf{b} \not\succ y\,\mathsf{a}$ by stability under grounding substitutions (with $\{y \mapsto \lambda x.\, \mathsf{c}\}$). But we can have $y\,\mathsf{b} \succsim y\,\mathsf{a}$ if $\mathsf{b} \succ \mathsf{a}$. In practice, the strict and the nonstrict term order should be chosen so that they can compare as many pairs of terms as possible while being computable and reasonably efficient.

Definition 5.8 (Maximality)**.** An element x of a multiset M is $\trianglerighteq$*-maximal* for some relation $\trianglerighteq$ if for all $y \in M$ with $y \trianglerighteq x$, we have $y \trianglelefteq x$. It is *strictly* $\trianglerighteq$*-maximal* if it is $\trianglerighteq$-maximal and occurs only once in M.

Definition 5.9 (Selection function)**.** A *selection function* is a function that maps each clause to a subclause consisting of negative literals, which we call the *selected* literals of that clause. A literal $L\langle y\rangle$ must not be selected if $y\,\bar{u}_n$, with $n > 0$, is a $\succeq$-maximal term of the clause.

The restriction on the selection function is needed for our proof, but it is an open question whether it is actually necessary for refutational completeness.

Our calculus is parameterized by a strict term order $\succ$, a nonstrict term order $\succsim$, and a selection function *HSel*. The calculus rules depend on the following auxiliary notions.

Definition 5.10 (Eligibility)**.** A literal L is (*strictly*) $\trianglerighteq$*-eligible* w.r.t. a substitution σ in C for some relation $\trianglerighteq$ if it is selected in C or there are no selected literals in C and $L\sigma$ is (strictly) $\trianglerighteq$-maximal in $C\sigma$. If σ is the identity substitution, we leave it implicit.

Definition 5.11 (Deep occurrences)**.** A variable *occurs deeply* in a clause C if it occurs inside a λ-expression or inside an argument of an applied variable.

For example, x and z occur deeply in $\mathsf{f}\,x\,y \approx y\,x \vee z \not\approx (\lambda w.\,z\,\mathsf{a})$, whereas y does not occur deeply. The purpose of this definition is to capture all variables with an occurrence that corresponds to a position inside a λ-expression in some ground instances of C.

The first rule of our calculus is the superposition rule. We regard positive and negative superposition as two cases of a single rule

$$\frac{\overbrace{D' \vee t \approx t'}^{D} \qquad \overbrace{C' \vee s\langle u\rangle \mathrel{\dot{\approx}} s'}^{C}}{(D' \vee C' \vee s\langle t'\rangle \mathrel{\dot{\approx}} s')\sigma}\ \textsc{Sup}$$

where $\dot{\approx}$ denotes either $\approx$ or $\not\approx$. The following side conditions apply:

1. u is not fluid;
2. u is not a variable deeply occurring in C;
3. *variable condition*: if u is a variable y, there must exist a grounding substitution θ such that $t\sigma\theta \succ t'\sigma\theta$ and $C\sigma\theta \prec C''\sigma\theta$, where $C'' = C\{y \mapsto t'\}$;
4. $\sigma \in \mathrm{CSU}(t, u)$;
5. $t\sigma \not\precsim t'\sigma$;
6. $s\langle u\rangle\sigma \not\precsim s'\sigma$;
7. $C\sigma \not\precsim D\sigma$;
8. $t \approx t'$ is strictly $\succsim$-eligible in D w.r.t. σ;
9. $s\langle u\rangle \mathrel{\dot{\approx}} s'$ is $\succsim$-eligible in C w.r.t. σ, and strictly $\succsim$-eligible if it is positive.

There are four main differences with the statement of the standard superposition rule: Contexts $s[\,]$ are replaced by green contexts $s\langle\,\rangle$. The standard condition $u \notin \mathcal{V}$ is generalized by conditions 2 and 3. Most general unifiers are replaced by complete sets of unifiers. And $\not\approx$ is replaced by the more precise $\not\precsim$.

The second rule is a variant of Sup that focuses on fluid green subterms:

$$\frac{\overbrace{D' \vee t \approx t'}^{D} \qquad \overbrace{C' \vee s\langle u\rangle \mathrel{\dot{\approx}} s'}^{C}}{(D' \vee C' \vee s\langle z\,t'\rangle \mathrel{\dot{\approx}} s')\sigma}\ \textsc{FluidSup}$$

with the following side conditions, in addition to SUP's conditions 5 to 9:

1. u is either a fluid term or a variable deeply occurring in C;
2. z is a fresh variable; 3. $\sigma \in \mathrm{CSU}(z\,t, u)$; 4. $(z\,t')\sigma \neq (z\,t)\sigma$.

The equality resolution and equality factoring rules are almost identical to their standard counterparts:

$$\frac{\overbrace{C' \vee u \not\approx u'}^{C}}{C'\sigma}\,\textsc{ERes} \qquad\qquad \frac{\overbrace{C' \vee u' \approx v' \vee u \approx v}^{C}}{(C' \vee v \not\approx v' \vee u \approx v')\sigma}\,\textsc{EFact}$$

For ERES: $\sigma \in \mathrm{CSU}(u, u')$ and $u \not\approx u'$ is $\gtrsim$-eligible in C w.r.t. σ. For EFACT: $\sigma \in \mathrm{CSU}(u, u')$, $u\sigma \not\lesssim v\sigma$, and $u \approx v$ is $\gtrsim$-eligible in C w.r.t. σ.

Argument congruence, a higher-order concern, is embodied by the rule

$$\frac{\overbrace{C' \vee s \approx s'}^{C}}{C'\sigma \vee s\sigma\,\bar{x}_n \approx s'\sigma\,\bar{x}_n}\,\textsc{ArgCong}$$

where σ is the most general type substitution that ensures well-typedness of the conclusion. In particular, if the result type of s is not a type variable, σ is the identity substitution; and if the result type is a type variable, it is instantiated with $\alpha_1 \to \cdots \to \alpha_m \to \beta$, where $\bar{\alpha}_m$ and β are fresh. This yields infinitely many conclusions, one for each m. The literal $s \approx s'$ must be strictly $\gtrsim$-eligible in C w.r.t. σ, and $\bar{x}_n$ is a nonempty tuple of distinct fresh variables.

The rules are complemented by the polymorphic functional extensionality axiom:

$$y\,(\mathsf{diff}\langle\alpha,\beta\rangle\,y\,z) \not\approx z\,(\mathsf{diff}\langle\alpha,\beta\rangle\,y\,z) \vee y \approx z \qquad (\textsc{Ext})$$

From now on, we will omit the type arguments to diff since they can be inferred from the term arguments.

5.3.2. Rationale for the Rules

The calculus realizes the following division of labor: SUP and FLUIDSUP are responsible for green subterms, which are outside λs, ARGCONG effectively gives access to the remaining positions outside λs, and the extensionality axiom takes care of subterms inside λs. The following examples illustrate these mechanisms. The unifiers in the examples were chosen to keep the clauses reasonably small.

Example 5.12. Prefix subterms such as g in the term g a b are not green subterms and thus cannot be superposed into. ARGCONG gives us access to those positions. Given the clauses $\mathsf{g} \approx \mathsf{f}$ and $\mathsf{g\,a\,b} \not\approx \mathsf{f\,a\,b}$, we can derive a contradiction as in example 3.5.

Example 5.13. Applied variables give rise to subtle situations with no counterparts in first-order logic. Consider the clauses $\mathsf{f\,a} \approx \mathsf{c}$ and $\mathsf{h}\,(y\,\mathsf{b})\,(y\,\mathsf{a}) \not\approx \mathsf{h}\,(\mathsf{g}\,(\mathsf{f\,b}))\,(\mathsf{g\,c})$, where $\mathsf{f\,a} \succ \mathsf{c}$. It is easy to see that the clause set is unsatisfiable, by grounding the second clause with $\theta = \{y \mapsto \lambda x.\, \mathsf{g}\,(\mathsf{f}\,x)\}$. However, to mimic the superposition inference that

can be performed at the ground level, it is necessary to superpose at an imaginary position *below* the applied variable y and yet *above* its argument a, namely, into the subterm $\mathsf{f}\,\mathsf{a}$ of $\mathsf{g}(\mathsf{f}\,\mathsf{a}) = (\lambda x.\,\mathsf{g}(\mathsf{f}\,x))\,\mathsf{a} = (y\,\mathsf{a})\theta$. We need FLUIDSUP:

$$\dfrac{\dfrac{\mathsf{f}\,\mathsf{a} \approx \mathsf{c} \quad \mathsf{h}\,(y\,\mathsf{b})\,(y\,\mathsf{a}) \not\approx \mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{b}))\,(\mathsf{g}\,\mathsf{c})}{\mathsf{h}\,(z\,(\mathsf{f}\,\mathsf{b}))\,(z\,\mathsf{c}) \not\approx \mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{b}))\,(\mathsf{g}\,\mathsf{c})}\,\textsc{FluidSup}}{\bot}\,\textsc{ERes}$$

FLUIDSUP's z variable effectively transforms $\mathsf{f}\,\mathsf{a} \approx \mathsf{c}$ into $z\,(\mathsf{f}\,\mathsf{a}) \approx z\,\mathsf{c}$, whose left-hand side can be unified with $y\,\mathsf{a}$ by taking $\{y \mapsto \lambda x.\,z\,(\mathsf{f}\,x)\}$.

Example 5.14. The clause set consisting of $\mathsf{f}\,\mathsf{a} \approx \mathsf{c}$, $\mathsf{f}\,\mathsf{b} \approx \mathsf{d}$, and $\mathsf{g}\,\mathsf{c} \not\approx y\,\mathsf{a} \vee \mathsf{g}\,\mathsf{d} \not\approx y\,\mathsf{b}$ has a similar flavor. ERES is applicable on either literal of the third clause, but the computed unifier, $\{y \mapsto \lambda x.\,\mathsf{g}\,\mathsf{c}\}$ or $\{y \mapsto \lambda x.\,\mathsf{g}\,\mathsf{d}\}$, is not the right one. Again, we need FLUIDSUP:

$$\dfrac{\mathsf{f}\,\mathsf{b} \approx \mathsf{d} \qquad \dfrac{\dfrac{\mathsf{f}\,\mathsf{a} \approx \mathsf{c} \quad \mathsf{g}\,\mathsf{c} \not\approx y\,\mathsf{a} \vee \mathsf{g}\,\mathsf{d} \not\approx y\,\mathsf{b}}{\mathsf{g}\,\mathsf{c} \not\approx z\,\mathsf{c} \vee \mathsf{g}\,\mathsf{d} \not\approx z\,(\mathsf{f}\,\mathsf{b})}\,\textsc{FluidSup}}{\mathsf{g}\,\mathsf{d} \not\approx \mathsf{g}\,(\mathsf{f}\,\mathsf{b})}\,\textsc{ERes}}{\mathsf{g}\,\mathsf{d} \not\approx \mathsf{g}\,\mathsf{d}}\,\textsc{Sup}}{\bot}\,\textsc{ERes}$$

Again, the FLUIDSUP inference uses the unifier $\{y \mapsto \lambda x.\,z\,(\mathsf{f}\,x)\} \in \mathrm{CSU}(z\,(\mathsf{f}\,\mathsf{a}), y\,\mathsf{a})$.

Example 5.15. Third-order clauses containing subterms of the form $y\,(\lambda x.\,t)$ can be even more stupefying. The clause set consisting of $\mathsf{f}\,\mathsf{a} \approx \mathsf{c}$ and $\mathsf{h}\,(y\,(\lambda x.\,\mathsf{g}\,(\mathsf{f}\,x))\,\mathsf{a})\,y \not\approx \mathsf{h}\,(\mathsf{g}\,\mathsf{c})\,(\lambda w\,x.\,w\,x)$ is unsatisfiable. To see why, apply $\theta = \{y \mapsto \lambda w\,x.\,w\,x\}$ to the second clause, yielding $\mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{a}))\,(\lambda w\,x.\,w\,x) \not\approx \mathsf{h}\,(\mathsf{g}\,\mathsf{c})\,(\lambda w\,x.\,w\,x)$. Let $\mathsf{f}\,\mathsf{a} \succ \mathsf{c}$. A SUP inference is possible between the first clause and this ground instance of the second one:

$$\dfrac{\mathsf{f}\,\mathsf{a} \approx \mathsf{c} \quad \mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{a}))\,(\lambda w\,x.\,w\,x) \not\approx \mathsf{h}\,(\mathsf{g}\,\mathsf{c})\,(\lambda w\,x.\,w\,x)}{\mathsf{h}\,(\mathsf{g}\,\mathsf{c})\,(\lambda w\,x.\,w\,x) \not\approx \mathsf{h}\,(\mathsf{g}\,\mathsf{c})\,(\lambda w\,x.\,w\,x)}\,\textsc{Sup}$$

But at the nonground level, the subterm $\mathsf{f}\,\mathsf{a}$ is not clearly localized: $\mathsf{g}(\mathsf{f}\,\mathsf{a}) = (\lambda x.\mathsf{g}(\mathsf{f}\,x))\mathsf{a} = (\lambda w\,x.\,w\,x)\,(\lambda x.\,\mathsf{g}\,(\mathsf{f}\,x))\,\mathsf{a} = (y\,(\lambda x.\,\mathsf{g}\,(\mathsf{f}\,x))\,\mathsf{a})\theta$. The FLUIDSUP rule can cope with this using the unifier $\{y \mapsto \lambda w\,x.\,w\,x, z \mapsto \mathsf{g}\} \in \mathrm{CSU}(z\,(\mathsf{f}\,\mathsf{a}),\, y\,(\lambda x.\,\mathsf{g}\,(\mathsf{f}\,x))\,\mathsf{a})$:

$$\dfrac{\dfrac{\mathsf{f}\,\mathsf{a} \approx \mathsf{c} \quad \mathsf{h}\,(y\,(\lambda x.\,\mathsf{g}\,(\mathsf{f}\,x))\,\mathsf{a})\,y \not\approx \mathsf{h}\,(\mathsf{g}\,\mathsf{c})\,(\lambda w\,x.\,w\,x)}{\mathsf{h}\,(\mathsf{g}\,\mathsf{c})\,(\lambda w\,x.\,w\,x) \not\approx \mathsf{h}\,(\mathsf{g}\,\mathsf{c})\,(\lambda w\,x.\,w\,x)}\,\textsc{FluidSup}}{\bot}\,\textsc{ERes}$$

Example 5.16. The FLUIDSUP rule is concerned not only with applied variables but also with λ-expressions that, after substitution, may be η-reduced to reveal new applied variables or green subterms. Consider the clause set consisting of $\mathsf{f}\,\mathsf{a} \approx \mathsf{c}$ and $\mathsf{h}\,(\lambda u.\,y\,u\,\mathsf{b})\,(\lambda u.\,y\,u\,\mathsf{a}) \not\approx \mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{b}))\,(\mathsf{g}\,\mathsf{c})$, where $\mathsf{f}\,\mathsf{a} \succ \mathsf{c}$. Applying the substitution $\{y \mapsto \lambda u'\,v.\,\mathsf{g}(\mathsf{f}\,v)u'\}$ to the second clause yields $\mathsf{h}(\lambda u.\,\mathsf{g}(\mathsf{f}\,\mathsf{b})u)(\lambda u.\,\mathsf{g}(\mathsf{f}\,\mathsf{a})u) \not\approx \mathsf{h}(\mathsf{g}(\mathsf{f}\,\mathsf{b}))(\mathsf{g}\,\mathsf{c})$ after β-reduction and $\mathsf{h}(\mathsf{g}(\mathsf{f}\,\mathsf{b}))(\mathsf{g}(\mathsf{f}\,\mathsf{a})) \not\approx \mathsf{h}(\mathsf{g}(\mathsf{f}\,\mathsf{b}))(\mathsf{g}\,\mathsf{c})$ after $\beta\eta$-reduction. A SUP inference is possible between the first clause and this new ground clause:

$$\dfrac{\mathsf{f}\,\mathsf{a} \approx \mathsf{c} \quad \mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{b}))\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{a})) \not\approx \mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{b}))\,(\mathsf{g}\,\mathsf{c})}{\mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{b}))\,(\mathsf{g}\,\mathsf{c}) \not\approx \mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{b}))\,(\mathsf{g}\,\mathsf{c})}\,\textsc{Sup}$$

By also considering λ-expressions, the FluidSup rule is applicable at the nonground level to derive a corresponding nonground clause using the unifier $\{y \mapsto \lambda u'\, v.\, z\,(\mathsf{f}\, v)\, u'\} \in \mathrm{CSU}(z\,(\mathsf{f}\,\mathsf{a}), \lambda u.\, y\, u\,\mathsf{a})$:

$$\dfrac{\dfrac{\mathsf{f}\,\mathsf{a} \approx \mathsf{c} \quad \mathsf{h}\,(\lambda u.\, y\, u\,\mathsf{b})\,(\lambda u.\, y\, u\,\mathsf{a}) \not\approx \mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{b}))\,(\mathsf{g}\,\mathsf{c})}{\mathsf{h}\,(z\,(\mathsf{f}\,\mathsf{b}))\,(z\,\mathsf{c}) \not\approx \mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{b}))\,(\mathsf{g}\,\mathsf{c})}\,\textsc{FluidSup}}{\bot}\,\textsc{ERes}$$

Example 5.17. Consider the clause set consisting of the facts $C_{\mathrm{succ}} = \mathsf{succ}\, x \not\approx \mathsf{zero}$, $C_{\mathrm{div}} = n \approx \mathsf{zero} \vee \mathsf{div}\, n\, n \approx \mathsf{one}$, $C_{\mathrm{prod}} = \mathsf{prod}\, K\, (\lambda k.\, \mathsf{one}) \approx \mathsf{one}$, and the negated conjecture $C_{\mathrm{conj}} = \mathsf{prod}\, K\, (\lambda k.\, \mathsf{div}\,(\mathsf{succ}\, k)\,(\mathsf{succ}\, k)) \not\approx \mathsf{one}$. Intuitively, the term $\mathsf{prod}\, K\, (\lambda k.\, u)$ is intended to denote the product $\prod_{k \in K} u$, where k ranges over a finite set K of natural numbers. The calculus derives the empty clause as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y\,(\mathsf{diff}\,\langle\alpha,\beta\rangle\, y\, z) \not\approx z\,(\mathsf{diff}\,\langle\alpha,\beta\rangle\, y\, z) \vee y \approx z}\,\textsc{Ext} \quad C_{\mathrm{div}}}{w\,(\mathsf{diff}\,\langle\alpha,\iota\rangle\,(\lambda k.\, \mathsf{div}\,(w\, k)\,(w\, k))\, z) \approx \mathsf{zero} \vee \mathsf{one} \not\approx z\,(\mathsf{diff}\,\langle\alpha,\iota\rangle\,(\lambda k.\, \mathsf{div}\,(w\, k)\,(w\, k))\, z) \vee (\lambda k.\, \mathsf{div}\,(w\, k)\,(w\, k)) \approx z}\,\textsc{FluidSup}}{w\,(\mathsf{diff}\,\langle\alpha,\iota\rangle\,(\lambda k.\, \mathsf{div}\,(w\, k)\,(w\, k))\,(\lambda k.\, \mathsf{one})) \approx \mathsf{zero} \vee (\lambda k.\, \mathsf{div}\,(w\, k)\,(w\, k)) \approx (\lambda k.\, \mathsf{one})}\,\textsc{ERes} \quad C_{\mathrm{succ}}}{\mathsf{zero} \not\approx \mathsf{zero} \vee (\lambda k.\, \mathsf{div}\,(\mathsf{succ}\, k)\,(\mathsf{succ}\, k)) \approx (\lambda k.\, \mathsf{one})}\,\textsc{Sup}}{(\lambda k.\, \mathsf{div}\,(\mathsf{succ}\, k)\,(\mathsf{succ}\, k)) \approx (\lambda k.\, \mathsf{one})}\,\textsc{ERes} \quad C_{\mathrm{conj}}}{\mathsf{prod}\, K\, (\lambda k.\, \mathsf{one}) \not\approx \mathsf{one}}\,\textsc{Sup} \quad C_{\mathrm{prod}}}{\mathsf{one} \not\approx \mathsf{one}}\,\textsc{Sup}}{\bot}\,\textsc{ERes}$$

Since the calculus does not superpose into λ-expressions, we must use the extensionality axiom to refute this clause set. We perform a FluidSup inference into the extensionality axiom with the unifier $\{\beta \mapsto \iota, z' \mapsto \lambda x.x, n \mapsto w\,(\mathsf{diff}\,\langle\alpha,\iota\rangle\,(\lambda k.\, \mathsf{div}\,(w\, k)\,(w\, k))\, z), y \mapsto \lambda k.\, \mathsf{div}\,(w\, k)\,(w\, k)\} \in \mathrm{CSU}(z'\,(\mathsf{div}\, n\, n), y\,(\mathsf{diff}\,\langle\alpha,\beta\rangle\, y\, z))$. Then we apply ERes with the unifier $\{z \mapsto \lambda k.\, \mathsf{one}\} \in \mathrm{CSU}(\mathsf{one}, z\,(\mathsf{diff}\,\langle\alpha,\iota\rangle\,(\lambda k.\, \mathsf{div}\,(w\, k)\,(w\, k))\, z))$ to eliminate the negative literal. Next, we perform a Sup inference from the resulting clause into C_{succ} with the unifier $\{\alpha \mapsto \iota, w \mapsto \mathsf{succ}, x \mapsto \mathsf{diff}\,\langle\alpha,\iota\rangle\,(\lambda k.\, \mathsf{div}\,(w\, k)\,(w\, k))\,(\lambda k.\, \mathsf{one})\} \in \mathrm{CSU}(w\,(\mathsf{diff}\,\langle\alpha,\iota\rangle\,(\lambda k.\, \mathsf{div}\,(w\, k)\,(w\, k))\,(\lambda k.\, \mathsf{one})), \mathsf{succ}\, x)$. To eliminate the trivial literal, we apply ERes. We then apply a Sup inference into C_{conj} and superpose into the resulting clause with C_{prod}. Finally we derive the empty clause by ERes.

Because it gives rise to flex–flex pairs, which are unification constraints where both sides are variable-headed, FluidSup can be very prolific. With variable-headed terms on both sides of its maximal literal, the extensionality axiom is another prime source of flex–flex pairs. Flex–flex pairs can also arise in the other rules (Sup, ERes, and EFact). Due to order restrictions and fairness, we cannot postpone solving flex–flex pairs indefinitely. Thus, we cannot use Huet's pre-unification procedure [74] and must instead choose a full unification procedure such as Jensen and Pietrzykowski's [76], Snyder and Gallier's [123], or the procedure that has recently been developed by Vukmirović, Bentkamp, and Nummelin [136]. On the positive side, optional inference rules can efficiently cover many cases where FluidSup or the extensionality axiom

would otherwise be needed (Section 5.5), and heuristics can help postpone the explosion. Moreover, flex–flex pairs are not always as bad as their reputation; for example, $y\,\mathsf{a}\,\mathsf{b} \stackrel{?}{\approx} z\,\mathsf{c}\,\mathsf{d}$ admits a most general unifier: $\{y \mapsto \lambda w\,x.\,y'\,w\,x\,\mathsf{c}\,\mathsf{d},\, z \mapsto y'\,\mathsf{a}\,\mathsf{b}\}$.

The calculus is a graceful generalization of standard superposition, except for the extensionality axiom. From simple first-order clauses, the axiom can be used to derive clauses containing λ-expressions, which are useless if the problem is first-order. For instance, the clause $\mathsf{g}\,x \approx \mathsf{f}\,x\,x$ can be used for a FLUIDSUP inference into the axiom (EXT) yielding the clause $w\,t\,(\mathsf{f}\,t\,t) \not\approx z\,t \vee (\lambda u.\,w\,u\,(\mathsf{g}u)) \approx z$ via the unifier $\{\alpha \mapsto \iota, \beta \mapsto \iota, x \mapsto t, v \mapsto \lambda u.\,w\,t\,u, y \mapsto \lambda u.\,w\,u\,(\mathsf{g}\,u)\} \in \mathrm{CSU}(v\,(\mathsf{g}x), y\,(\mathsf{diff}\,\langle\alpha,\beta\rangle\,y\,z))$ where $t = \mathsf{diff}\,\langle\iota,\iota\rangle\,(\lambda u.\,w\,u\,(\mathsf{g}\,u))\,z$, the variable w is freshly introduced by unification, and v is the fresh variable introduced by FLUIDSUP (named z in the definition of the rule). By ERES, with the unifier $\{z \mapsto \lambda u.\,w\,u\,(\mathsf{f}\,u\,u)\} \in \mathrm{CSU}(w\,t\,(\mathsf{f}\,t\,t), z\,t)$, we can then derive $(\lambda u.\,w\,u\,(\mathsf{g}\,u)) \approx (\lambda u.\,w\,u\,(\mathsf{f}\,u\,u))$, an equality of two λ-expressions, although we started with a simple first-order clause. This could be avoided if we could find a way to make the positive literal $y \approx z$ of (EXT) larger than the other literal, or to select $y \approx z$ without losing refutational completeness. The literal $y \approx z$ interacts only with green subterms of functional type, which do not arise in first-order clauses.

5

5.3.3. Soundness

To show soundness of the inferences, we need the substitution lemma for our logic:

Lemma 5.18 (Substitution lemma)**.** *Let $\mathcal{I} = (\mathcal{I}_{\mathsf{ty}}, \mathcal{J}, \mathcal{L})$ be a proper interpretation. Then*

$$[\![\tau\rho]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}} = [\![\tau]\!]^{\xi'}_{\mathcal{I}_{\mathsf{ty}}} \quad \textit{and} \quad [\![t\rho]\!]^{\xi}_{\mathcal{I}} = [\![t]\!]^{\xi'}_{\mathcal{I}}$$

for all terms t, all types τ, and all substitutions ρ, where $\xi'(\alpha) = [\![\alpha\rho]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}$ for all type variables α and $\xi'(x) = [\![x\rho]\!]^{\xi}_{\mathcal{I}}$ for all term variables x.

Proof. First, we prove that $[\![\tau\rho]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}} = [\![\tau]\!]^{\xi'}_{\mathcal{I}_{\mathsf{ty}}}$ by induction on the structure of τ. If $\tau = \alpha$ is a type variable,

$$[\![\alpha\rho]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}} = \xi'(\alpha) = [\![\alpha]\!]^{\xi'}_{\mathcal{I}_{\mathsf{ty}}}$$

If $\tau = \kappa(\bar{\upsilon})$ for some type constructor κ and types $\bar{\upsilon}$,

$$[\![\kappa(\bar{\upsilon})\rho]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}} = \mathcal{J}_{\mathsf{ty}}(\kappa)([\![\bar{\upsilon}\rho]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}) \overset{\text{IH}}{=} \mathcal{J}_{\mathsf{ty}}(\kappa)([\![\bar{\upsilon}]\!]^{\xi'}_{\mathcal{I}_{\mathsf{ty}}}) = [\![\kappa(\bar{\upsilon})]\!]^{\xi'}_{\mathcal{I}_{\mathsf{ty}}}$$

Next, we prove $[\![t\rho]\!]^{\xi}_{\mathcal{I}} = [\![t]\!]^{\xi'}_{\mathcal{I}}$ by induction on the structure of a λ-term representative of t, allowing arbitrary substitutions ρ in the induction hypothesis. If $t = y$, then by the definition of the denotation of a variable

$$[\![y\rho]\!]^{\xi}_{\mathcal{I}} = \xi'(y) = [\![y]\!]^{\xi'}_{\mathcal{I}}$$

If $t = \mathsf{f}\langle\bar{\tau}\rangle$, then by the definition of the term denotation

$$[\![\mathsf{f}\langle\bar{\tau}\rangle\rho]\!]^{\xi}_{\mathcal{I}} = \mathcal{J}(\mathsf{f}, [\![\bar{\tau}\rho]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}) \overset{\text{IH}}{=} \mathcal{J}(\mathsf{f}, [\![\bar{\tau}]\!]^{\xi'}_{\mathcal{I}_{\mathsf{ty}}}) = [\![\mathsf{f}\langle\bar{\tau}\rangle]\!]^{\xi'}_{\mathcal{I}}$$

If $t = u\,v$, then by the definition of the term denotation

$$[\![(u\,v)\rho]\!]^{\xi}_{\mathcal{I}} = [\![u\rho]\!]^{\xi}_{\mathcal{I}}([\![v\rho]\!]^{\xi}_{\mathcal{I}}) \overset{\text{IH}}{=} [\![u]\!]^{\xi'}_{\mathcal{I}}([\![v]\!]^{\xi'}_{\mathcal{I}}) = [\![u\,v]\!]^{\xi'}_{\mathcal{I}}$$

If $t = \lambda z.\, u$, let $\rho'(z) = z$ and $\rho'(x) = \rho(x)$ for $x \neq z$. Using properness of $\mathcal{I}$ in the second and the last step, we have

$$[\![(\lambda z.\, u)\rho]\!]^{\xi}_{\mathcal{I}}(a) = [\![(\lambda z.\, u\rho')]\!]^{\xi}_{\mathcal{I}}(a) = [\![u\rho']\!]^{\xi[z\mapsto a]}_{\mathcal{I}} \overset{\text{IH}}{=} [\![u]\!]^{\xi'[z\mapsto a]}_{\mathcal{I}} = [\![\lambda z.\, u]\!]^{\xi'}_{\mathcal{I}}(a) \qquad \square$$

Lemma 5.19. *If $\mathcal{I} \models C$ for some interpretation $\mathcal{I}$ and some clause C, then $\mathcal{I} \models C\rho$ for all substitutions ρ.*

Proof. We have to show that $C\rho$ is true in $\mathcal{I}$ for all valuations ξ. Given a valuation ξ, define ξ' as in Lemma 5.18. Then, by Lemma 5.18, a literal in $C\rho$ is true in $\mathcal{I}$ for ξ if and only if the corresponding literal in C is true in $\mathcal{I}$ for ξ'. There must be at least one such literal because $\mathcal{I} \models C$ and hence C is in particular true in $\mathcal{I}$ for ξ'. Therefore, $C\rho$ is true in $\mathcal{I}$ for ξ. $\square$

Theorem 5.20 (Soundness)**.** *The inference rules* Sup, FluidSup, ERes, EFact, *and* ArgCong *are sound (even without the variable condition and the side conditions on fluidity, deeply occurring variables, order, and eligibility).*

Proof. We fix an inference and an interpretation $\mathcal{I}$ that is a model of the premises. We need to show that it is also a model of the conclusion.

From the definition of the denotation of a term, it is obvious that congruence holds in our logic, at least for subterms that are not inside a λ-expression. In particular, it holds for green subterms and for the left subterm t of an application $t\,s$.

By Lemma 5.19, $\mathcal{I}$ is a model of the σ-instances of the premises as well, where σ is the substitution used for the inference. Let ξ be a valuation. By making case distinctions on the truth under $\mathcal{I}, \xi$ of the literals of the σ-instances of the premises, using the conditions that σ is a unifier, and applying congruence, it follows that the conclusion is true under $\mathcal{I}, \xi$. Hence, $\mathcal{I}$ is a model of the conclusion. $\square$

As in the λ-free higher-order logic of Chapter 3, skolemization is unsound in our logic. As a consequence, axiom (Ext) does not hold in all interpretations, but the axiom is consistent with our logic, i.e., there exist models of (Ext).

5.3.4. The Redundancy Criterion

As in Chapter 3, we define our redundancy criterion via an encoding $\mathcal{F}$ into ground monomorphic first-order logic. $\mathcal{F}$ indexes each symbol occurrence with the type arguments and the number of term arguments. For example, $\mathcal{F}(\mathsf{f}\,\mathsf{a}) = \mathsf{f}_1(\mathsf{a}_0)$ and $\mathcal{F}(\mathsf{g}\langle\kappa\rangle) = \mathsf{g}^{\kappa}_0$. In addition, $\mathcal{F}$ conceals λ-expressions by replacing them with fresh symbols. These measures effectively disable argument congruence and extensionality. For example, the clause sets $\{\mathsf{g} \approx \mathsf{f}, \mathsf{g}\,\mathsf{a} \not\approx \mathsf{f}\,\mathsf{a}\}$ and $\{\mathsf{b} \approx \mathsf{a}, (\lambda x.\, \mathsf{b}) \not\approx (\lambda x.\, \mathsf{a})\}$ are unsatisfiable in higher-order logic, but the encoded clause sets $\{\mathsf{g}_0 \approx \mathsf{f}_0, \mathsf{g}_1(\mathsf{a}_0) \not\approx \mathsf{f}_1(\mathsf{a}_0)\}$ and $\{\mathsf{b}_0 \approx \mathsf{a}_0, \mathsf{lam}_{\lambda x.\, \mathsf{b}} \not\approx \mathsf{lam}_{\lambda x.\, \mathsf{a}}\}$ are satisfiable in first-order logic, where $\mathsf{lam}_{\lambda x.\, t}$ is a family of fresh symbols.

Given a higher-order signature $(\Sigma_{\text{ty}}, \Sigma)$, we define a ground first-order signature $(\Sigma_{\text{ty}}, \Sigma_{\text{GF}})$ as follows. The type constructors Σ_{ty} are the same in both signatures, but $\to$ is uninterpreted in first-order logic. For each ground instance $\mathsf{f}\langle\bar{\upsilon}\rangle : \tau_1 \to \cdots \to \tau_n \to \tau$ of a symbol $\mathsf{f} \in \Sigma$, we introduce a first-order symbol $\mathsf{f}^{\bar{\upsilon}}_j \in \Sigma_{\text{GF}}$ with argument

types $\bar{\tau}_j$ and return type $\tau_{j+1} \to \cdots \to \tau_n \to \tau$, for each j. Moreover, for each ground term $\lambda x.\, t$, we introduce a symbol $\mathsf{lam}_{\lambda x.\, t} \in \Sigma_{\mathrm{GF}}$ of the same type.

Thus, we again consider three levels of logics: the higher-order level H over a given signature $(\Sigma_{\mathrm{ty}}, \Sigma)$, the ground higher-order level GH, which is the ground fragment of H, and the ground monomorphic first-order level GF over the signature $(\Sigma_{\mathrm{ty}}, \Sigma_{\mathrm{GF}})$ defined above. We use $\mathcal{T}_{\mathrm{H}}$, $\mathcal{T}_{\mathrm{GH}}$, and $\mathcal{T}_{\mathrm{GF}}$ to denote the respective sets of terms, $\mathcal{T}y_{\mathrm{H}}$, $\mathcal{T}y_{\mathrm{GH}}$, and $\mathcal{T}y_{\mathrm{GF}}$ to denote the respective sets of types, and $\mathcal{C}_{\mathrm{H}}$, $\mathcal{C}_{\mathrm{GH}}$, and $\mathcal{C}_{\mathrm{GF}}$ to denote the respective sets of clauses. Each of the three levels has an entailment relation $\models$. A clause set N_1 entails a clause set N_2, denoted $N_1 \models N_2$, if every model of N_1 is also a model of N_2. For H and GH, we use higher-order models; for GF, we use first-order models.

As in Chapter 3, the three levels are connected by two functions $\mathcal{G}$ and $\mathcal{F}$:

Definition 5.21 (Grounding function $\mathcal{G}$ on terms and clauses)**.** The grounding function $\mathcal{G}$ maps terms $t \in \mathcal{T}_{\mathrm{H}}$ to the set of their ground instances—i.e., the set of all $t\theta \in \mathcal{T}_{\mathrm{GH}}$ where θ is a substitution. It also maps clauses $C \in \mathcal{C}_{\mathrm{H}}$ to the set of their ground instances—i.e., the set of all $C\theta \in \mathcal{C}_{\mathrm{GH}}$ where θ is a substitution.

5

Definition 5.22 (Encoding $\mathcal{F}$ on terms and clauses)**.** The encoding $\mathcal{F} : \mathcal{T}_{\mathrm{GH}} \to \mathcal{T}_{\mathrm{GF}}$ is recursively defined as

$$\mathcal{F}(\lambda x.\, t) = \mathsf{lam}_{\lambda x.\, t} \qquad\qquad \mathcal{F}(\mathsf{f}\langle \bar{v} \rangle \bar{s}_j) = \mathsf{f}_j^{\bar{v}}(\mathcal{F}(\bar{s}_j))$$

using η-short β-normal representatives of terms. The encoding $\mathcal{F}$ is extended to map from $\mathcal{C}_{\mathrm{GH}}$ to $\mathcal{C}_{\mathrm{GF}}$ by mapping each literal and each side of a literal individually.

The schematic overview of the three levels also applies to our new construction:

$$\underset{\text{higher-order}}{\mathrm{H}} \xrightarrow{\ \mathcal{G}\ } \underset{\text{ground higher-order}}{\mathrm{GH}} \xrightarrow{\ \mathcal{F}\ } \underset{\text{ground first-order}}{\mathrm{GF}}$$

The mapping $\mathcal{F}$ is clearly bijective. Using the inverse mapping, the order $\succ$ can be transferred from $\mathcal{T}_{\mathrm{GH}}$ to $\mathcal{T}_{\mathrm{GF}}$ and from $\mathcal{C}_{\mathrm{GH}}$ to $\mathcal{C}_{\mathrm{GF}}$ by defining $t \succ s$ as $\mathcal{F}^{-1}(t) \succ \mathcal{F}^{-1}(s)$ and $C \succ D$ as $\mathcal{F}^{-1}(C) \succ \mathcal{F}^{-1}(D)$. The property that $\succ$ on clauses is the multiset extension of $\succ$ on literals, which in turn is the multiset extension of $\succ$ on terms, is maintained because $\mathcal{F}^{-1}$ maps the multiset representations elementwise.

For example, let $C = y\,\mathsf{b} \approx y\,\mathsf{a} \vee y \not\approx \mathsf{f}\,\mathsf{a} \in \mathcal{C}_{\mathrm{H}}$. Then $\mathcal{G}(C)$ contains, among many other clauses, $C\theta = \mathsf{f}\,\mathsf{b}\,\mathsf{b} \approx \mathsf{f}\,\mathsf{a}\,\mathsf{a} \vee (\lambda x.\mathsf{f}\,x\,x) \not\approx \mathsf{f}\,\mathsf{a} \in \mathcal{C}_{\mathrm{GH}}$, where $\theta = \{y \mapsto \lambda x.\mathsf{f}\,x\,x\}$. On the GF level, this clause corresponds to $\mathcal{F}(C\theta) = \mathsf{f}_2(\mathsf{b}_0, \mathsf{b}_0) \approx \mathsf{f}_2(\mathsf{a}_0, \mathsf{a}_0) \vee \mathsf{lam}_{\lambda x.\mathsf{f}\,x\,x} \not\approx \mathsf{f}_1(\mathsf{a}_0) \in \mathcal{C}_{\mathrm{GF}}$.

A key property of $\mathcal{F}$ is that green subterms in $\mathcal{T}_{\mathrm{GH}}$ correspond to subterms in $\mathcal{T}_{\mathrm{GF}}$. This allows us to show that well-foundedness, totality on ground terms, compatibility with contexts, and the subterm property hold for $\succ$ on $\mathcal{T}_{\mathrm{GF}}$.

Lemma 5.23. *Let $s, t \in \mathcal{T}_{\mathrm{GH}}$. We have $\mathcal{F}(t\langle s \rangle_p) = \mathcal{F}(t)[\mathcal{F}(s)]_p$. In other words, s is a green subterm of t at position p if and only if $\mathcal{F}(s)$ is a subterm of $\mathcal{F}(t)$ at position p.*

Proof. Analogous to Lemma 3.18. □

Lemma 5.24. *Well-foundedness, totality, compatibility with contexts, and the subterm property hold for* $\succ$ *in* $\mathcal{T}_{\mathrm{GF}}$.

Proof. Analogous to Lemma 3.20, using Lemma 5.23. □

The saturation procedures of superposition provers aggressively delete clauses that are strictly subsumed by other clauses. A clause C *subsumes* D if there exists a substitution σ such that $C\sigma \subseteq D$. A clause C *strictly subsumes* D if C subsumes D but D does not subsume C. For example, $x \approx \mathsf{c}$ strictly subsumes both $\mathsf{a} \approx \mathsf{c}$ and $\mathsf{b} \not\approx \mathsf{a} \vee x \approx \mathsf{c}$. The proof of refutational completeness of resolution and superposition provers relies on the well-foundedness of the strict subsumption relation. In the λ-free higher-order logic used in Chapter 3, strict subsumption is well founded. Unfortunately, this property does not hold for this chapter's logic, where $\mathsf{f}\,x\,x \approx \mathsf{c}$ is strictly subsumed by $\mathsf{f}\,(x\,\mathsf{a})\,(x\,\mathsf{b}) \approx \mathsf{c}$, which is strictly subsumed by $\mathsf{f}\,(x\,\mathsf{a}\,\mathsf{a}')\,(x\,\mathsf{b}\,\mathsf{b}') \approx \mathsf{c}$, and so on. To prevent such infinite chains, we use a well-founded partial order $\sqsupset$ on $\mathcal{C}_{\mathrm{H}}$. We can define $\sqsupset$ as $\gtrsim \cap >_{\mathrm{size}}$, where $\gtrsim$ stands for "subsumed by" and $D >_{\mathrm{size}} C$ if either $\mathit{size}(D) > \mathit{size}(C)$ or $\mathit{size}(D) = \mathit{size}(C)$ and D contains fewer distinct variables than C; the *size* function is some notion of syntactic size, such as the number of constants and variables contained in a clause. This yields for instance $\mathsf{a} \approx \mathsf{c} \sqsupset x \approx \mathsf{c}$ and $\mathsf{f}\,(x\,\mathsf{a}\,\mathsf{a}) \approx \mathsf{c} \sqsupset \mathsf{f}\,(y\,\mathsf{a}) \approx \mathsf{c}$. To justify the deletion of subsumed clauses, we set up our redundancy criterion to cover subsumption, following Waldmann et al. [140].

Based on these slightly altered definitions, we define clause redundancy verbatim as in Chapter 3:

- Given $C \in \mathcal{C}_{\mathrm{GF}}$ and $N \subseteq \mathcal{C}_{\mathrm{GF}}$, let $C \in \mathit{GFRed}_{\mathrm{C}}(N)$ if $\{D \in N \mid D \prec C\} \models C$.
- Given $C \in \mathcal{C}_{\mathrm{GH}}$ and $N \subseteq \mathcal{C}_{\mathrm{GH}}$, let $C \in \mathit{GHRed}_{\mathrm{C}}(N)$ if $\mathcal{F}(C) \in \mathit{GFRed}_{\mathrm{C}}(\mathcal{F}(N))$.
- Given $C \in \mathcal{C}_{\mathrm{H}}$ and $N \subseteq \mathcal{C}_{\mathrm{H}}$, let $C \in \mathit{HRed}_{\mathrm{C}}(N)$ if for every $D \in \mathcal{G}(C)$, we have $D \in \mathit{GHRed}_{\mathrm{C}}(\mathcal{G}(N))$ or there exists $C' \in N$ such that $C \sqsupset C'$ and $D \in \mathcal{G}(C')$.

For example, $(\mathsf{h}\,\mathsf{g})x \approx (\mathsf{h}\,\mathsf{f})x$ is redundant w.r.t. $\mathsf{g} \approx \mathsf{f}$, but $\mathsf{g}x \approx \mathsf{f}x$ and $(\lambda x.\,\mathsf{g}) \approx (\lambda x.\,\mathsf{f})$ are not, because $\mathcal{F}$ translates an unapplied g to g_0, whereas an applied g is translated to g_1 and the expression $\lambda x.\,\mathsf{g}$ is translated to $\mathsf{lam}_{\lambda x.\,\mathsf{g}}$. These different translations prevent entailment on the GF level. For an example of subsumption, we assume that $\mathsf{a} \approx \mathsf{c} \sqsupset x \approx \mathsf{c}$ holds, for instance using the above definition of $\sqsupset$. Then $\mathsf{a} \approx \mathsf{c}$ is redundant w.r.t. $x \approx \mathsf{c}$.

Along with the three levels of logics, we consider three inference systems, as in Chapter 3: *HInf*, *GHInf*, and *GFInf*. *HInf* is the inference system described in Section 5.3.1. For uniformity, we regard the extensionality axiom as a premise-free inference rule Ext whose conclusion is axiom (Ext). The rules of *GHInf* include Sup, ERes, and EFact from *HInf*, but with the restriction that premises and conclusion are ground and with all references to $\gtrsim$ replaced by $\succeq$. In addition, *GHInf* contains a premise-free rule GExt whose infinitely many conclusions are the ground instances of (Ext), and the following ground variant of ArgCong:

$$\frac{C' \vee s \approx s'}{C' \vee s\,\bar{u}_n \approx s'\,\bar{u}_n}\ \textsc{GArgCong}$$

where $s \approx s'$ is strictly $\succeq$-eligible in $C' \vee s \approx s'$ and $\bar{u}_n$ is a nonempty tuple of ground terms.

GFInf contains all SUP, ERES, and EFACT inferences from *GHInf* translated by $\mathcal{F}$. It coincides with standard first-order superposition.

Each of the three inference systems is parameterized by a selection function. For *HInf*, we globally fix one selection function *HSel*. For *GHInf* and *GFInf*, we need to consider different selection functions. We write $\mathit{GHInf}^{\mathit{GHSel}}$ for *GHInf* and $\mathit{GFInf}^{\mathit{GFSel}}$ for *GFInf* to make the dependency on the respective selection functions *GHSel* and *GFSel* explicit. Let $\mathcal{G}(\mathit{HSel})$ denote the set of all selection functions on $\mathcal{C}_{\mathrm{GH}}$ such that for each clause in $C \in \mathcal{C}_{\mathrm{GH}}$, there exists a clause $D \in \mathcal{C}_{\mathrm{H}}$ with $C \in \mathcal{G}(D)$ and corresponding selected literals. For each selection function *GHSel* on $\mathcal{C}_{\mathrm{GH}}$, via the bijection $\mathcal{F}$, we obtain a corresponding selection function on $\mathcal{C}_{\mathrm{GF}}$, which we denote by $\mathcal{F}(\mathit{GHSel})$.

We extend the functions $\mathcal{F}$ and $\mathcal{G}$ to inferences:

Notation 5.25. Given an inference ι, we write $\mathit{prems}(\iota)$ for the tuple of premises, $\mathit{mprem}(\iota)$ for the main (i.e., rightmost) premise, and $\mathit{concl}(\iota)$ for the conclusion.

5

Definition 5.26 (Encoding $\mathcal{F}$ on inferences)**.** Given a SUP, ERES, or EFACT inference $\iota \in \mathit{GHInf}$, let $\mathcal{F}(\iota) \in \mathit{GFInf}$ denote the inference defined by $\mathit{prems}(\mathcal{F}(\iota)) = \mathcal{F}(\mathit{prems}(\iota))$ and $\mathit{concl}(\mathcal{F}(\iota)) = \mathcal{F}(\mathit{concl}(\iota))$.

Definition 5.27 (Grounding function $\mathcal{G}$ on inferences)**.** Given an inference $\iota \in \mathit{HInf}$, and a selection function $\mathit{GHSel} \in \mathcal{G}(\mathit{HSel})$, we define the set $\mathcal{G}^{\mathit{GHSel}}(\iota)$ of ground instances of ι to be all inferences $\iota' \in \mathit{GHInf}^{\mathit{GHSel}}$ such that $\mathit{prems}(\iota') = \mathit{prems}(\iota)\theta$ and $\mathit{concl}(\iota') = \mathit{concl}(\iota)\theta$ for some grounding substitution θ.

This will map SUP and FLUIDSUP to SUP, EFACT to EFACT, ERES to ERES, EXT to GEXT, and ARGCONG to GARGCONG inferences, but it is also possible that $\mathcal{G}^{\mathit{GHSel}}(\iota)$ is the empty set for some inferences ι.

We define the sets of redundant inferences w.r.t. a given clause set as for the nonpurifying calculi in Chapter 3:

- Given $\iota \in \mathit{GFInf}^{\mathit{GFSel}}$ and $N \subseteq \mathcal{C}_{\mathrm{GF}}$, let $\iota \in \mathit{GFRed}_{\mathrm{I}}^{\mathit{GFSel}}(N)$ if we have $\mathit{prems}(\iota) \cap \mathit{GFRed}_{\mathrm{C}}(N) \neq \varnothing$ or $\{D \in N \mid D \prec \mathit{mprem}(\iota)\} \models \mathit{concl}(\iota)$.
- Given $\iota \in \mathit{GHInf}^{\mathit{GHSel}}$ and $N \subseteq \mathcal{C}_{\mathrm{GH}}$, let $\iota \in \mathit{GHRed}_{\mathrm{I}}^{\mathit{GHSel}}(N)$ if
  - ι is not a GARGCONG or GEXT inference and $\mathcal{F}(\iota) \in \mathit{GFRed}_{\mathrm{I}}^{\mathcal{F}(\mathit{GHSel})}(\mathcal{F}(N))$; or
  - ι is a GARGCONG or GEXT inference and $\mathit{concl}(\iota) \in N \cup \mathit{GHRed}_{\mathrm{C}}(N)$.
- Given $\iota \in \mathit{HInf}$ and $N \subseteq \mathcal{C}_{\mathrm{H}}$, let $\iota \in \mathit{HRed}_{\mathrm{I}}(N)$ if $\mathcal{G}^{\mathit{GHSel}}(\iota) \subseteq \mathit{GHRed}_{\mathrm{I}}(\mathcal{G}(N))$ for all $\mathit{GHSel} \in \mathcal{G}(\mathit{HSel})$.

Occasionally, we omit the selection function in the notation when it is irrelevant. A clause set N is *saturated* w.r.t. an inference system and the inference component $\mathit{Red}_{\mathrm{I}}$ of a redundancy criterion if every inference from clauses in N is in $\mathit{Red}_{\mathrm{I}}(N)$.

This redundancy criterion gracefully generalizes the usual first-order redundancy criterion and thus most of the simplification rules implemented in Schulz's first-order prover E [117, Sections 2.3.1 and 2.3.2] can be applied, with the same caveats as listed in Section 3.3.5.

5.3.5. A Derived Term Order

We stated some requirements on the term orders $\succ$ and $\succsim$ in Section 5.3.1 but have not shown how to fulfill them. To derive a suitable strict term order $\succ$, we propose to encode η-short β-normal forms into untyped first-order terms and apply an order $\succ_{\mathsf{fo}}$ of first-order terms such as the Knuth–Bendix order [87] or the lexicographic path order [82].

The encoding, denoted by $\mathcal{O}$, indexes symbols with their number of term arguments, similarly to the $\mathcal{F}$ encoding. Unlike the $\mathcal{F}$ encoding, $\mathcal{O}$ translates $\lambda x\colon\tau.\, t$ to $\mathsf{lam}(\mathcal{O}(\tau),\mathcal{O}(t))$ and uses De Bruijn [43] symbols to represent bound variables. The $\mathcal{O}$ encoding replaces fluid terms t by fresh variables z_t and maps type arguments to term arguments, while erasing any other type information. For example, $\mathcal{O}(\lambda x\colon\kappa.\,\mathsf{f}\,(\mathsf{f}\,(\mathsf{a}\langle\kappa\rangle))\,(y\,\mathsf{b})) = \mathsf{lam}(\kappa,\mathsf{f}_2(\mathsf{f}_1(\mathsf{a}_0(\kappa)),z_{y\mathsf{b}}))$. The use of De Bruijn indices and the monolithic encoding of fluid terms ensure stability under both α-renaming and substitution.

Definition 5.28 (Encoding $\mathcal{O}$)**.** Given a signature $(\Sigma_{\mathsf{ty}},\Sigma)$, $\mathcal{O}$ encodes types and terms as terms over the untyped first-order signature $\Sigma_{\mathsf{ty}} \uplus \{\mathsf{f}_k \mid \mathsf{f}\in\Sigma,\, k\in\mathbb{N}\} \uplus \{\mathsf{lam}\} \uplus \{\mathsf{db}_k^i \mid i,k\in\mathbb{N}\}$. We reuse higher-order type variables as term variables in the target untyped first-order logic. Moreover, let z_t be an untyped first-order variable for each higher-order term t. The auxiliary function $\mathcal{B}_x(t)$ replaces each free occurrence of the variable x by a symbol db^i, where i is the number of λ-expressions surrounding the variable occurrence. The type-to-term version of $\mathcal{O}$ is defined by $\mathcal{O}(\alpha)=\alpha$ and $\mathcal{O}(\kappa(\bar\tau))=\kappa(\mathcal{O}(\bar\tau))$. The term-to-term version is defined by

$$\mathcal{O}(t) = \begin{cases} z_t & \text{if } t = x \text{ or } t \text{ is fluid}\\ \mathsf{lam}(\mathcal{O}(\tau),\mathcal{O}(\mathcal{B}_x(u))) & \text{if } t=(\lambda x\colon\tau.\,u) \text{ and } t \text{ is not fluid}\\ \mathsf{f}_k(\mathcal{O}(\bar\tau),\mathcal{O}(\bar u_k)) & \text{if } t = \mathsf{f}\langle\bar\tau\rangle\,\bar u_k \end{cases}$$

For example, let $s=\lambda y.\,\mathsf{f}\,y\,(\lambda w.\,\mathsf{g}\,(y\,w))$ where y has type $\kappa\to\kappa$ and w has type κ. We have $\mathcal{B}_y(\mathsf{f}\,y(\lambda w.\mathsf{g}(yw))) = \mathsf{f}\mathsf{db}^0(\lambda w.\mathsf{g}(\mathsf{db}^1 w))$ and $\mathcal{B}_w(\mathsf{g}(\mathsf{db}^1 w)) = \mathsf{g}(\mathsf{db}^1\mathsf{db}^0)$. Neither s nor $\lambda w.\,\mathsf{g}\,(y\,w)$ are fluid. Hence, $\mathcal{O}(s) = \mathsf{lam}(\to(\kappa,\kappa),\mathsf{f}_2(\mathsf{db}_0^0,\mathsf{lam}(\kappa,\mathsf{g}_1(\mathsf{db}_1^1(\mathsf{db}_0^0)))))$.

Definition 5.29 (Derived strict term order)**.** Let the strict term order derived from $\succ_{\mathsf{fo}}$ be $\succ_\lambda$ where $t\succ_\lambda s$ if $\mathcal{O}(t)\succ_{\mathsf{fo}}\mathcal{O}(s)$.

We will show that the derived $\succ_\lambda$ fulfills all properties of a strict term order (Definition 5.6) if $\succ_{\mathsf{fo}}$ fulfills the corresponding properties on first-order terms. For the nonstrict term order $\succsim$, we can use the reflexive closure $\succeq_\lambda$ of $\succ_\lambda$.

Lemma 5.30. *Let $\succ_{\mathsf{fo}}$ be a strict partial order on first-order terms and $\succ_\lambda$ the derived term order on $\beta\eta$-equivalence classes. If the restriction of $\succ_{\mathsf{fo}}$ to ground terms enjoys well-foundedness, totality, the subterm property, and compatibility with contexts (w.r.t. first-order terms), the restriction of $\succ_\lambda$ to ground terms enjoys well-foundedness, totality, the green subterm property, and compatibility with green contexts (w.r.t. $\beta\eta$-equivalence classes).*

Proof. Transitivity and irreflexivity of $\succ_{\mathsf{fo}}$ imply transitivity and irreflexivity of $\succ_\lambda$.

WELL-FOUNDEDNESS: If there existed an infinite chain $t_1 \succ_\lambda t_2 \succ_\lambda \cdots$ of ground terms, there would also be the chain $O(t_1) \succ_{\mathsf{fo}} O(t_2) \succ_{\mathsf{fo}} \cdots$, contradicting the well-foundedness of $\succ_{\mathsf{fo}}$ on ground λ-free terms.

TOTALITY: By ground totality of $\succ_{\mathsf{fo}}$, for any ground terms t and s we have $O(t) \succ_{\mathsf{fo}} O(s)$, $O(t) \prec_{\mathsf{fo}} O(s)$, or $O(t) = O(s)$. In the first two cases, it follows that $t \succ_\lambda s$ or $t \prec_\lambda s$. In the last case, it follows that $t = s$ because O is clearly injective.

GREEN SUBTERM PROPERTY: Let s be a term. We show that $s \succeq_\lambda s|_p$ by induction on p, where $s|_p$ denotes the green subterm at position p. If $p = \varepsilon$, this is trivial. If $p = p'.i$, we have $s \succeq_\lambda s|_{p'}$ by the induction hypothesis. Hence, it suffices to show that $s|_{p'} \succeq_\lambda s|_{p'.i}$. From the existence of the position $p'.i$, we know that $s|_{p'}$ must be of the form $s|_{p'} = \mathsf{f}\langle\bar{\tau}\rangle\,\bar{u}_k$. Then $s|_{p'.i} = u_i$. The encoding yields $O(s|_{p'}) = \mathsf{f}_k(O(\bar{\tau}), O(\bar{u}_k))$ and hence $O(s|_{p'}) \succeq_{\mathsf{fo}} O(s|_{p'.i})$ by the ground subterm property of $\succ_{\mathsf{fo}}$. Hence, $s|_{p'} \succeq_\lambda s|_{p'.i}$ and thus $s \succeq_\lambda s|_p$.

COMPATIBILITY WITH GREEN CONTEXTS: By induction on the depth of the context, it suffices to show that $t \succ_\lambda s$ implies $\mathsf{f}\langle\bar{\tau}\rangle\,\bar{u}\,t\,\bar{v} \succ_\lambda \mathsf{f}\langle\bar{\tau}\rangle\,\bar{u}\,s\,\bar{v}$ for all t, s, f, $\bar{\tau}$, $\bar{u}$, and $\bar{v}$. This amounts to showing that $O(t) \succ_{\mathsf{fo}} O(s)$ implies $O(\mathsf{f}\langle\bar{\tau}\rangle\,\bar{u}\,t\,\bar{v}) = \mathsf{f}_k(O(\bar{\tau}), O(\bar{u}), O(t), O(\bar{v})) \succ_{\mathsf{fo}} \mathsf{f}_k(O(\bar{\tau}), O(\bar{u}), O(s), O(\bar{v})) = O(\mathsf{f}\langle\bar{\tau}\rangle\,\bar{u}\,s\,\bar{v})$, which follows directly from ground compatibility of $\succ_{\mathsf{fo}}$ with contexts and the induction hypothesis. □

Lemma 5.31. *Let $\succ_{\mathsf{fo}}$ be a strict partial order on first-order terms. If $\succ_{\mathsf{fo}}$ is stable under grounding substitutions (w.r.t. first-order terms), the derived term order $\succ_\lambda$ is stable under grounding substitutions (w.r.t. $\beta\eta$-equivalence classes).*

Proof. Assume $s \succ_\lambda s'$ for some terms s and s'. Let θ be a higher-order substitution grounding s and s'. We must show $s\theta \succ_\lambda s'\theta$. We will define a first-order substitution ρ grounding $O(s)$ and $O(s')$ such that $O(s)\rho = O(s\theta)$ and $O(s')\rho = O(s'\theta)$. Since $s \succ_\lambda s'$, we have $O(s) \succ_{\mathsf{fo}} O(s')$. By stability of $\succ_{\mathsf{fo}}$ under grounding substitutions, $O(s)\rho \succ_{\mathsf{fo}} O(s')\rho$. It follows that $O(s\theta) \succ_{\mathsf{fo}} O(s'\theta)$ and hence $s\theta \succ_\lambda s'\theta$.

We define the first-order substitution ρ as $\alpha\rho = \alpha\theta$ for type variables α and $z_u\rho = O(u\theta)$ for terms u. Strictly speaking, the domain of a substitution must be finite, so we restrict this definition of ρ to the finitely many variables that occur in the computation of $O(s)$ and $O(s')$.

Clearly $O(\tau)\rho = O(\tau\theta)$ for all types τ occurring in the computation of $O(s)$ and $O(s')$. Moreover, $O(t)\rho = O(t\theta)$ for all t occurring in the computation of $O(s)$ and $O(s')$, which we show by induction on the definition of the encoding. If $t = x$ or if t is fluid, $O(t)\rho = z_t\rho = O(t\theta)$. If $t = \mathsf{f}\langle\bar{\tau}\rangle\,\bar{u}$, then $O(t)\rho = \mathsf{f}_k(O(\bar{\tau})\rho, O(\bar{u})\rho) \overset{\text{IH}}{=} \mathsf{f}_k(O(\bar{\tau}\theta), O(\bar{u}\theta)) = O(\mathsf{f}\langle\bar{\tau}\theta\rangle(\bar{u}\theta)) = O(t\theta)$. If $t = (\lambda x\mathbin{:}\tau.\ u)$ and t is not fluid, then $O(t)\rho = \mathsf{lam}(O(\tau)\rho, O(\mathcal{B}_x(u))\rho) \overset{\text{IH}}{=} \mathsf{lam}(O(\tau\theta), O(\mathcal{B}_x(u)\theta)) = \mathsf{lam}(O(\tau\theta), O(\mathcal{B}_x(u)\theta[x \mapsto x])) = O(\lambda x\mathbin{:}\tau\theta.\ u\theta[x \mapsto x]) = O((\lambda x\mathbin{:}\tau.\ u)\theta) = O(t\theta)$. □

5.4. Refutational Completeness

Using the same general structure as in Chapter 3, we prove static and dynamic refutational completeness of *HInf* w.r.t. $(\mathit{HRed}_{\mathrm{I}}, \mathit{HRed}_{\mathrm{C}})$.

5.4.1. Outline of the Proof

The proof proceeds in three steps, corresponding to the three levels GF, GH, and H introduced in Section 5.3.4:

1. We use Bachmair and Ganzinger's work on the refutational completeness of standard (first-order) superposition [9] to prove static refutational completeness of *GFInf*.
2. From the first-order model constructed in Bachmair and Ganzinger's proof, we derive a clausal higher-order model and thus prove static refutational completeness of *GHInf*.
3. We use the saturation framework by Waldmann et al. [140] to lift the static refutational completeness of *GHInf* to static and dynamic refutational completeness of *HInf*.

The first step is identical with the first step of the completeness proof in Chapter 3. We will omit it here and refer to Section 3.4.2.

In the second step, we derive refutational completeness of *GHInf*. Given a saturated clause set $N \subseteq \mathcal{C}_{\mathrm{GH}}$ with $\bot \notin N$, we use the first-order model $R_{\mathcal{F}(N)}$ of $\mathcal{F}(N)$ constructed in the first step to derive a clausal higher-order interpretation that is a model of N. Under the encoding $\mathcal{F}$, occurrences of the same symbol with different numbers of arguments are regarded as different symbols—e.g., $\mathcal{F}(\mathsf{f}) = \mathsf{f}_0$ and $\mathcal{F}(\mathsf{f}\ \mathsf{a}) = \mathsf{f}_1(\mathsf{a}_0)$. All λ-expressions $\lambda x.\, t$ are regarded as uninterpreted symbols $\mathsf{lam}_{\lambda x.\, t}$. The difficulty is to construct a higher-order interpretation that merges the first-order denotations of all f_i into a single higher-order denotation of f and to show that the symbols $\mathsf{lam}_{\lambda x.\, t}$ behave like $\lambda x.\, t$. This step relies on saturation w.r.t. the GArgCong rule—which connects a term of functional type with its value when applied to an argument x—and on the presence of the extensionality rule GExt.

In the third step, we employ the saturation framework by Waldmann et al. [140]. The main proof obligation we must discharge to use the framework is that there should exist nonground inferences in *HInf* corresponding to all nonredundant inferences in *GHInf*. We face two specifically higher-order difficulties. First, since our term order lacks compatibility with contexts, we need to argue as in Chapter 3 that Sup into variables can make up for that flaw of the term order. The other difficulty also concerns applied variables. We must show that any nonredundant Sup inference in level GH into a position corresponding to a fluid term or a deeply occurring variable in level H can be lifted to a FluidSup inference. This involves showing that the z variable in FluidSup can represent arbitrary contexts around a term t.

For the entire proof of refutational completeness, $\beta\eta$-normalization is the proverbial dog that did not bark. On level GH, the rules Sup, ERes, and EFact preserve η-short β-normal form, and so does first-order term rewriting. Thus, we can completely ignore $\longrightarrow_\beta$ and $\longrightarrow_\eta$. On level H, instantiation can cause β- and η-reduction, but this poses no difficulties thanks to the clause order's stability under grounding substitutions.

5.4.2. The Ground Higher-Order Level

Since the refutational completeness proof for the ground first-order level GF is identical to the one in Section 3.4.2, we skip directly to the GH level.

In this subsection, let *GHSel* be a selection function on $\mathcal{C}_{\mathrm{GH}}$, let $N \subseteq \mathcal{C}_{\mathrm{GH}}$ be a clause set saturated w.r.t. $\mathit{GHInf}^{\mathit{GHSel}}$ and $\mathit{GHRed}_{\mathrm{I}}^{\mathit{GHSel}}$ such that $\bot \notin N$. Clearly, $\mathcal{F}(N)$ is then saturated w.r.t. $\mathit{GFInf}^{\mathcal{F}(\mathit{GHSel})}$ and $\mathit{GFRed}_{\mathrm{I}}^{\mathcal{F}(\mathit{GHSel})}$.

We abbreviate $R_{\mathcal{F}(N)}$ as R. Given two terms $s, t \in \mathcal{T}_{\mathrm{GH}}$, we write $s \sim t$ to abbreviate $R \models \mathcal{F}(s) \approx \mathcal{F}(t)$, which is equivalent to $[\![\mathcal{F}(s)]\!]_R = [\![\mathcal{F}(t)]\!]_R$.

Lemma 5.32. *For all terms $t, s : \tau \to \upsilon$ in $\mathcal{T}_{\mathrm{GH}}$, the following statements are equivalent:*

1. *$t \sim s$;*
2. *$t\,(\mathsf{diff}\,t\,s) \sim s\,(\mathsf{diff}\,t\,s)$;*
3. *$t\,u \sim s\,u$ for all $u \in \mathcal{T}_{\mathrm{GH}}$.*

Proof. (3) ⇒ (2): Take $u := \mathsf{diff}\,t\,s$.

(2) ⇒ (1): Since N is saturated, the GExt inference that generates the clause $C = t\,(\mathsf{diff}\,t\,s) \not\approx s\,(\mathsf{diff}\,t\,s) \vee t \approx s$ is redundant—i.e., $C \in N \cup \mathit{GHRed}_{\mathrm{C}}(N)$—and hence $R \models \mathcal{F}(C)$ by Theorem 3.29 and the assumption that $\bot \notin N$. Therefore, it follows from $t\,(\mathsf{diff}\,t\,s) \sim s\,(\mathsf{diff}\,t\,s)$ that $t \sim s$.

(1) ⇒ (3): We assume that $t \sim s$—i.e., $\mathcal{F}(t) \leftrightarrow_R^* \mathcal{F}(s)$. By induction on the number of rewrite steps between $\mathcal{F}(t)$ and $\mathcal{F}(s)$ and by transitivity of $\sim$, it suffices to show that $\mathcal{F}(t) \to_R \mathcal{F}(s)$ implies $t\,u \sim s\,u$. If the rewrite step $\mathcal{F}(t) \to_R \mathcal{F}(s)$ is not at the top level, then neither $s{\downarrow_{\beta\eta}}$ nor $t{\downarrow_{\beta\eta}}$ can be λ-expressions. Therefore, $(s{\downarrow_{\beta\eta}})(u{\downarrow_{\beta\eta}})$ and $(t{\downarrow_{\beta\eta}})(u{\downarrow_{\beta\eta}})$ are in η-short β-normal form, and there is an analogous rewrite step $\mathcal{F}(t\,u) \to_R \mathcal{F}(s\,u)$ using the same rewrite rule. It follows that $t\,u \sim s\,u$. If the rewrite step $\mathcal{F}(t) \to_R \mathcal{F}(s)$ is at the top level, $\mathcal{F}(t) \to \mathcal{F}(s)$ must be a rule of R. This rule must originate from a productive clause of the form $\mathcal{F}(C) = \mathcal{F}(C' \vee t \approx s)$. By Lemma 3.28, $\mathcal{F}(t \approx s)$ is strictly $\succeq$-eligible in $\mathcal{F}(C)$ w.r.t. $\mathcal{F}(\mathit{GHSel})$, and hence $t \approx s$ is strictly $\succeq$-eligible in C w.r.t. *GHSel*. Thus, the following GArgCong inference ι is applicable:

$$\frac{C' \vee t \approx s}{C' \vee t\,u \approx s\,u}\,\textsc{GArgCong}$$

By saturation, ι is redundant w.r.t. N—i.e., $\mathit{concl}(\iota) \in N \cup \mathit{GHRed}_{\mathrm{C}}(N)$. By Theorem 3.29 and the assumption that $\bot \notin N$, $\mathcal{F}(\mathit{concl}(\iota))$ is then true in R. By Lemma 3.28, $\mathcal{F}(C')$ is false in R. Therefore, $\mathcal{F}(t\,u \approx s\,u)$ must be true in R. □

Lemma 5.33. *Let $s \in \mathcal{T}_{\mathrm{H}}$ and θ, θ' grounding substitutions such that $x\theta \sim x\theta'$ for all variables x and $\alpha\theta = \alpha\theta'$ for all type variables α. Then $s\theta \sim s\theta'$.*

Proof. In this proof, we work directly on λ-terms. To prove the lemma, it suffices to prove it for any λ-term s. Here, for λ-terms t_1 and t_2, the notation $t_1 \sim t_2$ is to be read as $t_1{\downarrow_{\beta\eta}} \sim t_2{\downarrow_{\beta\eta}}$ because $\mathcal{F}$ is only defined on η-short β-normal terms.

Definition We extend the syntax of λ-terms with a new polymorphic function symbol $\oplus : \Pi\alpha.\, \alpha \to \alpha \to \alpha$. We will omit its type argument. It is equipped with two

reduction rules: $\oplus\, t\, s \longrightarrow t$ and $\oplus\, t\, s \longrightarrow s$. A *$\beta\oplus$-reduction step* is either a rewrite step following one of these rules or a β-reduction step.

The computability path order $\succ_{\mathsf{CPO}}$ [36] guarantees that

- $\oplus\, t\, s \succ_{\mathsf{CPO}} s$ by applying rule @▷;
- $\oplus\, t\, s \succ_{\mathsf{CPO}} t$ by applying rule @▷ twice;
- $(\lambda x.\, t)\, s \succ_{\mathsf{CPO}} t[x \mapsto s]$ by applying rule @β.

Since this order is moreover monotone, it decreases with $\beta\oplus$-reduction steps. The order is also well founded; thus, $\beta\oplus$-reductions terminate. And since the $\beta\oplus$-reduction steps describe a finitely branching term rewriting system, by Kőnig's lemma [85], there is a maximal number of $\beta\oplus$-reduction steps from each λ-term.

DEFINITION A λ-term is *term-ground* if it does not contain free term variables. It may contain polymorphic type arguments.

DEFINITION We introduce an auxiliary function $\mathcal{S}$ that essentially measures the size of a λ-term but assigns a size of 1 to term-ground λ-terms.

$$\mathcal{S}(s) = \begin{cases} 1 & \text{if } s \text{ is term-ground or is a bound or free variable or a symbol} \\ 1 + \mathcal{S}(t) & \text{if } s \text{ is not term-ground and has the form } \lambda x.\, t \\ \mathcal{S}(t) + \mathcal{S}(u) & \text{if } s \text{ is not term-ground and has the form } t\, u \end{cases}$$

We prove $s\theta \sim s\theta'$ by well-founded induction on s, θ, and θ' using the left-to-right lexicographic order on the triple $\big(n_1(s), n_2(s), n_3(s)\big) \in \mathbb{N}^3$, where

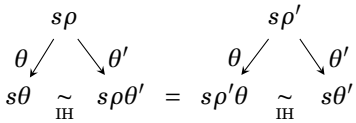
- $n_1(s)$ is the maximal number of $\beta\oplus$-reduction steps starting from $s\sigma$, where σ is the substitution mapping each term variable x to $\oplus\, x\theta\, x\theta'$;
- $n_2(s)$ is the number of free term variables occurring more than once in s;
- $n_3(s) = \mathcal{S}(s)$.

CASE 1: The λ-term s is term-ground. Then the lemma is trivial.

CASE 2: The λ-term s contains $k \geq 2$ free term variables. Then we can apply the induction hypothesis twice and use the transitivity of $\sim$ as follows. Let x be one of the free term variables in s. Let $\rho = \{x \mapsto x\theta\}$ the substitution that maps x to $x\theta$ and ignores all other variables. Let $\rho' = \theta'[x \mapsto x]$.

We want to invoke the induction hypothesis on $s\rho$ and $s\rho'$. This is justified because $s\sigma$ $\oplus$-reduces to $s\rho\sigma$ and to $s\rho'\sigma$. These $\oplus$-reductions have at least one step because x occurs in s and $k \geq 2$. Hence, $n_1(s) > n_1(s\rho)$ and $n_1(s) > n_1(s\rho')$.

This application of the induction hypothesis gives us $s\rho\theta \sim s\rho\theta'$ and $s\rho'\theta \sim s\rho'\theta'$. Since $s\rho\theta = s\theta$ and $s\rho'\theta' = s\theta'$, this is equivalent to $s\theta \sim s\rho\theta'$ and $s\rho'\theta \sim s\theta'$. Since moreover $s\rho\theta' = s\rho'\theta$, we have $s\theta \sim s\theta'$ by transitivity of $\sim$. The following illustration visualizes the above argument:



CASE 3: The λ-term s contains a free term variable that occurs more than once. Then we rename variable occurrences apart by replacing each occurrence of each free term variable x by a fresh variable x_i, for which we define $x_i\theta = x\theta$ and $x_i\theta' = x\theta'$. Let s' be the resulting λ-term. Since $s\sigma = s'\sigma$, we have $n_1(s) = n_1(s')$. All free term variables occur only once in s'. Hence, $n_2(s) > 0 = n_2(s')$. Therefore, we can invoke the induction hypothesis on s' to obtain $s'\theta \sim s'\theta'$. Since $s\theta = s'\theta$ and $s\theta' = s'\theta'$, it follows that $s\theta \sim s\theta'$.

CASE 4: The λ-term s contains only one free term variable x, which occurs exactly once.

CASE 4.1: The λ-term s is of the form $\mathsf{f}\langle\bar{\tau}\rangle\,\bar{t}$ for some symbol f, some types $\bar{\tau}$, and some λ-terms $\bar{t}$. Then let u be the λ-term in $\bar{t}$ that contains x. We want to apply the induction hypothesis to u, which can be justified as follows. Consider the longest sequence of $\beta\oplus$-reductions from $u\sigma$. This sequence can be replicated inside $s\sigma = (\mathsf{f}\langle\bar{\tau}\rangle\bar{t})\sigma$. Therefore, the longest sequence of $\beta\oplus$-reductions from $s\sigma$ is at least as long—i.e., $n_1(s) \geq n_1(u)$. Since both s and u have only one free term variable occurrence, we have $n_2(s) = 0 = n_2(u)$. But $n_3(s) > n_3(u)$ because u is a term-nonground subterm of s.

Applying the induction hypothesis gives us $u\theta \sim u\theta'$. By definition of $\mathcal{F}$, we have $\mathcal{F}((\mathsf{f}\langle\bar{\tau}\rangle\,\bar{t})\theta) = \mathsf{f}_m^{\bar{\tau}\theta}\,\mathcal{F}(\bar{t}\theta)$ and analogously for θ', where m is the length of $\bar{t}$. By congruence of $\approx$ in first-order logic, it follows that $s\theta \sim s\theta'$.

CASE 4.2: The λ-term s is of the form $x\bar{t}$ for some λ-terms $\bar{t}$. Then we observe that, by assumption, $x\theta \sim x\theta'$. By applying Lemma 5.32 repeatedly, we have $x\theta\,\bar{t} \sim x\theta'\,\bar{t}$. Since x occurs only once, $\bar{t}$ is term-ground and hence $s\theta = x\theta\,\bar{t}$ and $s\theta' = x\theta'\,\bar{t}$. Therefore, $s\theta \sim s\theta'$.

CASE 4.3: The λ-term s is of the form $\lambda z.\,u$ for some λ-term u. Then we observe that to prove $s\theta \sim s\theta'$, it suffices to show that $s\theta\,(\mathsf{diff}\,s\theta\,s\theta') \sim s\theta'\,(\mathsf{diff}\,s\theta\,s\theta')$ by Lemma 5.32. Via $\beta\eta$-conversion, this is equivalent to $u\rho\theta \sim u\rho\theta'$ where $\rho = \{z \mapsto \mathsf{diff}\,(s\theta\downarrow_{\beta\eta})(s\theta'\downarrow_{\beta\eta})\}$. To prove $u\rho\theta \sim u\rho\theta'$, we apply the induction hypothesis on $u\rho$.

It remains to show that the induction hypothesis is applicable on $u\rho$. Consider the longest sequence of $\beta\oplus$-reductions from $u\rho\sigma$. Since $z\rho$ starts with the diff symbol, $z\rho$ will not cause more $\beta\oplus$-reductions than z. Hence, the same sequence of $\beta\oplus$-reductions can be applied inside $s\sigma = (\lambda z.\,u)\sigma$, proving that $n_1(s) \geq n_1(u\rho)$. Since both s and $u\rho$ have only one free term variable occurrence, $n_2(s) = 0 = n_2(u\rho)$. But $n_3(s) = \mathcal{S}(s) = 1 + \mathcal{S}(u)$ because s is term-nonground. Moreover, $\mathcal{S}(u) \geq \mathcal{S}(u\rho) = n_3(u\rho)$ because ρ replaces a variable by a ground λ-term. Hence, $n_3(s) > n_3(u\rho)$, which justifies the application of the induction hypothesis.

CASE 4.4: The λ-term s is of the form $(\lambda z.\,u)\,t_0\,\bar{t}$ for some λ-terms u, t_0, and $\bar{t}$. We apply the induction hypothesis on $s' = u\{z \mapsto t_0\}\,\bar{t}$. To justify it, consider the longest sequence of $\beta\oplus$-reductions from $s'\sigma$. Prepending the reduction $s\sigma \longrightarrow_\beta s'\sigma$ to it gives us a longer sequence from $s\sigma$. Hence, $n_1(s) > n_1(s')$. The induction hypothesis gives us $s'\theta \sim s'\theta'$. Since $\sim$ is invariant under β-reductions, it follows that $s\theta \sim s\theta'$. □

We proceed by defining a higher-order interpretation $\mathcal{I}^{\mathrm{GH}} = (\mathcal{U}^{\mathrm{GH}}, \mathcal{J}^{\mathrm{GH}}_{\mathsf{ty}}, \mathcal{J}^{\mathrm{GH}}, \mathcal{L}^{\mathrm{GH}})$ derived from R. The interpretation R is an interpretation in monomorphic first-order logic. Let $\mathcal{U}_\tau$ be its universe for type τ and $\mathcal{J}$ its interpretation function.

To illustrate the construction, we will employ the following running example. Let the higher-order signature be $\Sigma_{\mathsf{ty}} = \{\iota, \to\}$ and $\Sigma = \{\mathsf{f} : \iota \to \iota, \mathsf{a} : \iota, \mathsf{b} : \iota\}$. The first-order signature accordingly consists of Σ_{ty} and $\Sigma_{\mathrm{GF}} = \{\mathsf{f}_0, \mathsf{f}_1, \mathsf{a}_0, \mathsf{b}_0\} \cup \{\mathsf{lam}_{\lambda x.\, t} \mid \lambda x.\, t \in \mathcal{T}_{\mathrm{GH}}\}$. We write $[t]$ for the equivalence class of $t \in \mathcal{T}_{\mathrm{GF}}$ modulo R. We assume that $[\mathsf{f}_0] = [\mathsf{lam}_{\lambda x.\, x}]$, $[\mathsf{a}_0] = [\mathsf{f}_1(\mathsf{a}_0)]$, $[\mathsf{b}_0] = [\mathsf{f}_1(\mathsf{b}_0)]$, and that f_0, $\mathsf{lam}_{\lambda x.\, \mathsf{a}}$, $\mathsf{lam}_{\lambda x.\, \mathsf{b}_0}$, a_0, and b_0 are in disjoint equivalence classes. Hence, $\mathcal{U}_{\iota\to\iota} = \{[\mathsf{f}_0], [\mathsf{lam}_{\lambda x.\, \mathsf{a}}], [\mathsf{lam}_{\lambda x.\, \mathsf{b}}], \ldots\}$ and $\mathcal{U}_\iota = \{[\mathsf{a}_0], [\mathsf{b}_0]\}$.

When defining the universe $\mathcal{U}^{\mathrm{GH}}$ of the higher-order interpretation, we need to ensure that it contains subsets of function spaces, since $\mathcal{J}^{\mathrm{GH}}_{\mathsf{ty}}(\to)(\mathcal{D}_1, \mathcal{D}_2)$ must be a subset of the function space from $\mathcal{D}_1$ to $\mathcal{D}_2$ for all $\mathcal{D}_1, \mathcal{D}_2 \in \mathcal{U}^{\mathrm{GH}}$. But the first-order universes $\mathcal{U}_\tau$ consist of equivalence classes of terms from $\mathcal{T}_{\mathrm{GF}}$ w.r.t. the rewriting system R, not of functions.

To repair this mismatch, we will define a family of functions $\mathcal{E}_\tau$ that give a meaning to the elements of the first-order universes $\mathcal{U}_\tau$. We will define a domain $\mathcal{D}_\tau$ for each ground type τ and then let $\mathcal{U}^{\mathrm{GH}}$ be the set of all these domains $\mathcal{D}_\tau$. Thus, there will be a one-to-one correspondence between ground types and domains. Since the higher-order and first-order type signatures are identical (including $\to$, which is uninterpreted in first-order logic), we can identify higher-order and first-order types.

We define $\mathcal{E}_\tau$ and $\mathcal{D}_\tau$ in a mutual recursion and prove that $\mathcal{E}_\tau$ is a bijection simultaneously. We start with nonfunctional types τ: Let $\mathcal{D}_\tau = \mathcal{U}_\tau$ and let $\mathcal{E}_\tau : \mathcal{U}_\tau \longrightarrow \mathcal{D}_\tau$ be the identity. We proceed by defining $\mathcal{E}_{\tau\to\upsilon}$ and $\mathcal{D}_{\tau\to\upsilon}$. We assume that $\mathcal{E}_\tau$, $\mathcal{E}_\upsilon$, $\mathcal{D}_\tau$, and $\mathcal{D}_\upsilon$ have already been defined and that $\mathcal{E}_\tau$, $\mathcal{E}_\upsilon$ are bijections. To ensure that $\mathcal{E}_{\tau\to\upsilon}$ will be bijective, we first define an injective function $\mathcal{E}^0_{\tau\to\upsilon} : \mathcal{U}_{\tau\to\upsilon} \longrightarrow (\mathcal{D}_\tau \longrightarrow \mathcal{D}_\upsilon)$, define $\mathcal{D}_{\tau\to\upsilon}$ as its image $\mathcal{E}^0_{\tau\to\upsilon}(\mathcal{U}_{\tau\to\upsilon})$, and finally define $\mathcal{E}_{\tau\to\upsilon}$ as $\mathcal{E}^0_{\tau\to\upsilon}$ with its codomain restricted to $\mathcal{D}_{\tau\to\upsilon}$:

$$\mathcal{E}^0_{\tau\to\upsilon} : \mathcal{U}_{\tau\to\upsilon} \longrightarrow (\mathcal{D}_\tau \longrightarrow \mathcal{D}_\upsilon)$$
$$\mathcal{E}^0_{\tau\to\upsilon}(\llbracket \mathcal{F}(s) \rrbracket_R)\big(\mathcal{E}_\tau(\llbracket \mathcal{F}(u) \rrbracket_R)\big) = \mathcal{E}_\upsilon(\llbracket \mathcal{F}(s\,u) \rrbracket_R)$$

This is a valid definition because each element of $\mathcal{U}_{\tau\to\upsilon}$ is of the form $\llbracket \mathcal{F}(s) \rrbracket_R$ for some s and each element of $\mathcal{D}_\tau$ is of the form $\mathcal{E}_\tau(\llbracket \mathcal{F}(u) \rrbracket_R)$ for some u. This function is well defined if it does not depend on the choice of s and u. To show this, we assume that there are other ground terms t and v such that $\llbracket \mathcal{F}(s) \rrbracket_R = \llbracket \mathcal{F}(t) \rrbracket_R$ and $\mathcal{E}_\tau(\llbracket \mathcal{F}(u) \rrbracket_R) = \mathcal{E}_\tau(\llbracket \mathcal{F}(v) \rrbracket_R)$. Since $\mathcal{E}_\tau$ is bijective, we have $\llbracket \mathcal{F}(u) \rrbracket_R = \llbracket \mathcal{F}(v) \rrbracket_R$. Using the $\sim$-notation, we can write this as $u \sim v$. Applying Lemma 5.33 to the term $x\,y$ and the substitutions $\{x \mapsto s, y \mapsto u\}$ and $\{x \mapsto t, y \mapsto v\}$, we obtain $s\,u \sim t\,v$—i.e., $\llbracket \mathcal{F}(s\,u) \rrbracket_R = \llbracket \mathcal{F}(t\,v) \rrbracket_R$. Thus, $\mathcal{E}^0_{\tau\to\upsilon}$ is well defined. It remains to show that $\mathcal{E}^0_{\tau\to\upsilon}$ is injective as a function from $\mathcal{U}_{\tau\to\upsilon}$ to $\mathcal{D}_\tau \longrightarrow \mathcal{D}_\upsilon$. Assume two terms $s, t \in \mathcal{T}_{\mathrm{GH}}$ such that for all $u \in \mathcal{T}_{\mathrm{GH}}$, we have $\llbracket \mathcal{F}(s\,u) \rrbracket_R = \llbracket \mathcal{F}(t\,u) \rrbracket_R$. By Lemma 5.32, it follows that $\llbracket \mathcal{F}(s) \rrbracket_R = \llbracket \mathcal{F}(t) \rrbracket_R$, which concludes the proof that $\mathcal{E}^0_{\tau\to\upsilon}$ is injective.

We define $\mathcal{D}_{\tau\to\upsilon} = \mathcal{E}^0_{\tau\to\upsilon}(\mathcal{U}_{\tau\to\upsilon})$ and $\mathcal{E}_{\tau\to\upsilon}(a) = \mathcal{E}^0_{\tau\to\upsilon}(a)$. This ensures that $\mathcal{E}_{\tau\to\upsilon}$ is bijective and concludes the inductive definition of $\mathcal{D}$ and $\mathcal{E}$. In the following, we will usually write $\mathcal{E}$ instead of $\mathcal{E}_\tau$, since the type τ is determined by the first argument of $\mathcal{E}_\tau$.

In our running example, we thus have $\mathcal{D}_\iota = \mathcal{U}_\iota = \{[\mathsf{a}_0], [\mathsf{b}_0]\}$ and $\mathcal{E}_\iota$ is the identity $\mathcal{U}_\iota \longrightarrow \mathcal{D}_\iota$, $c \mapsto c$. The function $\mathcal{E}^0_{\iota\to\iota}$ maps $[\mathsf{f}_0]$ to the identity $\mathcal{D}_\iota \longrightarrow \mathcal{D}_\iota$, $c \mapsto c$; it maps

$[\mathsf{lam}_{\lambda x.\,\mathsf{a}}]$ to the constant function $\mathcal{D}_\iota \longrightarrow \mathcal{D}_\iota,\ c \mapsto [\mathsf{a}_0]$; and it maps $[\mathsf{lam}_{\lambda x.\,\mathsf{b}}]$ to the constant function $\mathcal{D}_\iota \longrightarrow \mathcal{D}_\iota,\ c \mapsto [\mathsf{b}_0]$. The swapping function $[\mathsf{a}_0] \mapsto [\mathsf{b}_0], [\mathsf{b}_0] \mapsto [\mathsf{a}_0]$ is not in the image of $\mathcal{E}^0_{\iota\to\iota}$. Therefore, $\mathcal{D}_{\iota\to\iota}$ contains only the identity and the two constant functions, but not this swapping function.

We define the higher-order universe as $\mathcal{U}^{\mathrm{GH}} = \{\mathcal{D}_\tau \mid \tau \text{ ground}\}$. Moreover, we define $\mathcal{J}^{\mathrm{GH}}_{\mathrm{ty}}(\kappa)(\mathcal{D}_{\bar\tau}) = \mathcal{U}_{\kappa(\bar\tau)}$ for all $\kappa \in \Sigma_{\mathrm{ty}}$, completing the type interpretation $\mathcal{I}^{\mathrm{GH}}_{\mathrm{ty}} = (\mathcal{U}^{\mathrm{GH}}, \mathcal{J}^{\mathrm{GH}}_{\mathrm{ty}})$. We define the interpretation function as $\mathcal{J}^{\mathrm{GH}}(\mathsf{f}, \mathcal{D}_{\bar v_m}) = \mathcal{E}(\mathcal{J}(\mathsf{f}_0^{\bar v_m}))$ for all $\mathsf{f} : \Pi\bar\alpha_m.\,\tau$.

In our example, we thus have $\mathcal{J}^{\mathrm{GH}}(\mathsf{f}) = \mathcal{E}([\mathsf{f}_0])$, which is the identity on $\mathcal{D}_\iota \longrightarrow \mathcal{D}_\iota$.

Finally, we need to define the designation function $\mathcal{L}^{\mathrm{GH}}$, which takes a valuation ξ and a λ-expression as arguments. Given a valuation ξ, we choose a grounding substitution θ such that $\mathcal{D}_{\alpha\theta} = \xi(\alpha)$ and $\mathcal{E}([\![\mathcal{F}(x\theta)]\!]_R) = \xi(x)$ for all type variables α and all variables x. Such a substitution can be constructed as follows: We can fulfill the first equation in a unique way because there is a one-to-one correspondence between ground types and domains. Since $\mathcal{E}^{-1}(\xi(x))$ is an element of a first-order universe and R is term-generated, there exists a ground term t such that $[\![t]\!]^\xi_R = \mathcal{E}^{-1}(\xi(x))$. Choosing one such t and defining $x\theta = \mathcal{F}^{-1}(t)$ gives us a grounding substitution θ with the desired property.

We define $\mathcal{L}^{\mathrm{GH}}(\xi, (\lambda x.\,t)) = \mathcal{E}([\![\mathcal{F}((\lambda x.\,t)\theta)]\!]_R)$. To prove that this is well defined, we assume that there exists another substitution θ' with the properties $\mathcal{D}_{\alpha\theta'} = \xi(\alpha)$ for all α and $\mathcal{E}([\![\mathcal{F}(x\theta')]\!]_R) = \xi(x)$ for all x. Then we have $\alpha\theta = \alpha\theta'$ for all α due to the one-to-one correspondence between domains and ground types. We have $[\![\mathcal{F}(x\theta)]\!]_R = [\![\mathcal{F}(x\theta')]\!]_R$ for all x because $\mathcal{E}$ is injective. By Lemma 5.33 it follows that $[\![\mathcal{F}((\lambda x.\,t)\theta)]\!]_R = [\![\mathcal{F}((\lambda x.\,t)\theta')]\!]_R$, which proves that $\mathcal{L}^{\mathrm{GH}}$ is well defined.

In our example, for all ξ we have $\mathcal{L}^{\mathrm{GH}}(\xi, \lambda x.\,x) = \mathcal{E}([\mathsf{lam}_{\lambda x.\,x}]) = \mathcal{E}([\mathsf{f}_0])$, which is the identity. If $\xi(y) = [\mathsf{a}_0]$, then $\mathcal{L}^{\mathrm{GH}}(\xi, \lambda x.\,y) = \mathcal{E}([\mathsf{lam}_{\lambda x.\,\mathsf{a}}])$, which is the constant function $c \mapsto [\mathsf{a}_0]$. Similarly, if $\xi(y) = [\mathsf{b}_0]$, then $\mathcal{L}^{\mathrm{GH}}(\xi, \lambda x.\,y)$ is the constant function $c \mapsto [\mathsf{b}_0]$.

This concludes the definition of the interpretation $\mathcal{I}^{\mathrm{GH}} = (\mathcal{U}^{\mathrm{GH}}, \mathcal{J}^{\mathrm{GH}}_{\mathrm{ty}}, \mathcal{J}^{\mathrm{GH}}, \mathcal{L}^{\mathrm{GH}})$. It remains to show that $\mathcal{I}^{\mathrm{GH}}$ is proper. In a proper interpretation, the denotation $[\![t]\!]_{\mathcal{I}^{\mathrm{GH}}}$ of a term t does not depend on the representative of t modulo $\beta\eta$, but since we have not yet shown $\mathcal{I}^{\mathrm{GH}}$ to be proper, we cannot rely on this property. For this reason, we use λ-terms in the following three lemmas and mark all $\beta\eta$-reductions explicitly.

The higher-order interpretation $\mathcal{I}^{\mathrm{GH}}$ relates to the first-order interpretation R as follows:

Lemma 5.34. *Given a ground λ-term t, we have*

$$[\![t]\!]_{\mathcal{I}^{\mathrm{GH}}} = \mathcal{E}([\![\mathcal{F}(t\downarrow_{\beta\eta})]\!]_R)$$

Proof. By induction on t. Assume that $[\![s]\!]_{\mathcal{I}^{\mathrm{GH}}} = \mathcal{E}([\![\mathcal{F}(s\downarrow_{\beta\eta})]\!]_R)$ for all proper subterms s of t. If t is of the form $\mathsf{f}\langle\bar\tau\rangle$, then

$$\begin{aligned}
[\![t]\!]_{\mathcal{I}^{\mathrm{GH}}} &= \mathcal{J}^{\mathrm{GH}}(\mathsf{f}, \mathcal{D}_{\bar\tau}) \\
&= \mathcal{E}(\mathcal{J}(\mathsf{f}_0, \mathcal{U}_{\mathcal{F}(\bar\tau)})) \\
&= \mathcal{E}([\![\mathsf{f}_0\langle\mathcal{F}(\bar\tau)\rangle]\!]_R) \\
&= \mathcal{E}([\![\mathcal{F}(\mathsf{f}\langle\bar\tau\rangle)]\!]_R) \\
&= \mathcal{E}([\![\mathcal{F}(\mathsf{f}\langle\bar\tau\rangle\downarrow_{\beta\eta})]\!]_R) = \mathcal{E}([\![\mathcal{F}(t\downarrow_{\beta\eta})]\!]_R)
\end{aligned}$$

If t is an application $t = t_1\, t_2$, where t_1 is of type $\tau \to \upsilon$, then

$$\begin{aligned}[\![t_1\, t_2]\!]_{\mathcal{J}^{\mathrm{GH}}} &= [\![t_1]\!]_{\mathcal{J}^{\mathrm{GH}}}([\![t_2]\!]_{\mathcal{J}^{\mathrm{GH}}}) \\ &\overset{\mathrm{IH}}{=} \mathcal{E}_{\tau\to\upsilon}([\![\mathcal{F}(t_1\downarrow_{\beta\eta})]\!]_R)(\mathcal{E}_\tau([\![\mathcal{F}(t_2\downarrow_{\beta\eta})]\!]_R)) \\ &\overset{\mathrm{Def}\,\mathcal{E}}{=} \mathcal{E}_\upsilon([\![\mathcal{F}((t_1\, t_2)\downarrow_{\beta\eta})]\!]_R)\end{aligned}$$

If t is a λ-expression, then

$$\begin{aligned}[\![\lambda x.\, u]\!]^\xi_{\mathcal{J}^{\mathrm{GH}}} &= \mathcal{L}^{\mathrm{GH}}(\xi, (\lambda x.\, u)) \\ &= \mathcal{E}([\![\mathcal{F}((\lambda x.\, u)\theta\downarrow_{\beta\eta})]\!]_R) \\ &= \mathcal{E}([\![\mathcal{F}((\lambda x.\, u)\downarrow_{\beta\eta})]\!]_R)\end{aligned}$$

where θ is a substitution such that $\mathcal{D}_{\alpha\theta} = \xi(\alpha)$ and $\mathcal{E}([\![\mathcal{F}(x\theta)]\!]_R) = \xi(x)$. □

We need to show that the interpretation $\mathcal{J}^{\mathrm{GH}} = (\mathcal{U}^{\mathrm{GH}}, \mathcal{J}^{\mathrm{GH}}_{\mathrm{ty}}, \mathcal{J}^{\mathrm{GH}}, \mathcal{L}^{\mathrm{GH}})$ is proper. In the proof, we will need to employ the following lemma, which is very similar to the substitution lemma (Lemma 5.18), but we must prove it here for our particular interpretation $\mathcal{J}^{\mathrm{GH}}$ because we have not shown that $\mathcal{J}^{\mathrm{GH}}$ is proper yet.

Lemma 5.35 (Substitution lemma)**.** *$[\![\tau\rho]\!]^\xi_{\mathcal{J}^{\mathrm{GH}}_{\mathrm{ty}}} = [\![\tau]\!]^{\xi'}_{\mathcal{J}^{\mathrm{GH}}_{\mathrm{ty}}}$ and $[\![t\rho]\!]^\xi_{\mathcal{J}^{\mathrm{GH}}} = [\![t]\!]^{\xi'}_{\mathcal{J}^{\mathrm{GH}}}$ for all λ-terms t, all $\tau \in \mathcal{T}y_{\mathrm{H}}$ and all grounding substitutions ρ, where $\xi'(\alpha) = [\![\alpha\rho]\!]^\xi_{\mathcal{J}^{\mathrm{GH}}_{\mathrm{ty}}}$ for all type variables α and $\xi'(x) = [\![x\rho]\!]^\xi_{\mathcal{J}^{\mathrm{GH}}}$ for all term variables x.*

Proof. We proceed by induction on the structure of τ and t. The proof is identical to the one of Lemma 5.18, except for the last step, which uses properness of the interpretation, a property we cannot assume here. However, here, we have the assumption that ρ is a grounding substitution. Therefore, if t is a λ-expression, we argue as follows:

$$\begin{aligned}&[\![(\lambda z.\, u)\rho]\!]^\xi_{\mathcal{J}^{\mathrm{GH}}} \\ &= [\![(\lambda z.\, u\rho')]\!]^\xi_{\mathcal{J}^{\mathrm{GH}}} && \text{where } \rho'(z) = z \text{ and } \rho'(x) = \rho(x) \text{ for } x \neq z \\ &= \mathcal{L}^{\mathrm{GH}}(\xi, (\lambda z.\, u\rho')) && \text{by the definition of the term denotation} \\ &= \mathcal{E}([\![\mathcal{F}((\lambda z.\, u)\rho\theta\downarrow_{\beta\eta})]\!]^\xi_R) && \text{for some } \theta \text{ by the definition of } \mathcal{L}^{\mathrm{GH}} \\ &= \mathcal{E}([\![\mathcal{F}((\lambda z.\, u)\rho\downarrow_{\beta\eta})]\!]^\xi_R) && \text{because } (\lambda z.\, u)\rho \text{ is ground} \\ &\overset{*}{=} \mathcal{L}^{\mathrm{GH}}(\xi', \lambda z.\, u) && \text{by the definition of } \mathcal{L}^{\mathrm{GH}} \text{ and Lemma 5.34} \\ &= [\![\lambda z.\, u]\!]^{\xi'}_{\mathcal{J}^{\mathrm{GH}}} && \text{by the definition of the term denotation}\end{aligned}$$

The step $*$ is justified as follows: We have $\mathcal{L}^{\mathrm{GH}}(\xi', \lambda z.\, u) = \mathcal{E}([\![\mathcal{F}((\lambda z.\, u)\theta'\downarrow_{\beta\eta})]\!]^\xi_R)$ by the definition of $\mathcal{L}^{\mathrm{GH}}$, if θ' is a substitution such that $\mathcal{D}_{\alpha\theta'} = \xi'(\alpha)$ for all α and $\mathcal{E}([\![\mathcal{F}(x\theta'\downarrow_{\beta\eta})]\!]^\xi_R) = \xi'(x)$ for all x. By the definition of ξ' and by Lemma 5.34, ρ is such a substitution. Hence, $\mathcal{L}^{\mathrm{GH}}(\xi', \lambda z.\, u) = \mathcal{E}([\![\mathcal{F}((\lambda z.\, u)\rho\downarrow_{\beta\eta})]\!]^\xi_R)$. □

Lemma 5.36. *The interpretation $\mathcal{J}^{\mathrm{GH}}$ is proper.*

Proof. We must show that $[\![(\lambda x.\, t)]\!]^{\xi}_{\mathcal{I}^{\mathrm{GH}}}(a) = [\![t]\!]^{\xi[x \mapsto a]}_{\mathcal{I}^{\mathrm{GH}}}$ for all λ-expressions $\lambda x.\, t$, all valuations ξ, and all values a.

$$
\begin{aligned}
[\![\lambda x.\, t]\!]^{\xi}_{\mathcal{I}^{\mathrm{GH}}}(a) &= \mathcal{L}^{\mathrm{GH}}(\xi, \lambda x.\, t)(a) && \text{by the definition of } [\![\ ]\!]_{\mathcal{I}^{\mathrm{GH}}} \\
&= \mathcal{E}([\![\mathcal{F}((\lambda x.\, t)\theta{\downarrow_{\beta\eta}})]\!]_R)(a) && \text{by the definition of } \mathcal{L}^{\mathrm{GH}} \text{ for some } \theta \\
& && \text{such that } \mathcal{E}([\![\mathcal{F}(z\theta)]\!]_R) = \xi(z) \text{ for all } z \\
& && \text{and } \mathcal{D}_{a\theta} = \xi(a) \text{ for all } a \\
&= \mathcal{E}([\![\mathcal{F}(((\lambda x.\, t)\theta\ s){\downarrow_{\beta\eta}})]\!]_R) && \text{by the definition of } \mathcal{E} \\
& && \text{where } \mathcal{E}([\![\mathcal{F}(s)]\!]_R) = a \\
&= \mathcal{E}([\![\mathcal{F}(t(\theta[x \mapsto s]){\downarrow_{\beta\eta}})]\!]_R) && \text{by } \beta\text{-reduction} \\
&= [\![t(\theta[x \mapsto s])]\!]_{\mathcal{I}^{\mathrm{GH}}} && \text{by Lemma 5.34} \\
&= [\![t]\!]^{\xi[x \mapsto a]}_{\mathcal{I}^{\mathrm{GH}}} && \text{by Lemma 5.35}
\end{aligned}
$$

□

Lemma 5.37. *$\mathcal{I}^{\mathrm{GH}}$ is a model of N.*

Proof. By Lemma 5.34, we have $[\![t]\!]_{\mathcal{I}^{\mathrm{GH}}} = \mathcal{E}([\![\mathcal{F}(t)]\!]_R)$ for all $t \in \mathcal{T}_{\mathrm{GH}}$. Since $\mathcal{E}$ is a bijection, it follows that any (dis)equation $s \mathrel{\dot{\approx}} t \in \mathcal{C}_{\mathrm{GH}}$ is true in $\mathcal{I}^{\mathrm{GH}}$ if and only if $\mathcal{F}(s \mathrel{\dot{\approx}} t)$ is true in R. Hence, a clause $C \in \mathcal{C}_{\mathrm{GH}}$ is true in $\mathcal{I}^{\mathrm{GH}}$ if and only if $\mathcal{F}(C)$ is true in R. By Theorem 3.29 and the assumption that $\bot \notin N$, R is a model of $\mathcal{F}(N)$—that is, for all clauses $C \in N$, $\mathcal{F}(C)$ is true in R. Hence, all clauses $C \in N$ are true in $\mathcal{I}^{\mathrm{GH}}$ and therefore $\mathcal{I}^{\mathrm{GH}}$ is a model of N. □

We summarize the results of this subsection in the following theorem:

Theorem 5.38 (Ground static refutational completeness)**.** *Let GHSel be a selection function on $\mathcal{C}_{\mathrm{GH}}$. Then the inference system $\mathrm{GHInf}^{\mathrm{GHSel}}$ is statically refutationally complete w.r.t. $(\mathrm{GHRed}_{\mathrm{I}}, \mathrm{GHRed}_{\mathrm{C}})$. In other words, if $N \subseteq \mathcal{C}_{\mathrm{GH}}$ is a clause set saturated w.r.t. $\mathrm{GHInf}^{\mathrm{GHSel}}$ and $\mathrm{GHRed}_{\mathrm{I}}^{\mathrm{GHSel}}$, then $N \models \bot$ if and only if $\bot \in N$.*

The construction of $\mathcal{I}^{\mathrm{GH}}$ relies on specific properties of R. It would not work with an arbitrary first-order interpretation. Transforming a higher-order interpretation into a first-order interpretation is easier:

Lemma 5.39. *Given a clausal higher-order interpretation $\mathcal{I}$ on* GH*, there exists a first-order interpretation $\mathcal{I}^{\mathrm{GF}}$ on* GF *such that for any clause $C \in \mathcal{C}_{\mathrm{GH}}$ the truth values of C in $\mathcal{I}$ and of $\mathcal{F}(C)$ in $\mathcal{I}^{\mathrm{GF}}$ coincide.*

Proof. Let $\mathcal{I} = (\mathcal{I}_{\mathrm{ty}}, \mathcal{J}, \mathcal{L})$ be a clausal higher-order interpretation. Let $\mathcal{U}^{\mathrm{GF}}_{\tau} = [\![\tau]\!]_{\mathcal{I}_{\mathrm{ty}}}$ be the first-order type universe for the ground type τ. For a symbol $\mathsf{f}^{\bar{\upsilon}}_{j} \in \Sigma_{\mathrm{GF}}$, let $\mathcal{J}^{\mathrm{GF}}(\mathsf{f}^{\bar{\upsilon}}_{j}) = [\![\mathsf{f}\langle\bar{\upsilon}\rangle]\!]_{\mathcal{I}}$ (up to currying). For a symbol $\mathsf{lam}_{\lambda x.\, t} \in \Sigma_{\mathrm{GF}}$, let $\mathcal{J}^{\mathrm{GF}}(\mathsf{lam}_{\lambda x.\, t}) = [\![\lambda x.\, t]\!]_{\mathcal{I}}$. This defines a first-order interpretation $\mathcal{I}^{\mathrm{GF}} = (\mathcal{U}^{\mathrm{GF}}, \mathcal{J}^{\mathrm{GF}})$.

We need to show that for any $C \in \mathcal{C}_{\mathrm{GH}}$, $\mathcal{I} \models C$ if and only if $\mathcal{I}^{\mathrm{GF}} \models \mathcal{F}(C)$. It suffices to show that $[\![t]\!]_{\mathcal{I}} = [\![\mathcal{F}(t)]\!]_{\mathcal{I}^{\mathrm{GF}}}$ for all terms $t \in \mathcal{T}_{\mathrm{GH}}$. We prove this by induction on the structure of the η-short β-normal form of t. If t is a λ-expression, this is obvious. If t is of the form $\mathsf{f}\langle\bar{\upsilon}\rangle\bar{s}_j$, then $\mathcal{F}(t) = \mathsf{f}^{\bar{\upsilon}}_{j}(\mathcal{F}(\bar{s}_j))$ and hence $[\![\mathcal{F}(t)]\!]_{\mathcal{I}^{\mathrm{GF}}} = \mathcal{J}^{\mathrm{GF}}(\mathsf{f}^{\bar{\upsilon}}_{j})([\![\mathcal{F}(\bar{s}_j)]\!]_{\mathcal{I}^{\mathrm{GF}}}) \overset{\mathrm{IH}}{=} [\![\mathsf{f}\langle\bar{\upsilon}\rangle]\!]_{\mathcal{I}}([\![\bar{s}_j]\!]_{\mathcal{I}}) = [\![t]\!]_{\mathcal{I}}$. □

5.4.3. The Nonground Higher-Order Level

To lift the result to the nonground level, we employ the saturation framework of Waldmann et al. [140]. It is easy to see that the entailment relation $\models$ on GH is a consequence relation in the sense of the framework. We need to show that our redundancy criterion on GH is a redundancy criterion in the sense of the framework and that $\mathcal{G}$ is a grounding function in the sense of the framework:

Lemma 5.40. *The redundancy criterion for* GH *is a redundancy criterion in the sense of Section 2 of the saturation framework.*

Proof. The proof is analogous to the proof of Lemma 3.36. □

Lemma 5.41. *The grounding functions $\mathcal{G}^{GHSel}$ for $GHSel \in \mathcal{G}(HSel)$ are grounding functions in the sense of Section 3 of the saturation framework.*

Proof. We must prove the conditions (G1), (G2), and (G3) of the saturation framework. Adapted to our context, they state the following:

(G1) $\mathcal{G}(\bot) = \{\bot\}$;
(G2) for every $C \in \mathcal{C}_{\mathrm{H}}$, if $\bot \in \mathcal{G}(C)$, then $C = \bot$;
(G3) for every $\iota \in HInf$, $\mathcal{G}^{GHSel}(\iota) \subseteq GHRed_{\mathrm{I}}^{GHSel}(\mathcal{G}(concl(\iota)))$.

Clearly, $C = \bot$ if and only if $\bot \in \mathcal{G}(C)$ if and only if $\mathcal{G}(C) = \{\bot\}$, proving (G1) and (G2). For every $\iota \in HInf$, by the definition of $\mathcal{G}^{GHSel}$, we have $concl(\mathcal{G}^{GHSel}(\iota)) \subseteq \mathcal{G}(concl(\iota))$, and thus (G3) by (R4). □

As in Chapter 3, we use the saturation framework's lifting theorem to lift the completeness result of the previous subsection. It can be stated as a verbatim copy of Theorem 3.38, but referring to the notions established in this chapter:

Theorem 5.42 (Lifting theorem)**.** *If $GHInf^{GHSel}$ is statically refutationally complete w.r.t. $(GHRed_{\mathrm{I}}^{GHSel}, GHRed_{\mathrm{C}})$ for every $GHSel \in \mathcal{G}(HSel)$, and if for every $N \subseteq \mathcal{C}_{\mathrm{H}}$ that is saturated w.r.t. $HInf$ and $HRed_{\mathrm{I}}$ there exists a $GHSel \in \mathcal{G}(HSel)$ such that $GHInf^{GHSel}(\mathcal{G}(N)) \subseteq \mathcal{G}^{GHSel}(HInf(N)) \cup GHRed_{\mathrm{I}}^{GHSel}(\mathcal{G}(N))$, then also $HInf$ is statically refutationally complete w.r.t. $(HRed_{\mathrm{I}}, HRed_{\mathrm{C}})$ and $\models_{\mathcal{G}}$.*

Proof. The proof is analogous to the one of Theorem 3.38, but using Lemma 5.40 and 5.41. □

Let $N \subseteq \mathcal{C}_{\mathrm{H}}$ be a clause set saturated w.r.t. $HInf$ and $HRed_{\mathrm{I}}$. We assume that $HSel$ fulfills the selection restriction that a literal $L\langle y\rangle$ must not be selected if $y\,\bar{u}_n$, with $n > 0$, is a $\succeq$-maximal term of the clause, as required in Definition 5.9. For the above theorem to apply, we need to show that there exists a selection function $GHSel \in \mathcal{G}(HSel)$ such that all inferences $\iota \in GHInf^{GHSel}$ with $prems(\iota) \in \mathcal{G}(N)$ are liftable or redundant. Here, for ι to be *liftable* means that ι is a $\mathcal{G}^{GHSel}$-ground instance of a $HInf$-inference from N; for ι to be *redundant* means that $\iota \in GHRed_{\mathrm{I}}^{GHSel}(\mathcal{G}(N))$.

To choose the right selection function $GHSel \in \mathcal{G}(HSel)$, we observe that each ground clause $C \in \mathcal{G}(N)$ must have at least one corresponding clause $D \in N$ such that C is a ground instance of D. We choose one of them for each $C \in \mathcal{G}(N)$, which we denote by $\mathcal{G}^{-1}(C)$. Then let $GHSel$ select those literals in C that correspond to

literals selected by *HSel* in $\mathcal{G}^{-1}(C)$. With respect to this selection function *GHSel*, we can show that all inferences from $\mathcal{G}(N)$ are liftable or redundant:

Lemma 5.43. *Let $\mathcal{G}^{-1}(C) = D \in N$ and $D\theta = C$. Let σ and ρ be substitutions such that $x\sigma\rho = x\theta$ for all variables x in D. Let L be a (strictly) $\succeq$-eligible literal in C w.r.t. GHSel. Then there exists a (strictly) $\succsim$-eligible literal L' in D w.r.t. σ and HSel such that $L'\theta = L$.*

Proof. If $L \in GHSel(C)$, then there exists L' such that $L'\theta = L$ and $L' \in HSel(D)$ by the definition of $\mathcal{G}^{-1}$. Otherwise, L is $\succeq$-maximal in C. Since $C = D\sigma\rho$, there are literals L' in $D\sigma$ such that $L'\rho = L$. Choose L' to be a $\succsim$-maximal among them. Then L' is $\succsim$-maximal in $D\sigma$ because for any literal $L'' \in D$ with $L'' \succsim L'$, we have $L''\rho \succeq L'\rho = L$ and hence $L''\rho = L$ by $\succeq$-maximality of L.

If L is strictly $\succeq$-maximal in C, L' is also strictly $\succsim$-maximal in $D\sigma$ because a duplicate of L' in $D\sigma$ would imply a duplicate of L in C. □

Lemma 5.44 (Lifting of ERes, EFact, GArgCong, and GExt)**.** *All* ERes*,* EFact*,* GArgCong*, and* GExt *inferences from $\mathcal{G}(N)$ are liftable.*

Proof. ERes: Let $\iota \in GHInf^{GHSel}$ be an ERes inference with $prems(\iota) \in \mathcal{G}(N)$. Then ι is of the form

$$\frac{C\theta \;=\; C'\theta \vee s\theta \not\approx s'\theta}{C'\theta}\,\text{ERes}$$

where $\mathcal{G}^{-1}(C\theta) = C = C' \vee s \not\approx s'$ and the literal $s\theta \not\approx s'\theta$ is $\succeq$-eligible w.r.t. *GHSel*. Since $s\theta$ and $s'\theta$ are unifiable and ground, we have $s\theta = s'\theta$. Thus, there exists an idempotent $\sigma \in \mathrm{CSU}(s, s')$ such that for some substitution ρ and for all variables x in C, we have $x\sigma\rho = x\theta$. By Lemma 5.43, we may assume without loss of generality that $s \not\approx s'$ is $\succsim$-eligible in C w.r.t. σ and *HSel*. Hence, the following inference $\iota' \in HInf$ is applicable:

$$\frac{C' \vee s \not\approx s'}{C'\sigma}\,\text{ERes}$$

Then ι is the $\sigma\rho$-ground instance of ι' and is therefore liftable.

EFact: Analogously, if $\iota \in GHInf^{GHSel}$ is an EFact inference with $prems(\iota) \in \mathcal{G}(N)$, then ι is of the form

$$\frac{C\theta \;=\; C'\theta \vee s'\theta \approx t'\theta \vee s\theta \approx t\theta}{C'\theta \vee t\theta \not\approx t'\theta \vee s\theta \approx t'\theta}\,\text{EFact}$$

where $\mathcal{G}^{-1}(C\theta) = C = C' \vee s' \approx t' \vee s \approx t$, the literal $s\theta \approx t\theta$ is $\succeq$-eligible in C w.r.t. *GHSel*, and $s\theta \not\prec t\theta$. Then $s \not\prec t$. Moreover, $s\theta$ and $s'\theta$ are unifiable and ground. Hence, $s\theta = s'\theta$ and there exists an idempotent $\sigma \in \mathrm{CSU}(s, s')$ such that for some substitution ρ and for all variables x in C, we have $x\sigma\rho = x\theta$. By Lemma 5.43, we may assume without loss of generality that $s \approx t$ is $\succsim$-eligible in C w.r.t. σ and *HSel*. It follows that the following inference $\iota' \in HInf$ is applicable:

$$\frac{C' \vee s' \approx t' \vee s \approx t}{(C' \vee t \not\approx t' \vee s \approx t')\sigma}\,\text{EFact}$$

Then ι is the $\sigma\rho$-ground instance of ι' and is therefore liftable.

GArgCong: Let $\iota \in \mathit{GHInf}^{\mathit{GHSel}}$ be a GArgCong inference with $\mathit{prems}(\iota) \in \mathcal{G}(N)$. Then ι is of the form

$$\frac{C\theta \;=\; C'\theta \vee s\theta \approx s'\theta}{C'\theta \vee s\theta\,\bar{u}_n \approx s'\theta\,\bar{u}_n}\,\textsc{GArgCong}$$

where $\mathcal{G}^{-1}(C\theta) = C = C' \vee s \approx s'$, the literal $s\theta \approx s'\theta$ is strictly $\succeq$-eligible w.r.t. *GHSel*, and $s\theta$ and $s'\theta$ are of functional type. It follows that s and s' have either a functional or a polymorphic type. Let σ be the most general substitution such that $s\sigma$ and $s'\sigma$ take n arguments. By Lemma 5.43, we may assume without loss of generality that $s \not\approx s'$ is strictly $\succsim$-eligible in C w.r.t. σ and *HSel*. Hence the following inference $\iota' \in \mathit{HInf}$ is applicable:

$$\frac{C' \vee s \approx s'}{C'\sigma \vee s\sigma\,\bar{x}_n \approx s'\sigma\,\bar{x}_n}\,\textsc{ArgCong}$$

Since σ is the most general substitution that ensures well-typedness of the conclusion, ι is a ground instance of ι' and is therefore liftable.

GExt: The conclusion of a GExt inference in *GHInf* is by definition a ground instance of the conclusion of an Ext inference in *HInf*. Hence, the GExt inference is a ground instance of the Ext inference. Therefore it is liftable. □

Some of the Sup inferences in *GHInf* are liftable as well:

Lemma 5.45 (Instances of green subterms)**.** *Let s be a λ-term in η-short β-normal form, let σ be a substitution, and let p be a green position of both s and $s\sigma\downarrow_{\beta\eta}$. Then $(s|_p)\sigma\downarrow_{\beta\eta} = (s\sigma\downarrow_{\beta\eta})|_p$.*

Proof. By induction on p. If $p = \varepsilon$, then $(s|_p)\sigma\downarrow_{\beta\eta} = s\sigma\downarrow_{\beta\eta} = (s\sigma\downarrow_{\beta\eta})|_p$. If $p = i.p'$, then $s = \mathrm{f}\langle\bar{\tau}\rangle\, s_1 \ldots s_n$ and $s\sigma = \mathrm{f}\langle\bar{\tau}\sigma\rangle(s_1\sigma)\ldots(s_n\sigma)$, where $1 \le i \le n$ and p' is a green position of s_i. Clearly, $\beta\eta$-normalization steps of $s\sigma$ can take place only in proper subterms. So $s\sigma\downarrow_{\beta\eta} = \mathrm{f}\langle\bar{\tau}\sigma\rangle(s_1\sigma\downarrow_{\beta\eta})\ldots(s_n\sigma\downarrow_{\beta\eta})$. Since $p = i.p'$ is a green position of $s\sigma\downarrow_{\beta\eta}$, p' must be a green position of $(s_i\sigma)\downarrow_{\beta\eta}$. By the induction hypothesis, $(s_i|_{p'})\sigma\downarrow_{\beta\eta} = (s_i\sigma\downarrow_{\beta\eta})|_{p'}$. Therefore $(s|_p)\sigma\downarrow_{\beta\eta} = (s|_{i.p'})\sigma\downarrow_{\beta\eta} = (s_i|_{p'})\sigma\downarrow_{\beta\eta} = (s_i\sigma\downarrow_{\beta\eta})|_{p'} = (s\sigma\downarrow_{\beta\eta})|_p$. □

Lemma 5.46 (Lifting of Sup)**.** *Let $\iota \in \mathit{GHInf}^{\mathit{GHSel}}$ be a* Sup *inference*

$$\frac{\overbrace{D'\theta \vee t\theta \approx t'\theta}^{D\theta} \qquad \overbrace{C'\theta \vee s\theta\langle t\theta\rangle_p \mathrel{\dot{\approx}} s'\theta}^{C\theta}}{D'\theta \vee C'\theta \vee s\theta\langle t'\theta\rangle_p \mathrel{\dot{\approx}} s'\theta}\,\textsc{Sup}$$

where $\mathcal{G}^{-1}(D\theta) = D = D' \vee t \approx t' \in N$, $s\theta = s\theta\langle t\theta\rangle_p$, and $\mathcal{G}^{-1}(C\theta) = C = C' \vee s \mathrel{\dot{\approx}} s' \in N$. We assume that s, t, $s\theta$, and $t\theta$ are represented by λ-terms in η-short β-normal form. Let p' be the longest prefix of p that is a green position of s. Since ε is a green position of s, the longest prefix always exists. Let $u = s|_{p'}$. Suppose one of the following

conditions applies: (i) u *is a deeply occurring variable in* C*;* (ii) $p = p'$ *and the variable condition holds for* D *and* C*; or* (iii) $p \neq p'$ *and* u *is not a variable. Then* ι *is liftable.*

Proof. The SUP inference conditions for ι are that $t\theta \approx t'\theta$ is strictly $\succeq$-eligible, $s\theta \mathrel{\dot\approx} s'\theta$ is strictly $\succeq$-eligible if positive and $\succeq$-eligible if negative, $D\theta \not\subset C\theta$, $t\theta \not\subset t'\theta$, and $s\theta \not\subset s'\theta$. We assume that s, t, $s\theta$, and $t\theta$ are represented by λ-terms in η-short β-normal form. By Lemma 5.45, $u\theta$ agrees with $s\theta|_{p'}$ (considering both as terms rather than as λ-terms).

CASE 1: We have (a) $p = p'$, (b) u is not fluid, and (c) u is not a variable deeply occurring in C. Then $u\theta = s\theta|_{p'} = s\theta|_p = t\theta$. Since θ is a unifier of u and t, there exists an idempotent $\sigma \in \mathrm{CSU}(t,u)$ such that for some substitution ρ and for all variables x occurring in D and C, we have $x\sigma\rho = x\theta$. The inference conditions can be lifted: (Strict) eligibility of $t\theta \approx t'\theta$ and $s\theta \mathrel{\dot\approx} s'\theta$ w.r.t. *GHSel* implies (strict) eligibility of $t \approx t'$ and $s \mathrel{\dot\approx} s'$ w.r.t. σ and *HSel*; $D\theta \not\subset C\theta$ implies $D \not\subset C$; $t\theta \not\subset t'\theta$ implies $t \not\subset t'$; and $s\theta \not\subset s'\theta$ implies $s \not\subset s'$. Moreover, by (a) and (c), condition (ii) must hold and thus the variable condition holds for D and C. Hence there is the following SUP inference $\iota' \in \mathit{HInf}$:

$$\frac{D' \vee t \approx t' \qquad C' \vee s\langle u\rangle_p \mathrel{\dot\approx} s'}{(D' \vee C' \vee s\langle t'\rangle_p \mathrel{\dot\approx} s')\sigma}\,\textsc{Sup}$$

Then ι is the $\sigma\rho$-ground instance of ι' and therefore liftable.

CASE 2: We have (a) $p \neq p'$, or (b) u is fluid, or (c) u is a variable deeply occurring in C. We will first show that (a) implies (b) or (c). Suppose (a) holds but neither (b) nor (c) holds. Then condition (iii) must hold—i.e., u is not a variable. Moreover, since (b) does not hold, u cannot have the form $y\,\bar{u}_n$ for a variable y and $n \geq 1$. If u were of the form $\mathrm{f}\langle\bar\tau\rangle s_1 \ldots s_n$ with $n \geq 0$, $u\theta$ would have the form $\mathrm{f}\langle\bar\tau\theta\rangle(s_1\theta)\ldots(s_n\theta)$, but then there is some $1 \leq i \leq n$ such that $p'.i$ is a prefix of p and $s|_{p'.i}$ is a green subterm of s, contradicting the maximality of p'. So u must be a λ-expression, but since $t\theta$ is a proper green subterm of $u\theta$, $u\theta$ cannot be a λ-expression, yielding a contradiction. We may thus assume that (b) or (c) holds.

Let $p = p'.p''$. Let z be a fresh variable. Define a substitution θ' that maps this variable z to $\lambda y.(s\theta|_{p'})\langle y\rangle_{p''}$ and any other variable w to $w\theta$. Clearly, $(z\,t)\theta' = (s\theta|_{p'})\langle t\theta\rangle_{p''} = s\theta|_{p'} = u\theta = u\theta'$. Since θ' is a unifier of u and $z\,t$, there exists an idempotent $\sigma \in \mathrm{CSU}(z\,t,u)$ such that for some substitution ρ, for $x = z$, and for all variables x in C and D, we have $x\sigma\rho = x\theta'$. As in case 1, (strict) eligibility of the ground literals implies (strict) eligibility of the nonground literals. Moreover, by construction of θ', $t\theta' = t\theta \neq t'\theta = t'\theta'$ implies $(z\,t)\theta' \neq (z\,t')\theta'$, and thus $(z\,t)\sigma \neq (z\,t')\sigma$. Since we also have (b) or (c), there is the following inference ι':

$$\frac{D' \vee t \approx t' \qquad C' \vee s\langle u\rangle_{p'} \mathrel{\dot\approx} s'}{(D' \vee C' \vee s\langle z\,t'\rangle_{p'} \mathrel{\dot\approx} s')\sigma}\,\textsc{FluidSup}$$

Then ι is the $\sigma\rho$-ground instance of ι' and therefore liftable. □

The other SUP inferences might not be liftable, but they are redundant:

Lemma 5.47. *Let $\iota \in GHInf^{GHSel}$ be a* Sup *inference from $\mathcal{G}(N)$ not covered by Lemma 5.46. Then $\iota \in GHRed_{\mathrm{I}}^{GHSel}(\mathcal{G}(N))$.*

Proof. Let $C\theta = C'\theta \vee s\theta \mathrel{\dot{\approx}} s'\theta$ and $D\theta = D'\theta \vee t\theta \approx t'\theta$ be the premises of ι, where $s\theta \mathrel{\dot{\approx}} s'\theta$ and $t\theta \approx t'\theta$ are the literals involved in the inference, $s\theta \succ s'\theta$, $t\theta \succ t'\theta$, and C', D', s, s', t, t' are the respective subclauses and terms in $C = \mathcal{G}^{-1}(C\theta)$ and $D = \mathcal{G}^{-1}(D\theta)$. Then the inference ι has the form

$$\frac{D'\theta \vee t\theta \approx t'\theta \qquad C'\theta \vee s\theta\langle t\theta\rangle \mathrel{\dot{\approx}} s'\theta}{D'\theta \vee C'\theta \vee s\theta\langle t'\theta\rangle \mathrel{\dot{\approx}} s'\theta}\ \textsc{Sup}$$

To show that $\iota \in GHRed_{\mathrm{I}}^{GHSel}(\mathcal{G}(N))$, it suffices to show $\{D \in \mathcal{F}(\mathcal{G}(N)) \mid D \prec \mathcal{F}(C\theta)\} \models \mathcal{F}(concl(\iota))$. To this end, let $\mathcal{I}$ be an interpretation in GF such that $\mathcal{I} \models \{D \in \mathcal{F}(\mathcal{G}(N)) \mid D \prec \mathcal{F}(C\theta)\}$. We need to show that $\mathcal{I} \models \mathcal{F}(concl(\iota))$. If $\mathcal{F}(D'\theta)$ is true in $\mathcal{I}$, then obviously $\mathcal{I} \models \mathcal{F}(concl(\iota))$. So we assume that $\mathcal{F}(D'\theta)$ is false in $\mathcal{I}$. Since $C\theta \succ D\theta$ by the Sup order conditions, it follows that $\mathcal{I} \models \mathcal{F}(t\theta \approx t'\theta)$. Therefore, it suffices to show $\mathcal{I} \models \mathcal{F}(C\theta)$.

Let p be the position in $s\theta$ where ι takes place and p' be the longest prefix of p that is a green subterm of s. Let $u = s|_{p'}$. Since Lemma 5.46 does not apply to ι, u is not a deeply occurring variable; if $p = p'$, the variable condition does not hold for D and C; and if $p \neq p'$, u is a variable. This means either the position p does not exist in s, because it is below an unapplied variable that does not occur deeply in C, or $s|_p$ is an unapplied variable that does not occur deeply in C and for which the variable condition does not hold.

Case 1: The position p does not exist in s because it is below a variable x that does not occur deeply in C. Then $t\theta$ is a green subterm of $x\theta$ and hence a green subterm of $x\theta\,\bar{w}$ for any arguments $\bar{w}$. Let v be the term that we obtain by replacing $t\theta$ by $t'\theta$ in $x\theta$ at the relevant position. Since $\mathcal{I} \models \mathcal{F}(t\theta \approx t'\theta)$, by congruence, $\mathcal{I} \models \mathcal{F}(x\theta\,\bar{w} \approx v\,\bar{w})$ for any arguments $\bar{w}$. Hence, $\mathcal{I} \models \mathcal{F}(C\theta)$ if and only if $\mathcal{I} \models \mathcal{F}(C\{x \mapsto v\}\theta)$ by congruence. Here, it is crucial that the variable does not occur deeply in C because congruence does not hold in $\mathcal{F}$-encoded terms below λ-binders. By the inference conditions, we have $t\theta \succ t'\theta$, which implies $\mathcal{F}(C\theta) \succ \mathcal{F}(C\{x \mapsto v\}\theta)$ by compatibility with green contexts. Therefore, by the assumption about $\mathcal{I}$, we have $\mathcal{I} \models \mathcal{F}(C\{x \mapsto v\}\theta)$ and hence $\mathcal{I} \models \mathcal{F}(C\theta)$.

Case 2: The term $s|_p$ is a variable x that does not occur deeply in C and for which the variable condition does not hold. From this, we know that $C\theta \succeq C''\theta$, where $C'' = C\{x \mapsto t'\}$. We cannot have $C\theta = C''\theta$ because $x\theta = t\theta \neq t'\theta$ and x occurs in C. Hence, we have $C\theta \succ C''\theta$. By the definition of $\mathcal{I}$, $C\theta \succ C''\theta$ implies $\mathcal{I} \models \mathcal{F}(C''\theta)$. We will use equalities that are true in $\mathcal{I}$ to rewrite $\mathcal{F}(C\theta)$ into $\mathcal{F}(C''\theta)$, which implies $\mathcal{I} \models \mathcal{F}(C\theta)$ by congruence.

By saturation, every ArgCong inference ι' from D is in $HRed_{\mathrm{I}}(N)$, which by definition means that $\mathcal{G}(concl(\iota')) \subseteq \mathcal{G}(N) \cup GHRed_{\mathrm{C}}(\mathcal{G}(N))$. Hence, $D'\theta \vee t\theta\,\bar{u} \approx t'\theta\,\bar{u}$ is in $\mathcal{G}(N) \cup GHRed_{\mathrm{C}}(\mathcal{G}(N))$ for any ground arguments $\bar{u}$.

We observe that whenever $t\theta\,\bar{u}$ and $t'\theta\,\bar{u}$ are smaller than the $\succeq$-maximal term of $C\theta$ for some arguments $\bar{u}$, we have

$$\mathcal{I} \models \mathcal{F}(t\theta\,\bar{u}) \approx \mathcal{F}(t'\theta\,\bar{u}) \qquad (*)$$

To show this, we assume that $t\theta\,\bar{u}$ and $t'\theta\,\bar{u}$ are smaller than the $\succeq$-maximal term of $C\theta$ and we distinguish two cases: If $t\theta$ is smaller than the $\succeq$-maximal term of $C\theta$, all terms in $D'\theta$ are smaller than the $\succeq$-maximal term of $C\theta$ and hence $D'\theta \vee t\theta\,\bar{u} \approx t'\theta\,\bar{u} \prec C\theta$. If, on the other hand, $t\theta$ is equal to the $\succeq$-maximal term of $C\theta$, then $t\theta\,\bar{u}$ and $t'\theta\,\bar{u}$ are smaller than $t\theta$. Hence $t\theta\,\bar{u} \approx t'\theta\,\bar{u} \prec t\theta \approx t'\theta$ and $D'\theta \vee t\theta\,\bar{u} \approx t'\theta\,\bar{u} \prec D\theta \prec C\theta$. In both cases, since $D'\theta$ is false in $\mathcal{I}$, by the definition of $\mathcal{I}$, we have $(*)$.

Next, we show the equivalence of $C\theta$ and $C''\theta$ via rewriting with equations of the form $(*)$ where $t\theta\,\bar{u}$ and $t'\theta\,\bar{u}$ are smaller than the $\succeq$-maximal term of $C\theta$. Since x does not occur deeply in C, every occurrence of x in C is not inside a λ-expression and not inside an argument of an applied variable. Therefore, all occurrences of x in C are in a green subterm of the form $x\,\bar{v}$ for some terms $\bar{v}$ that do not contain x. Hence, every occurrence of x in C corresponds to a subterm $\mathcal{F}((x\,\bar{v})\theta) = \mathcal{F}(t\theta\,\bar{v}\theta)$ in $\mathcal{F}(C\theta)$ and to a subterm $\mathcal{F}((x\,\bar{v})\{x \mapsto t'\}\theta) = \mathcal{F}(t'\theta\,\bar{v}\{x \mapsto t'\}\theta) = \mathcal{F}(t'\theta\,\bar{v}\theta)$ in $\mathcal{F}(C''\theta)$. These are the only positions where $C\theta$ and $C''\theta$ differ.

To justify the necessary rewrite steps from $\mathcal{F}(t\theta\,\bar{v}\theta)$ into $\mathcal{F}(t'\theta\,\bar{v}\theta)$ using $(*)$, we must show that $\mathcal{F}(t\theta\,\bar{v}\theta)$ and $\mathcal{F}(t'\theta\,\bar{v}\theta)$ are smaller than the $\succeq$-maximal term in $\mathcal{F}(C\theta)$ for the relevant $\bar{v}$. If $\bar{v}$ is the empty tuple, we do not need to show this because $\mathcal{I} \models \mathcal{F}(t\theta \approx t'\theta)$ follows from $\mathcal{F}(D\theta)$'s being true and $\mathcal{F}(D'\theta)$'s being false. If $\bar{v}$ is nonempty, it suffices to show that $x\,\bar{v}$ is not a $\succeq$-maximal term in C. Then $\mathcal{F}(t\theta\,\bar{v}\theta)$ and $\mathcal{F}(t'\theta\,\bar{v}\theta)$, which correspond to the term $x\,\bar{v}$ in C, cannot be $\succeq$-maximal in $\mathcal{F}(C\theta)$ and $\mathcal{F}(C''\theta)$. Hence they must be smaller than the $\succeq$-maximal term in $\mathcal{F}(C\theta)$ because they are subterms of $\mathcal{F}(C\theta)$ and $\mathcal{F}(C''\theta) \prec \mathcal{F}(C\theta)$, respectively.

To show that $x\,\bar{v}$ is not a $\succeq$-maximal term in C, we make a case distinction on whether $s\theta \mathrel{\dot{\approx}} s'\theta$ is selected in $C\theta$ or $s\theta$ is the $\succeq$-maximal term in $C\theta$. One of these must hold because $s\theta \mathrel{\dot{\approx}} s'\theta$ is $\succeq$-eligible in $C\theta$. If it is selected, by the selection restrictions, x cannot be the head of a $\succeq$-maximal term of C. If $s\theta$ is the $\succeq$-maximal term in $C\theta$, we can argue that x is a green subterm of s and, since x does not occur deeply, s cannot be of the form $x\,\bar{v}$ for a nonempty $\bar{v}$. This justifies the necessary rewrites between $\mathcal{F}(C\theta)$ and $\mathcal{F}(C''\theta)$ and it follows that $\mathcal{I} \models \mathcal{F}(C\theta)$. □

With these properties of our inference systems in place, Theorem 5.42 guarantees static and dynamic refutational completeness of *HInf* w.r.t. $HRed_{\mathrm{I}}$. However, this theorem gives us refutational completeness w.r.t. the Herbrand entailment $\models_{\mathcal{G}}$, defined as $N_1 \models_{\mathcal{G}} N_2$ if $\mathcal{G}(N_1) \models \mathcal{G}(N_2)$, whereas our semantics is Tarski entailment $\models$, defined as $N_1 \models N_2$ if any model of N_1 is a model of N_2. To repair this mismatch, we use the following lemma, which can be proved along the lines of Lemma 3.43, using Lemma 5.18 and Lemma 5.19.

Lemma 5.48. *For $N \subseteq \mathcal{C}_{\mathrm{H}}$, we have $N \models_{\mathcal{G}} \bot$ if and only if $N \models \bot$.*

Theorem 5.49 (Static refutational completeness)**.** *The inference system HInf is statically refutationally complete w.r.t. ($HRed_{\mathrm{I}}$,$HRed_{\mathrm{C}}$). In other words, if $N \subseteq \mathcal{C}_{\mathrm{H}}$ is a clause set saturated w.r.t. HInf and $HRed_{\mathrm{I}}$, then we have $N \models \bot$ if and only if $\bot \in N$.*

Proof. We apply Theorem 5.42. By Theorem 5.38, $GHInf^{GHSel}$ is statically refutationally complete for all $GHSel \in \mathcal{G}(HSel)$. By Lemmas 5.44, 5.46, and 5.47, for every

saturated $N \subseteq \mathcal{C}_{\mathrm{H}}$, there exists a selection function $\mathit{GHSel} \in \mathcal{G}(\mathit{HSel})$ such that all inferences $\iota \in \mathit{GHInf}^{\mathit{GHSel}}$ with $\mathit{prems}(\iota) \in \mathcal{G}(N)$ either are $\mathcal{G}^{\mathit{GHSel}}$-ground instances of HInf-inferences from N or belong to $\mathit{GHRed}_{\mathrm{I}}^{\mathit{GHSel}}(\mathcal{G}(N))$.

Theorem 5.42 implies that if $N \subseteq \mathcal{C}_{\mathrm{H}}$ is a clause set saturated w.r.t. HInf and $\mathit{HRed}_{\mathrm{I}}$, then $N \models_{\mathcal{G}} \bot$ if and only if $\bot \in N$. By Lemma 5.48, this also holds for the Tarski entailment $\models$. That is, if $N \subseteq \mathcal{C}_{\mathrm{H}}$ is a clause set saturated w.r.t. HInf and $\mathit{HRed}_{\mathrm{I}}$, then $N \models \bot$ if and only if $\bot \in N$. □

From static refutational completeness, we can easily derive dynamic refutational completeness.

Theorem 5.50 (Dynamic refutational completeness)**.** *The inference system HInf is dynamically refutationally complete w.r.t.* ($\mathit{HRed}_{\mathrm{I}}$,$\mathit{HRed}_{\mathrm{C}}$)*, as defined in Definition 3.26.*

Proof. By Theorem 17 of the saturation framework, this follows from Theorem 5.49 and Lemma 5.48. □

5.5. Extensions

The core calculus can be extended with various optional rules. Although these are not necessary for refutational completeness, they can allow the prover to find more direct proofs. Most of these rules are concerned with the areas covered by the FLUIDSUP rule and the extensionality axiom.

Two of the optional rules below rely on the notion of "orange subterms."

Definition 5.51. A λ-term t is an *orange subterm* of a λ-term s if $s = t$; or if $s = \mathsf{f}\langle\bar{t}\rangle\bar{s}$ and t is an orange subterm of s_i for some i; or if $s = x\,\bar{s}$ and t is an orange subterm of s_i for some i; or if $s = (\lambda x.\, u)$ and t is an orange subterm of u.

For example, in the term $\mathsf{f}(\mathsf{g}\,\mathsf{a})(y\,\mathsf{b})(\lambda x.\,\mathsf{h}\,\mathsf{c}(\mathsf{g}\,x))$, the orange subterms are all the green subterms—a, $\mathsf{g}\,\mathsf{a}$, $y\,\mathsf{b}$, $\lambda x.\mathsf{h}\mathsf{c}(\mathsf{g}x)$ and the whole term—and in addition b, c, x, $\mathsf{g}\,x$, and $\mathsf{h}\,\mathsf{c}(\mathsf{g}\,x)$. Following Convention 5.1, this notion is lifted to $\beta\eta$-equivalence classes via representatives in η-short β-normal form. We write $t = s\langle\!\langle \bar{x}_n.\, u \rangle\!\rangle$ to indicate that u is an orange subterm of t, where $\bar{x}_n$ are the variables bound in the *orange context* around u, from outermost to innermost. If $n = 0$, we simply write $t = s\langle\!\langle u \rangle\!\rangle$.

Once a term $s\langle\!\langle \bar{x}_n.\, u \rangle\!\rangle$ has been introduced, we write $s\langle\!\langle \bar{x}_n.\, u' \rangle\!\rangle_\eta$ to denote the same context with a different subterm u' at that position. The η subscript is a reminder that u' is not necessarily an orange subterm of $s\langle\!\langle \bar{x}_n.\, u' \rangle\!\rangle_\eta$ due to potential applications of η-reduction. For example, if $s\langle\!\langle x.\,\mathsf{g}\,x\,x \rangle\!\rangle = \mathsf{h}\,\mathsf{a}\,(\lambda x.\,\mathsf{g}\,x\,x)$, then $s\langle\!\langle x.\,\mathsf{f}\,x \rangle\!\rangle_\eta = \mathsf{h}\,\mathsf{a}\,(\lambda x.\,\mathsf{f}\,x) = \mathsf{h}\,\mathsf{a}\,\mathsf{f}$.

Demodulation, which destructively rewrites using an equality $t \approx t'$, is available at green positions. In addition, a variant of demodulation rewrites in orange contexts:

$$\frac{t \approx t' \quad C\langle s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\rangle}{t \approx t' \quad C\langle s\langle\!\langle \bar{x}.\, t'\sigma \rangle\!\rangle_\eta\rangle \quad s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle \approx s\langle\!\langle \bar{x}.\, t'\sigma \rangle\!\rangle_\eta}\ \lambda\textsc{DemodExt}$$

where the term $t\sigma$ may refer to the bound variables $\bar{x}$. The following side conditions apply:

1. $s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\downarrow_{\beta\eta}$ is a λ-expression or a term of the form $y\,\bar{u}_n$ with $n > 0$;
2. $s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle \succ s\langle\!\langle \bar{x}.\, t'\sigma \rangle\!\rangle_\eta$; 3. $C\langle s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\rangle \succ s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle \approx s\langle\!\langle \bar{x}.\, t'\sigma \rangle\!\rangle_\eta$

Condition 3 ensures that the second premise is redundant w.r.t. the conclusions and may be removed. The double bar indicates that the conclusions collectively make the premises redundant and can replace them.

The third conclusion, which is entailed by $t \approx t'$ and (EXT), could be safely omitted if the corresponding (EXT) instance is smaller than the second premise. But in general, the third conclusion is necessary for the proof, and the variant of λDEMODEXT that omits it—let us call it λDEMOD—might not preserve refutational completeness.

An instance of λDEMODEXT, where $\mathsf{g}\,z$ is rewritten to $\mathsf{f}\,z\,z$ under a λ-binder, follows:

$$\frac{\mathsf{g}\,x \approx \mathsf{f}\,x\,x \quad \mathsf{k}(\lambda z.\,\mathsf{h}(\mathsf{g}\,z)) \approx \mathsf{c}}{\mathsf{g}\,x \approx \mathsf{f}\,x\,x \quad \mathsf{k}(\lambda z.\,\mathsf{h}(\mathsf{f}\,z\,z)) \approx \mathsf{c} \quad (\lambda z.\,\mathsf{h}(\mathsf{g}\,z)) \approx (\lambda z.\,\mathsf{h}(\mathsf{f}\,z\,z))}\,\lambda\text{DEMODEXT}$$

Lemma 5.52. *λDEMODEXT is sound and preserves refutational completeness of the calculus.*

Proof. Soundness of the first conclusion is obvious. Soundness of the second and third conclusion follows from congruence and extensionality using the premises. Preservation of completeness is justified by redundancy. Specifically, we justify the deletion of the second premise by showing that it is redundant w.r.t. the conclusions. By definition, it is redundant if for every ground instance $C\langle s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\rangle\theta \in \mathcal{G}(C\langle s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\rangle)$, its encoding $\mathcal{F}(C\langle s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\rangle\theta)$ is entailed by $\mathcal{F}(\mathcal{G}(N))$, where N are the conclusions of λDEMODEXT. The first conclusion cannot help us prove redundancy because $s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\theta\downarrow_{\beta\eta}$ might be a λ-expression and then $\mathcal{F}(s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\theta)$ is a symbol that is unrelated to $\mathcal{F}(t\sigma\theta)$. Instead, we use the θ-instances of the last two conclusions. By Lemma 5.23, $\mathcal{F}(C\langle s\langle\!\langle \bar{x}.\, t'\sigma \rangle\!\rangle_\eta\rangle\theta)$ has $\mathcal{F}(s\langle\!\langle \bar{x}.\, t'\sigma \rangle\!\rangle_\eta\theta)$ as a subterm. If this subterm is replaced by $\mathcal{F}(s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\theta)$, we obtain $\mathcal{F}(C\langle s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\rangle\theta)$. Hence, the $\mathcal{F}$-encodings of the θ-instances of the last two conclusions entail the $\mathcal{F}$-encoding of the θ-instance of the second premise by congruence. Due to the side condition that the second premise is larger than the second and third conclusion, by stability under grounding substitutions, the θ-instances of the last two conclusions must be smaller than the θ-instance of the second premise. Thus, the second premise is redundant. □

The next simplification rule can be used to prune arguments of applied variables if the arguments can be expressed as functions of the remaining arguments. For example, the clause $C[y\,\mathsf{a}\,\mathsf{b}(\mathsf{f}\,\mathsf{b}\,\mathsf{a}), y\,\mathsf{b}\,\mathsf{d}(\mathsf{f}\,\mathsf{d}\,\mathsf{b})]$, in which y occurs twice, can be simplified to $C[y'\,\mathsf{a}\,\mathsf{b}, y'\,\mathsf{b}\,\mathsf{d}]$. Here, for each occurrence of y, the third argument can be computed by applying f to the second and first arguments. The rule can also be used to remove the repeated arguments in $y\,\mathsf{b}\,\mathsf{b} \not\approx y\,\mathsf{a}\,\mathsf{a}$, the static argument a in $y\,\mathsf{a}\,\mathsf{c} \not\approx y\,\mathsf{a}\,\mathsf{b}$, and all four arguments in $y\,\mathsf{a}\,\mathsf{b} \not\approx z\,\mathsf{b}\,\mathsf{d}$. It is stated as

$$\frac{C}{C\sigma}\,\text{PRUNEARG}$$

where the following conditions apply:

1. $\sigma = \{y \mapsto \lambda \bar{x}_j.\, y'\, \bar{x}_{j-1}\}$; 2. y' is a fresh variable; 3. $C \sqsupset C\sigma$;
4. the minimum number k of arguments passed to any occurrence of y in the clause C is at least j;
5. there exists a term t containing no variables bound in the clause such that for all terms of the form $y\, \bar{s}_k$ occurring in the clause we have $s_j = t\, \bar{s}_{j-1}\, s_{j+1} \ldots s_k$.

Clauses with a static argument correspond to the case $t := (\lambda \bar{x}_{j-1}\, x_{j+1} \ldots x_k.\, u)$, where u is the static argument (containing no variables bound in t) and j is its index in y's argument list. The repeated argument case corresponds to $t := (\lambda \bar{x}_{j-1}\, x_{j+1} \ldots x_k.\, x_i)$, where i is the index of the repeated argument's mate.

Lemma 5.53. PRUNEARG *is sound and preserves refutational completeness of the calculus.*

Proof. The rule is sound because it simply applies a substitution to C. It preserves completeness because the premise C is redundant w.r.t. the conclusion $C\sigma$. This is because the sets of ground instances of C and $C\sigma$ are the same and $C \sqsupset C\sigma$. Clearly $C\sigma$ is an instance of C. We will show the inverse: that C is an instance of $C\sigma$. Let $\rho = \{y' \mapsto \lambda \bar{x}_{j-1}\, x_{j+1} \ldots x_k.\, y\, \bar{x}_{j-1}\, (t\, \bar{x}_{j-1}\, x_{j+1} \ldots x_k)\, x_{j+1} \ldots x_k\}$. We show $C\sigma\rho = C$. Consider an occurrence of y in C. By the side conditions, it will have the form $y\, \bar{s}_k\, \bar{u}$, where $s_j = t\, \bar{s}_{j-1}\, s_{j+1} \ldots s_k$. Hence, $(y\, \bar{s}_k)\sigma\rho = (y'\, \bar{s}_{j-1}\, s_{j+1} \ldots s_k)\rho = y\, \bar{s}_{j-1}\, (t\, \bar{s}_{j-1}\, s_{j+1} \ldots s_k)\, s_{j+1} \ldots s_k = y\, \bar{s}_k$. Thus, $C\sigma\rho = C$. □

We designed an algorithm that efficiently computes the subterm u of the term $t = (\lambda x_1 \ldots x_{j-1}\, x_{j+1} \ldots x_k.\, u)$ occurring in the side conditions of PRUNEARG. The algorithm is incomplete, but our tests suggest that it discovers most cases of prunable arguments that occur in practice. The algorithm works by maintaining a mapping of pairs (y, i) of functional variables y and indices i of their arguments to a set of candidate terms for u. For an occurrence $y\, \bar{s}_n$ of y and for an argument s_j, the algorithm approximates this set by computing all possible ways in which subterms of s_j that are equal to any other s_i can be replaced with the variable x_i corresponding to the ith argument of y. The candidate sets for all occurrences of y are then intersected. An arbitrary element of the final intersection is returned as the term u.

For example, suppose that $y\, \mathsf{a}\, (\mathsf{f}\, \mathsf{a})\, \mathsf{b}$ and $y\, z\, (\mathsf{f}\, z)\, \mathsf{b}$ are the only occurrences of y in the clause C. The initial mapping is $\{1 \mapsto \mathcal{T}_\mathrm{H}, 2 \mapsto \mathcal{T}_\mathrm{H}, 3 \mapsto \mathcal{T}_\mathrm{H}\}$. After computing the ways in which each argument can be expressed using the remaining ones for the first occurrence and intersecting the sets, we get $\{1 \mapsto \{\mathsf{a}\}, 2 \mapsto \{\mathsf{f}\, \mathsf{a}, \mathsf{f}\, x_1\}, 3 \mapsto \{\mathsf{b}\}\}$, where x_1 represents y's first argument. Finally, after computing the corresponding sets for the second occurrence of y and intersecting them with the previous candidate sets, we get $\{1 \mapsto \emptyset, 2 \mapsto \{\mathsf{f}\, x_1\}, 3 \mapsto \{\mathsf{b}\}\}$. The final mapping shows that we can remove the second argument, since it can be expressed as a function of the first argument: $t = (\lambda x_1\, x_3.\, \mathsf{f}\, x_1\, x_3)$. We can also remove the third argument, since its value is fixed: $t = (\lambda x_1\, x_3.\, \mathsf{b})$. An example where our procedure fails is the pair of occurrences $y\, (\lambda x.\, \mathsf{a})\, (\mathsf{f}\, \mathsf{a})\, \mathsf{c}$ and $y\, (\lambda x.\, \mathsf{b})\, (\mathsf{f}\, \mathsf{b})\, \mathsf{d}$. PRUNEARG can be used to eliminate the second argument by taking $t := (\lambda x_1\, x_3.\, \mathsf{f}\, (x_1\, x_3))$, but our algorithm will not detect this.

Following the literature [68, 126], we provide a rule for negative extensionality:

$$\frac{C' \vee s \not\approx s'}{C' \vee s\, (\mathsf{sk}\langle \bar{\alpha} \rangle\, \bar{y}) \not\approx s'\, (\mathsf{sk}\langle \bar{\alpha} \rangle\, \bar{y})}\ \textsc{NegExt}$$

5

where the following conditions apply:

1. sk is a fresh Skolem symbol;
2. $s \not\approx s'$ is $\succsim$-eligible in the premise;
3. $\bar{\alpha}$ and $\bar{y}$ are the type and term variables occurring free in the literal $s \not\approx s'$.

Negative extensionality can be applied as an inference rule at any time or as a simplification rule during preprocessing of the initial problem. The rule uses Skolem terms $\mathsf{sk}\,\bar{y}$ rather than $\mathsf{diff}\,s\,s'$ because they tend to be more compact.

Lemma 5.54 (NegExt's satisfiability preservation)**.** *Let $N \subseteq \mathcal{C}_{\mathrm{H}}$ and let E be the conclusion of a* NegExt *inference from N. If $N \cup \{(\textsc{Ext})\}$ is satisfiable, then $N \cup \{(\textsc{Ext}), E\}$ is satisfiable.*

Proof. Let $\mathcal{I}$ be a model of $N \cup \{(\textsc{Ext})\}$. We need to construct a model of $N \cup \{(\textsc{Ext}), E\}$. Since (Ext) holds in $\mathcal{I}$, so does its instance $s\,(\mathsf{diff}\,s\,s') \not\approx s'\,(\mathsf{diff}\,s\,s') \vee s \approx s'$. We extend the model $\mathcal{I}$ to a model $\mathcal{I}'$, interpreting sk such that $\mathcal{I}' \models \mathsf{sk}\langle\bar{\alpha}\rangle\,\bar{y} \approx \mathsf{diff}\,s\,s'$. The Skolem symbol sk takes the free type and term variables of $s \not\approx s'$ as arguments, which include all the free variables of $\mathsf{diff}\,s\,s'$, allowing us to extend $\mathcal{I}$ in this way.

By assumption, the premise $C' \vee s \not\approx s'$ is true in $\mathcal{I}$ and hence in $\mathcal{I}'$. Since the above instance of (Ext) holds in $\mathcal{I}$, it also holds in $\mathcal{I}'$. Hence, the conclusion $C' \vee s\,(\mathsf{sk}\langle\bar{\alpha}_m\rangle\,\bar{y}_n) \not\approx s'\,(\mathsf{sk}\langle\bar{\alpha}_m\rangle\,\bar{y}_n)$ also holds, which can be seen by resolving the premise against the (Ext) instance and unfolding the defining equation of sk. □

One reason why the extensionality axiom is so prolific is that both sides of its maximal literal, $y\,(\mathsf{diff}\,y\,z) \not\approx z\,(\mathsf{diff}\,y\,z)$, are fluid. As a pragmatic alternative to the axiom, we introduce the "abstracting" rules AbsSup, AbsERes, and AbsEFact with the same premises as the core Sup, ERes, and EFact, respectively. We call these rules collectively Abs. Each new rule shares all the side conditions of the corresponding core rule except that of the form $\sigma \in \mathrm{CSU}(s,t)$. Instead, it lets σ be the most general unifier of s and t's types and adds this condition: Let $v\langle s_1,\ldots,s_n\rangle = s\sigma$ and $v\langle t_1,\ldots,t_n\rangle = t\sigma$, where $v\langle\,\rangle$ is the largest common green context of $s\sigma$ and $t\sigma$. If any s_i is of functional type and the core rule has conclusion $E\sigma$, the new rule has conclusion $E\sigma \vee s_1 \not\approx t_1 \vee \cdots \vee s_n \not\approx t_n$. The NegExt rule can then be applied to those literals $s_i \not\approx t_i$ whose sides have functional type. Essentially the same idea was proposed by Bhayat and Reger as *unification with abstraction* in the context of combinatory superposition [28, Section 3.1]. The approach regrettably does not fully eliminate the need for axiom (Ext), as Visa Nummelin demonstrated via the following example.

Example 5.55. Consider the unsatisfiable clause set consisting of $\mathsf{h}\,x \approx \mathsf{f}\,x$, $\mathsf{k}\,\mathsf{h} \approx \mathsf{k}\,\mathsf{g}$, and $\mathsf{k}\,\mathsf{g} \not\approx \mathsf{k}\,\mathsf{f}$, where k takes at most one argument and $\mathsf{h} \succ \mathsf{g} \succ \mathsf{f}$. The only nonredundant Abs inference applicable is AbsERes on the third clause, resulting in $\mathsf{g} \not\approx \mathsf{f}$. Applying ExtNeg further produces $\mathsf{g}\,\mathsf{sk} \not\approx \mathsf{f}\,\mathsf{sk}$. The set consisting of all five clauses is saturated.

A different approach is to instantiate the extensionality axiom with arbitrary terms s, s' of the same functional type:

$$\frac{}{s\,(\mathsf{diff}\,s\,s') \not\approx s'\,(\mathsf{diff}\,s\,s') \vee s \approx s'}\ \textsc{ExtInst}$$

We would typically choose s, s' among the green subterms occurring in the current clause set. Intuitively, if we think in terms of eligibility, ExtInst demands $s(\mathsf{diff}\, s s') \approx s'(\mathsf{diff}\, s s')$ to be proved before $s \approx s'$ can be used. This can be advantageous because simplifying inferences (based on matching) will often be able to rewrite the applied terms $s\,(\mathsf{diff}\, s\, s')$ and $s'\,(\mathsf{diff}\, s\, s')$. In contrast, Abs assume $s \approx s'$ and delay the proof obligation that $s\,(\mathsf{diff}\, s\, s') \approx s'\,(\mathsf{diff}\, s\, s')$. This can create many long clauses, which will be subject to expensive generating inferences (based on full unification).

Superposition can be generalized to orange subterms as follows:

$$\frac{D' \vee t \approx t' \quad C' \vee s\langle\!\langle \bar{x}.\, u \rangle\!\rangle \mathrel{\dot{\approx}} s'}{(D' \vee C' \vee s\langle\!\langle \bar{x}.\, t' \rangle\!\rangle_\eta \mathrel{\dot{\approx}} s')\sigma\rho}\ \lambda\textsc{Sup}$$

where the substitution ρ is defined as follows: Let $P_y = \{y\}$ for all type and term variables $y \notin \bar{x}$. For each i, let P_{x_i} be recursively defined as the union of all P_y such that y occurs free in the λ-expression that binds x_i in $s\langle\!\langle \bar{x}.\, u \rangle\!\rangle\sigma$ or that occurs free in the corresponding subterm of $s\langle\!\langle \bar{x}.\, t' \rangle\!\rangle_\eta\sigma$. Then ρ is defined as $\{x_i \mapsto \mathsf{sk}_i\langle \bar{\alpha}_i \rangle \bar{y}_i$ for each $i\}$, where $\bar{y}_i$ are the term variables in P_{x_i} and $\bar{\alpha}_i$ are the type variables in P_{x_i} and the type variables occurring in the type of the λ-expression binding x_i. In addition, Sup's side conditions and the following conditions apply:

10. $\bar{x}$ has length $n > 0$;
11. $\bar{x}\sigma = \bar{x}$;
12. the variables $\bar{x}$ do not occur in $y\sigma$ for all variables y in u.

The substitution ρ introduces Skolem terms to represent bound variables that would otherwise escape their binders. The rule can be justified in terms of paramodulation and extensionality, with the Skolem terms standing for diff terms. We can shorten the derivation of Example 5.17 by applying this rule to the clauses C_{div} and C_{conj} as follows:

$$\frac{n \approx \mathsf{zero} \vee \mathsf{div}\, n\, n \approx \mathsf{one} \quad \mathsf{prod}\, K\, (\lambda k.\, \mathsf{div}\, (\mathsf{succ}\, k)\, (\mathsf{succ}\, k)) \not\approx \mathsf{one}}{\mathsf{succ}\, \mathsf{sk} \approx \mathsf{zero} \vee \mathsf{prod}\, K\, (\lambda k.\, \mathsf{one}) \not\approx \mathsf{one}}\ \lambda\textsc{Sup}$$

From this conclusion, $\bot$ can be derived using only Sup and EqRes inferences. We thus avoid both FluidSup and (Ext).

Lemma 5.56 (λSup's satisfiability preservation)**.** *Let $N \subseteq \mathcal{C}_{\mathrm{H}}$ and let E be the conclusion of a λSup inference from N. If $N \cup \{(\textsc{Ext})\}$ is satisfiable, then $N \cup \{(\textsc{Ext}), E\}$ is satisfiable.*

Proof. Let $\mathcal{I}$ be a model of $N \cup \{(\textsc{Ext})\}$. We need to construct a model of $N \cup \{(\textsc{Ext}), E\}$. For each i, let v_i be the λ-expression binding x_i in the term $s\langle\!\langle \bar{x}.\, u \rangle\!\rangle\sigma$ in the rule. Let v_i' be the variant of v_i in which the relevant occurrence of $u\sigma$ is replaced by $t'\sigma$. We define a substitution π recursively by $x_i\pi = \mathsf{diff}\,(v_i\pi)(v_i'\pi)$ for all i. This definition is well founded because the variables x_j with $j \geq i$ do not occur freely in v_i and v_i'. We extend the model $\mathcal{I}$ to a model $\mathcal{I}'$, interpreting sk_i such that $\mathcal{I}' \models \mathsf{sk}_i\langle \bar{\alpha}_i \rangle \bar{y}_i \approx \mathsf{diff}(v_i\pi)(v_i'\pi)$ for each i. Since the free type and term variables of any $x_i\pi$ are necessarily contained in P_{x_i}, the arguments of sk_i include the free variables of $\mathsf{diff}\,(v_i\pi)(v_i'\pi)$, allowing us to extend $\mathcal{I}$ in this way.

By assumption, the premises of the λSup inference are true in $\mathcal{I}$ and hence in $\mathcal{I}'$. We need to show that the conclusion $(D' \vee C' \vee s\langle\!\langle \bar{x}.\, t' \rangle\!\rangle_\eta \mathrel{\dot{\approx}} s')\sigma\rho$ is also true in $\mathcal{I}'$. Let ξ be a valuation. If $\mathcal{I}', \xi \models (D' \vee C')\sigma\rho$, we are done. So we assume that $D'\sigma\rho$ and $C'\sigma\rho$ are false in $\mathcal{I}'$ under ξ. In the following, we omit '$\mathcal{I}', \xi \models$', but all equations ($\approx$) are meant to be true in $\mathcal{I}'$ under ξ. Assuming $D'\sigma\rho$ and $C'\sigma\rho$ are false, we will show inductively that $v_i\pi \approx v'_i\pi$ for all $i = k, \ldots, 1$. By this assumption, the premises imply that $t\sigma\rho \approx t'\sigma\rho$ and $s\langle\!\langle \bar{x}.\, u \rangle\!\rangle\sigma\rho \mathrel{\dot{\approx}} s'\sigma\rho$. Due to the way we constructed $\mathcal{I}'$, we have $w\pi \approx w\rho$ for any term w. Hence, we have $t\sigma\pi \approx t'\sigma\pi$. The terms $v_k\pi(\mathsf{diff}(v_k\pi)(v'_k\pi))$ and $v'_k\pi(\mathsf{diff}(v_k\pi)(v'_k\pi))$ are the respective result of applying π to the body of the λ-expressions v_k and v'_k. Therefore, by congruence, $t\sigma\pi \approx t'\sigma\pi$ and $t\sigma = u\sigma$ imply that $v_k\pi(\mathsf{diff}(v_k\pi)(v'_k\pi)) \approx v'_k\pi(\mathsf{diff}(v_k\pi)(v'_k\pi))$. The extensionality axiom then implies $v_k\pi \approx v'_k\pi$.

It follows directly from the definition of π that for all i, $v_i\pi(\mathsf{diff}(v_i\pi)(v'_i\pi)) = s_i\langle\!\langle v_{i+1}\pi \rangle\!\rangle$ and $v'_i\pi(\mathsf{diff}(v_i\pi)(v'_i\pi)) = s_i\langle\!\langle v'_{i+1}\pi \rangle\!\rangle$ for some context $s_i\langle\!\langle\, \rangle\!\rangle$. The subterms $v_{i+1}\pi$ of $s_i\langle\!\langle v_{i+1}\pi \rangle\!\rangle$ and $v'_{i+1}\pi$ of $s_i\langle\!\langle v'_{i+1}\pi \rangle\!\rangle$ may be below applied variables but not below λs. Since substitutions avoid capture, in v_i and v'_i, π only substitutes x_j with $j < i$, but in v_{i+1} and v'_{i+1}, it substitutes all x_j with $j \leq i$. By an induction using these equations, congruence, and the extensionality axiom, we can derive from $v_k\pi \approx v'_k\pi$ that $v_1\pi \approx v'_1\pi$. Since $\mathcal{I}' \models w\pi \approx w\rho$ for any term w, we have $v_1\rho \approx v'_1\rho$. By congruence, it follows that $s\langle\!\langle \bar{x}.\, u \rangle\!\rangle\sigma\rho \approx s\langle\!\langle \bar{x}.\, t' \rangle\!\rangle_\eta\sigma\rho$. With $s\langle\!\langle \bar{x}.\, u \rangle\!\rangle\sigma\rho \mathrel{\dot{\approx}} s'\sigma\rho$, it follows that $(s\langle\!\langle \bar{x}.\, t' \rangle\!\rangle_\eta \mathrel{\dot{\approx}} s')\sigma\rho$. Hence, the conclusion of the λSup inference is true in $\mathcal{I}'$. □

The next rule, *duplicating flex subterm superposition*, is a lightweight alternative to FluidSup:

$$\frac{D' \vee t \approx t' \quad C' \vee s\langle y\, \bar{u}_n \rangle \mathrel{\dot{\approx}} s'}{(D' \vee C' \vee s\langle z\, \bar{u}_n\, t' \rangle \mathrel{\dot{\approx}} s')\rho\sigma}\ \textsc{DupSup}$$

where $n > 0$, $\rho = \{y \mapsto \lambda\bar{x}_n.\, z\, \bar{x}_n\, (w\, \bar{x}_n)\}$, and $\sigma \in \mathrm{CSU}(t, w(\bar{u}_n\rho))$ for fresh variables w, z. The order and eligibility restrictions are as for Sup. The rule can be understood as the composition of an inference that applies the substitution ρ and of a paramodulation inference into the subterm $w(\bar{u}_n\rho)$ of $s\langle z(\bar{u}_n\rho)(w(\bar{u}_n\rho))\rangle$. DupSup is general enough to replace FluidSup in Examples 5.13 and 5.14 but not in Example 5.15. On the other hand, FluidSup's unification problem is usually a flex–flex pair, whereas DupSup yields a less explosive flex–rigid pair unless t is variable-headed.

The last rule, *flex subterm superposition*, is an even more lightweight alternative to FluidSup:

$$\frac{D' \vee t \approx t' \quad C' \vee s\langle y\, \bar{u}_n \rangle \mathrel{\dot{\approx}} s'}{(D' \vee C' \vee s\langle t' \rangle \mathrel{\dot{\approx}} s')\sigma}\ \textsc{FlexSup}$$

where $n > 0$ and $\sigma \in \mathrm{CSU}(t, y\, \bar{u}_n)$. The order and eligibility restrictions are as for Sup.

5.6. Implementation

We have implemented our calculus in the Zipperposition prover. We use the order $\succ_\lambda$ (Section 5.3.5) derived from the Knuth–Bendix order [87] and the lexicographic path

order [82]. We currently use the corresponding nonstrict order $\succeq_\lambda$ as $\succsim$.

Except for FLUIDSUP, the core calculus rules already existed in Zipperposition in a similar form. To improve efficiency, we extended the prover to use a higher-order generalization [136] of fingerprint indices [118] to find inference partners for all new binary inference rules. To speed up the computation of the SUP conditions, we omit the condition $C\sigma \not\precsim D\sigma$ in the implementation, at the cost of performing some additional inferences. Among the optional rules, we implemented λDEMOD, PRUNEARG, NEGEXT, ABS, EXTINST, λSUP, DUPSUP, and FLEXSUP. For λDEMOD and λSUP, demodulation, subsumption, and other standard simplification rules (as implemented in E [120]), we use pattern unification. For generating inference rules that require enumerations of complete sets of unifiers, we use the complete procedure of Vukmirović et al. [136]. It has better termination behavior, produces fewer redundant unifiers, and can be implemented more efficiently than procedures such as Jensen and Pietrzykowski's [76] and Snyder and Gallier's [123]. The set of fluid terms is overapproximated in the implementation by the set of terms that are either nonground λ-expressions or terms of the form $y\,\bar{u}_n$ with $n > 0$. To efficiently retrieve candidates for ABS inferences without slowing down superposition term indexing structures, we implemented dedicated indexing for clauses that are eligible for ABS inferences [138, Section 3.3].

Zipperposition implements a DISCOUNT-style given clause procedure [7]. The proof state is represented by a set A of active clauses and a set P of passive clauses. To interleave nonterminating unification with other computation, we added a set T containing possibly infinite sequences of scheduled inferences. These sequences are stored as finite instructions of how to compute the inferences. Initially, all clauses are in P. At each iteration of the main loop, the prover heuristically selects a *given clause* C from P. If P is empty, sequences from T are evaluated to generate more clauses into P; if no clause can be produced in this way, A is saturated and the prover stops. Assuming a given clause C could be selected, it is first simplified using A. Clauses in A are then simplified w.r.t. C, and any simplified clause is moved to P. Then C is added to A and all sequences representing nonredundant inferences between C and A are added to T. This maintains the invariant that all nonredundant inferences between clauses in A have been scheduled or performed. Then some of the scheduled inferences in T are performed and the conclusions are put into P.

We can view the above loop as an instance of the abstract Zipperposition loop prover ZL of Waldmann et al. [140, Example 34]. Their Theorem 32 allows us to obtain dynamic completeness for this prover architecture from our static completeness result (Theorem 54). This requires that the sequences in T are visited fairly, that clauses in P are chosen fairly, and that simplification terminates, all of which are guaranteed by our implementation.

The unification procedure we use returns a sequence of either singleton sets containing the unifier or an empty set signaling that a unifier is still not found. Empty sets are returned to give back control to the caller of unification procedure and avoid getting stuck on nonterminating problems. These sequences of unifier subsingletons are converted into sequences containing subsingletons of clauses representing inference conclusions.

5.7. Evaluation

We evaluated our prototype implementation of the calculus in Zipperposition with other higher-order provers and with Zipperposition's modes for less expressive logics. All of the experiments were performed on StarExec nodes equipped with Intel Xeon E5-2609 0 CPUs clocked at 2.40 GHz. Following CASC 2019,[1] we use 180 s as the CPU time limit.

We used both standard TPTP benchmarks [130] and Sledgehammer-generated benchmarks [106]. From the TPTP, version 7.2.0, we used 1000 randomly selected first-order (FO) problems in CNF, FOF, or TFF syntax without arithmetic and all 499 monomorphic higher-order theorems in TH0 syntax without interpreted Booleans and arithmetic. We partitioned the TH0 problems into those containing no λ-expressions (TH0λf, 452 problems) and those containing λ-expressions (TH0λ, 47 problems). The Sledgehammer benchmarks, corresponding to Isabelle's Judgment Day suite [40], were regenerated to target clausal higher-order logic. They comprise 2506 problems, divided in two groups: SH-λ preserves λ-expressions, whereas SH-ll encodes them as λ-lifted supercombinators [106] to make the problems accessible to λ-free clausal higher-order provers. Each group of problems is generated from 256 Isabelle facts (definitions and lemmas). Our results are publicly available.[2]

5

Evaluation of Extensions To assess the usefulness of the extensions described in Section 5.5, we fixed a *base* configuration of Zipperposition parameters. For each extension, we then changed the corresponding parameters and observed the effect on the success rate. The base configuration uses the complete variant of the unification procedure of Vukmirović et al. [136]. It also includes the optional rules NEGEXT and PRUNEARG, substitutes FLEXSUP for the highly explosive FLUIDSUP, and excludes axiom (EXT). The base configuration is not refutationally complete.

The rules NEGEXT (NE) and PRUNEARG (PA) were added to the base configuration because our informal experiments showed that they usually help. Fig. 5.1 confirms this, although the effect is small. In all tables, $+R$ denotes the inclusion of a rule R not present in the base, and $-R$ denotes the exclusion of a rule R present in the base. Numbers given in parentheses denote the number of problems that are solved only by the given configuration and no other configuration in the same table.

The rules λDEMOD (λD) and λSUP extend the calculus to perform some rewriting under λ-binders. While experimenting with the calculus, we noticed that, in some configurations, λSUP performs better when the number of fresh Skolem symbols it introduces overall is bounded by some parameter n. As Fig. 5.2 shows, inclusion of these rules has different effect on the two benchmark sets. Different choices of n for λSUP (denoted by λSn) do not seem to influence the success rate much.

The evaluation of the ABS and EXTINST rules and axiom (EXT), presented in Fig. 5.3, confirms our intuition that including the extensionality axiom is severely detrimental to performance. The +(EXT) configuration solved two unique problems on SH-λ benchmarks, but the success of the +(EXT) configuration on these problems

[1] http://tptp.cs.miami.edu/CASC/27/
[2] https://doi.org/10.5281/zenodo.4032969

	−NE,−PA	−NE	−PA	Base
TH0	446 (0)	446 (0)	447 (0)	447 (0)
SH-λ	431 (0)	433 (0)	433 (0)	436 (1)

Figure 5.1: Number of problems proved without rules included in the base configuration

	Base	+λD	+λS0	+λS1	+λS2	+λS4	+λS8	+λS1024
TH0	447 (0)	448 (0)	449 (0)	449 (0)	449 (0)	449 (0)	449 (0)	449 (0)
SH-λ	436 (1)	435 (4)	430 (1)	429 (0)	429 (0)	429 (0)	429 (0)	429 (0)

Figure 5.2: Number of problems proved using rules that perform rewriting under λ-binders

	Base	+Abs	+ExtInst	+(Ext)
TH0	447 (0)	450 (1)	450 (1)	376 (0)
SH-λ	436 (11)	430 (11)	402 (1)	365 (2)

Figure 5.3: Number of problems proved using rules that perform extensionality reasoning

	−FlexSup	Base	−FlexSup,+DupSup	−FlexSup,+FluidSup
TH0	446 (0)	447 (0)	448 (1)	447 (0)
SH-λ	469 (10)	436 (4)	451 (3)	461 (7)

Figure 5.4: Number of problems proved with rules that perform superposition into fluid terms

	FO	TH0λf	TH0λ	SH-ll	SH-λ
CVC4	539	424	31	696	650
Ehoh	681	418	–	691	–
Leo-III-uncoop	198	389	42	226	234
Leo-III-coop	582	**438**	43	683	674
Satallax-uncoop	–	398	43	489	507
Satallax-coop	–	432	43	602	616
Vampire	**729**	432	42	**718**	707
FOZip	399	–	–	–	–
@+FOZip	363	400	–	478	–
λfreeZip	395	398	–	538	–
λZip-base	388	408	39	420	436
λZip-pragmatic	396	411	33	496	503
λZip-full	177	339	34	353	361
Zip-uncoop	514	426	**46**	661	677
Zip-coop	625	434	**46**	710	**717**

Figure 5.5: Number of problems proved by the different provers

appears to be due to a coincidental influence of the axiom on heuristics—the axiom is not referenced in the generated proofs.

The FLEXSUP rule included in the base configuration did not perform as well as we expected. Even the FLUIDSUP and DUPSUP rules outperformed FLEXSUP, as shown in Fig. 5.4. This effect is especially visible on SH-λ benchmarks. On TPTP, the differences are negligible.

Most of the extensions had a stronger effect on SH-λ than on TH0. A possible explanation is that the Boolean-free TH0 benchmark subset consists mostly of problems that are simple to solve using most prover parameters. On the other hand, SH-λ benchmarks are of varying difficulty and can thus benefit more from changing prover parameters.

Main Evaluation We selected all contenders in the THF division of CASC 2019 as representatives of the state of the art: CVC4 1.8 prerelease [13], Leo-III 1.4 [126], Satallax 3.4 [42], and Vampire 4.4 [27]. We also included Ehoh [137], the λ-free clausal higher-order mode of E 2.4. Leo-III and Satallax are cooperative higher-order provers that can be set up to regularly invoke first-order provers as terminal proof procedures. To assess the performance of their core calculi, we evaluated them with first-order backends disabled. We denote these "uncooperative" configurations by Leo-III-uncoop and Satallax-uncoop respectively, as opposed to the standard versions Leo-III-coop and Satallax-coop.

To evaluate the overhead our calculus incurs on first-order or λ-free higher-order problems, we ran Zipperposition in first-order (FOZip) and λ-free (λfreeZip) modes, as well as in a mode that encodes curried applications using a distinguished binary symbol @ before using first-order Zipperposition (@+FOZip). We evaluated the implementation of our calculus in Zipperposition (λZip) in three configurations: base, pragmatic, and full. Pragmatic builds on base by disabling FLEXSUP and replacing complete unification with the pragmatic variant procedure pv^2_{1121} of Vukmirović et al. Full is a refutationally complete extension of base that substitutes FLUIDSUP for FLEXSUP and includes axiom (EXT). Finally, we evaluated Zipperposition in a portfolio mode that runs the prover in various configurations (Zip-uncoop). We also evaluated a cooperative version of the portfolio which, in some configurations, after a predefined time invokes Ehoh as backend on higher-order problems (Zip-coop). In this version, Zipperposition encodes heuristically selected clauses from the current proof state to lambda-free higher-order logic supported by Ehoh [137]. On first-order problems, we ran Ehoh, Vampire, and Zip-uncoop using the provers' respective first-order modes.

A summary of these experiments is presented in Figure 5.5. In the pragmatic configuration, our calculus outperformed λfreeZip on TH0λf problems and incurred less than 1% overhead compared with FOZip, but fell behind λfreeZip on SH-ll problems. The full configuration suffers greatly from the explosive extensionality axiom and FLUIDSUP rule.

Except on TH0λ problems, both base and pragmatic configurations outperformed Leo-III-uncoop, which runs a fixed configuration, by a substantial margin. Zip-uncoop outperformed Satallax-uncoop, which uses a portfolio. Our most competitive configuration, Zip-coop, emerges as the winner on both problem sets containing

λ-expressions.

On higher-order TPTP benchmarks this configuration does not solve any problems that no other (cooperative) higher-order prover solves. By contrast, on SH-ll benchmarks Zip-coop solves 21 problems no other higher-order prover solves, and on SH-λ benchmarks, it uniquely solves 27 problems.

5.8. Conclusion

We presented the Boolean-free λ-superposition calculus, which targets a clausal fragment of extensional polymorphic higher-order logic. With the exception of a functional extensionality axiom, it gracefully generalizes standard superposition. Our prototype prover Zipperposition shows promising results on TPTP and Isabelle benchmarks.

Our calculus is based on the extensional nonpurifying calculus from Chapter 3. Initially, we considered to extend the other calculi as well. However, as we extended their work to support λ-expressions, we found the purification approach problematic and gave it up because it needs x to be smaller than $x\ t$, which is impossible to achieve with a term order on $\beta\eta$-equivalence classes. We also quickly gave up our attempt at supporting intensional higher-order logic. Extensionality is the norm for higher-order unification [53] and is mandated by the TPTP THF format [131] and in proof assistants such as HOL4, HOL Light, Isabelle/HOL, Lean, Nuprl, and PVS.

6

Superposition with Interpreted Booleans

Joint work with
Visa Nummelin, Sophie Tourret, and Petar Vukmirović

Before we extend our Boolean-free λ-superposition calculus to full higher-order logic, we first investigate how the first-order superposition calculus can be extended with Booleans and how clausification can be interleaved with other derivation steps. Besides being the basis for higher-order superposition, the calculus presented in this chapter works efficiently on first-order problems that would be obfuscated when using clausification as preprocessing, and it avoids the costly axiomatic encoding of the theory of Booleans into first-order logic.

The contents of this chapter are part of ongoing work with Visa Nummelin, Sophie Tourret, and Petar Vukmirović. I include parts of this work here because it forms the basis of Chapter 7 and has not been published yet. The main author is Nummelin. I contributed the core ideas of the calculus and of the ground completeness proof, which my coauthors improved and elaborated on.

6.1. Introduction

Standard superposition operates on problems given in clausal normal form (CNF). Sometimes, clausifying a problem into CNF can obfuscate an originally simple problem and thus hamper proof search. For example, given a conjecture of the form $\phi \to \phi$ where ϕ is some complicated formula, clausification can produce a variety of clauses, concealing that the problem is provable without even inspecting ϕ in detail. Especially when ϕ contains equivalences, clauses multiply quickly during clausification. For users of theorem provers, for instance within proof assistants, it can be frustrating to see provers fail on such seemingly trivial problems.

Ganzinger and Stuber [62] presented an approach to combat this issue by delaying clausification and interleaving it with the superposition calculus. They show that on set theoretic benchmarks delayed clausification can substantially reduce the number of derived clauses. With their approach, many equivalences do not need to be clausified and can be used for rewriting.

We rework Ganzinger and Stuber's approach to delayed clausification into a sound and refutationally complete calculus for first-order logic with an interpreted Boolean type. In standard first-order logic, Booleans are second-class citizens. The logic strictly separates terms, which can be typed with uninterpreted types, and formulas, which effectively have Boolean type, but which are only allowed to appear at the surface and not within terms. We lift this restriction, following Kotelnikov et al. [93]. In this thesis, our main motivation is to use the calculus as a basis for a calculus for higher-order logic. However, this extension of first-order logic is useful in itself for problems coming from program verification or proof assistants since they often contain functions with Boolean arguments or variables of Boolean type.

Furthermore, the term order requirements of our calculus are less restrictive than Ganzinger and Stuber's. In addition to the lexicographic path order, we also support the Knuth–Bendix order, which is known to yield better results in superposition provers. Moreover, Ganzinger and Stuber's rules work from the top down, prioritizing the topmost connective or quantifier. Our calculus is more flexible as its rules can manipulate Boolean subterms at any position. To restrict the number of inferences on Boolean subterms, we employ selection functions and simplification rules that allow us to steer the clausification process precisely. In particular, our calculus is parameterized by a Boolean subterm selection function, a mechanism resembling literal selection. We developed it based on a short paragraph in Ganzinger and Stuber's work.

Besides Ganzinger and Stuber, our work is based on work by Kotelnikov et al. on FOOL, which is essentially first-order logic with interpreted Booleans. Kotelnikov et al.'s first approach [93] describes an encoding of Booleans into first-order logic. The result is then clausified with a standard first-order clausification procedure. Kotelnikov et al.'s second approach [92] integrates the encoding of Booleans into the CNF procedure, yielding a clausified problem that superposition provers are better prepared to deal with. Our work takes Kotelnikov et al.'s approaches a step further by integrating clausification and all Boolean reasoning into the core calculus.

The main advantage over Kotelnikov et al.'s approaches is that the powerful simplification machinery of the superposition calculus can already be put to work

before the problem is clausified. For instance, this allows us to recognize trivial patterns such as formulas of the form $\phi \rightarrow \phi$ early and to rewrite using equivalences. Also more advanced simplification techniques such as subsumption resolution (also known as contextual literal cutting) can be applied on the formula level. A second advantage of delaying clausification is that many heuristics such as selection functions and the choice of the next given clause can be based on formulas instead of clauses.

On the other hand, Kotelnikov et al.'s second approach has the advantage that it eliminates all Boolean equalities during clausification, eliminating the need for a modification of the calculus. This approach is compatible neither with delayed clausification nor with higher-order reasoning because new Boolean equalities may emerge during the derivation.

In this chapter, we introduce first-order logic with interpreted Booleans (Section 6.2), present a ground calculus for the logic (Section 6.3), and prove it to be refutationally complete (Section 6.4). The nonground version of this calculus, simplification rules, and an empirical evaluation are in progress [109].

6.2. Logic

Our logic is a first-order logic with an interpreted Boolean type. It is essentially identical to Kotelnikov et al.'s FOOL [93], but does not include let- and if-then-else-expressions.

Syntax We fix a set Σ_{ty} of type constructors with associated arities. We require that Σ_{ty} contains the nullary type constructor o of Booleans. A type is inductively defined to be of the form $\kappa(\bar{\tau}_n)$ for an n-ary type constructor $\kappa \in \Sigma_{\mathsf{ty}}$ and types $\bar{\tau}_n$. We write κ for $\kappa()$. A type declaration is an expression of the form $\bar{\tau}_n \Rightarrow \upsilon$ for types $\bar{\tau}_n$ and υ. If $n = 0$, we simply write υ for $() \Rightarrow \upsilon$.

We fix a set Σ of (function) symbols f, each associated with a type declaration $\bar{\tau}_n \Rightarrow \upsilon$, written as $\mathsf{f} : \bar{\tau}_n \Rightarrow \upsilon$ or f, and a countably infinite set $\mathcal{V}$ of variables with associated types, written as $x : \tau$ or x. The notation $t : \tau$ will also be used to indicate the type of arbitrary terms t. We require that Σ contains the logical symbols $\mathbf{\top}, \mathbf{\bot} : o$; $\mathbf{\neg} : o \Rightarrow o$; $\mathbf{\wedge}, \mathbf{\vee}, \mathbf{\rightarrow} : (o \times o) \Rightarrow o$; and $\mathbf{\approx}, \mathbf{\not\approx} : (\tau \times \tau) \Rightarrow o$ for each type τ. The logical symbols are printed in bold to distinguish them from the notation used for clauses below. We use infix notation for the binary logical symbols. Moreover, we require that there is at least one nullary symbol for each type to avoid empty Herbrand universes.

A signature is a pair $(\Sigma_{\mathsf{ty}}, \Sigma)$. The set of *terms* is defined inductively as follows. Every $x : \tau \in \mathcal{V}$ is a term of type τ. If $\mathsf{f} : \bar{\tau}_n \Rightarrow \upsilon \in \Sigma$ and $\bar{t}_n : \bar{\tau}_n$ is a tuple of terms, then the application $\mathsf{f}(\bar{t}_n)$ (or simply f if $n = 0$) is a term of type υ. If $x : \tau$ and $t : o$, then the quantifier-headed terms $\forall x.\, t$ and $\exists x.\, t$ are terms of Boolean type. We view quantifier-headed terms modulo α-renaming.

The head of a term is x if the term is a variable x; it is f if the term is an application $\mathsf{f}(\bar{t}_n)$; and it is $\mathcal{Q}$ if the term is a quantifier-headed term $\mathcal{Q}x.\, t$. Here and elsewhere, we let $\mathcal{Q}$ stand for either $\forall$ or $\exists$.

A variable occurrence is *free* in a term if it is not bound by $\forall$ or $\exists$. A term is *ground* if it contains no free variables. We write $\mathcal{T}_{\mathrm{G}}$ for the set of all ground terms.

A literal is an equation $s \approx t$ or a disequation $s \not\approx t$. We write $s \mathrel{\dot\approx} t$ for a literal that can be either an equation or a disequation. Unlike terms constructed using the function symbols $\approx$ and $\not\approx$, literals are unordered—i.e., $s \mathrel{\dot\approx} t$ and $t \mathrel{\dot\approx} s$ denote the same literal. A clause $L_1 \vee \cdots \vee L_n$ is a finite multiset of literals L_j. The empty clause is written as $\bot$.

Terms t of Boolean type are not literals. They must be encoded as $t \approx \top$ and $t \approx \bot$. Both of these are considered positive literals because they are equations, not disequations. We also considered to make the calculus operate on positive literals only and to encode negative literals $s \not\approx t$ as $(s \mathbin{\not\approx} t) \approx \bot$, following Ganzinger and Stuber. However, this approach requires an additional term order condition to make the conclusion of EFact small enough, excluding the Knuth–Bendix order. To support both the Knuth–Bendix order and the lexicographic path order, we allow negative literals. As a consequence, the truth of a Boolean term can be expressed as $t \approx \top$ or $t \not\approx \bot$ and the falsity of a Boolean term can be expressed as $t \approx \bot$ or $t \not\approx \top$. Fortunately, the simplification mechanism in the next chapter will allow us to simplify negative literals of the form $t \mathbin{\not\approx} \bot$ and $t \mathbin{\not\approx} \top$ into $t \approx \top$ and $t \approx \bot$, respectively, eliminating the redundancy.

Subterms and positions are inductively defined as follows. A position in a term is a tuple of natural numbers. For any term t, the empty position ε is a position of t, and t is the subterm of t at position ε. If t is the subterm of u_i at position p, then $i.p$ is a position of $\mathsf{f}(\bar u)$, and t is the subterm of $\mathsf{f}(\bar u)$ at position $i.p$. If t is the subterm of u at position p, then $1.p$ is a position of $\mathbb{Q}x.\, u$, and t is the subterm of $\mathbb{Q}x.\, u$ at position $1.p$.

For positions in clauses, natural numbers are not appropriate because clauses and literals are unordered. A position in a clause C is a tuple $L.s.p$ where $L = s \mathrel{\dot\approx} t$ is a literal in C and p is a position in s. The subterm of C at position $L.s.p$ is the subterm of s at position p.

We write $s|_p$ to denote the subterm at position p in s. We write $s[u]_p$ to denote a term s with the subterm u at position p and call $s[\,]_p$ a *context*; the position p may be omitted in this notation.

A position p is *at or below* a position q if q is a prefix of p. A position p is *below* a position q if q is a proper prefix of p.

Substitutions are defined as usual in first-order logic and they rename quantified variables to avoid capture.

Semantics An interpretation $\mathcal{I} = (\mathcal{U}, \mathcal{J})$ is a pair, consisting of a universe $\mathcal{U}_\tau$ for each type τ and an *interpretation function* $\mathcal{J}$, which associates with each symbol $\mathsf{f} : \bar\tau \Rightarrow \upsilon$ and universe elements $\bar a \in \mathcal{U}_{\bar\tau}$ a universe element $\mathcal{J}(\mathsf{f})(\bar a) \in \mathcal{U}_\upsilon$. We require that $\mathcal{U}_o = \{0,1\}$; $\mathcal{J}(\top) = 1$; $\mathcal{J}(\bot) = 0$; $\mathcal{J}(\neg)(a) = 1 - a$; $\mathcal{J}(\wedge)(a,b) = \min\{a,b\}$; $\mathcal{J}(\vee)(a,b) = \max\{a,b\}$; $\mathcal{J}(\rightarrow)(a,b) = \max\{1-a,b\}$; $\mathcal{J}(\approx)(c,d) = 1$ if $c = d$ and 0 otherwise; $\mathcal{J}(\not\approx)(c,d) = 0$ if $c = d$ and 1 otherwise; for all $a,b \in \mathcal{U}_o$ and $c,d \in \mathcal{U}_\tau$ where τ is a type.

A *valuation* is a function assigning an element $\xi(x) \in \mathcal{U}_\tau$ to each variable $x : \tau$. For an interpretation $\mathcal{I}$ and a valuation ξ, the denotation of a term is inductively defined as $[\![x]\!]^\xi_{\mathcal{I}} = \xi(x)$ for a variable $x \in \mathcal{V}$; $[\![\mathsf{f}(\bar t)]\!]^\xi_{\mathcal{I}} = \mathcal{J}(\mathsf{f})([\![\bar t]\!]^\xi_{\mathcal{I}})$ for a symbol $\mathsf{f} \in \Sigma$ and appropriately typed terms $\bar t$; and $[\![\forall x.\, t]\!]^\xi_{\mathcal{I}} = \min\{[\![t]\!]^{\xi[x\mapsto a]}_{\mathcal{I}} \mid a \in \mathcal{U}_\tau\}$, $[\![\exists x.\, t]\!]^\xi_{\mathcal{I}} = \max\{[\![t]\!]^{\xi[x\mapsto a]}_{\mathcal{I}} \mid a \in \mathcal{U}_\tau\}$ for a variable $x : \tau \in \mathcal{V}$ and a term $t : o$. For ground terms t,

the denotation does not depend on the choice of the valuation ξ, which is why we sometimes write $[\![t]\!]_{\mathcal{I}}$ for $[\![t]\!]_{\mathcal{I}}^{\xi}$.

Given an interpretation $\mathcal{I}$ and a valuation ξ, an equation $s \approx t$ is true if $[\![s]\!]_{\mathcal{I}}^{\xi}$ and $[\![t]\!]_{\mathcal{I}}^{\xi}$ are equal and it is false otherwise. A disequation $s \not\approx t$ is true if $s \approx t$ is false. A clause is true if at least one of its literals is true. A clause set is true if all its clauses are true. An interpretation $\mathcal{I}$ is a *model* of a clause set N, written $\mathcal{I} \models N$, if N is true in $\mathcal{I}$ for all valuations ξ.

6.3. The Calculus

In this chapter, we will only consider the ground calculus for first-order logic with interpreted Booleans because this is sufficient for the completeness proof of our higher-order calculus.

6.3.1. Parameters of Our Calculus

The calculus is parameterized by a term order, a literal selection function, a Boolean subterm selection function, and a witness function. These concepts are defined below.

Definition 6.1 (Term order)**.** A *term order* is a well-founded strict total order $\succ$ on ground terms such that

(O1) compatibility with contexts holds, but not necessarily below quantifiers;
(O2) the subterm property holds, but not necessarily below quantifiers;
(O3) $u \succ \bot \succ \top$ for any term u that is not $\top$ or $\bot$; and
(O4) $\mathrm{Q}x.\, t \succ t\{x \mapsto u\}$ for any term $u \in \mathcal{T}_{\mathrm{G}}$ whose only Boolean subterms are $\top$ and $\bot$.

Such term orders exist. For example, we can use the transfinite Knuth–Bendix order [104]. For (O3), we assign $\top$ and $\bot$ minimal weight and minimal precedence. To cope with quantifier terms, we encode them using De Bruin indices, which we compare as if they were ordinary symbols. For (O3), we use the weights $\mathcal{W}(\forall) = \mathcal{W}(\exists) = \omega$ and finite weights for all other symbols.

In addition to negative literals, literals of the form $s \approx \bot$ can be selected:

Definition 6.2 (Literal selection)**.** A *literal selection function* is a mapping from each clause to a subset of its literals that are negative or of the form $s \approx \bot$. The literals in this subset are called *selected*.

Moreover, Boolean subterms can be selected. This resembles an idea described by Ganzinger and Stuber [62, Section 7], but we can weaken their restrictions on the selection as follows:

Definition 6.3 (Boolean subterm selection)**.** A *Boolean subterm selection function* is a mapping from each clause C to a subset of the positions of Boolean subterms in C. The positions in this subset are called *selected*. Informally, we also say that the Boolean subterms at these positions are selected. The following restrictions apply:

– The Boolean subterms $\top$ or $\bot$ cannot be selected.
– Positions below quantifiers cannot be selected.
– The topmost position on either side of a positive literal cannot be selected.

The last parameter of our calculus is a witness function. This function will be used in the next chapter to produce applied Skolem symbols that serve as witnesses for quantifier-headed terms.

Definition 6.4 (Witness function)**.** Given a ground clause C and a position p of a quantifier-headed term in C that is not below another quantifier, a *witness function* returns a ground term $\mathbf{w}(C,p) \in \mathcal{T}_{\mathrm{G}}$. We require that $\mathrm{Q}x.\, v \succ v\{x \mapsto \mathbf{w}(C,p)\}$ if $C|_p = \mathrm{Q}x.\, v$.

For soundness of the calculus, we would also need to require that $\mathbf{w}$ produces terms whose head is a fresh constant, but we will ignore soundness in this chapter and focus on refutational completeness only.

6.3.2. The Inference Rules

For this calculus, we define eligibility on literals and on all positions of a clause:

Definition 6.5 (Eligibility)**.** A literal L is (*strictly*) *eligible* in C if it is selected in C or there are neither selected literals nor selected Boolean subterms in C and L is (strictly) maximal in C. (A selected literal is strictly eligible.)

The eligible positions of a clause C are inductively defined as follows.

(E1) Any selected position is eligible.

(E2) If a literal $s \mathrel{\tilde{\approx}} t$ with $s \succ t$ is either eligible and negative or strictly eligible and positive, then $L.s.\varepsilon$ is eligible.

(E3) If the position p is eligible and the head of $C|_p$ is not $\approx$, $\not\approx$, $\forall$, or $\exists$, the positions of all direct subterms are eligible.

(E4) If the position p is eligible and $C|_p$ is of the form $s \approx t$ or $s \not\approx t$, the maximal sides of this (dis)equation are eligible.

Our calculus is parameterized by a term order $\succ$, a literal selection function, a Boolean subterm selection function, and a witness function $\mathbf{w}$. The rules of our calculus are the following. The first three rules closely resemble standard superposition:

$$\frac{\overbrace{D' \vee t \approx t'}^{D} \qquad C[t]_p}{D' \vee C[t']}\ \textsc{Sup}$$

1. p is eligible in C;
2. $t \approx t'$ is strictly eligible in D;
3. $t \succ t'$;
4. $D \prec C$;
5. the head of t is not a logical symbol;
6. if $t' = \bot$, the position p is at the top level of a positive equation.

$$\frac{C' \vee u \not\approx u}{C'}\ \textsc{ERes} \qquad\qquad \frac{C' \vee u \approx v' \vee u \approx v}{C' \vee v \not\approx v' \vee u \approx v'}\ \textsc{EFact}$$

For ERes we require that $u \not\approx u$ is eligible in the premise. For EFact we require:

1. $u \succ v$;
2. $u \approx v$ is maximal in the premise;

3. no literal is selected in the premise;
4. no Boolean subterms are selected in the premise.

The following BOOLHOIST rule enforces that T and $\bot$ are the only Boolean values. It resembles FOOL paramodulation [93, Section 4] but is restricted to nonlogical symbols.

$$\frac{C[u]}{C[\bot] \vee u \approx \mathsf{T}}\,\textsc{BoolHoist}$$

1. u is a Boolean term whose head is not a logical symbol
2. the position of u is eligible in C
3. the subterm u is not at the top level of a positive literal

The rule FALSEELIM is responsible for resolving literals of the form $\bot \approx \mathsf{T}$:

$$\frac{C' \vee \bot \approx \mathsf{T}}{C'}\,\textsc{FalseElim}$$

where $\bot \approx \mathsf{T}$ is strictly eligible in the premise.

Finally, we introduce rules to handle the various logical symbols. For each logical symbol, we need to consider the case where the term is false and the case where it is true. Whenever possible, we prefer rules that rewrite the Boolean subterm in place (with names ending in RW). When this cannot be done in a sound way, we resort to rules hoisting the Boolean subterm into a dedicated literal (with names ending in HOIST).

$$\frac{C[s \approx t]}{C[\bot] \vee s \approx t}\,\textsc{EqHoist} \qquad \frac{C[s \not\approx t]}{C[\mathsf{T}] \vee s \approx t}\,\textsc{NeqHoist}$$

$$\frac{C[\forall x.\, t]}{C[\bot] \vee t\{x \mapsto u\} \approx \mathsf{T}}\,\textsc{ForallHoist} \qquad \frac{C[\exists x.\, t]}{C[\mathsf{T}] \vee t\{x \mapsto u\} \approx \bot}\,\textsc{ExistsHoist}$$

1. the position of the indicated subterm is eligible in C;
2. u is ground and its only Boolean subterms are T and $\bot$.

The rules FORALLHOIST and EXISTSHOIST must enumerate all possible u, which would be impractical in an implementation. In the nonground calculus of the next chapter, we will resolve this issue by using a fresh variable instead of u.

$$\frac{C[t]}{C[t']}\,\textsc{BoolRw}$$

1. (t, t') is one of the following pairs for some term s:

$(\neg\bot, \mathsf{T})$	$(\bot \wedge \bot, \bot)$	$(\bot \vee \bot, \bot)$	$(\bot \rightarrow \bot, \mathsf{T})$
$(\neg\mathsf{T}, \bot)$	$(\mathsf{T} \wedge \bot, \bot)$	$(\mathsf{T} \vee \bot, \mathsf{T})$	$(\mathsf{T} \rightarrow \bot, \bot)$
$(s \approx s, \mathsf{T})$	$(\bot \wedge \mathsf{T}, \bot)$	$(\bot \vee \mathsf{T}, \mathsf{T})$	$(\bot \rightarrow \mathsf{T}, \mathsf{T})$
$(s \not\approx s, \bot)$	$(\mathsf{T} \wedge \mathsf{T}, \mathsf{T})$	$(\mathsf{T} \vee \mathsf{T}, \mathsf{T})$	$(\mathsf{T} \rightarrow \mathsf{T}, \mathsf{T})$

2. the position of t is eligible in C.

$$\frac{C[\exists z.\, v]_p}{C[v\{z \mapsto \mathbf{w}(C,p)\}]_p}\ \textsc{ExistsRw} \qquad \frac{C[\forall z.\, v]_p}{C[v\{z \mapsto \mathbf{w}(C,p)\}]_p}\ \textsc{ForallRw}$$

1. the position p is eligible in C;
2a. for ForallRw, $C[\mathsf{T}]_p$ is not a tautology;
2b. for ExistsRw, $C[\bot]_p$ is not a tautology.

6.4. Refutational Completeness

In the spirit of Bachmair and Ganzinger's completeness proof for first-order logic, our proof idea is, given a saturated set N such that $\bot \notin N$, to construct a term rewriting system—the candidate model. The construction of our candidate model is inspired by Ganzinger and Stuber's [62]. Finally, we employ Bachmair and Ganzinger's framework of reducing counterexamples [10, Section 4.2] to show static refutational completeness.

Our term rewriting systems are essentially standard first-order term rewriting systems. We generalize them to our terms with interpreted Booleans by treating all quantifier-headed terms as if they were constants, meaning that a term rewriting system does not rewrite below quantifiers. For example, the rewrite rule $\forall x.\, \mathsf{q} \longrightarrow \forall x.\, \mathsf{p}$ can rewrite $\mathsf{f}(\forall x.\, \mathsf{q})$ into $\mathsf{f}(\forall x.\, \mathsf{p})$, but the rewrite rule $\mathsf{q} \longrightarrow \mathsf{p}$ cannot. Term rewriting concepts such as confluence and critical pairs function as in standard first-order logic.

6.4.1. Viewing Term Rewriting Systems as Interpretations

It is well-known that in first-order logic, term rewriting systems can be used to describe interpretations. In the following, we will show under which requirements also a term rewriting system on our logic can be viewed as an interpretation.

Definition 6.6 (Viewing a rewriting system as an interpretation)**.** Let R be a rewriting system over $\mathcal{T}_\mathrm{G}$ such that

(I1) for all Boolean terms $t \in \mathcal{T}_\mathrm{G}$, either $t \leftrightarrow_R^* \mathsf{T}$ or $t \leftrightarrow_R^* \bot$;
(I2) $\neg\bot \leftrightarrow_R^* \mathsf{T}$; $\neg\mathsf{T} \leftrightarrow_R^* \bot$; and corresponding requirements for $\wedge$, $\vee$, and $\rightarrow$;
(I3) $s \approx s' \leftrightarrow_R^* \mathsf{T}$ if and only if $s \leftrightarrow_R^* s'$ for all $s, s' \in \mathcal{T}_\mathrm{G}$; and corresponding requirements for $\not\approx$;
(I4) $\forall x.\, s \leftrightarrow_R^* \mathsf{T}$ if and only if $s\{x \mapsto u\} \leftrightarrow_R^* \mathsf{T}$ for all $u \in \mathcal{T}_\mathrm{G}$; and corresponding requirements for $\exists$.

We define an interpretation $(\mathcal{U}, \mathcal{J})$ based on R. We will use R to denote both the rewriting system and the interpretation.

For each type τ, let $\mathcal{U}_\tau$ be the set of equivalence classes $[t]$ of terms $t \in \mathcal{T}_\mathrm{G}$ modulo $\leftrightarrow_R^*$. Let $\mathcal{J}(\mathsf{f})(a) = [\mathsf{f}(\bar{t})]$ where $\bar{t}$ are terms from the equivalence classes $\bar{a}$, respectively. This does not depend on the choice of $\bar{t}$ because if $\bar{t} \leftrightarrow_R^* \bar{t}'$, then $\mathsf{f}(\bar{t}) \leftrightarrow_R^* \mathsf{f}(\bar{t}')$.

We identify $[\mathsf{T}]$ with 1 and $[\bot]$ with 0. By (I1), this ensures that $\mathcal{U}_o = \{0, 1\}$, $\mathcal{J}(\mathsf{T}) = 1$, and $\mathcal{J}(\bot) = 0$. (I2) ensures that $\mathcal{J}(\neg)$, $\mathcal{J}(\wedge)$, $\mathcal{J}(\vee)$, and $\mathcal{J}(\rightarrow)$ adhere to the

requirements of an interpretation. (I3) ensures that $\mathcal{J}(\approx)$ and $\mathcal{J}(\not\approx)$ adhere to the requirements of an interpretation.

To show that this definition has the expected properties, we need the substitution lemma for our logic:

Lemma 6.7 (Substitution lemma)**.** *Let $\mathcal{I} = (\mathcal{U}, \mathcal{J})$ be an interpretation. Then*

$$[\![t\rho]\!]_{\mathcal{J}}^{\xi} = [\![t]\!]_{\mathcal{J}}^{\xi'}$$

for all terms t, all substitutions ρ, and all valuations ξ, ξ' such that $\xi'(x) = [\![x\rho]\!]_{\mathcal{J}}^{\xi}$ for all variables x.

Proof. We prove this by structural induction on t. If $t = x$ for some $x \in \mathcal{V}$, then

$$[\![t\rho]\!]_{\mathcal{J}}^{\xi} = \xi'(x) = [\![t]\!]_{\mathcal{J}}^{\xi'}$$

If $t = \mathsf{f}(\bar{t})$ for some $\mathsf{f} \in \Sigma$ and terms $\bar{t}$, then

$$[\![t\rho]\!]_{\mathcal{J}}^{\xi} = [\![\mathsf{f}(\bar{t}\rho)]\!]_{\mathcal{J}}^{\xi} = \mathcal{J}(\mathsf{f})([\![\bar{t}\rho]\!]_{\mathcal{J}}^{\xi}) \overset{\text{IH}}{=} \mathcal{J}(\mathsf{f})([\![\bar{t}]\!]_{\mathcal{J}}^{\xi'}) = [\![t]\!]_{\mathcal{J}}^{\xi'}$$

If $t = \forall x.s$ for some term s, then we can assume without loss of generality that $x\rho = x$ and that x does not appear in $y\rho$ for any $y \neq x$. We have

$$\begin{aligned}[\![t\rho]\!]_{\mathcal{J}}^{\xi} &= [\![\forall x.s\rho]\!]_{\mathcal{J}}^{\xi} = \min\{[\![s\rho]\!]_{\mathcal{J}}^{\xi[x\mapsto a]} \mid a \in \mathcal{U}_\tau\} \\ &\overset{\text{IH}}{=} \min\{[\![s]\!]_{\mathcal{J}}^{\xi'[x\mapsto a]} \mid a \in \mathcal{U}_\tau\} = [\![\forall x.s]\!]_{\mathcal{J}}^{\xi'} = [\![t]\!]_{\mathcal{J}}^{\xi'}\end{aligned}$$

The induction hypothesis is applicable here because $\xi'[x \mapsto a](x) = a = [\![x]\!]_{\mathcal{J}}^{\xi[x\mapsto a]} = [\![x\rho]\!]_{\mathcal{J}}^{\xi[x\mapsto a]}$ and for all $y \neq x$ we have $\xi'[x \mapsto a](y) = \xi'(y) = [\![y\rho]\!]_{\mathcal{J}}^{\xi} = [\![y\rho]\!]_{\mathcal{J}}^{\xi[x\mapsto a]}$. An analogous argument applies if $t = \exists x.s$. □

Lemma 6.8. *Under the requirements of Definition 6.6, we have $[\![t]\!]_R = [t]$ for all $t \in \mathcal{T}_{\mathrm{G}}$.*

Proof. By induction on the structure of t.

If $t = \mathsf{f}(\bar{s})$, then $[\![t]\!]_R = \mathcal{J}(\mathsf{f})([\![\bar{s}]\!]_R) \overset{\text{IH}}{=} \mathcal{J}(\mathsf{f})([s]) = [\mathsf{f}(s)] = [t]$. If $t = \exists x.s$, then, using the substitution lemma (Lemma 6.7), $[\![t]\!]_R = \min\{[\![s]\!]_R^{\{x\mapsto[u]\}} \mid u \in \mathcal{T}_{\mathrm{G}}\} = \min\{[\![s\{x \mapsto u\}]\!]_R \mid u \in \mathcal{T}_{\mathrm{G}}\} \overset{\text{IH}}{=} \min\{[s\{x \mapsto u\}] \mid u \in \mathcal{T}_{\mathrm{G}}\} \overset{(I4)}{=} [\forall x.s] = [t]$. If $t = \exists x.s$, we argue analogously. □

This lemma shows that $R \models t \approx t'$ if and only if $t \longleftrightarrow_R^* t'$, as in first-order logic. In the next step, we will define a closure operation on term rewriting systems that allows us to enforce that the conditions of Definition 6.6 are fulfilled.

Definition 6.9 (Boolean closure of a term rewriting system)**.** Let R be a term rewriting system over $\mathcal{T}_{\mathrm{G}}$. We define R^s and Δ_R^s by mutual recursion over all ground terms s. Let $R^s = R \cup \bigcup_{u<s} \Delta_R^u$. Thus, the base of the recursion is $R^{\mathsf{T}} = R$ since T is the smallest term by (O3). For Δ_R^s, we distinguish the following cases:

(B1) Let $\Delta_R^s = \varnothing$ if s is not Boolean, if s is reducible by R^s, if $s = \mathsf{T}$, or if $s = \bot$

(B2) Otherwise, let $\Delta_R^s = \{s \longrightarrow \mathsf{T}\}$ if one of the following conditions holds:

(i) $s = \neg\bot$;

(ii) $s = \top \wedge \top$;
(iii) $s = \top \vee t$ or $s = t \vee \top$;
(iv) $s = \bot \to t$ or $s = t \to \top$;
(v) $s = t \approx t$;
(vi) $s = t \not\approx t'$ and $t \neq t'$;
(vii) $s = \forall x.\, t$ and $t\{x \mapsto u\} \to_{R^s}^* \top$ for all $u \in \mathcal{T}_G$ in which all Boolean subterms are either $\top$ or $\bot$;
(viii) $s = \exists x.\, t$ and $t\{x \mapsto u\} \to_{R^s}^* \top$ for some $u \in \mathcal{T}_G$ in which all Boolean subterms are either $\top$ or $\bot$.

(B3) Otherwise, let $\Delta_R^s = \{s \to \bot\}$.

Finally, we define $R^* = R \cup \bigcup_u \Delta_R^u$.

The following lemma identifies under which conditions on R the Boolean closure R^* fulfills the requirements of Definition 6.6 and can thus be viewed as an interpretation.

Lemma 6.10. *Let R be a term rewriting system on $\mathcal{T}_G$ without critical pairs and oriented by $\succ$. Assume for each rule $s' \to t' \in R$ that all proper Boolean subterms of s' are $\top$ and $\bot$ and that the head of s' is not a logical symbol. Let s be a Boolean term. Then*

(1) R^s and R^* are oriented by $\succ$ and hence terminating.
(2) R^s and R^* do not have critical pairs and are thus confluent.
(3) The normal form of any Boolean term smaller than s w.r.t. R^s is $\bot$ or $\top$.
(4) The normal form of any Boolean term w.r.t. R^* is $\bot$ or $\top$.
(5) R^* fulfills the requirements of Definition 6.6.

Proof. (1) is obvious from Definition 6.9 and (O3).

For (2), suppose there is a critical pair. Since R does not have critical pairs, one of the rules of the critical pair must come from some Δ_R^u for some term u. Due to condition (B1), Δ_R^u cannot form a critical pair with some Δ_R^v. Thus, the other rule of the critical pair must stem from R, say $s' \to t' \in R$. Also due to condition (B1), s' cannot be smaller or equal to u because that would make u reducible by R^u. But $s' \succ u$ is not possible either because then s' would contain a proper Boolean subterm that is neither $\top$ nor $\bot$ by condition (B1) and by (O2). Contradiction by totality.

(3) and (4) are obvious from Definition 6.9.

For (5): Condition (I1) follows from part (4). For (I2), since R does not contain any rules reducing terms headed by logical symbols, R cannot reduce $\neg\bot$, $\neg\top$, or similar terms. Therefore, (I2) follows directly from the definition of the Boolean closure.

For (I3), first assume that $s \leftrightarrow_{R^*}^* s'$. Let u be their common normal form. Then $s \approx s' \to_{R^*}^* u \approx u$. Since u cannot be reduced further, and R does not contain any rules reducing terms headed by logical symbols, the Boolean closure will add the rule $u \approx u \to \top$. If on the other hand, $s \not\to_{R^*}^* s'$, then $s \approx s' \to_{R^*}^* u \approx u'$ for distinct normal forms u, u'. Then the Boolean closure will add the rule $u \approx u' \to \bot$.

For (I4), we first need to prove the following claim. Let t be a term. Let θ, θ' be grounding substitutions such that for each variable x in t, we have $x\theta \leftrightarrow_{R^*}^* x\theta'$. Then we claim that $t\theta \leftrightarrow_{R^*}^* t\theta'$. Since we do not allow term rewriting systems to rewrite below quantifiers, this is not entirely trivial. We prove the claim by induction

on the number of nested levels of quantifiers in t. If t contains no quantifiers, we can rewrite freely and the claim is obvious. If t contains a quantifier-headed term, say $\forall x.\, s$, it suffices to show that $(\forall x.\, s)\theta \leftrightarrow^*_{R^*} (\forall x.\, s)\theta'$. Since R does not contain any rules reducing terms headed by logical symbols, R cannot reduce $(\forall x.\, s)\theta$ or $(\forall x.\, s)\theta'$. Hence, either $(\forall x.\, s)\theta \longrightarrow^*_{R^t} \bot$ by (B3) or $s\{x \mapsto u\}\theta \longrightarrow^*_{R^t} \top$ for all ground terms u in which all Boolean subterms are either $\top$ or $\bot$ and $(\forall x.\, s)\theta \longrightarrow^*_{R^t} \top$ by (B2). The same holds for θ'. Therefore, it suffices to show that $s\{x \mapsto u\}\theta \leftrightarrow^*_{R^*} s\{x \mapsto u\}\theta'$, which holds by the induction hypothesis. An analogous argument applies if the quantifier-headed term has the form $\exists x.\, s$.

By the above claim, using $\theta = \{x \mapsto u\}$, and by (I1), we have $s\{x \mapsto u\}$ for all u if and only if $s\{x \mapsto u\}$ for all u in which all Boolean subterms are either $\top$ or $\bot$. Hence, if $s\{x \mapsto u\}$ for all u, the Boolean closure will add the rule $\forall x.\, s \longrightarrow \top$, and otherwise it will add the rule $\forall x.\, s \longrightarrow \bot$, proving (I4). □

A further property of our Boolean closure operation is that Δ^s_R depends only on the small rewrite rules in R. We formalize this observation in the following lemma. We write $R|_{\prec s}$ or $R|_{\preceq s}$ for the rewriting system consisting of the rules in R with a left-hand side $\prec s$ or $\preceq s$, respectively.

Lemma 6.11. *Let t be a ground Boolean term. Let $R_1|_{\prec t} = R_2|_{\prec t}$ for two rewriting systems R_1 and R_2 on $\mathcal{T}_G$ oriented by $\succ$. Then, for all $s \prec t$, we have $\Delta^s_{R_1} = \Delta^s_{R_2}$.*

Proof. By induction on s. The induction hypothesis states that $\Delta^u_{R_1} = \Delta^u_{R_2}$ for all $u \prec s$. With the assumption $R_1|_{\prec t} = R_2|_{\prec t}$, it follows that $R^s_1|_{\prec t} = R^s_2|_{\prec t}$. Since $s \prec t$, in particular $R^s_1|_{\preceq s} = R^s_2|_{\preceq s}$. Inspecting the dependencies on R in the definition of Δ^s_R, we observe that Δ^s_R depends only on rules in $R^s|_{\preceq s}$. Hence, $\Delta^s_{R_1} = \Delta^s_{R_2}$. □

6.4.2. Construction of the Candidate Model

Inspired by Ganzinger and Stuber [62], we define the following term rewriting system, which forms the basis of our candidate model for a given clause set N.

Definition 6.12. Let N with $\bot \notin N$ be a set of ground clauses. Although the rewriting systems R in this definition do not necessarily fulfill the requirements of Definition 6.6, we write $R \models D$ if and only if normalizing D with R yields a clause with a trivial literal (i.e., $s \approx s$ or $s \not\approx t$ for terms $s \neq t$). We define R_C and Δ_C by mutual recursion over all clauses $C \in N$, ordered by $\succ$. Let $R_C = \bigcup_{D \prec C} \Delta_D$. Let $\Delta_C = \{s \longrightarrow t\}$ if

(C1) $s \succ t$;
(C2) $R^s_C \not\models C$;
(C3) $C = C' \vee s \approx t$ where $s \approx t$ is eligible in C;
(C4) the head of s is not a logical symbol;
(C5) $s \approx t$ is maximal in C;
(C6) $R^s_C \cup \{s \longrightarrow t\} \not\models C'$; and
(C7) s is irreducible by R^s_C.

Then C is said to *produce* $s \longrightarrow t$ or to be *productive*. Otherwise $\Delta_C = \emptyset$. Finally, we define $R_N = \bigcup_C \Delta_C$.

R_C and R_N fulfill the conditions of Lemma 6.10, meaning that their respective Boolean closure R_C^* and R_N^* can be viewed as an interpretation.

Lemma 6.13. *R_C and R_N fulfill the conditions of Lemma 6.10. That is, they do not have critical pairs and are oriented by $\succ$. For each of their rules $s \longrightarrow t$, all proper Boolean subterms of s are $\top$ or $\bot$ and the head of s is not a logical symbol.*

Proof. By (C1), all rules are oriented by $\succ$. Suppose there is a critical pair, and let C be the larger one of the two clauses producing the critical pair. Then R_C^s would be reducible by the other rule of the critical pair, contradicting (C7). By (C4), the head of the rules' left-hand sides cannot be a logical symbol.

Finally, we must show that all proper Boolean subterms of each left-hand side are $\top$ or $\bot$. We proceed by induction on the clause C producing the rule $s \longrightarrow t$. By the induction hypothesis, Lemma 6.10 can be applied to R_C, meaning that by Lemma 6.10(3), the normal form w.r.t. R_C^s of any Boolean term smaller than s is $\top$ or $\bot$. Thus, if s had a proper Boolean subterm other than $\top$ or $\bot$, it would be reducible by R_C^s, contradicting (C7). □

The Boolean closure R_N^* will be our candidate model. We can express it in terms of R_C^s as follows:

Lemma 6.14. *Let s be the maximal term of a clause $C \in N$. Then we have $R_N^* = R_C^s \cup \bigcup_{D \geq C} \Delta_D \cup \bigcup_{u \geq s} \Delta_{R_N}^u$.*

Proof.

$$
\begin{aligned}
R_N^* &= \bigcup_{C \in N} \Delta_C \cup \bigcup_u \Delta_{R_N}^u && \text{by definition of } R_N \text{ and } R^* \\
&= R_C \cup \bigcup_{u \prec s} \Delta_{R_C}^u \cup \bigcup_{D \geq C} \Delta_D \cup \bigcup_{u \geq s} \Delta_{R_N}^u && \text{by definition of } R_C \text{ and Lemma 6.11} \\
&= R_C^s \cup \bigcup_{D \geq C} \Delta_D \cup \bigcup_{u \geq s} \Delta_{R_N}^u && \text{by definition of } R_C^s
\end{aligned}
$$

□

The following lemma shows that rules produced by clauses greater than a clause C do not matter for the truth of C. We use the notation $R_N^*|_{\prec C}$ for $R_C \cup (R_N^* \setminus R_N)$. In other words, $R_N^*|_{\prec C}$ is R_N^* but without all rules produced by clauses greater than or equal to C.

Lemma 6.15. *Let C be a clause. If $R_N^*|_{\prec C} \models C$, then $R_N^* \models C$.*

Proof. We assume that $R_N^*|_{\prec C} \models C$. Then we have $R_N^*|_{\prec C} \models L$ for some literal L of C. It suffices to show that $R_N^* \models L$.

If $L = t \approx t'$ is a positive literal, then $t \downarrow_{R_N^*|_{\prec C}} = t' \downarrow_{R_N^*|_{\prec C}}$. Since $R_N^*|_{\prec C} \subseteq R_N^*$, this implies $t \downarrow_{R_N^*} = t' \downarrow_{R_N^*}$. Thus, $R_N^* \models L$.

If $L = t \not\approx t'$ is a negative literal, then $t \downarrow_{R_N^*|_{\prec C}} \neq t' \downarrow_{R_N^*|_{\prec C}}$. Without loss, let $t \succ t'$. Let $s \mathrel{\dot{\approx}} s'$ be the maximal term in C with $s \succeq s'$. We have $s \succ t$ if $s \mathrel{\dot{\approx}} s'$ is positive and $s \succeq t$ if $s \mathrel{\dot{\approx}} s'$ is negative. Hence, the left-hand sides of rules in $\bigcup_{D \geq C} \Delta_D$ are larger than t. Since only rules with a left-hand side $\preceq t$ can be involved in normalizing t and t' and $R_N^*|_{\prec C} \cup \bigcup_{D \geq C} \Delta_D = R_N^*$, it follows that $t \downarrow_{R_N^*} \neq t' \downarrow_{R_N^*}$ and hence $R_N^* \models L$. □

If the maximal literal of a clause C is positive, even the larger rules introduced by the Boolean closure do not matter for the truth of C:

Lemma 6.16. *Let $C = C' \vee s \approx s'$ be a clause where $s \approx s'$ is maximal and $s \succeq s'$. If $R_C^s \models C$, then $R_N^* \models C$. As in Definition 6.12, although R_C^s does not represent an interpretation, we write $R_C^s \models C$ to mean that normalizing C by R_C^s yields a trivial literal.*

Proof. We assume that $R_C^s \models C$. Then we have $R_C^s \models L$ for some literal L of C. It suffices to show that $R_N^* \models L$.

If $L = t \approx t'$ is a positive literal, then $t \downarrow_{R_C^s} = t' \downarrow_{R_C^s}$. Since $R_C^s \subseteq R_N^*$ by Lemma 6.14, this implies $t \downarrow_{R_N^*} = t' \downarrow_{R_N^*}$. Thus, $R_N^* \models L$.

If $L = t \not\approx t'$ is a negative literal, then $s \succ t$ and $s \succ t'$ and $t \downarrow_{R_C^s} \neq t' \downarrow_{R_C^s}$. By Lemma 6.14, $R_C^s|_{\prec s} = R_N^*|_{\prec s}$. Since only rules with a left-hand side smaller than s can be involved in normalizing t and t', it follows that $t \downarrow_{R_N^*} \neq t' \downarrow_{R_N^*}$ and hence $R_N^* \models L$ (using (O2), (O1)). □

Moreover, we will need the following basic properties of R_N^*:

Lemma 6.17. *If $C \vee t \approx s$ produces $t \longrightarrow s$, then $R_N^* \not\models C$.*

Proof. Let $D = C \vee t \approx s$. By (C1) and (C5), all terms in D are $\preceq t$. By (C6), we have $R_D^t \cup \{t \longrightarrow s\} \not\models C$. The other rules $R_N^* \setminus (R_D^t \cup \{t \longrightarrow s\})$ cannot reduce C because their left-hand sides are $\succ t$. The $\succeq t$ in that claim is by Lemma 6.14 and $\neq t$ by absence of critical pairs (Lemma 6.10(2)). Consequently, $R_D^t \cup \{t \longrightarrow s\} \not\models C$ implies $R_N^* \not\models C$. □

Lemma 6.18. $\top$ *and* $\bot$ *are normal forms in* R_N^*.

Proof. By condition (B1) and condition (C4), there is no rule that reduces $\top$ or $\bot$. □

Lemma 6.19. *Let C be a clause. Let t be Boolean subterm. Let*

- *t be smaller than the maximal term in C; or*
- *t be selected*

Then $R_N^*|_{\prec C}$ reduces t to $\bot$ or $\top$.

Proof. By Lemma 6.10(4), R_N^* reduces t to $\bot$ or $\top$. If this reduction does not contain any rule from a Δ_D with $D \succeq C$, then $R_N^*|_{\prec C}$ reduces t to $\bot$ or $\top$ as well. If this reduction does contain a rule from a Δ_D with $D \succeq C$, then by (C1) and (C5), t must be the maximal term of D and of C. The term t must occur on the top level of a positive literal in D and in C. Such terms may not be selected by the selection restrictions (Definition 6.3). Thus we have a contradiction to the condition that t is smaller than the maximal term in C and to the condition that t is selected. □

6.4.3. Reduction of Counterexamples

We employ Bachmair and Ganzinger's framework of reducing counterexamples [10, Section 4.2], with small modifications to their standard redundancy criterion. The interpretation R_N^* is our candidate model. A clause $C \in N$ is called a *counterexample* if $R_N^* \not\models C$. It is a *minimal* counterexample if C is the smallest clause in N w.r.t. $\succ$

such that $R_N^* \not\models C$. An inference *reduces* a counterexample C if its main premise is C, its side premises are true in R_N^*, and its conclusion D is a counterexample smaller than C. An inference system has the *reduction property for counterexamples* if for all clause sets N with a minimal counterexample C, there exists an inference from N that reduces C. Bachmair and Ganzinger's framework lets us derive static refutational completeness from this property.

The following lemma will guide us to choose a subterm on which we will find a reducing inference:

Lemma 6.20. *Let C be a clause. Let t be a term that is eligible in C and reducible by $R_N^*|_{\prec C}$. Then t has a subterm u such that:*

1. *The term u is eligible in C.*
2. *The term u is neither $\top$ nor $\bot$.*
3. *If the head of u is not a logical symbol, there exists a rule $u \longrightarrow u' \in R_N^*|_{\prec C}$.*
4. *If the head of u is $\neg$, $\wedge$, $\vee$, or $\rightarrow$, each proper subterm of u is $\top$ or $\bot$.*
5. *If the head of u is $\approx$ and $u\downarrow_{R_N^*|_{\prec C}} = \top$, or head of u is $\not\approx$ and $u\downarrow_{R_N^*|_{\prec C}} = \bot$, then the two sides of the (dis)equation are equal.*

Proof. We proceed by structural induction on t. First, we observe that whenever we can apply the induction hypothesis, we are done. Given an eligible, $R_N^*|_{\prec C}$-reducible proper subterm t' of t, the induction hypothesis guarantees the existence of a subterm u of t' with the above five properties. Since u is then also a subterm of t and the properties do not refer to t itself, such an application of the induction hypothesis finishes the proof.

We make a case distinction on the head h of t. First, assume that h is not a logical symbol. If an argument t' of h is reducible by $R_N^*|_{\prec C}$, we can apply the induction hypothesis and are done. On the other hand, if no argument of h reduces, then $R_N^*|_{\prec C}$ must reduce t by a rule $u \longrightarrow u'$ where $u = t$. Clearly, u satisfies the required five properties in this case.

Otherwise, h must a logical symbol. We have $t \neq \top, \bot$ because Lemma 6.18 tells us that $\top$ and $\bot$ are irreducible by $R_N^*|_{\prec C} \subseteq R_N^*$. The cases of connectives, quantifiers and (dis)equations remain.

We assume h is $\neg$, $\wedge$, $\vee$, or $\rightarrow$. We proceed as in the case where h was not a logical symbol. If an argument is reducible by $R_N^*|_{\prec C}$, the induction hypothesis applies. If none of the Boolean arguments reduce, then each of them is either $\top$ or $\bot$ by Lemma 6.19. This takes care of property 4 as we choose $u = t$. Since t is eligible and the head of $t = u$ is a connective, the other properties are fulfilled as well.

We assume h is $\forall$ or $\exists$. Choose $u = t$. The required properties are easy to check because t is eligible and the head of $t = u$ is $\forall$ or $\exists$.

Finally, we assume $h = \approx$. (The case $h = \not\approx$ is analogous.) If a strictly larger argument t' of h reduces by $R_N^*|_{\prec C}$, then t' is eligible and the induction hypothesis applies to t'. Otherwise we choose $u = t$. Property 5 holds because if $u\downarrow_{R_N^*|_{\prec C}} = \top$, then the two arguments of h must have the same $R_N^*|_{\prec C}$-normal form, since case (v) of Definition 6.9 is the only rule that reduces a term headed by $\approx$ to $\top$. Thus, the two arguments of h must be equal because otherwise the strictly larger one would be reducible by $R_N^*|_{\prec C}$. The other properties hold because t is eligible and the head of $t = u$ is $\approx$. □

The next lemma tells us under which conditions we can apply a reducing inference from the calculus rules that operate on subterms:

Lemma 6.21. *Assume $R_N^* \not\models C[s]_p \in N$, such that*

(a) *p is eligible in C,*
(b) *s reducible by $R_N^*|_{\prec C}$, and*
(c) *if p is a topmost position of a positive literal and the head of s is not a logical symbol, s must be reducible by R_C^s.*

Then our inference system reduces the counterexample C.

Proof. We apply Lemma 6.20 to s to find an appropriate subterm u of s.

CASE 1: We assume that the head of u is not a logical symbol. Then, by property 3 of Lemma 6.20, there exists a rule $u \longrightarrow u' \in R_N^*|_{\prec C}$ for some u'. We can apply BOOLHOIST or SUP to reduce the counterexample:

CASE 1.1: We assume that if $u \longrightarrow u' \in R_C$, then $u' = \bot$ and p is not a topmost position of a positive literal in C.

We check that BOOLHOIST is applicable:

1. The term u is of Boolean type and its head is not a logical symbol.
2. The position of u in $C[u]$ is eligible by property 1 of Lemma 6.20.
3. Finally, we need to show that the position of u is not a topmost position of a positive literal in C. If it was, then necessarily $s = u$ and by condition (c) of this lemma, it follows that $s = u$ is reducible by R_C^s. By the assumption of case 1.1 and the fact that $u \longrightarrow u' \in R_N^*|_{\prec C}$, we have $u \longrightarrow u' \in R_N^* \setminus R_N$. The rules in $R_C^s \setminus R_N$ have left-hand sides smaller than $s = u$. So $u \longrightarrow u' \in R_N^* \setminus R_N \setminus R_C^s$. Since $s = u$ is reducible by R_C^s, this contradicts the absence of critical pairs in R_N^*, which we have shown in Lemma 6.10(2).

CASE 1.2: We assume that case 1.1 does not apply. That means that $u \longrightarrow u' \in R_C$ and if $u' = \bot$, then p is a topmost position of a positive literal in C.

Then some clause $D \vee u \approx u' \in N$ smaller than C produces the rule $u \longrightarrow u'$. We claim that the counterexample C is reduced by the superposition inference

$$\frac{D \vee u \approx u' \quad C[u]}{D \vee C[u']}\,\text{SUP}$$

This superposition is a valid inference:

1. The position of u in $C[u]$ is eligible by property 1 of Lemma 6.20.
2. The literal $u \approx u'$ is strictly eligible by (C3) and (C6).
3. We have $u \succ u'$ by (C1).
4. The head of u is not a logical symbol by the assumption of case 1.
5. By construction of $D \vee u \approx u'$, we have $D \vee u \approx u' \prec C$.
6. If $u' = \bot$, then p is at the top level of a positive equation by the assumption of case 1.2.

As $D \vee u \approx u'$ is productive, $R_N^* \not\models D$ by Lemma 6.17. Hence the conclusion $D \vee C[u']$ is equivalent to $C[u']$, which is equivalent to $C[u]$ with respect to R_N^*. It remains to show that the new counterexample $D \vee C[u']$, which C is transformed into, is strictly smaller than C. The maximal literal in C is at least as large as $u \approx u'$ because

$D \vee u \approx u' \prec C$ and $D \prec u \approx u'$ because D is productive. Thus, the counterexample C reduces.

Case 2: We assume that the head of u is a logical symbol. By property 2 of Lemma 6.20, $u \neq \mathsf{T}, \bot$. By Lemma 6.10(4), R_N^* reduces u to T or to $\bot$.

Case 2.1: The head of u is $\neg$, $\wedge$, $\vee$, or $\rightarrow$; or the head of u is $\approx$ and it reduces to T; or the head of u is $\not\approx$ and it reduces to $\bot$. We apply BoolRw. Only two of its side conditions are relevant on ground clauses. Eligibility of u in C holds by property 1 of Lemma 6.20. For the other relevant side condition, we must pick the right item from the list. By properties 4 and 5 of Lemma 6.20, the list will contain an applicable item. Clearly, the conclusion is false in R_N^* and smaller than C.

Case 2.2: The head of u is $\forall$ and it reduces to T. We apply ForallRw. Clearly, the rule is applicable and the conclusion is false in R_N^* and smaller than C.

Case 2.3: The head of u is $\exists$ and it reduces to $\bot$. Analogous to the previous case, using ExistsRw.

Case 2.4: The head of u is $\approx$, say $u = s \approx t$, and u reduces to $\bot$. We apply EqHoist:

$$\frac{C[s \approx t]}{C[\bot] \vee s \approx t}\ \text{EqHoist}$$

Clearly, the inference conditions are fulfilled and the conclusion is smaller than C. The reduction $u \rightarrow_{R_N^*}^+ \bot$ necessarily has the form

$$s \approx t \ \rightarrow_{R_N^*}^* \ s' \approx t' \ \rightarrow_{\Delta_{R_N^*}^{s' \approx t'}} \ \bot$$

because the final step is the only way to reduce $\approx$. Here, s' and t' are different R_N^* normal forms. Hence $R_N^* \not\models s' \approx t'$ and $R_N^* \not\models s \approx t$ and $R_N^* \not\models C[\bot] \vee s \approx t$.

Case 2.5: The head of u is $\not\approx$ and it reduces to T. Analogous to the previous case, using NeqHoist.

Case 2.6: The head of u is $\forall$, say $u = \forall x.t$, and u reduces to $\bot$. We apply ForallHoist:

$$\frac{C[\forall x.t]}{C[\bot] \vee t\{x \mapsto s\} \approx \mathsf{T}}\ \text{ForallHoist}$$

Clearly, the inference conditions are fulfilled and the conclusion is smaller than C. The reduction $u \rightarrow_{R_N^*}^+ \bot$ necessarily has the form $\forall x.\, t \rightarrow \bot$ and originates from case (B3) of Definition 6.9 because rewriting under quantifiers is forbidden. In particular, case (vii) of Definition 6.9 does not apply. Hence there exists a ground term s whose Boolean subterms are only T and $\bot$ such that $t\{x \mapsto s\} \rightarrow_{R_N^*}^* \bot$. This is the s we choose for ForallHoist. Then the conclusion is a strictly smaller counterexample.

Case 2.7: The head of u is $\exists$ and it reduces to T. Analogous to the previous case, using ExistsHoist. □

Lemma 6.22. *Our ground inference system has the reduction property for counterexamples.*

Proof. Let N be a set of ground clauses that does not contain the empty clause. Let C be a minimal counterexample for R^*_N in N, i.e. the smallest clause in N that is false in R^*_N. We must show that there is an inference from N that reduces C, i.e., the inference has main premise C, side premises in N, and a conclusion that is a smaller counterexample for R^*_N than C.

1. We assume that C contains a selected Boolean subterm. Then it cannot be at a topmost position of a positive literal by the selection restrictions (Definition 6.3). By Lemma 6.19, $R^*_N|_{\prec C}$ reduces the selected subterm. Hence we can apply Lemma 6.21 to that subterm and are done.
2. We assume that there is an eligible literal of the form $s \not\approx s \in C$. Then ERES reduces C.
3. We assume that there is an eligible literal $s \not\approx s' \in C$ where $s \succ s'$. Since $R^*_N \not\models C$, we have $R^*_N|_{\prec C} \not\models C$ by Lemma 6.15. Therefore $R^*_N|_{\prec C} \not\models s \not\approx s'$ and $R^*_N|_{\prec C} \models s \approx s'$. Thus, s must be reducible by $R^*_N|_{\prec C}$ because $s \succ s'$. Therefore, we can apply Lemma 6.21 to s.
4. We assume that
   - no literals or Boolean subterms are selected in C;
   - there is a maximal literal $s \approx s' \in C$ and second literal $s \approx t \in C$ where $s \succ s', t$; and
   - $R^*_N \models s' \approx t$.

   Then we can apply
   $$\frac{C = C'' \vee s \approx t \vee s \approx s'}{C'' \vee s' \not\approx t \vee s \approx t}\ \text{EFACT}$$
   Since C is false in R^*_N, we have $R^*_N \models s \not\approx t$. Since moreover $R^*_N \models s' \approx t$, it follows that the conclusion of this inference must be false in R^*_N. Since $s \succ s', t$ and $s \approx s'$ is maximal, the above inference reduces C.
5. We assume that there is a selected literal $s \approx \bot \in C$ with $s \succ \bot$. Since C is false in R^*_N, C is also false in $R^*_N|_{\prec C}$ by Lemma 6.15. Hence $R^*_N|_{\prec C} \models s \approx \top$. If the head of s is not a logical symbol, since s reduces to $\top$, s must thus be reducible by R_C. Therefore, we can apply Lemma 6.21.
6. We assume that there is a strictly eligible literal $s \approx s'$ where s is reducible by R^s_C. Hence, s is eligible in C. Thus, we can again apply Lemma 6.21 to s.
7. We assume that there is a strictly eligible literal $s \approx s' \in C$ where $s \succ s'$ and the head of s is a logical symbol. By (O3), $s = \top$ contradicts $s \succ s'$. If $s = \bot$, then $s' = \top$ by (O3), and FALSEELIM reduces C. If $s \neq \top, \bot$ then s is reducible by Lemma 6.10(4). Thus, we can again apply Lemma 6.21 to s.

We will now show that one of the above cases applies or the clause C is productive, which would be a contradiction to $R^*_N \not\models C$.

First, we assume that a Boolean subterm or a literal is selected in C. If a Boolean subterm is selected, case 1 applies. If a literal is selected, it is either negative or of the form $s \approx \bot$ by the selection restrictions. If it is negative, case 2 or 3 applies. If it is of the form $s \approx \bot$, s cannot be $\bot$ because C is false in R^*_N. If $s = \top$, then case 7 applies. If $s \succ \bot$, then case 5 applies.

Now we may assume that C contains no selections. Then the maximal literal must be eligible. If the maximal literal is negative ((C3) does not hold), case 2 or 3

applies. If (C1) does not hold, the literal must be negative because C is true in R^*_N. If (C2) does not hold, then by Lemma 6.16, the maximal literal must be negative.

So we may assume that C contains no selections and the maximal literal is positive. If (C6) does not hold, then $R^s_C \cup \{s \longrightarrow s'\} \models C'$ where C' is the subclause of C with the maximal literal removed. However, $R^s_C \not\models C$ by Lemma 6.16 and by the assumption $R^*_N \not\models C$. Therefore, $R^s_C \not\models C'$. Thus, we must have $C' = C'' \vee r \approx t$ for some terms r and t where $R^s_C \cup \{s \longrightarrow s'\} \models r \approx t$ and $R^s_C \not\models r \approx t$. So $r \neq t$ and without loss we assume $r \succ t$. Moreover $s \longrightarrow s'$ must participate in the normalization of r or t by $R^s_C \cup \{s \longrightarrow s'\}$. Since $r \preceq s \succ s'$, the rule $s \longrightarrow s'$ can only be used as the first step in the normalization of r. Hence $r = s$ and $R^s_C \models s' \approx t$. Then case 4 applies. In particular, it applies if the maximal literal is not strictly maximal.

Now we may assume that C contains no selections and the maximal literal is strictly maximal and positive. If (C7) does not apply, then case 6 applies. If (C4) does not apply, then case 7 applies. □

Bachmair and Ganzinger [10, Section 4.2] have shown that the reduction property for counterexamples implies static refutational completeness. We deviate slightly from their framework and define the set of redundant clauses $Red_C(N)$ and the set of redundant inferences $Red_I(N)$ w.r.t. a clause set N as we have in previous chapters.

6

Definition 6.23 (Redundancy)**.** Given a ground clause C and a set N of ground clauses, let $C \in Red_C(N)$ if $\{D \in N \mid D \prec C\} \models C$. Given an inference ι and a set N of ground clauses, let $\iota \in Red_I(N)$ if $prems(\iota) \cap Red_C(N) \neq \varnothing$ or $\{D \in N \mid D \prec mprem(\iota)\} \models concl(\iota)$. A clause set N is saturated if all inferences from N are in $Red_I(N)$.

Adapting the proof of Theorem 4.9 of Bachmair and Ganzinger [10] to match our redundancy criterion, we can show refutational completeness of our calculus as follows:

Theorem 6.24 (Ground static refutational completeness)**.** *Let N be a set of ground clauses saturated w.r.t. our calculus and our redundancy criterion and let $\bot \notin N$. Then $R^*_{N \setminus Red_C(N)}$ is a model of N.*

Proof. By Lemma 6.22, our inference system fulfills the reduction property of counterexamples w.r.t. $\succ$. This means that for any clause set M where C is the smallest clause in M that is false in R^*_M, there exists an inference from M with

- main premise C,
- side premises that are true in R^*_M, and
- a conclusion that is smaller than C and false in R^*_M.

To derive a contradiction, we assume that $R^*_{N \setminus Red_C(N)} \not\models N$. Then there must be a smallest clause in $C \in N$ that is false in $R^*_{N \setminus Red_C(N)}$. Using $M = N \setminus Red_C(N)$, we obtain an inference ι with the above properties. Since N is saturated and $prems(\iota) \subseteq M \subseteq N$, we have $\iota \in Red_I{}^q(N)$. By definition, this means $prems(\iota) \cap GFRed_C(N) \neq \varnothing$ or $\{D \in N \mid D \prec C\} \models concl(\iota)$. Since $prems(\iota) \subseteq M = N \setminus GFRed_C(N)$, it must be the latter. Then $R^*_M \not\models \{D \in N \mid D \prec C\}$ because $R^*_M \not\models concl(\iota)$. This contradicts the minimality of C. □

6.5. Conclusion

We presented a ground calculus for first-order logic with interpreted Booleans and proved it to be refutationally complete. The approach builds on the ideas of Ganzinger and Stuber [62] on the one hand and of Kotelnikov et al. [92,93] on the other hand. In contrast to Ganzinger and Stuber's calculus, our calculus supports an actual Boolean type, breaking the strict separation between terms and formulas. In contrast to Kotelnikov et al.'s approach, our approach integrates Booleans directly into the calculus instead of preprocessing them. These qualities of our calculus will allow us to extend it to higher-order logic in the following chapter.

7

Superposition for Full Higher-Order Logic

Joint work with
Jasmin Blanchette, Sophie Tourret, and Petar Vukmirović

We present a sound and refutationally complete calculus for full higher-order logic. It is based on both the Boolean-free λ-superposition calculus developed in Chapter 5 and the calculus for first-order logic with interpreted Booleans developed in Chapter 6. Merging the two approaches yields a calculus operating on $\beta\eta$-equivalence classes of λ-terms that supports delayed clausification. We have implemented the calculus in the Zipperposition prover, and our evaluation shows that it outperforms other modern higher-order provers.

My contributions to this chapter are the design of the core calculus, the $\mathsf{Q}_{\approx}$-preprocessing mechanism, the concrete term order, the redundancy criterion, and the ground first-order and nonground higher-order level of the completeness proof.

7.1. Introduction

Having constructed superposition calculi for various logics between first-order and higher-order logic, we have finally collected all the parts necessary to assemble a calculus for full higher-order logic. Besides the issues discussed in previous chapters and the sheer complexity of combining the different approaches, we encountered the following four main challenges.

First, Boolean subterms cannot always be hoisted to the clausal level. In first-order logic with an interpreted Boolean type as introduced by Kotelnikov et al. [93], a clause $C[t]$ with a subterm t of Boolean type is equivalent to the clauses $C[\top] \vee t \approx \bot$ and $C[\bot] \vee t \approx \top$. Hoisting all Boolean subterms in this way yields clauses whose only Boolean subterms below the top level are $\top$ and $\bot$. In higher-order logic, however, Boolean subterms can contain variables bound by λ-expressions—e.g., $\mathsf{f}(\lambda x.\, x \mathbin{\approx\!\!\!\approx} \mathsf{a}) \approx \mathsf{b}$. These Boolean subterms cannot be hoisted as described above.

Our solution is to perform *inprocessing* clausification (i.e. clausification *during* the derivation), similarly to Leo-III [126], instead of *preprocessing* clausification (i.e. clausification *before* the derivation). To justify the refutational completeness of this approach, we follow the structure of the proof in Chapter 5, but swap out the GF level to use the logic and completeness result of Chapter 6. In addition, this allows our redundancy criterion to incorporate Boolean reasoning and thus our simplification machinery can simplify Boolean terms in various ways.

Second, when grounding applied variables, Boolean subterms containing the applied variable's arguments may appear. For example, consider the clause $\mathsf{h}(y\,\mathsf{b}) \not\approx \mathsf{h}(\mathsf{g}\,\bot) \vee \mathsf{h}(y\,\mathsf{a}) \not\approx \mathsf{h}(\mathsf{g}\,\top)$. When grounding the clause with the substitution $\{y \mapsto \lambda x.\, \mathsf{g}(x \mathbin{\approx\!\!\!\approx} \mathsf{a})\}$, the Boolean subterms $\mathsf{b} \mathbin{\approx\!\!\!\approx} \mathsf{a}$ and $\mathsf{a} \mathbin{\approx\!\!\!\approx} \mathsf{a}$ appear. Because they contain the argument of y, simply ignoring them leads to incompleteness.

We solve this issue with a dedicated *fluid Boolean subterm hoisting rule* (FluidBoolHoist) and a closely related rule FluidLoobHoist. These rules resemble the rule FluidSup from Chapter 5, but introduce Boolean subterms and hoist them to the clausal level instead of introducing arbitrary subterms and superposing into them.

Third, like other calculi for higher-order logic, we must perform a form of primitive substitution [2, 23, 73, 126]. For example, given the clauses $C = \mathsf{a} \not\approx \mathsf{b}$ and $D = z\,\mathsf{a} \approx \bot \vee z\,\mathsf{b} \approx \top$, it is crucial to find the substitution $\{z \mapsto \lambda v.\, v \mathbin{\approx\!\!\!\approx} \mathsf{a}\}$, which does not arise through unification. Primitive substitution blindly substitutes logical connectives and quantifiers as a remedy—e.g., it applies $\rho = \{z \mapsto \lambda v.\, y\,v \mathbin{\approx\!\!\!\approx} y'\,v\}$ where y and y' are fresh variables. In combination with the superposition calculus, this is problematic because the clause $D\rho$ is subsumed by D and could immediately be discarded by our simplification machinery. Our solution is to directly clausify the introduced logical symbol, yielding a clause that is no longer subsumed.

Fourth, our core calculus cannot handle variables bound by unclausified quantifiers if these variables occur applied, within λ-expressions, or in arguments of applied variables. We solve the issue by replacing such quantified term $\forall y.\, t$ by $(\lambda y.\, t) \mathbin{\approx\!\!\!\approx} (\lambda y.\, \top)$ in a preprocessing step.

We have implemented the calculus in Zipperposition and evaluated it on TPTP and Sledgehammer benchmarks. The new Zipperposition outperforms all other

higher-order provers and is on a par with an ad hoc implementation of Booleans in the same prover by Vukmirović and Nummelin [138].

7.2. Logic

Our logic is higher-order logic with rank-1 polymorphism, Hilbert choice, and functional and Boolean extensionality. To be able to prove refutational completeness, we use Henkin semantics [21, 59, 71].

Thus, our logic is essentially the one of the TPTP TH1 standard [79], but we focus on a smaller yet comprehensive subset of connectives and binders, and—as in the Isabelle proof assistant—our type variables are always implicitly bound universally at the top level. To adhere to the restriction that we do not support explicit type variable binders, these binders can easily be eliminated by type skolemization [32, Section 5.2].

On top of the standard higher-order terms, we employ a clausal structure, forcing us to express the input problem as a conjunction of disjunctions of equations and disequations. This allows for a formulation of the calculus rules that closely resembles the first-order superposition calculus. It does not restrict the flexibility of the logic because an arbitrary term t of Boolean type can be written as the clause $t \approx \mathsf{T}$.

Syntax As a basis for our logic's types, we fix an infinite set $\mathcal{V}_{\mathsf{ty}}$ of type variables. A set Σ_{ty} of type constructors with associated arities is a *type signature* if it contains at least one nullary Boolean type constructor o and a binary function type constructor $\to$. A *type*, usually denoted by τ or υ, is inductively defined to either be a type variable $\alpha \in \mathcal{V}_{\mathsf{ty}}$ or have the form $\kappa(\bar{\tau}_n)$ for an n-ary type constructor $\kappa \in \Sigma_{\mathsf{ty}}$ and types $\bar{\tau}_n$. We write κ for $\kappa()$ and $\tau \to \upsilon$ for $\to(\tau, \upsilon)$. A *type declaration* is an expression $\Pi\bar{\alpha}_m.\,\tau$ (or simply τ if $m = 0$), where all type variables occurring in τ belong to $\bar{\alpha}_m$.

To define our logic's terms, we fix a type signature Σ_{ty} and a set $\mathcal{V}$ of term variables with associated types, written as $x : \tau$ or x. We require that there are infinitely many variables for each type.

A *term signature* is a set Σ of (function) symbols, usually denoted by a, b, c, f, g, h, each associated with a type declaration, written as $\mathsf{f} : \Pi\bar{\alpha}_m.\,\tau$. We require the presence of the logical symbols $\mathsf{T}, \mathsf{\bot} : o$; $\neg : o \to o$; $\wedge, \vee, \to : o \to o \to o$; $\forall, \exists : \Pi\alpha.\,(\alpha \to o) \to o$; and $\approx, \not\approx : \Pi\alpha.\,\alpha \to \alpha \to o$. Moreover, we require the presence of the Hilbert choice operator $\varepsilon : \Pi\alpha.\,(\alpha \to o) \to \alpha$. Although ε is interpreted in our semantics, we do not consider it to be a logical symbol. The reason is that our calculus will enforce the semantics of ε by an axiom, whereas the semantics of the logical symbols will be enforced by inference rules. In the following, we also fix a term signature Σ. A type signature and a term signature form a *signature*.

As in Chapter 5, we will define terms in three layers of abstraction: raw λ-terms, λ-terms, and terms; where λ-terms will be α-equivalence classes of raw λ-terms and terms will be $\beta\eta$-equivalence classes of λ-terms.

The set of *raw λ-terms* and their associated types is defined exactly as in Section 5.2. For the symbols $\approx$ and $\not\approx$, we will typically use infix notation and omit the type argument. For the definitions of subterms, free variable occurrences, and ground terms, we also refer to Section 5.2.

The α-renaming rule is defined as $(\lambda x.\, t) \longrightarrow_\alpha (\lambda y.\, t\{x \mapsto y\})$, where y does not occur free in t and is not captured by a λ-binder in t. Raw λ-terms form equivalence classes modulo α-renaming, called *λ-terms*. We lift the above notions on raw λ-terms to λ-terms. We define substitutions as in Section 5.2.

The β- and η-reduction rules are specified on λ-terms as $(\lambda x.\, t)u \longrightarrow_\beta t\{x \mapsto u\}$ and $(\lambda x.\, t\, x) \longrightarrow_\eta t$. For β, bound variables in t are implicitly renamed to avoid capture; for η, the variable x may not occur free in t. The λ-terms form equivalence classes modulo $\beta\eta$-reduction, called *$\beta\eta$-equivalence classes* or simply *terms*.

Deviating from Chapter 5, we do not use the η-short β-normal form, but the following related nonstandard normal form. The *$\beta\eta\mathsf{Q}_\eta$-normal* form $t{\downarrow_{\beta\eta\mathsf{Q}_\eta}}$ of a λ-term t is obtained by applying $\longrightarrow_\beta$ and $\longrightarrow_\eta$ exhaustively and finally applying the following rewrite rule Q_η exhaustively:

$$\mathsf{Q}\langle\tau\rangle t \longrightarrow_{\mathsf{Q}_\eta} \mathsf{Q}\langle\tau\rangle(\lambda x.\, t\, x)$$

where t is not a λ-expression. Here and elsewhere, Q stands for either $\forall$ or $\exists$.

We lift all of the notions defined on λ-terms to terms:

Convention 7.1. When defining operations that need to analyze the structure of terms, we use the $\beta\eta\mathsf{Q}_\eta$-normal representative as the default representative of a $\beta\eta$-equivalence class.

A literal is an equation $s \approx t$ or a disequation $s \not\approx t$ of terms s and t. In both cases, the order of s and t is not fixed. We write $s \mathrel{\dot{\approx}} t$ for a literal that can be either an equation or a disequation. A clause $L_1 \vee \cdots \vee L_n$ is a finite multiset of literals L_j. The empty clause is written as $\bot$. As in Chapter 6, nonequational literals must be encoded as $t \approx \mathsf{T}$ or $t \approx \mathsf{\bot}$.

As in the previous chapter, our calculus does not allow nonequational literals. These must be encoded as $t \approx \mathsf{T}$ or $t \approx \mathsf{\bot}$. Both of these are considered positive literals because they are equations, not disequations. Our simplification mechanism will allow us to simplify negative literals of the form $t \not\approx \mathsf{\bot}$ and $t \not\approx \mathsf{T}$ into $t \approx \mathsf{T}$ and $t \approx \mathsf{\bot}$, respectively.

For the definition of a complete set of unifiers $\mathrm{CSU}(s, t)$ of two terms s and t, we refer to Section 5.2.

Semantics A *type interpretation* $\mathcal{I}_{\mathsf{ty}} = (\mathcal{U}, \mathcal{J}_{\mathsf{ty}})$ is defined as follows. The *universe* $\mathcal{U}$ is a collection of nonempty sets, called *domains*. We require that $\{0,1\} \in \mathcal{U}$. The function $\mathcal{J}_{\mathsf{ty}}$ associates a function $\mathcal{J}_{\mathsf{ty}}(\kappa) : \mathcal{U}^n \to \mathcal{U}$ with each n-ary type constructor κ, such that $\mathcal{J}_{\mathsf{ty}}(o) = \{0,1\}$ and for all domains $\mathcal{D}_1, \mathcal{D}_2 \in \mathcal{U}$, the set $\mathcal{J}_{\mathsf{ty}}(\to)(\mathcal{D}_1, \mathcal{D}_2)$ is a subset of the function space from $\mathcal{D}_1$ to $\mathcal{D}_2$. The semantics is *standard* if $\mathcal{J}_{\mathsf{ty}}(\to)(\mathcal{D}_1, \mathcal{D}_2)$ is the entire function space for all $\mathcal{D}_1, \mathcal{D}_2$. A *type valuation* ξ is a function that maps every type variable to a domain. The *denotation* of a type for a type interpretation $\mathcal{I}_{\mathsf{ty}}$ and a type valuation ξ is recursively defined by $[\![\alpha]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}} = \xi(\alpha)$ and $[\![\kappa(\bar\tau)]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}} = \mathcal{J}_{\mathsf{ty}}(\kappa)([\![\bar\tau]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}})$.

A type valuation ξ can be extended to be a *valuation* by additionally assigning an element $\xi(x) \in [\![\tau]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}}$ to each variable $x : \tau$. An *interpretation function* $\mathcal{J}$ for a type interpretation $\mathcal{I}_{\mathsf{ty}}$ associates with each symbol $\mathsf{f} : \Pi\bar\alpha_m.\, \tau$ and domain tuple $\bar{\mathcal{D}}_m \in \mathcal{U}^m$

a value $\mathcal{J}(\mathsf{f}, \bar{\mathcal{D}}_m) \in [\![\tau]\!]^{\xi}_{\mathcal{J}_{\mathsf{ty}}}$, where ξ is the type valuation that maps each α_i to $\mathcal{D}_i$. We require that

$$\mathcal{J}(\mathsf{T}) = 1 \qquad \mathcal{J}(\bot) = 0 \qquad \mathcal{J}(\wedge)(a,b) = \min\{a,b\} \qquad \mathcal{J}(\vee)(a,b) = \max\{a,b\}$$
$$\mathcal{J}(\neg)(a) = 1 - a \qquad \mathcal{J}(\rightarrow)(a,b) = \max\{1-a,b\}$$
$$\mathcal{J}(\approx, \mathcal{D})(c,d) = 1 \text{ if } c = d \text{ and } 0 \text{ otherwise}$$
$$\mathcal{J}(\not\approx, \mathcal{D})(c,d) = 0 \text{ if } c = d \text{ and } 1 \text{ otherwise}$$
$$\mathcal{J}(\forall, \mathcal{D})(f) = \min\{f(a) \mid a \in \mathcal{D}\} \qquad \mathcal{J}(\exists, \mathcal{D})(f) = \max\{f(a) \mid a \in \mathcal{D}\}$$
$$f(\mathcal{J}(\varepsilon, \mathcal{D})(f)) = \max\{f(a) \mid a \in \mathcal{D}\}$$

for all $a, b \in \{0,1\}$, $\mathcal{D} \in \mathcal{U}$, $c, d \in \mathcal{D}$, and $f \in \mathcal{J}_{\mathsf{ty}}(\rightarrow)(\mathcal{D}, \{0,1\})$.

As in Chapter 5, loosely following Fitting [59, Section 2.4], we initially allow λ-expressions to designate arbitrary elements of the domain and impose restrictions afterwards using the notion of a proper interpretation.

A *λ-designation function* $\mathcal{L}$ for a type interpretation $\mathcal{I}_{\mathsf{ty}}$ is a function that maps a valuation ξ and a λ-expression of type τ to elements of $[\![\tau]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}$. A type interpretation, an interpretation function, and a λ-designation function form an *interpretation* $\mathcal{I} = (\mathcal{I}_{\mathsf{ty}}, \mathcal{J}, \mathcal{L})$.

For an interpretation $\mathcal{I}$ and a valuation ξ, the denotation of a term is defined as $[\![x]\!]^{\xi}_{\mathcal{I}} = \xi(x)$, $[\![\mathsf{f}\langle \bar{\tau}_m \rangle]\!]^{\xi}_{\mathcal{I}} = \mathcal{J}(\mathsf{f}, [\![\bar{\tau}_m]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}})$, $[\![s\,t]\!]^{\xi}_{\mathcal{I}} = [\![s]\!]^{\xi}_{\mathcal{I}}([\![t]\!]^{\xi}_{\mathcal{I}})$, and $[\![\lambda x.\, t]\!]^{\xi}_{\mathcal{I}} = \mathcal{L}(\xi, \lambda x.\, t)$. For ground terms t, the denotation does not depend on the choice of the valuation ξ, which is why we sometimes write $[\![t]\!]_{\mathcal{I}}$ for $[\![t]\!]^{\xi}_{\mathcal{I}}$.

An interpretation $\mathcal{I}$ is *proper* if $[\![\lambda x.\, t]\!]^{\xi}_{\mathcal{I}}(a) = [\![t]\!]^{\xi[x \mapsto a]}_{\mathcal{I}}$ for all λ-expressions $\lambda x.\, t$ and all valuations ξ. If a type interpretation $\mathcal{I}_{\mathsf{ty}}$ and a interpretation function $\mathcal{J}$ can be extended by a λ-designation function $\mathcal{L}$ to a proper interpretation $(\mathcal{I}_{\mathsf{ty}}, \mathcal{J}, \mathcal{L})$, then this $\mathcal{L}$ is unique [59, Proposition 2.18]. Truth of clauses under an interpretation and models are defined as in Section 5.2.

Skolem-Aware Interpretations Some of the rules in our calculus introduce Skolem symbols—i.e., symbols representing objects mandated by existential quantification. We define a Skolem-extended signature that contains all Skolem symbols that could possibly be needed by the calculus rules.

Definition 7.2. Given a term signature Σ, let the *Skolem-extended term signature* Σ_{sk} the smallest signature that contains all symbols from Σ and a symbol $\mathsf{sk}_{\Pi\bar{\alpha}.\forall\bar{x}.\exists z.\, t\, z} : \Pi\bar{\alpha}.\, \bar{\tau} \to \upsilon$ for all types υ, variables $z : \upsilon$, terms $t : \upsilon \to o$ over the signature $(\Sigma_{\mathsf{ty}}, \Sigma_{\mathsf{sk}})$, where $\bar{\alpha}$ are the free type variables occurring in t and $\bar{x} : \bar{\tau}$ are the free term variables occurring in t in order of first appearance.

Interpretations as defined above can interpret the Skolem symbols arbitrarily. For example, an interpretation $\mathcal{I}$ does not necessarily interpret the symbol $\mathsf{sk}_{\exists z.\, z \approx \mathsf{a}}$ as $[\![\mathsf{a}]\!]_{\mathcal{I}}$. Therefore, an inference producing $(\mathsf{sk}_{\exists z.\, z \approx \mathsf{a}} \approx \mathsf{a}) \approx \mathsf{T}$ from $\exists\langle \iota \rangle(\lambda z.\, z \approx \mathsf{a}) \approx \mathsf{T}$ is unsound w.r.t. $\models$. As a remedy, we define Skolem-aware interpretations as follows:

Definition 7.3. We call a proper interpretation over a Skolem-extended signature *Skolem-aware* if for all Skolem symbols $\mathcal{I} \models (\exists\langle \upsilon \rangle(\lambda z.\, t\, z)) \approx t(\mathsf{sk}_{\Pi\bar{\alpha}.\forall\bar{x}.\exists z.\, t\, z}\langle\bar{\alpha}\rangle\bar{x})$, where

$\bar{\alpha}$ are the free type variables and $\bar{x}$ are the free term variables occurring in t in order of first appearance. An interpretation is a *Skolem-aware model* of a clause set N, written $\mathcal{I} \models\!\!\approx N$, if $\mathcal{I}$ is Skolem-aware and $\mathcal{I} \models N$.

7.3. The Calculus

Our *λ-superposition calculus* presented here combines ideas from Chapters 3, 5, and 6. As in Chapters 3 and 5, we restrict superposition inferences to green subterms. We add several rules for the treatment of Booleans, inspired by Chapter 6. As in Chapter 5, we need a FLUIDSUP rule to simulate superposition inferences inside instantiated fluid terms. In addition, we require two new rules, FLUIDBOOLHOIST and FLUIDLOOBHOIST, that simulate inferences on Boolean terms inside instantiated fluid terms. We refine the definition of green subterms to exclude subterms below quantifiers:

Definition 7.4 (Green subterms and positions)**.** Green subterms and positions are inductively defined as follows. A green position is a tuple of natural numbers. For any λ-term t, the empty tuple ε is a green position of t, and t is the green subterm of t at position ε. For all symbols $\mathsf{f} \in \Sigma \setminus \{\forall, \exists\}$, if t is a green subterm of u_i at position p, then $i.p$ is a green position of $\mathsf{f}\langle\bar{\tau}\rangle\bar{u}$, and t is the green subterm of $\mathsf{f}\langle\bar{\tau}\rangle\bar{u}$ at position $i.p$.

For positions in clauses, natural numbers are not appropriate because clauses and literals are unordered. A green position in a clause C is a tuple $L.s.p$ where $L = s \mathrel{\dot{\approx}} t$ is a literal in C and p is a green position in s. The green subterm of C at position $L.s.p$ is the green subterm of s at position p.

We write $s|_p$ to denote the green subterm at position p in s. We write $s\langle u\rangle_p$ to denote a λ-term s with the green subterm u at position p and call $s\langle\rangle_p$ a *green context*; the position p may be omitted in this notation. A position p is *at or below* a position q if q is a prefix of p. A position p is *below* a position q if q is a proper prefix of p.

The notions of green positions, subterms, and context are lifted to $\beta\eta$-equivalence classes via the $\beta\eta\mathsf{Q}_\eta$-normal representative.

For example, the green subterms of $\mathsf{f}(\mathsf{g}(\neg\mathsf{p}))(\forall\langle\tau\rangle(\lambda x.\mathsf{q}))(y\,\mathsf{a})(\lambda x.\mathsf{h}\,\mathsf{b})$ are the term itself, $\mathsf{g}(\neg\mathsf{p})$, $\neg\mathsf{p}$, p, $\forall\langle\tau\rangle(\lambda x.\mathsf{q})$, $y\,\mathsf{a}$, and $\lambda x.\mathsf{h}\,\mathsf{b}$.

7.3.1. Preprocessing

Some combinations of applied variables with quantifiers could lead to incompleteness of our calculus. This is why we use preprocessing to eliminate problematic occurrences of quantifiers.

Definition 7.5. The rewrite rules $\forall_{\approx}$ and $\exists_{\approx}$, which we collectively denote $\mathsf{Q}_{\approx}$, are defined on λ-terms as

$$\forall\langle\tau\rangle \longrightarrow_{\forall_{\approx}} \lambda y.\, y \approx (\lambda x.\, \top) \qquad\qquad \exists\langle\tau\rangle \longrightarrow_{\exists_{\approx}} \lambda y.\, y \not\approx (\lambda x.\, \bot)$$

where the rewritten occurrence of $\mathsf{Q}\langle\tau\rangle$ is unapplied, has an argument that is not a λ-expression, or has an argument of the form $\lambda x.\, v$ such that x occurs in a nongreen position of v.

If neither of these rewrite rules can be applied to a given λ-term, the λ-term is $\mathsf{Q}_{\approx}$-normal; otherwise it is $\mathsf{Q}_{\approx}$-reducible. We lift this notion to $\beta\eta$-equivalence classes via the $\beta\eta\mathsf{Q}_\eta$-normal representative. A clause or clause set is $\mathsf{Q}_{\approx}$-normal if all contained terms are $\mathsf{Q}_{\approx}$-normal.

For example, the term $\lambda y.\, \exists\langle\iota\to\iota\rangle(\lambda x.\, \mathsf{g}\, x\, y\, (z\, y)\, (\mathsf{f}\, x))$ is $\mathsf{Q}_{\approx}$-normal. Reasons for a term to be $\mathsf{Q}_{\approx}$-reducible are: A quantifier appears unapplied, e.g., $\mathsf{g}\,\exists\langle\iota\rangle$; A quantifier-bound variable occurs applied, e.g., $\exists\langle\iota\to\iota\rangle(\lambda x.\, x\,\mathsf{a})$; A quantifier-bound variable occurs inside a nested λ-expression, e.g., $\forall\langle\iota\rangle(\lambda x.\mathsf{f}(\lambda y.x))$; A quantifier-bound variable occurs in the argument of a variable, either a free variable, e.g., $\forall\langle\iota\rangle(\lambda x.\, z\, x)$, or a variable bound above the quantifier, e.g., $\lambda y.\, \exists\langle\iota\rangle(\lambda x.\, y\, x)$.

In principle, we could remove the condition of the $\mathsf{Q}_{\approx}$ rewrite rules and thus eliminate all quantifiers. However, our calculus and in particular our redundancy criterion can deal better with quantifiers than with λ-expressions, which is why we restrict $\mathsf{Q}_{\approx}$-normalization as much as possible without compromising refutational completeness.

We can also characterize $\mathsf{Q}_{\approx}$-normality as follows:

Lemma 7.6. *Let t be a term with spine notation $t = s\,\bar{u}_n$. Then t is $\mathsf{Q}_{\approx}$-normal if and only if $\bar{u}_n$ are $\mathsf{Q}_{\approx}$-normal and*

(i) s is of the form $\mathsf{Q}\langle\tau\rangle$, $n = 1$, and u_1 is of the form $\lambda y.\, u'$ such that y occurs only in green positions of u'; or
(ii) s is a λ-expression whose body is $\mathsf{Q}_{\approx}$-normal; or
(iii) s is neither of the form $\mathsf{Q}\langle\tau\rangle$ nor a λ-expression.

Proof. This follows directly from Definition 7.5. □

In the following lemmas, our goal is to show that $\mathsf{Q}_{\approx}$-normality is invariant under $\beta\eta\mathsf{Q}_\eta$-normalization—i.e., if a λ-term t is $\mathsf{Q}_{\approx}$-normal, then so is $t{\downarrow_{\beta\eta\mathsf{Q}_\eta}}$. However, $\mathsf{Q}_{\approx}$-normality is not invariant under arbitrary $\beta\eta$-conversions. Clearly, a β-expansion can easily introduce $\mathsf{Q}_{\approx}$-reducible terms, e.g., $\mathsf{c} \longleftarrow_\beta (\lambda x.\, \mathsf{c})(\forall\langle\iota\rangle)$.

Lemma 7.7. *If t and v are $\mathsf{Q}_{\approx}$-normal λ-terms, then $t\,v$ is a $\mathsf{Q}_{\approx}$-normal λ-term.*

Proof. We prove this by induction on the structure of t. Let $s\,\bar{u}_n = t$ be the spine notation of t. By Lemma 7.6, $\bar{u}_n$ are $\mathsf{Q}_{\approx}$-normal and one of the lemma's three cases applies. Since t is of functional type and $\mathsf{Q}_{\approx}$-normal, s cannot be of the form $\mathsf{Q}\langle\tau\rangle$, excluding case (i). Cases (ii) and (iii) are independent of $\bar{u}_n$, and hence appending v to that tuple does not affect the $\mathsf{Q}_{\approx}$-normality of t. □

Lemma 7.8. *If t is a $\mathsf{Q}_{\approx}$-normal λ-term and ρ is a substitution such that $x\rho$ is $\mathsf{Q}_{\approx}$-normal for all x, then $t\rho$ is $\mathsf{Q}_{\approx}$-normal.*

Proof. We prove this by induction on the structure of t. Let $s\,\bar{u}_n = t$ be its spine notation. Since t is $\mathsf{Q}_{\approx}$-normal, by Lemma 7.6, u_n are $\mathsf{Q}_{\approx}$-normal and one of the following cases applies:

Case (i): s is of the form $\mathsf{Q}\langle\tau\rangle$, $n = 1$, and u_1 is of the form $\lambda y.\, u'$ such that y occurs only in green positions of u'. Since our substitutions avoid capture, $y\rho = y$ and y does not appear in $x\rho$ for all other variables x. It is clear from the definition of green positions that since y occurs only in green positions of u', y also occurs only in green positions of $u'\rho$. Moreover, by the induction hypothesis, $u_1\rho$ is $\mathsf{Q}_{\approx}$-normal. Hence, $t\rho = \mathsf{Q}\langle\tau\rho\rangle(u_1\rho) = \mathsf{Q}\langle\tau\rho\rangle(\lambda y.\, u'\rho)$ is also $\mathsf{Q}_{\approx}$-normal.

Case (ii): s is a λ-expression whose body is $\mathsf{Q}_{\approx}$-normal. Then $t\rho = (\lambda y.\, s'\rho)(\bar{u}_n\rho)$ for some $\mathsf{Q}_{\approx}$-normal λ-term s'. By the induction hypothesis, $s'\rho$ and $\bar{u}_n\rho$ are $\mathsf{Q}_{\approx}$-normal. Therefore, $t\rho$ is also $\mathsf{Q}_{\approx}$-normal.

Case (iii): s is neither of the form $\mathsf{Q}\langle\tau\rangle$ nor a λ-expression. If s is of the form $\mathsf{f}\langle\bar{\tau}\rangle$ for some $\mathsf{f} \notin \{\forall, \exists\}$, then $t\rho = \mathsf{f}\langle\bar{\tau}\rho\rangle(\bar{u}_n\rho)$. By the induction hypothesis, $\bar{u}_n\rho$ are $\mathsf{Q}_{\approx}$-normal, and therefore $t\rho$ is also $\mathsf{Q}_{\approx}$-normal. Otherwise, s is a variable x and hence $t\rho = x\rho\,(\bar{u}_n\rho)$. Since $x\rho$ is $\mathsf{Q}_{\approx}$-normal by assumption and $\bar{u}_n\rho$ are $\mathsf{Q}_{\approx}$-normal by the induction hypothesis, it follows from (repeated application of) Lemma 7.7 that $t\rho$ is also $\mathsf{Q}_{\approx}$-normal. □

Lemma 7.9. *Let t be a λ-term of functional type that does not contain the variable x. If $\lambda x.\, t\,x$ is $\mathsf{Q}_{\approx}$-normal, then t is $\mathsf{Q}_{\approx}$-normal.*

Proof. Since $\lambda x.\, t\,x$ is $\mathsf{Q}_{\approx}$-normal, $t\,x$ is also $\mathsf{Q}_{\approx}$-normal. Let $s\,\bar{u}_n = t$. Since $t\,x$ is $\mathsf{Q}_{\approx}$-normal and x is not a λ-expression, s cannot be a quantifier by Lemma 7.6. Cases (ii) and (iii) are independent of $\bar{u}_n$, and hence removing x from that tuple does not affect $\mathsf{Q}_{\approx}$-normality. Thus, $t\,x$ being $\mathsf{Q}_{\approx}$-normal implies that t is $\mathsf{Q}_{\approx}$-normal. □

Lemma 7.10. *Let t be a λ-term and x a variable occurring only in green positions of t. Let t' be a term obtained via a $\beta\eta\mathsf{Q}_\eta$-normalization step from t. Then x occurs only in green positions of t'.*

Proof. By induction on the structure of t. If x does not occur in t, the claim is obvious. If $t = x$, there is no possible $\beta\eta\mathsf{Q}_\eta$-normalization step because for these steps the head of the rewritten term must be either a λ-expression or a quantifier. So we now assume that x does occur in t and that $t \neq x$. Then, by the assumption that x occurs only in green positions, t must be of the form $\mathsf{f}\langle\bar{\tau}\rangle\,\bar{u}$ for some $\mathsf{f} \in \Sigma \setminus \{\forall, \exists\}$, some types $\bar{\tau}$ and some λ-terms $\bar{u}$. The $\beta\eta\mathsf{Q}_\eta$-normalization step must take place in one of the $\bar{u}$, yielding $\bar{u}'$ such that $t' = \mathsf{f}\langle\bar{\tau}\rangle\,\bar{u}'$. By the induction hypothesis, x occurs only in green positions of $\bar{u}'$ and therefore only in green positions of t'. □

Lemma 7.11. *Let t be $\mathsf{Q}_{\approx}$-normal and let t' be obtained from t by a $\beta\eta\mathsf{Q}_\eta$-normalization step. If it is an η-reduction step, we assume that it happens not directly below a quantifier. Then t' is also $\mathsf{Q}_{\approx}$-normal.*

Proof. Let $s\,\bar{u}_n = t$. By Lemma 7.6, $\bar{u}_n$ are $\mathsf{Q}_{\approx}$-normal, and one of the following cases applies:

Case (i): s is of the form $\mathsf{Q}\langle\tau\rangle$, $n = 1$, and u_1 is of the form $\lambda y.\, v$ such that y occurs only in green positions of v. Then the normalization cannot happen at t, because s is of the form $\mathsf{Q}\langle\tau\rangle$ and u_1 is a λ-expression already. It cannot happen at u_1 by the assumption of this lemma. So it must happen in v, yielding some λ-term v'. Then $t = s\,(\lambda x.\, v')$. The λ-term v' is $\mathsf{Q}_{\approx}$-normal by the induction hypothesis and hence

$(\lambda x.\, v')$ is $\mathrm{Q}_{\approx}$-normal. Since x occurs only in green positions of v, by Lemma 7.10, x occurs only in green positions of v'. Thus, t' is $\mathrm{Q}_{\approx}$-normal.

Cases (ii) and (iii): s is a λ-expression whose body is $\mathrm{Q}_{\approx}$-normal; or s is neither of the form $\mathrm{Q}\langle\tau\rangle$ nor a λ-expression.

If the $\beta\eta\mathrm{Q}_\eta$-normalization step happens in some u_i, yielding some λ-term u_i', then u_i' is $\mathrm{Q}_{\approx}$-normal by the induction hypothesis. Thus, $t' = s\, u_1 \cdots u_{i-1}\, u_i'\, u_{i+1} \cdots u_n$ is also $\mathrm{Q}_{\approx}$-normal.

Otherwise, if $s = \lambda x.\, v$ and the $\beta\eta\mathrm{Q}_\eta$-normalization step happens in v, yielding some λ-term v', then v' is $\mathrm{Q}_{\approx}$-normal by the induction hypothesis. Thus, $t' = (\lambda x.v')\bar{u}_n$ is also $\mathrm{Q}_{\approx}$-normal.

Otherwise, the $\beta\eta\mathrm{Q}_\eta$-normalization step happens at $s\,\bar{u}_m$ for some $m \le n$, yielding some λ-term v'. Then $t' = v'\, u_{m+1} \cdots u_n$. The λ-terms s and $\bar{u}_m$ are $\mathrm{Q}_{\approx}$-normal and by repeated application of Lemma 7.7, $s\,\bar{u}_m$ is also $\mathrm{Q}_{\approx}$-normal. The λ-term v' is $\mathrm{Q}_{\approx}$-normal by Lemma 7.8 (for β-reductions) or Lemma 7.9 (for η-reductions). The normalization step cannot be a Q_η-normalization because s is not a quantifier. Since $\bar{u}_n$ are also $\mathrm{Q}_{\approx}$-normal, by repeated application of Lemma 7.7, $t[v'] = v'\, u_{m+1} \cdots u_n$ is also $\mathrm{Q}_{\approx}$-normal. □

A direct consequence of this lemma is that $\mathrm{Q}_{\approx}$ normality is invariant under $\beta\eta\mathrm{Q}_\eta$-normalization, as we wanted to show:

Corollary 7.12. *If t is a $\mathrm{Q}_{\approx}$-normal λ-term, then $t{\downarrow_{\beta\eta\mathrm{Q}_\eta}}$ is also $\mathrm{Q}_{\approx}$-normal.*

As mentioned above, the converse does not hold. Therefore, following our convention, $\mathrm{Q}_{\approx}$-normality is defined on terms (i.e., $\beta\eta$-equivalence classes) via $\beta\eta\mathrm{Q}_\eta$-normal forms. It follows that $\mathrm{Q}_{\approx}$-normality is well-behaved under applications of terms as well:

Lemma 7.13. *If t and v are $\mathrm{Q}_{\approx}$-normal terms where t is of functional type, then $t\,v$ is also $\mathrm{Q}_{\approx}$-normal.*

Proof. Since $\mathrm{Q}_{\approx}$-normality is defined via $\beta\eta\mathrm{Q}_\eta$-normal forms, we must show that if $t{\downarrow_{\beta\eta\mathrm{Q}_\eta}}$ and $v{\downarrow_{\beta\eta\mathrm{Q}_\eta}}$ are $\mathrm{Q}_{\approx}$-normal, then $t\,v{\downarrow_{\beta\eta\mathrm{Q}_\eta}}$ is $\mathrm{Q}_{\approx}$-normal. By Lemma 7.7, the λ-term $(t{\downarrow_{\beta\eta\mathrm{Q}_\eta}})(v{\downarrow_{\beta\eta\mathrm{Q}_\eta}})$ is $\mathrm{Q}_{\approx}$-normal. By Corollary 7.12, $((t{\downarrow_{\beta\eta\mathrm{Q}_\eta}})(v{\downarrow_{\beta\eta\mathrm{Q}_\eta}})){\downarrow_{\beta\eta\mathrm{Q}_\eta}} = (t\,v){\downarrow_{\beta\eta\mathrm{Q}_\eta}}$ is $\mathrm{Q}_{\approx}$-normal. □

Our preprocessing mechanism $\mathrm{Q}_{\approx}$-normalizes the input problem. It clearly terminates because each $\mathrm{Q}_{\approx}$-step reduces the number of quantifiers. The $\mathrm{Q}_{\approx}$-normality of the initial clause set of a derivation will be a precondition of our completeness theorem. Although inferences may produce $\mathrm{Q}_{\approx}$-reducible clauses, we do not $\mathrm{Q}_{\approx}$-normalize during the derivation process itself. Instead, $\mathrm{Q}_{\approx}$-reducible ground instances of clauses will be considered redundant by our redundancy criterion. Thus, clauses whose ground instances are all $\mathrm{Q}_{\approx}$-reducible can be deleted. However, there are $\mathrm{Q}_{\approx}$-reducible clauses, such as $x\,(\forall\langle\iota\rangle) \approx \mathsf{a}$, that are $\mathrm{Q}_{\approx}$-reducible but have $\mathrm{Q}_{\approx}$-normal ground instances. They must be kept because our completeness proof relies on their $\mathrm{Q}_{\approx}$-normal ground instances.

7.3.2. Term Orders and Selection Functions

The calculus is parameterized by a strict and a nonstrict term order, a literal selection function, and a Boolean subterm selection function. These concepts are defined below.

Definition 7.14 (Strict ground term order)**.** A *strict ground term order* is a well-founded strict total order $\succ$ on ground terms satisfying the following criteria, where $\succeq$ denotes the reflexive closure of $\succ$:

(O1) compatibility with green contexts: $s' \succ s$ implies $t\langle s'\rangle \succ t\langle s\rangle$;
(O2) green subterm property: $t\langle s\rangle \succeq s$;
(O3) $u \succ \bot \succ \top$ for all terms $u \neq \top, \bot$;
(O4) $\mathrm{Q}\langle\tau\rangle\, t \succ t\, u$ for all types τ, terms t, and terms u such that $\mathrm{Q}\langle\tau\rangle\, t$ and u are $\mathrm{Q}_{\approx}$-normal and the only Boolean green subterms of u are $\top$ and $\bot$.

Given a strict ground term order, we extend it to literals and clauses via the multiset extensions in the standard way [9, Section 2.4].

Based on this new notion of a strict ground term order, we define the strict and the nonstrict term order as in Chapter 5:

Definition 7.15 (Strict term order)**.** A *strict term order* is a relation $\succ$ on terms, literals, and clauses such that its restriction to ground entities is a strict ground term order and such that it is stable under grounding substitutions (i.e., $t \succ s$ implies $t\theta \succ s\theta$ for all substitutions θ grounding the entities t and s).

Definition 7.16 (Nonstrict term order)**.** Given a strict term order $\succ$ and its reflexive closure $\succeq$, a *nonstrict term order* is a relation $\succsim$ on terms, literals, and clauses such that $t \succsim s$ implies $t\theta \succeq s\theta$ for all θ grounding the entities t and s.

For the selection functions, we combine the restrictions imposed in Chapters 5 and 6:

Definition 7.17 (Literal selection function)**.** A literal selection function is a mapping from each clause to a subset of its literals. The literals in this subset are called *selected*. The following restrictions apply:

- A literal must not be selected if it is positive and neither side is $\bot$.
- A literal $L\langle y\rangle$ must not be selected if $y\,\bar{u}_n$, with $n > 0$, is a $\succeq$-maximal term of the clause (as per Definition 5.8).

Definition 7.18 (Boolean subterm selection function)**.** A Boolean subterm selection function is a function mapping each clause C to a subset of the green positions with Boolean subterms in C. The positions in this subset are called *selected* in C. Informally, we also say that the Boolean subterms at these positions are selected. The following restrictions apply:

- A subterm $s\langle y\rangle$ must not be selected if $y\,\bar{u}_n$, with $n > 0$, is a $\succeq$-maximal term of the clause.
- A subterm must not be selected if it is $\top$ or $\bot$ or a variable-headed term.
- A subterm must not be selected if it is at a topmost position on either side of a positive literal.

7.3.3. The Core Inference Rules

Let $\succ$ be a strict term order, let $\succsim$ be a nonstrict term order, let *HLitSel* be a literal selection function, and let *HBoolSel* be a Boolean subterm selection function. The calculus rules depend on the following auxiliary notions. We generalize the notion of eligibility from Chapter 6 to nonground higher-order terms:

Definition 7.19 (Eligibility)**.** A literal L is (*strictly*) $\trianglerighteq$*-eligible* w.r.t. a substitution σ in C for some relation $\trianglerighteq$ if it is selected in C or there are no selected literals and no selected Boolean subterms in C and $L\sigma$ is (strictly) $\trianglerighteq$-maximal in $C\sigma$.

The $\trianglerighteq$-eligible positions of a clause C w.r.t. a substitution σ are inductively defined as follows:

(E1) Any selected position is $\trianglerighteq$-eligible.

(E2) If a literal $L = s \mathrel{\dot{\approx}} t$ with $s\sigma \not\trianglelefteq t\sigma$ is either $\trianglerighteq$-eligible and negative or strictly $\trianglerighteq$-eligible and positive, then $L.s.\varepsilon$ is $\trianglerighteq$-eligible.

(E3) If the position p is $\trianglerighteq$-eligible and the head of $C|_p$ is not $\approx$ or $\not\approx$, the positions of all direct green subterms are $\trianglerighteq$-eligible.

(E4) If the position p is $\trianglerighteq$-eligible and $C|_p$ is of the form $s \approx t$ or $s \not\approx t$, then the position of s is $\trianglerighteq$-eligible if $s\sigma \not\trianglelefteq t\sigma$ and the position of t is $\trianglerighteq$-eligible if $s\sigma \not\trianglerighteq t\sigma$.

If σ is the identity substitution, we leave it implicit.

We define deeply occurring variables and fluid terms as in Chapter 5, but based on our new $\beta\eta\mathsf{Q}_\eta$-normal form and excluding λ-expressions directly below quantifiers:

Definition 7.20 (Deep occurrences)**.** A variable *occurs deeply* in a clause C if it occurs inside an argument of an applied variable or inside a λ-expression that is not directly below a quantifier.

For example, x and z occur deeply in $\mathsf{f}\,x\,y \approx y\,x \vee z \not\approx (\lambda w.\, z\,\mathsf{a}) \vee \forall\langle\iota\rangle(\lambda u.\, \mathsf{p}\,y) \approx \top$, whereas y does not occur deeply. Fluid terms are defined as in Chapter 5, using the $\beta\eta\mathsf{Q}_\eta$-normal form:

Definition 7.21 (Fluid terms)**.** A term t is called *fluid* if (1) $t\downarrow_{\beta\eta\mathsf{Q}_\eta}$ is of the form $y\,\bar{u}_n$ where $n \geq 1$, or (2) $t\downarrow_{\beta\eta\mathsf{Q}_\eta}$ is a λ-expression and there exists a substitution σ such that $t\sigma\downarrow_{\beta\eta\mathsf{Q}_\eta}$ is not a λ-expression (due to η-reduction).

The rules of our calculus are stated as follows. The superposition rule strongly resembles the one from Chapter 5 but uses our new notion of eligibility, and the new conditions 9 and 10 stem from the SUP rule of Chapter 6:

$$\frac{\overbrace{D' \vee t \approx t'}^{D} \qquad C\langle u\rangle}{(D' \vee C\langle t'\rangle)\sigma}\ \textsc{Sup}$$

1. u is not fluid;
2. u is not a variable deeply occurring in C;
3. *variable condition*: if u is a variable y, there must exist a grounding substitution θ such that $t\sigma\theta \succ t'\sigma\theta$ and $C\sigma\theta \prec C''\sigma\theta$, where $C'' = C\{y \mapsto t'\}$;
4. $\sigma \in \mathrm{CSU}(t, u)$;
5. $t\sigma \not\succsim t'\sigma$;
6. $C\sigma \not\succsim D\sigma$;
7. $t \approx t'$ is strictly $\succsim$-eligible in D w.r.t. σ;

8. the position of u is $\succsim$-eligible in C w.r.t. σ;
9. $t\sigma$ is not a fully applied logical symbol;
10. if $t'\sigma = \bot$, the position of the subterm u is at the top level of a positive literal.

The FluidSup rule is like the one in Chapter 5 but based on the Sup rule defined above. It is responsible for fluid green subterms.

$$\frac{\overbrace{D' \vee t \approx t'}^{D} \quad \overbrace{C' \vee s\langle u \rangle \mathrel{\dot{\approx}} s'}^{C}}{(D' \vee C' \vee s\langle z\, t' \rangle \mathrel{\dot{\approx}} s')\sigma}\ \textsc{FluidSup}$$

with the following side conditions, in addition to Sup's conditions 5 to 10:

1. u is a variable deeply occurring in C or u is fluid;
2. z is a fresh variable;
3. $\sigma \in \mathrm{CSU}(z\, t, u)$;
4. $(z\, t')\sigma \neq (z\, t)\sigma$.

The ERes and EFact rules are copied from Chapter 5. As a minor optimization, we add a condition to EFact that $u \approx v$ is $\succsim$-maximal, which is only necessary because positive literals of the form $u \approx \top$ can be selected.

$$\frac{\overbrace{C' \vee u \not\approx u'}^{C}}{C'\sigma}\ \textsc{ERes} \qquad \frac{\overbrace{C' \vee u' \approx v' \vee u \approx v}^{C}}{(C' \vee v \not\approx v' \vee u \approx v')\sigma}\ \textsc{EFact}$$

For ERes: $\sigma \in \mathrm{CSU}(u, u')$ and $u \not\approx u'$ is $\succsim$-eligible in C w.r.t. σ. For EFact: $\sigma \in \mathrm{CSU}(u, u')$, $u\sigma \not\precsim v\sigma$, $(u \approx v)\sigma$ is $\succsim$-maximal in $C\sigma$, and nothing is selected in C.

We also employ the ArgCong rule, which is identical with the one in Chapter 5:

$$\frac{\overbrace{C' \vee s \approx s'}^{C}}{C'\sigma \vee s\sigma\, \bar{x}_n \approx s'\sigma\, \bar{x}_n}\ \textsc{ArgCong}$$

where σ is the most general type substitution that ensures well-typedness of the conclusion. The literal $s \approx s'$ must be strictly $\succsim$-eligible in C w.r.t. σ, and $\bar{x}_n$ is a nonempty tuple of distinct fresh variables.

The following rules are concerned with Boolean reasoning and stem from Chapter 6. However, the rules must be adapted to cope with polymorphism and applied variables:

$$\frac{C\langle u \rangle}{(C\langle \bot \rangle \vee u \approx \top)\sigma}\ \textsc{BoolHoist}$$

1. σ is a type unifier of the type of u with the Boolean type o (i.e., the identity if u is Boolean or $\alpha \mapsto o$ if u is of type α for some type variable α);
2. the head of u is neither a variable nor a fully applied logical symbol;
3. the position of u is $\succsim$-eligible in C;
4. the occurrence of u is not at the top level of a positive literal.

$$\frac{\overbrace{C' \vee s \approx s'}^{C}}{C'\sigma}\ \textsc{FalseElim}$$

1. $\sigma \in \mathrm{CSU}(s \approx s', \bot \approx \top)$; 2. $s \approx s'$ is strictly $\succsim$-eligible in C w.r.t. σ.

$$\frac{C\langle u\rangle}{(C\langle \bot\rangle \vee x \approx y)\sigma}\ \textsc{EqHoist} \qquad \frac{C\langle u\rangle}{(C\langle \top\rangle \vee x \approx y)\sigma}\ \textsc{NeqHoist}$$

$$\frac{C\langle u\rangle}{(C\langle \bot\rangle \vee y\, x \approx \top)\sigma}\ \textsc{ForallHoist} \qquad \frac{C\langle u\rangle}{(C\langle \top\rangle \vee y\, x \approx \bot)\sigma}\ \textsc{ExistsHoist}$$

1. $\sigma \in \mathrm{CSU}(u, x \mathrel{\approx\!\!\!\approx} y)$, $\sigma \in \mathrm{CSU}(u, x \not\approx y)$, $\sigma \in \mathrm{CSU}(u, \forall\langle\alpha\rangle\, y)$, or $\sigma \in \mathrm{CSU}(u, \exists\langle\alpha\rangle\, y)$, respectively;
2. x, y, and α are fresh variables; 3. the position of u is $\succsim$-eligible in C w.r.t. σ;
4. if the head of u is a variable, it must be applied and the affected literal must be of the form $u \approx \top$, $u \approx \bot$, or $u \approx v$ where v is a variable-headed term.

$$\frac{C\langle u\rangle}{C\langle t'\rangle\sigma}\ \textsc{BoolRw}$$

1. $\sigma \in \mathrm{CSU}(t, u)$ and (t, t') is one of the following pairs:

$(\neg\bot, \top)$	$(\bot \wedge \bot, \bot)$	$(\bot \vee \bot, \bot)$	$(\bot \to \bot, \top)$
$(\neg\top, \bot)$	$(\top \wedge \bot, \bot)$	$(\top \vee \bot, \top)$	$(\top \to \bot, \bot)$
$(y \mathrel{\approx\!\!\!\approx} y, \top)$	$(\bot \wedge \top, \bot)$	$(\bot \vee \top, \top)$	$(\bot \to \top, \top)$
$(y \not\approx y, \bot)$	$(\top \wedge \top, \top)$	$(\top \vee \top, \top)$	$(\top \to \top, \top)$

   where y is a fresh variable;
2. u is not a variable; 3. the position of u is $\succsim$-eligible in C w.r.t. σ;
4. if the head of u is a variable, it must be applied and the affected literal must be of the form $u \approx \top$, $u \approx \bot$, or $u \approx v$ where v is a variable-headed term.

$$\frac{C\langle u\rangle}{C\langle y\,(\mathsf{sk}_{\Pi\bar{\alpha}.\forall\bar{x}.\exists z.\neg(y\sigma\, z)}\langle\bar{\alpha}\rangle\bar{x})\rangle\sigma}\ \textsc{ForallRw}$$

$$\frac{C\langle u\rangle}{C\langle y\,(\mathsf{sk}_{\Pi\bar{\alpha}.\forall\bar{x}.\exists z.\, y\sigma\, z}\langle\bar{\alpha}\rangle\bar{x})\rangle\sigma}\ \textsc{ExistsRw}$$

1. $\sigma \in \mathrm{CSU}(\forall\langle\beta\rangle\, y, u)$ and $\sigma \in \mathrm{CSU}(\exists\langle\beta\rangle\, y, u)$, respectively, where β is a fresh type variable, y is a fresh term variable, $\bar{\alpha}$ are the free type variables and $\bar{x}$ are the free term variables occurring in $y\sigma$ in order of first appearance;
2. u is not a variable; 3. the position of u is $\succsim$-eligible in C w.r.t. σ;
4. if the head of u is a variable, it must be applied and the affected literal must be of the form $u \approx \top$, $u \approx \bot$, or $u \approx v$ where v is a variable-headed term;
5. for ForallRw, the indicated occurrence of u is not in a literal $u \approx \top$, and for ExistsRw, the indicated occurrence of u is not in a literal $u \approx \bot$.

In principle, the subscript of the Skolems above could be normalized using Boolean tautologies to share as many Skolem symbols as possible. This is an extension of our calculus that we did not investigate any further.

Like SUP, also the Boolean rules must be simulated inside fluid terms. As an analogue of FLUIDSUP, we introduce the rules FLUIDBOOLHOIST and FLUIDLOOBHOIST:

$$\frac{C\langle u\rangle}{(C\langle z\,\bot\rangle \vee x \approx \top)\sigma}\ \textsc{FluidBoolHoist}$$

1. u is fluid;
2. z and x are fresh variables;
3. $\sigma \in \mathrm{CSU}(z\,x, u)$;
4. $(z\,\bot)\sigma \neq (z\,x)\sigma$;
5. $x\sigma \neq \top$ and $x\sigma \neq \bot$;
6. the position of u is $\gtrsim$-eligible in C w.r.t. σ.

$$\frac{C\langle u\rangle}{(C\langle z\,\top\rangle \vee x \approx \bot)\sigma}\ \textsc{FluidLoobHoist}$$

Same conditions as FLUIDBOOLHOIST, but $\bot$ is replaced by $\top$ in condition 4.

Finally, in addition to the inference rules, our calculus employs two axioms. The axiom (EXT) is concerned with functional extensionality and is identical with the one in Chapter 5. The axiom (CHOICE) axiomatizes the semantics of the Hilbert choice operator ε.

$$z\,(\mathsf{diff}\langle\alpha,\beta\rangle\,z\,y) \not\approx y\,(\mathsf{diff}\langle\alpha,\beta\rangle\,z\,y) \vee z \approx y \qquad \text{(EXT)}$$

$$y\,x \approx \bot \vee y\,(\varepsilon\langle\alpha\rangle\,y) \approx \top \qquad \text{(CHOICE)}$$

where $\mathsf{diff}\langle\alpha,\beta\rangle$ abbreviates $\mathsf{sk}_{\Pi\alpha\,\beta.\forall z\,y.\,\exists x.\,z\,x \not\approx y\,x}$.

7.3.4. Rationale for the Rules

Most of the calculus rules are adapted from precursor calculi. The rules SUP, ERES, and EFACT were already present in Bachmair and Ganzinger's calculus for first-order logic, with slightly different side conditions. Most notably, as in Chapters 3 and 5, SUP inferences are only required into green contexts. Other subterms are accessed indirectly via the ARGCONG rule and the axiom (EXT).

The rules BOOLHOIST, FALSEELIM, EQHOIST, NEQHOIST, FORALLHOIST, EXISTSHOIST, BOOLRW, FORALLRW, and EXISTSRW, concerned with Boolean reasoning, stem from the calculus presented in the previous chapter, originally inspired by Ganzinger and Stuber's work. The rules have been lifted to the nonground calculus. Except for the first two, all of these rules have a condition stating: "if the head of u is a variable, it must be applied and the affected literal must be of the form $u \approx \top$, $u \approx \bot$, or $u \approx v$ where v is a variable-headed term." The inferences at variable-headed terms permitted by this condition are our form of primitive substitution—i.e., our mechanism to instantiate applied variables with logical symbols. The following example illustrates this mechanism:

Example 7.22 (Leibniz equality)**.** Our calculus can prove that Leibniz equality implies equality as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{z\,\mathsf{a} \approx \bot \vee z\,\mathsf{b} \approx \top}{(x\,\mathsf{a} \mathrel{\approx\!\!\!\approx} y\,\mathsf{a}) \approx \bot \vee \bot \approx \top \vee x\,\mathsf{b} \approx y\,\mathsf{b}}\,\textsc{EqHoist}}{\top \approx \bot \vee \bot \approx \top \vee w\,\mathsf{a}\,\mathsf{b}\,\mathsf{b} \approx w\,\mathsf{b}\,\mathsf{a}\,\mathsf{b}}\,\textsc{BoolRw}}{\bot \approx \top \vee w\,\mathsf{a}\,\mathsf{b}\,\mathsf{b} \approx w\,\mathsf{b}\,\mathsf{a}\,\mathsf{b}}\,\textsc{FalseElim}}{w\,\mathsf{a}\,\mathsf{b}\,\mathsf{b} \approx w\,\mathsf{b}\,\mathsf{a}\,\mathsf{b}}\,\textsc{FalseElim}}{\dfrac{\mathsf{a} \not\approx \mathsf{b} \qquad w\,\mathsf{a}\,\mathsf{b}\,\mathsf{b} \approx w\,\mathsf{b}\,\mathsf{a}\,\mathsf{b}}{\mathsf{a} \not\approx \mathsf{a}}\,\textsc{Sup}}\,\textsc{ERes}}{\bot}$$

The EqHoist step at the top illustrates how our calculus copes without a dedicated primitive substitution rule. Although $\mathrel{\approx\!\!\!\approx}$ does not appear in the premise, we still need to apply EqHoist on $z\,\mathsf{b}$ with $\mathrm{CSU}(z\,\mathsf{b}, x_0 \mathrel{\approx\!\!\!\approx} y_0) = \{\{z \mapsto \lambda v.\, x\,v \mathrel{\approx\!\!\!\approx} y\,v,\ x_0 \mapsto x\,\mathsf{b},\ y_0 \mapsto y\,\mathsf{b}\}\}$. Other calculi [2, 23, 73, 126] would apply an explicit primitive substitution rule instead, yielding essentially $(x\,\mathsf{a} \mathrel{\approx\!\!\!\approx} y\,\mathsf{a}) \approx \bot \vee (x\,\mathsf{b} \mathrel{\approx\!\!\!\approx} y\,\mathsf{b}) \approx \top$. However, this clause is subsumed by the premise and thus redundant according to our redundancy criterion—i.e., it could be discarded immediately. By hoisting the equality to the clausal level, we can bypass the redundancy criterion.

Next, BoolRw can be applied to $x\,\mathsf{a} \mathrel{\approx\!\!\!\approx} y\,\mathsf{a}$ with $\mathrm{CSU}(x\,\mathsf{a} \mathrel{\approx\!\!\!\approx} y\,\mathsf{a},\ y_0 \mathrel{\approx\!\!\!\approx} y_0) = \{\{x \mapsto \lambda v.\, w\,\mathsf{a}\,v\,v,\ y \mapsto \lambda v.\, w\,v\,\mathsf{a}\,v,\ y_0 \mapsto w\,\mathsf{a}\,\mathsf{a}\,\mathsf{a}\}\}$. The following two steps use FalseElim to remove the $\bot \approx \top$ literals. Then Sup applies with the unifier $\{w \mapsto \lambda x_1\, x_2\, x_3.\, x_2\} \in \mathrm{CSU}(\mathsf{b}, w\,\mathsf{a}\,\mathsf{b}\,\mathsf{b})$, and ERes derives the contradiction.

This mechanism resembling primitive substitution is not the only way our calculus can instantiate variables with logical symbols. Often, the correct instantiation can also be found by unification with a logical symbol that is already present:

Example 7.23. The following derivation shows that there exists a function y that is equivalent to the conjunction of $\mathsf{p}\,x$ and $\mathsf{q}\,x$ for all arguments x:

$$\dfrac{\dfrac{\dfrac{\dfrac{\exists\langle\iota\rangle(\lambda y.\, \forall\langle\iota\rangle(\lambda x.\, y\,x \mathrel{\approx\!\!\!\approx} (\mathsf{p}\,x \wedge \mathsf{q}\,x))) \approx \bot}{\top \approx \bot \vee \forall\langle\iota\rangle(\lambda x.\, y'\,x \mathrel{\approx\!\!\!\approx} (\mathsf{p}\,x \wedge \mathsf{q}\,x)) \approx \bot}\,\textsc{ExistsHoist}}{\top \approx \bot \vee (y'(\mathsf{sk}\,y') \mathrel{\approx\!\!\!\approx} (\mathsf{p}(\mathsf{sk}\,y') \wedge \mathsf{q}(\mathsf{sk}\,y'))) \approx \bot}\,\textsc{ForallRw}}{\top \approx \bot \vee \top \approx \bot}\,\textsc{BoolRw}}{\top \approx \bot}\,\textsc{FalseElim}}{\bot}\,\textsc{FalseElim}$$

Here, sk stands for $\mathsf{sk}_{\forall u.\, \exists v.\, \neg\, u\,v \mathrel{\approx\!\!\!\approx} (\mathsf{p}\,v \wedge \mathsf{q}\,v)}$. First, we use ExistsHoist to resolve the existential quantifier, using the unifier $\{\alpha \mapsto \iota,\ z \mapsto \lambda y.\, \forall\langle\iota\rangle(\lambda x.\, y\,x \mathrel{\approx\!\!\!\approx} (\mathsf{p}\,x \wedge \mathsf{q}\,x))\} \in \mathrm{CSU}(\exists\langle\iota\rangle(\lambda y.\, \forall\langle\iota\rangle(\lambda x.\, y\,x \mathrel{\approx\!\!\!\approx} (\mathsf{p}\,x \wedge \mathsf{q}\,x))), \exists\langle\alpha\rangle z)$ for fresh variables α, y', and z. Then ForallRw skolemizes the universal quantifier, using the unifier $\{\beta \mapsto \iota,\ z' \mapsto \lambda x.\, y'\,x \mathrel{\approx\!\!\!\approx} (\mathsf{p}\,x \wedge \mathsf{q}\,x)\} \in \mathrm{CSU}(\forall\langle\beta\rangle z', \forall\langle\iota\rangle(\lambda x.\, y'\,x \mathrel{\approx\!\!\!\approx} (\mathsf{p}\,x \wedge \mathsf{q}\,x)))$ for fresh variables β and z'. The Skolem symbol takes y' as argument because it occurs free in $\lambda x.\, y'\,x \mathrel{\approx\!\!\!\approx} (\mathsf{p}\,x \wedge \mathsf{q}\,x)$.

Then BOOLRW applies because the terms $y'(\mathsf{sk}\,y')$ and $\mathsf{p}(\mathsf{sk}\,y') \wedge \mathsf{q}(\mathsf{sk}\,y')$ are unifiable and thus $y'(\mathsf{sk}\,y') \approx (\mathsf{p}(\mathsf{sk}\,y') \wedge \mathsf{q}(\mathsf{sk}\,y'))$ is unifiable with $y \approx y$. Finally, two FALSEELIM inferences lead to the empty clause.

As in Chapter 5, the FLUIDSUP rule is responsible to simulate superposition inferences below applied variables, other fluid terms, or deeply occurring variables. In addition, our calculus introduces the rules FLUIDBOOLHOIST and FLUIDLOOBHOIST. They simulate the various Boolean inference rules below fluid terms. Initially, we considered adding a fluid version for each rule that operates on Boolean subterms. While possible, it seems excessive, and we discovered that the two rules FLUIDBOOLHOIST and FLUIDLOOBHOIST can achieve refutational completeness, too.

The following example demonstrates the need for the rules FLUIDBOOLHOIST and FLUIDLOOBHOIST.

Example 7.24. Consider the following clause set:

$$\mathsf{h}(y\,\mathsf{b}) \not\approx \mathsf{h}(\mathsf{g}\,\bot) \vee \mathsf{h}(y\,\mathsf{a}) \not\approx \mathsf{h}(\mathsf{g}\,\top) \qquad \mathsf{a} \not\approx \mathsf{b}$$

This clause set is unsatisfiable because the instantiation $y \mapsto \lambda x.\, \mathsf{g}\,(x \approx \mathsf{a})$ yields the clause $\mathsf{h}(\mathsf{g}\,(\mathsf{a} \approx \mathsf{b})) \not\approx \mathsf{h}(\mathsf{g}\,\bot) \vee \mathsf{h}(\mathsf{g}\,(\mathsf{a} \approx \mathsf{a})) \not\approx \mathsf{h}(\mathsf{g}\,\top)$, which is clearly unsatisfiable in conjunction with $\mathsf{a} \not\approx \mathsf{b}$.

The literal selection function could select either literal in the first clause. ERES is applicable in either case, but the unifiers $\{y \mapsto \lambda x.\, \mathsf{g}\,\bot\}$ and $\{y \mapsto \lambda x.\, \mathsf{g}\,\top\}$ do not lead to a contradiction. Instead, we need to apply FLUIDBOOLHOIST if the first literal is selected or FLUIDLOOBHOIST if the second literal is selected. The derivation with FLUIDBOOLHOIST proceeds as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{h}(y\,\mathsf{b}) \not\approx \mathsf{h}(\mathsf{g}\,\bot) \vee \mathsf{h}(y\,\mathsf{a}) \not\approx \mathsf{h}(\mathsf{g}\,\top)}{\mathsf{h}(z'\,\mathsf{b}\,\bot) \not\approx \mathsf{h}(\mathsf{g}\,\bot) \vee \mathsf{h}(z'\,\mathsf{a}\,(x'\mathsf{a})) \not\approx \mathsf{h}(\mathsf{g}\,\top) \vee x'\,\mathsf{b} \approx \top}\,\textsc{FluidBoolHoist}}{\mathsf{h}(\mathsf{g}\,(x'\,\mathsf{a})) \not\approx \mathsf{h}(\mathsf{g}\,\top) \vee x'\,\mathsf{b} \approx \top}\,\textsc{ERes} \qquad}{\mathsf{a} \not\approx \mathsf{b} \qquad \mathsf{h}(\mathsf{g}\,(x''\,\mathsf{a} \approx x'''\,\mathsf{a})) \not\approx \mathsf{h}(\mathsf{g}\,\top) \vee \bot \approx \top \vee x''\,\mathsf{b} \approx x'''\,\mathsf{b}}\,\textsc{EqHoist}}{\mathsf{h}(\mathsf{g}\,(\mathsf{a} \approx x'''\,\mathsf{a})) \not\approx \mathsf{h}(\mathsf{g}\,\top) \vee \bot \approx \top \vee \mathsf{a} \not\approx x'''\,\mathsf{b}}\,\textsc{Sup}}{\mathsf{h}(\mathsf{g}\,\top) \not\approx \mathsf{h}(\mathsf{g}\,\top) \vee \bot \approx \top \vee \mathsf{a} \not\approx \mathsf{a}}\,\textsc{BoolRw}}{\bot \approx \top \vee \mathsf{a} \not\approx \mathsf{a}}\,\textsc{ERes}}{\bot \approx \top}\,\textsc{ERes}}{\bot}\,\textsc{FalseElim}$$

The FLUIDBOOLHOIST inference uses the unifier $\{y \mapsto \lambda u.z'\,u\,(x'\,u),\ z \mapsto \lambda u.z'\,\mathsf{b}\,u,\ x \mapsto x'\,\mathsf{b}\} \in \mathrm{CSU}(z\,x,\ y\,\mathsf{b})$. We apply ERES to the first literal of the resulting clause, with unifier $\{z' \mapsto \lambda u v.\, \mathsf{g}\,v\} \in \mathrm{CSU}(\mathsf{h}(z'\,\mathsf{b}\,\bot),\ \mathsf{h}(\mathsf{g}\,\bot))$. Next, we apply EQHOIST with unifier $\{x' \mapsto \lambda u.\, x''\,u \approx x'''\,u,\ w \mapsto x''\,\mathsf{b},\ w' \mapsto x'''\,\mathsf{b}\} \in \mathrm{CSU}(x'\,\mathsf{b},\ w \approx w')$ to the literal created by FLUIDBOOLHOIST, effectively performing a primitive substitution. The resulting clause can superpose into $\mathsf{a} \not\approx \mathsf{b}$ with unifier $\{x'' \mapsto \lambda u.u\} \in \mathrm{CSU}(x''\,\mathsf{b},\ \mathsf{b})$. The two sides of the interpreted equality in the first literal can then be unified, allowing us to apply BOOLRW with unifier $\{y \mapsto \mathsf{a},\ x''' \mapsto \lambda u.\, \mathsf{a}\} \in \mathrm{CSU}(y \approx y,\ \mathsf{a} \approx x'''\,\mathsf{b})$. Finally, applying ERES twice and FALSEELIM once yields the empty clause.

7.3.5. Soundness

All of our inference rules and axioms are sound w.r.t. $\approx$ and the ones that do not introduce Skolem symbols are also sound w.r.t. $\models$. The preprocessing is sound w.r.t. both $\models$ and $\approx$:

Lemma 7.25. *$\mathsf{Q}_{\approx}$-normalization preserves denotations of terms and truth of clauses w.r.t. proper interpretations.*

Proof. It suffices to show that

$$[\![\forall\langle\tau\rangle]\!]^{\xi}_{\mathcal{I}} = [\![\lambda y.\, y \approx (\lambda x.\, \mathsf{T})]\!]^{\xi}_{\mathcal{I}} \quad \text{and} \quad [\![\exists\langle\tau\rangle]\!]^{\xi}_{\mathcal{I}} = [\![\lambda y.\, y \not\approx (\lambda x.\, \bot)]\!]^{\xi}_{\mathcal{I}}$$

for all types τ, proper interpretations $\mathcal{I} = (\mathcal{I}_{\mathrm{ty}}, \mathcal{J}, \mathcal{L})$, and all valuations ξ.

Let f be a function from $[\![\tau]\!]^{\xi}_{\mathcal{I}_{\mathrm{ty}}}$ to $\{0,1\}$. Then

$$[\![\forall\langle\tau\rangle]\!]^{\xi}_{\mathcal{I}}(f) = \mathcal{J}(\forall, [\![\tau]\!]^{\xi}_{\mathcal{I}_{\mathrm{ty}}})(f) = \min\{f(a) \mid a \in [\![\tau]\!]^{\xi}_{\mathcal{I}_{\mathrm{ty}}}\} = \begin{cases} 1 & \text{if } f \text{ is constantly 1} \\ 0 & \text{otherwise} \end{cases}$$

By the definition of proper interpretations, we have

$$[\![\lambda y.\, y \approx (\lambda x.\, \mathsf{T})]\!]^{\xi}_{\mathcal{I}}(f) = [\![y \approx (\lambda x.\, \mathsf{T})]\!]^{\xi[y\mapsto f]}_{\mathcal{I}} = \begin{cases} 1 & \text{if } [\![y]\!]^{\xi[y\mapsto f]}_{\mathcal{I}} = [\![\lambda x.\, \mathsf{T}]\!]^{\xi[y\mapsto f]}_{\mathcal{I}} \\ 0 & \text{otherwise} \end{cases}$$

Thus it remains to show that $[\![y]\!]^{\xi[y\mapsto f]}_{\mathcal{I}} = [\![\lambda x.\, \mathsf{T}]\!]^{\xi[y\mapsto f]}_{\mathcal{I}}$ if and only if f is constantly 1. This holds because by the definition of term denotation, $[\![y]\!]^{\xi[y\mapsto f]}_{\mathcal{I}} = f$ and because $[\![\lambda x.\, \mathsf{T}]\!]^{\xi[y\mapsto f]}_{\mathcal{I}}(a) = [\![\mathsf{T}]\!]^{\xi[x\mapsto a, y\mapsto f]}_{\mathcal{I}} = 1$ by properness of the interpretation, for all $a \in [\![\tau]\!]^{\xi}_{\mathcal{I}_{\mathrm{ty}}}$. The case of $\exists$ is analogous. □

To show satisfiability preservation of the inferences, we need the substitution lemma for our logic:

Lemma 7.26 (Substitution lemma)**.** *Let $\mathcal{I} = (\mathcal{I}_{\mathrm{ty}}, \mathcal{J}, \mathcal{L})$ be a proper interpretation. Then*

$$[\![\tau\rho]\!]^{\xi}_{\mathcal{I}_{\mathrm{ty}}} = [\![\tau]\!]^{\xi'}_{\mathcal{I}_{\mathrm{ty}}} \text{ and } [\![t\rho]\!]^{\xi}_{\mathcal{I}} = [\![t]\!]^{\xi'}_{\mathcal{I}}$$

for all terms t, all types τ, and all substitutions ρ, where $\xi'(\alpha) = [\![\alpha\rho]\!]^{\xi}_{\mathcal{I}_{\mathrm{ty}}}$ for all type variables α and $\xi'(x) = [\![x\rho]\!]^{\xi}_{\mathcal{I}}$ for all term variables x.

Proof. Analogous to Lemma 5.18. □

It follows that a model of a clause is also a model of its instances:

Lemma 7.27. *If $\mathcal{I} \models C$ for some interpretation $\mathcal{I}$ and some clause C, then $\mathcal{I} \models C\rho$ for all substitutions ρ.*

Proof. Analogous to Lemma 5.19, using Lemma 7.26. □

With this lemma in place, we can prove the soundness of our calculus. Some of the rules and axioms are only sound w.r.t. $\approx$.

Theorem 7.28 (Soundness)**.** *The axiom* (CHOICE) *and all of our inference rules, except for* FORALLRW *and* EXISTSRW*, are sound w.r.t.* $\models$*. All of our axioms and inference rules are sound w.r.t.* $\models\!\!\approx$*. Both of these claims hold even without the variable condition and the side conditions on fluidity, deeply occurring variables, order, and eligibility.*

Proof. Analogous to Lemma 5.20. For the Boolean rules, we make use of the special requirements on interpretations of logical symbols.

We elaborate on the soundness of FORALLRW, EXISTSRW, and EXT w.r.t. $\models\!\!\approx$.

For FORALLRW: Let $\mathcal{I}$ be a Skolem-aware model of $C\langle u\rangle$. By Lemma 7.26, $\mathcal{I}$ is a model of $C\langle u\rangle\sigma$ as well. Since $\sigma \in \mathrm{CSU}(\forall\langle\beta\rangle\, y, u)$, we have $C\langle u\rangle\sigma = C\langle\forall\langle\beta\rangle\, y\rangle\sigma$. Thus, to show that $\mathcal{I}$ is also a model of the conclusion $C\langle y\,(\mathsf{sk}_{\Pi\bar{\alpha}.\forall\bar{x}.\exists z.\neg y\sigma z}\langle\bar{\alpha}\rangle\bar{x})\rangle\sigma$, it suffices to show that $\mathcal{I} \models \forall\langle\beta\sigma\rangle(y\sigma) \approx y\sigma\,(\mathsf{sk}_{\Pi\bar{\alpha}.\forall\bar{x}.\exists z.\neg y\sigma z}\langle\bar{\alpha}\rangle\bar{x})$. This follows directly from the definition of Skolem-awareness, which states that

$$\mathcal{I} \models (\exists\langle\beta\sigma\rangle(\lambda z.\, \neg\, y\sigma\, z)) \approx \neg\, y\sigma\,(\mathsf{sk}_{\Pi\bar{\alpha}.\forall\bar{x}.\exists z.\neg y\sigma z}\langle\bar{\alpha}\rangle\bar{x})$$

For EXISTSRW, we can argue analogously.

For (EXT), we must show that any Skolem-aware model $\mathcal{I}$ is a model of the axiom (EXT) $z\,(\mathsf{diff}\langle\alpha,\beta\rangle z\, y) \not\approx y\,(\mathsf{diff}\langle\alpha,\beta\rangle z\, y) \vee z \approx y$. By the definition of Skolem-awareness, we have $\mathcal{I} \models (\exists\langle\alpha\rangle(\lambda x.\, z\, x \not\approx y\, x)) \approx (\lambda x.\, z\, x \not\approx y\, x)(\mathsf{diff}\langle\alpha,\beta\rangle z\, y)$. Thus, if the first literal of (EXT) is false in $\mathcal{I}$ for some valuation ξ, then

$$\begin{aligned} 0 &= [\![(\lambda x.\, z\, x \not\approx y\, x)(\mathsf{diff}\langle\alpha,\beta\rangle z\, y)]\!]^{\xi}_{\mathcal{I}} \\ &= [\![\exists\langle\alpha\rangle(\lambda x.\, z\, x \not\approx y\, x)]\!]^{\xi}_{\mathcal{I}} \\ &= \max\{[\![\lambda x.\, z\, x \not\approx y\, x]\!]^{\xi}_{\mathcal{I}}(a) \mid a \in [\![\alpha]\!]^{\xi}_{\mathcal{I}}\} \\ &= \max\{[\![z\, x \not\approx y\, x]\!]^{\xi[x\mapsto a]}_{\mathcal{I}} \mid a \in [\![\alpha]\!]^{\xi}_{\mathcal{I}}\} \end{aligned}$$

It follows that there exists no $a \in [\![\alpha]\!]^{\xi}_{\mathcal{I}}$ such that $[\![z\, x]\!]^{\xi[x\mapsto a]}_{\mathcal{I}} = [\![z]\!]^{\xi}_{\mathcal{I}}(a)$ and $[\![y\, x]\!]^{\xi[x\mapsto a]}_{\mathcal{I}} = [\![y]\!]^{\xi}_{\mathcal{I}}(a)$ are different. Thus, $[\![z]\!]^{\xi}_{\mathcal{I}} = [\![y]\!]^{\xi}_{\mathcal{I}}$ and hence the second literal of (EXT) must be true under $\mathcal{I}$ and ξ. □

7.3.6. The Redundancy Criterion

As in previous chapters, the redundancy criterion and the completeness proof distinguish three levels of logics. In this chapter, we have a higher-order level H, a $\mathrm{Q}_{\approx}$-normal ground higher-order level GH, and a ground monomorphic first-order level GF with an interpreted Boolean type. We use $\mathcal{T}_{\mathrm{H}}$, $\mathcal{T}_{\mathrm{GH}}$, and $\mathcal{T}_{\mathrm{GF}}$ to denote the respective sets of terms, $\mathcal{T}y_{\mathrm{H}}$, $\mathcal{T}y_{\mathrm{GH}}$, and $\mathcal{T}y_{\mathrm{GF}}$ to denote the respective sets of types, and $\mathcal{C}_{\mathrm{H}}$, $\mathcal{C}_{\mathrm{GH}}$, and $\mathcal{C}_{\mathrm{GF}}$ to denote the respective sets of clauses. We will define a grounding function $\mathcal{G}$ that connects levels H and GH and an encoding $\mathcal{F}$ that connects levels GH and GF.

Let $(\Sigma_{\mathsf{ty}}, \Sigma)$ be the signature of level H. The level GH has the same signature but is restricted to ground $\mathrm{Q}_{\approx}$-normal terms and clauses. For the GF level, we employ the logic defined in Section 6.2. Its signature $(\Sigma_{\mathsf{ty}}, \Sigma_{\mathrm{GF}})$ is defined as follows. The type constructors Σ_{ty} are the same in both signatures, but $\to$ is an uninterpreted symbol on GF.

For each ground instance $\mathsf{f}\langle\bar{\upsilon}\rangle : \tau_1 \to \cdots \to \tau_n \to \tau$ of a symbol $\mathsf{f} \in \Sigma$, we introduce a first-order symbol $\mathsf{f}_j^{\bar{\upsilon}} \in \Sigma_{\mathrm{GF}}$ of type $\bar{\tau}_j \Rightarrow (\tau_{j+1} \to \cdots \to \tau_n \to \tau)$, for each j. This is done for both logical and nonlogical symbols. Moreover, for each ground term $\lambda x.\, t$, we introduce a symbol $\mathsf{lam}_{\lambda x.\, t} \in \Sigma_{\mathrm{GF}}$ of the same type. The symbols $\bot_0$, $\top_0$, $\neg_1$, $\wedge_2$, $\vee_2$, $\to_2$, $\approx_2^\tau$, and $\not\approx_2^\tau$ are identified with the corresponding first-order logical symbols.

Definition 7.29 (The grounding function $\mathcal{G}$ on terms and clauses)**.** Given a clause $C \in \mathcal{C}_{\mathrm{H}}$, let its *ground instances* $\mathcal{G}(C)$ be the set of all clauses in $\mathcal{C}_{\mathrm{GH}}$ of the form $C\theta$ for some grounding substitution θ such that for all variables x occurring in C, the only Boolean green subterms of $x\theta$ are $\top$ and $\bot$.

Restricting the grounding to the Boolean terms $\top$ and $\bot$ allows condition (O4) to consider only terms u with Boolean subterms $\top$ and $\bot$. This is crucial because without the restriction no suitable term order would exist. The approach resembles basic superposition [11], where the redundancy criterion only considers ground instances that are irreducible w.r.t. an arbitrary term rewriting system. A disadvantage of basic superposition is that its redundancy criterion severely restricts the simplification machinery because the irreducible terms are unknown during a derivation. In our setting, however, we know that $\top$ and $\bot$ will be normal forms of the term rewriting system used in the completeness proof. Thus we can restrict grounding to the Boolean terms $\top$ and $\bot$ without compromising the simplification machinery.

Since we have defined all clauses in $\mathcal{C}_{\mathrm{GH}}$ to be $\mathsf{Q}_\approx$-normal, the ground instances $\mathcal{G}(C)$ of a clause are $\mathsf{Q}_\approx$-normal as well. The clauses in $\mathcal{C}_{\mathrm{GH}}$ being $\mathsf{Q}_\approx$-normal allows us to define the encoding $\mathcal{F}$ as follows:

Definition 7.30 (The encoding $\mathcal{F}$ on terms and clauses)**.** The encoding $\mathcal{F} : \mathcal{T}_{\mathrm{GH}} \to \mathcal{T}_{\mathrm{GF}}$ is recursively defined by

$$\mathcal{F}(\forall\langle\tau\rangle(\lambda x.\, t)) = \forall x.\, \mathcal{F}(t) \qquad \mathcal{F}(\exists\langle\tau\rangle(\lambda x.\, t)) = \exists x.\, \mathcal{F}(t)$$
$$\mathcal{F}(x) = x \qquad \mathcal{F}(\lambda x.\, t) = \mathsf{lam}_{\lambda x.\, t} \qquad \mathcal{F}(\mathsf{f}\langle\bar{\upsilon}\rangle\bar{s}_j) = \mathsf{f}_j^{\bar{\upsilon}}(\mathcal{F}(\bar{s}_j))$$

using $\downarrow_{\beta\eta\mathsf{Q}_\eta}$ representatives of terms. The encoding $\mathcal{F}$ is extended to map from $\mathcal{C}_{\mathrm{GH}}$ to $\mathcal{C}_{\mathrm{GF}}$ by mapping each literal and each side of a literal individually.

The encoding of variables is necessary for variables bound by $\forall$ and $\exists$. Since the terms $\mathcal{T}_{\mathrm{GH}}$ are $\mathsf{Q}_\approx$-normal, these variables occur neither applied nor inside λ-expressions.

Thus our schematic overview of the three levels looks as follows:

$$\begin{array}{ccccc} \text{H} & & \text{GH} & & \text{GF} \\ \text{higher-order} & \xrightarrow{\ \mathcal{G}\ } & \text{ground } \mathsf{Q}_\approx\text{-normal} & \xrightarrow{\ \mathcal{F}\ } & \text{ground first-order} \\ & & \text{higher-order} & & \text{with interpreted Booleans} \end{array}$$

The mapping $\mathcal{F}$ is clearly bijective. As in previous chapters, the order $\succ$ can be transferred from $\mathcal{T}_{\mathrm{GH}}$ to $\mathcal{T}_{\mathrm{GF}}$ and from $\mathcal{C}_{\mathrm{GH}}$ to $\mathcal{C}_{\mathrm{GF}}$ by defining $t \succ s$ as $\mathcal{F}^{-1}(t) \succ \mathcal{F}^{-1}(s)$ and $C \succ D$ as $\mathcal{F}^{-1}(C) \succ \mathcal{F}^{-1}(D)$. The property that $\succ$ on clauses is the multiset

extension of $\succ$ on literals, which in turn is the multiset extension of $\succ$ on terms, is maintained because $\mathcal{F}^{-1}$ maps the multiset representations elementwise.

As with previous incarnations of the function $\mathcal{F}$, there is a correspondence between green subterms on GH and subterms on GF. This time, it is restricted to subterms on GF that are not below quantifiers:

Lemma 7.31. *Let $s, t \in \mathcal{T}_{\mathrm{GH}}$. If p is a green position in t or a position in $\mathcal{F}(t)$ that is not below a quantifier, we have $\mathcal{F}(t\langle s\rangle_p) = \mathcal{F}(t)[\mathcal{F}(s)]_p$. In other words, s is a green subterm of t at position p if and only if $\mathcal{F}(s)$ is a subterm of $\mathcal{F}(t)$ at position p that is not below a quantifier.*

Proof. Analogous to Lemma 3.18. □

Lemma 7.32. *The relation $\succ$ on $\mathcal{T}_{\mathrm{GF}}$ is a term order in the sense of Definition 6.1.*

Proof. Transitivity and irreflexivity follow directly from transitivity and irreflexivity of $\succ$ on $\mathcal{T}_{\mathrm{GH}}$. Well-foundedness, compatibility with contexts, subterm property and totality can be shown analogously to Lemma 3.20, using Lemma 7.31. That T or $\bot$ are the smallest terms follows from (O3) of Definition 7.14. Finally, $\mathrm{Q}x.\, t \succ t\{x \mapsto u\}$ follows from (O4) of Definition 7.14. □

Each of the three levels has an entailment relation $\models$. A clause set N_1 entails a clause set N_2, denoted by $N_1 \models N_2$, if all models of N_1 are also a models of N_2. For H and GH, we use higher-order models; for GF, we use first-order models with interpreted Booleans as defined in Section 6.2. We write $N_1 \models_{\mathcal{F}} N_2$ to abbreviate $\mathcal{F}(N_1) \models \mathcal{F}(N_2)$ and similarly for $N_1 \models_{\mathcal{G}} N_2$. On the H level, we additionally define Skolem-aware entailment, denoted by $N_1 \models\!\!\!\approx N_2$, to hold if all Skolem-aware models of a clause set $N_1 \subseteq \mathcal{C}_{\mathrm{H}}$ are also models of a clause set $N_2 \subseteq \mathcal{C}_{\mathrm{H}}$.

Based on these slightly altered definitions, we define clause redundancy verbatim as in Chapters 3 and 5:

- Given $C \in \mathcal{C}_{\mathrm{GF}}$ and $N \subseteq \mathcal{C}_{\mathrm{GF}}$, let $C \in \mathit{GFRed}_{\mathrm{C}}(N)$ if $\{D \in N \mid D \prec C\} \models C$.
- Given $C \in \mathcal{C}_{\mathrm{GH}}$ and $N \subseteq \mathcal{C}_{\mathrm{GH}}$, let $C \in \mathit{GHRed}_{\mathrm{C}}(N)$ if $\mathcal{F}(C) \in \mathit{GFRed}_{\mathrm{C}}(\mathcal{F}(N))$.
- Given $C \in \mathcal{C}_{\mathrm{H}}$ and $N \subseteq \mathcal{C}_{\mathrm{H}}$, let $C \in \mathit{HRed}_{\mathrm{C}}(N)$ if for every $D \in \mathcal{G}(C)$, we have $D \in \mathit{GHRed}_{\mathrm{C}}(\mathcal{G}(N))$ or there exists $C' \in N$ such that $C \sqsupset C'$ and $D \in \mathcal{G}(C')$.

The tiebreaker $\sqsupset$ can be an arbitrary well-founded partial order on $\mathcal{C}_{\mathrm{H}}$, natural candidates being restrictions of (ill-founded) strict subsumption as explained in Section 5.3.4. The redundancy criterion on GF coincides with the one in Chapter 6.

Each of the three levels has an associated inference system *HInf*, *GHInf*, and *GFInf*. For H, it is the inference system *HInf* consisting of the rules described above. We view axioms (Ext) and (Choice) as premise-less inference rules Ext and Choice, respectively. We fix the selection functions *HLitSel* and *HBoolSel* globally.

The system *GHInf* is parameterized by selection functions and a witness function, which are defined as follows.

Definition 7.33 (GH level selection functions)**.** A GH level literal selection *GHLitSel* maps each clause $C \in \mathcal{C}_{\mathrm{GH}}$ to a subset of its literals. A GH level Boolean subterm selection function *GHBoolSel* maps each clause $C \in \mathcal{C}_{\mathrm{GH}}$ to a subset of its green positions with Boolean subterms. We require these selection functions to have the

property that for every $C \in \mathcal{C}_{\mathrm{GH}}$, there exists a $D \in \mathcal{C}_{\mathrm{H}}$ with $C \in \mathcal{G}(D)$ for which the selections $\mathit{HLitSel}(D)$, $\mathit{HBoolSel}(D)$ and the selections $\mathit{GHLitSel}(C)$, $\mathit{GHBoolSel}(C)$ correspond.

Definition 7.34 (Witness function)**.** A witness function GHWit maps a clause $C \in \mathcal{C}_{\mathrm{GH}}$ and a green position of a quantifier-headed term in C to a term $\mathit{GHWit}(C,p) \in \mathcal{T}_{\mathrm{GH}}$ such that $\mathsf{Q}\langle\tau\rangle t \succ t\,\mathit{GHWit}(C,p)$ if $C|_p = \mathsf{Q}\langle\tau\rangle t$.

The witness function will be used to provide appropriate Skolem terms that witness the existence of terms fulfilling the given formula.

Definition 7.35 (Set of parameter triples Q)**.** Let Q be the set of parameters triples $(\mathit{GHLitSel}, \mathit{GHBoolSel}, \mathit{GHWit})$ where $\mathit{GHLitSel}$ and $\mathit{GHBoolSel}$ are GH level selection functions and GHWit is a witness function.

We write GHInf^q with $q = (\mathit{GHLitSel}, \mathit{GHBoolSel}, \mathit{GHWit}) \in Q$ to specify the inference system for a given set of parameters. The rules of GHInf^q include Sup, ERes, EFact, BoolHoist, FalseElim, EqHoist, NeqHoist, and BoolRw with the restriction that all references to $\succsim$ are replaced by $\succeq$.

In addition, GHInf^q contains the rules GForallHoist, GExistsHoist, GArgCong, GExt, and GChoice, which enumerate ground terms in the conclusion where their HInf counterparts use fresh variables. They enumerate all terms $u \in \mathcal{T}_{\mathrm{GH}}$ such that the only Boolean green subterms of u are T and $\bot$. Let $\mathcal{T}^\star_{\mathrm{GH}}$ be the set of all such terms u. Then these rules are stated as follows:

$$\frac{C\langle\forall\langle\tau\rangle v\rangle_p}{C\langle\bot\rangle \vee v\,u \approx \mathsf{T}}\,\textsc{GForallHoist} \qquad \frac{C\langle\exists\langle\tau\rangle v\rangle_p}{C\langle\mathsf{T}\rangle \vee v\,u \approx \bot}\,\textsc{GExistsHoist}$$

where p is $\succeq$-eligible in C and $u \in \mathcal{T}^\star_{\mathrm{GH}}$.

$$\frac{C' \vee t \approx s}{C' \vee t\,\bar{u} \approx s\,\bar{u}}\,\textsc{GArgCong}$$

where $t \approx s$ is strictly $\succeq$-eligible in $C' \vee t \approx s$ and $u_i \in \mathcal{T}^\star_{\mathrm{GH}}$.

The rules GExt and GChoice are premise-free and their conclusions are the infinitely many $\mathcal{G}$-instances of (Ext) and (Choice), respectively.

Moreover, GHInf^q contains the following two rules, which use the witness function GHWit instead of Skolem terms:

$$\frac{C\langle\forall\langle\tau\rangle v\rangle_p}{C\langle v\,\mathit{GHWit}(C,p)\rangle_p}\,\textsc{GForallRw} \qquad \frac{C\langle\exists\langle\tau\rangle v\rangle_p}{C\langle v\,\mathit{GHWit}(C,p)\rangle_p}\,\textsc{GExistsRw}$$

where the p is $\succeq$-eligible in C, for GForallRw $\mathcal{F}(C\langle\mathsf{T}\rangle_p)$ is not a tautology, and for GExistsRw $\mathcal{F}(C\langle\bot\rangle_p)$ is not a tautology.

The inference systems GHInf^q are indeed inference systems on $\mathcal{C}_{\mathrm{GH}}$—i.e., if the premises are in $\mathcal{C}_{\mathrm{GH}}$, the conclusions are in $\mathcal{C}_{\mathrm{GH}}$, too. The conclusions are obviously ground. They are also $\mathsf{Q}_{\approx}$-normal:

Lemma 7.36. *If the premises of an GHInf^q inference are $\mathsf{Q}_{\approx}$-normal, then the conclusion is also $\mathsf{Q}_{\approx}$-normal.*

Proof. The conclusions of GEXT and GCHOICE are $\mathsf{Q}_{\approx}$-normal by the definition of $\mathcal{G}$.

The definition of $\mathsf{Q}_{\approx}$-normality clearly only depends on the contained quantifier-headed subterms. As long as no new quantifier-headed subterms are added, a clause set cannot become $\mathsf{Q}_{\approx}$-reducible.

The inference rules ERES, EFACT, and FALSEELIM do not introduce any subterms that were not already present in the premises. The inference rules SUP, BOOLHOIST, EQHOIST, NEQHOIST, BOOLRW only introduce new subterms by replacing a green subterm of a $\mathsf{Q}_{\approx}$-normal term by another $\mathsf{Q}_{\approx}$-normal term. Since green positions are never below quantifiers, these rules also do not add new quantifier-headed subterms.

For the inference rules GFORALLHOIST, GEXISTSHOIST, GARGCONG, GFORALLRW, and GEXISTSRW, we can use Lemma 7.13 to show that the conclusions are $\mathsf{Q}_{\approx}$-normal. □

The system *GFInf* is parameterized by an analogous triple (*GFLitSel*, *GFBoolSel*, *GFWit*). Using the bijection $\mathcal{F}$, we can translate a parameter triple q of *GHInf* to a parameter triple $\mathcal{F}(q)$ of *GFInf*. Let $\mathit{GFInf}^{\mathcal{F}(q)}$ be the inference system containing the inferences isomorphic to GHInf^q obtained by $\mathcal{F}$, except for GARGCONG, GEXT, and GCHOICE. This is identical to the ground inference system defined in Section 6.3.2.

We extend the functions $\mathcal{F}$ and $\mathcal{G}$ to inferences:

Notation 7.37. Given an inference ι, we write $\mathit{prems}(\iota)$ for the tuple of premises, $\mathit{mprem}(\iota)$ for the main (i.e., rightmost) premise, and $\mathit{concl}(\iota)$ for the conclusion.

7

Definition 7.38 (The encoding $\mathcal{F}$ on inferences)**.** Given an inference $\iota \in \mathit{GHInf}$ that is not a GARGCONG, GEXT, or GCHOICE inference, let $\mathcal{F}(\iota) \in \mathit{GFInf}$ denote the inference defined by $\mathit{prems}(\mathcal{F}(\iota)) = \mathcal{F}(\mathit{prems}(\iota))$ and $\mathit{concl}(\mathcal{F}(\iota)) = \mathcal{F}(\mathit{concl}(\iota))$.

Definition 7.39 (The grounding function $\mathcal{G}$ on inferences)**.** Given a parameter triple $q \in Q$ and an inference $\iota \in \mathit{HInf}$, we define the set $\mathcal{G}^q(\iota)$ of ground instances of ι to be all inferences $\iota' \in \mathit{GHInf}^q$ such that $\mathit{prems}(\iota') = \mathit{prems}(\iota)\theta$ and $\mathit{concl}(\iota') = \mathit{concl}(\iota)\theta$ for some grounding substitution θ.

Thus, $\mathcal{G}$ maps FLUIDSUP to SUP, FLUIDBOOLHOIST to BOOLHOIST, FORALLRW to GFORALLRW, EXISTSRW to GEXISTSRW, FORALLHOIST to GFORALLHOIST, EXISTSHOIST to GEXISTSHOIST, ARGCONG to GARGCONG, EXT to GEXT, CHOICE to GCHOICE, and inferences of other *HInf* rules to inferences of the identically named rules in *GHInf*. For FLUIDLOOBHOIST, which needs not be grounded, we let $\mathcal{G}^q(\iota) = \mathit{undef}$.

We define the sets of redundant inferences w.r.t. a given clause set as follows:

- Given $\iota \in \mathit{GFInf}^q$ and $N \subseteq \mathcal{C}_{\mathrm{GF}}$, let $\iota \in \mathit{GFRed}_{\mathrm{I}}^q(N)$ if $\mathit{prems}(\iota) \cap \mathit{GFRed}_{\mathrm{C}}(N) \neq \varnothing$ or $\{D \in N \mid D \prec \mathit{mprem}(\iota)\} \models \mathit{concl}(\iota)$.
- Given $\iota \in \mathit{GHInf}^q$ and $N \subseteq \mathcal{C}_{\mathrm{GH}}$, let $\iota \in \mathit{GHRed}_{\mathrm{I}}^q(N)$ if
  - ι is GARGCONG, GEXT, or GCHOICE and $\mathit{concl}(\iota) \in N \cup \mathit{GHRed}_{\mathrm{C}}(N)$; or
  - ι is any another inference and $\mathcal{F}(\iota) \in \mathit{GFRed}_{\mathrm{I}}^{\mathcal{F}(q)}(\mathcal{F}(N))$.
- Given $\iota \in \mathit{HInf}$ and $N \subseteq \mathcal{C}_{\mathrm{H}}$, let $\iota \in \mathit{HRed}_{\mathrm{I}}(N)$ if

– ι is a FLUIDLOOBHOIST inference and we have $\mathcal{G}(\mathit{concl}(\iota)) \subseteq \mathcal{G}(N) \cup \mathit{GHRed}_{\mathrm{C}}(\mathcal{G}(N))$; or
– ι is any other inference and $\mathcal{G}^q(\iota) \subseteq \mathit{GHRed}_{\mathrm{I}}(\mathcal{G}(N))$ for all $q \in \boldsymbol{Q}$.

The redundancy criterion on GF coincides with the one in Chapter 6. Some authors prefer not to define inferences with a redundant premise as redundant, but in our proof of refutational completeness, this will be crucial for the lifting lemma of FORALLRW and EXISTSRW.

As in previous chapters, a clause set N is *saturated* w.r.t. an inference system and the inference component $\mathit{Red}_{\mathrm{I}}$ of a redundancy criterion if every inference from clauses in N is in $\mathit{Red}_{\mathrm{I}}(N)$.

7.3.7. Simplification Rules

The redundancy criterion $(\mathit{HRed}_{\mathrm{I}}, \mathit{HRed}_{\mathrm{C}})$ supports the simplification rules implemented in Schulz's first-order prover E [117, Sections 2.3.1 and 2.3.2] to the extend described in Section 3.3.5, based on this chapter's notion of green subterms. In addition to green subterms, the simplification rules can also be applied to subterms below quantifiers that correspond to first-order subterms via the $\mathcal{F}$-encoding—e.g., the subterms $\mathsf{p}\,\mathsf{a}$ and a of $\mathsf{\forall}\langle\tau\rangle(\lambda x.\, \mathsf{p}\,\mathsf{a})$.

Under some circumstances, certain inference rules of our calculus can be applied as simplifications—i.e., a premise can be deleted after performing them. The FALSE-ELIM and BOOLRW rules can be applied as a simplification if σ is the identity. If the head of u is $\mathsf{\forall}$, FORALLHOIST and FORALLRW can be applied together as a simplification rule. The same holds for EXISTSHOIST and EXISTSRW if the head of u is $\mathsf{\exists}$. For all of these simplifications, the eligibility conditions can be ignored.

If σ is the identity, the rule BOOLHOIST can also be applied as a simplification in combination with the following rule to the same subterm u:

$$\frac{C\langle u\rangle}{C\langle \mathsf{T}\rangle \vee u \approx \bot}\,\text{LOOBHOIST}$$

Again, the eligibility condition can be ignored, and the head of u can even be a fully applied logical symbol as long as it is not T or $\bot$.

The proofs justifying all of these simplifications work essentially as exemplified in Lemma 3.24.

7.3.8. A Concrete Term Order

We will define a concrete order $\succ_\lambda$ that fulfills the properties of a strict term order as defined in Definition 7.15 to show that the requirements can indeed be fulfilled and to provide a concrete order for implementations of our calculus.

Given a signature $(\Sigma_{\mathsf{ty}}, \Sigma)$, we encode types and terms as terms over the untyped first-order signature $\Sigma_{\mathsf{ty}} \uplus \{\mathsf{f}_k \mid \mathsf{f} \in \Sigma, k \in \mathbb{N}\} \uplus \{\mathsf{lam}, \mathsf{\forall}'_1, \mathsf{\exists}'_1\} \uplus \{\mathsf{db}^i_k \mid i, k \in \mathbb{N}\}$. We define the encoding in two parts. The first part is the encoding O, resembling the one in Section 5.3.5. The auxiliary function $\mathcal{B}_x(t)$ replaces each free occurrence of the variable x by a De Bruijn index—that is, a symbol db^i where i is the number of λ-expressions surrounding the variable occurrence. The encoding O recursively

encodes higher-order types into untyped first-order terms as follows: $\mathcal{O}(\alpha) = \alpha$ and $\mathcal{O}(\kappa(\bar{\tau})) = \kappa(\mathcal{O}(\bar{\tau}))$. Using $\beta\eta\mathsf{Q}_\eta$-normal representatives, it recursively encodes higher-order terms into untyped first-order terms as follows:

$$\mathcal{O}(t) = \begin{cases} z_t & \text{if } t = x \text{ or } t \text{ is fluid} \\ \mathsf{lam}(\mathcal{O}(\tau), \mathcal{O}(\mathcal{B}_x(u))) & \text{if } t = (\lambda x\!:\!\tau.\, u) \text{ and } t \text{ is not fluid} \\ \mathsf{f}_k(\mathcal{O}(\bar{\tau}), \mathcal{O}(\bar{u}_k)) & \text{if } t = \mathsf{f}\langle\bar{\tau}\rangle\,\bar{u}_k \text{ and either } \mathsf{f} \notin \{\forall, \exists\} \text{ or } k = 0 \\ \mathsf{Q}_1(\mathcal{O}(\tau), \mathcal{O}(\mathcal{B}_x(u))) & \text{if } t = \mathsf{Q}\langle\tau\rangle(\lambda x\!:\!\tau.\, u) \end{cases}$$

Via this encoding, the term order conditions (O1), (O2), and (O3) can be easily achieved. For (O4), however, we need to transform the encoded term further to ensure that the symbols $\forall_1$ and $\exists_1$ occur only as translations of fully applied quantifiers in green contexts. Then we can achieve (O4) by assigning them a large weight. The function $\mathcal{P}$ transforms the result of $\mathcal{O}$ in this way by applying a function p to all subterms below lam symbols.

$$\mathcal{P}(t) = \begin{cases} \alpha & \text{if } t = \alpha \\ z_u & \text{if } t = z_u \\ \mathsf{lam}(\mathcal{P}(\tau), p(u)) & \text{if } t = \mathsf{lam}(\tau, u) \\ \mathsf{f}(\mathcal{P}(\bar{t})) & \text{if } t = \mathsf{f}(\bar{t}) \text{ and } \mathsf{f} \neq \mathsf{lam} \end{cases}$$

The function p replaces $\forall_1$ by $\forall_1'$, $\exists_1$ by $\exists_1'$, and z_u by a fresh variable z_u'.

$$p(t) = \begin{cases} \alpha & \text{if } t = \alpha \\ z_u' & \text{if } t = z_u \\ \forall_1'(p(\tau), p(u)) & \text{if } t = \forall_1(\tau, u) \\ \exists_1'(p(\tau), p(u)) & \text{if } t = \exists_1(\tau, u) \\ \mathsf{f}(p(\bar{t})) & \text{if } t = \mathsf{f}(\bar{t}) \text{ and } \mathsf{f} \notin \{\forall_1, \exists_1\} \end{cases}$$

For example, $\mathcal{O}$ encodes the term $\forall\langle\iota\rangle(\lambda x.\mathsf{p}\,y\,y(\lambda u.\mathsf{f}\,y\,y(\forall\langle\iota\rangle(\lambda v.u))))$ into the first-order term $\forall_1(\iota, \mathsf{p}_3(z_y, z_y, \mathsf{lam}(o, \mathsf{f}_3(z_y, z_y, \forall_1(\iota, \mathsf{db}^1)))))$ and $\mathcal{P}$ transforms it into $\forall_1(\iota, \mathsf{p}_3(z_y, z_y, \mathsf{lam}(o, \mathsf{f}_3(z_y', z_y', \forall_1'(\iota, \mathsf{db}^1)))))$.

Using the encoding $\mathcal{O}$ and the function $\mathcal{P}$, we define our term order $\succ_\lambda$. Let $\succ_{\mathsf{kb}}$ be the transfinite Knuth–Bendix order [104] on first-order terms. The weight of $\forall_1$ and $\exists_1$ must be ω, the weight of $\top$ and $\bot$ must be 1, and the weights of all other symbols must be smaller than ω. The precedence $>$ must be total and $\bot_0 > \top_0$ must be the symbols of lowest precedence. We do not use subterm coefficients (i.e., all coefficients are 1), nor a symbol of weight 0. The precedence function is arbitrary. Let $\succ_{\mathcal{P}}$ be the order induced by $\mathcal{P}$ from $\succ_{\mathsf{kb}}$, meaning $t \succ_{\mathcal{P}} s$ if and only if $\mathcal{P}(t) \succ_{\mathsf{kb}} \mathcal{P}(s)$. Let $\succ_\lambda$ be the order induced by $\mathcal{O}$ from $\succ_{\mathcal{P}}$, meaning $t \succ_\lambda s$ if and only if $\mathcal{O}(t) \succ_{\mathcal{P}} \mathcal{O}(s)$. We extend $\succ_\lambda$ to literals and clauses in the usual way. We will show that $\succ_\lambda$ fulfills the properties of a strict term order:

Lemma 7.40. *The restriction of $\succ_\lambda$ to ground terms is a strict ground term order, as defined in Definition 7.14.*

Proof. We follow the proof of Lemma 5.30.

The transfinite Knuth–Bendix order $\succ_{\mathsf{kb}}$ has been shown to enjoy irreflexivity, transitivity, well-foundedness, totality on ground terms, the subterm property, and compatibility with contexts [104].

Transitivity and irreflexivity of $\succ_{\mathsf{kb}}$ imply transitivity and irreflexivity of $\succ_\lambda$, respectively.

WELL-FOUNDEDNESS: If there existed an infinite chain $t_1 \succ_\lambda t_2 \succ_\lambda \cdots$ of ground terms, there would also be the chain $\mathcal{P}(\mathcal{O}(t_1)) \succ_{\mathsf{kb}} \mathcal{P}(\mathcal{O}(t_2)) \succ_{\mathsf{kb}} \cdots$, contradicting the well-foundedness of $\succ_{\mathsf{kb}}$.

TOTALITY: For any ground terms t and s we have $\mathcal{P}(\mathcal{O}(t)) \succ_{\mathsf{kb}} \mathcal{P}(\mathcal{O}(s))$, $\mathcal{P}(\mathcal{O}(t)) \prec_{\mathsf{kb}} \mathcal{P}(\mathcal{O}(s))$, or $\mathcal{P}(\mathcal{O}(t)) = \mathcal{P}(\mathcal{O}(s))$ by ground totality of $\succ_{\mathsf{kb}}$. In the first two cases, it follows that $t \succ_\lambda s$ or $t \prec_\lambda s$. In the last case, it follows that $t = s$ because $\mathcal{O}$ and $\mathcal{P}$ are clearly injective.

(O3): Since we do not have a symbol of weight 0, and T_0 and $\bot_0$ have weight 1, there cannot be any term of smaller weight. Since moreover T_0 and $\bot_0$ have the lowest precedence, they are the smallest terms w.r.t. $\succ_{\mathsf{kb}}$. We have $\bot_0 \succ_{\mathsf{kb}} \mathsf{T}_0$ because $\bot_0$ has higher precedence. Since $\mathcal{P}(\mathcal{O}(\mathsf{T})) = \mathsf{T}_0$ and $\mathcal{P}(\mathcal{O}(\mathsf{T})) = \bot_0$, it follows that T and $\bot$ are the smallest ground terms w.r.t. $\succ_\lambda$ and that $\bot \succ_\lambda \mathsf{T}$.

(O4): Let $\mathsf{Q}\langle\tau\rangle t$ and u be $\mathsf{Q}_\approx$-normal ground terms. We assume that the Boolean green subterms of u are T and $\bot$. We must show $\mathsf{Q}\langle\tau\rangle t \succ_\lambda t\, u$, which is equivalent to $\mathcal{P}(\mathcal{O}(\mathsf{Q}\langle\tau\rangle t)) \succ_{\mathsf{kb}} \mathcal{P}(\mathcal{O}(t\, u))$.

All symbols except $\forall_1$ and $\exists_1$ have finite weight. Only $\forall_1$ and $\exists_1$ have weight ω. Since all subterm coefficients are 1, the coefficient of ω in the weight of a given term indicates the number of occurrences of the symbols $\forall_1$ and $\exists_1$ in that term.

In η-expanded form, we have $t = \lambda x.\, s$ for some s. Then we have $\mathcal{P}(\mathcal{O}(\mathsf{Q}\langle\tau\rangle t)) = \mathsf{Q}_1(\mathcal{P}(\mathcal{O}(\tau)), \mathcal{P}(\mathcal{O}(\mathcal{B}_x(s))))$ and $\mathcal{P}(\mathcal{O}(t\, u)) = \mathcal{P}(\mathcal{O}(s\{x \mapsto u\}))$. By $\mathsf{Q}_\approx$-normality, x occurs only in green positions of s. Therefore, replacing x by u in s does not trigger any $\beta\eta\mathsf{Q}_\eta$-normalizations. Thus, $\mathcal{P}(\mathcal{O}(\mathcal{B}_x(s)))$ and $\mathcal{P}(\mathcal{O}(s\{x \mapsto u\}))$ are almost identical, except that $\mathcal{P}(\mathcal{O}(\mathcal{B}_x(s)))$ contains db^i where $\mathcal{P}(\mathcal{O}(s\{x \mapsto u\}))$ contains $\mathcal{P}(\mathcal{O}(u))$. Since the only Boolean green subterms of u are T and $\bot$, $\mathcal{P}(\mathcal{O}(u))$ does not contain $\forall_1$ or $\exists_1$. So $\mathcal{P}(\mathcal{O}(\mathcal{B}_x(s)))$ and $\mathcal{P}(\mathcal{O}(s\{x \mapsto u\}))$ contain the same number of $\forall_1$ and $\exists_1$ symbols. Hence, $\mathcal{P}(\mathcal{O}(\mathsf{Q}\langle\tau\rangle t))$ contains exactly one more of these symbols than $\mathcal{P}(\mathcal{O}(t\, u))$. This means that the weight of the former is larger than the weight of the latter and thus $\mathcal{P}(\mathcal{O}(\mathsf{Q}\langle\tau\rangle t)) \succ_{\mathsf{kb}} \mathcal{P}(\mathcal{O}(t\, u))$.

(O2): Let s be a term. We show that $s \succeq_\lambda s|_p$ by induction on p, where $s|_p$ denotes the green subterm at position p. If $p = \varepsilon$, this is trivial. If $p = p'.i$, we have $s \succeq_\lambda s|_{p'}$ by the induction hypothesis. Hence, it suffices to show that $s|_{p'} \succeq_\lambda s|_{p'.i}$. From the existence of the position $p'.i$, we know that $s|_{p'}$ must be of the form $s|_{p'} = \mathsf{f}\langle\bar\tau\rangle\, \bar u_k$ for some $\mathsf{f} \in \Sigma \setminus \{\forall, \exists\}$. Then $s|_{p'.i} = u_i$. The encoding yields $\mathcal{P}(\mathcal{O}(s|_{p'})) = \mathcal{P}(\mathsf{f}_k(\mathcal{O}(\bar\tau), \mathcal{O}(\bar u_k))) = \mathsf{f}_k(\mathcal{P}(\mathcal{O}(\bar\tau)), \mathcal{P}(\mathcal{O}(\bar u_k)))$ and hence $\mathcal{P}(\mathcal{O}(s|_{p'})) \succeq_{\mathsf{kb}} \mathcal{P}(\mathcal{O}(s|_{p'.i}))$ by the subterm property of $\succ_{\mathsf{kb}}$. Hence, $s|_{p'} \succeq_\lambda s|_{p'.i}$ and thus $s \succeq_\lambda s|_p$.

(O1): By induction on the depth of the context, it suffices to show that $t \succ_\lambda s$ implies $\mathsf{f}\langle\bar\tau\rangle\, \bar u\, t\, \bar v \succ_\lambda \mathsf{f}\langle\bar\tau\rangle\, \bar u\, s\, \bar v$ for all t, s, $\mathsf{f} \in \Sigma \setminus \{\forall, \exists\}$, $\bar\tau$, $\bar u$, and $\bar v$. This amounts to showing

that $\mathcal{P}(\mathcal{O}(t)) \succ_{\mathsf{kb}} \mathcal{P}(\mathcal{O}(s))$ implies

$$\begin{aligned}&\mathcal{P}(\mathcal{O}(\mathsf{f}\langle\bar\tau\rangle\,\bar u\,t\,\bar v)) = \mathcal{P}(\mathsf{f}_k(\mathcal{O}(\bar\tau),\mathcal{O}(\bar u),\mathcal{O}(t),\mathcal{O}(\bar v))) =\\ &\mathsf{f}_k(\mathcal{P}(\mathcal{O}(\bar\tau)),\mathcal{P}(\mathcal{O}(\bar u)),\mathcal{P}(\mathcal{O}(t)),\mathcal{P}(\mathcal{O}(\bar v))) \succ_{\mathsf{kb}} \mathsf{f}_k(\mathcal{P}(\mathcal{O}(\bar\tau)),\mathcal{P}(\mathcal{O}(\bar u)),\mathcal{P}(\mathcal{O}(s)),\mathcal{P}(\mathcal{O}(\bar v)))\\ &= \mathcal{P}(\mathsf{f}_k(\mathcal{O}(\bar\tau),\mathcal{O}(\bar u),\mathcal{O}(s),\mathcal{O}(\bar v))) = \mathcal{P}(\mathcal{O}(\mathsf{f}\langle\bar\tau\rangle\,\bar u\,s\,\bar v))\end{aligned}$$

which follows directly from compatibility of $\succ_{\mathsf{kb}}$ with contexts and the induction hypothesis. □

Lemma 7.41. *The relation $\succ_\lambda$ is a strict term order as defined in Definition 7.15.*

Proof. Given Lemma 7.40, it remains to show that $\succ_\lambda$ is stable under grounding substitutions. Assume $s \succ_\lambda s'$ for some terms s and s'. Let θ be a higher-order substitution grounding s and s'. We must show $s\theta \succ_\lambda s'\theta$. We will define a first-order substitution ρ grounding $\mathcal{P}(\mathcal{O}(s))$ and $\mathcal{P}(\mathcal{O}(s'))$ such that $\mathcal{P}(\mathcal{O}(s))\rho = \mathcal{P}(\mathcal{O}(s\theta))$ and $\mathcal{P}(\mathcal{O}(s'))\rho = \mathcal{P}(\mathcal{O}(s'\theta))$. Since $s \succ_\lambda s'$, we have $\mathcal{P}(\mathcal{O}(s)) \succ_{\mathsf{kb}} \mathcal{P}(\mathcal{O}(s'))$. The transfinite Knuth–Bendix order $\succ_{\mathsf{kb}}$ has been shown to be stable under substitutions [104]. Hence, $\mathcal{P}(\mathcal{O}(s))\rho \succ_{\mathsf{kb}} \mathcal{P}(\mathcal{O}(s'))\rho$ and therefore $\mathcal{P}(\mathcal{O}(s\theta)) \succ_{\mathsf{kb}} \mathcal{P}(\mathcal{O}(s'\theta))$ and $s\theta \succ_\lambda s'\theta$.

We define the first-order substitution ρ as $\alpha\rho = \alpha\theta$ for type variables α, $z_u\rho = \mathcal{P}(\mathcal{O}(u\theta))$, and $z'_u\rho = p(\mathcal{O}(u\theta))$ for terms u. Strictly speaking, the domain of a substitution must be finite, so we restrict this definition of ρ to the finitely many variables that occur in the computation of $\mathcal{P}(\mathcal{O}(s))$ and $\mathcal{P}(\mathcal{O}(s'))$.

Clearly, we have $\mathcal{P}(\mathcal{O}(\tau))\rho = \mathcal{P}(\mathcal{O}(\tau\theta))$ and $p(\mathcal{O}(\tau))\rho = p(\mathcal{O}(\tau\theta))$ for all types τ occurring in the computation of $\mathcal{P}(\mathcal{O}(s))$ and $\mathcal{P}(\mathcal{O}(s'))$. Moreover, $\mathcal{P}(\mathcal{O}(t))\rho = \mathcal{P}(\mathcal{O}(t\theta))$ and $p(\mathcal{O}(t))\rho = p(\mathcal{O}(t\theta))$ for all terms t occurring in the computation of $\mathcal{P}(\mathcal{O}(s))$ and $\mathcal{P}(\mathcal{O}(s'))$, which we show by induction on the structure of t.

If $t = x$ or if t is fluid, $\mathcal{P}(\mathcal{O}(t))\rho = z_t\rho = \mathcal{P}(\mathcal{O}(t\theta))$ and $p(\mathcal{O}(t))\rho = z'_t\rho = p(\mathcal{O}(t\theta))$.

If $t = \mathsf{f}\langle\bar\tau\rangle\,\bar u$ for $\mathsf{f} \notin \{\forall, \exists\}$, then

$$\begin{aligned}\mathcal{P}(\mathcal{O}(t))\rho &= \mathsf{f}_k(\mathcal{P}(\mathcal{O}(\bar\tau))\rho, \mathcal{P}(\mathcal{O}(\bar u))\rho)\\ &\overset{\text{IH}}{=} \mathsf{f}_k(\mathcal{P}(\mathcal{O}(\bar\tau\theta)),\mathcal{P}(\mathcal{O}(\bar u\theta))) = \mathcal{P}(\mathcal{O}(\mathsf{f}\langle\bar\tau\theta\rangle(\bar u\theta))) = \mathcal{P}(\mathcal{O}(t\theta))\end{aligned}$$

and analogously for p.

If $t = \mathsf{Q}\langle\tau\rangle(\lambda x.\,u)$, then

$$\begin{aligned}\mathcal{P}(\mathcal{O}(t))\rho &= \mathsf{Q}_1(\mathcal{P}(\mathcal{O}(\tau))\rho, \mathcal{P}(\mathcal{O}(\mathcal{B}_x(u)))\rho)\\ &\overset{\text{IH}}{=} \mathsf{Q}_1(\mathcal{P}(\mathcal{O}(\tau\theta)),\mathcal{P}(\mathcal{O}(\mathcal{B}_x(u)\theta)))\\ &= \mathsf{Q}_1(\mathcal{P}(\mathcal{O}(\tau\theta)),\mathcal{P}(\mathcal{O}(\mathcal{B}_x(u\theta[x \mapsto x]))))\\ &= \mathcal{P}(\mathcal{O}(\mathsf{Q}\langle\tau\rangle(\lambda x.(u\theta[x \mapsto x])))) = \mathcal{P}(\mathcal{O}(\mathsf{Q}\langle\tau\rangle((\lambda x.\,u)\theta))) = \mathcal{P}(\mathcal{O}(t\theta))\end{aligned}$$

and similarly for p, using Q'_1 instead of Q_1.

If $t = (\lambda x\!:\!\tau.\,u)$ and t is not fluid, then

$$\begin{aligned}\mathcal{P}(\mathcal{O}(t))\rho &= \mathsf{lam}(\mathcal{P}(\mathcal{O}(\tau))\rho, p(\mathcal{O}(\mathcal{B}_x(u)))\rho)\\ &\overset{\text{IH}}{=} \mathsf{lam}(\mathcal{P}(\mathcal{O}(\tau\theta)), p(\mathcal{O}(\mathcal{B}_x(u)\theta)))\\ &= \mathsf{lam}(\mathcal{P}(\mathcal{O}(\tau\theta)), p(\mathcal{O}(\mathcal{B}_x(u\theta[x \mapsto x]))))\\ &= \mathcal{P}(\mathcal{O}(\lambda x\!:\!\tau\theta.(u\theta[x \mapsto x]))) = \mathcal{P}(\mathcal{O}((\lambda x\!:\!\tau.\,u)\theta)) = \mathcal{P}(\mathcal{O}(t\theta))\end{aligned}$$

and analogously for p. □

7.4. Refutational Completeness

We present a proof of refutational completeness for our higher-order logic superposition calculus.

We have introduced three different notions of entailment on the H level: $\models_{\mathcal{G}}$, $\models$, and $\models\!\!\approx$. With respect to $\models_{\mathcal{G}}$, static and dynamic completeness hold unconditionally. For the other two notions of entailment, we will need to add an additional precondition that ensure that the initial clause set is $\mathsf{Q}_{\approx}$-normal, which can only be stated for dynamic completeness. For $\models\!\!\approx$, we need to require in addition that the initial clause set does not contain any sk symbols.

7.4.1. Outline of the Proof

Like in Chapters 5 and 6, the proof is done in three steps:

1. We prove static refutational completeness of *GFInf*.
2. We show static refutational completeness of *GHInf* by transforming the model constructed on the GF level into a higher-order interpretation.
3. We employ the saturation framework to lift the result to the H layer.

We have achieved the first step already in Chapter 6. The refutational completeness result holds for any tuple of parameters $q \in \mathcal{F}(Q)$. In addition to the refutational completeness of *GFInf*, the subsequent steps also depend on some properties of the constructed model.

For the second step, we fix a parameter triple $q \in Q$ and a set $N \subseteq \mathcal{C}_{\mathrm{GH}}$ saturated w.r.t. GHInf^q and not containing the empty clause. Then the first step guarantees us a model of $\mathcal{F}(N)$. Based on this model, we construct a higher-order interpretation that we show to be a model of N. In essence, the proof is analogous to the one in Chapter 5, but additionally, we need to consider $\mathsf{Q}_{\approx}$-normality and the logical symbols.

For the third step, the saturation framework leaves to us the proof of the lifting lemmas for our inference rules. For this lifting from GHInf^q to *HInf*, we must choose an appropriate parameter triple $q \in Q$, given a set $N \subseteq \mathcal{C}_{\mathrm{H}}$. In particular, we must specify the witness function to produce Skolem terms according to the given set N.

Once we have fulfilled the requirements of the saturation framework's lifting theorem, it guarantees static refutational completeness w.r.t.$\models_{\mathcal{G}}$. We can then show that this also implies dynamic refutational completeness w.r.t. $\models$ and $\models\!\!\approx$.

7.4.2. The Ground First-Order Level

We have established refutational completeness for the GF level in the previous chapter. Besides refutational completeness, we will need some additional properties of the model R_M^* we have constructed. Concretely, we will need the following lemma:

Lemma 7.42. *Let $N \subseteq \mathcal{C}_{\mathrm{GF}}$. If $C = C' \vee s \approx t \in N$ produces $s \to t$, then $s \approx t$ is strictly $\succeq$-eligible in C and C' is false in R_M^*.*

Proof. The literal $s \approx t$ is $\succeq$-eligible in C by (C3) of Definition 6.12. It is even strictly $\succeq$-eligible by (C6). The subclause C' is false in R_M^* by Lemma 6.17. □

Adapted to the context of this chapter, the completeness theorem (Theorem 6.24) can be restated as follows:

Theorem 7.43 (Ground first-order static refutational completeness)**.** *Let $q \in \mathcal{F}(\mathcal{Q})$. Then GFInf^q is statically refutationally complete w.r.t. $\models$ and $(\mathit{GFRed}_\mathrm{I}, \mathit{GFRed}_\mathrm{C})$. More precisely, if $N \subseteq \mathcal{C}_\mathrm{GF}$ is a clause set saturated w.r.t. GFInf^q and $\mathit{GFRed}_\mathrm{I}^q$ such that $\bot \notin N$, then $R^*_{N \setminus \mathit{GFRed}_\mathrm{C}(N)}$ is a model of N.*

7.4.3. The Ground Higher-Order Level

In this subsection, let $q = (\mathit{GHLitSel}, \mathit{GHBoolSel}, \mathit{GHWit}) \in \mathcal{Q}$ be a parameter triple and let $N \subseteq \mathcal{C}_\mathrm{GH}$. Since all terms on the GH level are $\mathsf{Q}_{\approx}$-normal, in particular N is $\mathsf{Q}_{\approx}$-normal. We assume that N is saturated w.r.t. GHInf^q and $\mathit{GHRed}_\mathrm{I}^q$, and that $\bot \notin N$. Clearly, the set $\mathcal{F}(N)$ is then saturated w.r.t. $\mathit{GFInf}^{\mathcal{F}(\mathit{sel})}$ and $\mathit{GFRed}_\mathrm{I}^{\mathcal{F}(\mathit{sel})}$ and $R^*_{\mathcal{F}(N) \setminus \mathit{GFRed}_\mathrm{C}(N)}$ is a model of $\mathcal{F}(N)$ by Theorem 7.43.

In the following, we abbreviate $R^*_{\mathcal{F}(N) \setminus \mathit{GFRed}_\mathrm{C}(N)}$ as R. Given two terms $s, t \in \mathcal{T}_\mathrm{GH}$, we write $s \sim t$ to abbreviate $R \models \mathcal{F}(s) \approx \mathcal{F}(t)$, which is equivalent to $[\![\mathcal{F}(s)]\!]_R = [\![\mathcal{F}(t)]\!]_R$.

Lemma 7.44. *For all terms $t, s : \tau \to \upsilon$ in $\mathcal{T}_\mathrm{GH}$, these statements are equivalent:*

1. *$t \sim s$;*
2. *$t\,(\mathsf{diff}\,t\,s) \sim s\,(\mathsf{diff}\,t\,s)$;*
3. *$t\,u \sim s\,u$ for all $u \in \mathcal{T}_\mathrm{GH}$.*

Proof. Analogous to Lemma 5.32, using Lemma 7.42 and the $\beta\eta\mathsf{Q}_\eta$-normal form. □

Lemma 7.45. *Let $s \in \mathcal{T}_\mathrm{H}$ and let θ, θ' be grounding substitutions such that $s\theta$ and $s\theta'$ are $\mathsf{Q}_{\approx}$-normal, $x\theta \sim x\theta'$ for all variables x, and $\alpha\theta = \alpha\theta'$ for all type variables α. Then $s\theta \sim s\theta'$.*

Proof. This proof is almost identical to the one of Lemma 5.33. The only difference lies in Case 4.1 that must deal with quantifiers. In Case 4.1 of the proof of Lemma 5.33, we consider a λ-term s that contains exactly one free term-variable that occurs exactly once in s and s is of the form $f\langle\bar{\tau}\rangle\bar{t}$, for some symbol f, some types $\bar{\tau}$, and some λ-terms $\bar{t}$. For any $f \notin \{\forall, \exists\}$ the proof proceeds as before, because then the definition of the encoding $\mathcal{F}$ coincides with the one of Chapter 5. The remaining case to handle is thus when s is of the form $\mathsf{Q}\langle\tau\rangle(\lambda z.\, t)$.

In that case, we have $[\![\mathcal{F}(s\theta)]\!]_R = [\![\mathsf{Q}z.\, \mathcal{F}(t\theta)]\!]_R$ where $\mathsf{Q} \in \{\forall, \exists\}$, but θ and θ' are not necessarily grounding for t since t may contain the variable z that is bound in s. Thus we cannot apply our induction hypothesis directly on t. Instead we want to apply it on $t\{z \mapsto u\}$, which we denote t_u, where $u \in \mathcal{T}_\mathrm{GH}$. It is possible to apply the induction hypothesis to obtain $t_u\theta \sim t_u\theta'$ because

- $n_1(s) = n_1(t_u)$ since all $\beta\oplus$-reductions in s are also in t_u;
- $n_2(s) = 0 = n_2(t_u)$ since the same unique free term variable occurs in s and in t_u; and
- $n_3(t_u) < n_3(s)$ because $\mathcal{S}(z) = \mathcal{S}(u) = 1$ implies $n_3(t_u) = n_3(t)$ and hence $n_3(s) = \mathcal{S}(\mathsf{Q}) + \mathcal{S}(\lambda z.\, t) = 1 + 1 + \mathcal{S}(t) = 1 + 1 + \mathcal{S}(t_u)$.

Moreover, since θ and θ' do not capture z, and since u is $\mathsf{Q}_{\approx}$-normal, $t_u\theta = (t\theta)\{z \mapsto u\}$ is $\mathsf{Q}_{\approx}$-normal by Lemma 7.8 and similarly for θ'. Thus we obtain $(t\theta)\{z \mapsto u\} \sim (t\theta')\{z \mapsto u\}$.

Now, let us consider the case where $\mathsf{Q} = \forall$: By the definition of interpretations on the GF level, by Lemma 6.7 (substitution lemma), and R being term-generated, $[\![\forall z.\, \mathcal{F}(t\theta)]\!]^{\xi}_R = \min\{[\![\mathcal{F}(t\theta)]\!]^{\xi[z\mapsto a]}_R \mid a \in \mathcal{U}_\tau\} = \min\{[\![\mathcal{F}(t\theta)\{z \mapsto v\}]\!]^{\xi}_R \mid v \in \mathcal{T}_{\mathrm{GF}}\} = \min\{[\![\mathcal{F}(t\theta\{z \mapsto u\})]\!]^{\xi}_R \mid u \in \mathcal{T}_{\mathrm{GH}}\}$. The same holds for θ'. Moreover, above we deduced $(t\theta)\{z \mapsto u\} \sim (t\theta')\{z \mapsto u\}$ from the induction hypothesis. Thus, $s\theta \sim s\theta'$, as desired in the case $\mathsf{Q} = \forall$. The case $\mathsf{Q} = \exists$ is analogous, with max instead of min. □

We proceed by defining a higher-order interpretation $\mathcal{I}^{\mathrm{GH}} = (\mathcal{U}^{\mathrm{GH}}, \mathcal{J}^{\mathrm{GH}}_{\mathrm{ty}}, \mathcal{J}^{\mathrm{GH}}, \mathcal{L}^{\mathrm{GH}})$ derived from R. We call this interpretation $\mathcal{I}^{\mathrm{GH}}$ because we use it to show refutational completeness of *GHInf*; it is a higher-order interpretation as defined in Section 7.2, which can interpret ground as well as nonground terms. The construction closely resembles the one in Section 5.4.2, but we need to consider questions of $\mathsf{Q}_{\approx}$-normality and the interpretation of logical symbols. Let $(\mathcal{U}, \mathcal{J}) = R$, meaning that $\mathcal{U}_\tau$ is the universe of R for type τ and $\mathcal{J}$ is the interpretation function of R.

To illustrate the construction, we will employ the following running example. Let $\Sigma_{\mathrm{ty}} = \{\iota, o, \to\}$ and let Σ contain $\mathsf{f} : \iota \to \iota$ and $\mathsf{a} : \iota$, as well as the logical symbols and the choice constant ε. Then, on the GF level, the type signature is also Σ_{ty}, and the term signature is the set Σ_{GF}, which contains f_0, f_1, a_0, subscripted versions of all logical symbols, such as T_0, $\mathsf{\bot}_0$, $\neg_0$, and $\neg_1$, as well as symbols ε^τ_i for each $\tau \in \mathcal{T}y_{\mathrm{GH}}$, and a symbol $\mathsf{lam}_{\lambda x.t}$ for each $\lambda x.\, t \in \mathcal{T}_{\mathrm{GH}}$. We write $[t]$ for the equivalence class of $t \in \mathcal{T}_{\mathrm{GF}}$ modulo R. The universes $\mathcal{U}_\tau$ are sets of such equivalence classes; for instance $[\mathsf{f}_1(\mathsf{a}_0)] \in \mathcal{U}_\iota$, $[\neg_0] \in \mathcal{U}_{o\to o}$, and $[\mathsf{lam}_{\lambda x.\mathsf{a}}] \in \mathcal{U}_{\iota\to\iota}$. We assume that R is such that $[\mathsf{a}_0]$, $[\mathsf{f}_1(\mathsf{a}_0)]$, $[\mathsf{f}_1(\mathsf{f}_1(\mathsf{a}_0))]$, ... are all different from each other, and therefore that $\mathcal{U}_\iota$ is infinite.

When defining the universe $\mathcal{U}^{\mathrm{GH}}$ of the higher-order interpretation, we need to ensure that it contains subsets of function spaces, since $\mathcal{J}^{\mathrm{GH}}_{\mathrm{ty}}(\to)(\mathcal{D}_1, \mathcal{D}_2)$ must be a subset of the function space from $\mathcal{D}_1$ to $\mathcal{D}_2$ for all $\mathcal{D}_1, \mathcal{D}_2 \in \mathcal{U}^{\mathrm{GH}}$. But the first-order universes $\mathcal{U}_\tau$ consist of equivalence classes of terms from $\mathcal{T}_{\mathrm{GF}}$ w.r.t. the rewriting system R, not of functions.

To repair this mismatch, we will define a family of functions $\mathcal{E}_\tau$ that give a meaning to the elements of the first-order universes $\mathcal{U}_\tau$. We will define a domain $\mathcal{D}_\tau$ for each ground type τ and then let $\mathcal{U}^{\mathrm{GH}}$ be the set of all these domains $\mathcal{D}_\tau$. Thus, there will be a one-to-one correspondence between ground types and domains. Since the higher-order and first-order type signatures are identical (including $\to$, which is uninterpreted in GF's logic), we can identify higher-order and first-order types.

We define $\mathcal{E}_\tau$ and $\mathcal{D}_\tau$ in a mutual induction and prove that $\mathcal{E}_\tau$ is a bijection simultaneously. We start with nonfunctional types τ: Let $\mathcal{D}_\tau = \mathcal{U}_\tau$ and let $\mathcal{E}_\tau : \mathcal{U}_\tau \to \mathcal{D}_\tau$ be the identity. We proceed by defining $\mathcal{E}_{\tau\to\upsilon}$ and $\mathcal{D}_{\tau\to\upsilon}$. We assume that $\mathcal{E}_\tau$, $\mathcal{E}_\upsilon$, $\mathcal{D}_\tau$, and $\mathcal{D}_\upsilon$ have already been defined and that $\mathcal{E}_\tau$, $\mathcal{E}_\upsilon$ are bijections. To ensure that $\mathcal{E}_{\tau\to\upsilon}$ will be bijective, we first define an injective function $\mathcal{E}^0_{\tau\to\upsilon} : \mathcal{U}_{\tau\to\upsilon} \to (\mathcal{D}_\tau \to \mathcal{D}_\upsilon)$, define $\mathcal{D}_{\tau\to\upsilon}$ as its image $\mathcal{E}^0_{\tau\to\upsilon}(\mathcal{U}_{\tau\to\upsilon})$, and finally define $\mathcal{E}_{\tau\to\upsilon}$ as $\mathcal{E}^0_{\tau\to\upsilon}$ with its

codomain restricted to $\mathcal{D}_{\tau\to\upsilon}$:

$$\mathcal{E}^0_{\tau\to\upsilon} : \mathcal{U}_{\tau\to\upsilon} \longrightarrow (\mathcal{D}_\tau \longrightarrow \mathcal{D}_\upsilon)$$
$$\mathcal{E}^0_{\tau\to\upsilon}(\llbracket\mathcal{F}(s)\rrbracket_R)\big(\mathcal{E}_\tau\big(\llbracket\mathcal{F}(u)\rrbracket_R\big)\big) = \mathcal{E}_\upsilon\big(\llbracket\mathcal{F}(s\,u)\rrbracket_R\big)$$

This is a valid definition because each element of $\mathcal{U}_{\tau\to\upsilon}$ is of the form $\llbracket\mathcal{F}(s)\rrbracket_R$ for some s and each element of $\mathcal{D}_\tau$ is of the form $\mathcal{E}_\tau\big(\llbracket\mathcal{F}(u)\rrbracket_R\big)$ for some u. This function is well defined if it does not depend on the choice of s and u. To show this, we assume that there are other ground terms t and v such that $\llbracket\mathcal{F}(s)\rrbracket_R = \llbracket\mathcal{F}(t)\rrbracket_R$ and $\mathcal{E}_\tau\big(\llbracket\mathcal{F}(u)\rrbracket_R\big) = \mathcal{E}_\tau\big(\llbracket\mathcal{F}(v)\rrbracket_R\big)$. Since $\mathcal{E}_\tau$ is bijective, we have $\llbracket\mathcal{F}(u)\rrbracket_R = \llbracket\mathcal{F}(v)\rrbracket_R$. Using the $\sim$-notation, we can write this as $u \sim v$. The terms s, t, u, and v are in GH, and thus $\mathsf{Q}_{\approx}$-normal, allowing us to apply Lemma 7.45 to the term $x\,y$ and the substitutions $\{x \mapsto s, y \mapsto u\}$ and $\{x \mapsto t, y \mapsto v\}$. Thus, we obtain $s\,u \sim t\,v$—i.e., $\llbracket\mathcal{F}(s\,u)\rrbracket_R = \llbracket\mathcal{F}(t\,v)\rrbracket_R$, indicating that $\mathcal{E}^0_{\tau\to\upsilon}$ is well defined. It remains to show that $\mathcal{E}^0_{\tau\to\upsilon}$ is injective as a function from $\mathcal{U}_{\tau\to\upsilon}$ to $\mathcal{D}_\tau \longrightarrow \mathcal{D}_\upsilon$. Assume two terms $s, t \in \mathcal{T}_{\mathrm{GH}}$ such that for all $u \in \mathcal{T}_{\mathrm{GH}}$, we have $\llbracket\mathcal{F}(s\,u)\rrbracket_R = \llbracket\mathcal{F}(t\,u)\rrbracket_R$. By Lemma 7.44, it follows that $\llbracket\mathcal{F}(s)\rrbracket_R = \llbracket\mathcal{F}(t)\rrbracket_R$, which concludes the proof that $\mathcal{E}^0_{\tau\to\upsilon}$ is injective.

We define $\mathcal{D}_{\tau\to\upsilon} = \mathcal{E}^0_{\tau\to\upsilon}(\mathcal{U}_{\tau\to\upsilon})$ and $\mathcal{E}_{\tau\to\upsilon}(a) = \mathcal{E}^0_{\tau\to\upsilon}(a)$. This ensures that $\mathcal{E}_{\tau\to\upsilon}$ is bijective and concludes the inductive definition of $\mathcal{D}$ and $\mathcal{E}$. In the following, we will usually write $\mathcal{E}$ instead of $\mathcal{E}_\tau$, since the type τ is determined by the first argument of $\mathcal{E}_\tau$.

7

In our running example, we thus have $\mathcal{D}_\iota = \mathcal{U}_\iota = \{[\mathsf{a}_0], [\mathsf{f}_1(\mathsf{a}_0)], [\mathsf{f}_1(\mathsf{f}_1(\mathsf{a}_0))], \ldots\}$ and $\mathcal{E}_\iota$ is the identity $\mathcal{U}_\iota \to \mathcal{D}_\iota$, $c \mapsto c$. The function $\mathcal{E}^0_{\iota\to\iota}$ maps $[\mathsf{lam}_{\lambda x.x}]$ to the identity $\mathcal{D}_\iota \to \mathcal{D}_\iota$, $c \mapsto c$; it maps $[\mathsf{lam}_{\lambda x.\mathsf{a}}]$ to the constant function $\mathcal{D}_\iota \to \mathcal{D}_\iota$, $c \mapsto [\mathsf{a}_0]$; it maps $[\mathsf{f}_0]$ to the function $\mathcal{D}_\iota \to \mathcal{D}_\iota$, $[t] \mapsto [\mathsf{f}_1(t)]$; and it maps $[\varepsilon_1^{\iota\to\iota}(\mathsf{lam}_{\lambda z.z\,(\mathsf{f}\,\mathsf{a})\approx\mathsf{a}})]$ to the function $\mathcal{D}_\iota \to \mathcal{D}_\iota$, $[t] \mapsto [\varepsilon_2^{\iota\to\iota}(\mathsf{lam}_{\lambda z.z\,(\mathsf{f}\,\mathsf{a})\approx\mathsf{a}}, t)]$. So there are a lot of different functions in $\mathcal{D}_{\iota\to\iota} = \mathcal{E}^0_{\iota\to\iota}(\mathcal{U}_{\iota\to\iota})$, but still $\mathcal{D}_{\iota\to\iota}$ is a proper subset of the function space $\mathcal{U}_\iota \to \mathcal{U}_\iota$ because the function space is uncountably infinite, whereas $\mathcal{T}_{\mathrm{GF}}$ and hence $\mathcal{D}_{\iota\to\iota}$ is countable. Thus, the construction works only because we allow nonstandard Henkin models.

We define the higher-order universe as $\mathcal{U}^{\mathrm{GH}} = \{\mathcal{D}_\tau \mid \tau \text{ ground}\}$. In particular, this implies that $\mathcal{D}_o = \{0,1\} \in \mathcal{U}^{\mathrm{GH}}$ as needed, where 0 is identified with $[\bot_0]$ and 1 with $[\mathsf{T}_0]$. Moreover, we define $\mathcal{J}^{\mathrm{GH}}_{\mathrm{ty}}(\kappa)(\mathcal{D}_{\bar\tau}) = \mathcal{U}_{\kappa(\bar\tau)}$ for all $\kappa \in \Sigma_{\mathrm{ty}}$, completing the type interpretation of $\mathcal{I}^{\mathrm{GH}}_{\mathrm{ty}} = (\mathcal{U}^{\mathrm{GH}}, \mathcal{J}^{\mathrm{GH}}_{\mathrm{ty}})$ and ensuring that $\mathcal{J}^{\mathrm{GH}}_{\mathrm{ty}}(o) = \mathcal{U}_o = \{0,1\}$.

We define the interpretation function $\mathcal{J}^{\mathrm{GH}}$ for non-quantifier symbols $\mathsf{f} : \Pi\bar\alpha_m.\,\tau \notin \{\forall, \exists\}$ by $\mathcal{J}^{\mathrm{GH}}(\mathsf{f}, \mathcal{D}_{\bar\upsilon_m}) = \mathcal{E}(\mathcal{J}(\mathsf{f}_0^{\bar\upsilon_m}))$, and for quantifiers by $\mathcal{J}^{\mathrm{GH}}(\forall, \mathcal{D}_\tau)(f) = \min\{f(a) \mid a \in \mathcal{D}_\tau\}$ and $\mathcal{J}^{\mathrm{GH}}(\exists, \mathcal{D}_\tau)(f) = \max\{f(a) \mid a \in \mathcal{D}_\tau\}$ for all $f \in \mathcal{J}^{\mathrm{GH}}_{\mathrm{ty}}(\to)(\mathcal{D}_\tau, \{0,1\})$.

In our example, we thus have $\mathcal{J}^{\mathrm{GH}}(\mathsf{f}) = \mathcal{E}([\mathsf{f}_0])$, which is the function $[t] \mapsto [\mathsf{f}_1(t)]$.

We must show that this definition indeed fulfills the requirements of an interpretation function. By definition, we have $\mathcal{J}^{\mathrm{GH}}(\mathsf{T}) = \mathcal{E}(\llbracket\mathsf{T}_0\rrbracket_R) = \llbracket\mathsf{T}_0\rrbracket_R = 1$ and $\mathcal{J}^{\mathrm{GH}}(\bot) = \mathcal{E}(\llbracket\bot_0\rrbracket_R) = \llbracket\bot_0\rrbracket_R = 0$.

Let $a, b \in \{0,1\}$, $u_0 = \bot$, and $u_1 = \top$. Then

$$\begin{aligned}
\mathcal{J}^{\mathrm{GH}}(\neg)(a) &= \mathcal{E}(\mathcal{J}(\mathcal{F}(\neg)))([\![\mathcal{F}(u_a)]\!]_R) = \mathcal{E}^0([\![\neg_0]\!]_R)([\![\mathcal{F}(u_a)]\!]_R) \\
&= \mathcal{E}([\![\mathcal{F}(\neg u_a)]\!]_R) = [\![\mathcal{F}(\neg u_a)]\!]_R = 1 - a \\
\mathcal{J}^{\mathrm{GH}}(\wedge)(a,b) &= \mathcal{E}^0([\![\mathcal{F}(\wedge)]\!]_R)([\![\mathcal{F}(u_a)]\!]_R, [\![\mathcal{F}(u_b)]\!]_R) = \mathcal{E}([\![\mathcal{F}(u_a \wedge u_b)]\!]_R) \\
&= \min\{a,b\} \\
\mathcal{J}^{\mathrm{GH}}(\vee)(a,b) &= \mathcal{E}([\![\mathcal{F}(u_a \vee u_b)]\!]_R) = \max\{a,b\} \\
\mathcal{J}^{\mathrm{GH}}(\rightarrow)(a,b) &= \mathcal{E}([\![\mathcal{F}(u_a \rightarrow u_b)]\!]_R) = \max\{1-a,b\}
\end{aligned}$$

Let $\mathcal{D} \in \mathcal{U}^{\mathrm{GH}}$ and $a', b' \in \mathcal{D}$. Since $\mathcal{E}$ is bijective and R is term-generated, there exist ground terms u and v such that $\mathcal{E}([\![\mathcal{F}(u)]\!]_R) = a'$ and $\mathcal{E}([\![\mathcal{F}(v)]\!]_R) = b'$. Then

$$\mathcal{J}^{\mathrm{GH}}(\approx)(a',b') = \mathcal{E}^0([\![\mathcal{F}(\approx)]\!]_R)(\mathcal{E}([\![\mathcal{F}(u)]\!]_R), \mathcal{E}([\![\mathcal{F}(v)]\!]_R)) = \mathcal{E}([\![\mathcal{F}(u \approx v)]\!]_R)$$

which is 1 if $a' = b'$ and 0 otherwise. Similarly $\mathcal{J}^{\mathrm{GH}}(\not\approx)(a',b') = 0$ if $a' = b'$ and 1 otherwise.

The requirements for $\forall$ and $\exists$ hold by definition of $\mathcal{J}^{\mathrm{GH}}$.

Let $\mathcal{D}_\tau \in \mathcal{U}^{\mathrm{GH}}$ and $f \in \mathcal{J}_{\mathrm{ty}}(\rightarrow)(\mathcal{D}_\tau, \{0,1\})$. For the requirement on ε, we must show that $f(\mathcal{J}^{\mathrm{GH}}(\varepsilon\langle\tau\rangle)(f)) = \max\{f(a) \mid a \in \mathcal{D}_\tau\}$.

First, we assume that $f(\mathcal{J}^{\mathrm{GH}}(\varepsilon\langle\tau\rangle)(f)) = 0$. We want to show that $\max\{f(a) \mid a \in \mathcal{D}_\tau\} = 0$. Let $a \in \mathcal{D}_\tau$. We have $f = \mathcal{E}([\![\mathcal{F}(p)]\!]_R)$ for some $p : \tau \rightarrow o$ and $a = \mathcal{E}([\![\mathcal{F}(u)]\!]_R)$ for some $u : \tau \in \mathcal{T}_{\mathrm{GH}}$ because $\mathcal{E}$ and $\mathcal{F}$ are bijective and R is term-generated. Since N is saturated, all conclusions of GCHOICE belong to N. In particular, we have $(p\,u \approx \bot \vee p(\varepsilon\langle\tau\rangle p) \approx \top) \in N$ and hence $[\![\mathcal{F}(p\,u \approx \bot \vee p\,(\varepsilon\langle\tau\rangle\,p) \approx \top)]\!]_R = 1$. Thus, we have $\max\{[\![\mathcal{F}(p\,u \approx \bot)]\!]_R, [\![\mathcal{F}(p\,(\varepsilon\langle\tau\rangle\,p) \approx \top)]\!]_R\} = 1$. Since $f(\mathcal{J}^{\mathrm{GH}}(\varepsilon\langle\tau\rangle)(f)) = [\![\mathcal{F}(p\,(\varepsilon\langle\tau\rangle\,p) \approx \top)]\!]_R$, our assumption implies that $[\![\mathcal{F}(p\,u \approx \bot)]\!]_R = 1$. Moreover,

$$\begin{aligned}
[\![\mathcal{F}(p\,u \approx \bot)]\!]_R = 1 \quad &\Leftrightarrow \quad [\![(\mathcal{F}(p\,u)) \approx \bot_0]\!]_R = 1 \\
&\Leftrightarrow \quad [\![\mathcal{F}(p\,u)]\!]_R = 0 \\
&\Leftrightarrow \quad \mathcal{E}([\![\mathcal{F}(p\,u)]\!]_R) = 0 \\
&\Leftrightarrow \quad \mathcal{E}([\![\mathcal{F}(p)]\!]_R)(\mathcal{E}([\![\mathcal{F}(u)]\!]_R)) = 0 \\
&\Leftrightarrow \quad f(a) = 0
\end{aligned}$$

Since our choice of a was arbitrary, this shows that $\max\{f(a) \mid a \in \mathcal{D}_\tau\} = 0$.

For the other direction, we assume that $\max\{f(a) \mid a \in \mathcal{D}_\tau\} = 0$. Then, by definition of max, and since $\mathcal{J}^{\mathrm{GH}}(\varepsilon\langle\tau\rangle)(f) \in \mathcal{D}_\tau$, we have in particular $f(\mathcal{J}^{\mathrm{GH}}(\varepsilon\langle\tau\rangle)(f)) = 0$. Thus, we have $f(\mathcal{J}^{\mathrm{GH}}(\varepsilon\langle\tau\rangle)(f)) = \max\{f(a) \mid a \in \mathcal{D}_\tau\}$ as required, which concludes the proof that $\mathcal{J}^{\mathrm{GH}}$ is an interpretation function.

Finally, we need to define the designation function $\mathcal{L}^{\mathrm{GH}}$, which takes a valuation ξ and a λ-expression as arguments. Given a valuation ξ, we choose a grounding substitution θ such that $\mathcal{D}_{\alpha\theta} = \xi(\alpha)$ and $\mathcal{E}([\![\mathcal{F}(x\theta)]\!]_R) = \xi(x)$ for all type variables α and all variables x. Such a substitution can be constructed as follows: We can fulfill the first equation in a unique way because there is a one-to-one correspondence between ground types and domains. Since $\mathcal{E}^{-1}(\xi(x))$ is an element of a first-order universe and R is term-generated, there exists a ground term s such that $[\![s]\!]_R^\xi =$

$\mathcal{E}^{-1}(\xi(x))$. Choosing one such t and defining $x\theta = \mathcal{F}^{-1}(s)$ gives us a grounding substitution θ with the desired property.

We define $\mathcal{L}^{\mathrm{GH}}(\xi,(\lambda x.\, t)) = \mathcal{E}([\![\mathcal{F}((\lambda x.\, t)\theta)]\!]_R)$ if $\lambda x.\, t$ is $\mathsf{Q}_{\approx}$-normal, and otherwise $\mathcal{L}^{\mathrm{GH}}(\xi,(\lambda x.\, t)) = \mathcal{L}^{\mathrm{GH}}(\xi,(\lambda x.\, t)\downarrow_{\mathsf{Q}_{\approx}})$. Since $\mathcal{F}$ is only defined on $\mathsf{Q}_{\approx}$-normal terms, we need to show that $(\lambda x.t)\theta$ is $\mathsf{Q}_{\approx}$-normal if $\lambda x.\, t$ is $\mathsf{Q}_{\approx}$-normal. This holds because by construction of θ all $x\theta$ are $\mathsf{Q}_{\approx}$-normal and thus so is $(\lambda x.t)\theta$ according to Lemma 7.8. Moreover we need to show that our definition does not depend on the choice of θ. We assume that there exists another substitution θ' with the properties $\mathcal{D}_{\alpha\theta'} = \xi(\alpha)$ for all α and $\mathcal{E}([\![\mathcal{F}(x\theta')]\!]_R) = \xi(x)$ for all x. Then we have $\alpha\theta = \alpha\theta'$ for all α due to the one-to-one correspondence between domains and ground types. We have $[\![\mathcal{F}(x\theta)]\!]_R = [\![\mathcal{F}(x\theta')]\!]_R$ for all x because $\mathcal{E}$ is injective. By Lemma 7.45 it follows that $[\![\mathcal{F}((\lambda x.\, t)\theta)]\!]_R = [\![\mathcal{F}((\lambda x.\, t)\theta')]\!]_R$, which proves that $\mathcal{L}^{\mathrm{GH}}$ is well defined.

In our running example, for all ξ we have $\mathcal{L}^{\mathrm{GH}}(\xi, \lambda x.\, x) = \mathcal{E}([\mathsf{lam}_{\lambda x.\, x}])$, which is the identity. If $\xi(y) = [\mathsf{a}_0]$, then $\mathcal{L}^{\mathrm{GH}}(\xi, \lambda x.\, y) = \mathcal{E}([\mathsf{lam}_{\lambda x.\, \mathsf{a}}])$, which is the constant function $c \mapsto [\mathsf{a}_0]$.

This concludes the definition of the interpretation $\mathcal{I}^{\mathrm{GH}} = (\mathcal{U}^{\mathrm{GH}}, \mathcal{J}^{\mathrm{GH}}_{\mathsf{ty}}, \mathcal{J}^{\mathrm{GH}}, \mathcal{L}^{\mathrm{GH}})$. It remains to show that $\mathcal{I}^{\mathrm{GH}}$ is proper. We need a lemma:

Lemma 7.46. *Let $\mathcal{I}$ be an interpretation such that $[\![\lambda x.\, t]\!]^{\xi}_{\mathcal{I}}(a) = [\![t\downarrow_{\mathsf{Q}_{\approx}}]\!]^{\xi[x\mapsto a]}_{\mathcal{I}}$ for all λ-terms t and all valuations ξ. Then $\mathsf{Q}_{\approx}$-normalization preserves denotations of terms and truth of clauses w.r.t. $\mathcal{I}$.*

Proof. We must show $[\![t]\!]^{\xi}_{\mathcal{I}} = [\![t\downarrow_{\mathsf{Q}_{\approx}}]\!]^{\xi}_{\mathcal{I}}$ for all λ-terms t and all valuations ξ. We cannot work with $\beta\eta$-equivalence classes here because that would require the interpretation to be proper and thus result in a circular argument. We proceed by induction on the structure of t.

If $t = \lambda x.\, u$, by applying our assumption on $\mathcal{I}$ twice and because $\mathsf{Q}_{\approx}$-normalization is idempotent, we have for all a

$$\begin{aligned}[\![t]\!]^{\xi}_{\mathcal{I}}(a) &= [\![\lambda x.\, u]\!]^{\xi}_{\mathcal{I}}(a) = [\![u\downarrow_{\mathsf{Q}_{\approx}}]\!]^{\xi[x\mapsto a]}_{\mathcal{I}} = [\![u\downarrow_{\mathsf{Q}_{\approx}}\downarrow_{\mathsf{Q}_{\approx}}]\!]^{\xi[x\mapsto a]}_{\mathcal{I}} = [\![\lambda x.\,(u\downarrow_{\mathsf{Q}_{\approx}})]\!]^{\xi}_{\mathcal{I}}(a)\\ &= [\![(\lambda x.\, u)\downarrow_{\mathsf{Q}_{\approx}}]\!]^{\xi}_{\mathcal{I}}(a) = [\![t\downarrow_{\mathsf{Q}_{\approx}}]\!]^{\xi}_{\mathcal{I}}(a)\end{aligned}$$

Otherwise, if $t = s\,\bar{u}$, where s is a head that is not of the form $\mathsf{Q}\langle\tau\rangle$,

$$[\![t]\!]^{\xi}_{\mathcal{I}} = [\![s\,\bar{u}]\!]^{\xi}_{\mathcal{I}} = [\![s]\!]^{\xi}_{\mathcal{I}}([\![\bar{u}]\!]^{\xi}_{\mathcal{I}}) \overset{\mathrm{IH}}{=} [\![s\downarrow_{\mathsf{Q}_{\approx}}]\!]^{\xi}_{\mathcal{I}}([\![\bar{u}\downarrow_{\mathsf{Q}_{\approx}}]\!]^{\xi}_{\mathcal{I}}) = [\![(s\,\bar{u})\downarrow_{\mathsf{Q}_{\approx}}]\!]^{\xi}_{\mathcal{I}} = [\![t\downarrow_{\mathsf{Q}_{\approx}}]\!]^{\xi}_{\mathcal{I}}$$

If $t = \mathsf{Q}\langle\tau\rangle u$, where t is not $\mathsf{Q}_{\approx}$-reducible at its head, then

$$\begin{aligned}[\![t]\!]^{\xi}_{\mathcal{I}} &= [\![\mathsf{Q}\langle\tau\rangle\, u]\!]^{\xi}_{\mathcal{I}} = [\![\mathsf{Q}\langle\tau\rangle]\!]^{\xi}_{\mathcal{I}}([\![u]\!]^{\xi}_{\mathcal{I}}) \overset{\mathrm{IH}}{=} [\![\mathsf{Q}\langle\tau\rangle]\!]^{\xi}_{\mathcal{I}}([\![u\downarrow_{\mathsf{Q}_{\approx}}]\!]^{\xi}_{\mathcal{I}})\\ &= [\![(\mathsf{Q}\langle\tau\rangle\, u)\downarrow_{\mathsf{Q}_{\approx}}]\!]^{\xi}_{\mathcal{I}} = [\![t\downarrow_{\mathsf{Q}_{\approx}}]\!]^{\xi}_{\mathcal{I}}\end{aligned}$$

Otherwise we have $t = \mathsf{Q}\langle\tau\rangle$ or $t = \mathsf{Q}\langle\tau\rangle u$ such that t is $\mathsf{Q}_{\approx}$-reducible at its head. In these cases, it suffices to show that $[\![\forall\langle\tau\rangle]\!]^{\xi}_{\mathcal{I}} = [\![\lambda y.\, y \approx (\lambda x.\, \top)]\!]^{\xi}_{\mathcal{I}}$ and $[\![\exists\langle\tau\rangle]\!]^{\xi}_{\mathcal{I}} = [\![\lambda y.\, y \not\approx (\lambda x.\, \bot)]\!]^{\xi}_{\mathcal{I}}$ for all types τ and all valuations ξ. The argument we use here resembles the proof of Lemma 7.25, but here we cannot assume $\mathcal{I}$ to be proper.

Let f be a function from $[\![\tau]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}$ to $\{0,1\}$. Then

$$[\![\forall\langle\tau\rangle]\!]^{\xi}_{\mathcal{I}}(f) = \mathcal{J}(\forall, [\![\tau]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}})(f) = \min\{f(a) \mid a \in [\![\tau]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}\} = \begin{cases} 1 & \text{if } f \text{ is constantly 1}\\ 0 & \text{otherwise}\end{cases}$$

By our assumption on $\mathcal{I}$, we have

$$\begin{aligned}[\![\lambda y.\, y \approx (\lambda x.\, \mathsf{T})]\!]^{\xi}_{\mathcal{I}}(f) &= [\![(y \approx (\lambda x.\, \mathsf{T})) \downarrow_{\mathsf{Q}_\approx}]\!]^{\xi[y\mapsto f]}_{\mathcal{I}} = [\![y \approx (\lambda x.\, \mathsf{T})]\!]^{\xi[y\mapsto f]}_{\mathcal{I}} \\ &= \begin{cases} 1 & \text{if } [\![y]\!]^{\xi[y\mapsto f]}_{\mathcal{I}} = [\![\lambda x.\, \mathsf{T}]\!]^{\xi[y\mapsto f]}_{\mathcal{I}} \\ 0 & \text{otherwise} \end{cases}\end{aligned}$$

Thus it remains to show that $[\![y]\!]^{\xi[y\mapsto f]}_{\mathcal{I}} = [\![\lambda x.\, \mathsf{T}]\!]^{\xi[y\mapsto f]}_{\mathcal{I}}$ if and only if f is constantly 1. This holds because $[\![y]\!]^{\xi[y\mapsto f]}_{\mathcal{I}}(a) = f(a)$ and, by our assumption on $\mathcal{I}$, $[\![\lambda x.\, \mathsf{T}]\!]^{\xi[y\mapsto f]}_{\mathcal{I}}(a) = [\![\mathsf{T}]\!]^{\xi[x\mapsto a,\, y\mapsto f]}_{\mathcal{I}} = 1$ for all $a \in [\![\tau]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}$.

The case of $\exists$ is analogous. □

Corollary 7.47. *Let $\mathcal{I}$ be an interpretation such that $[\![\lambda x.\, t]\!]^{\xi}_{\mathcal{I}}(a) = [\![t \downarrow_{\mathsf{Q}_\approx}]\!]^{\xi[x\mapsto a]}_{\mathcal{I}}$ for all λ-terms t and all valuations ξ. Then $\mathcal{I}$ is proper.*

Therefore, to show that $\mathcal{I}^{\mathrm{GH}}$ is proper, it suffices to prove that $[\![\lambda x.\, t]\!]^{\xi}_{\mathcal{I}^{\mathrm{GH}}}(a) = [\![t \downarrow_{\mathsf{Q}_\approx}]\!]^{\xi[x\mapsto a]}_{\mathcal{I}^{\mathrm{GH}}}$. For quantifiers, we have the following relation between the higher-order interpretation $\mathcal{I}^{\mathrm{GH}}$ and the first-order interpretation R:

Lemma 7.48. *Let $\mathsf{Q} \in \{\forall, \exists\}$ and $f \in \mathcal{J}^{\mathrm{GH}}_{\mathsf{ty}}(\to)(\mathcal{D}_\tau, \{0,1\})$. Then, for any term p such that $\mathsf{Q}\langle\tau\rangle(\lambda x.\, p)$ is $\mathsf{Q}_\approx$-normal and such that $f = \mathcal{E}([\![\mathcal{F}(\lambda x.\, p)]\!]_R)$,*

$$\mathcal{J}^{\mathrm{GH}}(\mathsf{Q}, \mathcal{D}_\tau)(f) = \mathcal{E}([\![\mathcal{F}(\mathsf{Q}\langle\tau\rangle(\lambda x.\, p))]\!]_R)$$

Proof. Let Q, f, and p be as in the preconditions of the lemma. If $\mathsf{Q} = \forall$, then

$$\begin{aligned}\mathcal{J}^{\mathrm{GH}}(\forall, \mathcal{D}_\tau)(f) &= \min\{f(a) \mid a \in \mathcal{D}_\tau\} \\ &= \min\{\mathcal{E}_{\tau\to o}([\![\mathcal{F}(\lambda x.\, p)]\!]_R)(\mathcal{E}_\tau([\![\mathcal{F}(u)]\!]_R)) \mid \mathcal{E}_\tau([\![\mathcal{F}(u)]\!]_R) \in \mathcal{D}_\tau\} \\ &= \min\{\mathcal{E}_o([\![\mathcal{F}((\lambda x.\, p)\, u)]\!]_R) \mid \mathcal{E}_\tau([\![\mathcal{F}(u)]\!]_R) \in \mathcal{D}_\tau\} \\ &= \min\{[\![\mathcal{F}((\lambda x.\, p)\, u)]\!]_R \mid [\![\mathcal{F}(u)]\!]_R \in \mathcal{U}_\tau\} \\ &= \min\{[\![\mathcal{F}(p\{x \mapsto u\})]\!]_R \mid [\![\mathcal{F}(u)]\!]_R \in \mathcal{U}_\tau\} \\ &= \min\{[\![\mathcal{F}(p)\{x \mapsto \mathcal{F}(u)\}]\!]_R \mid [\![\mathcal{F}(u)]\!]_R \in \mathcal{U}_\tau\} \\ &\qquad \text{since, by Lemma 7.6, } x \text{ never occurs under a } \lambda \text{ in } p, \\ &= \min\{[\![\mathcal{F}(p)]\!]^{\xi\{x\mapsto a'\}}_R \mid a' \in \mathcal{U}_\tau\} \\ &\qquad \text{by Lemma 6.7 (substitution lemma)}, \\ &= [\![\forall x.\, \mathcal{F}(p)]\!]_R = \mathcal{E}_o([\![\mathcal{F}(\forall\langle\tau\rangle(\lambda x.\, p))]\!]_R)\end{aligned}$$

If $\mathsf{Q} = \exists$, the proof is analogous, but uses max instead of min. □

A similar relation holds on all $\mathsf{Q}_\approx$-normal terms:

Lemma 7.49. *Given a ground $\mathsf{Q}_\approx$-normal λ-term t, we have*

$$[\![t]\!]_{\mathcal{I}^{\mathrm{GH}}} = \mathcal{E}([\![\mathcal{F}(t\downarrow_{\beta\eta\mathsf{Q}_\eta})]\!]_R)$$

Proof. The proof is analogous to that of Lemma 5.34, using the $\beta\eta\mathsf{Q}_\eta$-normal form instead of the $\beta\eta$-normal form and with a special case for quantifier-headed terms. We proceed by induction on t. Assume that $[\![s]\!]_{\mathcal{I}^{\mathrm{GH}}} = \mathcal{E}([\![\mathcal{F}(s\downarrow_{\beta\eta\mathsf{Q}_\eta})]\!]_R)$ for all proper

subterms s of t. If t is of the form $\mathsf{f}\langle\bar{\tau}\rangle$, then it cannot be a quantifier since t is $\mathsf{Q}_{\approx}$-normal. Thus:

$$\begin{aligned}
[\![t]\!]_{\mathcal{J}^{\mathrm{GH}}} &= \mathcal{J}^{\mathrm{GH}}(\mathsf{f}, \mathcal{D}_{\bar{\tau}}) \\
&= \mathcal{E}(\mathcal{J}(\mathsf{f}_0, \mathcal{U}_{\mathcal{F}(\bar{\tau})})) \\
&= \mathcal{E}([\![\mathsf{f}_0\langle\mathcal{F}(\bar{\tau})\rangle]\!]_R) \\
&= \mathcal{E}([\![\mathcal{F}(\mathsf{f}\langle\bar{\tau}\rangle)]\!]_R) \\
&= \mathcal{E}([\![\mathcal{F}(\mathsf{f}\langle\bar{\tau}\rangle{\downarrow_{\beta\eta\mathsf{Q}_\eta}})]\!]_R) = \mathcal{E}([\![\mathcal{F}(t{\downarrow_{\beta\eta\mathsf{Q}_\eta}})]\!]_R)
\end{aligned}$$

If t is of the form $t = \mathsf{Q}\langle\tau\rangle u$, then $u = \lambda x.\, v$ since t is $\mathsf{Q}_{\approx}$-normal, and

$$\begin{aligned}
[\![\mathsf{Q}\langle\tau\rangle(\lambda x.\, v)]\!]_{\mathcal{J}^{\mathrm{GH}}} &= [\![\mathsf{Q}\langle\tau\rangle]\!]_{\mathcal{J}^{\mathrm{GH}}}\, [\![\lambda x.\, v]\!]_{\mathcal{J}^{\mathrm{GH}}} \\
&= \mathcal{E}_o([\![\mathcal{F}(\mathsf{Q}\langle\tau\rangle(\lambda x.\, v))]\!]_R) && \text{by Lemma 7.48} \\
&= \mathcal{E}_o([\![\mathcal{F}((\mathsf{Q}\langle\tau\rangle(\lambda x.\, v)){\downarrow_{\beta\eta\mathsf{Q}_\eta}})]\!]_R) && \text{by the definition of } \mathcal{F}
\end{aligned}$$

If t is an application $t = t_1\, t_2$, where t_1 is of type $\tau \to \upsilon$ and t_1 is not a quantifier, then

$$\begin{aligned}
[\![t_1\, t_2]\!]_{\mathcal{J}^{\mathrm{GH}}} &= [\![t_1]\!]_{\mathcal{J}^{\mathrm{GH}}}([\![t_2]\!]_{\mathcal{J}^{\mathrm{GH}}}) \\
&\overset{\mathrm{IH}}{=} \mathcal{E}_{\tau\to\upsilon}([\![\mathcal{F}(t_1{\downarrow_{\beta\eta\mathsf{Q}_\eta}})]\!]_R)(\mathcal{E}_\tau([\![\mathcal{F}(t_2{\downarrow_{\beta\eta\mathsf{Q}_\eta}})]\!]_R)) \\
&= \mathcal{E}_\upsilon([\![\mathcal{F}((t_1\, t_2){\downarrow_{\beta\eta\mathsf{Q}_\eta}})]\!]_R)
\end{aligned}$$

If t is a λ-expression, then

$$\begin{aligned}
[\![\lambda x.\, u]\!]^{\xi}_{\mathcal{J}^{\mathrm{GH}}} &= \mathcal{L}^{\mathrm{GH}}(\xi, (\lambda x.\, u)) \\
&= \mathcal{E}([\![\mathcal{F}((\lambda x.\, u)\theta{\downarrow_{\beta\eta\mathsf{Q}_\eta}})]\!]_R) \\
&= \mathcal{E}([\![\mathcal{F}((\lambda x.\, u){\downarrow_{\beta\eta\mathsf{Q}_\eta}})]\!]_R)
\end{aligned}$$

where θ is a substitution such that $\mathcal{D}_{\alpha\theta} = \xi(\alpha)$ and $\mathcal{E}([\![\mathcal{F}(x\theta)]\!]_R) = \xi(x)$. □

We also need to employ the following lemma, which is very similar to the substitution lemma, but we must prove it here for our particular interpretation $\mathcal{J}^{\mathrm{GH}}$ because we have not shown that $\mathcal{J}^{\mathrm{GH}}$ is proper yet.

Lemma 7.50 (Substitution lemma)**.** *$[\![\tau\rho]\!]^{\xi}_{\mathcal{J}^{\mathrm{GH}}_{\mathrm{ty}}} = [\![\tau]\!]^{\xi'}_{\mathcal{J}^{\mathrm{GH}}_{\mathrm{ty}}}$ and $[\![t\rho]\!]^{\xi}_{\mathcal{J}^{\mathrm{GH}}} = [\![t]\!]^{\xi'}_{\mathcal{J}^{\mathrm{GH}}}$ for all $\mathsf{Q}_{\approx}$-normal λ-terms t, all $\tau \in \mathcal{T}y_{\mathrm{H}}$ and all grounding substitutions ρ, where $\xi'(\alpha) = [\![\alpha\rho]\!]^{\xi}_{\mathcal{J}^{\mathrm{GH}}_{\mathrm{ty}}}$ for all type variables α and $\xi'(x) = [\![x\rho]\!]^{\xi}_{\mathcal{J}^{\mathrm{GH}}}$ for all term variables x.*

Proof. Analogous to Lemma 5.35, using the $\beta\eta\mathsf{Q}_\eta$-normal form instead of the $\beta\eta$-normal form. □

Lemma 7.51. *The interpretation $\mathcal{J}^{\mathrm{GH}}$ is proper.*

Proof. By Corollary 7.47, it is enough to show that $[\![\lambda x.\, t]\!]^{\xi}_{\mathcal{J}^{\mathrm{GH}}}(a) = [\![t{\downarrow_{\mathsf{Q}_{\approx}}}]\!]^{\xi[x\mapsto a]}_{\mathcal{J}^{\mathrm{GH}}}$. First, we show it for all $\mathsf{Q}_{\approx}$-normal λ-expressions $\lambda x.\, t$, all valuations ξ, and all

values a:

$$\begin{aligned}
[\![\lambda x.\, t]\!]^{\xi}_{\mathcal{I}^{\mathrm{GH}}}(a) &= \mathcal{L}^{\mathrm{GH}}(\xi, \lambda x.\, t)(a) && \text{by the definition of } [\![\ ]\!]_{\mathcal{I}^{\mathrm{GH}}} \\
&= \mathcal{E}([\![\mathcal{F}((\lambda x.\, t)\theta \downarrow_{\beta\eta\mathsf{Q}_\eta})]\!]_R)(a) && \text{by the definition of } \mathcal{L}^{\mathrm{GH}} \text{ for some } \theta \\
& && \text{such that } \mathcal{E}([\![\mathcal{F}(z\theta)]\!]_R) = \xi(z) \text{ for all } z \\
& && \text{and } \mathcal{D}_{a\theta} = \xi(a) \text{ for all } a \\
&= \mathcal{E}([\![\mathcal{F}(((\lambda x.\, t)\theta\ s) \downarrow_{\beta\eta\mathsf{Q}_\eta})]\!]_R) && \text{by the definition of } \mathcal{E} \\
& && \text{where } \mathcal{E}([\![\mathcal{F}(s)]\!]_R) = a \\
&= \mathcal{E}([\![\mathcal{F}(t(\theta[x \mapsto s]) \downarrow_{\beta\eta\mathsf{Q}_\eta})]\!]_R) && \text{by } \beta\text{-reduction} \\
&= [\![t(\theta[x \mapsto s])]\!]_{\mathcal{I}^{\mathrm{GH}}} && \text{by Lemma 7.49} \\
&= [\![t]\!]^{\xi[x \mapsto a]}_{\mathcal{I}^{\mathrm{GH}}} && \text{by Lemma 7.50} \\
&= [\![t \downarrow_{\mathsf{Q}_{\approx}}]\!]^{\xi[x \mapsto a]}_{\mathcal{I}^{\mathrm{GH}}} && \text{because } t \text{ is } \mathsf{Q}_{\approx}\text{-normal by definition}
\end{aligned}$$

The case where $\lambda x.\, t$ is not $\mathsf{Q}_{\approx}$-normal is reduced to the previous case because then $[\![\lambda x.\, t]\!]^{\xi}_{\mathcal{I}^{\mathrm{GH}}}(a) = \mathcal{L}^{\mathrm{GH}}(\xi, (\lambda x.\, t) \downarrow_{\mathsf{Q}_{\approx}})(a)$ and $(\lambda x.\, t) \downarrow_{\mathsf{Q}_{\approx}} = \lambda x.t'$ where $t' = t \downarrow_{\mathsf{Q}_{\approx}}$ by definition. □

Lemma 7.52. *$\mathcal{I}^{\mathrm{GH}}$ is a model of N.*

Proof. Because all terms in N are $\mathsf{Q}_{\approx}$-normal, by Lemma 7.49, $[\![t]\!]_{\mathcal{I}^{\mathrm{GH}}} = \mathcal{E}([\![\mathcal{F}(t)]\!]_R)$ for all $t \in \mathcal{T}_{\mathrm{GH}}$. Since $\mathcal{E}$ is a bijection, it follows that any literal $s \mathrel{\dot{\approx}} t \in \mathcal{C}_{\mathrm{GH}}$ is true in $\mathcal{I}^{\mathrm{GH}}$ if and only if $\mathcal{F}(s \mathrel{\dot{\approx}} t)$ is true in R. Hence, a clause $C \in \mathcal{C}_{\mathrm{GH}}$ is true in $\mathcal{I}^{\mathrm{GH}}$ if and only if $\mathcal{F}(C)$ is true in R. By Theorem 7.43 and the assumption that $\bot \notin N$, R is a model of $\mathcal{F}(N)$—that is, for all clauses $C \in N$, $\mathcal{F}(C)$ is true in R. Hence, all clauses $C \in N$ are true in $\mathcal{I}^{\mathrm{GH}}$ and therefore $\mathcal{I}^{\mathrm{GH}}$ is a model of N. □

We summarize the results of this subsection in the following theorem:

Theorem 7.53 (Ground static completeness)**.** *Let $q \in Q$ be some parameter triple. Then GHInf^q is statically refutationally complete w.r.t. $\models$ and (GHRed$^q_{\mathrm{I}}$, GHRed$_{\mathrm{C}}$). In other words, if $N \subseteq \mathcal{C}_{\mathrm{GH}}$ is a clause set saturated w.r.t. GHInf^q and GHRed$^q_{\mathrm{I}}$, then $N \models \bot$ if and only if $\bot \in N$.*

The construction of $\mathcal{I}^{\mathrm{GH}}$ relies on the specific properties of R. It would not work with an arbitrary interpretation. In the other direction, transforming a higher-order model into a first-order model with interpreted Booleans is easier, as the following lemma shows:

Lemma 7.54. *Given a proper higher-order interpretation $\mathcal{I}$ on* GH*, there exists an interpretation $\mathcal{I}^{\mathrm{GF}}$ on* GF *such that for any clause $C \in \mathcal{C}_{\mathrm{GH}}$ the truth values of C in $\mathcal{I}$ and of $\mathcal{F}(C)$ in $\mathcal{I}^{\mathrm{GF}}$ coincide.*

Proof. Let $\mathcal{I} = (\mathcal{I}_{\mathsf{ty}}, \mathcal{J}, \mathcal{L})$ be a proper higher-order interpretation on GH. Let $\mathcal{U}^{\mathrm{GF}}_{\tau} = [\![\tau]\!]_{\mathcal{I}_{\mathsf{ty}}}$ be the GF universe for the ground type τ. For a symbol $\mathsf{f}^{\bar{\upsilon}}_j \in \Sigma_{\mathrm{GF}}$, let $\mathcal{J}^{\mathrm{GF}}(\mathsf{f}^{\bar{\upsilon}}_j) = [\![\mathsf{f}\langle\bar{\upsilon}\rangle]\!]_{\mathcal{I}}$ (up to currying). For a symbol $\mathsf{lam}_{\lambda x.\, t} \in \Sigma_{\mathrm{GF}}$, let $\mathcal{J}^{\mathrm{GF}}(\mathsf{lam}_{\lambda x.\, t}) = [\![\lambda x.\, t]\!]_{\mathcal{I}}$.

The requirements on the GF-interpretation of logical symbols are fulfilled because we have similar requirements on H: $\mathcal{U}^{\mathrm{GF}}_o = \mathcal{I}_{\mathsf{ty}}(o) = \{0, 1\}$; $\mathcal{J}^{\mathrm{GF}}(\mathsf{T}_0) = \mathcal{J}(\mathsf{T}) = 1$;

$\mathcal{J}^{\mathrm{GF}}(\neg_1)(a) = \mathcal{J}(\neg)(a) = 1 - a$; and similarly for the other logical symbols. Thus, this defines an interpretation $\mathcal{I}^{\mathrm{GF}} = (\mathcal{U}^{\mathrm{GF}}, \mathcal{J}^{\mathrm{GF}})$ on GF.

We need to show that for any $C \in \mathcal{C}_{\mathrm{GH}}$, $\mathcal{I} \models C$ if and only if $\mathcal{I}^{\mathrm{GF}} \models \mathcal{F}(C)$. It suffices to show that $[\![t]\!]^{\xi}_{\mathcal{I}} = [\![\mathcal{F}(t)]\!]^{\xi}_{\mathcal{I}^{\mathrm{GF}}}$ for all terms $t \in \mathcal{T}_{\mathrm{GH}}$. We prove this by induction on the structure of the $\beta\eta\mathsf{Q}_\eta$-normal form of t. If t is a λ-expression, this is obvious. If t is of the form $\mathsf{f}\langle\bar{v}\rangle\,\bar{s}_j$, then $\mathcal{F}(t) = \mathsf{f}^{\bar{v}}_j(\mathcal{F}(\bar{s}_j))$ and hence

$$[\![\mathcal{F}(t)]\!]^{\xi}_{\mathcal{I}^{\mathrm{GF}}} = \mathcal{J}^{\mathrm{GF}}(\mathsf{f}^{\bar{v}}_j)([\![\mathcal{F}(\bar{s}_j)]\!]^{\xi}_{\mathcal{I}^{\mathrm{GF}}}) = [\![\mathsf{f}\langle\bar{v}\rangle]\!]^{\xi}_{\mathcal{I}}([\![\mathcal{F}(\bar{s}_j)]\!]^{\xi}_{\mathcal{I}^{\mathrm{GF}}}) \overset{\mathrm{IH}}{=} [\![\mathsf{f}\langle\bar{v}\rangle]\!]^{\xi}_{\mathcal{I}}([\![\bar{s}_j]\!]^{\xi}_{\mathcal{I}}) = [\![t]\!]^{\xi}_{\mathcal{I}}$$

If t is of the form $\mathsf{\forall}\langle\tau\rangle(\lambda x.\, s)$, then $\mathcal{F}(t) = \forall x.\, \mathcal{F}(s)$ and hence

$$\begin{aligned}[\![\mathcal{F}(t)]\!]^{\xi}_{\mathcal{I}^{\mathrm{GF}}} &= \min\{[\![\mathcal{F}(s)]\!]^{\xi[x\mapsto a]}_{\mathcal{I}^{\mathrm{GF}}} \mid a \in \mathcal{U}^{\mathrm{GF}}_{\tau}\} \overset{\mathrm{IH}}{=} \min\{[\![s]\!]^{\xi[x\mapsto a]}_{\mathcal{I}} \mid a \in [\![\tau]\!]_{\mathcal{I}_{\mathrm{ty}}}\} \\ &= \min\{[\![\lambda x.\, s]\!]^{\xi}_{\mathcal{I}}(a) \mid a \in [\![\tau]\!]_{\mathcal{I}_{\mathrm{ty}}}\} = [\![t]\!]^{\xi}_{\mathcal{I}}\end{aligned}$$

A similar argument applies for $\mathsf{\exists}$. Since the definition of $\mathcal{F}$ recurses into subterms below quantifiers, we finally need to consider the case where t is a variable x. In that case, we have $\mathcal{F}(x) = x$ and hence $[\![\mathcal{F}(t)]\!]^{\xi}_{\mathcal{I}^{\mathrm{GF}}} = \xi(x) = [\![t]\!]^{\xi}_{\mathcal{I}}$. □

7.4.4. The Nonground Higher-Order Level

To lift the result to the nonground level, we employ the saturation framework of Waldmann et al. [140]. Clearly, the entailment relation $\models$ on GH qualifies as a consequence relation in the sense of the framework. We need to show that our redundancy criterion on GH qualifies as a redundancy criterion and that $\mathcal{G}$ qualifies as a grounding function:

7

Lemma 7.55. *The pair $(GHRed^{q}_{\mathrm{I}}, GHRed_{\mathrm{C}})$ is a redundancy criterion in the sense of the saturation framework.*

Proof. We must prove the conditions (R1) to (R4) of the saturation framework. Adapted to our context, they state the following for all clause sets $N, N' \subseteq \mathcal{C}_{\mathrm{GH}}$:

(R1) if $N \models \bot$, then $N \setminus GHRed_{\mathrm{C}}(N) \models \bot$;
(R2) if $N \subseteq N'$, then $GHRed_{\mathrm{C}}(N) \subseteq GHRed_{\mathrm{C}}(N')$ and $GHRed_{\mathrm{I}}(N) \subseteq GHRed_{\mathrm{I}}(N')$;
(R3) if $N' \subseteq GHRed_{\mathrm{C}}(N)$, then $GHRed_{\mathrm{C}}(N) \subseteq GHRed_{\mathrm{C}}(N \setminus N')$ and $GHRed_{\mathrm{I}}(N) \subseteq GHRed_{\mathrm{I}}(N \setminus N')$;
(R4) if $\iota \in GHInf$ and $concl(\iota) \in N$, then $\iota \in GHRed_{\mathrm{I}}(N)$.

For (R1), it suffices to show that $N \setminus GHRed_{\mathrm{C}}(N) \models N$. Let $\mathcal{I}$ be a model of $N \setminus GHRed_{\mathrm{C}}(N)$. By Lemma 7.54, there exists a model $\mathcal{I}^{\mathrm{GF}}$ of $\mathcal{F}(N \setminus GHRed_{\mathrm{C}}(N)) = \mathcal{F}(N) \setminus GFRed_{\mathrm{C}}(\mathcal{F}(N))$. We show that $\mathcal{I}^{\mathrm{GF}} \models C$ for each clause $C \in \mathcal{F}(N)$ by well-founded induction on C w.r.t. $\succ$. If $C \notin GFRed_{\mathrm{C}}(\mathcal{F}(N))$, we have already shown that $\mathcal{I}^{\mathrm{GF}} \models C$. Otherwise, $C \in GFRed_{\mathrm{C}}(\mathcal{F}(N))$ and hence $\{D \in \mathcal{F}(N) \mid D \prec C\} \models C$. By the induction hypothesis, it follows that $\mathcal{I}^{\mathrm{GF}} \models C$. Thus, we have shown that $\mathcal{I}^{\mathrm{GF}} \models \mathcal{F}(N)$. By Lemma 7.54, this implies $\mathcal{I} \models N$.

For the first part of (R2), let $N \subseteq N'$ and $C \in GHRed_{\mathrm{C}}(N)$, which is defined as $\{D \in \mathcal{F}(N) \mid D \prec \mathcal{F}(C)\} \models \mathcal{F}(C)$. We must show that $\{D \in \mathcal{F}(N') \mid D \prec \mathcal{F}(C)\} \models \mathcal{F}(C)$. This is obvious because $\{D \in \mathcal{F}(N) \mid D \prec \mathcal{F}(C)\} \subseteq \{D \in \mathcal{F}(N') \mid D \prec \mathcal{F}(C)\}$.

For the second part of (R2), let $N \subseteq N'$ and $\iota \in GHRed_{\mathrm{I}}(N)$. We must show that $\iota \in GHRed_{\mathrm{I}}(N')$. If ι is a GArgCong, GExt, or GChoice inference, we have $concl(\iota) \in$

$N \cup GHRed_C(N)$. Using the first part of (R2), it follows that $N \cup GHRed_C(N) \subseteq N' \cup GHRed_C(N')$, which implies $\iota \in GHRed_I(N')$. If ι is some other kind of inference, we have $prems(\mathcal{F}(\iota)) \cap GFRed_C(\mathcal{F}(N)) \neq \varnothing$ or $\{D \in \mathcal{F}(N) \mid D \prec mprem(\mathcal{F}(\iota))\} \models concl(\mathcal{F}(\iota))$. In the first case, $prems(\mathcal{F}(\iota)) \cap GFRed_C(\mathcal{F}(N')) \neq \varnothing$ because by the first part of (R2), we have $GFRed_C(\mathcal{F}(N)) \subseteq GFRed_C(\mathcal{F}(N'))$. In the second case, we have $\{D \in \mathcal{F}(N') \mid D \prec mprem(\mathcal{F}(\iota))\} \models concl(\mathcal{F}(\iota))$ because $\{D \in \mathcal{F}(N) \mid D \prec mprem(\mathcal{F}(\iota))\} \subseteq \{D \in \mathcal{F}(N') \mid D \prec mprem(\mathcal{F}(\iota))\}$.

For the first part of (R3), let $N' \subseteq GHRed_C(N)$ and $C \in GHRed_C(N)$, which is defined as $\{D \in \mathcal{F}(N) \mid D \prec \mathcal{F}(C)\} \models \mathcal{F}(C)$. We must show that $\{D \in \mathcal{F}(N \setminus N') \mid D \prec \mathcal{F}(C)\} \models \mathcal{F}(C)$. Let $\mathcal{I}$ be a model of $\{D \in \mathcal{F}(N \setminus N') \mid D \prec \mathcal{F}(C)\}$. It suffices to show that $\mathcal{I} \models \{D \in \mathcal{F}(N) \mid D \prec \mathcal{F}(C)\}$, meaning $\mathcal{I} \models E$ for every $E \in \mathcal{F}(N)$ such that $E \prec \mathcal{F}(C)$. We prove this by well-founded induction on E w.r.t. $\succ$. If $E \in \mathcal{F}(N \setminus N')$, the claim holds by assumption. Otherwise, $E \in \mathcal{F}(N') \subseteq GFRed_C(\mathcal{F}(N))$; hence $\{D \in \mathcal{F}(N) \mid D \prec E\} \models E$ and therefore $\mathcal{I} \models E$ by the induction hypothesis.

For the second part of (R3), let $N' \subseteq GHRed_C(N)$ and $\iota \in GHRed_I(N)$. We must show that $\iota \in GHRed_I(N \setminus N')$. If ι is a GArgCong, GExt, or GChoice inference, we have $concl(\iota) \in N \cup GHRed_C(N)$. Using $N' \subseteq GHRed_C(N)$, and by the first part of (R3), it follows that $concl(\iota) \in N \cup GHRed_C(N) = (N \setminus N') \cup GHRed_C(N) \subseteq (N \setminus N') \cup GHRed_C(N \setminus N')$ and therefore $\iota \in GHRed_I(N \setminus N')$. If ι is some other kind of inference, we have $prems(\mathcal{F}(\iota)) \cap GFRed_C(\mathcal{F}(N)) \neq \varnothing$ or $\{D \in \mathcal{F}(N) \mid D \prec mprem(\mathcal{F}(\iota))\} \models concl(\mathcal{F}(\iota))$. In the first case, $prems(\mathcal{F}(\iota)) \cap GFRed_C(\mathcal{F}(N \setminus N')) \neq \varnothing$ because by the first part of (R3), we have $GFRed_C(\mathcal{F}(N)) \subseteq GFRed_C(\mathcal{F}(N \setminus N'))$. In the second case, it suffices to show that $\{D \in \mathcal{F}(N) \mid D \prec mprem(\mathcal{F}(\iota))\} \models \{D \in \mathcal{F}(N \setminus N') \mid D \prec mprem(\mathcal{F}(\iota))\}$, which can be shown analogously to the induction used for the first part of (R3).

For (R4), let $\iota \in GHInf$ and $concl(\iota) \in N$. We must show that $\iota \in GHRed_I(N)$. If ι is a GArgCong, GExt, or GChoice inference, we must show $concl(\iota) \in N \cup GHRed_C(N)$, which obviously holds by assumption. If ι is some other kind of inference, it suffices to show $\{D \in \mathcal{F}(N) \mid D \prec mprem(\mathcal{F}(\iota))\} \models concl(\mathcal{F}(\iota))$. This holds because $concl(\mathcal{F}(\iota)) \in \mathcal{F}(N)$ and $concl(\mathcal{F}(\iota)) \prec mprem(\mathcal{F}(\iota))$. □

Lemma 7.56. *For every $q \in Q$, the function $\mathcal{G}^q$ is a grounding function in the sense of the saturation framework.*

Proof. We must prove the conditions (G1), (G2), and (G3) of the saturation framework. Adapted to our context, they state the following:

(G1) $\mathcal{G}(\bot) = \{\bot\}$;
(G2) for every $C \in \mathcal{C}_H$, if $\bot \in \mathcal{G}(C)$, then $C = \bot$;
(G3) for every $\iota \in HInf$, $\mathcal{G}^q(\iota) \subseteq GHRed_I^q(\mathcal{G}(concl(\iota)))$.

Clearly, $C = \bot$ if and only if $\bot \in \mathcal{G}(C)$ if and only if $\mathcal{G}(C) = \{\bot\}$, proving (G1) and (G2). For every $\iota \in HInf$, by the definition of $\mathcal{G}^q$ (Definition 7.39) and by Lemma 7.36, we have $concl(\mathcal{G}^q(\iota)) \subseteq \mathcal{G}(concl(\iota))$, and thus (G3) by (R4). □

As in Chapters 3 and 5, we employ Theorem 14 of the saturation framework to lift the completeness result of the previous subsection to the nonground calculus *HInf*. Adapted to our context, it resembles Theorem 3.38 and 5.42, but uses a parameter triple q instead of just the literal selection function. The theorem uses the notation

$\mathit{Inf}(N)$ to denote the set of Inf-inferences whose premises are in N, for an inference system Inf and a clause set N. It is stated as follows:

Theorem 7.57 (Lifting theorem)**.** *If GHInf^q is statically refutationally complete w.r.t. $(\mathit{GHRed}^q_{\mathrm{I}}, \mathit{GHRed}_{\mathrm{C}})$ for every parameter triple $q \in Q$, and if for every $N \subseteq \mathcal{C}_{\mathrm{H}}$ that is saturated w.r.t. HInf and $\mathit{HRed}_{\mathrm{I}}$ there exists a $q \in Q$ such that $\mathit{GHInf}^q(\mathcal{G}(N)) \subseteq \mathcal{G}^q(\mathit{HInf}(N)) \cup \mathit{GHRed}^q_{\mathrm{I}}(\mathcal{G}(N))$, then also HInf is statically refutationally complete w.r.t. $(\mathit{HRed}_{\mathrm{I}}, \mathit{HRed}_{\mathrm{C}})$ and $\models_{\mathcal{G}}$.*

Proof. This is almost an instance of Theorem 14 of the saturation framework. We take $\mathcal{C}_{\mathrm{H}}$ for $\mathbf{F}$, $\mathcal{C}_{\mathrm{GH}}$ for $\mathbf{G}$. Clearly, the entailment relation $\models$ on GH is a consequence relation in the sense of the framework. By Lemma 7.55 and 7.56, $(\mathit{GHRed}^q_{\mathrm{I}}, \mathit{GHRed}_{\mathrm{C}})$ is a redundancy criterion in the sense of the framework, and $\mathcal{G}^q$ are grounding functions in the sense of the framework, for all $q \in Q$. The redundancy criterion $(\mathit{HRed}_{\mathrm{I}}, \mathit{HRed}_{\mathrm{C}})$ matches exactly the intersected lifted redundancy criterion $\mathit{Red}^{\cap,\sqsupset}$ of the saturation framework. Their Theorem 14 states the theorem only for $\sqsupset = \varnothing$. By their Lemma 16, it also holds if $\sqsupset \neq \varnothing$. □

Let $N \subseteq \mathcal{C}_{\mathrm{H}}$ be a clause set saturated w.r.t. HInf and $\mathit{HRed}_{\mathrm{I}}$. For the above theorem to apply, we need to show that there exists a $q \in Q$ such that all inferences $\iota \in \mathit{GHInf}^q$ with $\mathit{prems}(\iota) \in \mathcal{G}(N)$ are liftable or redundant. Here, ι's being *liftable* means that ι is a $\mathcal{G}^q$-ground instance of a HInf-inference from N; ι's being *redundant* means that $\iota \in \mathit{GHRed}^q_{\mathrm{I}}(\mathcal{G}(N))$.

To choose the right $q \in Q$, we observe that each ground clause $C \in \mathcal{G}(N)$ must have at least one corresponding clause $D \in N$ such that C is a ground instance of D. We choose one of them for each $C \in \mathcal{G}(N)$, which we denote by $\mathcal{G}^{-1}(C)$. Then we choose $\mathit{GHLitSel}$ and $\mathit{GHBoolSel}$ such that the selections in C correspond to those in $\mathcal{G}^{-1}(C)$.

To choose the witness function GHWit, let $C \in \mathcal{C}_{\mathrm{GH}}$ and let p a green position of a quantifier-headed term $C|_p = \mathsf{Q}\langle\tau\rangle t$. Let $D = \mathcal{G}^{-1}(C)$ and let θ be the grounding substitution such that $D\theta = C$. Let p' be the corresponding green position in D. If there exists no such position p', we define $\mathit{GHWit}(C,p)$ to be some arbitrary term that fulfills the order requirements of a witness function. Otherwise, let β and y be fresh variables and we extend θ to a substitution θ' by defining $\beta\theta' = \tau$ and $y\theta' = t$. Then θ' is a unifier of $\mathsf{Q}\langle\beta\rangle y$ and $D|_{p'}$ and hence there exists an idempotent $\sigma \in \mathrm{CSU}(Q\langle\beta\rangle y, D|_{p'})$ such that for some substitution ρ and for all variables x in D and for $x \in \{y,\beta\}$, we have $x\sigma\rho = x\theta'$. We let $\mathit{GHWit}(C,p)$ be $\mathsf{sk}_{\Pi\bar{\alpha}.\forall\bar{x}.\exists z.\neg(y\sigma\, z)}\langle\bar{\alpha}\rangle\bar{x}\theta$ if the quantifier-headed term is a $\forall$-term and $\mathsf{sk}_{\Pi\bar{\alpha}.\forall\bar{x}.\exists z.(y\sigma\, z)}\langle\bar{\alpha}\rangle\bar{x}\theta$ if the quantifier-headed term is an $\exists$-term where $\bar{\alpha}$ are the free type variables and $\bar{x}$ are the free variables occurring in $D|_{p'}$ in order of first appearance.

By definition of $\mathcal{G}$ (Definition 7.29), for all variables x occurring in D the only Boolean green subterms of $x\theta$ are $\top$ and $\bot$. The term $\mathsf{Q}\langle\tau\rangle t$ must be $\mathsf{Q}_{\approx}$-normal because it occurs in $C \in \mathcal{C}_{\mathrm{GH}}$. Hence $\mathsf{Q}\langle\tau\rangle t \succ t\,\mathit{GHWit}(C,p)$ by order condition (O4).

With respect to this parameter triple $q = (\mathit{GHLitSel}, \mathit{GHBoolSel}, \mathit{GHWit})$, we can show that all inferences from $\mathcal{G}(N)$ are liftable or redundant:

Lemma 7.58. *Let $C\theta \in \mathcal{C}_{\mathrm{GH}}$ and $C = \mathcal{G}^{-1}(C\theta)$. Let σ and ρ be substitutions such that $x\sigma\rho = x\theta$ for all variables in C. (This holds for example if σ is an element of a*

CSU corresponding to a unifier θ.) If a literal in a clause $C\theta$ is (strictly) $\succeq$-eligible w.r.t. GHLitSel, then the corresponding literal in C is (strictly) $\succsim$-eligible w.r.t. σ and HLitSel. If a green position in a clause $C\theta$ is $\succeq$-eligible w.r.t. GHBoolSel and there exists a corresponding green position in C, then the corresponding position in C is $\succsim$-eligible w.r.t. σ and HBoolSel.

Proof. LITERALS: If the literal in $C\theta$ is selected w.r.t. *GHLitSel*, then the corresponding literal is also selected in $C = \mathcal{G}^{-1}(C\theta)$ w.r.t. *HLitSel* by definition of *GHLitSel*. If $L\theta$ is (strictly) $\succeq$-maximal in $C\theta$, then $L\sigma$ is (strictly) $\succsim$-maximal in $C\sigma$.

POSITIONS: Let p be the position in $C\theta$ and let p' be the corresponding position in C. We proceed by induction over the definition of eligible positions. If p is selected in $C\theta$ w.r.t. *GHBoolSel*, then p' is selected in $C = \mathcal{G}^{-1}(C\theta)$ w.r.t. *HBoolSel* by definition of *GHBoolSel*. Otherwise, if p is at the top level of a literal $L\theta = s\theta \mathrel{\tilde{\approx}} t\theta$, then $s\theta \not\preceq t\theta$ implies $s\sigma \not\precsim t\sigma$, (strict) $\succeq$-eligibility of $L\theta$ implies (strict) $\succsim$-eligibility of L w.r.t. σ (as shown above), and hence p' eligible in C w.r.t. σ. Otherwise, the position p is neither selected nor at the top level. Let q be the position directly above p and q' be the position directly above p'. By the induction hypothesis, q and q' are eligible. If the head of $C\theta_p$ is not $\approx$ or $\not\approx$, then the head of $C_{p'}$ cannot be $\approx$ or $\not\approx$ either. If the head $C\theta_p$ is $\approx$ or $\not\approx$, then $C_{p'}$ must also be $\approx$ or $\not\approx$ because the position p' is green. Hence, p' is eligible because $s\theta \not\preceq t\theta$ implies $s\sigma \not\precsim t\sigma$. □

In some edge cases, it is ambiguous what "the corresponding" literal is. When $C\theta$ contains multiple occurrences of a literal that correspond to different literals in C, the $\succsim$-larger one must be chosen as the corresponding literal to make the lemma above work. In the following, we will implicitly assume that the correct literal is chosen when we refer to "the corresponding" literal.

Lemma 7.59. *All* ERES, EFACT, GARGCONG, GEXT, GCHOICE, BOOLHOIST, *and* FALSEELIM *inferences are liftable.*

Proof. For ERES, EFACT, GARGCONG, and GEXT, the proof is as in Lemma 5.44. For GCHOICE, the proof is analogous to GEXT.

BOOLHOIST: Let $\iota \in \mathit{GHInf}$ be an BOOLHOIST inference with $\mathit{prems}(\iota) \in \mathcal{G}(N)$. Then ι is of the form

$$\frac{C\theta\langle u\rangle_p}{C\theta\langle \bot\rangle_p \vee u \approx \top}\ \text{BOOLHOIST}$$

where $\mathcal{G}^{-1}(C\theta) = C$.

If p corresponds to a position at or below an unapplied variable in C, u could only be $\top$ or $\bot$, contradicting the condition of BOOLHOIST that the head of u is not a fully applied logical symbol.

If p corresponds to a position at or below a fluid term in C, we will lift to a FLUIDBOOLHOIST inference. Let $p = p_1.p_2$ such that p_1 is the longest prefix of p that corresponds to a green position p'_1 in C. Let $v = C|_{p'_1}$. Then v is fluid. Let z and x be fresh variables. Define a substitution θ' that maps the variable z to $\lambda y.(v\theta)\langle y\rangle_{p_2}$, the variable x to $v\theta|_{p_2}$, and all other variables w to $w\theta$. Then $(z\,x)\theta' = (v\theta)\langle v\theta|_{p_2}\rangle_{p_2} = v\theta = v\theta'$. So θ' is a unifier of $z\,x$ and v and thus there exists an idempotent $\sigma \in \mathrm{CSU}(z\,x, v)$ such that for some substitution ρ, for all variables y

in C, and for $y \in \{x, z\}$, we have $y\sigma\rho = y\theta'$. By the conditions of BOOLHOIST, $u \neq \mathsf{T}$ and $u \neq \bot$. Then $x\sigma \neq \mathsf{T}$ and $x\sigma \neq \bot$ because $u = v\theta|_{p_2} = x\theta' = x\sigma\rho$. Hence, we have $(z\,\bot)\theta' = (v\theta)\langle\bot\rangle_{p_2} \neq (v\theta)\langle x\theta'\rangle_{p_2} = (z\,x)\theta'$ and thus $(z\,\bot)\sigma \neq (z\,x)\sigma$. The position p_1 must be eligible in $C\theta$ because p is eligible in $C\theta$ and p_1 is the longest prefix of p that corresponds to a green position p'_1 in C. Eligibility of p_1 in $C\theta$ implies eligibility of p'_1 in C by Lemma 7.58. Thus there exists the following FLUIDBOOLHOIST inference ι':

$$\frac{C\langle v\rangle_{p'_1}}{(C\langle z\,\bot\rangle_{p'_1} \vee x \approx \mathsf{T})\sigma}\,\text{FLUIDBOOLHOIST}$$

The inference ι is the $\sigma\rho$-ground instance of ι' and is therefore liftable.

Otherwise, we will lift to a BOOLHOIST inference. Since u is not at or below a variable-headed term, there is a subterm u' of C at position p' corresponding to the subterm u of $C\theta$ at position p. Since u is a Boolean term, there is a type unifier σ of the type of u' with the Boolean type. Eligibility of u in $C\theta$ implies eligibility of u' in C by Lemma 7.58. Since the occurrence of u in $C\theta$ is not at the top level of a positive literal, the corresponding occurrence of u' in C is not at the top level of a positive literal either. Thus there exists the following BOOLHOIST inference ι':

$$\frac{C\langle u'\rangle_{p'}}{(C\langle\bot\rangle_{p'} \vee u' \approx \mathsf{T})\sigma}\,\text{BOOLHOIST}$$

Then ι is a ground instance of ι' and is therefore liftable.

FALSEELIM: Let $\iota \in \mathit{GHInf}$ be an FALSEELIM inference with $\mathit{prems}(\iota) \in \mathcal{G}(N)$. Then ι is of the form

$$\frac{C\theta \;=\; C'\theta \vee s\theta \approx s'\theta}{C'\theta}\,\text{FALSEELIM}$$

where $\mathcal{G}^{-1}(C\theta) = C = C' \vee s \approx s'$ and the literal $s\theta \approx s'\theta$ is strictly $\succeq$-eligible w.r.t. $\mathit{GHLitSel}$. Since $s\theta \approx s'\theta$ and $\bot \approx \mathsf{T}$ are unifiable and ground, we have $s\theta = \bot$ and $s'\theta = \mathsf{T}$. Thus, there exists an idempotent $\sigma \in \mathrm{CSU}(s \approx s', \bot \approx \mathsf{T})$ such that for some substitution ρ and for all variables x in C, we have $x\sigma\rho = x\theta$. Then $s \approx s'$ is strictly $\succsim$-eligible in C w.r.t. σ. Hence, the following inference $\iota' \in \mathit{HInf}$ is applicable:

$$\frac{C' \vee s \approx s'}{C'\sigma}\,\text{FALSEELIM}$$

Then ι is the $\sigma\rho$-ground instance of ι' and is therefore liftable. □

Lemma 7.60. *All* SUP *inferences are liftable or redundant.*

Proof. The proof is as for Lemmas 5.46 and 5.47. The proof works with the altered definition of deeply occurring variables because congruence holds below quantifiers on the GF level. □

Lemma 7.61. *All* EQHOIST, NEQHOIST, GFORALLHOIST, GEXISTSHOIST, GFORALLRW, GEXISTSRW, *and* BOOLRW *inferences from* $\mathcal{G}(N)$ *are liftable or redundant.*

Proof. Let $\iota \in \mathit{GHInf}$ be a EqHoist, NeqHoist, GForallHoist, GExistsHoist, GForallRw, GExistsRw, or BoolRw inference from $\mathcal{G}(N)$. Let $C\theta = \mathit{prems}(\iota)$ where $C = \mathcal{G}^{-1}(C\theta) \in N$. Let p be the position of the affected subterm in $C\theta$.

We distinguish two cases. We will show that ι is liftable if

(A) p corresponds to a position in C that is not at or below a fluid term, or
(B) p is the position of a term v in a literal $v \approx \mathsf{T}$ or $v \approx \bot$ in $C\theta$.

Otherwise, we will show that ι is redundant.

Liftable Cases: If condition (A) or (B) holds, p corresponds to some position p' in C. Let $u = C|_{p'}$. By the definition of the grounding function $\mathcal{G}$, for all variables x occurring in C, the only Boolean green subterms of $x\theta$ are T and $\bot$. Since $u\theta$ is a fully applied logical symbol different from T and $\bot$, the term u cannot be a variable. Eligibility of p in $C\theta$ implies eligibility of p' in C by Lemma 7.58. If u is a fluid term, by conditions (A) and (B), it must be in a literal $u \approx \mathsf{T}$ or $u \approx \bot$ or $u \approx v$ of C, for some variable-headed term v.

– BoolRw: Let $t \approx t'$ the equation used among the equations listed for BoolRw. Then we can extend θ' to the variables in t such that the resulting substitution θ' is a unifier of t and u. Therefore, there exists an idempotent $\sigma \in \mathrm{CSU}(t,u)$ such that for some substitution ρ and for all variables x in C, we have $x\sigma\rho = x\theta'$. Thus, there is the following BoolRw inference $\iota' \in \mathit{HInf}$:

$$\frac{C\langle u\rangle}{C\langle t'\rangle\sigma}\ \textsc{BoolRw}$$

Then ι is the $\sigma\rho$-ground instance of ι' and is therefore liftable.

– GForallRw: Then $u\theta = \forall\langle\tau\rangle v$ and the inference ι is of the form

$$\frac{C\theta\langle\forall\langle\tau\rangle v\rangle_p}{C\theta\langle v\,\mathit{GHWit}(C\theta,p)\rangle_p}\ \textsc{GForallRw}$$

for some term v and some type τ.
Let β be a type variable and y a variable of type $\beta \to o$. We define a substitution θ' mapping y to v, β to τ, and all other variables x to $x\theta$. Then $(\forall\langle\beta\rangle y)\theta' = \forall\langle\tau\rangle v = u\theta = u\theta'$ and hence θ' is a unifier of $\forall\langle\beta\rangle y$ and u. Hence, there exists an idempotent $\sigma \in \mathrm{CSU}(\forall\langle\beta\rangle y, u)$ such that for some substitution ρ, for all variables x in C, and for $x \in \{\beta, y\}$, we have $x\sigma\rho = x\theta'$. If $\mathcal{F}(C\theta\langle\mathsf{T}\rangle_p) = \mathcal{F}(C\langle\mathsf{T}\rangle_{p'}\theta)$ is not a tautology, the affected literal in C cannot be of the form $u \approx \mathsf{T}$. Thus there exists the following inference $\iota' \in \mathit{HInf}$:

$$\frac{C\langle u\rangle}{C\langle y\,(\mathsf{sk}_{\Pi\bar\alpha.\forall\bar x.\exists z.\neg(y\sigma\, z)}\langle\bar\alpha\rangle\bar x)\rangle\sigma}\ \textsc{ForallRw}$$

We have $\mathit{GHWit}(C\theta,p) = \mathsf{sk}_{\Pi\bar\alpha.\forall\bar x.\exists z.\neg(y\sigma\, z)}\langle\bar\alpha\rangle\bar x\theta$ by definition of the witness function, where $\bar\alpha$ are the free type variables and $\bar x$ are the free variables occurring in $y\sigma$ in order of first appearance. Hence, ι is the $\sigma\rho$-ground instance of ι' and is therefore liftable.

– GExistsRw: Analogous to GForallRw.

– EqHoist: Let x and y be fresh variables. Then we can extend θ to a x and y such that the resulting substitution θ' is a unifier of u and $x \approx y$. Thus, there exists an idempotent $\sigma \in \mathrm{CSU}(u, x \approx y)$ such that for some substitution ρ, for all variables z in C, and for $z \in \{x, y\}$, we have $z\sigma\rho = z\theta'$. Hence, there is the following EqHoist inference $\iota' \in \mathit{HInf}$:

$$\frac{C\langle u\rangle}{(C\langle \bot\rangle \vee x \approx y)\sigma}\ \textsc{EqHoist}$$

Then ι is the $\sigma\rho$-ground instance of ι' and is therefore liftable.
– NeqHoist: Analogous to EqHoist.
– GForallHoist: Let y be a fresh variable. Then we can extend θ to y such that the resulting substitution θ' is a unifier of u and $\forall\langle\alpha\rangle y$. Thus, there exists an idempotent $\sigma \in \mathrm{CSU}(u, \forall\langle\alpha\rangle y)$ such that for some substitution ρ, for all variables x in C, and for $x = y$, we have $x\sigma\rho = x\theta'$. Thus, there is the following ForallHoist inference $\iota' \in \mathit{HInf}$:

$$\frac{C\langle u\rangle}{(C\langle \bot\rangle \vee y\, x \approx \top)\sigma}\ \textsc{ForallHoist}$$

Then ι is the $\sigma\rho$-ground instance of ι' and is therefore liftable.
– GExistsHoist: Analogous to GForallHoist.

7

Redundant Case: Neither condition (A) nor (B) holds. Then p corresponds to a position in C at or below a fluid term, but p is not the position of v in a literal $v \approx \top$ or $v \approx \bot$. Let $p = p_1.p_2$ such that p_1 is the longest prefix of p that corresponds to a green position p_1' in C. Let $u = C|_{p_1'}$. Let z and x be fresh variables. Define a substitution θ' that maps the variable z to $\lambda y.(u\theta)\langle y\rangle_{p_2}$, the variable x to $u\theta|_{p_2}$, and all other variables w to $w\theta$. Then $(z\,x)\theta' = (u\theta)\langle u\theta|_{p_2}\rangle_{p_2} = u\theta = u\theta'$. So θ' is a unifier of $z\,x$ and u. Thus, there exists an idempotent $\sigma \in \mathrm{CSU}(z\,x, u)$ such that for some substitution ρ, for all variables y in C, and for $y \in \{z, x\}$, we have $y\sigma\rho = y\theta'$. For all of the inference rules, $C\theta|_p = u\theta|_{p_2}$ cannot be $\bot$ or $\top$. Thus, $x\theta' \neq \bot, \top$ and therefore $x\sigma \neq \bot, \top$. Hence, we have $(z\,\bot)\theta' = (u\theta)\langle\bot\rangle_{p_2} \neq (u\theta)\langle x\theta'\rangle_{p_2} = (z\,x)\theta'$ and therefore $(z\,\bot)\sigma \neq (z\,x)\sigma$. Analogously, we have $(z\,\top)\sigma \neq (z\,x)\sigma$. The position p_1 must be eligible in $C\theta$ because p is eligible in $C\theta$ and p_1 is the longest prefix of p that corresponds to a green position p_1' in C. Eligibility of p_1 in $C\theta$ implies eligibility of p_1' in C by Lemma 7.58. Then there are the following inferences ι_{bool} and ι_{loob} from C:

$$\frac{C\langle u\rangle_{p_1'}}{(C\langle z\,\bot\rangle_{p_1'} \vee x \approx \top)\sigma}\ \textsc{FluidBoolHoist}$$

$$\frac{C\langle u\rangle_{p_1'}}{(C\langle z\,\top\rangle_{p_1'} \vee x \approx \bot)\sigma}\ \textsc{FluidLoobHoist}$$

Since N is saturated w.r.t. HInf and $\mathit{HRed}_{\mathrm{I}}$, these inferences are in $\mathit{HRed}_{\mathrm{I}}(N)$. We

have

$$\mathcal{F}((C\langle z\,\bot\rangle_{p'_1} \vee x \approx \mathsf{T})\theta') = \mathcal{F}(C\theta\langle\bot\rangle_p \vee C\theta|_p \approx \mathsf{T})$$
$$\text{and} \quad \mathcal{F}((C\langle z\,\mathsf{T}\rangle_{p'_1} \vee x \approx \bot)\theta') = \mathcal{F}(C\theta\langle\mathsf{T}\rangle_p \vee C\theta|_p \approx \bot)$$

These two clauses entail $\mathcal{F}(C\theta)$. Since p is not the position of v in a literal $v \approx \mathsf{T}$ or $v \approx \bot$, the two clauses are also smaller than $\mathcal{F}(C\theta)$. Since $\iota_{\text{bool}} \in \mathit{HRed}_{\text{I}}(N)$, we have $\mathcal{G}^q(\iota_{\text{bool}}) \subseteq \mathit{GHRed}_{\text{I}}(\mathcal{G}(N))$ and therefore the clauses $\mathcal{F}(\mathcal{G}(N))$ that are smaller than $C\theta' = C\theta$ entail $\mathcal{F}((C\langle z\,\bot\rangle_{p'_1} \vee x \approx \mathsf{T})\theta')$. Similarly, since $\iota_{\text{loob}} \in \mathit{HRed}_{\text{I}}(N)$, we have $\mathcal{G}(\mathit{concl}(\iota_{\text{loob}})) \subseteq \mathcal{G}(N) \cup \mathit{GHRed}_{\text{C}}(\mathcal{G}(N))$. Therefore $\mathcal{F}((C\langle z\,\mathsf{T}\rangle_{p'_1} \vee x \approx \bot)\theta')$ is entailed by clauses in $\mathcal{G}(N)$ that are smaller than or equal to itself. Thus, $C\theta$ is redundant and therefore ι is redundant. Here, it is crucial that we consider inferences with a redundant premise as redundant. □

By the above lemmas, every *HInf* inference is liftable or redundant. Using these lemmas, we can employ Theorem 7.57 to lift ground refutational completeness to nonground refutational completeness.

Lemma 7.62 (Static refutational completeness w.r.t. $\models_{\mathcal{G}}$)**.** *The inference system HInf is statically refutationally complete w.r.t.* $\models_{\mathcal{G}}$ *and* $(\mathit{HRed}_{\text{I}}, \mathit{HRed}_{\text{C}})$*. In other words, if* $N \subseteq \mathcal{C}_{\text{H}}$ *is a clause set saturated w.r.t. HInf and* HRed_{I}*, then* $N \models_{\mathcal{G}} \bot$ *if and only if* $\bot \in N$*.*

Proof. We want to apply Theorem 7.57. GHInf^q is statically refutationally complete for all $q \in Q$ by Theorem 7.53. By Lemmas 7.59, 7.60, and 7.61, for every saturated $N \subseteq \mathcal{C}_{\text{H}}$, there exists $q \in \mathcal{G}(Q)$ such that all inferences $\iota \in \mathit{GHInf}^q$ with $\mathit{prems}(\iota) \in \mathcal{G}(N)$ either are $\mathcal{G}^q$-ground instances of *HInf*-inferences from N or belong to $\mathit{GHRed}^q_{\text{I}}(\mathcal{G}(N))$. Thus, Theorem 7.57 applies. □

Dynamic refutational completeness is easy to derive from static refutational completeness.

Lemma 7.63 (Dynamic refutational completeness w.r.t. $\models_{\mathcal{G}}$)**.** *The inference system HInf is dynamically refutationally complete w.r.t.* $\models_{\mathcal{G}}$ *and* $(\mathit{HRed}_{\text{I}}, \mathit{HRedC})$*, as per Definition 3.26.*

Proof. By Theorem 17 of Waldmann et al., this follows from Lemma 7.62. □

To derive a corresponding result for the entailment relation $\models$, we employ the following lemma, which states equivalence of Herbrand entailment $\models_{\mathcal{G}}$ and Tarski entailment $\models$ on $\mathsf{Q}_{\approx}$-normal clauses.

Lemma 7.64. *Let* $N \subseteq \mathcal{C}_{\text{H}}$ *be* $\mathsf{Q}_{\approx}$*-normal. Then we have* $N \models_{\mathcal{G}} \bot$ *if and only if* $N \models \bot$*.*

Proof. By Lemma 7.27, any model of N is also a model of $\mathcal{G}(N)$. So $N \models_{\mathcal{G}} \bot$ implies $N \models \bot$.

For the other direction, let $\mathcal{I}$ be a model of $\mathcal{G}(N)$. We must show that there exists a model of N. Let $\mathcal{I}'$ be the interpretation obtained from $\mathcal{I}$ by removing all domains that cannot be expressed as $[\![\tau]\!]_{\mathcal{I}_{\text{ty}}}$ for some ground type τ and by removing

all domain elements that cannot be expressed as $[\![t]\!]_{\mathcal{I}}$ for some ground term t. We restrict the type interpretation function $\mathcal{I}_{\mathsf{ty}}$, the interpretation function $\mathcal{I}$, and the λ-designation function $\mathcal{L}$ of $\mathcal{I}$ accordingly.

The restriction $\mathcal{I}'$ of $\mathcal{I}$ still maps the logical symbols correctly: For most logical symbols, this is obvious. Only $\forall$ and $\exists$ deserve some further explanations. For all domains $\mathcal{D}$ of $\mathcal{I}$, we have $\mathcal{I}(\exists, \mathcal{D})(f) = \max\{f(a) \mid a \in \mathcal{D}\}$. For the corresponding domain $\mathcal{D}' \subseteq \mathcal{D}$, if it has not been removed entirely, we have just defined $\mathcal{I}'(\exists, \mathcal{D}')(f) = \mathcal{I}(\exists, \mathcal{D})(f)$. We must show that $\mathcal{I}'(\exists, \mathcal{D}')(f) = \max\{f(a) \mid a \in \mathcal{D}'\}$ for all f that can be expressed as $[\![t]\!]_{\mathcal{I}}$ for some ground term t. This claim can only be violated if there exist $a \in \mathcal{D}$ with $f(a) = 1$ and if all of them have been removed in $\mathcal{D}'$. But we have not removed all such elements a because one of them can be expressed as $[\![\varepsilon\, t]\!]_{\mathcal{I}}$ where t is the ground term such that $f = [\![t]\!]_{\mathcal{I}}$. We can argue similarly for $\forall$.

Clearly, all terms have the same denotation in $\mathcal{I}'$ as in $\mathcal{I}$. Thus, the truth values of ground clauses are identical in $\mathcal{I}$ and $\mathcal{I}'$. Since $\mathcal{I}$ is proper, $\mathcal{I}'$ is also proper. Hence, $\mathcal{I} \models \mathcal{G}(N)$ implies $\mathcal{I}' \models \mathcal{G}(N)$.

It remains to show that $\mathcal{I}' \models N$. Let $C \in N$ and let ξ be a valuation for $\mathcal{I}'$. We must show that C is true in $\mathcal{I}$ under ξ. By assumption, C is $\mathsf{Q}_{\approx}$-normal.

By the construction of $\mathcal{I}'$, there is a grounding substitution θ such that for all type variables α and all term variables x occurring in C, we have $\xi(\alpha) = [\![\alpha\theta]\!]_{\mathcal{I}'_{\mathsf{ty}}}$ and $\xi(x) = [\![x\theta]\!]_{\mathcal{I}'}$. Then, by Lemma 7.26, $[\![t\theta]\!]_{\mathcal{I}'} = [\![t]\!]^{\xi}_{\mathcal{I}'}$ for all subterms t of C. Moreover, we can choose θ such that for all variables x, the only Boolean green subterms of $x\theta$ are $\top$ and $\bot$ and such that $x\theta$ is $\mathsf{Q}_{\approx}$-normal. If $x\theta$ contains a Boolean green subterm different from $\top$ and $\bot$, we can replace it by $\top$ or $\bot$ while preserving its denotation and thus the property that $[\![t\theta]\!]_{\mathcal{I}'} = [\![t]\!]^{\xi}_{\mathcal{I}'}$ for all subterms t of C. If $x\theta$ is not $\mathsf{Q}_{\approx}$-normal, we $\mathsf{Q}_{\approx}$-normalize it, which preserves the denotation of terms by Lemma 7.46.

By Lemma 7.8, it follows that $C\theta$ is $\mathsf{Q}_{\approx}$-normal. Thus, $C\theta \in \mathcal{G}(C)$. Since $\mathcal{I}' \models \mathcal{G}(C)$ and thus $C\theta$ is true in $\mathcal{I}'$, also C is true in $\mathcal{I}'$ under ξ because $[\![t\theta]\!]_{\mathcal{I}'} = [\![t]\!]^{\xi}_{\mathcal{I}'}$ for all subterms t of C. □

Using this lemma, we can derive the following theorem, which is essentially dynamic refutational completeness with the caveat that the initial clause set must be $\mathsf{Q}_{\approx}$-normal, which in practice can be fulfilled by $\mathsf{Q}_{\approx}$-normalizing the input problem in preprocessing.

Theorem 7.65 (Dynamic refutational completeness w.r.t. $\models$)**.** *Let $(N_i)_i$ be a derivation w.r.t. $\mathit{HRed}_{\mathrm{C}}$, as defined in Definition 3.26, such that N_0 is $\mathsf{Q}_{\approx}$-normal and $N_0 \models \bot$. Moreover, assume that $(N_i)_i$ is fair w.r.t. HInf and $\mathit{HRed}_{\mathrm{I}}$. Then we have $\bot \in N_i$ for some i.*

Proof. This is a consequence of Lemmas 7.63 and 7.64. □

To derive a similar result for $\models\!\!\!\approx$, we need the following lemma:

Lemma 7.66. *Let $N_0 \subseteq \mathcal{C}_{\mathrm{H}}$ be a clause set that does not contain any* sk *symbols. If $N_0 \models\!\!\!\approx \bot$, then $N_0 \models \bot$.*

Proof. Equivalently, the lemma statement can be formulated as follows: If N_0 does not have Skolem-aware models, it does not have models at all. We assume that N_0 has a model $\mathcal{I}$ and must show that there exists a Skolem-aware model $\mathcal{I}'$ of N_0.

To transform the model $\mathcal{I} = (\mathcal{I}_{\mathsf{ty}}, \mathcal{J}, \mathcal{L})$ into an interpretation $\mathcal{I}' = (\mathcal{I}'_{\mathsf{ty}}, \mathcal{J}', \mathcal{L}')$, we redefine the interpretation of the Skolem symbol $\mathsf{sk}_{\Pi\bar{\alpha}.\forall\bar{x}.\exists z.\,t\,z} : \Pi\bar{\alpha}.\,\bar{\tau} \to \upsilon$ as follows. Given some domains $\bar{\mathcal{D}}$, let $\xi(\bar{\alpha}) = \bar{\mathcal{D}}$. Then define $\mathcal{J}'(\mathsf{sk}_{\Pi\bar{\alpha}.\forall\bar{x}.\exists z.\,t\,z}, \bar{\mathcal{D}}) = [\![\lambda\bar{x}.\,\varepsilon\langle\upsilon\rangle t]\!]^{\xi}_{\mathcal{I}}$ and $\mathcal{L}'(\xi, \lambda x.s) = \mathcal{L}(\xi, \lambda x.s')$ where s' is obtained from s by replacing each occurrence of a subterm $\mathsf{sk}_{\Pi\bar{\alpha}.\forall\bar{x}.\exists z.\,t\,z}\langle\bar{v}\rangle$ by $(\lambda\bar{x}.\,\varepsilon\langle\upsilon\rangle t)\{\bar{\alpha} \mapsto \bar{v}\}$. This modification of $\mathcal{I}$ yields a new interpretation $\mathcal{I}'$, which is still a model of N_0 because N_0 does not contain any sk symbols. Moreover, it is a Skolem-aware model of N_0 because our redefinition ensures $\mathcal{I}' \models (\mathbf{\exists}\langle\upsilon\rangle(\lambda z.\,t\,z)) \approx t\,(\mathsf{sk}_{\Pi\bar{\alpha}.\forall\bar{x}.\exists z.\,t\,z}\langle\bar{\alpha}\rangle\bar{x})$. □

Using this lemma, we can derive dynamic refutational completeness for $\not\approx$ with additional assumptions on $\mathsf{Q}_{\approx}$-normality and the absence of sk-symbols. These assumptions can be fulfilled by $\mathsf{Q}_{\approx}$-preprocessing and by making the sk symbols internal such that they can not be expressed by the input language of the prover.

Theorem 7.67 (Dynamic refutational completeness w.r.t. $\not\approx$)**.** *Let $(N_i)_i$ be a derivation w.r.t. $\mathit{HRed}_{\mathrm{C}}$, as defined in Definition 3.26, such that N_0 is $\mathsf{Q}_{\approx}$-normal, N_0 does not contain any sk symbols, and $N_0 \not\approx \bot$. Moreover, assume that $(N_i)_i$ is fair w.r.t. HInf and $\mathit{HRed}_{\mathrm{I}}$. Then we have $\bot \in N_i$ for some i.*

Proof. By Lemma 7.66, $N_0 \models \bot$. Hence, Theorem 7.65 applies. □

7.5. Clausification

Our calculus does not require the input problem to be clausified in preprocessing. Instead, it supports higher-order analogues of the three inprocessing clausification methods of my ongoing work with Nummelin, Tourret, and Vukmirović. *Inner delayed clausification* relies on our core calculus rules to destruct logical symbols. *Outer delayed clausification* adds the following clausification rules to the calculus:

$$\frac{s \approx \mathsf{T} \vee C}{\mathit{oc}(s, C)}\ \textsc{PosOuterClaus} \qquad \frac{s \approx \mathbf{\bot} \vee C}{\mathit{oc}(\neg s, C)}\ \textsc{NegOuterClaus}$$

$$\frac{s \approx t \vee C}{s \approx \bot \vee t \approx \top \vee C \quad s \approx \top \vee t \approx \bot \vee C}\ \textsc{EqOuterClaus}$$

$$\frac{s \not\approx t \vee C}{s \approx \bot \vee t \approx \bot \vee C \quad s \approx \top \vee t \approx \top \vee C}\ \textsc{NeqOuterClaus}$$

The double bars mark simplification rules—i.e., the conclusion makes the premise redundant and can replace it. The first two rules require that s has a logical symbol as its head, whereas the last two require that s and t are Boolean terms other than $\top$ and $\bot$. The function oc distributes the logical symbols over the clause C as follows:

$$\mathit{oc}(s \mathbin{\mathbf{\wedge}} t, C) = \{s \approx \top \vee C, t \approx \top \vee C\}$$
$$\mathit{oc}(s \mathbin{\mathbf{\vee}} t, C) = \{s \approx \top \vee t \approx \top \vee C\}$$

$$
\begin{aligned}
oc(s \mathrel{\mathbf{\rightarrow}} t, C) &= \{s \approx \bot \vee t \approx \top \vee C\} \\
oc(s \mathrel{\mathbf{\approx}} t, C) &= \{s \approx t \vee C\} \\
oc(s \mathrel{\mathbf{\not\approx}} t, C) &= \{s \not\approx t \vee C\} \\
oc(\mathbf{\forall}\langle\tau\rangle\, s, C) &= \{s\, y \approx \top \vee C\} \\
oc(\mathbf{\exists}\langle\tau\rangle\, s, C) &= \{s\, (\mathsf{sk}_{\Pi\bar{\alpha}.\forall\bar{x}.\exists z.(s\,z)}\langle\bar{\alpha}\rangle\bar{x}) \approx \top \vee C\} \\
oc(\mathbf{\neg}(s \mathbf{\wedge} t), C) &= \{s \approx \bot \vee t \approx \bot \vee C\} \\
oc(\mathbf{\neg}(s \mathbf{\vee} t), C) &= \{s \approx \bot \vee C, t \approx \bot \vee C\} \\
oc(\mathbf{\neg}(s \mathbf{\rightarrow} t), C) &= \{s \approx \top \vee C, t \approx \bot \vee C\} \\
oc(\mathbf{\neg}(s \mathbf{\approx} t), C) &= \{s \not\approx t \vee C\} \\
oc(\mathbf{\neg}(s \mathbf{\not\approx} t), C) &= \{s \approx t \vee C\} \\
oc(\mathbf{\neg}(\mathbf{\neg} s), C) &= oc(s, C) \\
oc(\mathbf{\neg}(\mathbf{\forall}\langle\tau\rangle\, s), C) &= \{s\, (\mathsf{sk}_{\Pi\bar{\alpha}.\forall\bar{x}.\exists z.\mathbf{\neg}(s\,z)}\langle\bar{\alpha}\rangle\bar{x}) \approx \bot \vee C\} \\
oc(\mathbf{\neg}(\mathbf{\exists}\langle\tau\rangle\, s), C) &= \{s\, y \approx \bot \vee C\}
\end{aligned}
$$

In the equations for $\mathbf{\forall}\langle\tau\rangle s$ and $\mathbf{\neg}(\mathbf{\exists}\langle\tau\rangle s)$, y is a fresh variable. In the equations for $\mathbf{\exists}\langle\tau\rangle s$ and $\mathbf{\neg}(\mathbf{\forall}\langle\tau\rangle s)$, $\bar{\alpha}$ are the free type variables and $\bar{x}$ are the free term variables occurring in s in order of first appearance.

It is easy to check that our redundancy criterion allows us to replace the premise of the OUTERCLAUS rules with their conclusion. Nonetheless, we apply EQOUTERCLAUS and NEQOUTERCLAUS as inferences because the premises might be useful in their original form.

7

Besides the two delayed clausification methods, a third inprocessing clausification method is *immediate* clausification. This clausifies the input problem's outer formula structure in one swoop, resulting in a set of higher-order clauses. If unclausified formulas rise to the top during saturation, the same algorithm is run to clausify them. In contrast to delayed clausification, immediate clausification is monolithic—it behaves as a black-box procedure and it is unaware of the proof state other than the formula it is applied to. Delayed clausification, on the other hand, clausifies formula step by step and at each step full superposition simplification machinery is at disposal.

Many rules of our calculus replace subterms with $\top$ or $\bot$. After such a replacement, the term containing $\top$ or $\bot$ can be simplified using Boolean equivalences that specify the behavior of logical operators on $\top$ and $\bot$. To this end we use the rule BOOLSIMP [138], which resembles the simp rule of Leo-III [125, Section 4.2.1.]:

$$\frac{C[s]}{C[t]}\ \text{BOOLSIMP}$$

This rule replaces s with t whenever $s \mathbf{\approx} t$ is an instance of an equivalence $u \mathbf{\approx} v$. In

addition to all equivalences that Leo-III uses in simp, we also use:

$$(\top \to s) \approx s \qquad (\bot \to s) \approx \top \qquad (s \to \bot) \approx \neg s \qquad (s \to \top) \approx \top$$
$$(s \to \neg s) \approx \neg s \qquad (\neg s \to s) \approx s \qquad (s \to s) \approx \top$$
$$(s_1 \to \cdots \to s_i \to \cdots \to \neg s_i \to \cdots \to t) \approx \top$$
$$(s_1 \to \cdots \to s_n \to t_1 \vee \cdots \vee s_i \vee \cdots \vee t_n) \approx \top$$
$$(s_1 \wedge \cdots \wedge s_n \to t_1 \vee \cdots \vee s_i \vee \cdots \vee t_n) \approx \top$$

It is easy to check that applying any equivalence reduces the size of s w.r.t. the order described in Section 7.3.8, assuming the weight of $\neg$ is not greater than that of $\to$.

With my colleagues, I am working on a more detailed account of this approach to inprocessing clausification in the context of first-order logic with Booleans [109].

7.6. Implementation

We implemented our calculus in Zipperposition. Like the calculus, its implementation is an extension of the implementation of Boolean-free λ-superposition, as presented in Chapter 5, and a preliminary implementation of superposition with Booleans, as presented in Chapter 6. From the former, we inherit the given clause loop which supports enumerating infinitely many inference conclusions, calculus extensions, and higher-order heuristics. From the latter, we inherit the encoding of negative predicate literals as $s \approx \bot$ and a basis for the implementation of FALSEELIM, BOOLRW, FORALLRW, EXISTSRW, and all HOIST rules.

The implementation of superposition with Booleans heavily relies on BOOLSIMP to simplify the proof state. We keep this rule as the basis of our Boolean simplification machinery. This means that BOOLRW can be reduced to two cases: In the first case, all arguments $\bar{s}_n$ of $u = h\,\bar{s}_n$ are variable-headed terms and thus we must unify s_i with all combinations of $\top$ and $\bot$. In the second case, either $u = s \approx t$ or $u = s \not\approx t$ and thus we must compute the unifiers of s and t.

As in the implementation of Boolean-free λ-superposition, we approximate fluid terms as terms that are either nonground λ-expressions or terms of the form $x\,\bar{s}_n$ with $n > 0$. We approximate deeply occurring variables by also counting occurrences below quantifiers as deep. Moreover, we perform EFACT inferences even if the maximal literal is selected. Since we expect FLUIDBOOLHOIST and FLUIDLOOBHOIST to be highly explosive, we penalize their inference streams, as well as conclusions of these inferences. In addition to the extensions of Boolean-free λ-superposition we also use all rules for Boolean reasoning described by Vukmirović and Nummelin [138] except for the BOOLEF rules.

As an optimization for the rules EQHOIST, NEQHOIST, FORALLHOIST, and EXISTSHOIST, if the subterm u is not variable-headed, we observe that there is an obvious most general unifier, which allows us to generate the conclusion directly, without invoking the unification procedure.

7.7. Evaluation

We evaluate our implementation and compare it with other higher-order provers. Our experiments were performed on StarExec Miami servers equipped with Intel Xeon E5-2620 v4 CPUs clocked at 2.10 GHz. We used all 2606 TH0 theorems from the TPTP 7.3.0 library [130] and 1253 "Judgment Day" problems [40] generated using Sledgehammer (SH) [111] as our benchmark set. An archive containing the benchmarks and the raw evaluation results is publicly available.[1] We divide the evaluation in two parts: evaluation of the calculus rules and comparison with other higher-order provers.

Calculus Evaluation In this first part, we evaluate selected parameters of Zipperposition by varying only the studied parameter in a fixed well-performing configuration. This base configuration disables axioms (CHOICE) and (EXT) and the FLUID- rules. It uses the complete unification algorithm of Vukmirović et al. [136]. It uses none of the early Boolean rules described by Vukmirović and Nummelin [138]. The preprocessor $\mathsf{Q}_{\approx}$ is disabled as well. All of the completeness-preserving simplification rules described in Section 7.3.7 are enabled, except for the simplifying BOOLHOIST (combined with LOOBHOIST). The preprocessor $\mathsf{Q}_{\approx}$ is disabled. Finally, the configuration uses immediate clausification. We set the CPU limit to 30 s in each of the three experiments.

In the first experiment, we assess the overhead incurred by our new rules. The FLUID- rules unify with a term whose head is a fresh variable. Thus, we expected that they need to be tightly controlled to achieve good performance. To test our hypothesis, we simultaneously modified the parameters of these three rules. In Figure 7.1, the *off* mode simply disables the rules, the *pragmatic* mode uses a terminating incomplete unification algorithm (the pragmatic variant of Vukmirović et al. [136]), and the *complete* mode uses a complete unification algorithm. The results show that disabling FLUID- rules altogether achieves the best performance. When the complete variant of the unification algorithm is used, inferences are scheduled in a queue designed to postpone explosive inferences, as described in Section 5.6. In contrast, in the pragmatic variant, a terminating algorithm is employed, but still flooding the proof state with FLUID- rules conclusions severely hinders performance. Even though enabling FLUID- rules degrades performance overall, *complete* finds 35 proofs not found by *off*, and *pragmatic* finds 22 proofs not found by *off*. On Sledgehammer benchmarks, this effect is much weaker, likely because the Sledgehammer benchmarks require less higher-order reasoning: *complete* finds only one new proof, and *pragmatic* finds only four.

In the second experiment, we explore the clausification methods introduced at the end of Section 7.3: *inner* delayed clausification, which relies on the core calculus to reason about logical symbols; *outer* delayed clausification, which clausifies step-by-step guided by the outermost logical symbols; and *immediate* clausification, which eagerly applies a monolithic clausification algorithm when encountering top-level logical symbols. The modes *inner* and *outer* employ the RENAME rule developed in my work with Nummelin et al. [109], which renames Boolean terms

[1] https://doi.org/10.5281/zenodo.4534759

	off	*pragmatic*	*complete*
TPTP	**1642**	1591	1619
SH	**467**	431	437

Figure 7.1: Evaluation of explosive calculus rules

	inner	*outer*	*immediate*
TPTP	1323	**1670**	1642
SH	406	**470**	467

Figure 7.2: Evaluation of clausification method

	off	$p = 64$	$p = 16$	$p = 4$	$p = 1$
TPTP	**1642**	1617	1613	1615	1594
SH	**467**	458	458	459	445

Figure 7.3: Evaluation of axiom (CHOICE)

	TPTP	*o*fSH	SH
CVC4	1796	680	619
Leo-III	2104	681	621
Satallax	2162	573	587
Vampire	2131	692	681
Zip	2301	**734**	**736**
New Zip	**2320**	724	720
Leo-III-uncoop	1619	223	240
Satallax-uncoop	2038	467	482
Zip-uncoop	2223	667	673
New Zip-uncoop	2236	640	644

Figure 7.4: Evaluation of all competitive higher-order provers

headed by logical symbols using a Tseitin-like transformation if they occur at least four times in the proof state. Vukmirović and Nummelin [138] observed that *outer* clausification can greatly help prove higher-order problems and we expected it perform well for our calculus, too. The results are shown in Figure 7.2. The results confirm our hypothesis: The *outer* mode outperforms *immediate* on both TPTP and Sledgehammer benchmarks. The *inner* mode performs worst, but on Sledgehammer benchmarks, it proves 17 problems beyond the reach of the other two. Looking at the proofs found by *inner*, we observed a pattern: in many cases (e.g., for the benchmarks `prob_295__3252866_1`, `prob_296__3252872_1`, `prob_366__5338-318_1`, `prob_419__5371618_1`) the problems contain axioms of the form $\phi \to \psi$. When such axioms are not clausified, superposition and demodulation can often reduce either ϕ or ψ to $\top$ or $\bot$. At this point, simplification rules will act on the resulting formula, simplifying it enough that the proof can easily be found.

In the third experiment, we investigate the effect of axiom (CHOICE), which is necessary to achieve refutational completeness. To evaluate (CHOICE), we either disabled it in a configuration labeled *off* or set the axiom's penalty p to different values. In Zipperposition, penalties are propagated through inference and simplification rules and are used to increase the heuristic weight of clauses, postponing the selection of penalized clauses. The results are shown in Figure 7.3. As expected, disabling (CHOICE), or at least penalizing it, improves performance. Yet enabling (CHOICE) can be crucial: For 19 TPTP problems, the proofs are found when (CHOICE) is enabled and $p = 4$, but not when the rule is disabled. On Sledgehammer problems, this effect is weaker, with only two new problems proved for $p = 4$.

Prover Comparison In this second part, we compare Zipperposition's performance with other higher-order provers. Like at CASC-J10, the wall-clock timeout

was 120 s, the CPU timeout was 960 s, and the provers were run on StarExec Miami. We used the following versions of all systems that took part in the THF division: CVC4 1.8 [14], Leo-III 1.5.2 [126], Satallax 3.5 [42], and Vampire 4.5 [28]. The developers of Vampire have informed us that its higher-order schedule is optimized for running on a single core. As a result, the prover suffers some degradation of performance when running on multiple cores. We evaluate both the version of Zipperposition that took part in CASC-J10 (*Zip*) and the updated version of Zipperposition that supports our new calculus (*New Zip*). Zip's portfolio of prover configurations is based on Chapter 5 and techniques described by Vukmirović and Nummelin [138]. New Zip's portfolio is specially designed for our new calculus and optimized for TPTP problems. Leo-III, Satallax, and Zipperposition are cooperative theorem provers: They invoke a backend reasoner to finish the proof attempt. To test the performance of their calculi in isolation, we also invoked them in uncooperative mode. To assess the performance of Boolean reasoning, we used Sledgehammer benchmarks generated both with native Booleans (SH) and with an encoding into Boolean-free higher-order logic (*o*fSH). For technical reasons, the encoding also performs λ-lifting, but this minor transformation should have little impact on results as our evaluation from Chapter 5 indicates.

The results are shown in Figure 7.4. The updated version of New Zip beats Zip on TPTP problems but lags behind Zip on Sledgehammer benchmarks as we have yet to further explore more general heuristics for our new calculus. The Sledgehammer benchmarks fail to demonstrate the superiority of native Booleans reasoning compared with an encoding, and in fact CVC4 and Leo-III perform dramatically better on the encoded Boolean problems, suggesting that there is room for tuning.

The uncooperative versions of Zipperposition show strong performance on both benchmark sets. This suggests that, with thorough parameter tuning, higher-order superposition outperforms tableaux, which had been the state of the art in higher-order reasoning for a decade. Without backend reasoners, Zipperposition proves fewer Sledgehammer problems than Vampire. We conjecture that implementing our calculus in Vampire or E would remove the need for a backend reasoner and make the calculus even more useful in practice.

7.8. Conclusion

We have created a superposition calculus for higher-order logic and proved it to be sound and refutationally complete. Most of the key ideas have been developed in the previous chapters, but combining them in the right way has been a challenging task. A key idea has been to eliminate quantified terms with inconvenient higher-order features by $\mathrm{Q}_{\approx}$-normalization. Unlike earlier refutationally complete calculi for full higher-order logic based on resolution or paramodulation, our calculus employs a term order to steer proof search. Based on the term order, we can specify a redundancy criterion that allows us to add various simplification rules without compromising refutational completeness. We believe that this is a key ingredient to the efficiency of the calculus.

The evaluation results show that our calculus is an excellent basis for higher-order theorem proving. In future work, we will further experiment with the different parameters of the calculus, for instance with Boolean subterm selection heuristics.

8

Conclusion

We have developed refutationally complete calculi for various formalisms between first-order and higher-order logic, most notably a calculus for full higher-order logic essentially as described by the TPTP TH1 standard. We have implemented all of these calculi in the Zipperposition prover and evaluated them. The results show that the implementation of our calculus for higher-order logic with some incomplete optimizations outperforms all other modern provers and can thus substantially improve the performance of Sledgehammer and similar applications by avoiding encodings into first-order logic. The work was acclaimed in the higher-order theorem proving community and reveals many promising directions for future work.

8.1. Results and Impact

For many years, the development of automated reasoning has been divided in two branches: first-order reasoning and higher-order reasoning. The simpler term structure and semantics of first-order logic allowed the first-order logic community to develop highly efficient calculi. The superposition calculus emerged already in the early 1990s and is arguably still the most efficient calculus for first-order logic with equality today. The different nature of higher-order logic—characterized by nonterminating unification, lack of a simple clausification procedure, and $\beta\eta$-conversion—deterred researchers from investigating generalizations to higher-order logic.

The higher-order logic camp developed various strategies to keep their logic's high expressivity under control. Unfortunately many concepts that led to success in first-order logic, such as term orders and redundancy criteria, seemed not to be applicable to the higher-order calculi.

The work presented in this thesis brings these two worlds together. The guiding principle has been that our calculus should operate on higher-order logic without giving up what has been successful for first-order logic since the 1990s—we always aimed for a *graceful* generalization. This principle has payed off: Since higher-order problems always have some first-order component, our prover's stable first-order foundation allows it to outperform all other higher-order provers.

To achieve the goal of a higher-order superposition calculus, it was crucial to divide the task into milestones, represented by intermediate logics. The main concern of the first milestone, our calculi for λ-free higher-order logic, was how to support nonmonotonic term orders. In this initial work, we have developed the approach of dividing the completeness proof into three levels, nested like Russian dolls: the ground first-order level GF, the ground higher-order level GH, and the nonground higher-order layer H. This extends the common approach of dividing completeness proofs into a ground and a nonground level. The three-level approach combines the best of two worlds: The encoding into first-order logic allows us to reuse established first-order concepts such as first-order term rewriting systems in the completeness proof, and yet the implementation operates directly on higher-order logic without any encodings. We found this approach useful for all three milestones.

In parallel, Vukmirović et al. [137] have implemented support for λ-free higher-order logic in the E prover, but only for monotone term orders. Implementing our calculi for nonmonotone orders in a highly optimized prover such as E seemed to be a daunting task, which led me to develop the monotone order EPO as a possible replacement for the nonmonotone RPO. However, in the long run, support for nonmonotone orders is unavoidable, in particular regarding extensions to full higher-order logic.

Next, we have developed Boolean-free λ-superposition calculus. Our central discovery in this step was that we can eagerly normalize terms into η-short β-normal form and still obtain a refutationally complete calculus. Unlike other higher-order calculi, our approach relies on a full unification procedure; preunification is not sufficient. Therefore, Vukmirović, Nummelin, and I have developed an efficient higher-order unification procedure [136], improving on Jensen and Pietrzykowski's procedure [76]. Based on our calculus and an early version of our procedure, Zipper-

position achieved the third place in the higher-order category of the CASC-27 prover competition in 2019.

Shortly after the development of Boolean-free λ-superposition, Bhayat and Reger [28] developed the closely related combinatory superposition calculus. It is modeled on our intensional nonpurifying λ-free calculus and targets extensional polymorphic clausal higher-order logic. Both combinatory and λ-superposition gracefully generalize the highly successful first-order superposition rules without sacrificing refutational completeness, and both are equipped with a redundancy criterion. Combinatory superposition's distinguishing feature is that it uses SK combinators to represent λ-expressions. Combinators can be implemented more easily starting from a first-order prover; β-reduction amounts to demodulation. However, according to its developers, "Narrowing terms with combinator axioms is still explosive and results in redundant clauses. It is also never likely to be competitive with higher-order unification in finding complex unifiers." Among the drawbacks of λ-superposition are the need to solve flex–flex pairs eagerly and the explosion caused by the extensionality axiom. We believe that this is a reasonable trade-off, especially for large problems with a substantial first-order component.

Based on the Boolean-free λ-superposition calculus, Vukmirović and Nummelin [138] have presented a pragmatic approach to Boolean reasoning for higher-order logic and have implemented it in Zipperposition. These efforts led Zipperposition to win the higher-order category of the CASC-J10 competition in 2020. Due to this success, the prover has gained recognition in the community and as a result, Zipperposition is included in the 2021 version of the Isabelle proof assistant.

In preparation for the final milestone, Nummelin, Tourret, Vukmirović, and I have developed a calculus for first-order logic with an interpreted Boolean type. Using the ground version of this calculus as the GF layer has then allowed us to extend the Boolean-free λ-superposition calculus with Booleans, finally yielding the λ-superposition calculus, a refutationally complete superposition calculus for higher-order logic.

8.2. Future Work

In the past, first-order provers have been generalized to richer and richer logics. For instance, many provers have been extended with support for sorts and some have been extended with polymorphism. Our work on higher-order superposition in Zipperposition, along with the higher-order extensions of Vampire and of the SMT solvers CVC4 and veriT, is a further significant step towards richer logics. From Isabelle user's informal feedback, we hear that such developments lead to an appreciable enhancement of Sledgehammer. Thus I expect that this trend towards richer logics will continue in the future.

One direction in which we could extend our superposition calculus further is the **inclusion of theories** such as arithmetic. For first-order logic, theory reasoning can be achieved with hierarchic superposition [15], and hopefully its principles can also be applied in higher-order logic.

A second direction is an extension of superposition to **dependent type theory**. Many popular proof assistants such as Agda, Coq, and Lean are based on

dependently-typed logics. The core feature of dependent type theory is that types can depend on terms. For instance, we can define a type vector n, representing vectors of length n where n is a term. To improve hammers for such systems, it would be desirable to extend superposition to dependent type theory, too. The state of the art is to translate into first-order logic via an unsound and incomplete encoding [49].

We have various ideas to **further improve our higher-order superposition calculus**. The evaluations show that some mechanisms required by the completeness proofs such as the extensionality axiom and FLUIDSUP do not help in practice. We want to explore other approaches to replace these mechanisms.

Possibly, the (EXT) axiom can be replaced by more restricted inference rules resembling λSUP without compromising refutational completeness. The FLUIDSUP rule could be improved by devising a specific unification procedure that produces only unifiers that are actually required in the part of the completeness proof discussing FLUIDSUP. We have not spent a lot of effort to avoid the inclusion of the (CHOICE) axiom in our calculus. Possibly, it can be replaced by more efficient calculus rules.

The need of the calculus to solve the full unification problem might also hinder performance. Existing procedures [76, 123], including the one I developed with Vukmirović and Nummelin [137], enumerate redundant unifiers. This can probably be avoided to some extent. It could also be useful to investigate unification procedures that would delay imitation/projection choices via special schematic variables, inspired by Libal's representation of regular unifiers [99].

Alternatively, it seems possible to modify the calculus to work with preunification and constraints, similar to Huet's resolution calculus [73]. This would avoid the need for full unification altogether.

Eventually, the our generalization of superposition should be **implemented in a high performance prover**. Zipperposition has been a convenient vehicle for experimenting and prototyping because it is easier to understand and modify than highly optimized C or C++ provers. Arguably, implementing our calculi in modern first-order provers such as E [119], SPASS [142], and Vampire [94] would lead to even better results. The main challenge might be to extend the internal data structures of such provers to allow for λ-expressions. A second challenge is to extend the prover with polymorphism, which has already been done in Vampire [29].

Vukmirović, Blanchette, and Schulz have concrete plans to implement a variant of our higher-order calculus based on Ehoh [137], the λ-free clausal higher-order version of E. With its stratified architecture, Otter-λ [17] is perhaps the closest to what they are aiming at, with the difference that Otter-λ is limited to second-order logic and offers no completeness guarantees.

Finally, **heuristics** will be a fruitful area for future research in higher-order reasoning. Proof assistants are an inexhaustible source of easy-looking benchmarks that are beyond the power of today's provers. Whereas "hard higher-order" may remain forever out of reach, there is a substantial "easy higher-order" fragment that awaits automation. By studying the behavior of our calculus on supposedly easy problems originating from proof assistants, we will probably be able to improve the success rate of hammers substantially.

References

[1] Andrews, P.B.: Resolution in type theory. J. Symb. Log. 36(3), 414–432 (1971)

[2] Andrews, P.B.: On connections and higher-order logic. J. Autom. Reason. 5(3), 257–291 (1989)

[3] Andrews, P.B.: Classical type theory. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. II, pp. 965–1007. Elsevier and MIT Press (2001)

[4] Andrews, P.B., Bishop, M., Issar, S., Nesmith, D., Pfenning, F., Xi, H.: TPS: A theorem-proving system for classical type theory. J. Autom. Reason. 16(3), 321–353 (1996)

[5] Asperti, A., Tassi, E.: Superposition as a logical glue. In: Hirschowitz, T. (ed.) TYPES 2009. EPTCS, vol. 53, pp. 1–15 (2009)

[6] Asperti, A., Tassi, E.: Smart matching. In: Autexier, S., Calmet, J., Delahaye, D., Ion, P.D.F., Rideau, L., Rioboo, R., Sexton, A.P. (eds.) CICM 2010. LNCS, vol. 6167, pp. 263–277. Springer (2010)

[7] Avenhaus, J., Denzinger, J., Fuchs, M.: DISCOUNT: A system for distributed equational deduction. In: Hsiang, J. (ed.) RTA-95. LNCS, vol. 914, pp. 397–402. Springer (1995)

[8] Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press (1998)

[9] Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. J. Log. Comput. 4(3), 217–247 (1994)

[10] Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. I, pp. 19–99. Elsevier and MIT Press (2001)

[11] Bachmair, L., Ganzinger, H., Lynch, C., Snyder, W.: Basic paramodulation and superposition. In: Kapur, D. (ed.) CADE-11. LNCS, vol. 607, pp. 462–476. Springer (1992)

[12] Backes, J., Brown, C.E.: Analytic tableaux for higher-order logic with choice. J. Autom. Reason. 47(4), 451–479 (2011)

[13] Barbosa, H., Reynolds, A., Ouraoui, D.E., Tinelli, C., Barrett, C.W.: Extending SMT solvers to higher-order logic. In: Fontaine, P. (ed.) CADE-27. LNCS, vol. 11716, pp. 35–54. Springer (2019)

[14] Barrett, C.W., Conway, C.L., Deters, M., Hadarean, L., Jovanovic, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: CAV 2011. LNCS, vol. 6806, pp. 171–177. Springer (2011)

[15] Baumgartner, P., Waldmann, U.: Hierarchic superposition with weak abstraction. In: Bonacina, M.P. (ed.) CADE-24. LNCS, vol. 7898, pp. 39–57. Springer (2013)

[16] Becker, H., Blanchette, J.C., Waldmann, U., Wand, D.: A transfinite Knuth–Bendix order for lambda-free higher-order terms. In: de Moura, L. (ed.) CADE-26. LNCS, vol. 10395, pp. 432–453. Springer (2017)

[17] Beeson, M.: Lambda logic. In: Basin, D.A., Rusinowitch, M. (eds.) IJCAR 2004. LNCS, vol. 3097, pp. 460–474. Springer (2004)

[18] Bentkamp, A.: Formalization of the embedding path order for lambda-free higher-order terms. Archive of Formal Proofs (2018), http://isa-afp.org/entries/Lambda_Free_EPO.html

[19] Benzmüller, C.: Extensional higher-order paramodulation and RUE-resolution. In: Ganzinger, H. (ed.) CADE-16. LNCS, vol. 1632, pp. 399–413. Springer (1999)

[20] Benzmüller, C., Kohlhase, M.: Extensional higher-order resolution. In: Kirchner, C., Kirchner, H. (eds.) CADE-15. LNCS, vol. 1421, pp. 56–71. Springer (1998)

[21] Benzmüller, C., Miller, D.: Automation of higher-order logic. In: Siekmann, J.H. (ed.) Computational Logic, Handbook of the History of Logic, vol. 9, pp. 215–254. Elsevier (2014)

[22] Benzmüller, C., Paulson, L.C.: Multimodal and intuitionistic logics in simple type theory. Log. J. IGPL 18(6), 881–892 (2010)

[23] Benzmüller, C., Paulson, L.C., Theiss, F., Fietzke, A.: LEO-II—A cooperative automatic theorem prover for higher-order logic. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS, vol. 5195, pp. 162–170. Springer (2008)

[24] Benzmüller, C., Sultana, N., Paulson, L.C., Theiss, F.: The higher-order prover LEO-II. J. Autom. Reason. 55(4), 389–404 (2015)

[25] Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science, Springer (2004)

[26] Bhayat, A.: Automated Theorem Proving in Higher-Order Logic. Ph.D. thesis, University of Manchester (2020)

[27] Bhayat, A., Reger, G.: Restricted combinatory unification. In: Fontaine, P. (ed.) CADE-27. LNCS, vol. 11716, pp. 74–93. Springer (2019)

[28] Bhayat, A., Reger, G.: A combinator-based superposition calculus for higher-order logic. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) IJCAR 2020, Part I. LNCS, vol. 12166, pp. 278–296. Springer (2020)

[29] Bhayat, A., Reger, G.: A polymorphic vampire - (short paper). In: Peltier, N., Sofronie-Stokkermans, V. (eds.) IJCAR 2020. LNCS, vol. 12167, pp. 361–368. Springer (2020)

[30] Blanchette, J.C., Böhme, S., Popescu, A., Smallbone, N.: Encoding monomorphic and polymorphic types. Log. Meth. Comput. Sci. 12(4) (2016)

[31] Blanchette, J.C., Kaliszyk, C., Paulson, L.C., Urban, J.: Hammering towards QED. J. Formaliz. Reas. 9(1), 101–148 (2016)

[32] Blanchette, J.C., Paskevich, A.: TFF1: The TPTP typed first-order form with rank-1 polymorphism. In: Bonacina, M.P. (ed.) CADE-24. LNCS, vol. 7898, pp. 414–420. Springer (2013)

[33] Blanchette, J.C., Waldmann, U., Wand, D.: A lambda-free higher-order recursive path order. Tech. report, http://people.mpi-inf.mpg.de/~jblanche/lambda_free_rpo_rep.pdf (2016)

[34] Blanchette, J.C., Waldmann, U., Wand, D.: A lambda-free higher-order recursive path order. In: Esparza, J., Murawski, A.S. (eds.) FoSSaCS 2017. LNCS, vol. 10203, pp. 461–479. Springer (2017)

[35] Blanqui, F.: Higher-order dependency pairs. CoRR abs/1804.08855 (2018)

[36] Blanqui, F., Jouannaud, J.P., Rubio, A.: The computability path ordering. Log. Meth. Comput. Sci. 11(4) (2015)

[37] Bobot, F., Paskevich, A.: Expressing polymorphic types in a many-sorted language. In: Tinelli, C., Sofronie-Stokkermans, V. (eds.) FroCoS 2011. LNCS, vol. 6989, pp. 87–102. Springer (2011)

[38] Bofill, M., Borralleras, C., Rodríguez-Carbonell, E., Rubio, A.: The recursive path and polynomial ordering for first-order and higher-order terms. J. Log. Comput. 23(1), 263–305 (2013)

[39] Bofill, M., Rubio, A.: Paramodulation with non-monotonic orderings and simplification. J. Autom. Reason. 50(1), 51–98 (2013)

[40] Böhme, S., Nipkow, T.: Sledgehammer: Judgement Day. In: Giesl, J., Hähnle, R. (eds.) IJCAR 2010. LNCS, vol. 6173, pp. 107–121. Springer (2010)

[41] Brand, D.: Proving theorems with the modification method. SIAM J. Comput. 4, 412–430 (1975)

[42] Brown, C.E.: Satallax: An automatic higher-order prover. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS, vol. 7364, pp. 111–117. Springer (2012)

[43] de Bruijn, N.G.: Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church–Rosser theorem. Indag. Math 75(5), 381–392 (1972)

[44] Cervesato, I., Pfenning, F.: A linear spine calculus. J. Log. Comput. 13(5), 639–688 (2003)

[45] Church, A.: A formulation of the simple theory of types. J. Symb. Log. 5(2), 56–68 (1940)

[46] Cruanes, S.: Extending Superposition with Integer Arithmetic, Structural Induction, and Beyond. Ph.D. thesis, École polytechnique (2015)

[47] Cruanes, S.: Superposition with structural induction. In: Dixon, C., Finger, M. (eds.) FroCoS 2017. LNCS, vol. 10483, pp. 172–188. Springer (2017)

[48] Czajka, Ł.: Improving automation in interactive theorem provers by efficient encoding of lambda-abstractions. In: Avigad, J., Chlipala, A. (eds.) CPP 2016. pp. 49–57. ACM (2016)

[49] Czajka, Ł., Kaliszyk, C.: Hammer for Coq: Automation for dependent type theory. J. Autom. Reason. 61(1-4), 423–453 (2018)

[50] Dershowitz, N., Manna, Z.: Proving termination with multiset orderings. Commun. ACM 22(8), 465–476 (1979)

[51] Digricoli, V.J., Harrison, M.C.: Equality-based binary resolution. J. ACM 33(2), 253–289 (1986)

[52] Dougherty, D.J.: Higher-order unification via combinators. Theor. Comput. Sci. 114(2), 273–298 (1993)

[53] Dowek, G.: Higher-order unification and matching. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. II, pp. 1009–1062. Elsevier and MIT Press (2001)

[54] Dowek, G., Hardin, T., Kirchner, C.: Higher-order unification via explicit substitutions (extended abstract). In: LICS '95. pp. 366–374. IEEE (1995)

[55] Eguchi, N.: A lexicographic path order with slow growing derivation bounds. Math. Log. Q. 55(2), 212–224 (2009)

[56] Ferreira, M.C.F., Zantema, H.: Well-foundedness of term orderings. In: Dershowitz, N., Lindenstrauss, N. (eds.) CTRS-94. LNCS, vol. 968, pp. 106–123. Springer (1994)

[57] Filliâtre, J., Paskevich, A.: Why3 - where programs meet provers. In: Felleisen, M., Gardner, P. (eds.) ESOP 2013. LNCS, vol. 7792, pp. 125–128. Springer (2013)

[58] Fitting, M.: First-Order Logic and Automated Theorem Proving. Springer-Verlag, 2nd edn. (1996)

[59] Fitting, M.: Types, Tableaus, and Gödel's God. Kluwer (2002)

[60] Fuhs, C., Kop, C.: A static higher-order dependency pair framework. In: Caires, L. (ed.) ESOP 2019. LNCS, vol. 11423, pp. 752–782. Springer (2019)

[61] Ganzinger, H., Stuber, J.: Superposition with equivalence reasoning and delayed clause normal form transformation. Information and Computation 199(1–2), 3–23 (2005)

[62] Ganzinger, H., Stuber, J.: Superposition with equivalence reasoning and delayed clause normal form transformation. Inf. Comput. 199(1-2), 3–23 (2005)

[63] Giesl, J., Rubio, A., Sternagel, C., Waldmann, J., Yamada, A.: The termination and complexity competition. In: Beyer, D., Huisman, M., Kordon, F., Steffen, B. (eds.) TACAS 2019. LNCS, vol. 11429, pp. 156–166. Springer (2019)

[64] Giesl, J., Thiemann, R., Schneider-Kamp, P., Falke, S.: Mechanizing and improving dependency pairs. J. Autom. Reason. 37(3), 155–203 (2006)

[65] Gonthier, G.: Formal proof—The four-color theorem. Notices of the AMS 55(11), 1382–1393 (2008)

[66] Gonthier, G., Asperti, A., Avigad, J., Bertot, Y., Cohen, C., Garillot, F., Le Roux, S., Mahboubi, A., O'Connor, R., Biha, S.O., et al.: A machine-checked proof of the odd order theorem. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) ITP 2013. LNCS, vol. 7998, pp. 163–179. Springer (2013)

[67] Gordon, M.J.C., Melham, T.F. (eds.): Introduction to HOL: A Theorem Proving Environment for Higher Order Logic. Cambridge University Press (1993)

[68] Gupta, A., Kovács, L., Kragl, B., Voronkov, A.: Extensional crisis and proving identity. In: Cassez, F., Raskin, J. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 185–200. Springer (2014)

[69] Hales, T.C.: Developments in formal proofs. CoRR abs/1408.6474 (2014)

[70] Hales, T.C., Adams, M., Bauer, G., Dang, D.T., Harrison, J., Hoang, T.L., Kaliszyk, C., Magron, V., McLaughlin, S., Nguyen, T.T., Nguyen, T.Q., Nipkow, T., Obua, S., Pleso, J., Rute, J.M., Solovyev, A., Ta, A.H.T., Tran, T.N., Trieu, D.T., Urban, J., Vu, K.K., Zumkeller, R.: A formal proof of the Kepler conjecture. CoRR abs/1501.02155 (2015)

[71] Henkin, L.: Completeness in the theory of types. J. Symb. Log. 15(2), 81–91 (1950)

[72] Hirokawa, N., Middeldorp, A., Zankl, H.: Uncurrying for termination and complexity. J. Autom. Reason. 50(3), 279–315 (2013)

[73] Huet, G.P.: A mechanization of type theory. In: Nilsson, N.J. (ed.) IJCAI-73. pp. 139–146. William Kaufmann (1973)

[74] Huet, G.P.: A unification algorithm for typed lambda-calculus. Theor. Comput. Sci. 1(1), 27–57 (1975)

[75] Hurd, J.: First-order proof tactics in higher-order logic theorem provers. In: Archer, M., Di Vito, B., Muñoz, C. (eds.) Design and Application of Strategies/Tactics in Higher Order Logics. pp. 56–68. NASA Technical Reports (2003)

[76] Jensen, D.C., Pietrzykowski, T.: Mechanizing ω-order type theory through unification. Theor. Comput. Sci. 3(2), 123–171 (1976)

[77] Jouannaud, J.P., Rubio, A.: Rewrite orderings for higher-order terms in eta-long beta-normal form and recursive path ordering. Theor. Comput. Sci. 208(1–2), 33–58 (1998)

[78] Jouannaud, J.P., Rubio, A.: Polymorphic higher-order recursive path orderings. J. ACM 54(1), 2:1–2:48 (2007)

[79] Kaliszyk, C., Sutcliffe, G., Rabe, F.: TH1: The TPTP typed higher-order form with rank-1 polymorphism. In: Fontaine, P., Schulz, S., Urban, J. (eds.) PAAR-2016. CEUR Workshop Proceedings, vol. 1635, pp. 41–55. CEUR-WS.org (2016)

[80] Kaliszyk, C., Urban, J.: Learning-assisted automated reasoning with Flyspeck. J. Autom. Reason. 53(2), 173–213 (2014)

[81] Kaliszyk, C., Urban, J.: HOL(y)Hammer: Online ATP service for HOL Light. Math. Comput. Sci. 9(1), 5–22 (2015)

[82] Kamin, S., Lévy, J.J.: Two generalizations of the recursive path ordering. Unpublished manuscript, University of Illinois (1980)

[83] Kennaway, R., Klop, J.W., Sleep, M.R., de Vries, F.: Comparing curried and uncurried rewriting. J. Symb. Comput. 21(1), 15–39 (1996)

[84] Kerber, M.: How to prove higher order theorems in first order logic. In: Mylopoulos, J., Reiter, R. (eds.) IJCAI-91. pp. 137–142. Morgan Kaufmann (1991)

[85] Kőnig, D.: Über eine Schlussweise aus dem Endlichen ins Unendliche. Acta Sci. Math. (Szeged) 3499/2009(3:2–3), 121–130 (1927)

[86] Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., Winwood, S.: seL4: formal verification of an OS kernel. In: Matthews, J.N., Anderson, T.E. (eds.) SOSP 2009. pp. 207–220. ACM (2009)

[87] Knuth, D.E., Bendix, P.B.: Simple word problems in universal algebras. In: Leech, J. (ed.) Computational Problems in Abstract Algebra. pp. 263–297. Pergamon Press (1970)

[88] Kohlhase, M.: Higher-order tableaux. In: Baumgartner, P., Hähnle, R., Posegga, J. (eds.) TABLEAUX '95. LNCS, vol. 918, pp. 294–309. Springer (1995)

[89] Konrad, K.: HOT: A concurrent automated theorem prover based on higher-order tableaux. In: Grundy, J., Newey, M.C. (eds.) TPHOLs '98. LNCS, vol. 1479, pp. 245–261. Springer (1998)

[90] Kop, C.: Higher Order Termination: Automatable Techniques for Proving Termination of Higher-Order Term Rewriting Systems. Ph.D. thesis, Vrije Universiteit Amsterdam (2012)

[91] Kop, C., van Raamsdonk, F.: A higher-order iterative path ordering. In: LPAR 2008. pp. 697–711 (2008)

[92] Kotelnikov, E., Kovács, L., Suda, M., Voronkov, A.: A clausal normal form translation for FOOL. In: Benzmüller, C., Sutcliffe, G., Rojas, R. (eds.) GCAI 2016. EPiC, vol. 41, pp. 53–71. EasyChair (2016)

[93] Kotelnikov, E., Kovács, L., Voronkov, A.: A first class Boolean sort in first-order theorem proving and TPTP. In: Kerber, M., Carette, J., Kaliszyk, C., Rabe, F., Sorge, V. (eds.) CICM 2015. LNCS, vol. 9150, pp. 71–86. Springer (2015)

[94] Kovács, L., Voronkov, A.: First-order theorem proving and Vampire. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 1–35. Springer (2013)

[95] Kusakari, K.: On proving termination of term rewriting systems with higher-order variables. IPSJ Transactions on Programming 42(7), 35–45 (2001)

[96] Kusakari, K., Sakai, M.: Static dependency pair method for simply-typed term rewriting and related techniques. IEICE Transactions 92-D(2), 235–247 (2009)

[97] Leivant, D.: Higher order logic. In: Gabbay, D.M., Hogger, C.J., Robinson, J.A., Siekmann, J.H. (eds.) Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 2, Deduction Methodologies, pp. 229–322. Oxford University Press (1994)

[98] Leroy, X.: Formal verification of a realistic compiler. Communications of the ACM 52(7), 107–115 (2009)

[99] Libal, T.: Regular patterns in second-order unification. In: Felty, A.P., Middeldorp, A. (eds.) CADE-25. LNCS, vol. 9195, pp. 557–571. Springer (2015)

[100] Lifantsev, M., Bachmair, L.: An LPO-based termination ordering for higher-order terms without λ-abstraction. In: Grundy, J., Newey, M.C. (eds.) TPHOLs '98. LNCS, vol. 1479, pp. 277–293. Springer (1998)

[101] Lindblad, F.: A focused sequent calculus for higher-order logic. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) IJCAR 2014. LNCS, vol. 8562, pp. 61–75. Springer (2014)

[102] Löchner, B.: Things to know when implementing KBO. J. Autom. Reason. 36(4), 289–310 (2006)

[103] Löchner, B.: Things to know when implementing LPO. Internat. J. Artificial Intelligence Tools 15(1), 53–80 (2006)

[104] Ludwig, M., Waldmann, U.: An extension of the Knuth-Bendix ordering with LPO-like properties. In: Dershowitz, N., Voronkov, A. (eds.) LPAR 2007. LNCS, vol. 4790, pp. 348–362. Springer (2007)

[105] Mayr, R., Nipkow, T.: Higher-order rewrite systems and their confluence. Theor. Comput. Sci. 192(1), 3–29 (1998)

[106] Meng, J., Paulson, L.C.: Translating higher-order clauses to first-order clauses. J. Autom. Reason. 40(1), 35–60 (2008)

[107] Miller, D.A.: A compact representation of proofs. Studia Logica 46(4), 347–370 (1987)

[108] Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)

[109] Nummelin, V., Bentkamp, A., Tourret, S., Vukmirović, P.: Superposition with first-class Booleans and inprocessing clausification, submitted

[110] Obermeyer, F.H.: Automated Equational Reasoning in Nondeterministic λ-Calculi Modulo Theories $\mathcal{H}^*$. Ph.D. thesis, Carnegie Mellon University (2009)

[111] Paulson, L.C., Blanchette, J.C.: Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In: Sutcliffe, G., Schulz, S., Ternovska, E. (eds.) IWIL-2010. EPiC, vol. 2, pp. 1–11. EasyChair (2012)

[112] Peltier, N.: A variant of the superposition calculus. Archive of Formal Proofs (2016), https://www.isa-afp.org/entries/SuperCalc.shtml

[113] Reich, J.S., Naylor, M., Runciman, C.: Advances in Lazy SmallCheck. In: Hinze, R. (ed.) IFL. LNCS, vol. 8241, pp. 53–70. Springer (2012)

[114] Robinson, J.: Mechanizing higher order logic. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence, vol. 4, pp. 151–170. Edinburgh University Press (1969)

[115] Robinson, J.: A note on mechanizing higher order logic. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence, vol. 5, pp. 121–135. Edinburgh University Press (1970)

[116] Schmidt-Schauß, M.: Unification in a combination of arbitrary disjoint equational theories. J. Symb. Comput. 8, 51–99 (1989)

[117] Schulz, S.: E - a brainiac theorem prover. AI Commun. 15(2-3), 111–126 (2002)

[118] Schulz, S.: Fingerprint indexing for paramodulation and rewriting. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS, vol. 7364, pp. 477–483. Springer (2012)

[119] Schulz, S.: System description: E 1.8. In: McMillan, K.L., Middeldorp, A., Voronkov, A. (eds.) LPAR-19. LNCS, vol. 8312, pp. 735–743. Springer (2013)

[120] Schulz, S., Cruanes, S., Vukmirović, P.: Faster, higher, stronger: E 2.3. In: Fontaine, P. (ed.) CADE-27. LNCS, vol. 11716, pp. 495–507. Springer (2019)

[121] Schulz, S., Sutcliffe, G., Urban, J., Pease, A.: Detecting inconsistencies in large first-order knowledge bases. In: de Moura, L. (ed.) CADE-26. LNCS, vol. 10395, pp. 310–325. Springer (2017)

[122] Snyder, W.: Higher order E-unification. In: Stickel, M.E. (ed.) CADE-10. LNCS, vol. 449, pp. 573–587. Springer (1990)

[123] Snyder, W., Gallier, J.H.: Higher-order unification revisited: Complete sets of transformations. J. Symb. Comput. 8(1/2), 101–140 (1989)

[124] Snyder, W., Lynch, C.: Goal directed strategies for paramodulation. In: Book, R.V. (ed.) RTA-91. LNCS, vol. 488, pp. 150–161. Springer (1991)

[125] Steen, A.: Extensional Paramodulation for Higher-order Logic and Its Effective Implementation Leo-III. Dissertationen zur künstlichen Intelligenz, Akademische Verlagsgesellschaft AKA GmbH (2018)

[126] Steen, A., Benzmüller, C.: The higher-order prover Leo-III. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) IJCAR 2018. LNCS, vol. 10900, pp. 108–116. Springer (2018)

[127] Stump, A., Sutcliffe, G., Tinelli, C.: StarExec: A cross-community infrastructure for logic solving. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) IJCAR 2014. LNCS, vol. 8562, pp. 367–373. Springer (2014)

[128] Sultana, N., Blanchette, J.C., Paulson, L.C.: LEO-II and Satallax on the Sledgehammer test bench. J. Applied Logic 11(1), 91–102 (2013)

[129] Sutcliffe, G.: The CADE-26 automated theorem proving system competition—CASC-26. AI Commun. 30(6), 419–432 (2017)

[130] Sutcliffe, G.: The TPTP problem library and associated infrastructure—from CNF to TH0, TPTP v6.4.0. J. Autom. Reason. 59(4), 483–502 (2017)

[131] Sutcliffe, G., Benzmüller, C., Brown, C.E., Theiss, F.: Progress in the development of automated theorem proving for higher-order logic. In: Schmidt, R.A. (ed.) CADE-22. LNCS, vol. 5663, pp. 116–130. Springer (2009)

[132] Sutcliffe, G., Schulz, S., Claessen, K., Baumgartner, P.: The TPTP typed first-order form with arithmetic. In: Bjørner, N., Voronkov, A. (eds.) LPAR-18. LNCS, vol. 7180, pp. 406–419. Springer (2012)

[133] Terese: Term rewriting systems, Cambridge tracts in theoretical computer science, vol. 55. Cambridge University Press (2003)

[134] Urban, J., Rudnicki, P., Sutcliffe, G.: ATP and presentation service for Mizar formalizations. J. Autom. Reason. 50(2), 229–241 (2013)

[135] Väänänen, J.: Second-order and higher-order logic. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy. Metaphysics Research Lab, Stanford University, fall 2019 edn. (2019)

[136] Vukmirović, P., Bentkamp, A., Nummelin, V.: Efficient full higher-order unification. In: Ariola, Z.M. (ed.) FSCD 2020. LIPIcs, vol. 167, pp. 5:1–5:17. Schloss Dagstuhl—Leibniz-Zentrum für Informatik (2020)

[137] Vukmirović, P., Blanchette, J.C., Cruanes, S., Schulz, S.: Extending a brainiac prover to lambda-free higher-order logic. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. LNCS, vol. 11427, pp. 192–210. Springer (2019)

[138] Vukmirović, P., Nummelin, V.: Boolean reasoning in a higher-order superposition prover. In: Fontaine, P., Korovin, K., Kotsireas, I.S., Rümmer, P., Tourret, S. (eds.) PAAR-2020. CEUR Workshop Proceedings, vol. 2752, pp. 148–166. CEUR-WS.org (2020)

[139] Vukmirović, P., Blanchette, J., Cruanes, S., Schulz, S.: Extending a brainiac prover to lambda-free higher-order logic. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. LNCS, vol. 11427, pp. 192–210. Springer (2019)

[140] Waldmann, U., Tourret, S., Robillard, S., Blanchette, J.: A comprehensive framework for saturation theorem proving. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) IJCAR 2020, Part I. LNCS, vol. 12166, pp. 316–334. Springer (2020)

[141] Wand, D.: Superposition: Types and Polymorphism. Ph.D. thesis, Universität des Saarlandes (2017)

[142] Weidenbach, C., Dimova, D., Fietzke, A., Kumar, R., Suda, M., Wischnewski, P.: SPASS version 3.5. In: Schmidt, R.A. (ed.) CADE-22. LNCS, vol. 5663, pp. 140–145. Springer (2009)

[143] Zantema, H.: Termination. In: Bezem, M., Klop, J.W., de Vrijer, R. (eds.) Term Rewriting Systems, Cambridge Tracts in Theoretical Computer Science, vol. 55, pp. 181–259. Cambridge University Press (2003)

Titles in the IPA Dissertation Series since 2018

A. Amighi. *Specification and Verification of Synchronisation Classes in Java: A Practical Approach*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-01

S. Darabi. *Verification of Program Parallelization*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-02

J.R. Salamanca Tellez. *Coequations and Eilenberg-type Correspondences*. Faculty of Science, Mathematics and Computer Science, RU. 2018-03

P. Fiterău-Broştean. *Active Model Learning for the Analysis of Network Protocols*. Faculty of Science, Mathematics and Computer Science, RU. 2018-04

D. Zhang. *From Concurrent State Machines to Reliable Multi-threaded Java Code*. Faculty of Mathematics and Computer Science, TU/e. 2018-05

H. Basold. *Mixed Inductive-Coinductive Reasoning Types, Programs and Logic*. Faculty of Science, Mathematics and Computer Science, RU. 2018-06

A. Lele. *Response Modeling: Model Refinements for Timing Analysis of Runtime Scheduling in Real-time Streaming Systems*. Faculty of Mathematics and Computer Science, TU/e. 2018-07

N. Bezirgiannis. *Abstract Behavioral Specification: unifying modeling and programming*. Faculty of Mathematics and Natural Sciences, UL. 2018-08

M.P. Konzack. *Trajectory Analysis: Bridging Algorithms and Visualization*. Faculty of Mathematics and Computer Science, TU/e. 2018-09

E.J.J. Ruijters. *Zen and the art of railway maintenance: Analysis and optimization of maintenance via fault trees and statistical model checking*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-10

F. Yang. *A Theory of Executability: with a Focus on the Expressivity of Process Calculi*. Faculty of Mathematics and Computer Science, TU/e. 2018-11

L. Swartjes. *Model-based design of baggage handling systems*. Faculty of Mechanical Engineering, TU/e. 2018-12

T.A.E. Ophelders. *Continuous Similarity Measures for Curves and Surfaces*. Faculty of Mathematics and Computer Science, TU/e. 2018-13

M. Talebi. *Scalable Performance Analysis of Wireless Sensor Network*. Faculty of Mathematics and Computer Science, TU/e. 2018-14

R. Kumar. *Truth or Dare: Quantitative security analysis using attack trees*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-15

M.M. Beller. *An Empirical Evaluation of Feedback-Driven Software Development*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2018-16

M. Mehr. *Faster Algorithms for Geometric Clustering and Competitive Facility-Location Problems*. Faculty of Mathematics and Computer Science, TU/e. 2018-17

M. Alizadeh. *Auditing of User Behavior: Identification, Analysis and Understanding of Deviations*. Faculty of Mathematics and Computer Science, TU/e. 2018-18

P.A. Inostroza Valdera. *Structuring Languages as Object-Oriented Libraries*. Faculty of Science, UvA. 2018-19

M. Gerhold. *Choice and Chance - Model-Based Testing of Stochastic Behaviour*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-20

A. Serrano Mena. *Type Error Customization for Embedded Domain-Specific Languages*. Faculty of Science, UU. 2018-21

S.M.J. de Putter. *Verification of Concurrent Systems in a Model-Driven Engineering Workflow*. Faculty of Mathematics and Computer Science, TU/e. 2019-01

S.M. Thaler. *Automation for Information Security using Machine Learning*. Faculty of Mathematics and Computer Science, TU/e. 2019-02

Ö. Babur. *Model Analytics and Management*. Faculty of Mathematics and Computer Science, TU/e. 2019-03

A. Afroozeh and A. Izmaylova. *Practical General Top-down Parsers*. Faculty of Science, UvA. 2019-04

S. Kisfaludi-Bak. *ETH-Tight Algorithms for Geometric Network Problems*. Faculty of Mathematics and Computer Science, TU/e. 2019-05

J. Moerman. *Nominal Techniques and Black Box Testing for Automata Learning*. Faculty of Science, Mathematics and Computer Science, RU. 2019-06

V. Bloemen. *Strong Connectivity and Shortest Paths for Checking Models*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2019-07

T.H.A. Castermans. *Algorithms for Visualization in Digital Humanities*. Faculty of Mathematics and Computer Science, TU/e. 2019-08

W.M. Sonke. *Algorithms for River Network Analysis*. Faculty of Mathematics and Computer Science, TU/e. 2019-09

J.J.G. Meijer. *Efficient Learning and Analysis of System Behavior*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2019-10

P.R. Griffioen. *A Unit-Aware Matrix Language and its Application in Control and Auditing*. Faculty of Science, UvA. 2019-11

A.A. Sawant. *The impact of API evolution on API consumers and how this can be affected by API producers and language designers*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2019-12

W.H.M. Oortwijn. *Deductive Techniques for Model-Based Concurrency Verification*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2019-13

M.A. Cano Grijalba. *Session-Based Concurrency: Between Operational and Declarative Views*. Faculty of Science and Engineering, RUG. 2020-01

T.C. Nägele. *CoHLA: Rapid Cosimulation Construction*. Faculty of Science, Mathematics and Computer Science, RU. 2020-02

R.A. van Rozen. *Languages of Games and Play: Automating Game Design & Enabling Live Programming*. Faculty of Science, UvA. 2020-03

B. Changizi. *Constraint-Based Analysis of Business Process Models*. Faculty of Mathematics and Natural Sciences, UL. 2020-04

N. Naus. *Assisting End Users in Workflow Systems*. Faculty of Science, UU. 2020-05

J.J.H.M. Wulms. *Stability of Geometric Algorithms*. Faculty of Mathematics and Computer Science, TU/e. 2020-06

T.S. Neele. *Reductions for Parity Games and Model Checking*. Faculty of Mathematics and Computer Science, TU/e. 2020-07

P. van den Bos. *Coverage and Games in Model-Based Testing*. Faculty of Science, RU. 2020-08

M.F.M. Sondag. *Algorithms for Coherent Rectangular Visualizations*. Faculty of Mathematics and Computer Science, TU/e. 2020-09

D.Frumin. *Concurrent Separation Logics for Safety, Refinement, and Security*. Faculty of Science, Mathematics and Computer Science, RU. 2021-01

A. Bentkamp. *Superposition for Higher-Order Logic*. Faculty of Sciences, Department of Computer Science, VUA. 2021-02