

Formally Correct Deduction Methods for Computational Logic

Asta Halkjær From

DTU



Kongens Lyngby 2023

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary (English)

Some logics make it easier to say what we want than others and some methods of deduction are simpler to work with than others. One thing typically remains important: that we can deduce all valid things (completeness). In this thesis I tackle completeness of various deduction methods for various logics. The work is formalized in the proof assistant Isabelle/HOL which offers a common language for mathematics and computer science where proofs can be checked mechanically. This thesis discusses the following contributions of my PhD project in particular:

- A historical overview of formalized completeness proofs and a formalization that explains the essence of synthetic completeness proofs.
- A formalization of a concise completeness proof for first-order logic in Isabelle/HOL, including solutions to the issues of formalizing quantifiers and proving completeness for formulas with free variables.
- A verified prover for first-order logic with functions, with a formalized completeness proof that takes the search strategy of the prover into account.
- A synthetic completeness proof for a tableau system for basic hybrid logic which is both terminating and in the Seligman style where the proof rules reflect the local perspective that modal logic is based on.
- Formalized soundness and completeness results for epistemic and public announcement logic, instantiated to a range of concrete axiom systems.
- A best-first proof search tactic for the proof assistant Lean 4.
- An abstract framework for synthetic completeness proofs.

Summary (Danish)

Nogle logikker gør det nemmere at udtrykke hvad vi gerne vil end andre og nogle deduktionsmetoder er nemmere at arbejde med end andre. En ting er som regel vigtig: at vi kan deducere alt hvad der er gyldigt (komplethed). I denne afhandling beskæftiger jeg mig med komplethed af diverse deduktionsmetoder for diverse logikker. Arbejdet er formaliseret i bevisassistenten Isabelle/HOL, der tilbyder et fællessprog for matematik og datalogi, hvor beviser kan verificeres automatisk. Afhandlingen diskuterer især følgende af mit PhD-projekts bidrag:

- Et historisk overblik over formaliserede komplethedsbeviser og en formalisering, der forklarer essensen af syntetiske komplethedsbeviser.
- En formalisering af et koncist komplethedsbevis for førsteordenslogik i Isabelle/HOL, inklusiv løsninger til udfordringerne ved at formalisere kvantorer og at bevise komplethed for formler med frie variable.
- En verificeret bevisfører for førsteordenslogik med funktioner, med et formaliseret komplethedsbevis, der tager højde for bevisførerens søgestrategi.
- Et syntetisk komplethedsbevis for et tableauxystem til grundlæggende hybridlogik, som er både terminerende og i Seligmans stil hvor bevisreglerne afspejler det lokale perspektiv som modallogik er baseret på.
- Formaliserede sundheds- og komplethedsresultater for epistemisk og *public announcement* logik, instantieret med en række konkrete aksiomsystemer.
- En bedst-først bevissøgningstaktik for bevisassistenten Lean 4.
- Et abstrakt framework for syntetiske komplethedsbeviser.

Preface

The 3 years of PhD studies started 2020-02-01 and ended 2023-01-31. My PhD studies took place at the Department of Applied Mathematics and Computer Science (DTU Compute) of the Technical University of Denmark (DTU) under DTU Compute's PhD school, and were funded by DTU Compute. My main supervisor was Jørgen Villadsen (DTU Compute), and my co-supervisor was Nina Gierasimczuk (DTU Compute).

Asta Halkjær From
Copenhagen, 31-January-2023

Acknowledgements

I am grateful to my supervisors Jørgen Villadsen and Nina Gierasimczuk for supporting me through my PhD studies. They have been excellent guides through the academic landscape and frequent sources of inspiration. I admire Patrick Blackburn for reigniting my interest in logic again and again. A great thanks to my PhD student colleagues for putting up with my shenanigans and to Frederik Krogsdal Jacobsen especially, for our collaborations, for proofreading a lot of my work and for traveling to FLoC 2022 in Israel with me. Thanks to Otto Mønsted Fonden for financially supporting that trip. I was fortunate to be employed in the AlgoLoG section at DTU with all the amazing people here. Thanks to Anders Schlichtkrull for being a great coauthor and for providing helpful feedback on this thesis. Thank you to Jasmin Blanchette for hosting my virtual research stay at VU Amsterdam and for introducing me to my cherished collaborator Jannis Limperg. Thanks to Jannis for teaching me about Lean internals during our Zoom sessions and for showing me around Amsterdam on my two trips there. Thanks to DTU's Executive Board for awarding me a travel grant to encourage such trips. I also want to thank Mikko Berggren Ettienne and his family for renting me their house during the first year of my studies. I want to thank my friends and family for supporting me through my PhD, be it by listening to me rave about some proof that finally went through, complain about some difficult deadline or just by taking my mind off the whole thing. Finally, I want to thank my wonderful partner Johanna for all that she has done for me.

Contents

Summary (English)	i
Summary (Danish)	ii
Preface	iii
Acknowledgements	iv
1 Introduction	1
1.1 Synopsis of the Following Chapters	2
1.1.1 Formalizing Henkin-Style Completeness of an Axiomatic System for Propositional Logic	2
1.1.2 A Succinct Formalization of the Completeness of First-Order Logic	2
1.1.3 Verifying a Sequent Calculus Prover for First-Order Logic with Functions in Isabelle/HOL	3
1.1.4 Synthetic Completeness for a Terminating Seligman-Style Tableau System	3
1.1.5 Formalized Soundness and Completeness of Epistemic and Public Announcement Logic	4
1.1.6 Aesop: White-Box Best-First Proof Search for Lean	5
1.1.7 An Abstract Framework for Synthetic Completeness	5
1.2 Other Developments	5
2 Formalizing Henkin-Style Completeness of an Axiomatic System for Propositional Logic	7
3 A Succinct Formalization of the Completeness of First-Order Logic	20
4 Verifying a Sequent Calculus Prover for First-Order Logic with Functions in Isabelle/HOL	45
5 Synthetic Completeness for a Terminating Seligman-Style Tableau System	68
6 Formalized Soundness and Completeness of Epistemic and Public Announcement Logic	86
7 Aesop: White-Box Best-First Proof Search for Lean	114

8 An Abstract Framework for Synthetic Completeness	129
8.1 Introduction	130
8.1.1 Related work	131
8.2 Maximal Consistent Sets	131
8.2.1 Ordinals and Cardinals in Isabelle/HOL	132
8.2.2 Enumeration	133
8.2.3 Lindenbaum’s Lemma	134
8.2.4 Lindenbaum’s Extension	135
8.2.5 Consistency	136
8.2.6 Relative Maximality	137
8.2.7 Relative Saturation	137
8.2.8 Concrete Limit Ordinals	137
8.3 Refutations, Derivations and MCSs	139
8.3.1 Refutations	139
8.3.2 Derivations	140
8.4 Truth Lemma	141
8.4.1 Introduction	141
8.4.2 Formalization in Isabelle/HOL	144
8.5 Examples	144
8.5.1 Languages and Proof Systems	145
8.5.2 Propositional Tableau	146
8.5.3 Propositional Sequent Calculus	149
8.5.4 First-Order Logic	151
8.5.5 Modal Logic	155
8.5.6 Hybrid Logic	157
8.6 Conclusion	162
References	162
Included Publications	164
Other Publications	165
Archive of Formal Proofs	167

CHAPTER 1

Introduction

My PhD work has taken place under the title “Formally Correct Deduction Methods for Computational Logic”. Here, *computational logic* is understood as logic in computer science. The following chapters will cover propositional, first-order, hybrid, modal/epistemic and public announcement logic. Each of these plays a role in the field by expressing different ways to reason about the world. This reasoning is enabled by different *deduction methods*, e.g. axiomatic systems, sequent calculus and natural deduction. I work in the language of higher-order logic, enabled by the proof assistant Isabelle/HOL, to prove that the deduction methods are *formally correct*. In particular, I focus on the *completeness* of each deduction method with respect to the logic: whether it can prove every validity. I have also put the theory into practice by co-developing proof automation for the Lean theorem prover.

The rigidity of working in a proof assistant, rather than formal English, places stricter requirements on our definitions and proofs. The induced uniformity enables collaborative efforts like Isabelle’s *Archive of Formal Proofs* or Lean’s *mathlib* where beginners and experts can contribute on equal footing because everyone “speaks the same language” and everyone’s work is automatically verified. The proof assistant can also generate counterexamples to flawed propositions, find the lemmas necessary to prove a theorem or pinpoint what breaks when a definition changes. I argue that working in a proof assistant encourages us to find better, more reusable abstractions, as I have tried to do in this thesis.

1.1 Synopsis of the Following Chapters

I give a brief overview of the main points of each of the following chapters. All but chapter 8 are peer-reviewed articles. Page 164 contains an overview of the included publications and their publication status. Page 165 lists other papers that I have contributed to during my PhD studies, but which are not included in this thesis. Many of these chapters and papers have accompanying Isabelle/HOL formalizations in the Archive of Formal Proofs (AFP). Submissions to the AFP are refereed and accepted entries are kept up to date with the latest Isabelle version. Page 167 contains a bibliography of entries in the AFP that I have contributed to during my PhD studies.

1.1.1 Formalizing Henkin-Style Completeness of an Axiomatic System for Propositional Logic

Chapter 2 serves as an introductory chapter and includes a historical account of formalized soundness and completeness proofs, including the distinction between synthetic and analytic completeness proofs. I formalize an axiomatic system for a minimal fragment of propositional logic in Isabelle/HOL and use it to walk through the core elements of the synthetic completeness method: maximal consistent sets, Lindenbaum’s lemma, Hintikka sets, etc. The chapter includes a lot of Isabelle syntax, and both the presentation and the proofs are improved in the subsequent chapter, but the minimal setting of propositional logic allows a focus on the techniques rather than the logic. The chapter also introduces a technique of using chains of implications to model natural deduction, which is used again in later chapters.

1.1.2 A Succinct Formalization of the Completeness of First-Order Logic

Chapter 3 extends the previous chapter to cover first-order logic. I abstract the notion of propositional tautologies using Smullyan’s idea of a “boolean valuation” for first-order logic. This is an idea I later reuse in the context of epistemic logic and which simplifies the formalization of the axiomatic systems, giving a style similar to that used in pen-and-paper presentations. The chapter focuses on succinctness in both the formalization of first-order logic itself, with its syntax and semantics, and in the synthetic completeness proof. Quantifiers are notoriously difficult to formalize conveniently, but I present a simple setup, based on just a few definitions and lemmas, that encapsulates the problem. Moreover,

the proof works effortlessly for open and closed formulas alike. That is, free variables are of no concern as they have been in previous formalizations.

The Hintikka sets in this chapter are formulated differently than those in chapter 2. This has two benefits: the model existence theorem becomes completely mechanical, and we can derive this definition of Hintikka sets directly from the semantics of first-order logic and the *canonical* model. I expand on this in chapter 8.

1.1.3 Verifying a Sequent Calculus Prover for First-Order Logic with Functions in Isabelle/HOL

Chapter 4 was written in collaboration with Frederik Krogsdal Jacobsen. We start from an existing sequent calculus called SeCaV and design a prover based on the calculus. We use an existing framework in Isabelle/HOL to formalize the prover and verify its soundness and completeness. The formalized prover can be exported to Haskell code for execution. The completeness proof in this chapter is analytic: we build a countermodel from a failed proof attempt, considering *saturated escape paths* rather than *maximal consistent sets*. Since we are verifying the completeness of not just a calculus but a prover with a concrete search strategy, we need to prove that this strategy is sufficiently sophisticated. Moreover, we cannot build a countermodel based on all possible first-order terms, since the prover is not guaranteed to use every term in its instantiations. Instead, we need a bounded countermodel over only the terms that actually appear in a run of the prover. This requires us to formalize the semantics differently than in the original SeCaV system and introduce a *bounded* semantics where the domain is represented concretely as a set rather than implicitly as the inhabitants of a type. We prove completeness under this notion of validity and prove that SeCaV is sound with respect to the bounded semantics. In the end we prove that for any formula derivable in SeCaV, the prover eventually derives it.

1.1.4 Synthetic Completeness for a Terminating Seligman-Style Tableau System

Chapter 5 switches focus to hybrid logic: modal logic with an extra sort of propositional symbols used to name the worlds in the Kripke frame. These *nominals* naturally give rise to satisfaction operators $@_i$ that, for each nominal i , cause the operand to be evaluated at the world denoted by i . These nominals can thus serve as witnesses for the possibility operator \Diamond , similarly to how constants can witness the existential quantifier \exists in first-order logic. In the chapter, I

combine existing work on tableau systems for basic hybrid logic and formalize it in Isabelle/HOL. I start from an existing *Seligman-style* tableau system for basic hybrid logic and impose several constraints from a different tableau system on the rules. These constraints are designed to ensure that no proof attempt can go on forever. I show how to lift some of the restraints and adapt the synthetic completeness proof for the original, unconstrained Seligman-style system to my new terminating version, formally verifying its completeness. It remains future work to verify the termination of the system in Isabelle/HOL. In the chapter, I prove it via a pen-and-paper translation to an existing terminating system.

The chapter relies on careful proof engineering to make sure that the necessary hybrid logic proof theory is formalizable in Isabelle/HOL. Informally, we can talk about the *root nominals* of a tableau as those that were present from the beginning rather than introduced by a tableau rule. This concept is crucial for ensuring termination, since we use it to give a direction to the rule that allows copying formulas between equivalent nominals. In the chapter, I instead parameterize the proof system with a set of *allowed nominals*. This set turns out to clarify the completeness proof by specifying the role played by root nominals more concretely as those we are allowed to copy.

In the chapter, I deviate from the existing work by giving a model based on entire sets of equivalent formulas rather than single nominals representing equivalence classes. This choice yields a slightly more complicated model, but frees me from having to prove the admissibility of the so-called *Bridge rule* which is otherwise used to guarantee that the equivalence class representatives are well behaved.

1.1.5 Formalized Soundness and Completeness of Epistemic and Public Announcement Logic

Chapter 6 continues the theme of modal logic, but returns to axiomatic systems. Here, I focus on *generic* soundness and completeness proofs for epistemic and public announcement logic.

To achieve this, I first formalize the family of normal modal logics by parameterizing the minimal proof system with a set of extra axioms A . We can instantiate this parameter with no extra axioms to get the system **K**, the axiom $K_i\phi \rightarrow \phi$ to get system **T** and so on. I then formalize the entire synthetic completeness proof for epistemic logic in this abstract setting to arrive at my generic completeness theorem: if the canonical model induced by axioms A has property P (e.g. reflexivity) and the formula ϕ is valid on all P -models, then we can derive ϕ using axioms A . This theorem is easily instantiated with concrete systems, by simply proving that, for instance, the **T** axiom enforces reflexivity.

I extend this to public announcement logic by formalizing the usual completeness proof based on a reduction to the underlying epistemic logic. This allows me to reuse the generic theorem for epistemic logic to obtain a generic completeness theorem for public announcement logic. In essence, I lift the completeness of the underlying epistemic logic to its version with public announcements.

1.1.6 Aesop: White-Box Best-First Proof Search for Lean

Chapter 7 was written in collaboration with Jannis Limperg. We develop proof automation for the Lean 4 theorem prover. The implemented proof tactic is based on best-first search and can prove a variety of results automatically, making it faster and more pleasant to formalize results in Lean. Users can extend the tactic to take new lemmas into account during its search. Lean is based on dependent type theory and makes heavy use of metavariables to express dependencies between proof goals. We present an elegant method of handling metavariables irrespective of the proof search strategy. Finally, we argue via case studies that Aesop is useful in practice.

1.1.7 An Abstract Framework for Synthetic Completeness

Chapter 8 collects the experience of previous chapters into an abstract framework for developing synthetic completeness proofs. I formalize a transfinite version of Lindenbaum’s lemma that just depends on a reasonable definition of consistency for the logical calculus and can be used to build maximal consistent sets for languages of any cardinality. The constructed maximal consistent sets are saturated, making them useful for logics like first-order logic and hybrid logic. I then show how to derive definitions of Hintikka sets from the logic’s semantics and the models induced by the maximal consistent sets. I instantiate this framework to five different examples: a propositional tableau system, a propositional sequent calculus, a natural deduction system for first-order logic, an axiomatic system for modal logic and a natural deduction system for hybrid logic.

1.2 Other Developments

My PhD studies have resulted in several papers not included in this thesis.

Eschen, Villadsen and I have written about the formalizations of several axiomatic

systems for propositional logic in Isabelle/HOL [6, 11], extending the work in chapter 2. Jacobsen and I develop a prover for the SeCaV system in chapter 4. This system was previously described by Villadsen and me [3] at the Isabelle Workshop 2020 and by Jacobsen, Villadsen and me [7] in more detail and with a teaching focus. We describe the benefits of using a subset of Isabelle/HOL as a proof checker. In doing so, the internal workings are not an opaque black box in some programming language, but Isabelle/HOL definitions that can be inspected by interested students. Schlichtkrull, Villadsen and I [8] have written in detail about the formalization of a completeness proof for first-order logic natural deduction. The formalization uses Fitting’s consistency properties and we reuse them to prove the completeness of a tableau system and sequent calculus. We also extend the original completeness proof to open formulas.

My work on hybrid logic, which originated in my MSc thesis, was first published as a short paper at an automated reasoning conference [2] with my MSc thesis advisors Villadsen and Blackburn. I described further progress at a modal logic venue [1], before chapter 5 was published. I described work in progress on formalizing termination of the tableau system at the Isabelle Workshop 2022 [12]. Chapter 6 on epistemic and public announcement logic originated as a paper about epistemic logic [5] only. Villadsen, Jensen, Schlichtkrull and I briefly described an earlier version of the formalization of public announcement logic in our paper on interactive theorem proving for logic and information [14].

Villadsen, Blackburn and I have written about using Isabelle/HOL as a meta-language for teaching logic [4]. Proof assistants encourage experimentation since feedback is easily available and the solution space is narrowed by the formal syntax. We have also written more specifically about using the generic Isabelle/Pure framework for teaching intuitionistic and classical propositional logic [10]. Villadsen and I have described our experience with teaching automated reasoning and formally verified functional programming in Agda and Isabelle/HOL [9]. Lund, Villadsen and I have written a case study in computer-assisted meta-reasoning based on a small prover verified in Isabelle/HOL [13].

CHAPTER 2

Formalizing Henkin-Style Completeness of an Axiomatic System for Propositional Logic

This chapter contains a lot of syntax. I recommend generous skimming during a first reading and to focus on the natural language explanations.

Preprint

Asta Halkjær From. “Formalizing Henkin-Style Completeness of an Axiomatic System for Propositional Logic”. In: *European Summer School in Logic, Language, and Information - ESSLLI 2019 & 2020 Student Sessions, Selected Papers*. Ed. by Alexandra Pavlova and Mina Young Pedersen. Lecture Notes in Computer Science. Springer, 2023. Forthcoming.

Formalizing Henkin-Style Completeness of an Axiomatic System for Propositional Logic

Asta Halkjær From^[0000–0002–3601–0804]

Technical University of Denmark

Abstract. I formalize a Henkin-style completeness proof for an axiomatic system for propositional logic in the proof assistant Isabelle/HOL. The formalization precisely details the structure of this proof method.

Keywords: Propositional logic · Henkin-style completeness · Isabelle/HOL.

1 Introduction

Hilbert proved the completeness of an axiomatic system for propositional logic in 1917-18 [32], Gödel proved the completeness of first-order logic in 1929 [13] and Henkin [14] simplified this proof in 1947, thus devising what we now know as the Henkin-style method. In this paper I study the structure of a Henkin-style completeness proof for an axiomatic Hilbert system for propositional logic by formalizing it in the proof assistant Isabelle/HOL [21].

Isabelle is a generic proof assistant and Isabelle/HOL is the instance based on higher-order logic. With it, we can state every definition, proposition and proof in the precise language of higher-order logic rather than in natural language. Our proof language is then completely formal, which makes it possible for the machine to assist us in our endeavor. By writing our proofs in the Isar language, an acronym of *intelligible semi-automated reasoning* [31], we can have Isabelle check everything that we type. In particular, Isar contains commands such as **assume** to introduce assumptions, **have** to state a partial result and **moreover** to chain several of these together. After these commands, we typically write so-called «cartouches», delimited by angle brackets, that contain our higher-order logic terms: definitions, statements and so on [21]. Our proofs are checked by the trusted Isabelle/HOL kernel but we do not typically use the kernel's axioms and inference rules directly. Instead we give the name of e.g. a tableaux or resolution prover that will generate the proof for us. By formalizing our proofs like this we know that our conclusions always follow from our assumptions.

Of course, Isabelle cannot verify that our definitions match our intentions, that part is up to us, but formalization still reduces the possibility of mistakes. In particular, it reduces the surface area where mistakes can happen, since the proofs themselves are checked by the machine. Not only does a formalization like this one increase the trust in the result, it can also serve as a reference to understand the proof since every detail is given: no case can be omitted as

“trivial” or left as an “exercise for the reader.” The work can also act as a starting point for formalizing other results based on the same techniques.

The full formalization, just below 400 lines, is available online:

<https://github.com/logic-tools/axiom>

I reproduce the essential pieces of it here and introduce parts of the syntax as needed, but forgo any thorough explanation of the Isabelle commands [21].

1.1 Structure of the paper

After a brief history of formalized completeness proofs, the paper continues with a formalization of the syntax and semantics of propositional logic (§ 2) and a sound proof system (§ 3). The idea of the completeness proof is as follows: given a formula ϕ valid under assumptions ψ_1, \dots, ψ_k , assume for the sake of contradiction that there is no corresponding derivation of ϕ under ψ_1, \dots, ψ_k :

$$\not\vdash \psi_1 \longrightarrow \dots \longrightarrow \psi_k \longrightarrow \phi$$

This means we cannot derive falsity, \perp , when also assuming $\neg\phi$, so:

$$\not\vdash \neg\phi \longrightarrow \psi_1 \longrightarrow \dots \longrightarrow \psi_k \longrightarrow \perp$$

The set $\{\neg\phi, \psi_1, \dots, \psi_k\}$ is therefore *consistent* and can be turned into a *maximal consistent set* (§ 4) through an *extension* (§ 5). Such sets are *Hintikka* sets (§ 6) and their elements have a model. This contradicts the validity assumption, proving that a derivation must exist. The proof system is therefore complete (§ 7) and the paper concludes with possible extensions (§ 8).

1.2 A history of formalized completeness proofs

The formalization is part of a long line of formalized completeness proofs.

Completeness proofs can generally be split into two categories based on their approach: semantic proofs in the style of Gödel [13] and Henkin [14] on the one hand and syntactic proofs in the style of Beth and Hintikka [17] and Gallier [12] on the other. Fitting and Mendelsohn call the semantic proofs “synthetic” because they start from a formula and *synthesize* new ones, building up larger and larger sets of formulas that are consistent with the starting point [9]. Formulas in such sets are then shown to have a model and this is the approach presented here. Fitting and Mendelsohn contrast this with the syntactic proofs that they dub “analytic” because they work by *analyzing* the given formula, breaking it into smaller and smaller subformulas and reasoning from those. In these proofs we typically construct a counterexample from the open leaves or an infinite path of a failed derivation attempt. The synthetic approach is remarked to have a *mathematical*, abstract feeling whereas the analytic approach is more *computational* and often resembles an actual prover for the logic [5].

The Henkin-style completeness method has been applied to modal logic from the beginning, notably to system S5 as early as 1959 by Bayart [1] (in French). Bentley [2] recently formalized such a proof in the proof assistant Lean. Jørgensen et al. [16] adapted the synthetic approach to a tableau system for hybrid logic and I formalized this proof [10] in Isabelle/HOL.

In 1985, Shankar [27] formalized Shoenfield’s first-order logic and axiomatic proof system in the Boyer-Moore theorem prover. He showed propositional completeness of the system analytically by defining a tautology checker for a fragment of the syntax based on negation and disjunction.

In 1996, Persson [24] showed completeness for intuitionistic first-order logic in Martin-Löf type theory using the proof assistant ALF. His proof had a synthetic flavor and the result was constructive: he obtained a program that transforms a proof of validity into a natural deduction or sequent calculus derivation. Persson also formalized an axiomatic system without proving completeness.

By early 2000, Margetson formalized the completeness of first-order logic and the cut elimination theorem for sequent calculus in Isabelle/HOL and Ridge later updated the formalization to the Isar language [18]. Their completeness proof, in the Beth-Hintikka style, was based on analyzing failing branches in proof trees.

In 2005, Braselmann and Koepke [6] followed in the Mizar system but using a Henkin-style argument for their sequent calculus.

In 2007, Berghofer [3] formalized Fitting’s synthetic work on natural deduction [8] in Isabelle/HOL. The formalized model existence theorem was based on Smullyan’s abstract consistency properties [29] and Berghofer followed Fitting in reusing the result to show the Löwenheim-Skolem theorem. I extended the formalized completeness result to also cover open formulas [3]. Inspired by Berghofer, Schlichtkrull [26] proved the completeness of first-order resolution in 2016.

In 2010, Ilik [15] investigated Henkin-style arguments for both classical and intuitionistic first-order logic in the proof assistant Coq.

In 2017, Michaelis and Nipkow [19,20] formalized a number of proof systems for propositional logic in Isabelle/HOL: natural deduction, sequent calculus, an axiomatic Hilbert system and resolution. They gave a syntactic completeness proof for the sequent calculus and showed that sequent calculus derivations can be translated into natural deduction and further into their Hilbert system, obtaining completeness for the three proof systems. Independently of this approach, they formalized the propositional model existence theorem by Fitting [8] and used this result to reprove completeness of the sequent calculus and Hilbert system, respectively. Their formalization is more ambitious than this one and therefore more involved. I start from a smaller syntax and focus on one proof system and one approach. This leads to a simpler formalization that helps us understand the essential pieces of the method.

Blanchette, Popescu and Traytel [5] recently advanced the state of completeness proofs for sequent calculus and tableau systems in Isabelle/HOL. They deliberately chose the Beth-Hintikka style and used codatatypes to model possibly infinite derivation trees. Their result can be instantiated for different variations of sequent calculus or tableau systems and various flavors of first-order logic.

Blanchette [4] gave an overview of the formalized metatheory of various other logical calculi and automatic provers in Isabelle.

If we move to Gödel’s incompleteness theorems, the first one was formalized in the Boyer-Moore theorem prover by Shankar in 1986 [28] and in Coq by O’Connor in 2003 [22]. Both incompleteness theorems have been formalized in Isabelle/HOL by Paulson in 2013 [23] and by Popescu and Traytel in 2019 [25].

In summary, the Henkin style is ubiquitous and we have seen it applied to examples such as sequent calculus and natural deduction for first-order logic, system S5 for modal logic and a tableau system for hybrid logic. Most work either extends the technique to cover more advanced logics or abstracts it so that it applies to several at once. My contribution is to boil this proof style down to its essence, motivate each step as it is presented and to use a proof assistant to ensure precision, correctness and comprehensiveness. The paper may also serve as a fast-paced introduction to Isabelle/HOL on a concrete example.

2 Syntax and Semantics

The syntax is minimal and consists of a logical constant representing falsity, natural numbers as propositional symbols and implication. The datatype *form* models it in Isabelle/HOL with a constructor for each case (falsity, propositional symbols and implication) separated by “|”:

datatype *form* = *Falsity* (\perp) | *Pro* *nat* | *Imp* *form form* (**infixr** \longrightarrow) 25)

The annotations in parentheses allow us to construct formulas using standard notation (in bold, to avoid conflict with built-in Isabelle syntax). The definition of negation as an abbreviation makes use of this:

abbreviation *Neg* (\neg) [40] 40) **where** $\neg p \equiv p \longrightarrow \perp$

The meaning of formulas is given by their interpretation in the meta-logic, where the higher-order logic type *bool* gives the truth of the formula. The semantics are thus a primitive recursive predicate on formulas given an interpretation of propositional symbols:

primrec *semantics* :: ($nat \Rightarrow bool$) \Rightarrow *form* \Rightarrow *bool* (\vdash [50, 50] 50) **where**
 $\langle (I \models \perp) = False \rangle$
 $| \langle (I \models Pro\ n) = I\ n \rangle$
 $| \langle (I \models (p \longrightarrow q)) = ((I \models p) \longrightarrow (I \models q)) \rangle$

The first line gives the type and infix notation while the remaining lines define the predicate by each case of the syntax. The first case states that no interpretation models \perp , the second case that the semantics of a propositional symbol is given by the interpretation and finally the meta-logical implication \longrightarrow interprets the object logic implication \longrightarrow .

3 Proof System

Church's axiom system P_1 [7] is delightfully simple. It consists of modus ponens and three axiom schemas and can be defined in Isabelle/HOL as follows:

inductive *Axiomatics* :: $\langle \text{form} \Rightarrow \text{bool} \rangle (\vdash \rightarrow [50] 50)$ **where**

MP: $\vdash p \Rightarrow \vdash (p \rightarrow q) \Rightarrow \vdash q$
 $|$ *Imp1*: $\vdash (p \rightarrow q \rightarrow p)$
 $|$ *Imp2*: $\vdash ((p \rightarrow q \rightarrow r) \rightarrow (p \rightarrow q) \rightarrow p \rightarrow r)$
 $|$ *Neg*: $\vdash (((p \rightarrow \perp) \rightarrow \perp) \rightarrow p)$

The specification defines the inductive predicate *Axiomatics* with the symbolic name \vdash . The judgment holds for a given formula if the formula can be derived from the specified rule and axioms in a finite number of steps.

This proof system is sound with respect to the semantics, which means that every derivable formula is true under any interpretation:

theorem *soundness*: $\vdash p \Rightarrow I \models p$
by (*induct rule: Axiomatics.induct*) *simp-all*

The **by** command completes the proof in two proof method invocations. The *induct* part states that the proof should be performed by induction over the rules of the proof system, transforming the one goal into a subgoal for each case of the predicate. Next, the simplifier, *simp-all*, easily discharges these subgoals.

4 Consistency and Maximality

A list of formulas is *consistent* when we cannot derive \perp from it. The judgment \vdash has no notion of such entailment but implication, \rightarrow , can serve the same purpose. The below technique is widely applicable, whereas extending the judgment \vdash would make the result harder to translate to e.g. modal logic. The following function, *imply*, builds a chain of implications from a list of assumptions to a conclusion. Here, a list is a finite sequence that is either empty, $[]$, or built from an element, the separator $\#$, and a smaller list. We then say that q can be *derived from* ps when we can derive $\vdash \text{imply } ps \ q$.

primrec *imply* :: $\langle \text{form list} \Rightarrow \text{form} \Rightarrow \text{form} \rangle$ **where**
 $\langle \text{imply } [] \ q = q \rangle$
 $| \langle \text{imply } (p \# ps) \ q = (p \rightarrow \text{imply } ps \ q) \rangle$

A potentially infinite set S is consistent exactly when all its finite subsets are consistent. That is, when there is no list S' that, when treated as a *set*, is a subset of S and that entails \perp in the sense of *imply*:

definition *consistent* :: $\langle \text{form set} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{consistent } S \equiv \nexists S'. \text{ set } S' \subseteq S \wedge \vdash \text{imply } S' \ \perp \rangle$

A set is *maximal* when any proper extension makes it inconsistent:

definition *maximal* :: $\langle \text{form set} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{maximal } S \equiv \forall p. p \notin S \rightarrow \neg \text{consistent } (\{p\} \cup S) \rangle$

Note that, to separate concerns, this allows for inconsistent maximal sets.

5 Extension

We want to grow a consistent set into a *maximal* one while preserving consistency. According to Lindenbaum's lemma, attributed to him by Tarski [30], we can always do this. First, given an enumeration of formulas, (ϕ_n) , construct a corresponding sequence of consistent sets (S_n) in the following way.

Assuming S_n has been constructed, its immediate extension is given by

$$S_{n+1} = \begin{cases} \{\phi_n\} \cup S_n & \text{if } \{\phi_n\} \cup S_n \text{ is consistent,} \\ S_n & \text{otherwise.} \end{cases}$$

That is, we only add the corresponding formula to the previous set if consistency is preserved. In the Isabelle code, the function *extend S f n* constructs S_n from $S = S_0$ given an enumeration of formulas represented by f :

```
primrec extend :: (form set  $\Rightarrow$  (nat  $\Rightarrow$  form)  $\Rightarrow$  nat  $\Rightarrow$  form set) where
  (extend S f 0 = S)
| (extend S f (Suc n) =
  (if consistent ({f n}  $\cup$  extend S f n)
   then {f n}  $\cup$  extend S f n
   else extend S f n))
```

To construct the *maximal consistent set*, take the infinite union $\bigcup S_n$:

```
definition Extend :: (form set  $\Rightarrow$  (nat  $\Rightarrow$  form)  $\Rightarrow$  form set) where
  (Extend S f  $\equiv \bigcup n.$  extend S f n)
```

It is easy to see that the starting set is a subset of the union:

```
lemma Extend-subset: (S  $\subseteq$  Extend S f)
unfolding Extend-def by (metis Union-upper extend.simps(1) range-eqI)
```

And, by induction, that any element S_m is a superset of previous elements:

```
lemma extend-bound: (( $\bigcup n \leq m.$  extend S f n) = extend S f m)
by (induct m) (simp-all add: atMost-Suc)
```

5.1 Consistency

When the initial set S is consistent, so is any S_n by construction:

```
lemma consistent-extend: (consistent S  $\implies$  consistent (extend S f n))
by (induct n) simp-all
```

Finally, the limit, $\bigcup S_n$, is also consistent:

```
lemma consistent-Extend:
assumes (consistent S)
shows (consistent (Extend S f))
```

The proof starts by classical contradiction using the *ccontr* rule:

unfolding *Extend-def*
proof (rule *ccontr*)

Assume, towards a contradiction, that the union is inconsistent. Then we can derive \perp from some subset S' :

assume $\langle \neg \text{consistent } (\bigcup n. \text{extend } S f n) \rangle$
then obtain S' **where** $\langle \vdash \text{imply } S' \perp \rangle$ $\langle \text{set } S' \subseteq (\bigcup n. \text{extend } S f n) \rangle$
unfolding *consistent-def* **by** *blast*

This subset is finite so it must be a subset of a finite segment of the union, say, $S_0 \cup \dots \cup S_m$ for some m :

then obtain m **where** $\langle \text{set } S' \subseteq (\bigcup n \leq m. \text{extend } S f n) \rangle$
using *UN-finite-bound* **by** (metis *List.finite-set*)

But every element in (S_n) is a subset of the next, so S' is a subset of S_m :

then have $\langle \text{set } S' \subseteq \text{extend } S f m \rangle$
using *extend-bound* **by** *blast*

And we already established that any such element is consistent:

moreover have $\langle \text{consistent } (\text{extend } S f m) \rangle$
using *assms consistent-extend* **by** *blast*

So there cannot be an inconsistent subset S' and we have our contradiction:

ultimately show *False*
unfolding *consistent-def* **using** $\langle \vdash \text{imply } S' \perp \rangle$ **by** *blast*
qed

In conclusion, $\bigcup S_n$ is consistent when S_0 is.

5.2 Maximality

Importantly, the union $\bigcup S_n$ is also maximal (regardless of the choice of S_0) when the enumeration f is surjective. That is, when it enumerates every formula:

lemma *maximal-Extend*:
assumes $\langle \text{surj } f \rangle$
shows $\langle \text{maximal } (\text{Extend } S f) \rangle$
(proof omitted)

The proof is similar to the one for consistency. If the union is not maximal then there is some $\phi_k \notin \bigcup S_n$ such that $\{\phi_k\} \cup \bigcup S_n$ is consistent. Since $\phi_k \notin \bigcup S_n$, it was not added to the sequence, i.e. $\phi_k \notin S_{k+1}$, and by construction this must be because $\{\phi_k\} \cup S_k$ is inconsistent. But $\{\phi_k\} \cup \bigcup S_n$ is a superset of $\{\phi_k\} \cup S_k$, so $\{\phi_k\} \cup \bigcup S_n$ must be inconsistent too, contradicting the assumption.

6 Hintikka Sets

The completeness proof works by showing that every maximal consistent set is a Hintikka set, where Hintikka sets are defined by the following four conditions:

```

locale Hintikka =
  fixes  $H :: \langle \text{form set} \rangle$ 
  assumes
    NoFalsity:  $\langle \perp \notin H \rangle$  and
    Pro:  $\langle \text{Pro } n \in H \implies (\neg \text{Pro } n) \notin H \rangle$  and
    ImpP:  $\langle (p \longrightarrow q) \in H \implies (\neg p) \in H \vee q \in H \rangle$  and
    ImpN:  $\langle (\neg (p \longrightarrow q)) \in H \implies p \in H \wedge (\neg q) \in H \rangle$ 

```

The idea is to ensure that every formula in a set is satisfiable through syntactic criteria. This is done by making the set *downwards saturated* [29] such that the satisfiability of any complex formula is guaranteed by conditions on its subformulas. Since \perp is unsatisfiable it should never occur (*NoFalsity*), and if a propositional symbol occurs then its negation should not (*Pro*). An implication is satisfied if either the antecedent is false or the consequent is true, so if an implication occurs in a Hintikka set, then either the negated antecedent or the consequent should too (*ImpP*). If a negated implication occurs in a Hintikka set then so should both the antecedent and negated consequent (*ImpN*).

6.1 Model existence

The downwards saturation ensures that if we interpret every proposition in a Hintikka set as true, then every larger formula in the set will be modelled inductively by this interpretation. Therefore, the model is based on set membership:

abbreviation (*input*) $\langle \text{model } H \ n \equiv \text{Pro } n \in H \rangle$

This satisfies any formula in a Hintikka set (and falsifies its negation):

```

lemma Hintikka-model:
   $\langle \text{Hintikka } H \implies (p \in H \longrightarrow \text{model } H \models p) \wedge ((\neg p) \in H \longrightarrow \neg \text{model } H \models p) \rangle$ 
  by (induct p) (simp; unfold Hintikka-def, blast)+

```

The proof goes by induction on the structure of the formula and standard proof methods to handle each resulting case. We need to prove both satisfaction and falsification at the same time to obtain a strong enough induction hypothesis.

6.2 Maximal consistency

The penultimate task is to show that a maximal consistent set is a Hintikka set:

```

lemma Hintikka-Extend:
  assumes  $\langle \text{maximal } S \rangle \langle \text{consistent } S \rangle$ 
  shows  $\langle \text{Hintikka } S \rangle$ 

```

The proof has four similar cases based on the cases of the Hintikka definition. The following gives the essence. Consider first propositional symbols:

```

fix  $n$ 
assume  $\langle \text{Pro } n \in S \rangle$ 
moreover have  $\langle \vdash \text{ imply } [\text{Pro } n, \neg \text{Pro } n] \perp \rangle$ 
  by  $(\text{simp add: FalsityE})$ 
ultimately show  $\langle (\neg \text{Pro } n) \notin S \rangle$ 
  using  $\text{assms}(2)$  unfolding  $\text{consistent-def}$ 
  by  $(\text{metis bot.extremum empty-set insert-subset list.set}(2))$ 

```

Assume a fixed but arbitrary propositional symbol n that occurs positively in S . We can derive \perp from this in combination with a negative occurrence. Thus, the latter cannot appear in the consistent S and this case of the Hintikka definition is proved.

Next, assume that a negated implication occurs in S . By contradiction, so does the antecedent:

```

assume *:  $\langle (\neg (p \longrightarrow q)) \in S \rangle$ 
show  $\langle p \in S \wedge (\neg q) \in S \rangle$ 
proof  $(\text{rule conjI}; \text{rule ccontr})$ 

```

The set S is maximal, so if it does not contain p there must be some finite subset S' of S that we can derive falsity from when adding p :

```

assume  $\langle p \notin S \rangle$ 
then obtain  $S'$  where  $S': \langle \vdash \text{ imply } (p \# S') \perp \rangle \langle \text{set } S' \subseteq S \rangle$ 
  using  $\text{assms inconsistent-head}$  by  $\text{blast}$ 

```

But p follows from the negated implication so we can *cut* out p in favor of it:

```

moreover have  $\langle \vdash \text{ imply } ((\neg (p \longrightarrow q)) \# S') p \rangle$ 
  using  $\text{add-imply ImpE1 deduct}$  by  $\text{blast}$ 
ultimately have  $\langle \vdash \text{ imply } ((\neg (p \longrightarrow q)) \# S') \perp \rangle$ 
  using  $\text{cut'}$  by  $\text{blast}$ 

```

These assumptions, however, are a subset of S , contradicting its consistency:

```

moreover have  $\langle \text{set } ((\neg (p \longrightarrow q)) \# S') \subseteq S \rangle$ 
  using  $*(1)$   $S'(2)$  by  $\text{fastforce}$ 
ultimately show  $\text{False}$ 
  using  $\text{assms unfolding consistent-def}$  by  $\text{blast}$ 

```

7 Completeness

Isabelle can automatically prove the countability of formulas, providing a surjective function *from-nat* for obtaining specific elements of the enumeration (ϕ_n) :

```

instance  $\text{form} :: \text{countable}$  by  $\text{countable-datatype}$ 

```


With this we finally reach the completeness lemma itself. If we assume that p is valid under the assumptions ps , then p can be derived from ps :

lemma *imply-completeness*:
assumes *valid*: $\langle \forall I \text{ s. list-all } (\lambda q. I \models q) \text{ ps} \longrightarrow I \models p \rangle$
shows $\langle \vdash \text{imply ps } p \rangle$

The proof proceeds by contradiction and a similar derivation rule:

proof (*rule ccontr*)
assume $\langle \neg \vdash \text{imply ps } p \rangle$
then have \ast : $\langle \neg \vdash \text{imply } ((\neg p) \# ps) \perp \rangle$
using *Boole* **by** *blast*

Abbreviate the starting consistent set $?S$ and its maximal extension $?H$:

let $?S = \langle \text{set } ((\neg p) \# ps) \rangle$
let $?H = \langle \text{Extend } ?S \text{ from-nat} \rangle$

Then use the previous results to show that $?H$ is a Hintikka set:

have $\langle \text{consistent } ?S \rangle$
unfolding *consistent-def* **using** \ast *imply-weaken* **by** *blast*
then have $\langle \text{consistent } ?H \rangle$ $\langle \text{maximal } ?H \rangle$
using *consistent-Extend maximal-Extend surj-from-nat* **by** *blast+*
then have $\langle \text{Hintikka } ?H \rangle$
using *Hintikka-Extend* **by** *blast*

We have seen that we have a model for any formula in such an $?H$:

have $\langle \text{model } ?H \models p \rangle$ **if** $\langle p \in ?S \rangle$ **for** p
using *that Extend-subset Hintikka-model* $\langle \text{Hintikka } ?H \rangle$ **by** *blast*

So in particular for $\neg p$ and all of ps :

then have $\langle \text{model } ?H \models (\neg p) \rangle$ $\langle \text{list-all } (\lambda p. \text{model } ?H \models p) \text{ ps} \rangle$
unfolding *list-all-def* **by** *fastforce+*

The validity assumption then gives us that *model* $?H$ also models p :

then have $\langle \text{model } ?H \models p \rangle$
using *valid* **by** *blast*

But this is a contradiction:

then show *False*
using $\langle \text{model } ?H \models (\neg p) \rangle$ **by** *simp*
qed

As such, any valid formula must be derivable:

theorem *completeness*: $\langle \forall I. I \models p \implies \vdash p \rangle$
using *imply-completeness* [**where** $ps = \langle [] \rangle$] **by** *simp*

8 Conclusion

We have seen how to formalize the soundness and completeness of a simple axiomatic proof system for propositional logic in Isabelle/HOL. The proof assistant is sophisticated enough to do the soundness proof almost automatically and enables constructions like infinite sets in the proof of completeness.

The choice of propositional logic means that we missed out on an aspect of Henkin's original proof: the use of special constants to witness existential statements. I have included this in a larger formalization of first-order logic [11].

The formalization is simple to extend. The supplementary material contains a file with binary disjunction and conjunction operators added to the syntax, proof system and completeness proof. The result is around 130 lines longer and only adds new lines. The biggest changes are in the Hintikka definition and maximal consistency lemma while model existence remains completely automatic.

Acknowledgements I thank Jørgen Villadsen, Alexander Birch Jensen, Frederik Krogsdal Jacobsen, Anders Schlichtkrull, Agnes Moesgård Eschen and the anonymous reviewers for valuable comments.

References

1. Bayart, A.: Quasi-adéquation de la logique modale du second ordre S5 et adéquation de la logique modale du premier ordre S5. *Logique et Analyse* **2**(6/7), 99–121 (1959)
2. Bentzen, B.: A henkin-style completeness proof for the modal logic S5. In: Baroni, P., Benzmüller, C., Wang, Y.N. (eds.) *Logic and Argumentation - 4th International Conference, CLAR 2021, Hangzhou, China, October 20-22, 2021, Proceedings*. *Lecture Notes in Computer Science*, vol. 13040, pp. 459–467. Springer (2021). https://doi.org/10.1007/978-3-030-89391-0_25
3. Berghofer, S.: First-Order Logic According to Fitting. *Archive of Formal Proofs* (August 2007), <https://isa-afp.org/entries/FOL-Fitting.html>
4. Blanchette, J.C.: Formalizing the metatheory of logical calculi and automatic provers in Isabelle/HOL (invited talk). In: Mahboubi, A., Myreen, M.O. (eds.) *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019*. pp. 1–13. ACM (2019)
5. Blanchette, J.C., Popescu, A., Traytel, D.: Soundness and completeness proofs by coinductive methods. *Journal of Automated Reasoning* **58**(1), 149–179 (2017)
6. Braselmann, P., Koepke, P.: Gödel's completeness theorem. *Formalized Mathematics* **13**(1), 49–53 (2005)
7. Church, A.: *Introduction to Mathematical Logic*. Princeton Mathematical Series, Princeton University Press (1956)
8. Fitting, M.: *First-Order Logic and Automated Theorem Proving*, Second Edition. Graduate Texts in Computer Science, Springer (1996)
9. Fitting, M., Mendelsohn, R.L.: *First-Order Modal Logic*. Springer (2012)
10. From, A.H.: Formalizing a Seligman-style tableau system for hybrid logic. *Archive of Formal Proofs* (December 2019), <https://isa-afp.org/entries/Hybrid.Logic.html>

11. From, A.H.: Soundness and completeness of an axiomatic system for first-order logic. Archive of Formal Proofs (September 2021), https://isa-afp.org/entries/FOL_Axiomatic.html
12. Gallier, J.H.: Logic for computer science: foundations of automatic theorem proving. Courier Dover Publications (2015)
13. Gödel, K.: Über die Vollständigkeit des Logikkalküls. Ph.D. thesis, University of Vienna (1929)
14. Henkin, L.: The Discovery of My Completeness Proofs. Bulletin of Symbolic Logic **2**(2), 127–158 (1996)
15. Ilik, D.: Constructive completeness proofs and delimited control. Ph.D. thesis, École polytechnique (2010)
16. Jørgensen, K.F., Blackburn, P., Bolander, T., Bräuner, T.: Synthetic completeness proofs for Seligman-style tableau systems. In: Proceedings of the 11th conference on Advances in Modal Logic. pp. 302–321 (2016)
17. Kleene, S.C.: Mathematical Logic. Wiley, London (1967)
18. Margetson, J., Ridge, T.: Completeness theorem. Archive of Formal Proofs (September 2004), <https://isa-afp.org/entries/Completeness.html>
19. Michaelis, J., Nipkow, T.: Propositional proof systems. Archive of Formal Proofs (June 2017), https://isa-afp.org/entries/Propositional_Proof_Systems.html
20. Michaelis, J., Nipkow, T.: Formalized proof systems for propositional logic. In: Abel, A., Forsberg, F.N., Kaposi, A. (eds.) 23rd Int. Conf. Types for Proofs and Programs (TYPES 2017). LIPICs, vol. 104, pp. 6:1–6:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2018)
21. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL - A Proof Assistant for Higher-Order Logic, Lecture Notes in Computer Science, vol. 2283. Springer (2002)
22. O'Connor, R.: Essential incompleteness of arithmetic verified by Coq. In: International Conference on Theorem Proving in Higher Order Logics. pp. 245–260. Springer (2005)
23. Paulson, L.C.: A machine-assisted proof of Gödel's incompleteness theorems for the theory of hereditarily finite sets. The Review of Symbolic Logic **7**(3), 484–498 (2014)
24. Persson, H.: Constructive completeness of intuitionistic predicate logic. Licenciata thesis, Chalmers University of Technology (1996)
25. Popescu, A., Traytel, D.: A formally verified abstract account of Gödel's incompleteness theorems. In: Fontaine, P. (ed.) Automated Deduction – CADE 27. pp. 442–461. Springer International Publishing, Cham (2019)
26. Schlichtkrull, A.: Formalization of the resolution calculus for first-order logic. Journal of Automated Reasoning **61**(1-4), 455–484 (2018)
27. Shankar, N.: Towards mechanical metamathematics. Journal of Automated Reasoning **1**(4), 407–434 (1985)
28. Shankar, N.: Metamathematics, machines, and Gödel's proof, Cambridge tracts in theoretical computer science, vol. 38. Cambridge University Press (1994)
29. Smullyan, R.M.: First-Order Logic. Springer-Verlag (1968)
30. Tarski, A.: Logic, Semantics, Metamathematics: Papers from 1923 to 1938. Hackett Publishing (1983)
31. Wenzel, M.: Isabelle/Isar—a generic framework for human-readable proof documents. From Insight to Proof—Festschrift in Honour of Andrzej Trybulec **10**(23), 277–298 (2007)
32. Zach, R.: Completeness before Post: Bernays, Hilbert, and the development of propositional logic. Bulletin of Symbolic Logic **5**(3), 331–366 (1999)

CHAPTER 3

A Succinct Formalization of the Completeness of First-Order Logic

Origin

Asta Halkjær From. “A Succinct Formalization of the Completeness of First-Order Logic”. In: *27th International Conference on Types for Proofs and Programs, TYPES 2021, June 14-18, 2021, Leiden, The Netherlands (Virtual Conference)*. Ed. by Henning Basold, Jesper Cockx, and Silvia Ghilezan. Vol. 239. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 8:1–8:24. DOI: [10.4230/LIPIcs.TYPES.2021.8](https://doi.org/10.4230/LIPIcs.TYPES.2021.8).

A Succinct Formalization of the Completeness of First-Order Logic

Asta Halkjær From   

Technical University of Denmark, Kongens Lyngby, Denmark

Abstract

I succinctly formalize the soundness and completeness of a small Hilbert system for first-order logic in the proof assistant Isabelle/HOL. The proof combines and details ideas from de Bruijn, Henkin, Herbrand, Hilbert, Hintikka, Lindenbaum, Smullyan and others in a novel way, and I use a declarative style, custom notation and proof automation to obtain a readable formalization. The formalized definitions of Hintikka sets and Herbrand structures allow open and closed formulas to be treated uniformly, making free variables a non-concern. This paper collects important techniques in mathematical logic in a way suited for both study and further work.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases First-Order Logic, Completeness, Isabelle/HOL

Digital Object Identifier 10.4230/LIPIcs.TYPES.2021.8

Supplementary Material

Software (Formalization (stable)): https://isa-afp.org/entries/FOL_Axiomatic.html

Software (Formalization (latest)): https://devel.isa-afp.org/entries/FOL_Axiomatic.html

Acknowledgements Thanks to Frederik Krogsdal Jacobsen, Alexander Birch Jensen and Jørgen Villadsen for their useful comments. I am especially grateful to the anonymous reviewers for their insightful remarks which have improved both the formalization and the paper.

1 Introduction

The completeness of first-order logic has been known since Gödel’s work in 1929 [19]. Proof assistants too have a long history [18], with de Bruijn initiating the Automath project in 1967 and the first system of LCF, an Isabelle/HOL predecessor, being developed in 1972. Despite of this, I am unaware of a formalization of completeness in a proof assistant with a focus on explaining the core techniques. The ideas involved in such a proof deserve to be documented and detailed in a formalization where the proof assistant guarantees precision and correctness. This effort benefits students trying to understand mathematical logic and researchers looking for a base for their own work. I start from a Hilbert system, partly because I am unaware of a formalization which does the same, and partly because its simplicity allows me to focus on the ideas of the completeness proof itself. While other deduction systems may be more popular for first-order logic, Hilbert systems are still prominent in areas like modal logic.

This paper builds especially on work by Smullyan [40] and Henkin [21]. The Hilbert system of choice is Smullyan’s System Q1 [40, p. 81] and the completeness proof resembles the “more direct construction” near the end of his book [40, p. 96] (a construction that was pointed out to him by Henkin himself). This paper formalizes ideas by de Bruijn, Henkin, Herbrand [23], Hilbert, Hintikka, Lindenbaum and Smullyan in an attempt to give a “strikingly direct” [40, p. 96] completeness proof formalized in a modern proof assistant.

Smullyan includes a generalization rule for the universal quantifier that works under an assumption (i.e. to the right of an implication) rather than on a standalone formula. This extra generality makes it very suited for my purposes, where I always work under a number



© Asta Halkjær From;
licensed under Creative Commons License CC-BY 4.0

27th International Conference on Types for Proofs and Programs (TYPES 2021).

Editors: Henning Basold, Jesper Cockx, and Silvia Ghilezan; Article No. 8; pp. 8:1–8:24



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of assumptions. Smullyan does not consider function symbols as part of his syntax, but his System Q1 is easily extended to cover these: simply allow for any term in the instantiation axiom. Barwise [1] makes the same modification.

I use the proof assistant Isabelle/HOL [34]. Isabelle is a generic proof assistant and Isabelle/HOL is the version based on higher-order logic. This paper includes a number of Isabelle listings, all taken from the formalization after it has been exported to L^AT_EX. These listings appear either in bulleted lists or prefixed by an Isabelle command in **bold** and should therefore be clearly distinguishable from the surrounding text. Any such listing has been checked and verified by the proof assistant. I sometimes use these listings to explain proofs. In these cases, I do not include the commands that justify each step of reasoning, but only the intermediate statements themselves. For clarity, I have omitted many types from the main text. Some of these can be found in Table 1 on page 6. The full formalization (under 700 lines) is available in the Archive of Formal Proofs [17], which referees Isabelle formalizations and, when accepted, keeps them up to date with the latest version of the proof assistant.

Contributions

The main contribution of this paper is a succinct formalization of the definitions and proofs that make up the synthetic style, a widely applicable method of proving completeness.

As a smaller contribution, this is, to my knowledge, the first formalization of completeness for classical first-order logic that starts from a Hilbert system. However, several formalizations that start from other proof systems are available (cf. Section 2) and the relations between various proof systems have also been formalized, see for instance recent work by Laurent [27] in Coq on translating between Hilbert systems and natural deduction for first-order logic.

On the more technical side, I formalize a Herbrand universe which, like in Herbelin and Ilik’s [22] unformalized proof, consists of all terms, not just those without variables. Combined with a Hintikka set in the style of Forster et al. [11] in Coq, based on the absence of formulas rather than the presence of their negations, but which, unlike theirs, contains open formulas as well as closed, I naturally formalize completeness for all valid formulas.

Isabelle/HOL Overview

This section seeks to give a quick overview of the Isabelle/HOL features used later. Nipkow and Klein [33, Part 1] give a more complete introduction.

The higher-order logic of Isabelle/HOL can be reasonably understood as functional programming + logic [33]. The **datatype** command defines a new type from a series of constructors, where each can be given custom syntax. The natural numbers are built from the nullary constructor *0* and unary *Suc*. The constructors *True* and *False* belong to the built-in type *bool*. The usual connectives and quantifiers from first-order logic (\rightarrow , \forall , etc.) are available for *bool*, as well as *if-then-else* expressions. We can write total functions over datatypes using **primrec** for primitive recursive functions and **fun** for more advanced definitions. The type constructor $A \Rightarrow B$ denotes a function from *A* to *B*. Instead of concrete types, we can also use type variables *'a*, *'b*, etc. These stand in the place of other types. The term *UNIV* stands for the set of all values of a given type.

Another important built-in type is *'a list*, the type of lists whose elements are of type *'a*. These are built from $[]$, the empty list, and $\#$, an infix constructor that adjoins an element to an existing list. The notation $[a, b, c]$ is shorthand for these primitive operations. The function *set* turns a list into a set of its elements. The higher-order function *map* applies a given function to every element of a list.

Function application resembles functional programming languages in that $f(x, y)$ is written as $f\ x\ y$. The function $f(x := y)$ maps x to y and every other element x' to $f\ x'$. Anonymous functions can be built using λ -expressions, e.g. $\lambda n. n + n$ for $f(n) = n + n$.

In proofs, the meta-logical implication \implies separates assumptions from conclusions. These can also be distinguished using the **assumes** and **shows** keywords, using **and** as a separator when there are multiple assumptions or conclusions. The keyword **have** states an intermediate fact in a proof and the keywords **then**, **moreover** and **ultimately** bind these statements together in different ways. The keyword **let** introduces a local abbreviation and **obtain** eliminates an existential statement; **for** quantifies a statement universally.

The command **definition** introduces a new definition that is hidden behind a name and must be explicitly unfolded, while an **abbreviation** is unfolded by the parser. The **inductive** command also makes use of the meta-logical implication. This command allows us to specify the least predicate closed under some given rules. For instance a predicate that denotes whether a formula can be derived, specified by axioms and inference rules. A **locale** in Isabelle, as used here, names an association between terms and assumptions about them. We could, for instance, specify a group as a set and a binary operation coupled with the group axioms. To instantiate the locale we must give concrete terms and show that they satisfy the assumptions. When assuming a locale, we assume the conditions hold for the terms.

The axiom of choice is available as Hilbert's choice operator: the expression $SOME\ x.\ P\ x$ returns some element x that satisfies the predicate P , when such an element exists.

Overview of Paper

I discuss related work next (Section 2). In Section 3, I formalize the syntax of first-order logic in Isabelle/HOL, including the idea of de Bruijn indices. This idea is developed further in Section 4 on the semantics of terms and formulas. Section 5 formalizes the proof system and its soundness, including all the operations necessary to do so. This includes the instantiation of universal quantifiers, propositional tautologies and a range of lemmas. I prove the completeness in Section 6 where I introduce the notion of a maximal consistent set, the Lindenbaum construction and the model existence theorem for Hintikka sets. I discuss challenges and choices in Section 7 and conclude with thoughts on future work in Section 8.

2 Related Work

The completeness of first-order logic itself has a long history, starting with Gödel's proof [19] and Henkin's later simplification [21]. Smullyan [40], Barwise [1] and Fitting [10] all include completeness proofs in their texts. Smullyan's main completeness proof is an "analytic" proof for a tableau system. He devotes only two pages to the "synthetic" (also called Henkin-style) completeness proof of System Q1 [40, pp. 96–97] that I formalize a version of here. Barwise [1] considers System Q1 extended with axioms for equality and gives a quite different proof that relies on a reduction to propositional logic (and the completeness of propositional logic). Fitting [10] proves completeness for tableaux and resolution similarly to Smullyan and leaves the completeness of his Hilbert system as an exercise for the reader. This paper spells out the synthetic completeness proof for first-order logic, starting from a Hilbert system rather than from tableaux, resolution or similar.

The synthetic style has been successfully applied in several formalizations lately. From [13] used it to formalize the completeness of a Hilbert system for propositional logic in Isabelle/HOL. Berghofer [3] formalized natural deduction for first-order logic in Isabelle/HOL.

following the work by Fitting [10]. My formalization of the syntax and semantics of first-order logic and the Lindenbaum construction is inspired by his work. My formalization of Hintikka sets and proof of the model existence theorem, however, differ from his and is inspired by Herbelin and Ilik [22] and Forster et al. [11]. In particular, I prove completeness for open and closed formulas together, where Berghofer’s completeness proof only covers closed formulas and is extended to cover open formulas afterwards. From, Schlichtkrull and Villadsen [14, 16] built on Berghofer’s work to formalize the completeness of both a sequent calculus and tableau system for first-order logic. They also described Berghofer’s formalization in detail. Bentzen [2] formalized the completeness of a Hilbert system for the modal logic S5 in Lean. Jørgensen et al. [26] gave a synthetic completeness proof for a tableau system for basic hybrid logic, which From [12, 15] formalized in Isabelle/HOL.

I am far from the first to formalize the completeness of first-order logic, but my formalization is the only one that proves completeness for a Hilbert system for classical first-order logic. Shankar [39] formalized a tautology checker for first-order logic in the Boyer-Moore theorem prover, but notably did not formalize first-order completeness. Harrison [20] also formalized first-order logic in higher-order logic (but HOL rather than Isabelle/HOL). He did not formalize a proof system but rather model-theoretic results like compactness and the Löwenheim-Skolem theorem. Margetson and Ridge [29] formalized the completeness of first-order logic without functions in Isabelle/HOL via a sequent calculus. Braselmann and Koepke [7] did the same in their Mizar formalization. Schlichtkrull [37, 38] formalized the completeness of first-order resolution in Isabelle/HOL. Michaelis and Nipkow [30, 31] did not formalize first-order logic, but did formalize a Hilbert system for propositional logic in Isabelle/HOL. They proved completeness via translation from a sequent calculus with an analytic completeness proof. Blanchette, Popescu and Traytel [5, 6] formalized analytic completeness of abstract sequent calculus and tableau systems for first-order logic in Isabelle/HOL. Blanchette [4] outlines formalizations of logical meta-theory in Isabelle/HOL.

The completeness proof presented here relies on Lindenbaum’s lemma: that any consistent set of formulas has a maximal consistent extension. The proof is non-constructive since, for the given semantics, Lindenbaum’s lemma is equivalent to Weak König’s Lemma [22, 24]. There are, however, a number of formalizations of completeness in other meta-theories (and necessarily using other semantics). Veldman [43] gave an intuitionistic completeness theorem for intuitionistic predicate logic in 1976. Persson [35] formalized the completeness of sequent calculus and natural deduction for intuitionistic first-order logic in the ALF proof assistant, but only defined a Hilbert system without further proof. His models are based on formal topology. Constable and Bickford [8] constructively proved completeness for intuitionistic first-order logic in the proof assistant Nuprl. Ilik [25] formalized multiple constructive proofs of first-order completeness in the proof assistant Coq using novel variants of Kripke models for full classical and intuitionistic first-order logic. Forster et al. [11] recently analyzed the computational content of completeness theorems for various semantics and for natural deduction and sequent calculus systems. They mechanized their results in constructive type theory using Coq.

3 Syntax

The following syntax will be our starting point.

A term t is either a variable x or a function symbol f applied to a number of other terms:

$$s, t ::= x \mid f(t_1, \dots, t_n)$$

A function symbol applied to zero terms is called a constant and is denoted by a .

A formula p is either falsity (denoted \perp), a predicate symbol P applied to a list of terms, an implication (\longrightarrow) between two formulas or a universally quantified formula:

$$p, q ::= \perp \mid P(t_1, \dots, t_n) \mid p \longrightarrow q \mid \forall x. p(x)$$

I apply a number of techniques to make this syntax suitable for formalization. First, I represent the variables with de Bruijn indices [9]. Instead of connecting quantifiers and variables by using the same variable symbol x , each variable is a natural number n that is understood to be connected to the n th quantifier, starting from the innermost. For example, the formula $\forall x. \forall y. P(x, y)$ is represented as $\forall \forall P(1, 0)$. This makes it simpler to define capture-avoiding instantiation, which we need for the proof system.

Second, to distinguish variables, function symbols and predicate symbols in the proof assistant, I prefix each kind by a distinct symbol: \dagger for function symbols, \ddagger for predicate symbols and $\#$ for variables. The formula $\forall P(f(0))$ is then written (for now) as $\forall \ddagger P(\dagger f(\#0))$.

Finally, lists of argument terms are represented as proper Isabelle/HOL lists, so the term $f(a)$ becomes $\dagger f [a]$.

The (parameterized) datatype $'f \text{ tm}$ of terms with function symbols of type $'f$ becomes:

```
datatype (params-tm: 'f) tm
  = Var nat (#)
  | Fun 'f ('f tm list) ( $\dagger$ )
```

The annotation *params-tm* generates a function of that name from terms to $'f$ sets: it collects all values of type $'f$ in a given term. I discuss its properties in Section 5.1.

The following abbreviates a constant, as I use these frequently:

```
abbreviation Const ( $\star$ ) where  $\star a \equiv \dagger a []$ 
```

The datatype $('f, 'p) \text{ fm}$ of formulas with functions symbols of type $'f$ and predicate symbols of type $'p$ becomes:

```
datatype (params-fm: 'f, 'p) fm
  = Falsity ( $\perp$ )
  | Pre 'p ('f tm list) ( $\ddagger$ )
  | Imp (('f, 'p) fm) (('f, 'p) fm) (infixr  $\longrightarrow$  55)
  | Uni (('f, 'p) fm) ( $\forall$ )
```

The custom notation for each syntactic case is enclosed in parentheses (**infixr** specifies right associativity and 55 specifies parsing priority). I use bold symbols to avoid conflicts with existing syntax. The notation *params-fm*, similarly to for terms, generates a function which produces a set of all function symbols in a given formula.

The Isabelle command **term** checks the type of an expression. Given the above definitions, we can try our syntax, here with strings for the types of function and predicate symbols:

```
term  $\forall (\perp \longrightarrow \ddagger''P'' [\dagger''f'' [\#0]])$ 
```

In regular notation this would be $\forall x. \perp \longrightarrow P(f(x))$.

The following abbreviation for negation will ease notation: $\neg p \equiv p \longrightarrow \perp$.

Similar notations can easily be introduced for conjunction, disjunction, the existential quantifier etc. since in classical logic, these can be defined using the given syntax.

It should be noted that since arities are implicit in the datatypes above, we unfortunately cannot represent finite signatures. The awarded benefit is that we do not need a predicate to distinguish between correct and incorrect syntax.

■ **Table 1** Type signatures for selected functions.

<i>semantics-tm</i>	$(nat \Rightarrow 'a) \Rightarrow ('f \Rightarrow 'a\ list \Rightarrow 'a) \Rightarrow 'f\ tm \Rightarrow 'a$
<i>semantics-fm</i>	$(nat \Rightarrow 'a) \Rightarrow ('f \Rightarrow 'a\ list \Rightarrow 'a) \Rightarrow ('p \Rightarrow 'a\ list \Rightarrow bool) \Rightarrow ('f, 'p)\ fm \Rightarrow bool$
<i>shift</i>	$(nat \Rightarrow 'a) \Rightarrow nat \Rightarrow 'a \Rightarrow nat \Rightarrow 'a$
<i>boolean</i>	$('p \Rightarrow 'f\ tm\ list \Rightarrow bool) \Rightarrow (('f, 'p)\ fm \Rightarrow bool) \Rightarrow ('f, 'p)\ fm \Rightarrow bool$
<i>Axiomatic</i>	$('f, 'p)\ fm \Rightarrow bool$
<i>imply</i>	$('f, 'p)\ fm\ list \Rightarrow ('f, 'p)\ fm \Rightarrow ('f, 'p)\ fm$
<i>consistent</i>	$('f, 'p)\ fm\ set \Rightarrow bool$
<i>extend</i>	$('f, 'p)\ fm\ set \Rightarrow (nat \Rightarrow ('f, 'p)\ fm) \Rightarrow nat \Rightarrow ('f, 'p)\ fm\ set$
<i>witness</i>	$'f\ set \Rightarrow ('f, 'p)\ fm \Rightarrow ('f, 'p)\ fm\ set$
<i>hmodel</i>	$('f, 'p)\ fm\ set \Rightarrow ('f, 'p)\ fm \Rightarrow bool$

4 Semantics

The semantics of first-order logic has two parts: one for terms and one for formulas. I formalize both as functions.

4.1 Terms

A term evaluates to an element of the *domain*. It does so under an *environment* (a mapping from variables to the domain) and a *function denotation* (a mapping from function symbols to functions on the domain).

In Isabelle, I represent the domain as a type (variable) and the environment as a function E from the natural numbers (the de Bruijn indices) to that type. Similarly, the function denotation becomes the function F from function symbols to functions on the domain. This results in the following definition:

primrec *semantics-tm* ($\llbracket -, - \rrbracket$) **where**
 $\llbracket E, F \rrbracket (\#n) = E\ n$
 $\mid \llbracket E, F \rrbracket (\dagger f\ ts) = F\ f\ (map\ \llbracket E, F \rrbracket\ ts)$

The semantics of a variable is given by the environment and in the case of a function application $\dagger f\ ts$, we must first evaluate all the argument terms ts (using *map*) and then apply the function denoted by f .

Here $\llbracket E, F \rrbracket$ denotes the function from terms to the domain, given by the environment E and function denotation F . As seen in the clause above for functions, this notation lets me conveniently “bundle” a given E and F so they can be applied to any term without the need for anonymous functions. I use a similar notation $\llbracket E, F, G \rrbracket$ for the semantics of formulas.

4.2 Formulas

I use a deep embedding where formulas evaluate to a truth value under an environment E , a function denotation F and a *predicate denotation*, dubbed G , that maps predicate symbols to predicates on the domain. I formalize it as follows:

primrec *semantics-fm* ($\llbracket -, -, - \rrbracket$) **where**
 $\llbracket -, -, - \rrbracket \perp = False$
 $\mid \llbracket E, F, G \rrbracket (\dagger P\ ts) = G\ P\ (map\ \llbracket E, F \rrbracket\ ts)$
 $\mid \llbracket E, F, G \rrbracket (p \longrightarrow q) = (\llbracket E, F, G \rrbracket p \longrightarrow \llbracket E, F, G \rrbracket q)$
 $\mid \llbracket E, F, G \rrbracket (\forall p) = (\forall x. \llbracket E(\theta:x), F, G \rrbracket p)$

The formula \perp is always *False* and the semantics of a predicate is similar to that of a function application. For implication each subformula is evaluated to a truth value and the connective is interpreted using the built-in implication. Similarly, I use the built-in universal quantifier to interpret the object-level quantifier. The notation $E\langle 0:x \rangle$ is explained next.

4.3 Shifting

The expression $E\langle n:x \rangle$ modifies the environment E such that variable n is assigned x , any smaller variable m is assigned $E\ m$ and any larger variable m is assigned $E\ (m-1)$. This *shift* operation has the following definition:

definition *shift* $(-\langle \cdot \rangle)$ **where**
 $E\langle n:x \rangle\ m \equiv \text{if } m < n \text{ then } E\ m \text{ else if } m = n \text{ then } x \text{ else } E\ (m-1)$

To understand the shifting operation on larger variables, consider the following:

$$\llbracket E, F, G \rrbracket (\forall \forall \sharp P [\#0, \#1])$$

By the semantics, this reduces to:

$$\forall x. \llbracket E\langle 0:x \rangle, F, G \rrbracket (\forall \sharp P [\#0, \#1])$$

where the outer quantifier comes from the meta-logic. This again reduces to:

$$\forall x. \forall y. \llbracket E\langle 0:x \rangle\langle 0:y \rangle, F, G \rrbracket (\sharp P [\#0, \#1])$$

Thus, the terms are evaluated by $\llbracket E\langle 0:x \rangle\langle 0:y \rangle, F \rrbracket$. This is clearly correct for variable $\#0$ since $E\langle 0:x \rangle\langle 0:y \rangle\ 0 = y$ as desired. We also want that $\#1$ corresponds to the outer meta-logic quantifier, namely that $E\langle 0:x \rangle\langle 0:y \rangle\ 1 = x$. This is exactly what happens since $E\langle 0:x \rangle\langle 0:y \rangle\ 1 = E\langle 0:x \rangle\ (1-1) = x$. Thus, the semantics reduces to the expected:

$$\forall x. \forall y. G\ P\ [y, x]$$

Notice that any *free* variable in a formula (those with no corresponding quantifier) are not affected by this shifting when it is coupled with the removal of an outer quantifier: they are mapped to whatever E originally assigned them to. In this sense they behave like constants.

The following four lemmas will be used implicitly.

► **Lemma 1 (Shifting).** *The first three results characterize the function and the last one commutes a shift of variable 0 with another shift.*

- $n = m \implies E\langle n:x \rangle\ m = x$
- $m < n \implies E\langle n:x \rangle\ m = E\ m$
- $n < m \implies E\langle n:x \rangle\ m = E\ (m-1)$
- $(E\langle n:y \rangle\langle 0:x \rangle) = (E\langle 0:x \rangle\langle n+1:y \rangle)$

Proof. Immediate from the definition. ◀

5 Proof System

To define the proof system I must first define a number of operations needed to state the side conditions and transformations of formulas.

5.1 Parameters

The proof rule for the universal quantifier will generalize a *fresh* constant to a quantified variable. To perform this freshness check, I use the functions *params-tm* and *params-fm* that Isabelle generates automatically from the datatype declarations above. These collect all function symbols in terms and formulas, respectively, and would also be easy to define recursively. Similarly to Smullyan [40], I abbreviate function symbol to *parameter*.

The following definition generalizes *params-fm* to a set of formulas:

abbreviation $\text{params } S \equiv \bigcup p \in S. \text{params-fm } p$

I need a few lemmas about parameters for later.

► **Lemma 2** (Finite parameters). *Terms and formulas contain only finitely many parameters:*

- *finite* (*params-tm* t)
- *finite* (*params-fm* p)

Proof. By simple inductions. ◀

► **Lemma 3** (Unused parameters). *The denotation of an unused parameter does not affect the semantics of either terms or formulas:*

- $f \notin \text{params-tm } t \implies \llbracket E, F(f := x) \rrbracket t = \llbracket E, F \rrbracket t$
- $f \notin \text{params-fm } p \implies \llbracket E, F(f := x), G \rrbracket p = \llbracket E, F, G \rrbracket p$

Proof. By simple inductions. ◀

5.2 Instantiation

I will need to instantiate a universally quantified formula with a concrete term: to go from $\forall p$ to “ p with t inserted for variable 0 and free variables in p adjusted.” I will denote this formula by $\langle t/0 \rangle p$. Note that when instantiating for n in $\forall p$, we need to then instantiate for $n+1$ in p , since we enter the scope of another quantifier (the formula $\forall x. \forall y. P(x, y)$ becomes $\forall \forall P(1, 0)$ with de Bruijn indices, so to instantiate for x we must actually replace variable 1).

There are two additional considerations. Consider first why we need to adjust the free variables in p . Say that we are instantiating $\forall P [\#0, \#3]$ with the term t . When evaluating $\forall P [\#0, \#3]$ under an environment E , the free variable 3 will be interpreted as $(E(0:x)) \ 3 = E \ 2$. We expect that the interpretation of free variables under the same environment does not change when we instantiate a quantifier. However, when evaluating the naively instantiated formula $P [t, \#3]$, the free variable 3 will be evaluated as $E \ 3$, which might be a different value than $E \ 2$. Therefore, we should decrement any free variables we encounter during the instantiation. The result here should then be $P [t, \#2]$.

Second, it is important that any free variable in t remains free in $\langle t/0 \rangle p$ (i.e. that the instantiation avoids capturing a free variable). With named variables we would ensure this by renaming any bound variables in p that would conflict. By using de Bruijn indices we are free from having to come up with fresh names for such an operation. Instead, we increment every variable in t by one whenever we pass under a quantifier. Thus $\langle \dagger f [\#0]/0 \rangle (\forall (\dagger P)) = \forall (\langle \dagger f [\#1]/1 \rangle (\dagger P))$.

I call this last operation *lifting* the term:

primrec *lift-tm* (\uparrow) **where**
 $\uparrow(\#n) = \#(n+1)$
 $\mid \uparrow(\dagger f \ ts) = \dagger f \ (\text{map } \uparrow \ ts)$

While this terminology is common (cf. Nipkow [32], Berghofer [3]) it unfortunately conflicts with the terminology in explicit substitutions (cf. Lescanne [28]) where *lifting* and *shifting* have roughly opposite meanings compared to this paper.

With the above considerations in mind, we can now define instantiation on terms:

primrec *inst-tm* ($\langle\langle\cdot/\cdot\rangle\rangle$) **where**
 $\langle\langle s/m \rangle\rangle(\#n) = (\text{if } n < m \text{ then } \#n \text{ else if } n = m \text{ then } s \text{ else } \#(n-1))$
 $|\langle\langle s/m \rangle\rangle(\dagger f \text{ ts}) = \dagger f (\text{map } \langle\langle s/m \rangle\rangle \text{ ts})$

The notation $\langle\langle s/m \rangle\rangle$ “bundles” an instantiation of term s for variable m , ready to be applied to a term. For formulas, the only interesting case is for the universal quantifier, where we lift the term we are instantiating with and increment the variable we are instantiating for:

primrec *inst-fm* ($\langle\langle\cdot/\cdot\rangle\rangle$) **where**
 $\langle\langle\cdot/\cdot\rangle\rangle\perp = \perp$
 $|\langle\langle s/m \rangle\rangle(\dagger P \text{ ts}) = \dagger P (\text{map } \langle\langle s/m \rangle\rangle \text{ ts})$
 $|\langle\langle s/m \rangle\rangle(p \longrightarrow q) = \langle\langle s/m \rangle\rangle p \longrightarrow \langle\langle s/m \rangle\rangle q$
 $|\langle\langle s/m \rangle\rangle(\forall p) = \forall (\langle\langle \dagger s/m+1 \rangle\rangle p)$

Despite the complexity of instantiation when using de Bruijn indices, it can be captured in the three simple definitions above that involve little more than simple arithmetic.

A more standard name for $\langle\langle t/n \rangle\rangle p$ is substitution, but I prefer instantiation since it potentially does more than simple syntactic substitution of term t for variable n : namely lifts t and decrements variables in p .

The only results about instantiation that I need for the formalization are the following.

► **Lemma 4** (Lifting and shifting). *Lifting cancels out with shifting the environment at 0.*

■ $\langle\langle E\langle 0:x \rangle, F \rangle\rangle(\dagger t) = \langle\langle E, F \rangle\rangle t$

Proof. By structural induction. ◀

► **Lemma 5** (Instantiation and shifting). *Instantiating with a term at m is the same as shifting the environment at m with the value denoted by that term.*

■ $\langle\langle E, F \rangle\rangle(\langle\langle s/m \rangle\rangle t) = \langle\langle E\langle m:\langle\langle E, F \rangle\rangle s \rangle, F \rangle\rangle t$

■ $\llbracket E, F, G \rrbracket(\langle\langle t/m \rangle\rangle p) = \llbracket E\langle m:\langle\langle E, F \rangle\rangle t \rangle, F, G \rrbracket p$

Proof. By structural induction, using Lemma 4. ◀

5.3 Size

To prove the model existence theorem, I will need to do induction on formulas. However, structural induction does not work, since in the case for $\forall p$, the induction hypothesis must be applied to the instance $\langle\langle t/0 \rangle\rangle p$, for some term t , rather than simply to p . This calls for induction on the size of the formula. Unfortunately, the pre-defined *size* measure for our datatype takes the size of terms into account and is therefore not invariant under instantiation. The following definition suffices:

primrec *size-fm* **where**
 $\text{size-fm } \perp = 1$
 $|\text{size-fm } (\dagger \cdot) = 1$
 $|\text{size-fm } (p \longrightarrow q) = 1 + \text{size-fm } p + \text{size-fm } q$
 $|\text{size-fm } (\forall p) = 1 + \text{size-fm } p$

► **Lemma 6** (Size). *Instantiation preserves size.*

■ $\text{size-fm } (\langle\langle t/m \rangle\rangle p) = \text{size-fm } p$

Proof. By structural induction. ◀

5.4 Propositional Semantics

Instead of picking a suitable set of propositional axioms, Smullyan [40], Barwise [1] and others simply include all tautologies as one of their axioms. I follow their lead and need a suitable way to express which formulas are tautologies. Smullyan [40, p. 51] extends his notion of a *Boolean valuation* from propositional logic to the syntax of first-order logic by treating quantified formulas as another sort of propositional symbols. A *tautology* is then a formula that is true under all Boolean valuations.

The following definition uses the same principle, where G is a predicate denotation as before and A is a special “universally quantified formula denotation.”

primrec *boolean where*
 boolean - - $\perp = \text{False}$
 | *boolean* $G - (\dagger P \text{ } ts) = G \text{ } P \text{ } ts$
 | *boolean* $G \text{ } A (p \longrightarrow q) = (\text{boolean } G \text{ } A \text{ } p \longrightarrow \text{boolean } G \text{ } A \text{ } q)$
 | *boolean* - $A (\forall p) = A (\forall p)$

The hyphens stand for ignored arguments. Compare this semantics to the first-order one: it is indeed a Boolean valuation [40] of first-order logic. We can now take Smullyan’s notion of tautology as definition:

abbreviation *tautology* $p \equiv \forall G \text{ } A. \text{boolean } G \text{ } A \text{ } p$

Smullyan gives no details on his extension of Boolean valuations to first-order logic. The way I set it up, with a separate denotation for the quantified formulas, it can be directly related to the first-order semantics.

► **Lemma 7** (Boolean semantics). *The Boolean and first-order semantics coincide when G matches the first-order predicate semantics and A is the first-order semantics itself.*

■ *boolean* $(\lambda a. G \text{ } a \circ \text{map } (\llbracket E, F \rrbracket)) \llbracket E, F, G \rrbracket = \llbracket E, F, G \rrbracket$

Proof. By structural induction. ◀

► **Lemma 8** (Tautologies). *All tautologies are valid.*

■ *tautology* $p \implies \llbracket E, F, G \rrbracket p$

Proof. Since a tautology holds for any choice of G and A it holds in particular for those that coincide with the first-order semantics (cf. Lemma 7). ◀

For reassurance, Isabelle easily verifies that not all first-order validities are propositional tautologies (e.g. $(\forall x. P(x)) \longrightarrow P(a)$ is only the former):

proposition $\exists p. (\forall E \text{ } F \text{ } G. \llbracket E, F, G \rrbracket p) \wedge \neg \text{tautology } p$

5.5 The Inductively Defined Calculus

Finally, we are ready to define the calculus itself. I define it as an inductive predicate \vdash that holds exactly when a formula can be derived from the given axioms and rules. The previous work has made the definition simple:

inductive *Axiomatic* $(\vdash - [50] \text{ } 50)$ **where**
 TA: *tautology* $p \implies \vdash p$
 | *IA*: $\vdash \forall p \longrightarrow \langle t/0 \rangle p$
 | *MP*: $\vdash p \longrightarrow q \implies \vdash p \implies \vdash q$
 | *GR*: $\vdash q \longrightarrow \langle \star a/0 \rangle p \implies a \notin \text{params } \{p, q\} \implies \vdash q \longrightarrow \forall p$

The Tautology Axiom (*TA*) derives any tautology. The Instantiation Axiom (*IA*) states that a quantified formula implies its instantiation with any term. The Modus Ponens (*MP*) rule is stated as usual and lifts an implication between formulas to an implication between derivations. Finally, the Generalization Rule (*GR*) works under assumptions q and generalizes from an instance to a quantified formula, given that the witness (the constant) is fresh.

► **Theorem 9** (Soundness). *Any derivable formula is valid:*

■ $\vdash p \implies \llbracket E, F, G \rrbracket p$

Proof. By induction over the inductive definition of the axiomatic system for arbitrary function denotation F .

All cases except for *GR* can be proven automatically, with the case for *TA* relying on Lemma 8 about tautologies. In the *GR* case I apply the induction hypothesis not just once at plain F but at $F(a := x)$ for every element x of the domain:

have $\llbracket E, F(a := x), G \rrbracket (q \longrightarrow \langle \star a / 0 \rangle p)$ for x

This is enough help for Isabelle to prove the case. ◀

► **Corollary 10.** *Falsity cannot be derived:*

■ $\neg (\vdash \perp)$

5.5.1 Notation

For the proof of completeness I need to express that a formula can be derived from a set of *assumptions*. Instead of building this notion into the definition of the proof system, I am going to simulate it using chains of implications. The expression $[p_1, p_2, \dots, p_n] \rightsquigarrow q$ expands to $p_1 \longrightarrow p_2 \longrightarrow \dots \longrightarrow p_n \longrightarrow q$. It is defined by recursion on the list of assumptions:

primrec *imply* (infixr \rightsquigarrow 56) where
 $([] \rightsquigarrow q) = q$
 $| (p \# ps \rightsquigarrow q) = (p \longrightarrow ps \rightsquigarrow q)$

I then write $ps \vdash q$ to abbreviate $\vdash ps \rightsquigarrow q$:

When I talk about *assumptions* in a derivation I will always mean a finite list of formulas.

5.6 Derived Formulas

Due to my semantic characterization of the Tautology Axiom, the automation in Isabelle can easily prove that various propositional formulas (schemas) can be derived.

► **Lemma 11** (Derivations). *The S and K combinators, double negation elimination and contraposition in both directions can all be derived:*

■ $\vdash (p \longrightarrow q \longrightarrow r) \longrightarrow (p \longrightarrow q) \longrightarrow p \longrightarrow r$
 ■ $\vdash q \longrightarrow p \longrightarrow q$
 ■ $\vdash \neg \neg p \longrightarrow p$
 ■ $\vdash (p \longrightarrow q) \longrightarrow \neg q \longrightarrow \neg p$
 ■ $\vdash (\neg q \longrightarrow \neg p) \longrightarrow p \longrightarrow q$

Proof. By the Tautology Axiom. ◀

5.6.1 Generalization Rule

My use of chains of implications is a disadvantage to the *GR* rule since it works on the consequent but implication is right associative. Consider the following: we know that $ps \vdash \langle \star a/0 \rangle p$, for fresh a and want to use *GR* to derive $ps \vdash \forall p$. We can only do so if ps consists of exactly one formula q , as $ps \vdash p$ is short for $\vdash ps \rightsquigarrow q$. To circumvent this restriction, I derive the following variant of the rule.

► **Lemma 12** (*GR'* rule). *The following rule is derivable:*

$$GR': \vdash \neg \langle \star a/0 \rangle p \longrightarrow q \implies a \notin \text{params } \{p, q\} \implies \vdash \neg (\forall p) \longrightarrow q$$

Proof. Follows from the *GR* rule, modus ponens and the derivations in Lemma 11. ◀

Since this rule works on the left-hand side of the implication, the right-hand side can, without issues, be an arbitrarily long chain of implications. Smullyan [40, p. 83] himself uses this version of the rule in his System Q1' (but for notational reasons).

An alternative is to start from the existential quantifier, \exists , as primitive, rather than \forall , as the generalization rule for \exists works on the left-hand side of the implication [40]. However, it is less immediately clear why this rule for \exists can be called a generalization rule.

5.6.2 Working with Assumptions

The following is an assortment of useful lemmas for working with assumptions.

► **Lemma 13** (Assumptions). *The following are derivable: modus ponens under assumptions, derivation of any assumption, the deduction theorem in both directions, a cut rule, classical reasoning and finally a structural rule encompassing weakening, contraction and exchange:*

- $ps \vdash p \longrightarrow q \implies ps \vdash p \implies ps \vdash q$
- $p \in \text{set } ps \implies ps \vdash p$
- $ps \vdash p \longrightarrow q \implies p \# ps \vdash q$
- $p \# ps \vdash q \implies ps \vdash p \longrightarrow q$
- $p \# ps \vdash r \implies q \# ps \vdash p \implies q \# ps \vdash r$
- $(\neg p) \# ps \vdash \perp \implies ps \vdash p$
- $ps \vdash q \implies \text{set } ps \subseteq \text{set } ps' \implies ps' \vdash q$

Proof. By a mix of induction over the list of assumptions and propositional reasoning. ◀

6 Completeness

We are now ready to delve into the completeness proof itself. The plan is as follows. If we cannot derive a formula p under any assumptions from X then we cannot derive falsity from $\neg p$ and any assumptions from X either. Sets like $\{\neg p\} \cup X$ are *consistent* with respect to the proof system, as we cannot derive a contradiction from them. I formalize them in Section 6.1. These sets are defined based on the proof system but we will use them to build a model that contradicts the validity of p under X . For this purpose we must prove that two important types of formulas preserve consistency: fresh witnesses of existential formulas (*Henkin witnesses*) and instances of universal formulas.

Lindenbaum (according to Tarski [41]) showed how to extend a consistent set into a *maximal consistent set (MCS)*. Any proper superset of a maximal consistent set is inconsistent. In particular this means that for any formula p , an MCS contains exactly p or $\neg p$. Henkin [21],

showed the utility of adding the Henkin witnesses for existential formulas during Lindenbaum's construction. I formalize the construction and its consistency in Section 6.2 and prove that the result is maximal in Section 6.3.

The addition of Henkin witnesses ensures that our MCSs are *saturated*. Section 6.4 outlines the benefits of ensuring this by construction.

Instead of building a model directly from a maximal consistent saturated set, I introduce a standard layer of abstraction. In Section 6.5, I formalize the notion of a *Hintikka* set [40] using three simple conditions and prove a model existence theorem: given a Hintikka set H , I build a model from a Herbrand structure [10, 22] that satisfies exactly the formulas in H . I then prove that maximal consistent saturated sets are Hintikka sets.

In Section 6.6, I put all the pieces together. The model existence theorem gives us a model for $\neg p$ and all of X . Therefore, if p is in fact valid under assumptions from X , then it must be derivable or we have a contradiction.

6.1 Consistent Sets

The definition of consistency is straightforward. The set of formulas S is consistent when there is no list of assumptions S' , coming from S , that can be used to derive falsity:

definition *consistent* $S \equiv \nexists S'. \text{ set } S' \subseteq S \wedge S' \vdash \perp$

The following lemma will be useful.

► **Lemma 14** (Inconsistent addition). *Assume that S is consistent on its own but becomes inconsistent with the addition of a formula p . Then there exists a concrete list of assumptions S' , coming from S , such that $p \n# S' \vdash \perp$:*

assumes *consistent* S **and** $\neg \text{consistent } (\{p\} \cup S)$
obtains S' **where** *set* $S' \subseteq S$ **and** $p \n# S' \vdash \perp$

Proof. It follows from consistency and the structural lemma for assumptions (Lemma 13). ◀

It is important to prove that two types of formulas preserve consistency. The first type is fresh witnesses for existential formulas.

► **Lemma 15** (Consistency of fresh witnesses). *If a consistent set contains an existential formula $\neg (\forall p)$ then adding a witness $\neg \langle \star a/0 \rangle p$, for a fresh a , preserves consistency:*

assumes *consistent* S **and** $\neg (\forall p) \in S$ **and** $a \notin \text{params } S$
shows *consistent* $(\{\neg \langle \star a/0 \rangle p\} \cup S)$

Proof. We need to show that there is no finite subset from which we can derive falsity, so assume that indeed there is one. From Lemma 14 we can name the problematic assumptions:

then obtain S' **where** *set* $S' \subseteq S$ **and** $\neg \langle \star a/0 \rangle p \n# S' \vdash \perp$

After showing that the side conditions are fulfilled, we can apply the GR' rule:

then have $\neg (\forall p) \n# S' \vdash \perp$

But every assumption is now in S , which we assumed to be consistent, so we have reached the desired contradiction and proved the lemma. ◀

We shall also need that instantiating a universally quantified formula preserves consistency.

► **Lemma 16** (Consistency of instantiation). *If a consistent set contains a universal formula $\forall p$ then adding an instance $\langle t/0 \rangle p$, for any term t , preserves consistency:*

assumes *consistent* S **and** $\forall p \in S$
shows *consistent* $(\{\langle t/0 \rangle p\} \cup S)$

Proof. The proof proceeds as before and we start by naming the problematic assumptions from an assumed inconsistency (Lemma 14):

then obtain S' **where** *set* $S' \subseteq S$ **and** $\langle t/0 \rangle p \# S' \vdash \perp$

This time we make use of the Instantiation Axiom, instantiated to p and t :

moreover have $\vdash \forall p \longrightarrow \langle t/0 \rangle p$

With the deduction theorem, the cut rule and the structural lemma (Lemma 13), we can apply this implication to weaken the derivation of falsity:

ultimately have $\forall p \# S' \vdash \perp$

But again, these assumptions are all in S , which we assumed to be consistent, so this is a contradiction and adding the instance must also be consistent. ◀

6.2 Lindenbaum Extension

We turn now to a central construction. Note first that if the sets of variable, function and predicate symbols are countable, so too are the sets of terms and formulas (formalized in Section 6.6). Thus, we can enumerate the formulas as p_0, p_1, \dots and so on. Starting from a consistent set S_0 , which leaves infinitely many parameters unused, we then build a sequence of consistent sets in the following way. Given S_n , construct S_{n+1} as:

$$S_{n+1} = \begin{cases} w(*, p_n) \cup \{p_n\} \cup S_n & \text{if } \{p_n\} \cup S_n \text{ is consistent} \\ S_n & \text{otherwise} \end{cases}$$

where $*$ is the set of parameters in $\{p_n\} \cup S_n$.

The function w returns a singleton set with a fresh witness when p_n is an existential formula and the empty set otherwise. Usually, the availability of such fresh witnesses is guaranteed by extending the set of function symbols. I assume instead that the set of function symbols is infinite from the start and that S_0 leaves infinitely many parameters unused. I pass the parameters of $\{p_n\} \cup S_n$ to w . It can then pick a parameter that has not been used already. This is simpler than dealing with two sorts of function symbols.

In the Isabelle formalization, the enumeration of formulas is represented by a (surjective) function f from the set of natural numbers to the set of formulas (cf. Section 6.6). The expression $extend\ S\ f\ n$ constructs the set S_n starting from $S_0 = S$:

primrec *extend* **where**
 $extend\ S\ f\ 0 = S$
 $| extend\ S\ f\ (Suc\ n) =$
 $(let\ S_n = extend\ S\ f\ n\ in$
 $\quad if\ consistent\ (\{f\ n\} \cup S_n)$
 $\quad then\ witness\ (params\ (\{f\ n\} \cup S_n))\ (f\ n) \cup \{f\ n\} \cup S_n$
 $\quad else\ S_n)$

The function *witness* is simple:

```

fun witness where
  witness used ( $\neg (\forall p)$ ) =  $\{\neg (\star(\text{SOME } a. a \notin \text{used})/0)p\}$ 
| witness - - = {}

```

Its definition uses Hilbert's choice operator to pick a fresh parameter.

The maximal consistent set is given by taking the union of this sequence of sets: $\bigcup_{n \in \mathbb{N}} S_n$. In Isabelle, it becomes:

definition $\text{Extend } S \ f \equiv \bigcup n. \text{extend } S \ f \ n$

The following lemmas are needed later.

► **Lemma 17** (Lindenbaum bounds). *The starting set is included in its maximal extension and each set in the constructed sequence bounds the previous sets:*

- $S \subseteq \text{Extend } S \ f$
- $(\bigcup n \leq m. \text{extend } S \ f \ n) = \text{extend } S \ f \ m$

Proof. By definition and by induction on m , respectively. ◀

► **Lemma 18** (Lindenbaum parameters). *A witness includes only finitely many parameters and each set S_n contains finitely many more parameters than the starting set S_0 :*

- $\text{finite } (\text{params } (\text{witness used } p))$
- $\text{finite } (\text{params } (\text{extend } S \ f \ n) - \text{params } S)$

Proof. Since p contains finitely many parameters and by induction on n , respectively. ◀

6.2.1 Consistency

The consistency of each constructed set S_n is apparent.

► **Lemma 19** (Consistency of S_n). *When starting from a consistent S_0 with infinitely many unused parameters, any constructed S_n is consistent:*

assumes $\text{consistent } S$ **and** $\text{infinite } (\text{UNIV} - \text{params } S)$
shows $\text{consistent } (\text{extend } S \ f \ n)$

Proof. By induction on n . The consistency of adding the witness follows from Lemma 15. The only complication is to prove that there are indeed always fresh parameters available and therefore that the parameter given by Hilbert's choice operator is usable, but this follows from Lemma 18. ◀

The consistency of the union $\bigcup_n S_n$ is more interesting.

► **Lemma 20** (Consistency of $\bigcup_n S_n$). *The maximal extension of a consistent set S with infinitely many unused parameters is consistent:*

assumes $\text{consistent } S$ **and** $\text{infinite } (\text{UNIV} - \text{params } S)$
shows $\text{consistent } (\text{Extend } S \ f)$

Proof. Assume towards a contradiction that we can derive falsity from some finite subset:

then obtain S' **where** $S' \vdash \perp$ **and** $\text{set } S' \subseteq \text{Extend } S \ f$

Since this subset is finite, it must be a subset of some initial segment of the union:

then obtain m **where** $\text{set } S' \subseteq (\bigcup n \leq m. \text{extend } S \ f \ n)$

But, by Lemma 17, each such segment is bounded by its last element:

then have $\text{set } S' \subseteq \text{extend } S \ f \ m$

And since we have already shown the consistency of each S_n (Lemma 19), we reach our desired contradiction. ◀

6.3 Maximal Sets

A maximal set is inconsistent under any proper extension:

definition $\text{maximal } S \equiv \forall p. p \notin S \longrightarrow \neg \text{consistent } (\{p\} \cup S)$

Maximal consistent sets are truly maximal:

► **Lemma 21** (Maximality of Maximal Consistent Sets). *If S is a maximal consistent set, then for every formula p , $p \in S$ if and only if $\neg p \notin S$.*

assumes $\text{consistent } S$ **and** $\text{maximal } S$
shows $p \in S \longleftrightarrow (\neg p) \notin S$

Proof. The left-to-right direction follows from consistency alone and the right-to-left direction follows from consistency and maximality. ◀

That the Lindenbaum extension results in a maximal set is very easy to see.

► **Lemma 22** (Maximality of $\bigcup_n S_n$). *Given a surjective enumeration f , $\bigcup_n S_n$ is maximal:*

assumes $\text{surj } f$
shows $\text{maximal } (\text{Extend } S f)$

Proof. Assume towards a contradiction that some formula p is not included even though its inclusion preserves consistency:

assume $p \notin \text{Extend } S f$ **and** $\text{consistent } (\{p\} \cup \text{Extend } S f)$

Say that p is formula number k in the enumeration. Since p is not in the result, it must be inconsistent with S_k :

then have $\neg \text{consistent } (\{p\} \cup \text{extend } S f k)$

And this set is a subset of the final result:

moreover have $\{p\} \cup \text{extend } S f k \subseteq \{p\} \cup \text{Extend } S f$

Ultimately, this contradicts the assumption that adding p preserves consistency. ◀

6.4 Saturation

We shall need saturation to show that our constructed sets are Hintikka sets:

definition $\text{saturated } S \equiv \forall p. \neg (\forall p) \in S \longrightarrow (\exists a. (\neg \langle \star a / 0 \rangle p) \in S)$

So, in a saturated set there is a corresponding Henkin witness for each existential formula.

► **Lemma 23** (Saturation of $\bigcup_n S_n$). *A consistent Lindenbaum extension is saturated:*

assumes $\text{consistent } (\text{Extend } S f)$ **and** $\text{surj } f$
shows $\text{saturated } (\text{Extend } S f)$

Proof. Guaranteed by construction. ◀

If we only constructed our set to be maximal consistent and tried to show that it was also saturated, we would run into trouble [40, p. 96]. First, given an arbitrary maximal consistent set S , it might be that a Henkin witness is missing because S includes every parameter available and every reuse of a parameter results in an inconsistency. Second, we might be unlucky and enumerate the negation of every suitable witness before enumerating the witness itself: we might always add the negation and never the witness. Following Henkin [21], I ensure saturation by adding the Henkin witnesses together with the existential formulas.

6.5 Hintikka Sets

Instead of showing the model existence theorem directly for maximal consistent saturated sets, it will be cleaner to show that Hintikka sets induce a model for their formulas and that our sets are in fact Hintikka sets.

The following definition characterizes a Hintikka set H over our syntax:

```

locale Hintikka =
  fixes  $H :: ('f, 'p) \text{ fm set}$ 
  assumes
     $FlsH: \perp \notin H$  and
     $ImpH: (p \longrightarrow q) \in H \longleftrightarrow (p \in H \longrightarrow q \in H)$  and
     $UniH: (\forall p \in H) \longleftrightarrow (\forall t. \langle t/0 \rangle p \in H)$ 

```

Hintikka sets are sets that are *saturated downwards* [40, p. 27] and induce a model for the formulas in them. Since the set should induce a model, \perp should never be present ($FlsH$). Following Forster et al. [11, Lemma 11], I enforce that the set *respects* both implication ($ImpH$) and universal quantification ($UniH$): a formula is in the Hintikka set if and only if the “evidence” for that formula is also present. Here, evidence is to be understood in terms of the Herbrand model given below.

6.5.1 Model Existence

The model induced by a Hintikka set H is very simple. It consists of a Herbrand structure [10] and a predicate denotation based on H itself:

Domain Herbrand universe: the universe of terms.

Function denotation The constructor \dagger , i.e. every function symbol evaluates to itself.

Predicate denotation Predicate P is true for terms ts exactly when $\dagger P \ ts \in H$.

Like in the work by Herbelin and Ilik [22], but unlike for instance the formalizations by Berghofer [3] and Forster et al. [11], the Herbrand universe includes all terms, not just those with no variables. I never formalize what it means for a formula to be closed. The Herbrand structure famously evaluates any term without variables to itself [10]. Or in this case:

► **Lemma 24** (Herbrand semantics). *Under any Herbrand structure and the specific environment $\#$, every term evaluates to itself:*

■ $(\# , \dagger) \ t = t$

Proof. By structural induction. ◀

I reuse the notation for semantics and abbreviate the model induced by H as $\llbracket H \rrbracket$:

abbreviation $hmodel \ (\llbracket - \rrbracket)$ **where** $\llbracket H \rrbracket \equiv \llbracket \# , \dagger , \lambda P \ ts. \dagger P \ ts \in H \rrbracket$

We now reach the model existence theorem.

► **Theorem 25** (Model existence). *When H is a Hintikka set, $\llbracket H \rrbracket$ satisfies exactly the formulas in H .*

```

assumes Hintikka  $H$ 
shows  $p \in H \longleftrightarrow \llbracket H \rrbracket \ p$ 

```

Proof. By well-founded induction on the size of the formula as given by *size-fm*. Thus the induction hypothesis applies to any formula that is smaller by this measure, i.e. to subformulas and to instances of universally quantified formulas (cf. Lemma 6). These are exactly the

formulas that appear in the Hintikka conditions. The proof proceeds by considering each type of formula. Since there is a Hintikka condition for every type, which corresponds exactly to the semantics of the induced model, Isabelle automatically proves each case. For instance, a universal formula p is in the Hintikka set iff every instance $\langle t/0 \rangle p$ is in the Hintikka set (*UniH*) iff every instance $\langle t/0 \rangle p$ holds in the induced model (by the induction hypothesis). ◀

6.5.2 Saturated MCSs are Hintikka Sets

Consider first the following correspondence between derivability and MCSs.

► **Lemma 26** (Derivability and MCSs). *A formula p is derivable from an MCS S iff p is in S :*

assumes *consistent S and maximal S*
shows $(\exists ps. \text{set } ps \subseteq S \wedge ps \vdash p) \longleftrightarrow p \in S$

Proof. The left to right direction follows from the maximality of MCSs. The right to left direction follows trivially from the derivability of any assumption (Lemma 13). ◀

I now show that maximal consistent saturated sets are Hintikka sets.

► **Lemma 27** (Saturated MCSs are Hintikka sets). *If the set H is consistent, maximal and saturated, it is a Hintikka set:*

assumes *consistent H and maximal H and saturated H*
shows *Hintikka H*

Proof. We need to prove each case of the Hintikka definition. Take first the *FIsH* case:

show $\perp \notin H$

We need to show that falsity does not appear in our set. This follows directly from Lemma 26 and the assumed consistency of H .

Consider next the *ImpH* case:

show $(p \longrightarrow q) \in H \longleftrightarrow (p \in H \longrightarrow q \in H)$

From left to right, by using Lemma 26 this simply becomes modus ponens: if both $p \longrightarrow q$ and p are derivable from H then q must be derivable from H . The right to left direction is similar. It relies on Lemma 26, contraposition and Lemma 21: that exactly one of a formula and its negation is present in an MCS.

Consider next the *UniH* case:

show $(\forall p \in H) \longleftrightarrow (\forall t. \langle t/0 \rangle p \in H)$

One direction follows directly from consistency of instantiation (Lemma 16) and the maximality of H . The other direction follows from saturation (and Lemma 21). ◀

6.6 Completeness Theorem

Isabelle can automatically prove the countability of our syntax:

instance *tm* :: (countable) countable
instance *fm* :: (countable, countable) countable

These commands provide instances of the surjective function *from-nat* that takes natural numbers and returns terms and formulas, respectively. I state the main theorem as follows.

► **Theorem 28** (Completeness). *Assume that formula p is valid under assumptions X and that X leaves infinitely many parameters unused. Then we can derive p from X .*

fixes $p :: ('f :: \text{countable}, 'p :: \text{countable}) \text{ fm}$
assumes $\forall (E :: - \Rightarrow 'f \text{ tm}) F G. (\forall q \in X. \llbracket E, F, G \rrbracket q) \longrightarrow \llbracket E, F, G \rrbracket p$
and *infinite* $(UNIV - \text{params } X)$
shows $\exists ps. \text{set } ps \subseteq X \wedge ps \vdash p$

Proof. By contradiction:

assume $\nexists ps. \text{set } ps \subseteq X \wedge ps \vdash p$
then have $*$: $\nexists ps. \text{set } ps \subseteq X \wedge ((\neg p) \# ps \vdash \perp)$

If no such list of assumptions exists, then (by classical reasoning on the object level) there is also no list that allows us to derive falsity when assuming $\neg p$.

I introduce some local abbreviations $?S$ and $?H$ (where $?$ is required by Isabelle):

let $?S = \{\neg p\} \cup X$
let $?H = \text{Extend } ?S \text{ from-nat}$

It is easy to see from $*$ above that $?S$ must be consistent and the extension $?H$ is therefore maximal consistent (Lemmas 20, 22):

have *consistent* $?S$
moreover have *infinite* $(UNIV - \text{params } ?S)$
ultimately have *consistent* $?H$ **and** *maximal* $?H$

$?H$ is saturated (Lemma 23) and Hintikka (Lemma 27):

moreover from this have *saturated* $?H$
ultimately have *Hintikka* $?H$

The model induced by $?H$ satisfies any formula in $?H$ (Theorem 25), including the starting set $?S$ (Lemma 17):

have $\llbracket ?H \rrbracket p$ **if** $p \in ?S$ **for** p
then have $\llbracket ?H \rrbracket (\neg p)$ **and** $\forall q \in X. \llbracket ?H \rrbracket q$

But this includes all formulas in X so by the assumed validity, $\llbracket H \rrbracket$ must also satisfy p and we reach our contradiction:

moreover from this have $\llbracket ?H \rrbracket p$
ultimately show *False*

The proof system is complete. ◀

The following abbreviation of validity in a specific Herbrand universe, with countably infinite function and predicate symbols, makes the result simpler to state:

abbreviation *valid* $:: (\text{nat}, \text{nat}) \text{ fm} \Rightarrow \text{bool}$ **where**
 $\text{valid } p \equiv \forall (E :: \text{nat} \Rightarrow \text{nat tm}) F G. \llbracket E, F, G \rrbracket p$

I fix the function and predicate symbols to be natural numbers but any countably infinite type works. One thing to note is that I only assume validity in one domain (the Herbrand universe), as I cannot quantify over the type I use to represent the domain. This is, however, a weaker assumption than assuming validity in all domains as is usually done.

► **Theorem 29** (Soundness and completeness). *Exactly the valid formulas are derivable:*

theorem *main*: $\text{valid } p \longleftrightarrow (\vdash p)$

Proof. By Theorems 9, 28. ◀

Only the definitions in Sections 3, 4 and Sections 5.1 to 5.5 must be inspected to trust the result. The definitions in this Section are only used for the proof.

7 Discussion

There are many choices to make in a formalization like this one. I choose to work with de Bruijn indices rather than named variables or Nominal Isabelle [42], which provides automation for this situation. While this choice makes it more complicated to explain the formalizations of e.g. semantics and quantifier instantiation, it makes the formalization self-contained. I hope to have demonstrated that the definitions themselves are simple, the functions are short and only a few simple lemmas are needed about them.

Recall the *GR* rule which is used in Lemma 15 to justify the consistency of fresh witnesses:

$$GR: \vdash q \longrightarrow \langle \star a / 0 \rangle p \implies a \notin \text{params } \{p, q\} \implies \vdash q \longrightarrow \forall p$$

Since I use de Bruijn indices, this could also be formalized without the use of a parameter a by *lifting* q , in the sense of \uparrow , to ensure that variable 0 in p is safe to generalize directly:

$$\vdash \uparrow q \rightarrow p \implies \vdash q \rightarrow \forall p$$

However, we would then need to ensure that the entire set S' in Lemma 15 is *lifted* in order to apply the rule. With the present *GR* rule, we simply ensure that a is chosen to be fresh. It would be interesting to try Laurent’s anti-locally nameless approach to quantifiers [27] and see whether this would yield a simpler formalization.

Another choice has been to simulate assumptions in derivations by a chain of implications. This trick applies directly to a one-sided calculus and makes it a lot simpler to work with, especially with some custom notation. It works especially well with Smullyan’s System Q1 where the generalization rule (*GR*) works under an implication. The semantic characterization of the tautology axiom, which works well with Isabelle’s proof automation, makes it even smoother since propositional reasoning becomes a non-issue.

One challenge was the realization that the variant *GR'* is more suitable than *GR*. Isabelle cannot tell us something like this, nor is the proof automation powerful enough to derive the rule automatically. The insight comes from experimenting with the formalization and proofs.

Some of these issues could also be resolved by starting from a natural deduction system rather than Smullyan’s Hilbert system. Natural deduction systems have a context built in, where I must simulate it with implications, and more *natural* rules for the connectives, which could be used instead of the semantic characterization of tautologies. It remains future work to adapt the formalization to this setting and review the potential benefits.

At this point in time there is a large body of formalizations to draw on. I am inspired by Berghofer’s formalization [3] of the completeness of natural deduction for first-order logic. Berghofer also formalizes Lindenbaum’s construction and my definition is close to his. My formalization of Hintikka sets and the model existence theorem, however, is both shorter (due to Forster et al. [11]) and, unlike Berghofer’s, works directly for open formulas (cf. Herbelin and Ilik [22]). As such, even though some notion has already been formalized, it can be beneficial to revisit it.

8 Conclusion and Future Work

I have used techniques from computer science like de Bruijn indices and functional programming to work in the meta-logic of the proof assistant Isabelle/HOL. Here, I have formalized the syntax and semantics of first-order logic and defined a simple axiomatic proof system for it. This definition has included careful considerations of the interplay between syntax and semantics, a semantic characterization of tautologies suitable for formalization and notational tricks like the use of implications to simulate assumptions.

I have then carried out a completeness proof for the Hilbert system in the style of Henkin, and using ideas from Lindenbaum, Hintikka and Herbrand along the way. The proof is direct: use Lindenbaum’s construction to extend a consistent set to a maximal consistent set, add Henkin witnesses of existential formulas during this construction, notice that the result is a Hintikka set and build a model in the Herbrand universe. Section 2 demonstrated the usefulness of this style in the formalization of other logics and proof systems. My formalization may serve as starting point for such endeavors: researchers can modify the existing definitions and proofs rather than start from scratch. Isabelle/HOL ensures that such modifications are correct and can help fill in gaps in the proofs when they arise. This provides an entry point to formalizing such a completeness proof.

In the future, however, I want to abstract this proof along several dimensions. First, the entire construction outlined above could potentially be given in the abstract and instantiated with a concrete proof system, witnessing function, notion of saturation, etc. Then it might be shared among the several formalizations of this method, and potential new ones. Popescu and Traytel [36] have already developed some syntax-independent logical infrastructure in their formal verification of an abstract account of Gödel’s incompleteness theorems. This future work could potentially build on theirs, extending it with the model existence theorem and more. Second, Smullyan gives many constructions in his uniform notation that abstracts over the concrete choice of syntax. I would like to abstract this formalization in a similar way: witnesses could be added for “ δ -formulas”, which might happen to be of the form $\neg (\forall p)$ like here or maybe of the form $\Diamond p$ as seen in hybrid logic [26].

I also want to extend the syntax, semantics, proof system and completeness proof to first-order logic with equality. I already handle function symbols, unlike Smullyan, but to get on par with Barwise, equality needs to be considered too. The Henkin style should scale well for this extension. The current formalization does, however, have the benefit of outlining the fundamental ideas of the completeness proof without too many auxiliary considerations. This is an advantage for adapting it to other logics.

I hope this formalization will serve as inspiration, and perhaps as a starting point, for further formalizations of logic.

References

- 1 Jon Barwise. An introduction to first-order logic. In Jon Barwise, editor, *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, pages 5–46. Elsevier, 1977. doi:10.1016/S0049-237X(08)71097-8.
- 2 Bruno Bentzen. A Henkin-style completeness proof for the modal logic S5. In Pietro Baroni, Christoph Benzmüller, and Yi N. Wáng, editors, *Logic and Argumentation - 4th International Conference, CLAR 2021, Hangzhou, China, October 20-22, 2021, Proceedings*, volume 13040 of *Lecture Notes in Computer Science*, pages 459–467. Springer, 2021. doi:10.1007/978-3-030-89391-0_25.
- 3 Stefan Berghofer. First-order logic according to Fitting. *Archive of Formal Proofs*, August 2007. Formal proof development. URL: <https://isa-afp.org/entries/FOL-Fitting.html>.
- 4 Jasmin Christian Blanchette. Formalizing the metatheory of logical calculi and automatic provers in Isabelle/HOL (invited talk). In Assia Mahboubi and Magnus O. Myreen, editors, *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019*, pages 1–13. ACM, 2019.
- 5 Jasmin Christian Blanchette, Andrei Popescu, and Dmitriy Traytel. Abstract completeness. *Archive of Formal Proofs*, April 2014. Formal proof development. URL: https://isa-afp.org/entries/Abstract_Completeness.html.

- 6 Jasmin Christian Blanchette, Andrei Popescu, and Dmitriy Traytel. Soundness and completeness proofs by coinductive methods. *Journal of Automated Reasoning*, 58(1):149–179, 2017. doi:10.1007/s10817-016-9391-3.
- 7 Patrick Braselmann and Peter Koepke. Gödel’s completeness theorem. *Formalized Mathematics*, 13(1):49–53, 2005.
- 8 Robert L. Constable and Mark Bickford. Intuitionistic completeness of first-order logic. *Annals of Pure and Applied Logic*, 165(1):164–198, 2014. doi:10.1016/j.apal.2013.07.009.
- 9 N.G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. In R.P. Nederpelt, J.H. Geuvers, and R.C. de Vrijer, editors, *Selected Papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*, pages 375–388. Elsevier, 1994. Reprinted from: *Indagationes Math.*, 34, 5, p. 381–392, by courtesy of the Koninklijke Nederlandse Akademie van Wetenschappen, Amsterdam. doi:10.1016/S0049-237X(08)70216-7.
- 10 Melvin Fitting. *First-Order Logic and Automated Theorem Proving, Second Edition*. Graduate Texts in Computer Science. Springer, 1996.
- 11 Yannick Forster, Dominik Kirst, and Dominik Wehr. Completeness theorems for first-order logic analysed in constructive type theory. *Journal of Logic and Computation*, 31(1):112–151, 2021. doi:10.1093/logcom/exaa073.
- 12 Asta Halkjær From. Synthetic completeness for a terminating Seligman-style tableau system. In Ugo de’Liguoro, Stefano Berardi, and Thorsten Altenkirch, editors, *26th International Conference on Types for Proofs and Programs, TYPES 2020, March 2-5, 2020, University of Turin, Italy*, volume 188 of *LIPICs*, pages 5:1–5:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.TYPES.2020.5.
- 13 Asta Halkjær From. Formalizing Henkin-style completeness of an axiomatic system for propositional logic. In *WESSLLI + ESSLLI Virtual Student Session*, 2021. Accepted for post-proceedings.
- 14 Asta Halkjær From, Anders Schlichtkrull, and Jørgen Villadsen. A sequent calculus for first-order logic formalized in Isabelle/HOL. In Stefania Monica and Federico Bergenti, editors, *Proceedings of the 36th Italian Conference on Computational Logic, Parma, Italy, September 7-9, 2021*, volume 3002 of *CEUR Workshop Proceedings*, pages 107–121. CEUR-WS.org, 2021. URL: <http://ceur-ws.org/Vol-3002/paper7.pdf>.
- 15 Asta Halkjær From. Formalizing a Seligman-style tableau system for hybrid logic. *Archive of Formal Proofs*, December 2019. Formal proof development. URL: http://isa-afp.org/entries/Hybrid_Logic.html.
- 16 Asta Halkjær From. A sequent calculus for first-order logic. *Archive of Formal Proofs*, July 2019. Formal proof development. URL: https://isa-afp.org/entries/FOL_Seq_Calc1.html.
- 17 Asta Halkjær From. Soundness and completeness of an axiomatic system for first-order logic. *Archive of Formal Proofs*, September 2021. Formal proof development. URL: https://isa-afp.org/entries/FOL_Axiomatic.html.
- 18 Herman Geuvers. Proof assistants: History, ideas and future. *Sadhana*, 34(1):3–25, 2009. doi:10.1007/s12046-009-0001-5.
- 19 Kurt Gödel. Die Vollständigkeit der Axiome des logischen Funktionenkalküls. *Monatshefte für Mathematik und Physik*, 37(1):349–360, 1930.
- 20 John Harrison. Formalizing basic first order model theory. In Jim Grundy and Malcolm C. Newey, editors, *Theorem Proving in Higher Order Logics, 11th International Conference, TPHOLs’98, Canberra, Australia, September 27 - October 1, 1998, Proceedings*, volume 1479 of *Lecture Notes in Computer Science*, pages 153–170. Springer, 1998. doi:10.1007/BFb0055135.
- 21 Leon Henkin. The discovery of my completeness proofs. *Bulletin of Symbolic Logic*, 2(2):127–158, 1996. doi:10.2307/421107.
- 22 Hugo Herbelin and Danko Ilik. An analysis of the constructive content of Henkin’s proof of Gödel’s completeness theorem. *Manuscript available online*, 2016.

- 23 Jacques Herbrand. *Recherches sur la théorie de la démonstration*. Number 110 in Thèses de l'entre-deux-guerres. Faculté des Sciences de L'Université de Paris, 1930.
- 24 Denis R. Hirschfeldt. *Slicing the Truth - On the Computable and Reverse Mathematics of Combinatorial Principles*, volume 28 of *Lecture Notes Series / Institute for Mathematical Sciences / National University of Singapore*. World Scientific, 2014. doi:10.1142/9208.
- 25 Danko Ilik. *Constructive Completeness Proofs and Delimited Control. (Preuves constructives de complétude et contrôle délimité)*. PhD thesis, École Polytechnique, Palaiseau, France, 2010. URL: <https://tel.archives-ouvertes.fr/tel-00529021>.
- 26 Klaus Froyen Jørgensen, Patrick Blackburn, Thomas Bolander, and Torben Braüner. Synthetic completeness proofs for Seligman-style tableau systems. In Lev D. Beklemishev, Stéphane Demri, and András Maté, editors, *Advances in Modal Logic 11, proceedings of the 11th conference on "Advances in Modal Logic," held in Budapest, Hungary, August 30 - September 2, 2016*, pages 302–321. College Publications, 2016. URL: <http://www.aiml.net/volumes/volume11/Joergensen-Blackburn-Bolander-Brauner.pdf>.
- 27 Olivier Laurent. An anti-locally-nameless approach to formalizing quantifiers. In Catalin Hritcu and Andrei Popescu, editors, *CPP '21: 10th ACM SIGPLAN International Conference on Certified Programs and Proofs, Virtual Event, Denmark, January 17-19, 2021*, pages 300–312. ACM, 2021. doi:10.1145/3437992.3439926.
- 28 Pierre Lescanne. From lambda-sigma to lambda-epsilon: a journey through calculi of explicit substitutions. In Hans-Juergen Boehm, Bernard Lang, and Daniel M. Yellin, editors, *Conference Record of POPL'94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, Oregon, USA, January 17-21, 1994*, pages 60–69. ACM Press, 1994. doi:10.1145/174675.174707.
- 29 James Margetson and Tom Ridge. Completeness theorem. *Archive of Formal Proofs*, September 2004. Formal proof development. URL: <http://isa-afp.org/entries/Completeness.html>.
- 30 Julius Michaelis and Tobias Nipkow. Propositional proof systems. *Archive of Formal Proofs*, June 2017. Formal proof development. URL: http://isa-afp.org/entries/Propositional_Proof_Systems.html.
- 31 Julius Michaelis and Tobias Nipkow. Formalized Proof Systems for Propositional Logic. In Andreas Abel, Fredrik Nordvall Forsberg, and Ambrus Kaposi, editors, *23rd International Conference on Types for Proofs and Programs (TYPES 2017)*, volume 104 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:16, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.TYPES.2017.5.
- 32 Tobias Nipkow. More Church-Rosser proofs. *Journal of Automated Reasoning*, 26(1):51–66, 2001. doi:10.1023/A:1006496715975.
- 33 Tobias Nipkow and Gerwin Klein. *Concrete Semantics - With Isabelle/HOL*. Springer, 2014. doi:10.1007/978-3-319-10542-0.
- 34 Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-45949-9.
- 35 Henrik Persson. Constructive completeness of intuitionistic predicate logic. *Licentiate thesis, Chalmers University of Technology*, 1996.
- 36 Andrei Popescu and Dmitriy Traytel. A formally verified abstract account of Gödel's incompleteness theorems. In Pascal Fontaine, editor, *Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings*, volume 11716 of *Lecture Notes in Computer Science*, pages 442–461. Springer, 2019. doi:10.1007/978-3-030-29436-6_26.
- 37 Anders Schlichtkrull. The resolution calculus for first-order logic. *Archive of Formal Proofs*, June 2016. Formal proof development. URL: http://isa-afp.org/entries/Resolution_FOL.html.
- 38 Anders Schlichtkrull. Formalization of the resolution calculus for first-order logic. *Journal of Automated Reasoning*, 61(1-4):455–484, 2018. doi:10.1007/s10817-017-9447-z.

- 39 Natarajan Shankar. Towards mechanical metamathematics. *Journal of Automated Reasoning*, 1(4):407–434, 1985.
- 40 Raymond M. Smullyan. *First-Order Logic*. Springer-Verlag, 1968.
- 41 Alfred Tarski. *Logic, Semantics, Metamathematics: Papers from 1923 to 1938*. Hackett Publishing, 1983.
- 42 Christian Urban. Nominal techniques in Isabelle/HOL. *Journal of Automated Reasoning*, 40(4):327–356, 2008. doi:10.1007/s10817-008-9097-2.
- 43 Wim Veldman. An intuitionistic completeness theorem for intuitionistic predicate logic. *Journal of Symbolic Logic*, 41(1):159–166, 1976. doi:10.1017/S0022481200051859.

CHAPTER 4

Verifying a Sequent Calculus Prover for First-Order Logic with Functions in Isabelle/HOL

Origin

Asta Halkjær From and Frederik Krogsdal Jacobsen. “Verifying a Sequent Calculus Prover for First-Order Logic with Functions in Isabelle/HOL”. In: *13th International Conference on Interactive Theorem Proving, ITP 2022, August 7-10, 2022, Haifa, Israel*. Ed. by June Andronick and Leonardo de Moura. Vol. 237. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 13:1–13:22. DOI: [10.4230/LIPIcs.ITP.2022.13](https://doi.org/10.4230/LIPIcs.ITP.2022.13).

Verifying a Sequent Calculus Prover for First-Order Logic with Functions in Isabelle/HOL

Asta Halkjær From ✉ 

DTU Compute, Technical University of Denmark, Kongens Lyngby, Denmark

Frederik Krogsdal Jacobsen ✉ 

DTU Compute, Technical University of Denmark, Kongens Lyngby, Denmark

Abstract

We describe the design, implementation and verification of an automated theorem prover for first-order logic with functions. The proof search procedure is based on sequent calculus and we formally verify its soundness and completeness in Isabelle/HOL using an existing abstract framework for coinductive proof trees. Our analytic completeness proof covers both open and closed formulas. Since our deterministic prover considers only the subset of terms relevant to proving a given sequent, we do so as well when building a countermodel from a failed proof. Finally, we formally connect our prover with the proof system and semantics of the existing SeCaV system. In particular, the prover can generate human-readable SeCaV proofs which are also machine-verifiable proof certificates.

2012 ACM Subject Classification Theory of computation → Automated reasoning; Theory of computation → Proof theory; Theory of computation → Program verification

Keywords and phrases Isabelle/HOL, SeCaV, First-Order Logic, Prover, Soundness, Completeness

Digital Object Identifier 10.4230/LIPIcs.ITP.2022.13

Supplementary Material *Software (Isabelle/HOL formalization in the Archive of Formal Proofs):* https://www.isa-afp.org/entries/FOL_Seq_Calc2.html [18]

Acknowledgements We would like to thank Agnes Moesgård Eschen, Alexander Birch Jensen, Anders Schlichtkrull, Simon Tobias Lund and Jørgen Villadsen for comments on drafts. We are very grateful to the anonymous reviewers for their thoughtful comments.

1 Introduction

While there are many automated theorem provers capable of proving theorems involving very large formulas and many lemmas, very few of them have formalized proofs of metatheoretical properties such as soundness and completeness. This leads to issues of trust: how do we know that the answers returned by automated theorem provers are actually correct? And do we know that our automated theorem provers will actually be able to prove what we want them to? Even those provers that can generate proof certificates to support their answers may not always be trustworthy, since some proof techniques lead to proofs that are very difficult to follow for a human, and are thus difficult to check for correctness.

Formalizing the soundness and completeness of a prover provides two crucial benefits. With a soundness result, we know that the prover does not erroneously accept an invalid formula and outputs a wrong proof of the formula. Thus, advanced features and optimizations cannot cause unforeseen flaws in the prover. Completeness of the prover is especially useful in combination with the possibility of generating readable proof certificates. With formalized completeness, we can use the prover as a tool to generate step-by-step proofs of any valid formula, and it can thus also be used to gain understanding, e.g. by students trying to understand why a counter-intuitive formula is valid. While there are some systems with formalized metatheory, they rarely include executable provers, often cannot generate proof certificates, and are often quite limited in their expressive power (cf. Section 1.1).



© Asta Halkjær From and Frederik Krogsdal Jacobsen;
licensed under Creative Commons License CC-BY 4.0

13th International Conference on Interactive Theorem Proving (ITP 2022).

Editors: June Andronick and Leonardo de Moura; Article No. 13; pp. 13:1–13:22

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we present an automated theorem prover for first-order logic with functions based on sequent calculus. We formalize its soundness and completeness in Isabelle/HOL. We reuse the syntax and semantics of first-order logic from the Sequent Calculus Verifier (SeCaV) system [16] (Section 2.1). We state the soundness and completeness of the prover with respect to the SeCaV proof system, its semantics and a bounded semantics that we introduce here. The prover can generate human-readable and machine-verifiable SeCaV proofs for valid formulas.

Our formalization instantiates an abstract framework of coinductive proof trees by Blanchette et al. [11] (Section 2.2). By instantiating the framework with concrete functions implementing our sequent calculus, it builds a prover for us (Section 3). By discharging further proof obligations, the framework proves that any proof tree built by our prover is either finite or contains an infinite path with certain properties. We then build either a SeCaV proof from the finite tree (Section 4) or a countermodel from the failed proof attempt (Section 5). As far as we are aware, we are the first to use the framework to prove soundness and completeness of a non-trivial executable prover (as opposed to simply a calculus).

Our prover is deterministic, fair and works on finite sequents. To handle the quantifiers we must thus build our countermodel in a Herbrand universe that contains only the subset of terms that actually appear in the failed proof. This idea is inspired by Ben-Ari’s textbook proof [2], where terms are either variables or constants, and by Ridge’s Isabelle proof [38], where only variables are considered. We are not aware of any previous formalization of this construction that handles functions. We consider all terms in our Herbrand universe, including those with free variables, yielding completeness for both open and closed formulas.

The prover is free software and the source code is available as supplementary material. This consists of around 3000 lines of Isabelle/HOL and 1300 lines of supplementary Haskell.

We summarize our main contributions:

- A formally verified sound and complete automated theorem prover for full first-order logic with functions.
- An analytic proof of completeness for both open and closed formulas for a deterministic prover via a bounded semantics.
- A method of translating the prover-generated certificates of validity into human-readable and machine-verifiable proofs in SeCaV.
- A concrete application of the abstract completeness framework, and a demonstration of how to obtain soundness and completeness of an actual, executable prover using the framework as a starting point.

We summarize the results and discuss the generated proofs, challenges encountered during the verification, prover limitations and future work in Section 6 before concluding in Section 7.

1.1 Related work

The present paper is a much improved version of the work started in the second author’s master’s thesis [21]. The Sequent Calculus Verifier (SeCaV) is a well-established proof system, and both soundness and completeness have been proven for the system [19]. The system has been used to teach students in several courses at the Technical University of Denmark [20, 47]. An online tool called the SeCaV Unshortener has been developed to allow input of proofs in a simple format which is then translated to an Isabelle proof [16].

Our prover is based on the abstract completeness framework by Blanchette et al. [10, 11]. The framework contains a simple example prover for propositional logic, and the original application of the framework was in the formalization of the metatheory of the Sledgehammer

tool for automated theorem proving within Isabelle/HOL [8]. Blanchette et al. [11] have used the framework to formalize soundness and completeness of a calculus for first-order logic with equality and in negation normal form. Their search is nondeterministic and they do not generate an executable prover like we do. As such, we improve on their work by using the framework to prove soundness and completeness of an executable prover.

A number of other systems have formally verified metatheories. NaDeA (Natural Deduction Assistant) by Villadsen et al. [49] is a web application that allows users to prove formulas with natural deduction. The metatheory of a model of the system is formalized in Isabelle/HOL, and the application allows export of proofs for verification in Isabelle. The Incredible Proof Machine by Breitner [12] is a web application that allows users to create proofs using a specialized graphical interface. The proof system is as strong as natural deduction, and a model of the system is formalized in Isabelle using the abstract framework by Blanchette et al. [11]. Neither system includes automated theorem provers; they are essentially simple proof assistants designed to aid students in understanding logical systems.

THINKER by Pelletier [35] is a proof system and an attached automated theorem prover. THINKER is a natural deduction system designed to allow for what the author calls “direct proofs”, as opposed to proofs based on reduction to a resolution system. THINKER was perhaps the first automated theorem prover designed specifically with “naturalness” in mind, as a reaction to the indirectness of resolution-based proof systems. MUSCADET by Pastre [34] is also an automated theorem prover based on natural deduction. The system distinguishes itself by also supporting usage of prior knowledge such as previously proven theorems through a Prolog knowledge base.

While there are many very advanced automated theorem provers such as Vampire [24], Zipperposition [3] and Z3 [13], their metatheory and implementations are rarely formalized. As a first step towards formally verifying modern provers, Schlichtkrull et al. [40] have formalized an ordered resolution prover for *clausal* first-order logic in Isabelle/HOL. Jensen et al. [22] formalized the soundness, but not the completeness, of a prover for first-order logic with equality in Isabelle/HOL. Villadsen et al. [50] verified a simple prover for first-order logic in Isabelle/HOL with the aim of allowing students to understand both the prover and the formalization. That work is based on an earlier formalization by Ridge and Margetson [38], but simplifies both the prover and the proofs to enable easier understanding by students. Neither of these two provers provide support for functions or generation of proof certificates.

Blanchette [5] gives an overview of a number of verification efforts including the metatheory of SAT solvers [6, 14, 29, 30, 43] and certificate checkers [26, 27], SMT solvers [28, 31, 45], the superposition calculus [37], resolution [36, 39, 41], a number of non-classical logics [15, 17, 42, 46, 48], and a wide range of proof systems for classical propositional logic [32, 33]. Some of these efforts are part of the IsaFoL project (Isabelle Formalization of Logic). Part of the goal is to develop “a methodology for formalizing modern research in automated reasoning”. Our work points in this direction too, by formally verifying a non-saturation-based prover.

2 Background

In this section, we briefly introduce the two existing things we build on: the Sequent Calculus Verifier (SeCaV) system and the abstract framework by Blanchette et al [11]. In particular, we have not modified these projects in any way for our use, and their designs thus significantly influence the design of our prover.

13:4 Verifying a Sequent Calculus Prover for First-Order Logic with Functions

```
datatype tm = Fun nat (tm list) | Var nat
```

```
datatype fm =
```

```
  Pre nat (tm list) | Imp fm fm | Dis fm fm | Con fm fm | Exi fm | Uni fm | Neg fm
```

■ **Figure 1** The syntax of the Sequent Calculus Verifier (parentheses added for clarity).

```
definition shift e v x  $\equiv \lambda n.$  if  $n < v$  then e n else if  $n = v$  then x else e ( $n - 1$ )
```

```
primrec semantics-term and semantics-list where
```

```
  semantics-term e f (Var n) = e n  
| semantics-term e f (Fun i l) = f i (semantics-list e f l)  
| semantics-list e f [] = []  
| semantics-list e f (t # l) = semantics-term e f t # semantics-list e f l
```

```
primrec semantics where
```

```
  semantics e f g (Pre i l) = g i (semantics-list e f l)  
| semantics e f g (Imp p q) = (semantics e f g p  $\longrightarrow$  semantics e f g q)  
| semantics e f g (Dis p q) = (semantics e f g p  $\vee$  semantics e f g q)  
| semantics e f g (Con p q) = (semantics e f g p  $\wedge$  semantics e f g q)  
| semantics e f g (Exi p) = ( $\exists x.$  semantics (shift e 0 x) f g p)  
| semantics e f g (Uni p) = ( $\forall x.$  semantics (shift e 0 x) f g p)  
| semantics e f g (Neg p) = ( $\neg$  semantics e f g p)
```

■ **Figure 2** The semantics of the Sequent Calculus Verifier (# separates the head and tail of a list).

2.1 The Sequent Calculus Verifier

The system is a one-sided sequent calculus for first-order logic with functions. Constants are encoded as functions with arity 0. Figure 1 gives the syntax of terms and formulas as Isabelle/HOL datatypes. The system uses de Bruijn indices to identify variables, while functions and predicates are named by natural numbers. Besides predicates, the system includes implication, disjunction, conjunction, existential quantification, universal quantification, and negation (in that order in Figure 1). Predicates and functions take their arguments as ordered lists of terms, which may be empty. Sequents are ordered lists of formulas. Parameterized datatypes are written in postfix notation, e.g. the type *tm list* of lists containing terms.

The semantics of a formula is due to Berghofer [4], who models the universe as a type variable like we do for now. The interpretation consists of an environment *e* for variables, a function denotation *f* and a predicate denotation *g*. The semantics of the system is standard and defined using the three recursive functions in Figure 2. The semantics of the logical connectives is defined using the connectives from the meta-logic in Isabelle/HOL. The *shift* function handles shifting de Bruijn-indices when interpreting quantifiers. We say that a sequent is valid when, under all interpretations, some formula in the sequent is satisfied.

The system has a number of proof rules, some of which are displayed in Figure 3 (abusing set notation for the membership and inclusion relations on lists – see the formalization for details and the remaining rules). The rules should be read from the bottom up, since we generally work backwards from a sequent we wish to prove. The rules are classified according to Smullyan’s uniform notation [44].

The first proof rule, BASIC, terminates the branch and applies when the sequent contains both a formula and its negation. Isabelle/HOL allows pattern matching only on the head of a list, so to simplify the specification of this rule, the positive formula must come first.

$$\begin{array}{c}
\frac{\text{Neg } p \in z}{\Vdash p, z} \text{ BASIC} \qquad \frac{\Vdash z \quad z \subseteq y}{\Vdash y} \text{ EXT} \qquad \frac{\Vdash p, z}{\Vdash \text{Neg } (\text{Neg } p), z} \text{ NEGNEG} \\
\\
\frac{\Vdash p, q, z}{\Vdash \text{Dis } p \, q, z} \text{ ALPHADIS} \qquad \frac{\Vdash \text{Neg } p, z \quad \Vdash \text{Neg } q, z}{\Vdash \text{Neg } (\text{Dis } p \, q), z} \text{ BETADIS} \\
\\
\frac{\Vdash p [\text{Var } 0/t], z}{\Vdash \text{Exi } p, z} \text{ GAMMAEXI} \qquad \frac{\Vdash \text{Neg } (p [\text{Var } 0/\text{Fun } i \, []]), z \quad i \text{ fresh}}{\Vdash \text{Neg } (\text{Exi } p), z} \text{ DELTAEXI}
\end{array}$$

■ **Figure 3** Sample proof rules for the Sequent Calculus Verifier (rules omitted here are similar).

The structural EXT rule can be applied to change the position of formulas in a sequent (permutation), duplicate an existing formula (contraction), and remove formulas that are not needed (weakening). It is crucial, since most rules in the system work only on the first formula in a sequent. Duplicating a formula is necessary if a quantified formula needs to be instantiated several times, since γ -rules (starting with GAMMA) destroy the original formula.

The NEGNEG rule removes a double negation from the first formula in a sequent. It can be considered an α -rule, but we keep it separate from the others because it does not generate two formulas. The ALPHADIS rule decomposes disjunctions (and similar for the ALPHAIMP and ALPHA CON rules omitted here). The BETADIS rule decomposes negated disjunctions and requires that two sequents are proven separately, creating branches in the proof tree (and similar for the BETACON, BETAIMP rules omitted here). This essentially moves the connective into the proof tree itself, since both branches now need to be proven separately. The GAMMAEXI rule instantiates an existential quantifier with any term t by substituting t for variable 0 in the quantified formula. The GAMMAUNI rule omitted here is similar. The DELTAEXI rule instantiates a negated existential quantifier in the first formula in a sequent with a fresh constant function, with fresh here meaning that the function identifier does not already occur anywhere in the sequent. The fresh constant cannot have any relationship to other terms in the sequent: it is *arbitrary*. Thus we could have used any other term without affecting the validity of the formula, which is exactly what is needed to prove a universally quantified (“there does not exist”) formula. The DELTAUNI rule omitted here is similar.

The proof system in Figure 3 has been formally verified to be sound and complete with regards to the semantics in Figure 2 by From et al. [19]. We use these results to relate our prover to SeCaV.

2.2 Abstract frameworks for soundness and completeness

Blanchette et al. [11] have formalized an abstract framework to facilitate soundness and completeness proofs by coinductive methods. In particular, they give abstract definitions that can be instantiated to a concrete sequent calculus or tableau prover. They facilitate proofs in the Beth-Hintikka style: the search “builds either a finite deduction tree yielding a proof (...) or an infinite tree from which a countermodel (...) can be extracted.” The framework consists of a number of Isabelle/HOL locales that must be instantiated and in return provide various definitions and proofs.

Locales [1, 23] allow the abstraction of definitions and proofs over given parameters. As an example, consider groups in algebra defined by a carrier set, a binary operation and the group axioms. With a locale, these can be specified abstractly and a number of operations

■ **Table 1** The *RuleSystem* locale with premises above the line and important conclusions below.

<i>eff</i>	Effect relation between a rule, a state and a finite set of resulting states.
<i>rules</i>	Stream of rules. The set of these is called <i>R</i> .
<i>S</i>	Set of well formed states.
<i>eff-S</i>	Proof that for any rule in <i>R</i> and proof state in <i>S</i> the <i>eff</i> -related states are in <i>S</i> .
<i>enabled-R</i>	Proof that for any state in <i>S</i> , some rule in <i>R</i> is <i>enabled</i> , i.e. applies to that state.
<i>mkTree</i>	A function from a stream of rules and a starting state to a tree of states and rules.
<i>wf-mkTree</i>	Proof that the tree generated by <i>mkTree</i> is well formed wrt. <i>eff</i> .

■ **Table 2** The *PersistentRuleSystem* locale which extends *RuleSystem* from Table 1.

<i>per</i>	Proof that if a rule <i>r</i> in <i>R</i> is enabled in a well formed state <i>s</i> and <i>s'</i> is <i>eff</i> -related to <i>s</i> by a rule <i>r'</i> in <i>R</i> distinct from <i>r</i> , then <i>r</i> is enabled in <i>s'</i> .
<i>epath-completeness-Saturated</i>	Proof that for any well formed state <i>s</i> , there exists either a well formed finite tree with <i>s</i> as root or a saturated escape path with <i>s</i> as root.

and results can then be given in the abstract. Later, we can instantiate the locale with a concrete group by providing the carrier set and binary operation, and proving that the group axioms are fulfilled. We then obtain instantiations of the results for our concrete group.

In this section we give an overview of the locales provided by the abstract framework: what they require and what they provide. We have condensed the Isabelle code into four tables for brevity, since the specific details of the framework are not our main focus. The exact definitions can be found in the Archive of Formal Proofs entry by Blanchette et al. [9].

First, two coinductive datatypes are crucial: a *tree* is finitely branching but can be infinitely deep, while a *stream* has no branching but is decidedly infinite (a list with no end).

Tables 1 and 2 cover the two locales *RuleSystem* and *PersistentRuleSystem* which are central for proving completeness. The locale premises are given above each vertical line and the (important) conclusions are given below. The locales require us to prove a number of things about three definitions. First, the *eff* relation specifies the effect of applying a rule to a state in our proof search. By (proof) state we mean a sequent, potentially coupled with additional information. The nodes of our proof tree will be proof states in this sense. Second, *rules* is a stream of rules for the prover to attempt to apply. Third, *S* is a set of well formed states (in our case simply the set of all states).

For the *RuleSystem* locale we must prove two things about these definitions. First, *eff-S*, that the set of well formed states *S* is closed under the *eff* relation on rules from the stream *rules*. Second, *enabled-R*, that no matter the proof state we have reached (in *S*), some rule in *rules* applies. In return we get the function *mkTree* which embodies our prover and a proof, *wf-mkTree*, that the tree produced by this prover is well formed. A tree is well formed (*wf*) when its children are well formed and the set of child states is *eff*-related to the node's state and applied rule.

For the *PersistentRuleSystem* locale, we must additionally prove *per*. This essentially states that rules do not interfere with each other: when we apply a rule, any other rules that were applicable before are still applicable. In return we get a theorem called *epath-completeness-Saturated*. An escape path (*epath*) is an infinite path in a well formed proof tree. Such a path is saturated (*Saturated*) when any rule which is enabled at some point on the path is eventually applied. Thus, this theorem states a completeness property for the *mkTree* function (on valid input): either it returns a well formed finite tree or a tree containing a saturated escape path (from which we can build a countermodel).

■ **Table 3** The *Soundness* locale.

<i>eff</i> , <i>rules</i>	As in Table 1 but states are now called sequents.
<i>structure</i>	Set of models.
<i>sat</i>	Satisfaction predicate on sequents and models.
<i>local-soundness</i>	Proof that the validity of a sequent (as given by <i>sat</i> and <i>structure</i>) follows from the validity of its children (as given by <i>eff</i> and <i>rules</i>).
<i>soundness</i>	Proof that any finite, well formed tree has a valid root.

■ **Table 4** The *RuleSystem-Code* locale.

<i>eff</i>	Effect <i>function</i> from a rule and a state to a finite set of resulting states.
<i>rules</i>	Stream of rules.
<i>i.mkTree</i>	Executable version of the <i>mkTree</i> function.

Table 3 covers the *Soundness* locale used to prove the soundness of resulting proof trees. Here, besides *eff* and *rules*, we must state a set of models, *structure*, and a satisfaction predicate, *sat*, on sequents and models. The locale then turns a local soundness proof, *local-soundness*, that validity of a sequent follows from validity of its children, into a global result, *soundness*, that any finite, well formed tree has a valid root.

Finally, to generate code we need to instantiate the locale *RuleSystem-Code* in Table 4, where *eff* must now be a deterministic relation, i.e. a function and *rules* is as before. In return we get an executable version of *mkTree* above, called *i.mkTree*.

RuleSystem-Code provides no guarantees on its own but we use the same underlying function in all four locales. We export this function to Haskell using Isabelle’s (unverified) code generation, code lemmas and a few (unverified) custom code-printing facilities. This step moves us from a verified prover inside Isabelle to a prover in Haskell which is based on a verified prover, but which is not itself verified.

3 Prover

In this section we explain the design of the proof search procedure driving our prover. The procedure does not use the proof system of SeCaV directly, but introduces a new set of similar proof rules that apply to entire sequents at once. This obviates the need for the structural EXT rule, which is therefore not present. Additionally, we remove the BASIC rule and let the prover close proof branches implicitly.

Before we can define what the rules do, we need a few auxiliary definitions. The function *generateNew* generates a function name that is fresh to a given list of terms. The function *subtermFms* computes the list of terms occurring in a list of functions. We define *subterms* as the list of all terms in a sequent, except that the list contains exactly *Fun 0 []* when it would otherwise be empty. This ensures that we always have some term to instantiate γ -formulas with. The function *sub* implements substitution in a standard way using de Bruijn indices. See the formalization [18] or the original SeCaV work [19] for details. The function *branchDone* computes whether a sequent is an axiom, i.e. whether the sequent contains both a formula and its negation. The prover uses this to determine when a branch of the proof tree is proven and can be closed.

We first define which “parts” of a single formula must be proven for a rule to apply:

13:8 Verifying a Sequent Calculus Prover for First-Order Logic with Functions

```

definition parts :: tm list  $\Rightarrow$  rule  $\Rightarrow$  fm  $\Rightarrow$  fm list list where
  parts A r f  $\equiv$  (case (r, f) of
    (NegNeg, Neg (Neg p))  $\Rightarrow$  [[p]]
  | (AlphaDis, Dis p q)  $\Rightarrow$  [[p, q]]
  | (BetaDis, Neg (Dis p q))  $\Rightarrow$  [[Neg p], [Neg q]]
  | (DeltaExi, Neg (Exi p))  $\Rightarrow$  [[Neg (sub 0 (Fun (generateNew A) []) p)]]
  | (GammaExi, Exi p)  $\Rightarrow$  [Exi p # map ( $\lambda$ t. sub 0 t p) A]
  | -  $\Rightarrow$  [[f]])

```

We have omitted some similar cases here (and will continue to do so in the sequel; see the formalization for the full definitions). The result of applying a rule is a list of lists of formulas with an implicit conjunction between lists and disjunction between inner formulas. For instance, the parts of *Dis p q* under *AlphaDis* state that we must prove either *p* or *q*. The definition takes a parameter *A*, which should be a list of terms present on the proof branch. For δ -rules, a function which does not appear in *A* is generated (ensuring soundness), and for γ -rules, the quantifier is instantiated with every term in *A* (ensuring completeness). Note that if the rule and formula do not match, the result simply contains the original formula. This means that rules are always enabled, but that they do nothing to most formulas.

To construct a proof tree, we need a function that computes the result of applying a rule to (all formulas in) a sequent. This is done by the following function (@ appends two lists):

```

primrec children :: tm list  $\Rightarrow$  rule  $\Rightarrow$  sequent  $\Rightarrow$  sequent list where
  children - - [] = []
| children A r (p # z) =
  (let hs = parts A r p; A' = remdups (A @ subtermFms (concat hs))
   in list-prod hs (children A' r z))

```

It first computes the effect of applying the rule to the first formula in the sequent (using the definition *parts*) and gives a name to the updated list of terms in the sequent (since δ - and γ -rules may introduce new terms). The function then goes through the rest of the sequent recursively, combining the generated child branches with the function *list-prod*:

```

primrec list-prod :: 'a list list  $\Rightarrow$  'a list list  $\Rightarrow$  'a list list where
  list-prod - [] = []
| list-prod hs (t # ts) = map ( $\lambda$ h. h @ t) hs @ list-prod hs ts

```

The type variable *'a* in the type signature means that the function works on lists of lists containing any type of elements.

It behaves in the following way (similar to the Cartesian product):

$$\text{set } (\text{list-prod } hs \ ts) = \{h @ t \mid h \ t. \ h \in \text{set } hs \wedge t \in \text{set } ts\}$$

For β -rules, the end result is a list of 2^n child branches, where *n* is the number of β -formulas in the sequent. These branches are ordered such that they correspond to the branches one would have obtained by applying the corresponding SeCaV β -rule *n* times. For all other rules, the end result is a single child branch. The parameter *A* to *children* should again be a list of terms present on the proof branch. We should be clear that *children* does not apply rules recursively to sub-formulas, but only to the “top layer.” If the application of a rule reveals a formula that this rule applies to again, this formula is left as is and only considered the next time *children* is applied to the sequent with that rule. For example, the result of calling *children* with the rule *ALPHADIS* and the sequent containing only the formula *Dis (Dis p q) r* is *Dis p q, r* and not *p, q, r*.

The prover needs to ensure that bound variables are instantiated with all terms on the current branch when a γ -rule is applied. For this reason, we define the *state* in a proof tree node to be a pair consisting of a list of terms appearing on the branch and a sequent. The list of terms will be used to instantiate the parameter A in the definitions above.

We are now ready to define the effect of applying a proof rule to a proof state:

```

primrec effect :: rule  $\Rightarrow$  state  $\Rightarrow$  state fset where
  effect r (A, z) =
    (if branchDone z then {} else
     fimage ( $\lambda z'. (remdups (A @ subterms z @ subterms z'), z')$ )
     (fset-of-list (children (remdups (A @ subtermFms z)) r z)))

```

To fit the types of the framework, the function returns a finite set (*fset*) instead of a list. If the sequent is an axiom, the branch is proven, and the function returns an empty set of child nodes, closing the branch. Otherwise, the function converts the result of the *children* function to a finite set, and adds any new terms to the list of terms in each child node.

Having defined what rules do, we now need a *stream* of them (*rules* in Table 1). We, somewhat arbitrarily, define a list of rules in the order α , δ , β , γ and cycle it to obtain a stream. For efficiency, we could run, say, all α - and δ -rules to completion before branching with the β -rules, but this cannot be encoded in the simple stream of rules without further machinery: one could imagine having larger “meta-rules” corresponding to groups of SeCaV rules. This would give a notion of “phases” where we would first run all the rules in one group, then all the rules in the next group in the stream etc. For simplicity (see Section 6.4) we apply single rules in a fixed order. This also trivially ensures fairness.

3.1 Applying the framework

We are now ready to apply the abstract completeness framework to obtain the actual proof search procedure (cf. Section 2.2). First, we define a relational version of the *effect* of a rule, called *eff*. To use the framework, we need to prove three properties: that the set of well formed proof states is closed under *eff* (*eff-S*), that it is always possible to apply some rule (*enabled-R*), and that the rules that can be applied are still possible to apply after applying other rules (*per*). We do not need to restrict the set of well formed proof states, so the first property is trivial. Since all of our rules can always be applied (they simply do nothing if they do not match the sequent), the other two properties are also trivial. We can thus instantiate the framework with our effect relation and stream of rules. This allows us to define the prover using the *mkTree* function from the framework:

definition *secavProver* \equiv *mkTree rules*

This function takes a list of terms and a sequent, and applies the rules in the stream in order to build a proof tree with the given sequent at the root, using our *eff* relation to determine the children of each node. The list of terms is used to collect the terms that occur in the sequents on each branch and should initially be empty (in the exported prover, the function is wrapped in another function to ensure that the list of terms is empty).

We call the sequent at the root of this proof tree the *root sequent*:

abbreviation *rootSequent* $t \equiv$ *snd* (*fst* (*root* t))

3.2 Making the prover executable

To actually make the prover executable, we need to specify that the stream of rules should be lazily evaluated, or the prover will never terminate. Additionally, we need to define the prover using the code interpretation of the framework to enable computation of some parts of the framework (cf. Table 4). After telling Isabelle how to translate operations on the *option* type to the *Maybe* type, this also allows us to export the prover to Haskell code.

We have implemented a few Haskell modules to drive the exported prover, and translate found proofs into the proof system of SeCaV. These modules are not formally verified, but the proofs generated in this manner can be verified by Isabelle. We have written an automated test suite that tests the unverified code for soundness and completeness by applying the prover to a number of valid formulas, then calling Isabelle to verify the generated proofs, and by applying the prover to a number of invalid formulas and confirming that it does not generate a proof (within 10 seconds). While these tests do not give us absolute certainty that the exported code and the hand-written Haskell modules are correct, they provide a reasonable amount of certainty when combined with the formal proofs of correctness of the proof search procedure within Isabelle.

4 Soundness

We use the abstract soundness framework (cf. Section 2.2) to prove that any sequent with a well formed and finite proof tree can be proved in SeCaV. It follows from the soundness of SeCaV that such sequents for which the prover terminates are semantically valid. The following lemma comprises the core of the result:

► **Lemma 1.** *If for all sequents z' in children $A \vdash z$, we can derive $\vdash pre @ z'$, and the term list A contains all parameters of pre and z , then we can derive $\vdash pre @ z$ itself:*

assumes $\forall z' \in \text{set}(\text{children } A \vdash z). (\vdash pre @ z')$
and $\text{paramss}(pre @ z) \subseteq \text{paramsts } A$
shows $\vdash pre @ z$

Proof. By induction on z for arbitrary pre and A .

For the empty sequent, the thesis holds immediately as we get by assumption and the definition of *children* that we can derive $\vdash pre$.

For the non-empty sequent with formula p as head and z as tail we have the following induction hypothesis (for any pre and A):

then have *ih*: $\forall z' \in \text{set}(\text{children } A \vdash z). (\vdash pre @ z') \implies (\vdash pre @ z)$
if $\text{paramss}(pre @ z) \subseteq \text{paramsts } A$ **for** $pre \vdash A$

We abbreviate the term list that the prover actually recurses on as $?A$. From the first assumption and the definition of *list-prod* we then have (*):

$\forall hs \in \text{set}(\text{parts } A \vdash p). \forall ts \in \text{set}(\text{children } ?A \vdash z). (\vdash pre @ hs @ ts)$

The proof continues by examining the possible cases for *parts*.

Take first the case where $r = \text{AlphaDis}$ and $p = \text{Dis } q \vdash r$. Then (*) states that we can derive $\vdash pre @ q \# r \# z'$ for all z' in *children* $?A \vdash r \vdash z$. We apply the induction hypothesis at pre extended with q and r , which is allowed since they are subformulas of p . We then get the derivation $\vdash pre @ q \# r \# z$. By the EXT and ALPHADIS rules from SeCaV we obtain the desired derivation $\vdash pre @ \text{Dis } q \vdash r \# z$.

The remaining α - and β -cases are similar. In the δ -cases we prove that the constant used by the prover is new to the sequent, as required by the SeCaV δ -rules.

In the γ -cases we get a derivation that includes both the γ -formula and all instances of it using terms from the list A . Here we induct on A to generalize each instance into the corresponding γ -formula and use EXT to contract this γ -formula with the existing occurrence.

When *parts* $A \ r \ p$ returns p , the thesis holds from $(*)$ and the induction hypothesis. \blacktriangleleft

We only need *pre* in the above lemma to make the induction hypothesis strong enough for the proof, so we can instantiate it afterwards.

► **Corollary 2** (Proof tree to SeCaV). *We derive a sequent from derivations of its children:*

assumes $\forall z' \in \text{set}(\text{children } A \ r \ z). (\Vdash z')$ **and** $\text{params } z \subseteq \text{paramts } A$
shows $\Vdash z$

We obtain the following soundness theorem from the abstract soundness framework.

► **Theorem 3** (Prover soundness wrt. SeCaV). *The root sequent of any finite, well formed proof tree has a derivation in SeCaV:*

assumes $t \text{ finite } t \text{ and } wf \ t$
shows $\Vdash \text{rootSequent } t$

5 Completeness

The completeness proof is heavily based on the abstract completeness framework. As noted in Section 2.2, however, the framework only takes us so far. First, we duplicate the output of Table 2, since the *mkTree* function is unhelpfully abstracted away by an existential quantifier. This could easily be changed in the framework and should be considered for the next release.

► **Lemma 4** (Prover cases). *The proof tree generated by the prover is either finite and well formed or there exists a saturated escape path with our initial state as root:*

defines $t \equiv \text{secavProver } (A, z)$
shows $(fst(\text{root } t) = (A, z) \wedge wf \ t \wedge t \text{ finite } t) \vee$
 $(\exists \text{ steps. } fst(\text{shd steps}) = (A, z) \wedge \text{epath steps} \wedge \text{Saturated steps})$

In the first case, the sequent has a proof (cf. Section 4). In the second case, we need to build a countermodel from the saturated escape path to contradict validity of the sequent. The rest of this section does exactly that. Inspired by Ben-Ari [2] and Ridge [38], we start off by giving a definition of Hintikka sets over a restricted set of terms (Section 5.1). We show that the set of formulas on saturated escape paths fulfill all Hintikka requirements when we take the set of terms to be the terms on the path (Section 5.2). We then define a countermodel for any formula in such a set using a new semantics that bounds quantifiers by an explicit set rather than by types alone (Section 5.3). Finally we tie these results together to show that the prover terminates for all sequents that are valid under our new semantics (Section 5.4). In Section 6.1 we use existing results to prove completeness of the prover wrt. the SeCaV semantics.

13:12 Verifying a Sequent Calculus Prover for First-Order Logic with Functions

```

locale Hintikka =
  fixes  $H :: fm\ set$ 
  assumes
    Basic:  $Pre\ n\ ts \in H \implies Neg\ (Pre\ n\ ts) \notin H$  and
    AlphaDis:  $Dis\ p\ q \in H \implies p \in H \wedge q \in H$  and
    BetaDis:  $Neg\ (Dis\ p\ q) \in H \implies Neg\ p \in H \vee Neg\ q \in H$  and
    GammaExi:  $Exi\ p \in H \implies \forall t \in terms\ H. sub\ 0\ t\ p \in H$  and
    DeltaExi:  $Neg\ (Exi\ p) \in H \implies \exists t \in terms\ H. Neg\ (sub\ 0\ t\ p) \in H$  and
     $\vdots$ 
    Neg:  $Neg\ (Neg\ p) \in H \implies p \in H$ 

```

■ **Figure 4** Abridged list of requirements for a set of formulas H to be a Hintikka set.

5.1 Hintikka

First, by the *terms* of a set of formulas H we mean the following:

definition $terms\ H \equiv if\ (\bigcup p \in H. set\ (subtermFm\ p)) = \{\} \text{ then } \{Fun\ 0\ []\}$
 $\text{ else } (\bigcup p \in H. set\ (subtermFm\ p))$

This set contains an arbitrary (but fixed) constant, $Fun\ 0\ []$, when H itself contains no terms. Otherwise it contains all subterms of all formulas in H .

Figure 4 contains an abridged definition of a Hintikka set H . Here, we use a locale slightly differently to the previous ones, in that we have specify no conclusions, only premises: the formula set H and the requirements *Basic*, *AlphaDis*, etc. The omitted requirements are similar to the ones shown. This use simply allows us to assume *Hintikka* H in a theorem and know that the set H then fulfills the stated requirements. Similarly, we can prove that a set H is *Hintikka* by proving that it fulfills the requirements. It is important to note that in the γ - and δ -cases, the quantifiers only range over the *terms* of H .

5.2 Saturated escape paths are Hintikka

The following definition forgets all structure of a path and reduces it to a set of formulas:

definition $tree-fms\ steps \equiv \bigcup ss \in sset\ steps. set\ (pseq\ ss)$

The function *sset* returns the set of steps and *pseq* extracts the sequent from each.

Given a saturated escape path *steps*, we want to prove that *tree-fms steps* is a Hintikka set. For instance, if *Dis* $p\ q$ appears on the path, then both p and q should too. The prover is designed to make this property of its proof trees as evident as possible: formulas unaffected by a given rule are easily shown to be preserved by the application of that rule and any rule immediately applies to all its affected formulas, regardless of their position in the sequent.

We will need a number of intermediate results.

5.2.1 Unaffected formulas

We define the predicate *affects* to hold for a rule and a formula, when that rule does not preserve the formula (thus no rule *affects* a γ -formula, since the γ -rules of the prover, unlike those of SeCaV, preserve the original formula). For instance, *affects* *AlphaDis* (*Dis* $p\ q$) holds while *affects* *BetaCon* (*Dis* $p\ q$) does not.

We then prove the following key preservation lemma:

► **Lemma 5** (*effect preserves unaffected formulas*). Assume formula p occurs in sequent z and the rule r does not affect p . Then p also occurs in all children of z as given by effect ($|\in|$ denotes membership of a finite set):

assumes $p \in \text{set } z$ and $\neg \text{affects } r \ p$ and $(B, z') |\in| \text{effect } r \ (A, z)$
 shows $p \in \text{set } z'$

Proof. The function *parts* preserves unaffected formulas (proof by cases) so *children* does as well (proof by induction on the sequent) and thus *effect* does too. ◀

We lift this to escape paths:

► **Lemma 6** (*Escape paths preserve unaffected formulas*). Assume formula p occurs in some sequent at the head of an escape path which consists of a prefix *pre*, where none of the rules affect p , and a suffix *suf*. Then p occurs at the head of *suf*:

assumes $p \in \text{set } (\text{pseq } (\text{shd steps}))$ and *epath steps* and $\text{steps} = \text{pre} @ - \text{suf}$ and
 $\text{list-all } (\text{not } (\lambda \text{step. affects } (\text{snd step}) \ p)) \ \text{pre}$
 shows $p \in \text{set } (\text{pseq } (\text{shd suf}))$

Next, notice the following property of streams:

► **Lemma 7** (*Eventual prefix*). When a property P eventually holds of a stream, then the stream is comprised of a prefix of n (possibly zero) elements for which P does not hold and then a suffix that starts with an element for which P does hold:

assumes $\text{ev } (\text{holds } P) \ xs$
 shows $\exists n. \text{list-all } (\text{not } P) \ (\text{stake } n \ xs) \wedge \text{holds } P \ (\text{sdrop } n \ xs)$

Saturation states that a rule is eventually applied and Lemmas 6 and 7 combine to state that any affected formulas are preserved until then.

5.2.2 Affected formulas

Knowing that formulas are preserved as desired, we need to know that they are broken down as desired. The following lemma (proof omitted here) states this in general via *parts*:

► **Lemma 8** (*Parts in effect*). For any formula p in a sequent z , the effect of rule r on z includes some part of r 's effect on p :

assumes $p \in \text{set } z$ and $(B, z') |\in| \text{effect } r \ (A, z)$
 shows $\exists C \ xs. \text{set } A \subseteq \text{set } C \wedge xs \in \text{set } (\text{parts } C \ r \ p) \wedge \text{set } xs \subseteq \text{set } z'$

This is easier to understand when we specialize the rule and the formula:

► **Corollary 9.** Example effect of the *NegNeg* rule on a double-negated formula p :

corollary $\text{Neg } (\text{Neg } p) \in \text{set } z \implies (B, z') |\in| \text{effect } \text{NegNeg } (A, z) \implies p \in \text{set } z'$

5.2.3 Hintikka requirements

We then need to prove the following:

► **Theorem 10** (*Hintikka escape paths*). Saturated escape paths fulfill all Hintikka requirements:

assumes *epath steps* and *Saturated steps*
 shows *Hintikka* (*tree-fms steps*)

Proof. This boils down to proving each requirement of Figure 4 (and those omitted there). We give a couple of examples and refer to the formalization for the full details.

For *Basic*, assume towards a contradiction that both a predicate and its negation appear on the branch. By preservation of formulas (Lemma 6), both appear in the same sequent at some point. But then *branchDone* holds for that sequent, so it has no children and the branch would terminate. This contradicts that escape paths are infinite, so *Basic* must hold.

For *AlphaDis*, assume that *Dis p q* appears on the branch. Then it appears at some step n . By saturation of the escape path, *AlphaDis* is eventually applied at some (earliest) step $n + k$. By Lemma 6, *Dis p q* is preserved until then. So by the effect of rule *AlphaDis*, both p and q appear at step $n + k + 1$. The cases for the β - and δ -requirements are very similar.

For *GammaExi* assume that *Exi p* occurs at step n . We need to show that it is instantiated with all terms that (eventually) appear on the branch. Fix an arbitrary such term t . There must be some point m where t appears in a sequent. Thus at every point greater than m , term t appears in the term list which is part of the proof state. By saturation, at some step greater than $n + m + 1$, rule *GammaExi* is applied. The formula *Exi p* is preserved until this stage (Lemma 6) and the term list only grows, so t is too. Thus, at the next step, *sub 0 t p* occurs on the branch as desired. ◀

5.3 Countermodel

We need to build a countermodel for any formula in a Hintikka set to contradict the validity of any formula on a saturated escape path. We do this in the usual term model with a (bounded) Herbrand interpretation. Unfortunately, we cannot build a countermodel in the original semantics where the universe is specified as a type, since we cannot form the type of terms in a given Hintikka set (the *typedef* command does not support free variables). Instead, we introduce a custom bounded semantics.

5.3.1 Bounded semantics

The bounded semantics is exactly like the usual semantics (cf. Figure 2) except for an extra argument u , standing for the universe, which bounds the range of the quantifiers in the following cases:

$$\begin{aligned} | \text{usemantics } u \ e \ f \ g \ (\text{Exi } p) &= (\exists x \in u. \text{usemantics } u \ (\text{SeCaV.shift } e \ 0 \ x) \ f \ g \ p) \\ | \text{usemantics } u \ e \ f \ g \ (\text{Uni } p) &= (\forall x \in u. \text{usemantics } u \ (\text{SeCaV.shift } e \ 0 \ x) \ f \ g \ p) \end{aligned}$$

This leads to the following natural requirements on environments e and function denotations f , namely that they must stay inside u :

definition *is-env* $u \ e \equiv \forall n. \ e \ n \in u$

definition *is-fdenot* $u \ f \equiv \forall i \ l. \ \text{list-all } (\lambda x. \ x \in u) \ l \longrightarrow f \ i \ l \in u$

In general, we only consider environments and function denotations that satisfy these requirements and call them (and any model based on them) *well formed*. When $u = \text{UNIV}$, we do not actually bound the quantifiers and the two semantics coincide.

The SeCaV proof system (cf. Figure 3) is sound for the bounded semantics too.

► **Theorem 11** (SeCaV is sound for the bounded semantics). *Given a SeCaV derivation of sequent z and a well formed model, some formula p in z is satisfied in that model:*

assumes $\vdash z$ **and** *is-env* $u \ e$ **and** *is-fdenot* $u \ f$
shows $\exists p \in \text{set } z. \ \text{usemantics } u \ e \ f \ g \ p$

Proof. The proof closely resembles the original soundness proof (cf. [19]). ◀

We abbreviate validity of a sequent in the bounded semantics as *uvalid*:

abbreviation $uvalid\ z \equiv \forall u\ (e :: nat \Rightarrow tm)\ f\ g.\ is\text{-}env\ u\ e \longrightarrow is\text{-}fdenot\ u\ f \longrightarrow (\exists p \in set\ z.\ usemantics\ u\ e\ f\ g\ p)$

Namely, for all universes and well formed models, some formula in the sequent is satisfied in the bounded semantics at that universe by that model.

5.3.2 Model construction

Our countermodel is given by a bounded Herbrand interpretation where terms are interpreted as themselves when they appear in the universe *terms* H and as an arbitrary term otherwise.

► **Definition 12** (Countermodel induced by Hintikka set S). We abbreviate the model as MS :

abbreviation $ES\ n \equiv \text{if } Var\ n \in \text{terms } S \text{ then } Var\ n \text{ else } SOME\ t.\ t \in \text{terms } S$
abbreviation $FS\ i\ l \equiv \text{if } Fun\ i\ l \in \text{terms } S \text{ then } Fun\ i\ l \text{ else } SOME\ t.\ t \in \text{terms } S$
abbreviation $GS\ n\ ts \equiv Neg\ (Pre\ n\ ts) \in S$
abbreviation $MS \equiv usemantics\ (\text{terms } S)\ (ES)\ (FS)\ (GS)$

The definition of G is what makes this a countermodel rather than a model: a predicate is satisfied exactly when its negation is present in the Hintikka set.

Importantly, these definitions are *well formed*:

► **Lemma 13** (Well formed countermodel). *Definition 12 is well formed:*

shows $is\text{-}env\ (\text{terms } S)\ (ES)$
shows $is\text{-}fdenot\ (\text{terms } S)\ (FS)$

Proof. By the construction of E and F and the nonemptiness of *terms* S . ◀

► **Theorem 14** (Model existence). *The given model falsifies any formula p in Hintikka set S :*

assumes *Hintikka* S
shows $(p \in S \longrightarrow \neg MS\ p) \wedge (Neg\ p \in S \longrightarrow MS\ p)$

Proof. By induction on the size of the formula p (substitution instances are smaller than the quantified formulas they arise from). The second part of the thesis is needed when the Hintikka requirements concern negated formulas. We show a few cases here and refer to the formalization for the full details. The cases omitted here are similar to those shown.

Assume $p = Pre\ n\ ts$ occurs in S . We need to show that the given model falsifies p . Since *terms* S is downwards closed by construction, ts is interpreted as itself by the bounded Herbrand interpretation. Moreover, by the *Basic* requirement, we know that $Neg\ p$ is not in S and is therefore satisfied. Thus, p is falsified.

Assume $p = Dis\ q\ r$ occurs negated in S . Then by the *BetaDis* requirement, either $Neg\ q$ or $Neg\ r$ occurs in S . The induction hypothesis applies to these, so p is satisfied as desired.

Assume $p = Uni\ q$ occurs in S . By the *DeltaUni* requirement, so does some instance $sub\ 0\ t\ q$ for a term t in *terms* S . By the induction hypothesis, this is falsified by MS , and by its origin, t is interpreted as itself. Thus, we have a counterexample that falsifies p .

Assume $p = Exi\ q$ occurs in S . By the *GammaExi* requirement, so do all instances using terms from S . Thus, these are all falsified by the model. These terms from S are interpreted as themselves by definition so we have no witness for p in *terms* S and MS falsifies it. ◀

We note that the above proof works for open and closed formulas alike because we consider both bound and free variables to be subterms of a formula.

5.4 Result

We start off by proving completeness for *uvalid* sequents. We need to relate these to saturated escape paths.

► **Lemma 15** (Saturated escape paths contradict uvalidity). *A sequent z with a saturated escape path, steps, cannot be uvalid:*

assumes $\text{fst}(\text{shd steps}) = (A, z)$ **and** epath steps **and** *Saturated steps*
shows $\neg \text{uvalid } z$

Proof. Assume towards a contradiction that z is *uvalid*. By Theorem 10 the formulas on *steps* form a Hintikka set S . Every formula p in z also occurs in S , so by Theorem 14, the well formed model $M S$ (Lemma 13) falsifies all of them. This contradicts the uvalidity of z . ◀

This leads to completeness for *uvalid* sequents:

► **Theorem 16** (Completeness wrt. *uvalid*). *The prover terminates for uvalid sequents:*

assumes *uvalid* z
defines $t \equiv \text{secavProver } (A, z)$
shows $\text{fst}(\text{root } t) = (A, z) \wedge \text{wf } t \wedge \text{tfinite } t$

Proof. From the abstract framework (Lemma 4), either the thesis holds or a saturated escape path exists for our sequent, but assumed uvalidity and Lemma 15 contradict the latter. ◀

► **Corollary 17** (Completeness wrt. SeCaV). *Termination for sequents derivable in SeCaV:*

assumes $\Vdash z$
defines $t \equiv \text{secavProver } (A, z)$
shows $\text{fst}(\text{root } t) = (A, z) \wedge \text{wf } t \wedge \text{tfinite } t$

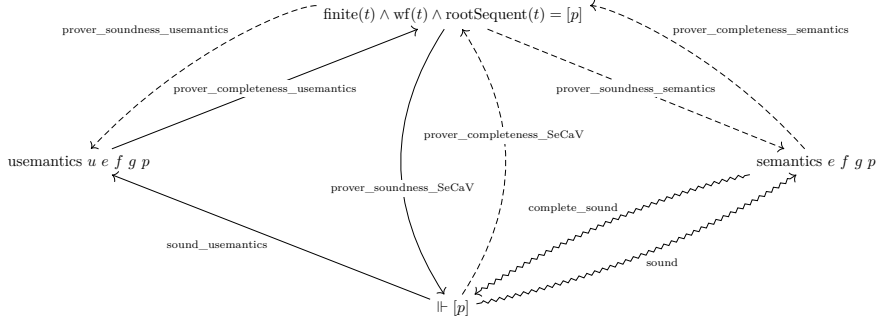
Proof. By the soundness of SeCaV (Theorem 11) and Theorem 16 for *uvalid* sequents. ◀

6 Results and discussion

We have presented an automated theorem prover for the Sequent Calculus Verifier system. The prover is capable of proving a number of selected exercise formulas very quickly, including formulas which are quite difficult for humans to prove. The prover does have some limitations, mostly related to performance and length of the generated proofs, since our proof search procedure is not very optimized for either of these metrics. In particular, our prover always instantiates quantified formulas with all terms in the sequent and breaks down all formulas as much as possible, even when some formulas are “obviously” irrelevant to the proof.

6.1 Summary of theorems

We have proven soundness and completeness of the proof search procedure with regards to the proof system of SeCaV (see Figure 3). For soundness, this was done directly (in Theorem 3), while we took a detour through our notion of a bounded semantics to prove completeness (in Theorems 11 and 16, which lead to Corollary 17). To justify the introduction of our bounded semantics, we can use the existing soundness and completeness theorems of the SeCaV proof system [19] and our results to prove that validity in the two semantics coincide. Additionally, a number of easy corollaries further linking the prover, the proof system and the two semantics follow from our results, and have been collected in Figure 5. In the figure, the interpretations are implicitly universally quantified and for the bounded semantics we only consider well formed interpretations.



■ **Figure 5** Overview of our results. Solid arrows represent our main contributions, squiggly arrows represent theorems of the existing SeCaV system, and dashed arrows represent easy corollaries.

6.2 Example proofs

Famously, we must beware of a program that has only been proven correct, but not tested. To demonstrate that the automated theorem prover works, we examine some simple generated proofs. The prover generates proofs in the SeCaV Unshortener format: first comes the formula to be proven, then the names of proof rules to apply and the resulting sequent after each application, with each formula in a sequent on its own line. Arguments to predicates and functions are given in square brackets and parentheses are used to disambiguate formulas.

We start with perhaps the simplest possible classical example, that $\neg p \vee p$. Figure 6a shows the proof generated by the prover. This is the shortest possible proof of the formula in the SeCaV system, and the prover is thus on par with a human in this very simple case.

The next example is $\neg p(a) \vee \exists x.p(x)$. Figure 6b contains the generated proof. It can be shortened since the quantified formula only needs to be instantiated once, by a . However, the prover always duplicates a γ -formula before instantiating it with *all* terms on the branch.

6.3 Verification challenges

While verifying the prover, we discovered that our initial version was unsound due to a missing update of the term list when applying (multiple) δ -rules to a sequent. The attempted soundness proof failed in exactly this case, pointing us directly to the issue. Thus, the formal verification caught a critical flaw that we had missed in our testing and helped us fix it.

We have designed the prover to be easily verified and it mostly was. Especially the abstract framework worked well for our novel case with a deterministic prover for first-order logic. One obstacle, however, was in using a type to represent the domain in the SeCaV semantics (cf. Figure 2). To build the countermodel, we need the domain to contain only the terms on the saturated escape path, but we cannot form this type, which depends on a local variable, in Isabelle/HOL. Here we would benefit from Isabelle integration of the work by Kunčar and Popescu [25] which adds exactly this capability to higher-order logic. Instead we introduced the bounded semantics (“the set-based relativization” in their terminology [25]) and proved a new soundness result for it (cf. Section 5.3.1). Otherwise the largest issue was dealing with substitutions using de Bruijn indices. We are excited to see how recent work by Blanchette et al. [7] for reasoning about syntax with bindings improves matters in this area.

<pre> Dis (Neg p) p AlphaDis Neg p p Ext p Neg p Basic </pre>	<pre> Dis (Neg (p [a])) (Exi (p [0])) AlphaDis Neg (p [a]) Exi (p [0]) Ext Exi (p [0]) Exi (p [0]) Neg (p [a]) GammaExi [a] p [a] Exi (p [0]) Neg (p [a]) </pre>	<pre> Ext Exi (p [0]) Exi (p [0]) Neg (p [a]) p [a] GammaExi [0] p [0] Exi (p [0]) Neg (p [a]) p [a] Ext p [a] Neg (p [a]) Exi (p [0]) p [0] Basic </pre>
(a) $\neg p \vee p$.	(b) $\neg p(a) \vee \exists x.p(x)$.	

Figure 6b continued.

■ **Figure 6** Proofs generated by the prover in SeCaV Unshortener format.

6.4 Limitations and future work

There are a number of limitations and possibilities for optimization in the proof search itself. Most importantly, the focus of the procedure is on completeness, not performance. Our prover is much slower than state-of-the-art provers such as Vampire [24], but our goal was not to compete on speed, but simply to show that formal verification of provers with advanced features such as generation of proof certificates and support for functions is possible. The prover also cannot output counterexamples, even though these can be detected in some cases: our prover simply never terminates on invalid formulas.

We believe that the approach used for our prover is extendable to more sophisticated and optimized proof search procedures, albeit with considerably more work needed to formally verify them. The most obvious opportunity for optimization is controlling the order of proof rules. In systems with unordered sequents, it is generally better to apply as many α -rules as possible before applying β -rules to avoid duplicating work, but the prover simply applies rules in a fixed order. As mentioned in Section 3, this optimization can be done by working with “meta-rules” corresponding to groups of SeCaV rules such that a meta-rule e.g. applies as many α -rules as possible before continuing to the next “phase” of the proof. We have attempted to implement this, but found that it complicates the proofs considerably since this idea makes it much harder to determine when a proof rule is actually applied. In the proof of fairness and the proof that the formulas on saturated escape paths form Hintikka sets, we need to know that certain formulas are preserved until proof rules are eventually applied to them. By introducing phases in the proof, proving this becomes much more difficult, since we then need to prove that each phase actually ends (requiring some measure which depends on the specific sequents in question), and to locate each rule within the meta-rule it is part of. We thus leave optimizations in this vein as future work. We note that, since the SeCaV system requires application of the EXT rule to permute sequents, and proof rules only apply to the first formula in a sequent, the optimization described above may not always reduce the number of SeCaV proof steps needed to prove a formula, and some heuristics would probably be needed to produce reasonably short proofs in all cases.

Another optimization could be to only support closed formulas and thus reduce the number of subterms of a given formula. For our current Herbrand interpretation, we need variables to be subterms, but if we only considered closed terms, we could do away with this.

The length of proofs could also be optimized by performing more post-processing of the found proofs, for example by removing unnecessary instantiations or rule applications that do not contribute to proving a branch. This would not improve the performance in the sense that the prover would still spend the same amount of time finding the proof, but it could reduce the length of some proofs significantly. The proof trees generated by the prover already require some (unverified) post-processing to obtain proofs in the SeCaV system. It would be interesting to move these steps from Haskell into Isabelle/HOL and extend the proofs to cover them.

7 Conclusion

We have designed, implemented and verified an automated theorem prover for first-order logic with functions in Isabelle/HOL. We have used an existing framework in a novel way to get us part of the way towards completeness and extended existing techniques on countermodels over restricted domains to reach our destination. We build on the existing SeCaV system and contribute an automatic way of finding derivations to the project. Thus, we have demonstrated the utility of Isabelle/HOL for implementing and verifying executable software and the strength of its libraries in doing so. Our prover handles the full syntax of first-order logic with functions and constructs human-readable proof certificates in a sequent calculus. We hope our work inspires others to verify more sophisticated provers in the same vein.

References

- 1 Clemens Ballarín. Locales: A module system for mathematical theories. *Journal of Automated Reasoning*, 52(2):123–153, 2014. doi:10.1007/s10817-013-9284-7.
- 2 Mordechai Ben-Ari. *Mathematical Logic for Computer Science*. Springer London, 2012. doi:10.1007/978-1-4471-4129-7.
- 3 A. Bentkamp, J. Blanchette, S. Tourret, and P. Vukmirović. Superposition for Full Higher-order Logic. In A. Platzer and G. Sutcliffe, editors, *Proceedings of the 28th International Conference on Automated Deduction*, number 12699 in Lecture Notes in Computer Science, pages 396–412. Springer-Verlag, 2021. doi:10.1007/978-3-030-79876-5_23.
- 4 Stefan Berghofer. First-order logic according to Fitting. *Archive of Formal Proofs*, August 2007. Formal proof development. URL: <https://isa-afp.org/entries/FOL-Fitting.html>.
- 5 Jasmin Christian Blanchette. Formalizing the metatheory of logical calculi and automatic provers in Isabelle/HOL (invited talk). In Assia Mahboubi and Magnus O. Myreen, editors, *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019*, pages 1–13. ACM, 2019. doi:10.1145/3293880.3294087.
- 6 Jasmin Christian Blanchette, Mathias Fleury, Peter Lammich, and Christoph Weidenbach. A verified SAT solver framework with learn, forget, restart, and incrementality. *Journal of Automated Reasoning*, 61(1-4):333–365, 2018. doi:10.1007/s10817-018-9455-7.
- 7 Jasmin Christian Blanchette, Lorenzo Gheri, Andrei Popescu, and Dmitriy Traytel. Bindings as bounded natural functors. *Proc. ACM Program. Lang.*, 3(POPL):22:1–22:34, 2019. doi:10.1145/3290335.
- 8 Jasmin Christian Blanchette and Andrei Popescu. Mechanizing the metatheory of Sledgehammer. In Pascal Fontaine, Christophe Ringeissen, and Renate A. Schmidt, editors, *Frontiers of Combining Systems*, pages 245–260, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-40885-4_17.

- 9 Jasmin Christian Blanchette, Andrei Popescu, and Dmitriy Traytel. Abstract completeness. *Archive of Formal Proofs*, April 2014. Formal proof development. URL: https://isa-afp.org/entries/Abstract_Completeness.html.
- 10 Jasmin Christian Blanchette, Andrei Popescu, and Dmitriy Traytel. Unified classical logic completeness. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *Automated Reasoning*, pages 46–60, Cham, 2014. Springer International Publishing. doi:10.1007/978-3-319-08587-6_4.
- 11 Jasmin Christian Blanchette, Andrei Popescu, and Dmitriy Traytel. Soundness and completeness proofs by coinductive methods. *Journal of Automated Reasoning*, 58:149–179, 2017. doi:10.1007/s10817-016-9391-3.
- 12 Joachim Breitner. Visual theorem proving with the Incredible Proof Machine. In J. Blanchette and S. Merz, editors, *Interactive Theorem Proving*, volume ITP 2016. Springer, Cham, 2016. doi:10.1007/978-3-319-43144-4_8.
- 13 Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. doi:10.1007/978-3-540-78800-3_24.
- 14 Mathias Fleury. Optimizing a verified SAT solver. In Julia M. Badger and Kristin Yvonne Rozier, editors, *NASA Formal Methods – 11th International Symposium, NFM 2019, Houston, TX, USA, May 7-9, 2019, Proceedings*, volume 11460 of *Lecture Notes in Computer Science*, pages 148–165. Springer, 2019. doi:10.1007/978-3-030-20652-9_10.
- 15 Asta Halkjær From. Formalized soundness and completeness of epistemic logic. In Alexandra Silva, Renata Wassermann, and Ruy J. G. B. de Queiroz, editors, *Logic, Language, Information, and Computation – 27th International Workshop, WoLLIC 2021, Virtual Event, October 5-8, 2021, Proceedings*, volume 13038 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2021. doi:10.1007/978-3-030-88853-4_1.
- 16 Asta Halkjær From, Frederik Krogsdal Jacobsen, and Jørgen Villadsen. SeCaV: A sequent calculus verifier in Isabelle/HOL. In Mauricio Ayala-Rincon and Eduardo Bonelli, editors, *Proceedings 16th Logical and Semantic Frameworks with Applications*, Buenos Aires, Argentina (Online), 23rd – 24th July, 2021, volume 357 of *Electronic Proceedings in Theoretical Computer Science*, pages 38–55. Open Publishing Association, 2022. doi:10.4204/EPTCS.357.4.
- 17 Asta Halkjær From. Epistemic logic: Completeness of modal logics. *Archive of Formal Proofs*, October 2018. Formal proof development. URL: https://isa-afp.org/entries/Epistemic_Logic.html.
- 18 Asta Halkjær From and Frederik Krogsdal Jacobsen. A sequent calculus prover for first-order logic with functions. *Archive of Formal Proofs*, January 2022. Formal proof development. URL: https://isa-afp.org/entries/FOL_Seq_Calc2.html.
- 19 Asta Halkjær From, Alexander Birch Jensen, Anders Schlichtkrull, and Jørgen Villadsen. Teaching a formalized logical calculus. *Electronic Proceedings in Theoretical Computer Science*, 313:73–92, 2020. doi:10.4204/EPTCS.313.5.
- 20 Asta Halkjær From, Jørgen Villadsen, and Patrick Blackburn. Isabelle/HOL as a meta-language for teaching logic. *Electronic Proceedings in Theoretical Computer Science*, 328:18–34, October 2020. doi:10.4204/eptcs.328.2.
- 21 Frederik Krogsdal Jacobsen. Formalization of logical systems in Isabelle: An automated theorem prover for the Sequent Calculus Verifier. Master's thesis, Technical University of Denmark, June 2021. URL: <https://findit.dtu.dk/en/catalog/2691928304>.
- 22 Alexander Birch Jensen, John Bruntse Larsen, Anders Schlichtkrull, and Jørgen Villadsen. Programming and verifying a declarative first-order prover in Isabelle/HOL. *AI Communications*, 31(3):281–299, 2018. doi:10.3233/AIC-180764.
- 23 Florian Kammüller, Markus Wenzel, and Lawrence C. Paulson. Locales – A sectioning concept for Isabelle. In Yves Bertot, Gilles Dowek, André Hirschowitz, Christine Paulin-Mohring, and Laurent Théry, editors, *Theorem Proving in Higher Order Logics, 12th International Conference, TPHOLs'99, Nice, France, September, 1999, Proceedings*, volume 1690 of *Lecture Notes in Computer Science*, pages 149–166. Springer, 1999. doi:10.1007/3-540-48256-3_11.

- 24 Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. *Lecture Notes in Computer Science*, 8044, 2013. doi:0.1007/978-3-642-39799-8_1.
- 25 Ondrej Kunčar and Andrei Popescu. From types to sets by local type definition in higher-order logic. *Journal of Automated Reasoning*, 62(2):237–260, 2019. doi:10.1007/s10817-018-9464-6.
- 26 Peter Lammich. The GRAT tool chain. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing – SAT 2017*, pages 457–463, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-66263-3_29.
- 27 Peter Lammich. Efficient verified (UN)SAT certificate checking. *Journal of Automated Reasoning*, 64(3):513–532, 2020. doi:10.1007/s10817-019-09525-z.
- 28 Stephane Lescuyer. *Formalizing and Implementing a Reflexive Tactic for Automated Deduction in Coq*. Phd thesis, Université Paris Sud – Paris XI, January 2011. URL: <https://tel.archives-ouvertes.fr/tel-00713668>.
- 29 Filip Marić. Formal verification of modern SAT solvers. *Archive of Formal Proofs*, July 2008. Formal proof development. URL: <https://isa-afp.org/entries/SATSolverVerification.html>.
- 30 Filip Marić. Formal verification of a modern SAT solver by shallow embedding into Isabelle/HOL. *Theoretical Computer Science*, 411(50):4333–4356, 2010. doi:10.1016/j.tcs.2010.09.014.
- 31 Filip Marić, Mirko Spasić, and René Thiemann. An incremental simplex algorithm with unsatisfiable core generation. *Archive of Formal Proofs*, August 2018. Formal proof development. URL: <https://isa-afp.org/entries/Simplex.html>.
- 32 Julius Michaelis and Tobias Nipkow. Propositional proof systems. *Archive of Formal Proofs*, June 2017. Formal proof development. URL: https://isa-afp.org/entries/Propositional_Proof_Systems.html.
- 33 Julius Michaelis and Tobias Nipkow. Formalized Proof Systems for Propositional Logic. In Andreas Abel, Fredrik Nordvall Forsberg, and Ambrus Kaposi, editors, *23rd International Conference on Types for Proofs and Programs (TYPES 2017)*, volume 104 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:16, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.TYPES.2017.5.
- 34 Dominique Pastre. MUSCADET 2.3: A knowledge-based theorem prover based on natural deduction. *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2083:685–689, 2001. doi:10.1007/3-540-45744-5_56.
- 35 Francis Jeffry Pelletier. Automated natural deduction in THINKER. *Studia Logica*, 60(1):3–43, 1998. doi:10.1023/A:1005035316026.
- 36 Nicolas Peltier. Propositional resolution and prime implicates generation. *Archive of Formal Proofs*, March 2016. Formal proof development. URL: <https://isa-afp.org/entries/PropResPI.html>.
- 37 Nicolas Peltier. A variant of the superposition calculus. *Archive of Formal Proofs*, September 2016. Formal proof development. URL: <https://isa-afp.org/entries/SuperCalc.html>.
- 38 Tom Ridge and James Margetson. A mechanically verified, sound and complete theorem prover for first order logic. *Lecture Notes in Computer Science*, 3603:294–309, 2005. doi:10.1007/11541868_19.
- 39 Anders Schlichtkrull. Formalization of the resolution calculus for first-order logic. *Journal of Automated Reasoning*, 61(1–4):455–484, 2018. doi:10.1007/s10817-017-9447-z.
- 40 Anders Schlichtkrull, Jasmin Christian Blanchette, and Dmitriy Traytel. A verified prover based on ordered resolution. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019*, pages 152–165, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3293880.3294100.

- 41 Anders Schlichtkrull, Jasmin Christian Blanchette, Dmitriy Traytel, and Uwe Waldmann. Formalizing Bachmair and Ganzinger’s ordered resolution prover. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Automated Reasoning*, pages 89–107, Cham, 2018. Springer International Publishing. doi:10.1007/978-3-319-94205-6_7.
- 42 Anders Schlichtkrull and Jørgen Villadsen. Paraconsistency. *Archive of Formal Proofs*, December 2016. Formal proof development. URL: <https://isa-afp.org/entries/Paraconsistency.html>.
- 43 Natarajan Shankar and Marc Vaucher. The mechanical verification of a DPLL-based satisfiability solver. *Electronic Notes in Theoretical Computer Science*, 269:3–17, 2011. Proceedings of the Fifth Logical and Semantic Frameworks, with Applications Workshop (LSFA 2010). doi:10.1016/j.entcs.2011.03.002.
- 44 Raymond M. Smullyan. *First-order logic*. Dover Publications, 1995.
- 45 Mirko Spasić and Filip Marić. Formalization of incremental simplex algorithm by stepwise refinement. In Dimitra Giannakopoulou and Dominique Méry, editors, *FM 2012: Formal Methods*, pages 434–449, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi:10.1007/978-3-642-32759-9_35.
- 46 Jørgen Villadsen, Asta Halkjær From, Alexander Birch Jensen, and Anders Schlichtkrull. Interactive theorem proving for logic and information. In Roussanka Loukanova, editor, *Natural Language Processing in Artificial Intelligence — NLPinAI 2021*, pages 25–48, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-030-90138-7_2.
- 47 Jørgen Villadsen and Frederik Krogsdal Jacobsen. Using Isabelle in two courses on logic and automated reasoning. In João F. Ferreira, Alexandra Mendes, and Claudio Menghi, editors, *Formal Methods Teaching*, pages 117–132, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-91550-6_9.
- 48 Jørgen Villadsen and Anders Schlichtkrull. Formalizing a paraconsistent logic in the Isabelle proof assistant. In Abdelkader Hameurlain, Josef Küng, Roland Wagner, and Hendrik Decker, editors, *Transactions on Large-Scale Data- and Knowledge-Centered Systems XXXIV: Special Issue on Consistency and Inconsistency in Data-Centric Applications*, pages 92–122, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg. doi:10.1007/978-3-662-55947-5_5.
- 49 Jørgen Villadsen, Andreas Halkjær From, and Anders Schlichtkrull. Natural deduction assistant (NaDeA). *Electronic Proceedings in Theoretical Computer Science*, 290(290):14–29, 2019. doi:10.4204/EPTCS.290.2.
- 50 Jørgen Villadsen, Anders Schlichtkrull, and Andreas Halkjær From. A verified simple prover for first-order logic. *CEUR Workshop Proceedings*, 2162:88–104, 2018. URL: <http://ceur-ws.org/Vol-2162/#paper-08>.




CHAPTER 5

Synthetic Completeness for a Terminating Seligman-Style Tableau System

Origin

Asta Halkjær From. “Synthetic Completeness for a Terminating Seligman-Style Tableau System”. In: *26th International Conference on Types for Proofs and Programs, TYPES 2020, March 2-5, 2020, University of Turin, Italy*. Ed. by Ugo de'Liguoro, Stefano Berardi, and Thorsten Altenkirch. Vol. 188. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 5:1–5:17. DOI: [10.4230/LIPIcs.TYPES.2020.5](https://doi.org/10.4230/LIPIcs.TYPES.2020.5).

Synthetic Completeness for a Terminating Seligman-Style Tableau System

Asta Halkjær From   

Technical University of Denmark, Kongens Lyngby, Denmark

Abstract

Hybrid logic extends modal logic with nominals that name worlds. Seligman-style tableau systems for hybrid logic divide branches into blocks named by nominals to achieve a local proof style. We present a Seligman-style tableau system with a formalization in the proof assistant Isabelle/HOL. Our system refines an existing system to simplify formalization and we claim termination from this relationship. Existing completeness proofs that account for termination are either analytic or based on translation, but synthetic proofs have been shown to generalize to richer logics and languages. Our main result is the first synthetic completeness proof for a terminating hybrid logic tableau system. It is also the first formalized completeness proof for any hybrid logic proof system.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics

Keywords and phrases Hybrid logic, Seligman-style tableau, synthetic completeness, Isabelle/HOL

Digital Object Identifier 10.4230/LIPIcs.TYPES.2020.5

Supplementary Material *Model (Isabelle/HOL formalization in the Archive of Formal Proofs (4900+ lines))*: https://isa-afp.org/entries/Hybrid_Logic.html

archived at `swb:1:cnt:5a830ce17c70be797b343d9078bf19c43b0f2145`

Acknowledgements We thank Patrick Blackburn, Thomas Bolander, Torben Braüner, Klaus Froyen Jørgensen and Jørgen Villadsen for discussions and Lars Hupel, Jasmin Blanchette and the anonymous reviewers for comments on the paper.

1 Introduction

Hybrid logic increases the expressiveness of modal logic by adding a special sort of propositional symbol called *nominals* to the syntax. In regular modal logic we can only reference worlds indirectly through the modalities, but nominals, that are true at exactly one world, name worlds explicitly. A nominal i gives rise to the satisfaction operator $@_i$ that states what world a formula is true “at.” These features make hybrid logic well suited for applications like temporal logic [3], description logic [5] and epistemic logics for social networks [24].

There are many proof systems for classical hybrid logic [4] and we focus on tableau systems in the following. Early work relied on loop checks to ensure termination [10] but Bolander and Blackburn introduced a calculus that guarantees finite branches through local restrictions [9]. Their completeness proof is *analytic*, meaning that they reason about open branches directly. Blackburn et al. [4] introduced the Seligman-style [25] system ST with a more local proof style than previous systems. Jørgensen et al. [21] later introduced a *synthetic* completeness proof for ST and showed that it scales with extensions to the logic. The synthetic approach involves reasoning about maximal consistent sets and their properties [13, 26] and this also opens the way for other developments, notably interpolation results [1].

Blackburn et al. [4] restricted ST into the terminating ST^* but showed completeness by translation from the system by Bolander and Blackburn [9]. The synthetic completeness proof for ST relies on a symmetry in branches that neither terminating system has. We present system ST^A , a refinement of ST^* suitable for formalization, which is formalized in the simple type theory of Isabelle/HOL [23]. Its proof of completeness fills a gap as the first synthetic completeness proof for a terminating tableau system for hybrid logic. It is also the first standalone completeness proof for a terminating Seligman-style system and, to our knowledge, the first formalization of any proof system for hybrid logic.



© Asta Halkjær From;
licensed under Creative Commons License CC-BY 4.0

26th International Conference on Types for Proofs and Programs (TYPES 2020).

Editors: Ugo de'Liguoro, Stefano Berardi, and Thorsten Altenkirch; Article No. 5; pp. 5:1–5:17



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The formalization provides absolute trust in the correctness of the completeness proof and serves as a companion to this paper, where the proofs can be seen in full detail.

Our system closely resembles ST^* but with restrictions that are simpler to formalize and we argue for termination based on this relationship. Formalizing termination remains future work since we want a direct proof, not one based on translation. Blanchette [6] gives an overview of efforts to formalize the metatheory of logical calculi and provers in Isabelle.

Other formalizations of hybrid logic itself exist. Doczkal and Smolka [12] formalized hybrid logic with nominals in constructive type theory using the proof assistant Coq. They gave algorithmic proofs of small model theorems and computational decidability of satisfiability, validity, and equivalence of formulas. In Isabelle/HOL, Linker [22] formalized the semantic embedding of a spatio-temporal multi-modal logic with a hybrid logic-inspired *at*-operator.

Our work is classical but hybrid logic also has a constructive variant. Braüner and de Paiva [11] defined intuitionistic hybrid logic, and a natural deduction system, and Galmiche and Salhi [19] showed its decidability via a sequent calculus. Jia and Walker [20] interpreted modal proofs as distributed programs with nominals denoting places in the network.

We formalized the synthetic completeness of ST with some of the simpler ST^* restrictions required for termination in our MSc thesis [17]. A short paper by From et al. [14] briefly described an even earlier version of the formalization and we mentioned the present completeness proof in a short presentation at Advances in Modal Logic 2020 [18].

The paper continues as follows. First, we give the syntax and semantics of basic hybrid logic (Section 2). We introduce the proof system, corresponding rule restrictions and some consequences (Section 3). Next, we show a number of properties of the system that are useful for the completeness proof (Section 4). After that, we prove completeness of the system and show how our proof relates to existing work (Section 5). We then show how ST^A relates to ST^* and argue for our choice of restrictions. From this relationship we claim that ST^A must be terminating by sketching a possible translation (Section 6). We briefly discuss some points about the formalization (Section 7) and conclude with future work (Section 8).

2 Syntax and Semantics

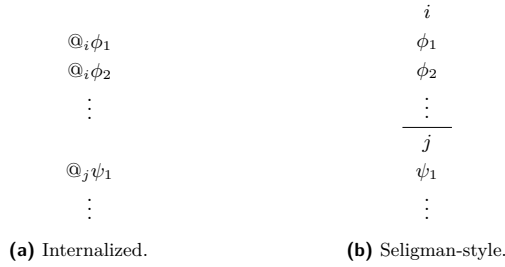
The well-formed formulas of the basic hybrid logic are given by the following grammar, where we use p as a propositional symbol and i, j, k, a, b for nominals.

$$\phi, \psi ::= p \mid i \mid \neg\phi \mid \phi \vee \psi \mid \Diamond\phi \mid @_i\phi$$

The \Diamond operator is the usual possibility modality and $@_i$ is the aforementioned *satisfaction operator*. A formula of the form $@_i\phi$ is called a *satisfaction statement*.

We interpret the language on Kripke models $\mathfrak{M} = (W, R, V)$. The frame (W, R) consists of a non-empty set of worlds W and a binary accessibility relation R between them. V is the valuation of propositional symbols. An *assignment* g maps nominals to elements of W ; if $g(i) = w$ we say that nominal i *denotes* w . Formula satisfiability is defined as follows:

$\mathfrak{M}, g, w \models p$	iff	$w \in V(p)$
$\mathfrak{M}, g, w \models i$	iff	$g(i) = w$
$\mathfrak{M}, g, w \models \neg\phi$	iff	$\mathfrak{M}, g, w \not\models \phi$
$\mathfrak{M}, g, w \models \phi \vee \psi$	iff	$\mathfrak{M}, g, w \models \phi$ or $\mathfrak{M}, g, w \models \psi$
$\mathfrak{M}, g, w \models \Diamond\phi$	iff	for some w', wRw' and $\mathfrak{M}, g, w' \models \phi$
$\mathfrak{M}, g, w \models @_i\phi$	iff	$\mathfrak{M}, g, g(i) \models \phi$



■ **Figure 1** Internalized and Seligman-style tableau branches.

3 Our Seligman-Style Tableau System

Our proof system of choice is tableau. In tableau we decompose an initial set of *root* formulas into a tree structure and show unsatisfiability by reaching a contradiction on each branch. This is called “closing” the branch and a branch that cannot be closed remains “open.” If we can close every branch that emerges then the root formulas have a *closing tableau*.

A hybrid logic formula is true relative to a given world and our proof system must handle this. Internalized tableau systems, as depicted in Figure 1a, encode the information in every formula on the branch by working exclusively with satisfaction statements. We follow instead the Seligman style [25] adapted to tableau systems by Blackburn et al. [4]. Here, the information is attached to a group of formulas at once by dividing the branch into *blocks* as depicted in Figure 1b. The first formula on each block is ensured to be a nominal and called the *opening nominal*. It denotes the world that the formulas on the block are true at. We occasionally call a block’s opening nominal its “type” and use the following shorthands:

► **Definition 1** (ϕ at i). *If a formula ϕ occurs on a block with opening nominal i , then we say that ϕ occurs “on an i -block” or simply that ϕ occurs “at i .”*

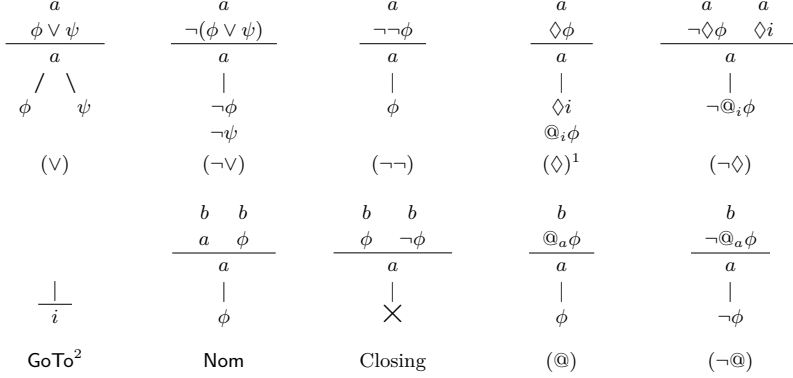
3.1 Proof System

Figure 2 gives our tableau rules. We give the rule output below the *vertical* lines and the rule input above them. The opening nominal of the latest, *current*, block is given below the horizontal line. Above each input formula we write the opening nominal of the block it occurs on. When a rule has multiple input we write these pairs side by side. Any formula on the current block may be used as input under the same restrictions on opening nominals.

► **Example 2.** Consider the $(\neg\neg)$ rule: if $\neg\neg\phi$ occurs on an a -block and the current block is an a -block, then ϕ is a legal extension of the branch. The intuition for the **Nom** rule is that the current opening nominal a occurs on a b -block so nominals a and b must denote the same world and it is sound to copy ϕ from b to a . The (\Diamond) rule witnesses its input formula, $\Diamond\phi$, with a fresh *witnessing nominal* i by producing an accessibility formula, $\Diamond i$, and a satisfaction statement, $@_i\phi$, saying that ϕ holds at the reachable world denoted by i .

► **Remark 3.** In the internalized system, cf. Figure 1a, we may work on a formula prefixed by $@_i$ one moment and one prefixed by $@_k$ the next. The Seligman-style blocks give rise to a more local proof style by delegating this perspective switch, e.g. from i to k , to the **GoTo** rule that opens a new block with corresponding opening nominal.

The soundness proof for ST^A follows existing work [4, 14] (cf. the formalization).



¹ i is fresh and ϕ is not a nominal.

² i is not fresh.

■ **Figure 2** Our Seligman-style tableau system ST^A .

3.2 Restrictions for Termination

Besides the side conditions, we need to impose the following four restrictions on the system to ensure that we eventually run out of applicable rules (inspired by Blackburn et al. [4]):

- S1** The output of a non-GoTo rule must include a formula new to the current block type.
- S2** The (\Diamond) rule can only be applied to input $\Diamond\phi$ on an a -block if $\Diamond\phi$ is not already witnessed at a by formulas $\Diamond i$ and $@_i\phi$ for some *witnessing nominal* i .
- S3** We associate *potential*, a natural number n , with each line in the tableau. GoTo must decrement the number, the other rules increment it and we may start from any amount.
- S4** We parameterize the proof system by a fixed set of nominals A and impose the following:
 - a. The nominal introduced by the (\Diamond) rule is not in A .
 - b. For any nominal i , Nom only applies to a formula $\phi = i$ or $\phi = \Diamond i$ when $i \in A$.

Restrictions S1 and S2 prevent us from applying the same rule to the same input repeatedly. We motivate restriction S3 by the following examples and restriction S4 in Section 3.3.

► **Example 4.** In Figure 3a we prove the validity of $\neg@_i\phi \vee @_i\phi$ by constructing a closing tableau for its negation. We start from potential 0 in the fourth column. Notice how regular rule applications build up potential that is then discharged to open a new block on line 5.

► **Example 5.** In Figure 3b we start from the unsatisfiable formula $@_i\neg i$ and potential n . Restriction S3 prevents infinite applications of GoTo and eventually forces us to make progress (or we might get stuck if no rules apply).

► **Remark 6.** The choice of a fresh opening nominal for the root block ensures that we do not close the branch because of an interplay between the formula itself and the opening nominal (imagine starting from $\neg i$ on a block with opening nominal i).

Given restrictions S3 and S4 we say that a branch has a closing tableau *with respect to* a set of allowed nominals A and potential n . We also introduce the following shorthand:

► **Definition 7** (Allowed ϕ). A formula ϕ is allowed by A if it meets condition S4b.

0.	a		
1.	$\neg(\neg @_i \phi \vee @_i \phi)$	[0]	
2.	$\neg\neg @_i \phi$	($\neg\vee$) 1	[1]
3.	$\neg @_i \phi$	($\neg\vee$) 1	[1]
4.	$@_i \phi$	($\neg\neg$) 2	[2]
5.	i	GoTo	[1]
6.	$\neg\phi$	($\neg @$) 3	[2]
7.	ϕ	($@$) 4	[3]
	\times		

(a) Building up potential.

0.	a		
1.	$@_i \neg i$		[n]
2.	i	GoTo	[$n-1$]
3.	i	GoTo	[$n-2$]
\vdots	\vdots	\vdots	\vdots
$n+1.$	i	GoTo	[0]
$n+2.$	$\neg i$	($@$) 1	[1]
	\times		

(b) Running out of potential.

■ **Figure 3** Two examples of potential.

3.3 Nominal Asymmetry

See Blackburn et al. [4] for why a restriction like **S4** is needed. They conclude:

We ... have to enforce some control on the “direction” we allow the copying of formulas, so that we can establish a decreasing length argument. It is OK to copy a formula true at a nominal i to a nominal j if j generated i , but not if i generated j [4].

Essentially, we need to ensure that blocks of generated nominals contain strictly smaller formulas, so that any chain of them eventually terminates. It is the (\diamond) rule that *generates* a fresh nominal i by producing the formulas $\diamond i$ and $@_i \phi$. Only **GoTo** can decompose either formula into the raw nominal i . Our restriction **S4a** ensures $i \notin A$ so by **S4b**, nominal i cannot be copied to another block. Thus, unlike root nominals, the nominals generated by (\diamond) can only appear raw as opening nominals. Since **Nom** requires the opening nominal of the current block to appear on its own, formulas can only be copied *to* blocks with (\diamond)-generated opening nominals, not *from* them. This matches the quote. It also shows how generated nominals are treated differently, causing a “nominal asymmetry.”

We revisit termination in Section 6. For now, note that the *fixed* set A frees us from formalizing the *growing* set of nominals generated by (\diamond). The reader may imagine the set A to contain all root nominals, as it will in Section 5, such that these can be copied freely.

4 Properties

We briefly remark on some properties of ST^A that are useful for the completeness proof. We start by noting that while restriction **S3** allows us to start from any amount of potential, a single unit is always sufficient to close a branch. Then we lift the **S1** and **S2** restrictions by showing that unrestricted versions of the proof rules are admissible. This makes it simpler to show further properties of the system, since we do not have to worry about the restrictions any longer. Finally we show a structural property.

4.1 Sufficient Potential

That a single unit is sufficient is not surprising: simply never make a detour (i.e. two applications of **GoTo** in a row) and the other rule applications will build up the potential as needed. Similarly, given an existing tableau, construct a more “efficient” counterpart by collapsing sequences of **GoTo** so only the last one remains. **GoTo** serves no other purpose than starting a new block so any subsequent rule applications only depend on the final **GoTo**. The single starting unit may, however, be needed for an initial application of the rule.

► **Lemma 8** (A single unit of potential). *If branch Θ closes with respect to A and potential n then Θ closes with respect to A and potential 1.*

Proof. By induction on the closing tableau for Θ (see the formalization for details). ◀

4.2 Strengthening

► **Lemma 9** (Strengthening). *Let Θ be a branch and Δ a set of occurrences of ϕ on i -blocks in Θ . Assume that at least one “lasting occurrence” of ϕ at i is not in Δ . If Θ closes wrt. A and potential n then so does Θ with all occurrences in Δ removed.*

Proof. By induction on the construction of the closing tableau for Θ . When an occurrence in Δ is used as rule input, use the lasting occurrence of ϕ instead to construct the tableau for the strengthened branch. No rule applications are invalidated, so the new branch closes under the same amount of potential. Similarly, we only apply rules that were applicable before, so restriction S2 cannot be violated. See the formalization for exact details. ◀

In the formalization we represent the set of occurrences as a set of indices into the branch. We state the lemma over such a set to make it work with the induction principle given by Isabelle/HOL. To lift restriction S1, fix the set of occurrences to contain only the rule output, which must occur elsewhere since S1 is violated, and apply the lemma to justify it.

4.3 Substitution

Next we show a substitution lemma. Note that substitution across a tableau can collapse formulas such that an occurrence suddenly violates restriction S1 and cannot be justified as before the substitution. This is why Lemma 9 is useful. But it also means that our substitution lemma will quantify existentially over the potential needed to close the transformed branch: we may need to start from more potential to account for the fewer rule applications. Another complication is that restriction S2 may suddenly be violated by this collapsing but, as we have also shown previously [14], collapsing witnessing nominals allows us to lift S2.

► **Definition 10** ($\Theta\sigma$). *Given a substitution σ , i.e. a mapping from nominals to nominals, and a branch Θ , $\Theta\sigma$ denotes the branch obtained by replacing every nominal i in Θ by $\sigma(i)$.*

Substitutions are allowed to change the type of nominals, e.g. from numbers to strings, so in the following lemma we need to ensure that it leaves enough fresh nominals available.

► **Lemma 11** (Substitution). *Let Θ be a branch, A be a finite set of allowed nominals and σ a substitution whose co-domain is at least as large as its domain. If Θ closes with respect to A then $\Theta\sigma$ closes with respect to the image of A under σ .*

Proof. By induction on the construction of the closing tableau for an arbitrary σ .

In the (\diamond) case, let i be the generated witnessing nominal. After the (collapsing) substitution, the rule input may become witnessed by some nominal $\sigma(j)$, violating S2. In this case, utilize that we can pick σ in the induction hypothesis such that it maps i to $\sigma(j)$. By the side condition on (\diamond) , the image of A under the updated σ is the same, but now Lemma 9 justifies the rule output. The rest of the branch is unaffected since i is fresh.

If S2 is not violated, it may still be that $\sigma(i)$ is no longer fresh like i was before the substitution. Therefore, use the finiteness of both the branch and A , and the size of the co-domain of σ , to obtain a fresh nominal k . Apply the induction hypothesis at σ mapping i to k . This guarantees that the (\diamond) rule applies to justify the rule output. ◀

To lift S2, collapse the involved witnessing nominals in the same way as in the proof of Lemma 11 and apply Lemma 9. The finiteness assumption on A is stronger than we need, but we forgo generalization since we work with finite sets in Section 5 anyway.

4.4 Branch Structure

The following lemma shows that we can add, contract and rearrange blocks on a branch without affecting the existence of a closing tableau. Such operations may violate both S1 and S2, but we have lifted these restrictions already, so we do not need to worry about them.

► **Lemma 12** (Adding, contracting and rearranging blocks). *Let Θ be a branch consisting of the set of blocks $\{B_1, \dots, B_n\}$ and let Θ' be a branch whose blocks are a finite superset of $\{B_1, \dots, B_n\}$. If Θ closes wrt. finite A then so does Θ' .*

Proof. By induction on the construction of the closing tableau for arbitrary Θ' . In each case we apply the induction hypothesis at Θ' extended by B , where B is the current block of the original branch. This makes the opening nominals agree on the two branches, so that the original rule applies to the new branch as well. After applying this rule, we justify the B block by Lemma 9 and the GoTo rule. Lemma 11 resolves (\diamond) cases where the fresh nominal is not fresh on the new branch since we can substitute it with another fresh nominal. ◀

5 Completeness

Our completeness proof is a synthesis of two approaches, both based on showing completeness via contradiction by constructing a model for formulas on open, exhausted branches.

Bolander and Blackburn reason about the shape of such branches directly from the proof rules in their terminating, internalized calculus [9]. Jørgensen et al., on the other hand, define Hintikka sets of blocks as an abstraction of their open, exhausted branches and show model existence for formulas in such sets. They show that any set of blocks without a closing tableau can be extended to a maximal consistent set of blocks and that these are Hintikka sets [21]. Their model construction, however, assumes that all nominals are treated uniformly, which our termination restrictions prevent (cf. Section 3.3). We define Hintikka sets of blocks that characterize open branches *exhausted with respect to* a set of allowed nominals A . We then abstract the model existence result by Bolander and Blackburn, which is compatible with such branches, and apply it to our Hintikka sets. In Section 5.4 we contrast our approach with the existing work but the proof itself is self-contained.

5.1 Hintikka Sets

Figure 4 shows our definition of Hintikka sets of blocks. We reuse the “at” notation from Definition 1 and suppress “in H ” for brevity. Our goal is to show a model existence result for formulas on blocks in such sets. **ProP** and **NomP** ensure consistency at the bottom by forbidding certain contradictions. The remaining requirements match the proof rules. The ones up to **Nom** ensure *downwards saturation* such that the satisfiability of a complex formula is guaranteed by conditions on its subformulas [21]. The novel condition **Nom** ensures *lateral saturation* of allowed formulas across blocks whose opening nominals denote the same world. This allows us to treat such blocks uniformly when it comes to allowed formulas.

► **Remark 13.** **Nom** replaces three requirements by Jørgensen et al. [21, (iv, v, vii)] that serve the same purpose for a smaller range of formulas.

- ProP** If nominal b occurs at a and prop. symbol p occurs at b then $\neg p$ does not occur at a .
NomP If nominal i occurs at a then $\neg i$ does not occur at a .
NegN If $\neg\phi$ occurs at a then ϕ occurs at a .
DisP If $\phi \vee \psi$ occurs at a then either ϕ or ψ occurs at a .
DisN If $\neg(\phi \vee \psi)$ occurs at a then both $\neg\phi$ and $\neg\psi$ occur at a .
DiaP If $\Diamond\phi$ occurs at a and ϕ is not a nominal then for some i , $\Diamond i$ and $@_i\phi$ occur at a .
DiaN If $\neg\Diamond\phi$ and $\Diamond i$ both occur at a then $\neg@_i\phi$ occurs at a .
SatP If $@_a\phi$ occurs at b then ϕ occurs at a .
SatN If $\neg@_a\phi$ occurs at b then $\neg\phi$ occurs at a .
GoTo If ϕ occurs at a and i is a nominal in ϕ then some block in H has opening nominal i .
Nom If ϕ and nominal a both occur at b and ϕ is *allowed* by A then ϕ occurs at a .

■ **Figure 4** Eleven requirements for a set of blocks H to be a Hintikka set with respect to A .

5.1.1 Equivalence

Assume for the rest of the section that H is a Hintikka set with respect to the set of allowed nominals A . We define an equivalence between nominals:

► **Definition 14** (Equivalence). *Nominals i, j are equivalent, $i \sim_H j$, if j occurs at i in H .*

► **Note 15** (\sim and ϕ at i). In the following we typically suppress the subscript in \sim_H and likewise the fragment “in H ” in sentences like “ ϕ occurs at i in H ”.

The equivalence $i \sim j$ only implies $j \sim i$ if $i \in A$ as otherwise **Nom** does not apply: only allowed nominals are symmetric. This motivates the restriction on the following lemma:

► **Lemma 16** (Equivalence relation). *\sim_H is an equivalence relation on the set of allowed opening nominals in H .*

Proof. *Reflexivity:* $i \sim_H i$ for any opening nominal i in H since opening nominals occur on their own block. *Symmetry:* Assume $i \sim_H j$ with $i \in A$. That is, j occurs at i in H so by **Nom**, i occurs at j in H : $j \sim_H i$. *Transitivity:* Assume $i \sim_H j$ and $j \sim_H k$ with $i, k \in A$. By symmetry, i occurs at j in H : $j \sim_H i$. Moreover, $k \in A$ occurs at j in H so by **Nom**, k occurs at i in H : $i \sim_H k$. ◀

► **Note 17.** Due to the *GoTo* Hintikka restriction, any nominal occurring in H also occurs as opening nominal, so \sim_H is an equivalence relation on the allowed nominals in H .

5.1.2 Model Construction

Let $|i|_{\sim_H}$ denote the set of nominals equivalent to i with respect to H .

We make use of the following shorthand in our model construction:

► **Definition 18** (ϕ at a^*). *We say that ϕ occurs at a set of nominals $a^* = \{a_0, a_1, \dots\}$ if it occurs at some nominal $a_k \in a^*$ and that ϕ occurs at all a^* if it occurs at all nominals in a^* .*

We can now define the model induced by Hintikka set H and allowed nominals A :

► **Definition 19** (The model $\mathfrak{M}_{H,A}$ and assignment $g_{H,A}$ induced by H and A).

Worlds The worlds of $\mathfrak{M}_{H,A}$ are sets of equivalent nominals, written a^* , from H .

Assignment The assignment $g_{H,A}$ maps a nominal to the equivalence class of an equivalent, allowed nominal or a singleton set if no such nominal exists:

$$g_{H,A}(a) = \begin{cases} |b|_{\sim_H} & \exists b \in A. a \sim_H b \\ \{a\} & \text{otherwise} \end{cases}$$

Reachability From world a^* we can reach a world exactly if it is denoted by some nominal b that is reachable at a^* (as witnessed by $\Diamond b$ occurring at a^*):

$$R_{H,A}(a^*) = \{g_{H,A}(b) \mid \exists a \in a^*. \Diamond b \text{ occurs at } a \text{ in } H\}$$

Valuation Propositional symbol p holds at world a^* exactly if p occurs at a^* in H :

$$V_{H,A}(a^*)(p) = \exists a \in a^*. p \text{ occurs at } a \text{ in } H$$

5.1.3 Properties of the Model

Consider first a property of the assignment:

► **Lemma 20** (Non-empty assignment). *The induced assignment $g_{H,A}$ is always non-empty.*

Proof. Fix an arbitrary nominal a . If $g_{H,A}(a) = \{a\}$ the thesis holds immediately. So assume there is some $b \in A$ such that $a \sim_H b$ and $g_{H,A}(a) = |b|$. $b \in |b|$ witnesses the thesis. ◀

The following lemma showcases the lateral saturation guaranteed by the **Nom** condition:

► **Lemma 21** (Assignment closure). *If ϕ is allowed by A and ϕ occurs at a in H then ϕ occurs at all $g_{H,A}(a)$ in H (and at least one such world exists).*

Proof. If $g_{H,A}(a) = \{a\}$ the thesis holds immediately. So assume there is some $b \in A$ where b occurs at a in H and $g_{H,A}(a) = |b|$. Then by Hintikka requirement **Nom**, ϕ occurs not only at b in H but at all $a \in |b|$ in H , proving the thesis. Lemma 20 gives the parenthetical. ◀

5.1.4 Model Existence

We can now prove model existence:

► **Lemma 22** (Model existence). *Let H be a Hintikka set with respect to allowed nominals A . We show two statements by mutual induction:*

- *If ϕ occurs at i in H and ϕ is allowed by A then $\mathfrak{M}_{H,A}, g_{H,A}, g_{H,A}(i) \models \phi$.*
- *If $\neg\phi$ occurs at i in H and ϕ is allowed by A then $\mathfrak{M}_{H,A}, g_{H,A}, g_{H,A}(i) \not\models \phi$.*

Proof. By induction on the structure of ϕ for an arbitrary nominal i . The proof follows the one by Bolander and Blackburn [9]. We suppress subscripts for readability.

If p at i then p at $g(i)$ by Lemma 21, which matches the valuation, so $\mathfrak{M}, g, g(i) \models p$.

If $\neg p$ at i then $\neg p$ at all $g(i)$ so by **ProP**, p does not occur at $g(i)$, so $\mathfrak{M}, g, g(i) \not\models p$.

If a at i then from the assumption $a \in A$ we have $g(i) = |a|$ and $g(a) = |a|$ and thereby $g(i) = g(a)$ so $\mathfrak{M}, g, g(i) \models a$.

If $\neg a$ at i then $\neg a$ at $g(i)$ by Lemma 21. Moreover, $a \in A$ by assumption so from Lemma 21 we have that a occurs at all $g(a)$. We thus have $\neg a$ at $g(i)$ but a at all $g(a)$ so by **NomN**, $g(i) \neq g(a)$ and therefore $\mathfrak{M}, g, g(i) \not\models a$.

If $\neg\phi$ at i then $\mathfrak{M}, g, g(i) \not\models \phi$ by the induction hypothesis so $\mathfrak{M}, g, g(i) \models \neg\phi$.

If $\neg\neg\phi$ at i then ϕ at i by **NegN** and $\mathfrak{M}, g, g(i) \models \neg\phi$ by the induction hypothesis.

The cases for $\phi \vee \psi$, $\neg(\phi \vee \psi)$, $@_j\phi$ and $\neg@_j\phi$ at i all follow similarly to $\neg\phi$ and $\neg\neg\phi$.

If $\Diamond j$ at i then $j \in A$ by assumption. Thus $\Diamond j$ at $g(i)$ so $g(i) R g(j)$ and $\mathfrak{M}, g, g(i) \models \Diamond j$.

If $\Diamond\phi$ at i where ϕ is not a nominal then by **DiaP** (and Lemma 21) there is some witnessing nominal k such that $\Diamond k$ and $@_k\phi$ both appear at $g(i)$. By **SatP**, ϕ then occurs at k and by the induction hypothesis at k we have $\mathfrak{M}, g, g(k) \models \phi$. From $\Diamond k$ at $g(i)$ we have $g(i) R g(k)$ so combined we get $\mathfrak{M}, g, g(i) \models \Diamond\phi$.

If $\neg\Diamond\phi$ at i then $\neg\Diamond\phi$ at $g(i)$ by Lemma 21. We need to show that all worlds reachable from $g(i)$ falsify ϕ . So assume for some arbitrary j that $\Diamond j$ occurs at some $a \in g(i)$. By **Nom**, we also have $\neg\Diamond\phi$ at a so by **DiaN** we get $\neg@_j\phi$ at a and finally by **SatN** we have $\neg\phi$ at j . The induction hypothesis at j then tells us that $\mathfrak{M}, g, g(j) \not\models \phi$ as needed. Since j was chosen arbitrarily, $\mathfrak{M}, g, g(i) \not\models \Diamond\phi$.

Each appeal to the induction hypothesis requires showing that the subformula is allowed by A but since it is a subformula this holds trivially. \blacktriangleleft

5.2 Maximal Consistent Sets

Our next task is to follow the classical synthetic recipe: extend a consistent set of blocks to be maximally consistent, show that such sets fulfill all Hintikka requirements and thus that formulas in them are satisfiable. Consistency and maximality are standard but wrt. A :

► **Definition 23** (Consistency). *The set of blocks S is consistent wrt. A if there is no finite subset $S' \subseteq S$ such that S' has a closing tableau wrt. A and any amount of potential.*

► **Definition 24** (Maximality). *The set of blocks S is maximal wrt. A if it is consistent wrt. A and for any block $B \notin S$ the set $S \cup \{B\}$ is inconsistent wrt. A .*

Besides maximally consistent, our constructed set will also be \Diamond -saturated [21]:

► **Definition 25** (\Diamond -Saturation). *The set of blocks S is \Diamond -saturated if for any ϕ at any a in S , where ϕ is not a nominal, there is a nominal i such that $@_i\phi$ and $\Diamond i$ both occur at a in S .*

We now construct our \Diamond -saturated maximally consistent set and show it is a Hintikka set:

► **Definition 26** (Lindenbaum-Henkin construction). *Assume an enumeration of all blocks B_0, B_1, B_2, \dots in the language. From a consistent set S_0 we build an infinite sequence of consistent sets S_0, S_1, S_2, \dots in the following way. Given S_n , construct S_{n+1} like so:*

$$S_{n+1} = \begin{cases} S_n & \text{if } S_n \cup \{B_n\} \text{ is inconsistent wrt. } A \\ S_n \cup \{B_n\} \cup \{B'\} & \text{otherwise, where } B' \text{ is a } \Diamond\text{-witness for } B_n \end{cases}$$

A \Diamond -witness for a block B is a block with the same opening nominal that witnesses all $\Diamond\phi$ -formulas in B using fresh and disallowed nominals (when ϕ is not a nominal).

► **Lemma 27** (Lindenbaum-Henkin). *Let S_0 be a consistent set of blocks with respect to finite A and over a finite set of nominals. Then $\bigcup S_n$ as given by Definition 26 is a \Diamond -saturated maximally consistent set.*

Proof. The three-part proof follows the one by Jørgensen et al. [21].

Consistency. Proof by contradiction. Assume $\bigcup S_n$ is inconsistent. Then some finite subset $S' \subseteq \bigcup S_n$ has a closing tableau. But the sequence S_0, S_1, S_2, \dots grows with respect to \subseteq so there must be an m such that $S' \subseteq S_m$. And since S_0 is consistent, it follows by induction on m that S_m is too (each \Diamond -witness preserves consistency due to the (\Diamond) rule). This contradicts the existence of an inconsistent, finite subset S' .

Maximality. Proof by contradiction. Assume that there is some block $B_m \notin \bigcup S_n$ such that $\bigcup S_n \cup \{B_m\}$ is still consistent. This block is part of the enumeration of blocks, but was not added to S_{m+1} . This can only be because $S_m \cup \{B_m\}$ is inconsistent. However, $S_m \cup \{B_m\} \subseteq \bigcup S_n \cup \{B_m\}$ contradicting the consistency of the right-hand side.

\Diamond -**Saturation.** Follows directly from the addition of \Diamond -witnesses. \blacktriangleleft

► **Lemma 28** (Smullyan-Fitting block lemma). *Assume S is a \Diamond -saturated maximal consistent set of blocks wrt. a finite set A and a finite set of nominals. Then S is a Hintikka set.*

Proof. The proof follows the one by Jørgensen et al. [21] but we have fewer cases since we have fewer Hintikka requirements. The cases are straight-forward so we only exemplify three, with the last being the typical one. The remaining cases can be found in the formalization.

Case ProP. Proof of negation. Assume that b occurs at a , p occurs at b and $\neg p$ occurs at a in S for some a, b, p . The set S is assumed to be consistent but we can construct a closing tableau from these blocks by applying the **Nom** rule to get $\neg p$ at b and immediately close due to the existing p at b .

Case DiaP. Follows directly from \Diamond -saturation.

Case Nom. Assume that both ϕ and a occur at b in S and that ϕ is allowed by A . Assume towards a contradiction that ϕ does not occur at a in S . Then by the maximality of S , we can find an inconsistent finite subset $S' \cup \{([\phi], a)\} \subseteq S \cup \{([\phi], a)\}$ where $([\phi], a)$ is an a -block that only contains ϕ . If a closing tableau exists for $S' \cup \{([\phi], a)\}$ then it also exists for the larger set $S' \cup \{([\phi], a)\} \cup \{([\phi], a), b)\}$ (Lemma 12). But now the **Nom** rule tells us that ϕ at a is redundant, so just $S' \cup \{([\phi], a)\} \cup \{([\phi], a), b)\}$ is inconsistent. The **GoTo** rule gets us to $S' \cup \{([\phi], a), b)\}$ and this set is trivially a subset of S , contradicting its consistency. \blacktriangleleft

5.3 Tying It All Together

Completeness follows by constructing a model for any formula whose tableau does not close.

► **Theorem 29** (Completeness). *Assume that ϕ is a valid formula and a is some nominal. Let A be the set containing all nominals in ϕ . Then the branch consisting solely of $\neg\phi$ on an a -block has a closing tableau with respect to A and 1 unit of potential.*

Proof. Assume towards a contradiction that the branch does not close. Then the set $S_0 = \{([\neg\phi], a)\}$ is consistent with respect to A . We construct $\bigcup S_n$, which by Lemma 27 is a \Diamond -saturated maximal consistent set of blocks, so by Lemma 28 $\bigcup S_n$ is a Hintikka set.

Since $\neg\phi$ occurs at a in $\bigcup S_n$, we obtain from Lemma 22 a model that does not satisfy ϕ , namely $\mathfrak{M}_{H,A}, g_{H,A}, g_{H,A}(a) \not\models \phi$. This contradicts our validity assumption, so the branch must close. By Lemma 8 it must close from a single unit of potential. \blacktriangleleft

5.4 Relation to Existing Work

In this section we provide context for our induced model, Definition 19, and the corresponding Lemma 22. Readers less familiar with tableau systems for hybrid logic may skip this section. To refresh, Bolander and Blackburn give an analytic proof for a terminating, internalized calculus [9] and Jørgensen et al. give a synthetic proof for the non-terminating system ST [21].

5.4.1 Worlds

Jørgensen et al. have no restrictions on their **Nom** rule so they have no nominal asymmetry (cf. Section 3.3) and \sim_H is an equivalence relation on all nominals. They use representatives of such equivalence classes as their worlds [21]. Since \sim_H is only an equivalence relation on a subset of our nominals, we cannot use equivalence classes directly. Instead we use sets of equivalent nominals. Bolander and Blackburn use plain nominals as their worlds.

5.4.2 Assignment

Jørgensen et al. map each nominal i in H to its equivalence class $|i|_{\sim_H}$ [21]. If we artificially fix A to contain all nominals in H then \sim_H becomes an equivalence relation on all nominals. Our assignment then reduces to its first clause and becomes equivalent to theirs.

Bolander and Blackburn map each nominal a to its “urfather” $u(a)$: either an equivalent “right nominal” or the nominal itself if no such nominal exists [9]. This is very similar to our assignment that maps each nominal to the equivalence class of an equivalent *allowed nominal* or the singleton set if no such nominal exists.

A *right nominal*, understood in terms of our setting, is a non-opening nominal that occurs on its own. Since there may be multiple equivalent right nominals, Bolander and Blackburn impose an ordering on them and always choose the smallest one to ensure that their assignment is well-defined [9]. Working with sets of nominals frees us from such concerns.

5.4.3 Reachability and the Bridge Rule

It is worthwhile to compare the three different reachability relations from the considered systems. By writing them in similar notation we get:

Jørgensen et al.	$ i R_H j $	iff $\Diamond j$ occurs at i in H
Bolander and Blackburn	$i R_H u(j)$	iff $\Diamond j$ occurs at i in H
The present paper	$i^* R_H g_{H,A}(j)$	iff $\Diamond j$ occurs at i^* in H

If we further note that $g(j) = |j|$ for Jørgensen et al. [21] and $g(j) = u(j)$ for Bolander and Blackburn [9] we see that the relations are all defined in the same way over the assignment: *a world is reachable iff it is denoted by a nominal j such that $\Diamond j$ occurs at the current world.* Only the treatment of the worlds differ. Since Jørgensen et al. use representatives of their sets they need the following Hintikka requirement to ensure well-definedness:

If there is an i -block in H with $\Diamond j$ on it, and a j -block in H with k on it, then there is an i -block in H with $\Diamond k$ on it [21, (vi)].

To see why, imagine that the premises hold but the conclusion does not. Then $|i| R_H |j|$ and $j \sim_H k$ but not $|i| R_H |k|$ even though $|j| = |k|$ by the second premise, so the choice of representative matters when it should not. In our setting we side-step the problem completely by having no representatives but quantifying existentially over the nominals in our worlds.

If we view the requirement as a rule, we get the known **Bridge** rule that produces $\Diamond k$ at i given $\Diamond j$ at i and nominal k at j . Jørgensen et al. prove the admissibility of **Bridge** as part of their completeness proof [21]. We include this result in the formalization (when $j \in A$) because it is interesting in its own right [4] but do not need it for completeness.

5.4.4 Valuation

Our valuation is standard but our use of sets instead of representatives slightly complicates the **ProP** Hintikka requirement, where we take equivalence of nominals into account. For Jørgensen et al. the following suffices: “if there is an i -block in H with atomic formula a on it then there is no i -block in H with $\neg a$ on it.” [21].

5.4.5 Model Existence

We turn now to the model existence result, Lemma 22, inspired by Blackburn and Bolander [9].

The two nominal cases and the $\diamond j$ case rely on the involved nominals being in A . Bolander and Blackburn work with right nominals instead of allowed nominals [9]. This gives them the positive nominal case for free, since the formula in that case is a right nominal. In the negative nominal case, however, they need to rely on a special (\neg) rule that upgrades a negated nominal, “ $@_i \neg a$ ”, to a right nominal “ $@_a a$ ”. They need this rule because of the nature of internalized tableau systems: the nominal i in a satisfaction statement $@_i a$ has lower status than the right nominal a . The status of nominals in our system is not defined structurally but by the set A . Thus, we make the (\neg) rule unnecessary by picking A carefully.

Finally, Bolander and Blackburn assume that the formula in question is not a $\diamond j$ formula produced by the (\diamond) rule. Our assumption $j \in A$ matches this, since the (\diamond) rule cannot generate an allowed nominal, but we are free from keeping track of actual rule applications.

6 Relation to ST^*

Here, we relate our restrictions S1-S4 to the restrictions R1-R5 and **Nom*** rule in ST^* [4].

6.1 System ST^*

For reasons of space we introduce ST^* only briefly. To obtain ST^* , take the rules in Figure 2, add another rule called **Name** that introduces a fresh nominal to the branch and impose restrictions R1-R5 and **Nom*** that we explain in the following. Since the rules of ST^A are a subset of ST^* , it is meaningful to compare the strength of our restrictions to those of ST^* .

Blackburn et al. [4] need the **Name** rule since they allow the very first block to have no opening nominal. We have dispensed with this flexibility to obtain a simpler formalization.

6.2 Restrictions R1-R5

Restriction R1 states that “a formula is never added to an i -block if it already occurs in an i -block on the same branch” [4]. This formulation is more ambiguous than our S1, which states when a rule is applicable. Any rule application outlawed by R1 is also outlawed by S1:

► **Lemma 30** (R1 implies S1). *If R1 outlaws a rule application then so does S1.*

Proof. R1 outlaws the rule application so it must include no formulas new to the block type. Therefore, S1 outlaws it too. ◀

Restriction R2 states that “the (\diamond) rule can not be applied twice to the same formula occurrence” [4]. Note that formalizing this would require keeping track of (\diamond) rule applications. This is why S2 is formulated in terms of branch content instead. It is at least as strict as R2:

► **Lemma 31** (R2 implies S2). *If R2 outlaws an application of (\diamond) then so does S2.*

Proof. Assume that an application of the rule (\Diamond) to formula $\Diamond\phi$ at a is outlawed by R2. This means that (\Diamond) has already been applied to $\Diamond\phi$ at a . So for some nominal i there must be formulas $@_i\phi$ and $\Diamond i$ witnessing $\Diamond\phi$ at a . Thus the application is also outlawed by S2. ◀

Restriction R3 applies to the omitted **name** rule so we have no equivalent of it [4].

Restriction R4 states that “the **GoTo** rule can not be applied twice in a row” [4]. Our counterpart is S3 that does allow repeated applications but still prevents repeating the rule ad infinitum (cf. Figure 3b). We see in Section 6.5 why this extra flexibility is desirable. For now recall the idea from Section 4.1 that any tableau with repeated applications of **GoTo** can be translated into one where just the final application remains. We have the following:

► **Lemma 32** (From S3 to R4). *A tableau satisfying S3 collapses into one that satisfies R4 where only finite sequences of **GoTo** are removed and all non-**GoTo** applications are preserved.*

Proof. By collapsing all sequences of **GoTo** applications into the last one (cf. Lemma 8). All such sequences are finite due to decreasing potential so “the last one” is well-defined. ◀

Finally, restriction R5 can be ignored here: it restricts the more liberal variants of rules $(@)$ and $(\neg@)$ in system ST to the versions present in ST^* and ST^A [4].

6.3 Nom^* and Allowed Nominals

We turn now to the Nom^* rule in ST^* and its relationship to our set of allowed nominals A in restriction S4. We first need the following by Blackburn et al. [4]: “A quasi-root subformula is a formula of the form ϕ , $\neg\phi$, $@_i\phi$ or $\neg@_i\phi$ where ϕ is a subformula of the root.”

Their Nom^* rule is then defined as follows:

Suppose i and j are nominals, ϕ is a quasi-root subformula and $j \neq i, \phi$. If j and ϕ both occur in i -blocks on a branch Θ , then ϕ can be added to any j -block on Θ [4].

By inspecting the rules of ST^* and ST^A we see that only the (\Diamond) rule can produce formulas that are not quasi-root subformulas [4]. As such, the only formulas that Nom^* does not allow us to copy are formulas i and $\Diamond i$ where i was introduced by (\Diamond) . This is exactly what restriction S4 enforces on our **Nom** rule (cf. Section 3.3). So S4 is at least as strict:

► **Lemma 33** (Nom implies Nom^*). *Suppose that ϕ and a both occur at b in a tableau constructed under the allowed set of nominals A . If **Nom** can add ϕ to a then so can Nom^* .*

Proof. If ϕ can be added by **Nom** it must be allowed by A . Thus ϕ must be a quasi-root subformula. Moreover, since adding ϕ to a does not violate S1 (or R1), $a \neq \phi$ and likewise $a \neq b$. Ultimately, Nom^* can also add ϕ to a . ◀

6.4 Termination

We have covered all differences between ST^* and ST^A and seen how the restrictions compare. This motivates the following unformalized theorem and proof sketch:

► **Theorem 34** (ST^A is terminating). *Any ST^A tableau is finite.*

Proof. Lemmas 30–33 imply that we can translate any ST^A tableau into an ST^* tableau of similar size by collapsing repeated applications of **GoTo** (and adding an initial application of the **Name** rule). Since all ST^* tableaux are finite [4] so must any ST^A tableau be. ◀

Blackburn et al. [4] exemplify a number of infinite branches possible in system ST and show that they are illegal in system ST^* . In support of the above theorem, we note that the sequences of rule applications leading to those infinite branches are also outlawed in ST^A .

a		a	
ϕ		ϕ	
$\hline a'$	GoTo	$\hline a$	GoTo
ϕ'	R	ϕ	R
$\hline i$	GoTo	$\hline \sigma(i)$	GoTo
ψ		$\psi\sigma$	

(a) Possible segment on original closing tableau. (b) R becomes invalid causing two GoTos in a row.

■ **Figure 5** Unjustified GoTo after applying substitution σ that unifies a and a' as well as ϕ and ϕ' .

6.5 Restricting the GoTo Rule

We should motivate our choice of S3 over R4. As Section 4 shows, we typically show lemmas of the form “if branch Θ has a closing tableau then so does $f(\Theta)$ ”, where f is some operation like substitution or restructuring. In a proof by induction on the closing tableau under restriction R4 we need to show in each non-GoTo case that GoTo becomes applicable, since we need that assumption to discharge the GoTo case. However, the transformation may invalidate a previously valid rule application and prevent us from making this promise. Figure 5 depicts a possible case when proving the substitution lemma. Before the substitution, the application of rule R was legal, but afterwards it violates restriction R1. We can still justify the extension ϕ with the Strengthening Lemma 9 but doing so does not make GoTo applicable afterwards.

We might give a more intricate transformation that also prunes detours but that would complicate an otherwise simple idea like substitution. We could also state the lemma in weaker terms that allow for a different branch structure, but we prefer to give straight-forward lemmas and transformations. Our S3 restriction resolves the issue by dealing with detours separately. Consider Figure 5 from the perspective of potential: we need to start from more potential to close the transformed branch since we lose a rule application, but we can simply do this, so the detour becomes benign. Thus, we can give the transformation we want, we just need to existentially quantify the potential required to close the resulting branch.

7 Formalization

In general, the formalization consists of close to 5000 lines in the *intelligible semi-automated reasoning* language Isar [27] and follows the structure of the paper. It is accepted into the Archive of Formal Proofs and thus kept up to date with new versions of Isabelle/HOL.

We formalize the logic as a deep embedding into higher-order logic by specifying the syntax as a datatype and the semantics as a predicate on that datatype (alongside a model and an assignment). Types in higher-order logic are non-empty so we represent the set of worlds as a type variable $'w$. Similarly, we use $'a$ for the universe of propositional symbols and $'b$ for the universe of nominals. We formalize a block as a list of formulas paired with its opening nominal and a branch as a list of blocks, where lists in Isabelle/HOL are finite, ordered sequences. We use the **inductive** command to specify the proof system as ten inductive cases. The command provides a predicate \vdash for whether or not a given branch closes with respect to a set A and potential n . Thus, we abstract away the concrete shape of a closing tableau and reason only about its existence. This suffices for formalizing completeness but not termination where we would need to inspect well-formed but infinite branches. However, it permits induction over the proof rules instead of the trickier coinduction.

Imagine that we formalized ST^* instead of ST^A . Section 6.5 motivated our choice of S3 over R4. Restriction R2 on the (\diamond) rule would require us to additionally index our predicate \vdash by a list of indices, each pointing to a formula occurrence that (\diamond) cannot be applied to. When

proving lemmas by induction, we would need to make suitable assumptions about this list. Instead, our formulation S2 identifies the applicability of (\Diamond) from the branch content itself, which we already know. The Nom^* rule considers quasi-root subformulas and would require us to remember the root segment of the tableau as we extend it, complicating induction proofs too. Our parameterization of the rules by the set A causes no such complications.

Imagine next that we adapted the completeness proof for ST^* to ST^A . That proof works by translation from a different system with an analytic completeness proof, which we would have to formalize as well. This could be done: Blanchette, Popescu and Traytel [7, 8] have formalized analytic completeness proofs for first-order logic in Isabelle/HOL. Instead, our *standalone* synthetic completeness proof joins a family of such proofs in Isabelle/HOL [2, 15, 16]. While possible, a similar proof for ST^* would, as described, be harder to formalize.

8 Conclusion and Future Work

We have presented a Seligman-style tableau system for hybrid logic with a formalization in Isabelle/HOL of its soundness and completeness and argued that it is terminating. The restrictions required for termination cause an asymmetry in branches that makes a previous synthetic completeness proof for hybrid logic tableau systems inapplicable. We have presented a novel proof that works in this case and described its relation to existing work. The use of plain sets instead of representatives in the model construction relieves us of some concerns about well-definedness. Our work is the first sound and complete formalized proof system for hybrid logic and the first synthetic proof for a terminating hybrid logic tableau system.

Blackburn et al. showed termination of ST^* by a translation of any branch into a terminating system and we claim termination of ST^A by possible translation into ST^* . We are currently working on a direct, formalized termination proof for ST^A through a decreasing measure argument in the style of Bolander and Blackburn [9]. This will allow code generation for a verified decision procedure based on the tableau system. We also want to explore extensions to the logic and investigate a Seligman-style system for intuitionistic hybrid logic.

References

- 1 Carlos Areces, Patrick Blackburn, and Maarten Marx. Hybrid logics: Characterization, interpolation and complexity. *The Journal of Symbolic Logic*, 66(3):977–1010, 2001.
- 2 Stefan Berghofer. First-Order Logic According to Fitting. *Archive of Formal Proofs*, 2007. Formal proof development. URL: <http://isa-afp.org/entries/FOL-Fitting.html>.
- 3 Patrick Blackburn. Representation, reasoning, and relational structures: A hybrid logic manifesto. *Logic Journal of the IGPL*, 8(3):339–365, 2000.
- 4 Patrick Blackburn, Thomas Bolander, Torben Braüner, and Klaus Froyen Jørgensen. Completeness and termination for a Seligman-style tableau system. *Journal of Logic and Computation*, 27(1):81–107, 2017.
- 5 Patrick Blackburn and Miroslava Tzakova. Hybridizing concept languages. *Annals of Mathematics and Artificial Intelligence*, 24(1-4):23–49, 1998.
- 6 Jasmin Christian Blanchette. Formalizing the metatheory of logical calculi and automatic provers in isabelle/hol (invited talk). In Assia Mahboubi and Magnus O. Myreen, editors, *Certified Programs and Proofs, CPP. Proceedings*, pages 1–13. ACM, 2019.
- 7 Jasmin Christian Blanchette, Andrei Popescu, and Dmitriy Traytel. Abstract completeness. *Archive of Formal Proofs*, 2014. Formal proof development. URL: https://isa-afp.org/entries/Abstract_Completeness.html.
- 8 Jasmin Christian Blanchette, Andrei Popescu, and Dmitriy Traytel. Soundness and completeness proofs by coinductive methods. *Journal of Automated Reasoning*, 58(1):149–179, 2017.

- 9 Thomas Bolander and Patrick Blackburn. Termination for Hybrid Tableaus. *Journal of Logic and Computation*, 17(3):517–554, 2007.
- 10 Thomas Bolander and Torben Braüner. Tableau-based decision procedures for hybrid logic. *Journal of Logic and Computation*, 16(6):737–763, 2006.
- 11 Torben Braüner and Valeria de Paiva. Intuitionistic hybrid logic. *Journal of Applied Logic*, 4(3):231–255, 2006.
- 12 Christian Doczkal and Gert Smolka. Constructive formalization of hybrid logic with eventualities. In Jean-Pierre Jouannaud and Zhong Shao, editors, *Certified Programs and Proofs, CPP. Proceedings*, volume 7086 of *Lecture Notes in Computer Science*, pages 5–20. Springer, 2011.
- 13 Melvin Fitting. *Proof Methods for Modal and Intuitionistic Logics*, volume 169. Springer Science & Business Media, 1983.
- 14 Asta Halkjær From, Patrick Blackburn, and Jørgen Villadsen. Formalizing a Seligman-style tableau system for hybrid logic. In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Automated Reasoning*, pages 474–481, Cham, 2020. Springer International Publishing.
- 15 Asta Halkjær From. Epistemic Logic. *Archive of Formal Proofs*, 2018. Formal proof development. URL: http://isa-afp.org/entries/Epistemic_Logic.html.
- 16 Asta Halkjær From. A sequent calculus for first-order logic. *Archive of Formal Proofs*, 2019. Formal proof development. URL: http://isa-afp.org/entries/FOL_Seq_Calc1.html.
- 17 Asta Halkjær From. Hybrid Logic. Master's thesis, Technical University of Denmark, 2020.
- 18 Asta Halkjær From. Hybrid logic in the Isabelle proof assistant: Benefits, challenges and the road ahead. In Nicola Olivetti and Rineke Verbrugge, editors, *Short Papers: Advances in Modal Logic (AiML) 2020*, pages 23–27, 2020.
- 19 Didier Galmiche and Yakoub Salhi. Sequent calculi and decidability for intuitionistic hybrid logic. *Information and Computation*, 209(12):1447–1463, 2011.
- 20 Limin Jia and David Walker. Modal proofs as distributed programs (extended abstract). In David A. Schmidt, editor, *Programming Languages and Systems, ESOP, Proceedings*, volume 2986 of *Lecture Notes in Computer Science*, pages 219–233. Springer, 2004.
- 21 Klaus Froyen Jørgensen, Patrick Blackburn, Thomas Bolander, and Torben Braüner. Synthetic completeness proofs for Seligman-style tableau systems. In *Advances in Modal Logic, Volume 11*, pages 302–321, 2016.
- 22 Sven Linker. Hybrid Multi-Lane Spatial Logic. *Archive of Formal Proofs*, 2017. Formal proof. URL: http://isa-afp.org/entries/Hybrid_Multi_Lane_Spatial_Logic.html.
- 23 Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.
- 24 Jeremy Seligman, Fenrong Liu, and Patrick Girard. Facebook and the epistemic logic of friendship. In Burkhard C. Schipper, editor, *Proceedings of the 14th Conference on Theoretical Aspects of Rationality and Knowledge (TARK 2013)*, pages 229–238, Chennai, India, 2013.
- 25 Jerry Seligman. Internalization: The case of hybrid logics. *Journal of Logic and Computation*, 11(5):671–689, 2001.
- 26 Raymond M Smullyan. *First-Order Logic*. Dover Publications, 1995.
- 27 Makarius Wenzel. Isabelle/Isar – a generic framework for human-readable proof documents. *From Insight to Proof – Festschrift in Honour of Andrzej Trybulec, Studies in Logic, Grammar and Rhetoric*, 10(23):277–298, 2007.

CHAPTER 6

Formalized Soundness and Completeness of Epistemic and Public Announcement Logic

This chapter extends my WoLLIC 2021 paper [5] and has been accepted for a special issue of the Journal of Logic and Computation.

Preprint

Asta Halkjær From. “Formalized Soundness and Completeness of Epistemic and Public Announcement Logic”. In: *Special issue: Selected papers from WoLLIC 2021, the 27th Workshop on Logic, Language, Information, and Computation*. Ed. by Alexandra Silva, Renata Wassermann, and Ruy J. G. B. de Queiroz. Journal of Logic and Computation. Forthcoming.

Formalized Soundness and Completeness of Epistemic and Public Announcement Logic

Asta Halkjær From^[0000–0002–3601–0804]

Technical University of Denmark, Kongens Lyngby, Denmark
ahfrom@dtu.dk

Abstract. I strengthen the foundations of epistemic logic by formalizing the family of normal modal logics in the proof assistant Isabelle/HOL. I define an abstract canonical model over any set of axioms and formalize completeness-via-canonicity: when the canonical model for the chosen axioms belong to a certain class of frames, strong completeness over that class follows immediately. I instantiate the result with logics based on various epistemic principles to obtain completeness results for systems from **K** to **S5**. I then move to a family of public announcement logics (PAL) and prove abstract results for strong soundness and completeness. I lift the completeness results from epistemic logic to the setting with public announcements in a modular way. This work formulates the completeness-via-canonicity technique as a proper theorem and demonstrates its applicability. Additionally, it succinctly formalizes the requirements for lifting completeness from bare epistemic logic to the addition of public announcements.

Keywords: Epistemic Logic · Public Announcement Logic · Soundness · Completeness · Isabelle/HOL · Canonicity.

1 Introduction

Epistemic logic (EL) provides a foundation for reasoning about the knowledge of agents, both factual (“I know the sky is blue”) and higher-order (“I know that you know that I know the sky is blue”). This has applications in computer science and artificial intelligence [25]. Basic epistemic logic extends propositional logic with a knowledge modality K_i that, for each agent i , expresses that agent’s knowledge. For example, the following formula states that: (i) agent 1 knows p , (ii) agent 2 knows that agent 1 knows p , but (iii) agent 1 does not know (ii):

$$K_1p \wedge K_2K_1p \wedge \neg K_1K_2K_1p$$

Such formulas are understood on possible-worlds models that represent the different situations that agents consider possible, including which situations are indistinguishable to them (say, due to a lack of observations). Different things can be true at each possible world and the uncertainty of each agent is modeled by relating the worlds through so-called accessibility relations, one for each agent. The agent finds itself at some world and it considers those worlds possible that

are accessible from this world. Thus, an agent *knows* something if it holds at every world that the agent considers possible, i.e. all accessible worlds.

The addition of *public announcements* increases the expressiveness. Agents trust these announcements and thus rule out worlds that are incompatible with them. This allows us to model dynamic problems, like the muddy children puzzle [2], where a series of public announcements increases the knowledge of the agents. For example, the following formula states that after the announcement that agent 1 knows p , agent 2 knows that agent 1 knows p :

$$[K_1p]_1 (K_2K_1p)$$

To model different kinds of knowledge, we can consider classes of possible-worlds models. We may want *true knowledge*: if something is known then it is true. Then we consider models with reflexive accessibility relations, where the agents must always consider the current world. If we want *positive introspection* (if the agent knows something then it knows that it knows it) we want transitive accessibility relations. There is a large range of epistemic principles to consider.

With a deductive proof system, we can use just a few axioms and inference rules, with different epistemic principles resulting in different axioms, to reason about the consequences of such epistemic principles: what formulas classify different possible-worlds models and what formulas are true on this class. To trust such reasoning we need to know that the system is sound and thus only allows us to deduce formulas that hold on our considered class. Moreover, we want the system to be complete: if we cannot prove a formula, then it is not due to a limitation of the proof system but because the formula is “incorrect”, i.e. does not hold on our considered class.

In this paper I formalize epistemic logic [16] in the Isabelle/HOL [27] proof assistant. I consider the so-called normal modal logics, from the smallest, system **K**, which is valid over all models, to **S5**, where the accessibility relations must be reflexive, symmetric and transitive. I base my proofs on the textbooks *Reasoning About Knowledge* by Fagin, Halpern, Moses and Vardi [13], which mainly proves completeness for system **K**, and *Modal Logic* by Blackburn, de Rijke and Venema [7], which goes further. The formalization of public announcement logic (PAL) is based on the entry by Baltag and Renne in the Stanford Encyclopedia of Philosophy [2]. The chosen proof system is $PA + DIST! + NEC!$ according to the classification by Wang and Cao [30].

Unfortunately, textbooks tend to treat extensions of system **K** informally. For the completeness of system **T** on reflexive models, Fagin et al. [13] write: “A proof identical to that of Theorem 3.1.3 can now be used.” In a formalized setting we do not want to *copy/paste* our efforts but rather to find a suitable abstraction of the theorem that works for both cases. We should seek to achieve the compositionality expressed by Blackburn et al. [7] (emphasis mine):

The canonical frame of any normal logic containing T is reflexive, the canonical frame of any normal logic containing B is symmetric, and the canonical frame of any normal logic containing D is right unbounded. *This allows us to ‘add together’ our results.*

To this effect, I give a disciplined treatment of normal modal logics and completeness-via-canonicity [7] by formalizing an abstract account of the Henkin-style completeness method. This is made possible by parameterizing the proof system and the notion of a maximal consistent set to allow for an open-ended number of additional axioms. I obtain an abstract completeness theorem that just depends on the class of the canonical model. My scheme then allows me to add together canonicity results in exactly the way expressed by Blackburn et al. [7]. Where Fagin et al. [13] suggest using an identical proof, I simply instantiate my abstract result. For the formalization of PAL, I implicitly rely on the idea of a PAL-friendly theory [2] to state my results. The use and interplay of three predicates, A on axioms, P on models and, for PAL, B on announced formulas, make the soundness and completeness results easy to both state abstractly and work with concretely.

1.1 Contributions and Previous Work

A short extended abstract [18] describes an earlier version of the EL formalization, covering only system **K** and not the family of normal modal logics. We used an earlier version of the formalization of PAL as one of the examples in a chapter on interactive theorem proving for logic and information [29]. It included a weak soundness and completeness result for PAL with no additional axioms. Finally, my WoLLIC paper [14], which this extends, described the EL formalization of normal modal logics and completeness for various choices of axioms. The following contributions are new to the present paper:

- The completeness-via-canonicity technique is lifted to an abstract theorem for normal modal logics, resulting in shorter proofs for extensions of **K**.
- I use transfinite induction to prove Lindenbaum’s lemma, removing the previous restriction to countably many agents.
- I prove *strong* soundness and completeness for PAL.
- I consider extensions of PAL with various axioms.
- I parameterize the proof system for PAL with a predicate on announced formulas, giving more precise and reusable soundness and completeness results.
- I prove soundness and completeness for systems **K5** and $\text{PAL} + \mathbf{K5}$ over Euclidean accessibility relations.
- The presentation of the Isabelle code and the notation used in the formalization are both improved compared to previous work.

I continue with some remarks on the formalizations before discussing related work (Section 2). Then follow a section on epistemic logic (Section 3) and one on public announcement logic (Section 4). I conclude by pointing out potential future work (Section 5).

1.2 Remarks on the Formalization

I have specified my definitions and proofs in the precise language of higher-order logic and the Isabelle/HOL proof assistant has checked every step of the

reasoning mechanically. While the end results are not new, this approach ensures a new level of precision and guarantee of correctness. The formalizations [16, 17] are available online in the (development version) of the Archive of Formal Proofs (AFP). They have been refereed by Isabelle experts to appear in the AFP. This paper provides a recipe for extending the work with similar logics and the formalizations can serve as starting point for related and future work.

The formalization of EL consists of around 1600 lines of Isabelle/HOL text, while for PAL the number is around 1000. Around 1100 lines of the EL formalization prove the abstract completeness result and the remaining 500 concern concrete systems. For PAL, I prove abstract soundness and completeness in around 700 lines and use 300 (plus the EL formalization) to apply the results.

I reproduce a number of definitions and results from the formalization here (in *italics*), but none of the proofs. The Isabelle text is close enough to formal English, with notation from mathematical logic, that I trust the accompanying explanations to make it understandable. I present the paper in this way for two reasons. One, to remind the reader of the formal precision behind the results: the proof of each stated result was checked by the proof assistant. Two, to provide familiarity with the formalization for those who want to examine or extend it. The presentation of the results, e.g. the surrounding Isabelle keywords, is optimized for clarity and may not match the formalization exactly. I omit the usual delimiters around statements from the presentation of the Isabelle code.

Producing a formalization of this size is a significant undertaking: every significant step of reasoning in every proof must be written down explicitly and the gaps between them must be sufficiently small that the proof assistant can fill them in. This is a craft. Some of the process is pure labor, but a significant part is getting the definitions right, not just to match our intended meaning but so that they are easy to work with. For instance, we can formalize substitution instances of propositional tautologies (cf. Section 3.2) or the worlds of the canonical model (cf. Section 3.4) in different ways. One choice can lead to a lot less work than another. I want to stress a benefit of this too: duplicating work becomes that much more expensive. It is much better to prove a general result than to copy, paste and tweak a concrete result a handful of times. This encourages reuse. For instance, the results for PAL rely heavily on the work done for EL and we know that it does so correctly, since Isabelle ensures that the definitions etc. line up according to our expectations.

2 Related Work

Wu and Goré [31] formalize modal tableaux with histories for the modal logics **K**, **KT** and **S4** in Lean, giving formally verified decision procedures for these logics. Bentzen [3] also works in Lean but formalizes a Hilbert-style system for single-agent **S5** with a Henkin-style completeness proof similar to mine. Bentzen only considers **S5** built from axioms T, B and 4, while I also consider the combination of T and 5, as well as a wider range of normal modal logics. Both Li [23] and Neeley [26] have recently formalized dynamic epistemic logic [12] in

Lean including Henkin-style completeness proofs for multi-agent **S5** and public announcement logic. Hagemeyer [20] formalized intuitionistic epistemic logic in Coq including completeness of a natural deduction proof system. Slightly differently, Magessi and Brogi [24] have given a formal proof of modal completeness for provability logic in HOL Light. All of this work focuses on specific systems rather than an abstract family of logics, as I do here. The unverified tool SQEMA by Conradie et al. [11] proves canonicity of modal formulas, giving a purely algorithmic approach to proving canonical completeness in modal logic. Balco et al. [1] have built software tool support for reasoning about dynamic modal logics in Isabelle/HOL. Their toolbox can generate both shallow and deep embeddings of a calculus and they use it to formally verify the solution of the muddy children puzzle for any number of children.

Benzmüller and Reiche [4, 5] have formalized public announcement logic and the wise men puzzle in Isabelle/HOL. They employ a shallow embedding of the logic as a fragment of HOL, such that they can benefit maximally from the automation provided by Isabelle/HOL. This, however, makes it impossible for them to prove soundness and completeness as I do here.

Xiong et al. [32] present a variant of epistemic logic that adds the notion of secret knowledge as a first-class citizen. They introduce a new modality of secrets instead of defining it in terms of the knowledge operator. The authors argue that the main principles can be studied this way, for instance when considering a language with an operator for secrets and without the usual knowledge operator. It would be interesting to formalize their work in a proof assistant.

Kądziołka [22] formalized a solution to a logic puzzle in Isabelle/HOL, using a logic tailored to the problem that is very similar to the possible-worlds model of epistemic logic. My formalization could potentially be used for this reasoning. Guzman [19] has built on my epistemic logic entry in the Archive of Formal proofs to formalize the soundness and completeness of Stalnaker’s epistemic logic.

Blanchette et al. [9] formalize an abstract completeness result for various flavors of first-order logic. Besides the different logic, they consider Gentzen and tableau systems, where I consider axiomatic proof systems. Their technique is analytic, based on inspecting infinite proof attempts, where the one here is synthetic, in the Henkin style, using maximal consistent sets of formulas. I have also used the synthetic technique to formalize completeness for a tableau system for hybrid logic [15].

3 Epistemic Logic

I first discuss the syntax and semantics of the epistemic logic and the formalization of normal modal logics before diving into abstract soundness and completeness, and finally results for concrete systems.

3.1 Syntax and Semantics

The well-formed formulas of the epistemic logic are given by the following grammar, where I denote propositional symbols by x and agent labels by i :

$$p ::= \perp \mid x \mid p \vee p \mid p \wedge p \mid p \rightarrow p \mid K_i p$$

The K_i operator, “agent i knows”, is typically written \Box_i in non-epistemic multimodal logic. I write L_i for the dual modality, “agent i considers possible”, typically written \Diamond_i . I use x for propositional symbols, since I use p instead of φ for formulas to follow the Isabelle/HOL text. This is a common choice [3, 6, 24] in formalizations of logic and their presentations, regardless of the chosen proof assistant. Similarly, I sometimes write $K\ i$ for K_i .

I deeply embed the language as a datatype in the higher-order logic of Isabelle/HOL, with a constructor for each case of the grammar. Formulas in epistemic logic then become objects in the meta-logic that we can manipulate. For instance, the semantics is a predicate on this datatype, a model and a world. The datatype declaration in Isabelle/HOL looks as follows, where id is a type synonym for strings and the type variable $'i$ represents the type of agent labels:

```
datatype 'i fm
  = FF ( $\perp$ )
  | Pro  $id$ 
  | Dis ('i fm) ('i fm) (infixr  $\vee$  60)
  | Con ('i fm) ('i fm) (infixr  $\wedge$  65)
  | Imp ('i fm) ('i fm) (infixr  $\longrightarrow$  55)
  | K 'i ('i fm)
```

I formalize Kripke models as a kind of datatype known as a record, here using type variables $'w$ to represent the non-empty domain of worlds and $'i$ for the type of agent labels. I include the set of worlds, drawn from the domain, explicitly as \mathcal{W} . In doing so, I can consider models over sets of worlds that are cumbersome to define as subtypes. I write π for the valuation of propositional symbols and \mathcal{K} for the accessibility relations, which are indexed by an agent label. In Isabelle/HOL, the record is split in two. First the frame ($X \Rightarrow Y$ in Isabelle denotes a function from X to Y):

```
record ('i, 'w) frame =
   $\mathcal{W} :: 'w\ set$ 
   $\mathcal{K} :: 'i \Rightarrow 'w \Rightarrow 'w\ set$ 
```

Kripke models extend frames (id is again a type synonym for strings):

```
record ('i, 'w) kripke =
  ('i, 'w) frame +
   $\pi :: 'w \Rightarrow id \Rightarrow bool$ 
```

As mentioned, formula satisfiability has the following type:

```
('i, 'w) kripke  $\Rightarrow 'w \Rightarrow 'i\ fm \Rightarrow bool$ 
```

$$\begin{aligned}
 & \text{tautology } p \implies A \vdash p \\
 & A \vdash K_i p \wedge K_i (p \longrightarrow q) \longrightarrow K_i q \\
 & A p \implies A \vdash p \\
 & A \vdash p \implies A \vdash p \longrightarrow q \implies A \vdash q \\
 & A \vdash p \implies A \vdash K_i p
 \end{aligned}$$

Fig. 1. Proof system for EL using axiom predicate A .

The predicate itself is defined in Isabelle/HOL by the following clauses:

$$\begin{aligned}
 & M, w \models \perp \longleftrightarrow \text{False} \\
 & M, w \models \text{Pro } x \longleftrightarrow \pi M w x \\
 & M, w \models p \vee q \longleftrightarrow M, w \models p \vee M, w \models q \\
 & M, w \models p \wedge q \longleftrightarrow M, w \models p \wedge M, w \models q \\
 & M, w \models p \longrightarrow q \longleftrightarrow M, w \models p \longrightarrow M, w \models q \\
 & M, w \models K_i p \longleftrightarrow (\forall v \in \mathcal{W} M \cap \mathcal{K} M i w. M, v \models p)
 \end{aligned}$$

The notation πM stands for the valuation of model M . Note that for K_i , the semantics only considers accessible worlds in the set \mathcal{W} (of M).

3.2 Normal Modal Logic

I consider the family of normal modal logics, namely those that contain all substitution instances of propositional tautologies and all formulas of the form $K_i \phi \wedge K_i (\phi \rightarrow \psi) \rightarrow K_i \psi$ and are closed under modus ponens and necessitation.

Propositional Tautologies We need a way to characterize substitution instances of propositional tautologies, so we can give an axiom that derives them. For instance, $K_i x \vee \neg K_i x$ should be derivable because it is obtained by substituting $K_i x$ for p in the tautology $p \vee \neg p$. To avoid formalizing substitution, which can be messy, I classify tautologies semantically, giving a propositional semantics for the language. That is, I treat formulas of the form $K_i \phi$ as a different sort of propositional symbols whose truth value is given by another valuation. I use this characterization because it is easier to formalize but it also corresponds to Smullyan's [28] *boolean valuations*.

Definition 1 (Tautologies). *The propositional semantics, $eval$, corresponds to \models but treats formulas $K_i p$ as a second sort of propositions:*

$$eval - h (K_i p) = h (K_i p)$$

The substitution instances of propositional tautologies are the formulas valid under this propositional semantics:

abbreviation $tautology p \equiv \forall g h. eval g h p$

Proof System I formalize the family of normal modal logics as the inductive predicate \vdash in Fig. 1, which is parameterized by a predicate A . This *axiom predicate* is used to admit additional formulas as axioms. The implication \implies in Isabelle allows us to conclude the right-hand side from the left.

The first axiom (schema) derives any tautology (under any axiom predicate) and the second derives any instance of the distributivity of K_i over implication. The special third axiom derives any formula admitted by A . Finally, the first rule (using \implies) is modus ponens and the second one is necessitation.

Note that logics in this family are only closed under substitution when A is: it is harmless to permit an A that admits $K_i x \rightarrow x$ but not $K_i x' \rightarrow x'$ for distinct propositional symbols x and x' , but I do not consider any concrete logics with this property here.

We obtain the smallest normal modal logic **K** when A admits no extra axioms.

I sometimes say that a formula is *A-derivable* to clarify that it is derivable under the axiom predicate A .

Working with Assumptions We need a notion of deriving a formula from a number of assumptions. Instead of baking this into the proof system, requiring extra rules, I build it from the existing syntax for implication. The following function builds a chain of these. The notation \square stands for the empty list in Isabelle/HOL and $\#$ separates the head and tail of a list.

Definition 2 (Chains of implications). *If $ps = [p_1, \dots, p_n]$ is some (finite and potentially empty) list of formulas then the expression $ps \rightsquigarrow q$ builds the formula $p_1 \rightarrow \dots \rightarrow p_n \rightarrow q$ using the following Isabelle/HOL rules:*

$$\begin{aligned} (\square \rightsquigarrow q) &= q \\ (p \# ps \rightsquigarrow q) &= (p \longrightarrow ps \rightsquigarrow q) \end{aligned}$$

The following notation codifies the idea of deriving a formula from a set of assumptions.

Definition 3 (Deriving from assumptions). *We can A-derive formula p from the set of assumptions G if there exists a finite subset qs of G such that we can A-derive $qs \rightsquigarrow p$:*

$$\text{abbreviation } A; G \vdash p \equiv \exists qs. \text{ set } qs \subseteq G \wedge (A \vdash qs \rightsquigarrow p)$$

We now have two notions: the axioms admitted by predicate A and the assumptions in the set G . We could make do with either one on its own, modeling axioms as assumptions or using A to admit assumptions. However, in doing so we would lose accuracy: while axioms can be derived as formulas on their own, assumptions always appear in the antecedent. This distinction makes it possible to show that certain choices of A force the canonical model to take a certain shape, regardless of the assumptions we are working with.

Derived Rules The formalization contains a number of derived rules that ease the completeness proof. The precise meta-language of the proof assistant makes it possible to state many such rules in a methodical way. Consider the following lemma, which lifts any derivation of an implication into the K_i operator:

assumes $A \vdash p \longrightarrow q$
shows $A \vdash K\ i\ p \longrightarrow K\ i\ q$

In the derivation of another formula, we can readily reuse this result and the proof assistant makes sure we have applied it correctly: that p and q can be instantiated so the assumption is met and that the resulting conclusion matches the desired one. In this way, we can compose larger derivations out of smaller pieces in a disciplined way and the proof automation of Isabelle can even help find the right pieces and put them together for us.

3.3 Abstract Soundness

I now give a generalized soundness result for the family of logics. The Isabelle symbol \bigwedge quantifies universally over a variable.

Theorem 1 (Soundness of normal modal logics). *Fix a predicate P on models (e.g. admitting reflexive ones). If the axioms admitted by A are sound on P -models, then the normal modal logic based on A is sound on P -models.*

assumes $\bigwedge M\ w\ p. A\ p \implies P\ M \implies w \in \mathcal{W}\ M \implies M, w \models p$
shows $A \vdash p \implies P\ M \implies w \in \mathcal{W}\ M \implies M, w \models p$

Proof. By induction over the derivation and relying on the simple fact that any *tautology* is valid in all models.

After deriving some proof rules, we can extend the result above to strong soundness. First, I introduce an abbreviation for validity.

Definition 4 (P -validity). *The formula p is P -valid under G if for all P -models M and worlds w in M , whenever all of G is satisfied at w so is p :*

abbreviation $P; G \models^\star p \equiv \forall M. P\ M \longrightarrow$
 $(\forall w \in \mathcal{W}\ M. (\forall q \in G. M, w \models q) \longrightarrow M, w \models p)$

The (hidden) type variables $'i$ and $'w$ in the above definition are arbitrary. Later I introduce another abbreviation without the star with exactly the same definition but for a specific type of worlds needed for the completeness proof. We will see that the specific validity implies the general one. First, strong soundness.

Theorem 2 (Strong soundness of normal modal logics). *Fix a predicate P on models. If the axioms admitted by A are sound on P -models, then the normal modal logic based on A is strongly sound on P -models.*

assumes $\bigwedge M\ w\ p. A\ p \implies P\ M \implies w \in \mathcal{W}\ M \implies M, w \models p$
shows $A; G \vdash p \implies P; G \models^\star p$

Proof. By induction on the list of assumptions and for arbitrary target formula p . In the case of no assumptions, Theorem 1 applies. Otherwise, the chain of implications looks like $q \longrightarrow qs \rightsquigarrow p$ for some formula q and list qs . Rearrange this to $qs \rightsquigarrow (q \longrightarrow p)$. The induction hypothesis then proves the thesis.

While the predicate P in the above theorem can in principle inspect the valuation of the model, I will only use predicates on the frame part.

3.4 Abstract Completeness

I follow the completeness proof for system **K** by Fagin et al. [13] but in my generalized setting. As such, I do not just talk about maximal consistent sets (MCSs) but MCSs with respect to a choice of axiom predicate A (A -MCSs). Likewise, the canonical model is parameterized by such an A , which I fix later to obtain completeness of various logics over various classes of frames.

Maximal Consistent Sets I first define consistency and maximality with respect to an axiom predicate.

Definition 5 (A -Consistency). *A (potentially infinite) set of formulas S is A -consistent if we cannot A -derive a contradiction (\perp) from it:*

definition *consistent* $A S \equiv \neg (A; S \vdash \perp)$

Definition 6 (A -Maximality). *A set of formulas is A -maximal if it contains all formulas consistent with it:*

definition *maximal* $S \equiv \forall p. \text{consistent } (\{p\} \cup S) \longrightarrow p \in S$

It is straightforward to verify some classic properties about A -MCSs [7, 13]:

Theorem 3 (A -MCS properties). *For V an A -MCS, we have (i) any derivable formula (using A) is in V (including any formula admitted by A), (ii) exactly one of ϕ and $\neg\phi$ is in V and (iii) V is closed under modus ponens.*

assumes *consistent* $A V$ **and** *maximal* $A V$

shows $A \vdash p \implies p \in V$

and $p \in V \longleftrightarrow (\neg p) \notin V$

and $p \in V \implies (p \longrightarrow q) \in V \implies q \in V$

Proof. See the formalization or a textbook like the one by Blackburn et al. [7].

Lindenbaum's Lemma Our goal is now to extend any A -consistent set to an A -MCS. To do this, we will employ Lindenbaum's construction (cf. Chang and Keisler [10]). Typical presentations assume an enumeration of formulas from the ordinals, but since higher-order logic is weaker than set theory and cannot, for instance, express a type of all ordinals, we work a bit differently. The set of all formulas is trivially infinite, and the cardinality of an infinite set is a limit ordinal.

Limit ordinals have the property that every element has a successor. So we will work “inside” this limit ordinal, using the formulas themselves as the numbers. This better matches the way Isabelle/HOL supports transfinite recursion and induction [8] than the traditional representation using an enumeration.

Starting from an A -consistent set of formulas S , we build an infinite sequence of consistent sets $S_0, S_1, S_2, \dots, S_\alpha, \dots$ using transfinite recursion and take their union to form the A -MCS. There are three cases for constructing S_α depending on which type of ordinal element α is. For $\alpha = 0$, $S_\alpha = S$. For $\alpha + 1$, a successor element, construct $S_{\alpha+1}$ like so:

$$S_{\alpha+1} = \begin{cases} \{\alpha\} \cup S_\alpha & \text{if } \{\alpha\} \cup S_\alpha \text{ is } A\text{-consistent} \\ S_\alpha & \text{otherwise} \end{cases}$$

To understand this definition, recall that the elements α are actually our formulas serving a dual purpose as numbers. When α is a limit element, construct S_α as the union over strictly smaller elements $S_\alpha = \bigcup_{\beta < \alpha} S_\beta$.

In Isabelle, *extend S* α constructs element S_α of the sequence, starting from S . *Extend S* gives the infinite union $\bigcup_\alpha S_\alpha$. The result is A -consistent when the starting point is A -consistent and always A -maximal.

Lemma 1 (A-consistent and A-maximal Lindenbaum extension).

If S is A-consistent, so is the Lindenbaum extension of S :

assumes *consistent S*
shows *consistent (Extend S)*

The Lindenbaum extension is always A-maximal:

shows *maximal (Extend S)*

Proof. Same proofs as for normal maximal consistent sets (cf. [7,10,16]). In the maximality proof, we use the fact that all elements have successors to show that $S_{\alpha+1}$ is defined and includes α when $\{\alpha\} \cup S_\alpha$ is A -consistent.

Model Existence I first define the canonical model based on A -MCSs.

Definition 7 (Canonical Model).

The worlds are A-MCSs:

abbreviation $mcss\ A \equiv \{W. \text{consistent } A\ W \wedge \text{maximal } A\ W\}$

The valuation, π , and accessibility relation, $reach$, are as follows:

abbreviation $\pi\ V\ x \equiv \text{Pro } x \in V$

abbreviation $known\ V\ i \equiv \{p. K\ i\ p \in V\}$

abbreviation $reach\ A\ i\ V \equiv \{W. \text{known } V\ i \subseteq W\}$

In total, I abbreviate the canonical model built from axiom predicate A as:

abbreviation $\text{canonical } A \equiv (\mathcal{W} = mcss\ A, \mathcal{K} = reach\ A, \pi = \pi)$

The valuation pi states that proposition x holds in a world V if and only if $x \in V$. Where Fagin et al. [13] write V/K_i for the set of formulas known by agent i at V , i.e. $\{p \mid K_i p \in V\}$, I write $known\ V\ i$. The worlds *reachable* by i from V are those A -MCSs that contain all formulas *known* at V by i .

I formalize the usual truth lemma.

Lemma 2 (Truth lemma). *The canonical model for axiom predicate A satisfies a formula p at an A -MCS V if and only if $p \in V$:*

assumes consistent $A\ V$ **and** maximal $A\ V$
shows $p \in V \iff \text{canonical } A, V \models p$

Proof. By structural induction on the formula p . The only non-trivial case is for the K_i -operator where we need to show that when $K_i q$ is satisfied at V then $K_i q \in V$. I follow the proof by Fagin et al. [13] and note that $\neg q$ must be inconsistent with the formulas *known* at V by i . If they were consistent, they could be extended into an A -MCS satisfying both $\neg q$ and $known\ V\ i$, making it accessible from V , which would then satisfy $\neg K_i q$ as well as $K_i q$ (a contradiction). Thus, we can derive q from some finite subset $L \subseteq known\ V\ i$ of the known formulas. By necessitation, agent i knows this and by distributivity of K_i over implication, if agent i knows all of L then it knows q . Since L is a subset of what agent i knows, the thesis follows immediately.

Since the canonical model has A -MCS worlds, it has a very specific type: the type of Kripke models whose worlds are sets of formulas. For these to be our countermodels, the completeness results must assume validity over this type of models. To ease notation, I specialize the existing validity abbreviation.

Definition 8 (P -validity in the universe of MCSs).

Abbreviate $P; G \models p \equiv P; G \models_\star p$ at the specific type:

$$((\text{'i, 'i fm set})\ \text{kripke} \Rightarrow \text{bool}) \Rightarrow \text{'i fm set} \Rightarrow \text{'i fm} \Rightarrow \text{bool}$$

We can now state and prove abstract strong completeness succinctly.

Theorem 4 (Strong completeness). *If p is P -valid under G and the canonical model for A is a P -model, then we can A -derive p from G :*

assumes $P; G \models p$ **and** P (canonical A)
shows $A; G \vdash p$

Proof. If p is valid under assumptions G but has no A -derivation from G , then $\{\neg p\} \cup G$ is A -consistent and the canonical model satisfies both $\neg p$ and all of G . This, however, contradicts the P -validity of p under G since the canonical model is assumed to be a P -model.

In the following I consider choices of A that impose structure on the canonical model, yielding completeness over various classes of frames.

Table 1. Epistemic axioms.

Axiom	Formula	Frame condition	Principle
T	$K_i\varphi \rightarrow \varphi$	Reflexive	True knowledge
B	$\varphi \rightarrow K_i L_i \varphi$	Symmetric	Knowledge of consistency of truths ^a
4	$K_i\varphi \rightarrow K_i K_i \varphi$	Transitive	Positive introspection
5	$L_i\varphi \rightarrow K_i L_i \varphi$	Euclidean	Negative introspection

^a Name suggested by Rineke Verbrugge.

Table 2. Soundness and completeness results.

System	Axioms	Class
K		All frames
T	T	Reflexive frames
KB	B	Symmetric frames
K4	4	Transitive frames
K5	5	Euclidean frames
S4	T, 4	Reflexive and transitive frames
S5	T, B, 4	Frames with equivalence relations
S5'	T, 5	Frames with equivalence relations

3.5 Concrete Systems

I now consider logics based on the axioms in Table 1 and show how we can compose axioms to easily obtain completeness over the class of frames resulting from each axiom's contribution.

I abbreviate validity over a class of frames as \models_X - where X is a system from Table 2, which identifies the corresponding class. For instance \models_{K4} - abbreviates validity over transitive frames. Moreover, the proof system \vdash_X consists of the axioms listed for system X in Table 2.

Recipe: System T I consider system **T** as an example of my general recipe for formalizing strong soundness and completeness of a canonical normal modal logic. Define a predicate that admits the right axioms: AxT ($K i p \rightarrow p$).

Next, define the proof system based on the axiom predicate:

abbreviation $G \vdash_T p \equiv AxT; G \vdash p$

Prove soundness of the axiom on the proper class of models:

shows $AxT p \implies \text{reflexive } M \implies w \in W M \implies M, w \models p$

By Theorem 2, System **T** is then strongly sound on reflexive models:

shows $G \vdash_T p \implies \text{reflexive}; G \models_\star p$

Note that this derives validity in any type of reflexive model.

Next, prove that the axioms force the canonical model to belong to the right class. If the predicate A admits all the formulas admitted by axiom T then every A -MCS can *reach* itself, by virtue of the canonical accessibility relation:

assumes $AxT \leq A$ **and** *consistent* $A \ V$ **and** *maximal* $A \ V$
shows $V \in \text{reach } A \ i \ V$

Therefore, when A admits everything T does, the canonical model is reflexive:

assumes $AxT \leq A$
shows *reflexive* (*canonical* A)

Strong completeness follows directly from Theorem 4:

shows $G \models_T p \implies G \vdash_T p$

In total we have the following theorem.

Theorem 5 (System T wrt. reflexive models).

T-validity corresponds exactly with T-derivability:

shows $G \models_T p \longleftrightarrow G \vdash_T p$

Proof. Using the generalized results (Theorems 2 and 4).

As noted, for the completeness result I assume validity over one type of worlds only (as restricted by $-$; $- \models -$ rather than $-$; $- \models_\star -$). This a stronger result than assuming validity in all universes. Moreover, validity in the specific universe implies validity in any other.

Corollary 1. *If p is T-valid under G on the type of reflexive models with sets of formulas as worlds, then p is T-valid under G on any type of reflexive models:*

shows $G \models_T p \implies \text{reflexive}; G \models_\star p$

Proof. By composing strong soundness and completeness of system T.

The formalization contains a similar corollary for every concrete system.

System K System K is the smallest normal modal logic so A always returns false. It is strongly sound and complete on all models.

Theorem 6 (System K wrt. all models).

K-validity corresponds exactly with K-derivability:

shows $G \models_K p \longleftrightarrow G \vdash_K p$

Proof. Follows directly from the generalized results (Theorems 2 and 4).

System KB I formalize axiom B from Table 1 as I did with T. The corresponding logic **KB** is strongly sound and complete over symmetric models.

Theorem 7 (System KB wrt. symmetric models).

KB-validity corresponds exactly with KB-derivability:

shows $G \models_{KB} p \longleftrightarrow G \vdash_{KB} p$

Proof. Isabelle easily verifies that axiom B is sound on symmetric frames:

have $AxB \implies \text{symmetric } M \implies w \in \mathcal{W} \ M \implies M, w \models p$

And, with a bit of help, that imposes symmetry on the canonical frame:

have $AxB \leq A \implies AxB \leq A$

Soundness and completeness then follow from Theorems 2 and 4.

System K4

Theorem 8 (System K4 wrt. transitive models).

K4-validity corresponds exactly with K4-derivability:

shows $G \models_{K4} p \longleftrightarrow G \vdash_{K4} p$

Proof. As for Theorem 5. See the formalization for details [16].

System K5

Theorem 9 (System K5 wrt. Euclidean models).

K5-validity corresponds exactly with K5-derivability:

shows $G \models_{K5} p \longleftrightarrow G \vdash_{K5} p$

Proof. As for Theorem 5. See the formalization for details [16].

System S4 The following abbreviation combines axiom predicates.

Definition 9 (Combining axiom predicates).

$A \oplus A'$ admits p as an axiom if either A or A' does:

abbreviation $(A \oplus A') p \equiv A p \vee A' p$

We can then define system **S4** as the combination of axioms T and 4:

abbreviation $G \vdash_{S4} p \equiv AxT \oplus Ax4; G \vdash p$

Theorem 10 (System S4 wrt. reflexive and transitive models).

S4-validity corresponds exactly with S4-derivability:

shows $G \models_{S4} p \longleftrightarrow G \vdash_{S4} p$

Proof. Each axiom imposes the required condition on the canonical frame (cf. the proofs of Theorems 7 and 8) so the result follows from Theorems 2 and 4.

System S5 I formalize two versions of System **S5**. The first version is obtained by combining the axioms T, B and 4. In formalizing the systems **T**, **KB** and **K4**, I have already proved that these axioms force the canonical model to be reflexive, symmetric and transitive, respectively. Thus, their composition guarantees equivalence relations. The second version uses the more traditional combination of axioms T and 5. I show completeness for this system by proving that reflexive and Euclidean models have equivalence relations. Afterwards, I outline how to syntactically derive the axioms of each system from the axioms of the other.

Axioms T + B + 4 I combine the axioms and follow the recipe from before, reusing the canonicity results, and leading to the expected result. First:

abbreviation $G \vdash_{S5} p \equiv AxT \oplus AxB \oplus Ax4; G \vdash p$

Theorem 11 (System S5 wrt. models with equivalence relations).

S5-validity corresponds exactly with S5-derivability:

shows $G \models_{S5} p \longleftrightarrow G \vdash_{S5} p$

Proof. Each axiom is sound on models with equivalence relations and imposes the required conditions on the canonical frame (cf. the proofs of Theorems 5, 7 and 8) so the result follows from Theorems 2 and 4.

Axioms T + 5 I define a predicate for axiom 5: $Ax5 (L i p \longrightarrow K i (L i p))$. I then define a different version of **S5** and call it **S5'**:

abbreviation $G \vdash_{S5'} p \equiv AxT \oplus Ax5; G \vdash p$

Theorem 12 (System S5' wrt. models with equivalence relations).

S5'-validity corresponds exactly with S5'-derivability:

shows $G \models_{S5} p \longleftrightarrow G \vdash_{S5'} p$

Proof. We have already seen that axiom 5 is sound on models with Euclidean relations. Isabelle can show in one line that symmetric and transitive models have Euclidean relations:

have $\text{symmetric } M \implies \text{transitive } M \implies \text{Euclidean } M$

The soundness then follows from the previous results (Theorems 5 and 9):

have $AxT5 p \implies \text{equivalence } M \implies w \in \mathcal{W} M \implies M, w \models p$

We know that axiom T imposes reflexivity on the canonical model and that axiom 5 imposes Euclideaness. Isabelle easily verifies that the two conditions guarantee equivalence relations:

have $\text{reflexive } M \implies \text{Euclidean } M \implies \text{equivalence } M$

Each axiom is thus sound on models with equivalence relations and imposes the required conditions on the canonical frame (cf. the proofs of Theorems 5 and 9) so the result follows from Theorems 2 and 4. We have achieved the compositionality promised by the completeness-via-canonicity technique.

Syntactic derivations The formalization makes it easy to prove compositional rules. For instance, whenever we have at least axiom 4 available, we can prove the dual version of it, which uses L_i instead of K_i :

assumes $Ax_4 \leq A$
shows $A \vdash L\ i\ (L\ i\ p) \longrightarrow L\ i\ p$

We can then derive axiom 5 as follows.

Lemma 3 (Axiom 5 from B and 4).

We can derive axiom 5 from B and 4:

assumes $AxB \leq A$ **and** $Ax_4 \leq A$
shows $A \vdash L\ i\ p \longrightarrow K\ i\ (L\ i\ p)$

Proof. The proof has four steps. I show the intermediate results here as they appear in the Isabelle formalization.

First, we instantiate axiom B:

have $A \vdash L\ i\ p \longrightarrow K\ i\ (L\ i\ (L\ i\ p))$

Moreover, we use the version of axiom 4 from above:

have $A \vdash L\ i\ (L\ i\ p) \longrightarrow L\ i\ p$

We then lift this into K_i operator:

have $A \vdash K\ i\ (L\ i\ (L\ i\ p)) \longrightarrow K\ i\ (L\ i\ p)$

And by transitivity we have the result:

have $A \vdash L\ i\ p \longrightarrow K\ i\ (L\ i\ p)$

Lemma 4 (Axiom B from T and 5).

We can derive axiom B from T and 5:

assumes $AxT \leq A$ **and** $Ax_5 \leq A$
shows $A \vdash p \longrightarrow K\ i\ (L\ i\ p)$

Proof. We can derive the dual version of axiom T:

have $A \vdash p \longrightarrow L\ i\ p$

And instantiate axiom 5 to:

have $A \vdash L\ i\ p \longrightarrow K\ i\ (L\ i\ p)$

The thesis follows by transitivity.

Lemma 5 (Axiom 4 from T and 5).

assumes $AxT \leq A$ **and** $Ax_5 \leq A$
shows $A \vdash K\ i\ p \longrightarrow K\ i\ (K\ i\ p)$

Proof. The crucial derivations are the following instantiation of axiom 5:

have $A \vdash L\ i\ (K\ i\ p) \longrightarrow K\ i\ (L\ i\ (K\ i\ p))$

The dual version of axiom T:

have $A \vdash K\ i\ p \longrightarrow L\ i\ (K\ i\ p)$

And the dual version of axiom 5 (lifted into K_i :

have $A \vdash K\ i\ (L\ i\ (K\ i\ p)) \longrightarrow K\ i\ (K\ i\ p)$

The thesis follows by transitivity (cf. [16]).

We can then prove derivational equivalence between the two systems.

Lemma 6 (Systems S5 and S5' are equivalent).

S5-derivability corresponds exactly with S5'-derivability:

shows $G \vdash_{S5} p \longleftrightarrow G \vdash_{S5'} p$

Proof. By inductions on each proof system and using Lemmas 3 to 5.

4 Public Announcement Logic

I first discuss the syntax and semantics of the public announcement logic and the formalization of a proof system for any choice of axioms before diving into abstract soundness and completeness, and finally results for concrete systems.

4.1 Syntax and Semantics

The well-formed formulas of PAL are given by the following grammar:

$$p ::= \perp \mid x \mid p \vee \text{!} p \mid p \wedge \text{!} p \mid p \rightarrow \text{!} p \mid K_i p \mid [p] \text{!} p$$

I index connectives by an exclamation point to distinguish them formally from the previous syntax. The new type of formula $[r] \text{!} p$ stands for the truth of p after an announcement of r . The same abbreviations as before apply. In Isabelle/HOL, the formulas are represented by the following datatype:

```
datatype 'i pfm
= FF (⊥!)
| Pro' id (Pro!)
| Dis ('i pfm) ('i pfm) (infixr ∨! 60)
| Con ('i pfm) ('i pfm) (infixr ∧! 65)
| Imp ('i pfm) ('i pfm) (infixr →! 55)
| K' 'i ('i pfm) (K!)
| Ann ('i pfm) ('i pfm) ([·]! - [80, 80] 80)
```

I reuse the formalization of Kripke models from EL directly. The semantics for PAL has the same clauses as for EL for the shared connectives. For the announcements, we have the following:

$$\begin{aligned}
 & ptautology\ p \implies A; B \vdash_! p \\
 & A; B \vdash_! K_! i\ p \wedge_! K_! i\ (p \longrightarrow_! q) \longrightarrow_! K_! i\ q \\
 & A\ p \implies A; B \vdash_! p \\
 & A; B \vdash_! p \implies A; B \vdash_! p \longrightarrow_! q \implies A; B \vdash_! q \\
 & A; B \vdash_! p \implies A; B \vdash_! K_! i\ p \\
 & A; B \vdash_! p \implies B\ r \implies A; B \vdash_! [r]_! p \\
 \\
 & A; B \vdash_! [r]_! \perp_! \longleftrightarrow_! (r \longrightarrow_! \perp_!) \\
 & A; B \vdash_! [r]_! Pro_! x \longleftrightarrow_! (r \longrightarrow_! Pro_! x) \\
 & A; B \vdash_! [r]_! (p \vee_! q) \longleftrightarrow_! [r]_! p \vee_! [r]_! q \\
 & A; B \vdash_! [r]_! (p \wedge_! q) \longleftrightarrow_! [r]_! p \wedge_! [r]_! q \\
 & A; B \vdash_! [r]_! (p \longrightarrow_! q) \longleftrightarrow_! ([r]_! p \longrightarrow_! [r]_! q) \\
 & A; B \vdash_! [r]_! K_! i\ p \longleftrightarrow_! (r \longrightarrow_! K_! i\ ([r]_! p))
 \end{aligned}$$

Fig. 2. Proof system for PAL with axiom predicate A and announcement predicate B .

$$M, w \models [r]_! p \longleftrightarrow M, w \models r \longrightarrow M[r]_!, w \models p$$

If the announcement r is true then the truth of the formula is given by p in a restricted model defined as follows:

$$M[r]_! = M \restriction \mathcal{W} := \{w. w \in \mathcal{W} \mid M, w \models r\}$$

That is, restricting the model M to the formula r simply means to take only the subset of worlds in M that satisfy r .

The functions *lift* and *lower*, move from EL to PAL and vice versa. The effect of lowering an announcement is undefined and cannot be depended on.

4.2 Proof System

Figure 2 presents the proof system.

The first six lines are similar to the normal modal logics of EL (cf. Fig. 1). The tautology predicate is now called *ptautology* but it is based on the same idea. The idea of the axiom predicate A is exactly the same but it now works on PAL formulas. Notably, we now have a necessitation rule for announcements which states that if *announcement predicate* B admits the formula r then we can derive $[r]_! p$ from a derivation of p . This predicate effectively allows us to work in fragments of PAL where only certain types of formulas can be announced. As such it is analogous to the axiom predicate A .

The six lines that follow in Fig. 2 give reduction axioms that describe how announcements distribute over the other connectives. Since the announcement disappears at the base cases, these axioms can be used to reduce a PAL formula to the announcement-free syntax of EL.

I employ the same trick as for EL to work with assumptions (cf. Section 3.2):

$$\textbf{abbreviation } A; B; G \vdash_! p \equiv \exists qs. \text{ set } qs \subseteq G \wedge (A; B \vdash_! qs \rightsquigarrow_! p)$$

4.3 Abstract Soundness

As before, we can state an abstract soundness result by placing appropriate restrictions on the axiom and announcement predicates.

Theorem 13 (Soundness of PAL). *If all axioms admitted by A are sound on P -models and all formulas admitted by B restrict P -models to P -models, then any formula p derived under A and B is valid on P -models:*

assumes $\bigwedge M p w. A p \implies P M \implies w \in \mathcal{W} M \implies M, w \models_! p$
and $\bigwedge M r. P M \implies B r \implies P (M[r!])$
shows $A; B \vdash_! p \implies P M \implies w \in \mathcal{W} M \implies M, w \models_! p$

Proof. By induction on the derivation. Isabelle discharges all cases automatically after we have proved that any *ptautology* is valid.

Like before we can lift this to strong soundness.

Theorem 14 (Strong soundness of PAL). *If axioms A are sound on P -models and announcements B restrict P -models to P -models, then any formula p derived from assumptions G under A and B is valid under G on P -models:*

assumes $\bigwedge M w p. A p \implies P M \implies w \in \mathcal{W} M \implies M, w \models_! p$
and $\bigwedge M r. P M \implies B r \implies P (M[r!])$
shows $A; B; G \vdash_! p \implies P; G \models_! \star p$

Proof. By induction on the derivation. Isabelle discharges all cases automatically after we have proved that any *ptautology* is valid.

Similar to EL, the abbreviation $-; - \models_! \star$ - abbreviates validity under the PAL semantics in any type of models while $-; - \models_!$ - assumes the type we need to prove completeness.

4.4 Abstract Completeness

Completeness for PAL is based on the following idea: (i) reduce the valid PAL formula to a *static* one without announcements (ii) prove that it is valid in EL (iii) use a completeness result from EL to obtain a derivation of the static formula in EL (iv) transfer the derivation to PAL (v) use the reduction axioms to derive the original formula.

Reduction There are two parts to the reduction: the function *reduce'* reduces $[r]_! p$ when both r and p are static. The function *reduce* repeatedly applies *reduce'* to reduce any PAL formula to its static counterpart.

Definition 10 (Reduction of PAL formulas).

*The function *reduce'* is defined by the following rules:*

$$\begin{aligned}
 \text{reduce}' r \perp_! &= (r \longrightarrow_! \perp_!) \\
 \text{reduce}' r (Pro_! x) &= (r \longrightarrow_! Pro_! x) \\
 \text{reduce}' r (p \vee_! q) &= (\text{reduce}' r p \vee_! \text{reduce}' r q) \\
 \text{reduce}' r (p \wedge_! q) &= (\text{reduce}' r p \wedge_! \text{reduce}' r q) \\
 \text{reduce}' r (p \longrightarrow_! q) &= (\text{reduce}' r p \longrightarrow_! \text{reduce}' r q) \\
 \text{reduce}' r (K_! i p) &= (r \longrightarrow_! K_! i (\text{reduce}' r p)) \\
 \text{reduce}' r ([p]_! q) &= \text{undefined}
 \end{aligned}$$

The defining case for the function *reduce* is for announcements:

$$\text{reduce} ([r]_! p) = \text{reduce}' (\text{reduce } r) (\text{reduce } p)$$

In the other cases it simply recurses on subformulas.

These functions are not designed for efficiency but can be executed in practice to reduce a given PAL formula.

Two lemmas are critical: that *reduce* actually produces static formulas and that it preserves the semantics.

Lemma 7 (Correctness of reduction).

Any reduced formula is static:

shows *static* (*reduce* *p*)

And reduction preserves truth:

shows $M, w \models p \iff M, w \models \text{reduce } p$

Proof. By simple inductions.

Static completeness If a PAL formula is static and its lowering is derivable in EL, then it is derivable in PAL.

Lemma 8 (Strong static completeness for PAL).

If p and all of G are static, p is P -valid under G and we have strong completeness for EL, then we can derive p from G under any A and B :

assumes *static* p **and** $\forall q \in G. \text{static } q$ **and** $P; G \models_! p$
and $\bigwedge G p. P; G \models p \implies A \text{ o lift}; G \vdash p$
shows $A; B; G \vdash_! p$

Proof. First note that the semantics of PAL and EL agree on static formulas so validity in PAL implies validity in EL for this class. By assumption we then get a derivation in EL of the lowered formula. Now note that we can derive in PAL the lifted version of any formula that we can derive in EL since we have the same rules and axioms available. Finally, the result of lifting a lowered, static formula is just the formula.

Strong completeness We need a number of lemmas about reduced formulas.

Lemma 9 (Deriving reduce'). *When p is static we can derive equivalence between $[r]_!$ p and reduce' p :*

assumes *static p*
shows $A; B \vdash_! [r]_! p \longleftrightarrow_! \text{reduce}' r p$

Proof. By induction on p using the reduction axioms.

Lemma 10 (Equivalent announcements). *When p is static we can derive an equivalence between announcements from a derivation of equivalence between the announced formulas:*

assumes $A; B \vdash_! r \longleftrightarrow_! r'$ **and** *static p*
shows $A; B \vdash_! [r]_! p \longleftrightarrow_! [r']_! p$

And when r is admitted by B we can derive an equivalence between announcements from an equivalence between target formulas:

assumes $A; B \vdash_! p \longleftrightarrow_! p'$ **and** $B r$
shows $A; B \vdash_! [r]_! p \longleftrightarrow_! [r]_! p'$

Proof. By induction on p using the reduction axioms and in the second case also announcement necessitation.

We arrive at the main result of this section.

Lemma 11 (Equivalent reduction). *When all announcements in p are admitted by B , then we can derive an equivalence between p and $\text{reduce } p$:*

assumes $\forall r \in \text{anns } p. B r$
shows $A; B \vdash_! p \longleftrightarrow_! \text{reduce } p$

Proof. By induction on p using Lemmas 9 and 10.

Theorem 15 (Strong completeness for PAL). *If p is P -valid under G , all announced formulas in p and G are admitted by B and we have strong completeness for A and P in EL , then we can derive p from G under A and B :*

assumes $P; G \models_! p$
and $\forall r \in \text{anns } p. B r$ **and** $\forall q \in G. \forall r \in \text{anns } q. B r$
and $\bigwedge G p. P; G \models_\star p \implies A \text{ o lift}; G \vdash p$
shows $A; B; G \vdash_! p$

Proof. First note that reduced p is P -valid under reduced G (Lemma 7) and that the results are static. Then by the assumption and strong static completeness (Lemma 8) we can derive reduced p from reduced G . Then there is a finite list of assumptions qs in G such that we can derive the chain $\text{reduce } (qs \rightsquigarrow p)$. Finally by Lemma 11 we can derive the unreduced chain.

4.5 Concrete Systems

For each of the systems in Table 2 we can now formalize the corresponding system in PAL. As before I present the general recipe and then the main results.

Recipe: PAL + T We again need to define the axiom predicate since the syntax is formalized as a different datatype: $AxPT (K! \ i \ p \longrightarrow! \ p)$. We can then define the resulting proof system (with no restrictions on announcements):

abbreviation $G \vdash_{!T} p \equiv AxPT; (\lambda\cdot. True); G \vdash! p$

I use the same notation to denote systems as before (cf. Table 2) but with a subscript exclamation point to indicate we are working in PAL.

Isabelle easily proves that restriction preserves reflexivity of models:

shows $reflexive \ M \implies reflexive \ (M[r!])$

We also need to prove a connection between the EL and PAL axiom predicates but again this can be done automatically:

shows $AxT = AxPT \ o \ lift$

Then strong soundness and completeness follow from Theorems 14 and 15:

shows $G \models_{!T} p \longleftrightarrow G \vdash_{!T} p$

Again, validity in the specific universe implies validity in general:

shows $G \models_{!T} p \implies reflexive; G \models_{! \star} p$

Results

Theorem 16 (PAL + T, B, 4, 5, T+B+4, T+5). *We have the following soundness and completeness results for different versions of PAL:*

shows $G \models_{!K} p \longleftrightarrow G \vdash_{!K} p$
shows $G \models_{!T} p \longleftrightarrow G \vdash_{!T} p$
shows $G \models_{!KB} p \longleftrightarrow G \vdash_{!KB} p$
shows $G \models_{!K4} p \longleftrightarrow G \vdash_{!K4} p$
shows $G \models_{!K5} p \longleftrightarrow G \vdash_{!K5} p$
shows $G \models_{!S4} p \longleftrightarrow G \vdash_{!S4} p$
shows $G \models_{!S5} p \longleftrightarrow G \vdash_{!S5} p$
shows $G \models_{!S5} p \longleftrightarrow G \vdash_{!S5'} p$

Proof. By completeness for the underlying EL systems and the abstract results (Theorems 14 and 15). See the formalization for the details.

5 Conclusion and Future Work

I have formalized the syntax and semantics of epistemic logic in the proof assistant Isabelle/HOL, following the textbook by Fagin et al. [13] and providing a precise, mechanically-checked elaboration of their completeness proof (extended to also cover uncountably many agents). Instead of considering just one proof system for epistemic logic I have formalized the family of normal modal logics and given an abstract account of the Henkin-style completeness method, which can be instantiated with any logic in the family. I have formalized the canonicity of various epistemic principles and used this, alongside the abstract completeness result, to prove strong completeness of systems **K**, **T**, **KB**, **K4**, **K5**, **S4** and two variants of **S5** over their respective classes of frames. In the composite systems I have reused the results about the constituent axioms, confirming Blackburn et al.’s [7] statement that we can ‘add our results together’ in such completeness-via-canonicity arguments. I have then formalized public announcement logic over the same Kripke models and proved how we can lift completeness results from EL to completeness results for PAL. By doing so, I have succinctly formalized completeness for the PAL counterparts of the above systems. In both formalizations, three predicates, A , B and P , have played significant roles in allowing me to state and prove each result abstractly before instantiating it for the concrete systems. Working in a proof assistant, where duplicated work is doubly expensive, encourages finding solutions like these.

In the initial version of my WoLLIC paper [14], I did not include the set of worlds in the model explicitly, as I do now, but only implicitly as the inhabitants of the type variable w . This, however, required me to explicitly define a sub-type of each kind of A -MCS, to build the canonical model over. While doable, this quickly proved tedious. A recently suggested addition to Isabelle/HOL by Kunčar and Popescu [21] might alleviate this problem. I am grateful to the anonymous reviewer who pointed me back towards the current solution with an explicit set. While it may seem less intuitive to have both a domain of worlds, the type, and a concrete set of considered worlds, it currently turns out simpler.

In the future one might extend the formalization with a broader range of results about epistemic and modal logics, including a formalization of Salhquist’s correspondence theorem, which would subsume the present results for EL. It would also be interesting to consider limitative results like the non-canonicity of **KL** or the existence of incomplete logics when moving to the basic temporal language [7]. It might also be interesting to consider algebraic semantics [7], drawing on the large library of algebraic results in the Archive of Formal Proofs.

Acknowledgements

I am grateful to Alexander Birch Jensen, Frederik Krogsdal Jacobsen and Jørgen Villadsen for discussions and comments on drafts, to Valentin Goranko for suggesting both related and future work and to Bruno Bentzen for informing me of relevant formalizations. I want to also thank the anonymous reviewers for their valuable comments.

References

1. Balco, S., Frittella, S., Greco, G., Kurz, A., Palmigiano, A.: Software tool support for modular reasoning in modal logics of actions. In: Avigad, J., Mahboubi, A. (eds.) *Interactive Theorem Proving - 9th International Conference, ITP 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings. Lecture Notes in Computer Science*, vol. 10895, pp. 48–67. Springer (2018). https://doi.org/10.1007/978-3-319-94821-8_4
2. Baltag, A., Renne, B.: *Dynamic Epistemic Logic*. In: Zalta, E.N. (ed.) *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Winter 2016 edn. (2016)
3. Bentzen, B.: A Henkin-style completeness proof for the modal logic S5. In: Baroni, P., Benzmüller, C., Wang, Y.N. (eds.) *Logic and Argumentation - 4th International Conference, CLAR 2021, Hangzhou, China, October 20-22, 2021, Proceedings. Lecture Notes in Computer Science*, vol. 13040, pp. 459–467. Springer (2021). https://doi.org/10.1007/978-3-030-89391-0_25
4. Benz Müller, C., Reiche, S.: Automating public announcement logic and the wise men puzzle in Isabelle/HOL. *Archive of Formal Proofs* (Nov 2021), <https://isa-afp.org/entries/PAL.html>, Formal proof development
5. Benz Müller, C., Reiche, S.: Automating public announcement logic with relativized common knowledge as a fragment of HOL in LogiKEy (2022)
6. Berghofer, S.: First-order logic according to Fitting. *Archive of Formal Proofs* (2007), <https://isa-afp.org/entries/FOL-Fitting.html>, Formal proof development
7. Blackburn, P., de Rijke, M., Venema, Y.: *Modal Logic*, Cambridge Tracts in Theoretical Computer Science, vol. 53. Cambridge University Press (2001). <https://doi.org/10.1017/CBO9781107050884>
8. Blanchette, J.C., Popescu, A., Traytel, D.: Cardinals in Isabelle/HOL. In: Klein, G., Gamboa, R. (eds.) *Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014, Proceedings. Lecture Notes in Computer Science*, vol. 8558, pp. 111–127. Springer (2014). https://doi.org/10.1007/978-3-319-08970-6_8
9. Blanchette, J.C., Popescu, A., Traytel, D.: Soundness and completeness proofs by coinductive methods. *Journal of Automated Reasoning* **58**(1), 149–179 (2017). <https://doi.org/10.1007/s10817-016-9391-3>
10. Chang, C.C., Keisler, H.J.: *Model theory*, Third Edition, *Studies in logic and the foundations of mathematics*, vol. 73. North-Holland (1992)
11. Conradie, W., Goranko, V., Vakarelov, D.: Algorithmic correspondence and completeness in modal logic. I. The core algorithm SQEMA. *Logical Methods in Computer Science* **2**(1) (2006). [https://doi.org/10.2168/LMCS-2\(1:5\)2006](https://doi.org/10.2168/LMCS-2(1:5)2006)
12. Ditmarsch, H.v., van der Hoek, W., Kooi, B.: *Dynamic Epistemic Logic*. Springer (2008)
13. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: *Reasoning About Knowledge*. MIT Press (1995). <https://doi.org/10.7551/mitpress/5803.001.0001>
14. From, A.H.: Formalized soundness and completeness of epistemic logic. In: Silva, A., Wassermann, R., de Queiroz, R.J.G.B. (eds.) *Logic, Language, Information, and Computation - 27th International Workshop, WoLLIC 2021, Virtual Event, October 5-8, 2021, Proceedings. Lecture Notes in Computer Science*, vol. 13038, pp. 1–15. Springer (2021). https://doi.org/10.1007/978-3-030-88853-4_1

15. From, A.H.: Synthetic Completeness for a Terminating Seligman-Style Tableau System. In: de'Liguoro, U., Berardi, S., Altenkirch, T. (eds.) 26th International Conference on Types for Proofs and Programs (TYPES 2020). Leibniz International Proceedings in Informatics (LIPIcs), vol. 188, pp. 5:1–5:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2021). <https://doi.org/10.4230/LIPIcs.TYPES.2020.5>, <https://drops.dagstuhl.de/opus/volltexte/2021/13884>
16. From, A.H.: Epistemic logic: Completeness of modal logics. Archive of Formal Proofs (2018), https://devel.isa-afp.org/entries/Epistemic_Logic.html, Formal proof development
17. From, A.H.: Public announcement logic. Archive of Formal Proofs (Jun 2021), https://devel.isa-afp.org/entries/Public_Announcement_Logic.html, Formal proof development
18. From, A.H., Jensen, A.B., Villadsen, J.: Formalized soundness and completeness of epistemic logic (2021), <https://lamassr.github.io/papers/Formalized-Soundness.pdf>, extended abstract. International Workshop on Logical Aspects in Multi-Agent Systems and Strategic Reasoning (LAMAS & SR)
19. Guzman, L.P.G.: Stalnaker's epistemic logic. Archive of Formal Proofs (September 2022), https://devel.isa-afp.org/entries/Stalnaker_Logic.html, Formal proof development
20. Hagemeyer, C.: Formalizing intuitionistic epistemic logic in Coq (2021), <https://www.ps.uni-saarland.de/~hagemeyer/bachelor.php>, BSc thesis.
21. Kunčar, O., Popescu, A.: From types to sets by local type definition in higher-order logic. *J. Autom. Reason.* **62**(2), 237–260 (2019). <https://doi.org/10.1007/s10817-018-9464-6>
22. Kądziołka, J.: Solution to the xkcd Blue Eyes puzzle. Archive of Formal Proofs (2021), https://isa-afp.org/entries/Blue_Eyes.html, Formal proof development
23. Li, J.: Formalization of PAL-S5 in proof assistant. *CoRR abs/2012.09388* (2020), <https://arxiv.org/abs/2012.09388>
24. Maggesi, M., Brogi, C.P.: A formal proof of modal completeness for provability logic. In: Cohen, L., Kaliszyk, C. (eds.) 12th International Conference on Interactive Theorem Proving, ITP 2021, June 29 to July 1, 2021, Rome, Italy (Virtual Conference). *LIPIcs*, vol. 193, pp. 26:1–26:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). <https://doi.org/10.4230/LIPIcs.ITP.2021.26>
25. Meyer, J.J.C., Hoek, W.v.d.: *Epistemic Logic for AI and Computer Science*. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press (1995). <https://doi.org/10.1017/CBO9780511569852>
26. Neeley, P.: Results in modal and dynamic epistemic logic: A formalization in Lean (2021), <https://leanprover-community.github.io/lt2021/slides/paula-LeanTogether2021.pdf>, slides.
27. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)
28. Smullyan, R.M.: *First-order logic*. Dover Publications (1995)
29. Villadsen, J., From, A.H., Jensen, A.B., Schlichtkrull, A.: Interactive theorem proving for logic and information. In: Loukanova, R. (ed.) *Natural Language Processing in Artificial Intelligence — NLPinAI 2021*. pp. 25–48. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-030-90138-7_2
30. Wang, Y., Cao, Q.: On axiomatizations of public announcement logic. *Synth.* **190**(Supplement-1), 103–134 (2013). <https://doi.org/10.1007/s11229-012-0233-5>

31. Wu, M., Goré, R.: Verified Decision Procedures for Modal Logics. In: Harrison, J., O’Leary, J., Tolmach, A. (eds.) 10th International Conference on Interactive Theorem Proving (ITP 2019). Leibniz International Proceedings in Informatics (LIPIcs), vol. 141, pp. 31:1–31:19. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2019). <https://doi.org/10.4230/LIPIcs.ITP.2019.31>, <http://drops.dagstuhl.de/opus/volltexte/2019/11086>
32. Xiong, Z., Ågotnes, T., Zhang, Y.: The logic of secrets. In: LAMAS 2020 - 10th Workshop on Logical Aspects of Multi-Agent Systems (2020)

CHAPTER 7

Aesop: White-Box Best-First Proof Search for Lean

Jannis and I received a distinguished paper award for this work.

Origin

Jannis Limperg and Asta Halkjær From. “Aesop: White-Box Best-First Proof Search for Lean”. In: *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2023, Boston, MA, USA, January 16-17, 2023*. Ed. by Robbert Krebbers et al. ACM, 2023, pp. 253–266. DOI: [10.1145/3573105.3575671](https://doi.org/10.1145/3573105.3575671).

Aesop: White-Box Best-First Proof Search for Lean

Jannis Limperg

Vrije Universiteit Amsterdam
Department of Computer Science
Amsterdam, The Netherlands
j.b.limperg@vu.nl

Asta Halkjær From

Technical University of Denmark
DTU Compute
Kongens Lyngby, Denmark
ahfrom@dtu.dk

Abstract

We present Aesop, a proof search tactic for the Lean 4 interactive theorem prover. Aesop performs a tree-based search over a user-specified set of proof rules. It supports safe and unsafe rules and uses a best-first search strategy with customisable prioritisation. Aesop also allows users to register custom normalisation rules and integrates Lean’s simplifier to support equational reasoning. Many details of Aesop’s search procedure are designed to make it a white-box proof automation tactic, meaning that users should be able to easily predict how their rules will be applied, and thus how powerful and fast their Aesop invocations will be.

Since we use a best-first search strategy, it is not obvious how to handle metavariables which appear in multiple goals. The most common strategy for dealing with metavariables relies on backtracking and is therefore not suitable for best-first search. We give an algorithm which addresses this issue. The algorithm works with any search strategy, is independent of the underlying logic and makes few assumptions about how rules interact with metavariables. We conjecture that with a fair search strategy, the algorithm is as complete as the given set of rules allows.

CCS Concepts: • **Mathematics of computing** → *Mathematical software*; • **Theory of computation** → *Type theory*; *Logic and verification*; *Automated reasoning*.

Keywords: proof search, tactic, Lean, interactive theorem proving, deductive verification, type theory

ACM Reference Format:

Jannis Limperg and Asta Halkjær From. 2023. Aesop: White-Box Best-First Proof Search for Lean. In *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP ’23)*, January 16–17, 2023, Boston, MA, USA. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3573105.3575671>



This work is licensed under a Creative Commons Attribution 4.0 International License.

CPP ’23, January 16–17, 2023, Boston, MA, USA

© 2023 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-0026-2/23/01.

<https://doi.org/10.1145/3573105.3575671>

1 Introduction

One of the biggest barriers to a more widespread adoption of interactive theorem provers is the tedium of proving lemmas which are entirely obvious to the human eye. The provers force us to explicitly demonstrate that $n * 2$ is even, that $[z, y, x]$ is a permutation of $[x, y, z]$ or that a homomorphism of groups is also a homomorphism of the underlying semi-groups. This adds substantially to the cost of using theorem provers, which is still too high for many applications.

To help address this issue, we present Aesop (Automated Extensible Search for Obvious Proofs), a new proof search tactic for the upcoming version 4 of the Lean theorem prover [7]. In essence, Aesop is a tree-based search procedure which operates on a user-specified set of rules. The rules are arbitrary Lean tactics which, given a goal, either succeed — generating zero or more subgoals — or fail. Aesop applies these rules to the initial goal, then to the subgoals, etc., to build a search tree. On top of this basic setup, we provide the following features:

- Aesop uses a best-first search strategy, prioritising more promising rules (and their subgoals) over less promising ones. Which rules are considered promising is specified by the users themselves, using a simple prioritisation mechanism.
- Aesop distinguishes between safe rules, which are applied eagerly without backtracking, and unsafe rules, which may be backtracked. Safe rules are efficient since the goals to which they apply never need to be revisited.
- Aesop introduces a normalisation phase in which special normalisation rules are applied in a fixpoint loop to normalise the goal, before any other rules are applied. We use normalisation to establish invariants which the subsequent rules can rely on. For example, we split hypotheses of the form $P_1 \wedge \dots \wedge P_n$ into separate hypotheses P_i , establishing the invariant that no hypothesis is a conjunction.
- The normalisation phase includes an invocation of Lean’s simplifier, which performs rewriting with user-specified, possibly conditional equations. This allows us to benefit from the large collection of simplification rules which are typically defined by Lean projects.
- With best-first search, it is not obvious how to deal with metavariables which appear in multiple goals. In search procedures based on backtracking, such as

depth-first search, if a metavariable assignment turns out to be wrong, we can simply backtrack it and try a different one (assuming that the theorem prover's data structures efficiently support this). By contrast, a best-first algorithm must be able to consider multiple assignments in parallel. To address this issue, we present a new algorithm which handles metavariables and is independent of the search strategy.

Users of Isabelle's auto [18, 20] will recognise some of these features. More broadly, Aesop stands in the tradition of *white-box* proof automation tools, which also include Coq's (e)auto, PVS's grind [5] and ACL2's waterfall [11]. White-box tools require users to curate a set of rules which the tool applies. In return, users gain control over the power-performance tradeoff: many explosive rules make the automation stronger but slower; few conservative rules make it weaker but faster. Due to their customisability, white-box tools can also serve as a foundation for domain-specific automation, using domain-specific rule sets.

In contrast, *black-box* or *push-button* tools such as hammer [3], which invoke external automated theorem provers to find proofs, or machine learning systems which write a tactic script [2, 8, 10, 13], aim to operate with little or no user interaction. This makes them very convenient when they succeed, but the complex algorithms which deliver high success rates can be brittle. Sometimes a minor reformulation makes the difference between finding or not finding a proof. When a black-box tool does not find a proof or is slow to find one, it is often unclear how to improve the tool's performance.

White-box and black-box tools thus have complementary strengths and weaknesses, and so we believe it is worthwhile to explore both approaches. Aesop is an attempt to move far to the white-box end of the spectrum while retaining some of the useful features of Isabelle's auto and other systems. This is why we choose tree-based search as a base: it is easy to understand and close to interactive proof, which helps users predict how their rules will affect the search. We choose best-first search with customisable prioritisation (rather than some opaque heuristic) to give users more control over Aesop's performance. And we introduce fixpoint-based normalisation as an intuitive and reliable way to establish invariants. Taken together, these features should enable users to design effective and reasonably efficient rule sets for many domains.

Aesop is available as a Lean 4 library.¹ The specific version described here is available as a supplement to this paper.²

2 Best-First Proof Search

At its core, Aesop performs a tree-based, best-first proof search. This approach is independent of the underlying logic,

so it could also be used as a proof method for, say, first-order or higher-order logic, though we will use the notation of dependent type theory for examples. For now, we assume that goals do not contain metavariables, which simplifies the algorithm considerably.

2.1 Goals and Rules

We assume a set of *goals* given by the underlying logic. These could, for example, be first-order sequents or higher-order formulas. In Lean, they are structures of the form $\vec{h} : \vec{T} \vdash U$, where \vec{h} is a list of hypotheses with types \vec{T} and U is a type. Each hypothesis may depend on earlier hypotheses (so \vec{h} is a telescope) and U may depend on all hypotheses. We call U the goal's *target*.

We also assume a finite set of *rules*, which are partial functions that map a goal to a finite set of goals. When a goal is in the domain of a rule, we say that the rule is *applicable* to the goal. In the Aesop implementation, rules are arbitrary tactics.

When applied to a goal G , a rule produces a set of subgoals G_1, \dots, G_n . Rules should be *provability-reflecting*, meaning that if the subgoals G_i are provable in the underlying logic, then the initial goal G is also provable. For instance, an \wedge -introduction rule would map the goal $\Gamma \vdash P \wedge Q$ to the set $\{\Gamma \vdash P, \Gamma \vdash Q\}$. If a rule generates no subgoals, it proves the goal outright.

2.2 Search Tree

Aesop's central data structure is a search tree containing two alternating kinds of nodes: *goal nodes* and *rule application* ('*rapp*') nodes. The children of a goal node are rapp nodes representing rules which have been applied to the goal. The children of a rapp node are goal nodes representing the subgoals generated by the rule. For example, the goal $\Gamma \vdash P \wedge Q$ could have a child rapp for \wedge -introduction with two subgoals $\Gamma \vdash P$ and $\Gamma \vdash Q$.

At any point during the search, a node (goal or rapp) is in one of three states:

- *proved*: the node is proved. For a goal node, this means that *at least one* of its child rapps is proved. For a rapp node, it means that *all* of its child goals are proved. So sibling goal nodes are implicitly conjoined and sibling rapp nodes are implicitly disjoint, making the tree an AND/OR tree.
- *stuck*: the node cannot be proved with the given rules. For a goal node, this means that (a) all rules which can be applied to the goal have been applied and (b) *all* resulting child rapps are stuck. For a rapp node, it means that *at least one* of its child goals is stuck.
- *unknown*: the node is neither proved nor stuck.

The state of a node matters only insofar as it is necessary to determine the state of its parent node, then the parent's parent, etc., until we ultimately learn whether the root goal is

¹<https://github.com/JLimperg/aesop>

²<https://doi.org/10.5281/zenodo.7424818>

proved or stuck. This means that nodes can become *irrelevant* during the search. For example, if a goal $\vdash P \vee Q$ has child rapps for left or-introduction (with subgoal $\vdash P$) and right or-introduction (with subgoal $\vdash Q$), and $\vdash P$ is already proved, then $\vdash P \vee Q$ is also proved and there is no point in trying to prove $\vdash Q$. In general, we say that a node is irrelevant if at least one of its ancestors, including the node itself, is already proved or stuck. Incidentally, when the search terminates successfully, the root goal becomes proved and therefore, by our definition, irrelevant. So ‘irrelevant’ means ‘irrelevant for the rest of the search’, not ‘irrelevant for the proof’.

2.3 Search Algorithm

The search procedure starts with a search tree containing a single goal. It then enters a loop which, in each iteration, picks a goal node G with unknown state and a rule R which has not yet been applied to G . We then apply R to G . If this fails, we continue with the next rule and goal; if it succeeds, we add a rapp node for R with parent G and subgoals $R(G)$ to the tree. We call this operation the *expansion* of G along R . We exit the loop when the root goal becomes proved or stuck (or when one of several configurable limits, e.g. on the depth of the search tree, is reached).

Which goal is expanded first, and along which rule, is determined by a best-first search strategy. Usually, best-first search is realised by a heuristic which ranks goals and rules according to simple numeric properties, e.g. the size of a goal or the number of subgoals of a rule. This goes against Aesop’s white-box philosophy since the heuristics tend to be fixed (so users cannot easily change them) and opaque (so users cannot easily predict which goals will be prioritised). Instead, we implement a scheme whereby the rules carry a user-defined priority which is used to rank both goals and rules.

Specifically, Aesop users give each rule a *success probability* between 0% and 100%. This probability is a rough estimate of how useful a rule is, i.e. how likely it is to lead to a proof. For example, left and right \vee -introduction could each be given a success probability of 50%.

For rules whose success probability is less obvious, we have found it sufficient in practice to pick probabilities from a six-point scale: last resort (1%), low (25%), medium (50%), high (75%) and almost always (99%). The probabilities could also be determined by automated methods, for example by determining the actual success probability of each rule in an existing corpus of proofs. But while such automated tuning would perhaps improve Aesop’s overall performance in a larger library, it would also likely make some previously successful proofs fail, leading to maintenance challenges.

From the rules’ success probabilities we derive, for each goal in the search tree, a *priority* between 0% and 100%. The root goal has priority 100%. Then, whenever we apply a rule R to a goal G , the priority of the subgoals is the priority of G multiplied with the success probability of R . In each iteration

of the search loop, Aesop picks the highest-priority goal and expands it along the rule with the highest success probability. We could also allow rules to give different priorities to their subgoals, e.g. to prioritise goals which are known to quickly become unprovable if the initial goal is unprovable.

2.4 Safe and Unsafe Rules

So far, we have treated all rules as *unsafe*. An unsafe rule is one that does not necessarily preserve provability: when applied to a provable goal G , it may generate unprovable subgoals. For our search, this means that we must continue to expand both G and the subgoals.

However, in practice there are many rules which preserve provability and are therefore *safe*. For instance, \wedge -introduction is safe: to prove $\Gamma \vdash P \wedge Q$, it suffices to prove $\Gamma \vdash P$ and $\Gamma \vdash Q$. So after this rule has been applied, the original goal $\Gamma \vdash P \wedge Q$ does not need to be considered any more, shrinking the search space.

To take advantage of this insight, Aesop, like Isabelle’s auto, lets users mark rules as safe. To accommodate these safe rules, we split the expansion of a goal G into two phases. First, Aesop tries to apply all safe rules to G . If one of them succeeds, the resulting subgoals are added to the tree as usual. An unsafe rule would then re-insert G into the goal queue which we maintain throughout the search, to give other rules a chance to fire. For safe rules, we simply skip this step, ensuring that G is never expanded again. If no safe rules are applicable to G , Aesop moves to the second phase, in which unsafe rules are applied as explained above.

Safe rules are considered to have success probability 100%, so the subgoals of a safe rule receive the same priority as the parent goal. To control the order in which safe rules are tried, users can give them an integer priority. This order does not affect provability — assuming that rules marked as safe are actually safe — but it does affect the search performance. For instance, suppose we have, in addition to safe \wedge -introduction, a safe rule R that transforms a hypothesis $h : A$ into $h : B$. Then for the goal $h : A \vdash P \wedge Q$, it is better to apply R before \wedge -introduction; otherwise we would have to apply R twice.

In practice, the distinction between safe and unsafe rules can be tricky since safe rules must preserve provability relative to the whole rule set. When we mark \wedge -introduction as safe, we require the rest of the rule set to maintain the invariant that whenever we can prove $\Gamma \vdash P \wedge Q$, we can also prove $\Gamma \vdash P$ and $\Gamma \vdash Q$. This invariant can be violated, for example, by registering an unsafe rule R which proves $P \wedge Q$: the rule will never be applied since any goal $\Gamma \vdash P \wedge Q$ gets split by “safe” \wedge -introduction before R can be tried. So we must add more rules to ensure that $\Gamma \vdash P$ and $\Gamma \vdash Q$ can also be proved — or consider \wedge -introduction unsafe after all.

2.5 Normalisation

Besides safe and unsafe rules, Aesop introduces a third category of *normalisation* rules. These are rules which normalise

and simplify a goal, preparing it for further rule applications. Like safe rules, normalisation rules should preserve provability. Unlike safe rules, they must either prove the goal outright or return a single subgoal. For example, Aesop's default normalisation rules introduce assumptions, unfold certain definitions and prove trivial equations, reducing the goal $\vdash \forall f, \text{map } f \ [] = []$ first to $f \vdash [] = []$ and then to $f \vdash \text{True}$.

Normalisation rules are applied in yet another expansion phase, before the safe and unsafe phases. Like safe rules, they have a user-specified integer priority determining the order in which they are applied. Let R_1, \dots, R_n be the normalisation rules in this order. During the normalisation phase, Aesop then runs a loop which updates the goal. In each iteration, this loop tries to apply first R_1 , then, if it fails, R_2 , and so on. As soon as one of the R_i succeeds, the goal is set to the subgoal generated by R_i and the loop restarts. (If R_i produces no subgoal, the goal is proved and we are done.) If all the R_i fail, the loop ends. Compared with a simpler fixpoint loop which executes $R_1, \dots, R_n, R_1, \dots$ until all the R_i fail, this method has the advantage that the order of rules is always respected, so on each intermediate goal R_1 is tried before R_2 .

Like safe rules, normalisation rules must be chosen carefully to ensure that they preserve provability relative to the whole rule set. If we, for example, rewrite with the unfolding rule $[x] ++ xs = x :: xs$ during normalisation, rules about concatenation no longer apply to the normalised goal. If this is not desired, the unfolding is better performed as an unsafe rule or added as a local rule when needed. A common pattern is to register unfolding equations as normalisation rules while we prove facts about the respective definition (which almost always requires unfolding) and then remove them again for the remainder of the library.

2.6 Safe Goals

When Aesop fails to prove a goal, it reports the *safe goals*. These are the goals that would remain if we were to run Aesop with only normalisation and safe rules. Since normalisation and safe rules are non-branching (meaning each goal expanded by such a rule has exactly one child rapp), applying them exhaustively results in a single set of safe goals.

The safe goals are interesting because they indicate how much progress Aesop has made in the safe, non-branching part of its search. A typical Aesop proof workflow looks like this:

- Run Aesop on a goal G . If this proves the goal, we are done. Otherwise Aesop produces safe goals G_1, \dots, G_n .
- Manually perform some proof steps on each safe goal G_i , producing a goal G'_i .
- For each G'_i , apply this workflow recursively.

Once the proof is complete, we collect the manual steps and turn them into Aesop rules. This allows Aesop to prove

the initial goal G fully automatically — and hopefully other, similar goals as well.

To report the safe goals, we must address one minor complication. It is possible for the search to terminate before all safe goals have been generated. For example, suppose we register \wedge -introduction as a safe rule and search for a proof of the goal $\perp \wedge (P \wedge Q)$. Then the safe goals are \perp , P and Q . But Aesop may terminate after the first \wedge -introduction, realising that the goal \perp cannot be proved, without ever applying the second \wedge -introduction to $P \wedge Q$. So it would wrongly report \perp and $P \wedge Q$ as safe goals. Hence we must expand all relevant safe rules (here: \wedge -introduction on $P \wedge Q$) before computing the safe goals.

2.7 Multi-Rules

It is sometimes useful for a rule to add multiple rapps at once. For example, we will shortly see a rule which tries to apply the constructors of an inductive type. If more than one constructor can be applied, it is more natural (and slightly faster) to let the rule add one rapp per applicable constructor, rather than making each constructor a separate rule. We call such rules *multi-rules*.

Unsafe multi-rules are a straightforward generalisation of unsafe regular rules and require almost no changes to the search procedure. Safe and normalisation multi-rules are trickier. Normalisation multi-rules are not allowed at all since normalisation cannot branch. Safe multi-rules could be allowed, but their behaviour would be unintuitive: the whole *raison d'être* of safe rules is that they, too, do not branch. So we also forbid safe multi-rules.

The prohibition of safe and normalisation multi-rules is enforced dynamically, meaning that users may register safe and normalisation multi-rules but they fail if they actually generate multiple rapps. This is convenient because, for example, a rule that applies the constructors of an inductive family can be perfectly safe if for any given goal at most one constructor is applicable, which is the case for many inductive predicates and relations. Such multi-rules are effectively non-branching, so we should not ban them outright.

3 Best-First Proof Search in Lean

We now instantiate our best-first search framework to obtain a practical proof method for Lean.

3.1 Rule Builders

Aesop rules are arbitrary tactics, but it would be highly inconvenient if users had to write a tactic whenever they want to add, say, a lemma as a rule. We therefore provide several *rule builders* which register theorems, definitions or types as rules. Rules are registered either locally, i.e. for a single Aesop invocation, or globally in a rule set. Rule sets are collections of rules which can be activated or deactivated for

each Aesop invocation. The distinguished default rule set is activated by default.

3.1.1 apply. Given a term f of type $\forall \vec{x} : \vec{T}, P \vec{x}$, the apply builder creates a rule which applies f to goals $\Gamma \vdash P \vec{y}$. The arguments \vec{x} are either inferred (by unification or type-class search) or become subgoals.

When we write $P \vec{x}$, P is an arbitrary type-valued function (e.g. $\lambda x y, x = y + 1$), so $P \vec{x}$ is essentially an arbitrary type-valued term involving the variables \vec{x} . However, due to the inherent limitations of higher-order unification, our apply builder cannot support all functions P ; it uses the same heuristics as Lean’s apply tactic to support a useful subset. Similar caveats also apply to some of the following rule builders.

3.1.2 constructors. The constructors builder creates a rule which applies the constructors of an inductive type I . The rule has the same effect as if each constructor of I had been added as an apply rule, except that these apply rules are combined into one multi-rule.

3.1.3 forward. Given a term $f : \forall \vec{x} : \vec{T}, P \vec{x}$, the forward builder creates a rule which performs forward reasoning with f . This means that whenever a goal’s local context contains hypotheses $\vec{h} : \vec{T}$, the rule adds a new hypothesis $h' : P \vec{h}$. For example, the left \wedge -elimination lemma $\forall A B, A \wedge B \rightarrow A$, when used as a forward rule, reduces the goal $h : A \wedge B \vdash T$ to $h : A \wedge B, h' : A \vdash T$. If there are multiple sets of hypotheses with types \vec{T} , one new hypothesis is added for each set.

More generally, users can partition the arguments \vec{x} into *immediate* arguments $\vec{a} : \vec{A}$ and *non-immediate* arguments $\vec{b} : \vec{B}$. Then, Aesop searches only for hypotheses $\vec{h} : \vec{A}$ corresponding to the immediate arguments and, if successful, adds a hypothesis of type $\forall \vec{b} : \vec{B}, P \vec{h} \vec{b}$. (This notation suggests that the immediate arguments must precede the non-immediate ones, but in fact they can be interleaved freely.) So the immediate arguments must be “immediately available” as hypotheses while the non-immediate ones remain premises to be proved later. By default — and in our example above — all arguments which cannot be inferred are considered immediate.

In the example, the left \wedge -elimination rule is again applicable to the subgoals it generated. This is a general issue with forward rules: when a rule applies to a set of hypotheses \vec{h} , the subgoals still contain \vec{h} , so the rule is still applicable. To prevent this sort of looping, whenever a forward rule tries to add a hypothesis $h : T$, we check whether any forward rule that was applied earlier on this branch of the search tree already added a hypothesis $h' : T$. If so, the new hypothesis is not added and the rule fails.

There is also a variant of forward, destruct, which removes any hypotheses that matched the immediate arguments. If we use left \wedge -elimination as a destruct rule, it

reduces the goal $h : A \wedge B \vdash T$ to $h : A \vdash T$. Since the matched hypotheses are removed, destruct rules do not generally apply to their own subgoals, so there is no need to prevent cycles.

3.1.4 cases. Given an inductive family I with arguments (parameters and indices) $\vec{x} : \vec{T}$, the cases builder creates a rule which performs case analysis on any hypothesis $h : I \vec{x}$. For example, the cases rule for Or, the inductive type behind the notation $P \vee Q$, reduces the goal $h : P \vee Q \vdash T$ to two subgoals $h : P \vdash T$ and $h : Q \vdash T$. To perform this case analysis, we use Lean’s built-in cases tactic, which uses the standard elimination principle for I .

To perform case analysis according to a non-standard elimination principle, we can use the *view pattern* [15]: define a data type J whose constructors correspond to the desired cases, register a function $f : I \rightarrow J$ as a destruct rule and register a cases rule for J . With this setup, the goal $h : I \vdash T$ is first reduced to $h : J \vdash T$ and then h is split into the desired cases.

Like forward rules, cases rules for recursive types, such as lists or trees, can loop. If we register a cases rule for the List type, the goal $l : \text{List } \alpha \vdash P l$ is split into two goals $\vdash P []$ and $a : \alpha, l : \text{List } \alpha \vdash P (a :: l)$ and the cases rule is again applicable to the second goal.

One solution for this problem is to register the cases rule as an unsafe rule with very low priority. Aesop then uses it only as a last resort. This method is simple and effective, but it is problematic if Aesop does not find a proof: once there are no other rules left to apply, the cases rule is, as before, applied ad infinitum. This sort of cases rule is therefore only suitable as an ad hoc rule.

To support global cases rules as well, we provide a variant of the cases builder which avoids looping in some common cases. Consider the inductive predicate $\text{All } P \text{ xs}$, which encodes the proposition that all elements of the list xs satisfy the predicate P :

```
inductive All (P :  $\alpha \rightarrow \text{Prop}$ ) : List  $\alpha \rightarrow \text{Prop}$ 
| nil : All P []
| cons : P x  $\rightarrow$  All P xs  $\rightarrow$  All P (x :: xs)
```

When a goal contains a hypothesis $\text{All } P (x :: xs)$, we almost always want to perform a case split on this hypothesis, leaving us with two simpler hypotheses $P x$ and $\text{All } P xs$. Crucially, neither of these hypotheses has the same form as the initial one, so there is no infinite regress. To take advantage of this insight, Aesop allows users to annotate a cases rule with a *pattern* which restricts the hypotheses to which the rule is applicable. In our example, we would use the pattern $\text{All } _ (_ :: _)$ to ensure that an All hypothesis is only split if it refers to a non-empty list. Multiple patterns can also be given; the rule is then applied if at least one of the patterns matches a hypothesis.

We also considered a third solution to the infinite regress issue: we could stipulate that once a cases rule has been applied to a hypothesis, it cannot be applied again to the descendants of that hypothesis; or, more generally, that it can only be applied to the first n descendants. The vast majority of proofs should still work for, say, $n = 3$. Unfortunately the restriction is somewhat tricky to implement since Lean does not provide a reliable way to associate metadata with a hypothesis, but we want to support this in the future.

3.1.5 tactic. The last and most fundamental rule builder, *tactic*, allows users to register any tactic as a rule. The tactic can generate arbitrary subgoals (justified by a proof term that is later checked by Lean’s kernel). We only require that tactic rules either change the goal or fail, so they cannot be no-ops.

3.2 Simplifier Integration

Lean’s simplifier, which performs rewriting with a user-provided set of conditional rewrite rules, is used heavily in all big Lean projects. In particular, *mathlib* [4], a large library of formalised mathematics which contains most Lean code written to date, defines an extensive set of simplifier rules. To make Aesop practical, we should leverage this existing automation.

To that end, we integrate simplification into the normalisation process, adding a built-in normalisation rule which runs the simplifier on the entire goal (target type and hypotheses). This invocation of the simplifier uses the default global set of rewrite rules, plus a separate Aesop-specific rule set. Aesop users can add rules to this set by using a special *simp* rule builder.

An important detail of the simplifier integration concerns how we use local hypotheses. Lean’s simplifier can be configured to use them in two ways. First, local equations can be used as rewrite rules, transforming the goal $h : x = y \vdash P x$ into $h : x = y \vdash P y$. This can be dangerous since local equations are not necessarily oriented in a way that works well with other rules. For example, a rule that proves $P x$ may not fire any more. Worse, a rogue equation can easily make the simplifier loop.

Second, local hypotheses which are propositions (but not equations) can be rewritten to truth values, transforming the goal $h_1 : P, h_2 : \neg Q \vdash P \vee Q$ first into $\dots \vdash \top \vee \perp$ and then, via a global rewrite rule, into $\dots \vdash \top$. This functionality allows the simplifier to perform some propositional reasoning. In particular, conditional rewrite rules such as $P \rightarrow x = y$ are, by default, used only if the antecedent P simplifies to \top .

In practice we have found that, despite the danger of rewriting with local equations, letting the simplifier use local hypotheses substantially increases Aesop’s utility. We therefore enable this behaviour by default, but users can disable it for specific Aesop invocations.

3.3 Rule Indexing

So far we have been pretending that when a goal is expanded, we run all registered Aesop rules in order of priority. But Aesop is intended to be used with a large rule set, so this naive approach would be prohibitively slow. We therefore introduce a *rule index* which, given a goal G , efficiently determines a small subset of rules that may apply to G .

The index offers several *indexing schemes*. An indexing scheme determines, given a rule and a goal, whether the rule is potentially applicable to the goal. We currently implement three schemes:

- *Target*: the rule specifies a pattern expression T , which may contain holes. It is considered potentially applicable when the goal has the form $\Gamma \vdash U$ and U unifies with T . We use this scheme for *apply* rules.
- *Hypothesis*: the rule again specifies a pattern expression T . It is considered potentially applicable when the goal has the form $\Gamma, h : U, \Delta \vdash V$ and U unifies with T . We use this scheme for *cases* and *forward* rules. For forward rules, we take as the pattern T the last immediate argument of the rule, since later arguments are often more specific than earlier ones.
- *Disjunction*: the rule specifies a list of indexing schemes. It is considered potentially applicable when any of the schemes match the goal. We use disjunctive indexing for *constructors* rules (one by-target scheme for each constructor) and for *cases* rules with multiple patterns (one by-hypothesis scheme for each pattern).

The first two schemes are implemented by one discrimination tree each. A discrimination tree is a trie-like data structure that maps expressions T to arbitrary data (here: rules) and enables efficient retrieval of all values in the map whose key T may unify with a query expression U [16]. (In Lean 4, discrimination trees are also used to index typeclass instances and simplifier lemmas.) For the by-target scheme, we query the discrimination tree with the goal’s target. For the by-hypothesis scheme, we query the discrimination tree once per hypothesis. The disjunction scheme is implemented by inserting the rule into the relevant discrimination trees multiple times with different keys.

Most rule builders have a natural indexing scheme. The exception is the *tactic* builder, which wraps arbitrary tactics. For *tactic* rules, users can specify a suitable indexing scheme themselves, if there is one.

When an indexed rule matches a goal, we communicate to the rule the set of *match locations*. Each match location is either the goal’s target or a specific hypothesis. Using the match locations, a cases rule, for example, does not need to scan the hypotheses of the goal to find those of the right type. Instead, it can immediately focus on the hypotheses that were matched by its indexing scheme.

Like other Lean proof methods, notably the simplifier, our indexing schemes perform unification up to *reducible*

computation. Each Lean definition is annotated with one of several *transparency modes*, which govern how eagerly the definition is unfolded during unification. Most definitions have default transparency and are not unfolded by the unification methods used by automation tactics; only those with reducible transparency are. Aesop’s indexing follows this scheme. This, along with a convention that only non-recursive definitions are tagged as reducible, ensures that discrimination tree indexing does not miss any possible matches (with rare exceptions), but it also weakens certain rules. For example, a rule which proves the goal $a :: as = b :: bs$ could also prove $[a] ++ as = [b] ++ bs$ since the two goals unify once we unfold the list concatenation operator $++$. But $++$ has default transparency, so our index does not unfold it and the rule is never tried on the second goal. To compensate, we could register a simplification rule which normalises $[a] ++ as$ to $a :: as$.

3.4 Default Rules

Aesop’s default rules perform uncontroversial reasoning steps, mostly pertaining to the logical connectives. Hypotheses $h : P \wedge Q$ are eliminated during normalisation, yielding separate hypotheses $h_1 : P$ and $h_2 : Q$, and similar for products $P \times Q$. Goals of the form $\Gamma \vdash P \wedge Q$ are reduced to $\Gamma \vdash P$ and $\Gamma \vdash Q$ by registering \wedge -introduction as a low-priority safe rule. For sum-like types such as disjunction, the respective elimination rule, which splits the goal into two subgoals, is safe with low priority. The respective introduction rules, which select one branch of the sum, are unsafe with 50% success probability.

Universally quantified goals $\Gamma \vdash \forall \vec{x} : \vec{T}, P \vec{x}$ are normalised to $\Gamma, \vec{x} : \vec{T} \vdash P \vec{x}$. When a goal with target $P \vec{x}$ contains a hypothesis $h : \forall \vec{y}, P \vec{y}$, h is applied as an unsafe rule. We give this rule 75% success probability, assuming that when a local hypothesis can be applied, it is usually a good idea to do so. In the special case where h has no premises, it is applied safely and proves the goal.

Existentially quantified hypotheses are split eagerly. For goals with an existentially quantified target, we register \exists -introduction, which creates a metavariable for the witness, as an unsafe rule. (See the next section for details on how we handle metavariables during the search.) It is important that this rule is unsafe because the goal’s context determines which hypotheses can be used in the assignment of the witness metavariable. Thus, if we create this metavariable too eagerly, hypotheses which are added afterwards, e.g. by an unsafe cases rule, cannot be used in the metavariable’s assignment.

Goals whose target is an equation $t = u$ are proved by reflexivity if t and u are already definitionally equal. Equational hypotheses $h : t = u$ are by default rewritten left-to-right during normalisation, as described in Sec. 3.2. In the special case where t is a local hypothesis, we substitute u for t

everywhere in the goal and remove both t and the equation h . This is safe since t , having been removed from the goal, can never appear in a subgoal again, so the equation h has become superfluous. Symmetrically, if u is a local hypothesis, we substitute t for u and remove u and h .

Goals of the form $\Gamma \vdash P \leftrightarrow Q$ are split into subgoals $\Gamma \vdash P \rightarrow Q$ and $\Gamma \vdash Q \rightarrow P$. Hypotheses of type $P \leftrightarrow Q$ are treated like equalities $P = Q$ by appealing to propositional extensionality, an axiom which Lean uses pervasively.

The only default rule which does not pertain to logical connectives (apart from some rules for technicalities) is a low-priority safe case-splitting rule. If a goal’s target contains an expression of the form $\text{if } t \text{ then } \dots \text{ else } \dots$ or $\text{match } t \text{ with } \dots$, then this rule performs a case split on t , producing a simpler goal for each possible case. A similar rule applies to hypotheses containing if or match expressions, with even lower priority.

Designating so many default rules as safe can lead to unintuitive results. For example, as mentioned in Sec. 2.4, splitting a goal with target $P \wedge Q$ into goals with targets P and Q is unsafe if the rule set contains an unsafe rule which proves $P \wedge Q$, but not rules which prove P and Q . However, we believe it would be worse to make these rules unsafe, both for performance and because the printing of safe goals, which is an important debugging aid, becomes less useful if our safe rules are overly conservative.

4 Best-First Proof Search with Metavariables

We now extend the search algorithm to support goals containing metavariables. A *metavariable* (sometimes called schematic variable, existential variable or just free variable) is an expression which represents a typed term to be determined later. For instance, the goal $?m > 0 \wedge ?m < 3$, where $?m$ is a metavariable of type \mathbb{N} , can be proved if $?m$ is assigned the value 1 ($?m := 1$) or if $?m$ is assigned the value 2.

In interactive proofs, metavariables are created when we use a tactic without specifying all relevant information. A typical example is \exists -introduction, which reduces a goal $\exists w, P w$ to $P ?w$, leaving the witness $?w$ to be determined later. Of course, we can also specify the witness up front, but using a metavariable can be convenient: perhaps we can reduce $P ?w$ to $?w = 0$, in which case we can appeal to the reflexivity of equality to prove the goal, assigning $?w := 0$ as a side-effect.

Mirroring the interactive use of metavariables, Aesop allows rules like \exists -introduction to create and assign metavariables. This way of handling existential quantification is obviously incomplete since only witness terms induced by a subsequent rule application are considered. But it is also cheap, reasonably effective and familiar to users from their interactive proofs.

Another important class of rules which create metavariables are transitivity rules, which reduce a goal $x \leq z$ to subgoals $x \leq ?y$ and $?y \leq z$. These rules illustrate the main challenge of dealing with metavariables: they couple goals. A metavariable represents the same term everywhere it appears. So when we apply, say, reflexivity to the first subgoal $x \leq ?y$, we assign $?y := x$ as a side-effect and the second subgoal becomes $x \leq z$. How we prove the first subgoal now determines how, and indeed whether, we can prove the second.

This is a problem because our search procedure assumes that goals are independent. When we apply the reflexivity rule to $x \leq ?y$, we do not intend to commit to the resulting assignment $?y := x$ for the remainder of the search. We may, after all, have an assumption $x \leq a$ in the context which induces another instance of the second subgoal: $a \leq z$ with $?y := a$. And since we are doing best-first search, we may visit the second subgoal first and apply a rule which assigns $?y := b$, changing the first subgoal to $x \leq b$. Our search procedure should consider all these possibilities.

If we were to use a search strategy based on backtracking, such as depth-first search, this would be easy. We would merely have to ensure that when a rule application is backtracked, any metavariable assignments it has performed are erased. But for best- or breadth-first search, all assignments must be considered in parallel. So for the above example, the search tree must reflect the fact that we may prove any of the sets of goals $\{x \leq x, x \leq z\}$, $\{x \leq a, a \leq z\}$ and $\{x \leq b, b \leq z\}$. In the remainder of this section, we present an extension of our search algorithm which achieves just that.

4.1 Overview

To see the core issue with metavariables, suppose we have a rapp R with subgoals $G[?x]$ and $H[?x]$ that depend on $?x$. We say that G and H are *m-coupled* ('metavariable-coupled') since they share a metavariable $?x$ such that if G is proved for some assignment $?x := a$, then we must also prove $H[?x := a]$ (i.e. H with a substituted for $?x$) to get a proof of the parent rapp R . We can view $H[?x := a]$ as a "virtual subgoal" of the rule which proves G .

Our solution for this issue is simply to make the virtual subgoal an actual subgoal: when a rule S is applied to G and assigns $?x := a$, then $H[?x := a]$ is added as an additional subgoal of the S rapp. We call this additional subgoal an *m-copy* of H . Symmetrically, when a rule T is applied to H and assigns $?x := b$, then $G[?x := b]$ is added as an additional subgoal of T .

More generally, it is not only the siblings of G which may need copying. Suppose we first apply a rule to G which does not interact with $?x$ and produces a goal $G'[?x]$. We then apply R' to G' , assigning $?x := a$. Then $H[?x := a]$ still needs to be copied even though it is not a sibling of G' . Accordingly, we expand our notion of m-coupled goals. Let G_1, \dots, G_n be

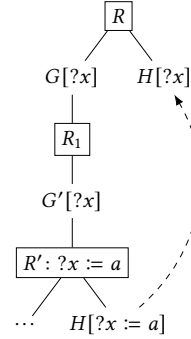


Figure 1. Copying of m-coupled nodes

the path from G' (so $G_1 = G'$) towards the root of the tree such that G_n is the first goal in which $?x$ appears. Each goal I which depends on $?x$ and which is a sibling of a goal G_i on the path is m-coupled to G' and is therefore copied.

Fig. 1 visualises this example, showing the incomplete search tree with root R . Here and in the next figure, rapp nodes are displayed as rectangles and are annotated with the metavariables they assign. Goal nodes are annotated with the metavariables they depend on, including the metavariables' assignments. Dashed arrows point from each copied goal to the goal it was copied from.

Once we perform copying, we must also modify our notion of when a goal is proved. Suppose we have three subgoals of a rule R : $G_1[?x]$, $G_2[?x, ?y]$ and $G_3[?y]$. If we prove G_3 , then $?y$ must be assigned somewhere in this proof, say to $?y := a$. At this point, G_2 is copied since it also depends on $?y$, so the proof of G_3 contains a proof of the goal $G_2[?x, ?y := a]$. This proof, in turn, must assign $?x$, say to $?x := b$, at which point G_1 is copied, so the proof of G_3 also contains a proof of $G_1[?x := b]$. In general, any goal that is m-coupled to G_3 must already be included in a proof of G_3 . To prove R , therefore, it suffices to prove G_3 (plus any other subgoals of R that are not m-coupled to G_3).

Fig. 2 visualises this example. Proved nodes are underlined. The dotted boxes around goals will become relevant shortly. We have added an additional subgoal of R , G_4 , which is not m-coupled to G_3 and therefore needs to be proved separately. To keep the figure simple, each goal is proved by a single rule application with one subgoal, but in general, there could be an entire subtree between, say, G_3 and R_1 . Moreover, the figure shows a proof attempt in which we happen to apply exactly those rules which lead to a proof. A less fortunate attempt would explore subtrees below the various goals before it finds the closing rapps R_3 and R_4 .

Our modified definition of when a goal is proved relies on a crucial assumption: when we apply a rule R to a goal $G[?x]$, then either R must assign $?x$ or at least one of the subgoals

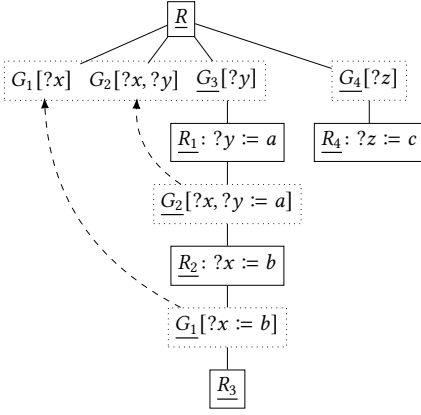


Figure 2. Proved nodes with copying

generated by R must also contain $?x$. Otherwise, we say that $?x$ has been *dropped*. If we were to allow dropped metavariables, a proof of $G[?x]$ would not necessarily have to assign $?x$ and an m-coupled sibling $H[?x]$ would not necessarily be proved. However, completely disallowing dropped metavariables turns out to be too strict for some applications, so we revisit this restriction in Sec. 4.6.

4.2 Search Tree

To support metavariables, we first augment the search tree to track some metavariable-related information. These data could also be computed on demand; we only cache them for efficiency.

In each goal node, we store the set of metavariables which the goal depends on. These are the metavariables which occur in the goal’s hypotheses and in its target type. We assume for simplicity that assigned metavariables are immediately substituted everywhere, so only unassigned metavariables can occur in a goal. Additionally, the metavariables which occur in the goal may in turn depend on other metavariables since the type or context of a metavariable may contain other metavariables. We collect these recursively. The recursion terminates since cyclic dependencies between metavariables are not allowed. Obtaining the metavariables which occur in an expression is cheap since Lean optimises the common case in which an expression does not contain any metavariables.

In each rapp node, we store two additional pieces of information. First, we store the metavariables created by the rule application, which are those metavariables which the reported subgoals depend on and which the initial goal does not depend on. Second, we store the metavariables assigned by the rule application, which are those metavariables which the initial goal depends on and which are assigned after the rule has been run. We thus assume that rules do not assign

metavariables which are not reachable from the initial goal, which is true for all (bug-free) Lean tactics.

We also need to keep track of which goals are m-coupled. This requires a more substantial augmentation of the search tree: we partition each rapp’s set of subgoals $\{G_1, \dots, G_n\}$ into *metavariable clusters*, which are, informally, sets of transitively m-coupled sibling goals. For example, suppose we have subgoals $G_1[?x]$, $G_2[?x, ?y]$, $G_3[?y]$ and $G_4[?z]$ as in Fig. 2. Then we partition these subgoals into metavariable clusters $\{G_1, G_2, G_3\}$ and $\{G_4\}$. Note that G_1 and G_3 do not have a metavariable in common, but they are still transitively m-coupled via G_2 . In the figure, metavariable clusters are indicated by dotted boxes around sets of goals, but all clusters except for one are trivial, containing only one goal each.

Formally, for two goals G and H we write $G \sim H$ if there is a metavariable on which both G and H depend. We define \approx as the transitive closure of \sim . Since \sim is already reflexive and symmetric, \approx is an equivalence relation. The metavariable clusters of a rapp are the equivalence classes of the rapp’s subgoals with respect to \approx .

We can view metavariable clusters as a third type of node in the tree. The children of a rapp are then metavariable clusters; the children of a metavariable cluster are the goals contained in it; and the children of a goal are (still) rapps. This view leads to a natural generalisation of the node states:

- *proved*: as before, a goal node is proved if *at least one* of its child rapps is proved; a rapp node is proved if *all* its children are proved. But the children of a rapp are now metavariable clusters, and a metavariable cluster is proved if *at least one* of its goals is proved. This is motivated by the observation we made above: if we have a metavariable cluster with goals $\{G_1, \dots, G_n\}$ and we prove some G_i , then all the G_j with $j \neq i$ must have been proved as part of the proof of G_i .
- *stuck*: as before, a goal node is stuck if *all* its child rapps are stuck and there are no more rules which could be applied to it; a rapp node is stuck if *at least one* of its children is stuck. A metavariable cluster is stuck if *all* its goals are stuck. This is because even if a goal $G[?x]$ is stuck, as long as some other goal H in the same metavariable cluster is non-stuck, it is still possible that the proof of H will discover a new assignment $?x := a$ and we can prove $G[?x := a]$.
- *unknown*: as before, a node is unknown if it is neither proved nor stuck.

The definition of irrelevance also remains unchanged: a node (which can now also be a metavariable cluster) is irrelevant if any of its ancestors, including the node itself, is proved or stuck.

When a search tree contains no metavariables, each goal is only m-coupled to itself, so there is one metavariable cluster per goal. The metavariable-free version of our algorithm

from Sec. 2 then emerges as a special case of the metavariable-encumbered one.

4.3 Copying

The search procedure with metavariables is largely the same as without metavariables. The only change is that when we add a rapp which assigns a metavariable, we must copy the m-coupled goals.

To see how, suppose we are adding a rapp R with parent goal $G[?x_1, \dots, ?x_n, ?\bar{y}]$ such that R assigns $?x_i := a_i$ for $1 \leq i \leq n$. Aesop then walks the path from G up the tree towards the root goal, stopping at the topmost rapp node R_m that creates any of the metavariables $?x_i$. (Thus the $?x_i$ can only appear in the subtree below R_m .) Let this path be $G_1, R_1, G_2, R_2, \dots, G_m, R_m$, where $G_1 = G$ and for each i , R_i is the parent rapp of G_i and G_{i+1} is the parent goal of R_i . Aesop then copies every sibling H of the G_i which depends on any of the $?x_j$, adding $H[?x_1 := a_1, \dots, ?x_n := a_n]$ as an additional subgoal of R .

However, there are two special cases in which it is not useful to copy a sibling goal H . First, H may be a copy of one of the G_i on the path. This means we are already in the subtree that will serve as a proof of G_i , so adding H as a subgoal would be pointless. Second, we may discover multiple goals H_1, \dots, H_k which are copies of the same original goal. In this case, we only need to copy one of them.

Note that we copy only the sibling goals themselves and not their subtrees. This means that any rules which were applied to the sibling goals must be re-applied to their copies. In general, this is necessary because the copied goals have different types and hypotheses (on account of the metavariable substitution we applied to them), so re-applying the rules may yield different results. But it is still somewhat inefficient. We discuss a possible solution to this issue in Sec. 4.7.

4.4 Interaction with Safe Rules

Most safe rules become unsafe if they assign metavariables. This applies even to such unassuming rules as proof by assumption. Suppose we have goals $h_1 : \alpha$, $h_2 : \beta \vdash ?x$ and $\vdash \beta \rightarrow ?x$. If we prove the first goal via h_1 , the second goal may well become unprovable. If we use h_2 instead, the second goal is trivial. So proof by assumption does not preserve provability in the presence of metavariables.

Accordingly, Aesop treats any safe rule that assigns a metavariable as unsafe. This means that when we expand a goal G , we first run the safe rules applicable to G , as usual. Whenever one of these rules assigns at least one metavariable, we do not add the rule to the search tree. Instead, we treat the rule as failed but store its result (subgoals and some metadata) in a list of *postponed* safe rapps. We then continue to apply the remaining safe rules. If one of them succeeds without assigning metavariables, we apply it directly and throw away the postponed rapps. Otherwise — if all safe rules either fail

or assign metavariables — we apply the unsafe rules as usual, but we also add the postponed rapps as unsafe rules with success probability 90%. When Aesop selects a postponed rapp to be applied as an unsafe rule, it does not re-execute the rule but simply adds its stored result to the search tree.

In principle, one could imagine situations in which a safe rule assigns metavariables in a safe manner and thus does not need to become unsafe. But in practice, we have yet to encounter such a situation.

4.5 Interaction with Normalisation Rules

For normalisation rules, we need to restrict metavariable assignments even more than for safe rules. Since normalisation rules must also be safe, we have the same issue as with safe rules. Additionally, normalisation rules are applied in a fixpoint loop, so there is no natural way to postpone a normalisation rule. So we simply forbid normalisation rules from creating or assigning metavariables.

This restriction is, for the most part, unobtrusive in practice, with one unfortunate exception. Our implementation of cases rules uses Lean's built-in cases tactic to perform case analysis. When a goal contains a metavariable, cases may replace this metavariable with a new one, which to Aesop looks as if an existing metavariable had been assigned and a new one created. We have not found a reliable way to detect this situation, so we currently do not allow cases normalisation rules (which could otherwise be used to, for example, split a hypothesis $h : A \wedge B$ into $h_1 : A$ and $h_2 : B$).

4.6 Synthesis of Dropped Metavariables

Recall that when a rule R is applied to a goal $G[?x]$, there must be at least one subgoal of R which depends on $?x$. If this is not the case, we say that $?x$ has been dropped, and so far we have disallowed dropped metavariables.

However, this restriction turns out to be too harsh. One application — Jesse Vogel's Duck tool³, which aims to use Aesop to find examples of structures with certain properties in algebraic geometry — provided this trivial test goal: under the assumption that the integers form a ring and that every ring R has a ring automorphism $\text{id} : \forall R : \text{Ring}, \text{RingHom } R R$, show

$$\exists R : \text{Ring}, \text{RingHom } R R.$$

To prove this goal, Aesop first applies \exists -introduction, obtaining the goal $\text{RingHom } ?R ?R$. It then tries to apply id , which proves the goal without assigning $?R$, so $?R$ is dropped. Since this is forbidden, the application of id fails and the goal cannot be proved.

To address this obvious deficiency, we must allow dropped metavariables. But at the same time, we must take care not to violate the conditions that led us to disallow them in the first place:

³<https://github.com/jessetvogel/duck>

- Dropped metavariables must be assigned eventually. This is necessary in dependent type theory since the type of a metavariable could be uninhabited. An unassigned metavariable of type T corresponds to an assumption that we can inhabit T . (The situation is different for logics in which all types are inhabited, such as the logic of Isabelle/HOL.)
- When we prove a goal $G[?x]$ and drop $?x$ in the process, we must ensure that related goals containing $?x$ are proved as well.

To address the first requirement, when a rule R is applied to a goal $G[?x]$ and drops $?x$, we add an additional subgoal to R which corresponds to $?x$. In our ring example, applying id proves the goal $\text{RingHom } ?R \ ?R$, but since $?R : \text{Ring}$ is dropped, we add an additional subgoal of type Ring , which is then proved by assumption.

To address the second requirement, we modify the procedure for copying metavariable-related goals such that it treats dropped metavariables as assigned for the purposes of copying. So if, in our example, we had an m -coupled goal, say $\text{RingHom } ?R \ \mathbb{Z}$, then the id application would copy this goal as an additional subgoal. Thus, the proof of the original goal, $\text{RingHom } ?R \ ?R$, again contains proofs for all m -coupled goals.

Importantly, whether a dropped metavariable appears in the subgoals of a rapp — and therefore whether a subgoal for it is added — is determined after copying. This ensures that a subgoal for a metavariable $?x$ is only created once we can no longer obtain an assignment from the proof of any goal in this branch of the search tree. If this were not the case, we could end up with a solution a for the subgoal $?x$ which is different from an assignment $?x := b$ performed by a later rapp .

4.7 Discussion

Our algorithm is conceptually attractive for two reasons. First, it is a strict and fairly simple generalisation of the algorithm without metavariables. Second, it is very general: it works for any search strategy and makes almost no assumptions about how rules interact with metavariables. We only require that rules limit their assignments to metavariables appearing in the goals to which the rules are applied.

The downside of this generality is some inefficiency. In particular, as mentioned in Sec. 4.3, our algorithm treats goals which are m -copies of each other as entirely independent, so a rule applied to one has no effect on the others. For an example of how this leads to inefficiency, suppose the goal $h : n < ?x \vdash A$ appears in the search tree. Then, during the search, we likely create a number of m -copies of this goal with different instantiations for $?x$. Now suppose we have a rule $R : B \rightarrow A$. When this rule is applied to one m -copy of the goal, we could recognise that R is independent

of the instantiation of $?x$ and therefore applies to every m -copy. As it stands, our algorithm does not take advantage of this optimisation opportunity. However, the optimisation is also valid only for certain rules. A rule which searches for contradictory hypotheses $n < 0$ (where n is a natural number) is not independent of the instantiation of $?x$ and therefore cannot be shared between m -copies of the goal.

We believe that despite its generality, our algorithm is as complete as possible, in the following sense. Suppose we use a fair search strategy, i.e. one which guarantees that every rule will eventually be applied to every goal. Now take a goal G that can be proved by applying a sequence of rules from the rule set, creating and assigning arbitrary metavariables in the process. Then, we conjecture, our algorithm will also find a proof of G . Intuitively, this is because our algorithm only adds to the search tree, so it is not possible to apply a rule in such a way that another rule cannot be applied any more. Thus, since we assume a fair search strategy, each rule in the proving sequence of rules is applied eventually (unless the goal to which it would be applied is already proved). We plan to prove this conjecture in future work.

5 Case Studies

As evidence that Aesop provides a reasonable level of automation, we present two case studies: one in which we prove a variety of basic theorems about lists and one in which we formalise a simple automated theorem prover for intuitionistic propositional logic. Both case studies are available in the supplement to this paper.⁴

Ideally, we would evaluate Aesop on a standardised benchmark such as the TPTP problem set [21]. But this is conceptually difficult: without an extensive rule set, Aesop is not expected to prove many theorems, and with an extensive rule set, we could game many benchmark problems by providing just the right rules. Perhaps as a result, there is currently no standard benchmark for white-box proof search tools.

5.1 Lists

As a first test of Aesop, we port some lemmas about lists from Lean 3 to Lean 4. We consider a file from `mathlib`, `data/list/basic.lean`. This file contains a large number of lemmas about basic list functions such as `length`, `append` and `reverse`, and about predicates such as `subset` and `membership`. We take the first 200 of these lemmas and port them to Lean 4.

Of the 200 lemmas, we exclude 16 which merely state definitional equations. (Such lemmas are used to register definitional equations with the simplifier.) For lemmas which reference notations or concepts that are not available in Lean 4, we either add the necessary definitions or, in 11 cases, exclude the lemma from our case study. Some of the remaining 173 lemmas are already proved in Lean 4, in which case we

⁴<https://doi.org/10.5281/zenodo.7424818>

re-prove them. If these lemmas are registered as global simplifier rules, we remove them first; otherwise Aesop's job would be a bit too easy.

Whenever we prove a lemma which makes a good global Aesop rule, we add the lemma to the global Aesop rule set. We also add a small number of lemmas about other concepts (injective/surjective/involutive functions and the Option type) which could sensibly be included in a library-wide Aesop rule set.

With this setup, Aesop proves 109 (63%) of the list lemmas outright. If we manually perform induction where necessary (which Aesop by design does not do), Aesop proves 163 (94%) of the lemmas. Specifically, for lemmas which require induction, we either add one or more calls to the induction tactic (after possibly unfolding some definitions and introducing hypotheses) or we write the lemma as a match statement, use a recursive call to prove the induction hypotheses and let Aesop do the rest. The latter is the most ergonomic way to perform functional induction in Lean.

Of the 10 lemmas Aesop cannot prove, 4 involve existentially quantified statements with non-trivial witnesses, e.g.

```

∀ (a : α) (l : List α), a ∈ l →
  ∃ (s t : List α), l = s ++ a :: t

```

Aesop's quantifier instantiation method, which relies solely on unification, is too weak to determine the proper witnesses for each case of the induction. The other 6 unsolved lemmas fail either because a lemma is missing from the library (2) or because Aesop's rule set misfires in specific situations (4).

Of the 163 lemmas Aesop (plus induction) can prove, 48 (29%) require local rules; the rest are solved using only global rules. By far the most common local rule, with 25 occurrences, is a low-priority unsafe rule which performs a case split on hypotheses of type List. Since each such case split produces another hypothesis of type List, this rule can loop, so it is not suitable as a global rule. But for lemmas which require such a case split, we can add the rule and due to its low priority, Aesop applies it only as a last resort. This makes sure that if a proof is found, it is found quickly.

Another notable local adjustment involves Aesop's simplifier integration. As we discussed in Sec. 3.2, Aesop by default rewrites with equations in the local context. This can be dangerous because such equations are not necessarily properly oriented. For example, a hypothesis of type $n = n + 0$ would, together with the global rule $n + 0 = n$, send the simplifier into a loop. In our case study, this happens two times; in both cases, Aesop succeeds once we disallow the use of hypotheses during simplification.

To get a broad idea of how fast Aesop is, we also ran a small benchmark involving this case study. For the benchmark, we prepared a version of the case study in which all lemmas are proved by hand. The proofs are written in the runtime-efficient style of mathlib (most proofs are translated from

Lean 3), meaning they involve no expensive tactics except the simplifier, which moreover is always given the exact set of lemmas it should simplify with. Thus, we believe that this hand-written version of the case study has close to optimal performance. We then compared the total time Lean takes to typecheck the hand-written version and the Aesop version of the case study, averaging over 10 runs each. On one particular machine, the hand-written version took on average 2.48 seconds to typecheck (min = 2.46, max = 2.50, $\sigma = 0.015$) whereas the Aesop version took 6.25 seconds (min = 6.17, max = 6.32, $\sigma = 0.045$). This means delegating all proofs to Aesop resulted in a slowdown of 2.53x. When run on other machines, the benchmark yielded slowdowns of 2.59x and 2.60x.

It is perhaps not surprising that Aesop is fairly successful in this case study: most of the lemmas we consider are very simple. But automating trivial goals about basic data structures is still an important part of making interactive theorem provers less onerous to use. And many lemmas which are straightforward consequences of facts known to Lean would not have had to be written if Aesop had been available at the time.

5.2 Propositional Sequent Calculus Prover

As a second test of Aesop, we have programmed a small prover for propositional logic [22] in Lean 4 and used Aesop to verify the soundness and completeness of both the prover and the sequent calculus proof system it is based on.

To that end, we first define the type Form of propositional formulas. Given an interpretation i of propositional variables Φ , the predicate $\text{Val} : \text{Form } \Phi \rightarrow \text{Prop}$ gives the truth value of a formula. Aesop proves, after manual induction over Form, that if i is decidable, then so is Val.

Satisfiability of formulas extends to satisfiability of sequents: a sequent with premises Γ and conclusions Δ is valid if, whenever all premises are true, at least one conclusion is true. Formally, $\text{All } (\text{Val } i) \Gamma$ implies $\text{Any } (\text{Val } i) \Delta$. We saw All earlier; Any is similar but encodes the fact that *some* element in the list satisfies the predicate. The cases rule for Any makes good use of patterns: case analysis on a hypothesis which matches the pattern $\text{Any } _ []$ is safe since the hypothesis is contradictory; case analysis on a hypothesis which matches $\text{Any } _ (_ :: _)$ is unsafe but often useful.

With the help of this cases rule, we prove some fundamental lemmas about All and Any, such as a weakening lemma for All:

$$(\forall x, P x \rightarrow Q x) \rightarrow \text{All } P \text{ xs} \rightarrow \text{All } Q \text{ xs}$$

After induction on the All premise, Aesop finishes the proof. We use this weakening lemma to prove that all elements of a list are members of that list: $\text{All } (\cdot \in \text{xs}) \text{ xs}$. The application of weakening requires support for metavariables

since P is unconstrained and becomes a metavariable. Similarly, metavariables are crucial when proving existentially quantified lemmas, e.g.

Any $P \text{ xs} \leftrightarrow \exists a : \alpha, P \ a \wedge a \in \text{xs}$

The prover itself, `Cal`, attempts to prove a sequent by breaking down connectives according to the classical sequent calculus rules and collecting lists of positive and negative propositional variables when they appear on either side of the sequent. A branch of the proof terminates successfully when the same variable occurs both positively and negatively, corresponding to the usual *Axiom* rule. We use `Any ($\bullet \in \text{ys}$) xs` to check if two lists `xs` and `ys` share a common element. The computational behaviour of `Cal` thus depends on the decidability of `Any`, which is proved using `Aesop`. We verify soundness and completeness of the prover simultaneously, using induction on the call structure of `Cal`. The main theorem states that `Cal` proves a sequent if and only if the sequent is valid for all decidable interpretations.

Since the prover rearranges formulas, the proof relies on the fact that `All` and `Any` respect list permutations, as encoded by an inductive predicate taken from the `Agda` standard library.⁵ Here, `Aesop` significantly reduces our workload: that permutations are symmetric, that they are preserved by `map` and that `All` and `Any` respect permutations can be proven automatically after we perform induction.

As a consequence of the soundness and completeness of `Cal`, we additionally obtain soundness and completeness of its underlying proof system, formulated as an inductive predicate `Proof $\Gamma \Delta$` . A key ingredient of the proof is this weakening lemma:

`Proof $\Gamma \Delta \rightarrow \text{Proof } \Gamma (\delta :: \Delta)$`

After induction on the premise, `Aesop` proves the lemma automatically, apart from one case which requires an explicit application of the induction hypothesis. This is because two of the constructors of `Proof` allow us to apply arbitrary permutations to the sequents, which `Aesop`'s metavariable handling is too weak to find. These constructors also apply to every goal, so it is important that `Aesop` is not limited to depth-first search, which might get lost in infinite permutations.

6 Related Work

The closest relative of `Aesop` is Isabelle's `auto` [18, 20]. Like `Aesop`, it performs a tree-based search with integrated simplification and it distinguishes between safe and unsafe rules. `Aesop` adds a best-first strategy (`auto` is depth-first) and normalisation as a separate phase. It also adds a number of rule builders apart from `auto`'s `intro`, `elim` and `destruct` rules,

though some of these can be emulated with auxiliary Isabelle tools.

More fundamentally, `auto` is used as a semi-black-box tool in practice. It is essentially undocumented, so it is difficult to understand the details of its search procedure, e.g. how exactly the simplifier is invoked, how it integrates `blast` [19] (a tableau prover) and how metavariables interact with safe rules. Indeed, our conversations with experienced Isabelle users indicate that they are unaware of these details and that as a result, their interactions with `auto` are partly based on trial and error, adding and removing rules until `auto` is able to prove a goal.

Other semi-black-box proof tools include PVS's `grind` [5] and the 'waterfalls' of ACL2 [11] and its descendants. While these tools are based on simple search algorithms and are extensively documented, they use, at least in their default configurations, a large number of proof methods (e.g. several forms of simplification; decision procedures for certain fragments of the logic; several methods for quantifier instantiation) in a fixpoint loop surrounded by pre- and post-processing steps. As a result, it again becomes somewhat difficult for users to predict and adjust their behaviour.

`Aesop`, by contrast, attempts to remain firmly white-box by limiting itself to a small number of simple concepts (essentially: normalisation, safe and unsafe rules) with no opaque heuristics and no pre- or post-processing. This should make it possible to design predictable special-purpose rule sets for specific domains. With larger rule sets, `Aesop` may also become somewhat unpredictable, but at least its transparency should make it easier to debug unexpected failures or performance issues. Of course, the downside of `Aesop`'s simplicity is that it is considerably less powerful than, say, `grind`; for example, it does not currently have any support for arithmetic beyond that provided by Lean's simplifier.

Even farther towards the white-box end of the scale lie Coq's `auto` and `eauto`. These tactics perform backtracking depth-first search (up to some configurable depth limit) with arbitrary rules, so they are essentially `Aesop` without safe or normalisation rules and with a different search strategy. Matita's `auto` [1] augments `eauto` with a superposition calculus for equational reasoning and provides a GUI which allows users to inspect the search tree.

A rare white-box tool not based on tree search is Isabelle's `auto2` [23], which uses a saturation algorithm instead. This means that rules can be applied without backtracking, but the proof procedure is also farther removed from interactive proof and therefore perhaps less easy to customise.

There are also black-box tools based on tree search, notably Coq's `sauto` [6] and the `AgSy` tool [14] for `Agda` [17]. These tools use fairly strong default rules, some of which could also be interesting for `Aesop`. But since they are push-button tools, their rules are also quite opaque.

For Lean 3, `mathlib` [4] already contains some search tactics which are currently being ported to Lean 4: `continuity`,

⁵<https://github.com/agda/agda-stdlib/blob/ebfb8814b4330b314da8fb9cae527e6a6fab01aa/src/Data/List/Relation/Binary/Permutation/Propositional.agda>

measurability, tidy, tautology and finish. These tactics perform essentially depth-first search with various rule sets, so Aesop should supersede them. However, finish uses e-matching and so makes better use of unoriented equations.

Our handling of metavariables is most closely related to that of `THEOREMV` [12], which, like Aesop, uses an AND/OR search tree. Variations of the `THEOREMV` algorithm are also used for tableaux with metavariables ('free variable tableaux') [9]. However, these algorithms are specific to first-order logic and do not obviously generalise to our setting. In particular, they require that rules behave uniformly for different metavariable assignments.

7 Conclusion

We have presented Aesop, a white-box proof search tactic for Lean. Starting with a straightforward tree search framework, we have added features that increase the power of the search while keeping its semantics simple and transparent: best-first search with customisable prioritisation, which lets us effectively use rules that are only occasionally useful or that may loop; safe rules, which are useful both for performance and for debugging; normalisation, to establish invariants which other rules can rely on; and simplification, which enables equational reasoning. Taken together, these features should allow users to design effective and predictable rule sets.

To support goals with metavariables, we have developed a generic algorithm for tree-based search with metavariables. The algorithm is independent of the search strategy and of the underlying logic and is, we believe, as complete as the given rule set allows.

Acknowledgments

We thank Jasmin Blanchette for advising us throughout the development process and giving detailed comments on drafts of this paper; Sebastian Ullrich for providing excellent feedback on early versions of Aesop; Gabriel Ebner for helping with Aesop's implementation; and the anonymous reviewers for providing comprehensive and insightful reviews.

Limperg was funded by the NWO under the Vidi programme (project No. 016.Vidi.189.037, Lean Forward).

References

- [1] Andrea Asperti, Wilmer Ricciotti, Claudio Sacerdoti Coen, and Enrico Tassi. 2011. The Matita interactive theorem prover. In *CADE 2011*. 64–69. https://doi.org/10.1007/978-3-642-22438-6_7
- [2] Kshitij Bansal, Sarah Loos, Markus Rabe, Christian Szegedy, and Stewart Wilcox. 2019. HOList: an environment for machine learning of higher order logic theorem proving. In *ICML 2019*. 454–463. <https://proceedings.mlr.press/v97/bansal19a.html>
- [3] Jasmin Christian Blanchette, Cezary Kaliszyk, Lawrence C. Paulson, and Josef Urban. 2016. Hammering towards QED. *Journal of Formalized Reasoning* 9, 1 (2016), 101–148. <https://doi.org/10.6092/issn.1972-5787/4593>
- [4] The mathlib Community. 2020. The lean mathematical library. In *CPP 2020*. 367–381. <https://doi.org/10.1145/3372885.3373824>
- [5] Judy Crow, Sam Owre, John Rushby, Natarajan Shankar, and Mandayam Srivas. 1995. A Tutorial Introduction to PVS. In *Workshop on Industrial-Strength Formal Specification Techniques 1995*. <http://www.csl.sri.com/papers/wift-tutorial/>
- [6] Łukasz Czajka. 2020. Practical proof search for Coq by type inhabitation. In *IJCAR 2020*. 28–57. https://doi.org/10.1007/978-3-030-51054-1_3
- [7] Leonardo de Moura and Sebastian Ullrich. 2021. The Lean 4 theorem prover and programming language. In *CADE 2021*. 625–635. https://doi.org/10.1007/978-3-030-79876-5_37
- [8] Thibault Gauthier, Cezary Kaliszyk, Josef Urban, Ramana Kumar, and Michael Norrish. 2021. TacticToe: Learning to prove with tactics. *Journal of Automated Reasoning* 65, 2 (2021), 257–286. <https://doi.org/10.1007/s10817-020-09580-x>
- [9] Martin Giese. 2001. Incremental closure of free variable tableaux. In *IJCAR 2001*. 545–560. https://doi.org/10.1007/3-540-45744-5_46
- [10] Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward W. Ayers, and Stanislas Polu. 2021. Proof artifact co-training for theorem proving with language models. <https://doi.org/10.48550/ARXIV.2102.06203>
- [11] M. Kaufmann and J. Strother Moore. 1996. ACL2: an industrial strength version of Nqthm. In *COMPASS 1996*. 23–34. <https://doi.org/10.1109/COMPASS.1996.507872>
- [12] Boris Konev and Tudor Jebelean. 2005. Solution lifting method for handling meta-variables in `THEOREMV`. *Journal of Mathematical Sciences* 126, 3 (2005), 1182–1194. <https://doi.org/10.1007/s10958-005-0090-6>
- [13] Guillaume Lample, Marie-Anne Lachaux, Thibaut Lavril, Xavier Martinet, Amaury Hayat, Gabriel Ebner, Aurélien Rodriguez, and Timothée Lacroix. 2022. HyperTree proof search for neural theorem proving. <https://doi.org/10.48550/ARXIV.2205.11491>
- [14] Fredrik Lindblad and Marcin Benke. 2004. A tool for automated theorem proving in Agda. In *TYPES 2004*. 154–169. https://doi.org/10.1007/11617990_10
- [15] Conor McBride and James McKeena. 2004. The view from the left. *Journal of Functional Programming* 14, 1 (2004), 69–111. <https://doi.org/10.1017/S0956796803004829>
- [16] William McCune. 1992. Experiments with discrimination-tree indexing and path indexing for term retrieval. *Journal of Automated Reasoning* 9, 2 (1992), 147–167. <https://doi.org/10.1007/BF00245458>
- [17] Ulf Norell. 2007. *Towards a practical programming language based on dependent type theory*. Ph.D. Dissertation. Chalmers University of Technology, Göteborg, Sweden. <https://www.cse.chalmers.se/~ulfn/papers/thesis.pdf>
- [18] Lawrence C. Paulson. 1996. *Generic automatic proof tools*. Technical Report UCAM-CL-TR-396. University of Cambridge. <https://doi.org/10.48456/tr-396>
- [19] Lawrence C. Paulson. 1999. A generic tableau prover and its integration with Isabelle. *Journal of Universal Computer Science* 5, 3 (1999), 73–87. <https://doi.org/10.3217/jucs-005-03-0073>
- [20] Lawrence C. Paulson, Tobias Nipkow, and Makarius Wenzel. 2019. From LCF to Isabelle/HOL. *Formal Aspects of Computing* 31, 6 (2019), 675–698. <https://doi.org/10.1007/s00165-019-00492-1>
- [21] Geoff Sutcliffe. 2017. The TPTP problem library and associated infrastructure: from CNF to TH0, TPTP v6.4.0. *Journal of Automated Reasoning* 59, 4 (2017), 483–502. <https://doi.org/10.1007/s10817-017-9407-7>
- [22] Jørgen Villadsen. 2020. A micro prover for teaching automated reasoning. In *PAAR 2020*.
- [23] Bohua Zhan. 2016. AUTO2, a saturation-based heuristic prover for higher-order logic. In *ITP 2016*. 441–456. https://doi.org/10.1007/978-3-319-43144-4_27

Received 2022-09-21; accepted 2022-11-21

CHAPTER 8

An Abstract Framework for Synthetic Completeness

The Axiom of Choice is obviously true, the well-ordering principle obviously false, and who can tell about Zorn's lemma?

—Jerry L. Bona

In this chapter I present an abstract framework for synthetic completeness proofs formalized in Isabelle/HOL. By working abstractly we avoid relying on the features of a particular calculus and are instead forced to focus on the actual essence of the completeness argument. This allows us to reuse the results for a large class of proof systems. The constructions and theorems are all available in the Archive of Formal Proofs in my entry “Synthetic Completeness” which is listed on page [167](#). However, at time of writing, the Isabelle listings below differ cosmetically from the online entry: I have changed some names to ease the presentation. Isabelle has also verified the version below.

8.1 Introduction

We have seen several synthetic completeness proofs so far: for an axiomatic system for propositional logic in chapter 2, an axiomatic system for first-order logic in chapter 3, a tableau system for hybrid logic in chapter 5 and a range of epistemic logics in chapter 6. In each of these chapters, I formalized Lindenbaum’s lemma separately in order to build maximal consistent sets (MCSs) with or without saturation. Moreover, I defined Hintikka sets from scratch for each logic to prove a truth lemma: formulas in MCSs have a satisfying model. In this chapter I provide an abstract, formalized framework for synthetic completeness proofs. The framework builds (saturated) MCSs for any calculus that satisfies a few properties, and provides a way of deriving Hintikka sets mechanically from the logic’s semantics and the models induced by the MCSs: given the canonical models, it says what we need to prove about the MCSs for each constructor to prove model existence. With this framework, I make the following contributions:

- I formalize a transfinite, reusable version of Lindenbaum’s lemma applicable to languages of any cardinality.
- My framework builds saturated maximal consistent sets, as long as the cardinality of the type of parameters (e.g. constant symbols) is large enough.
- I provide a *Hintikka equation* that describes sets which are often useful when proving the truth lemma.

I have instantiated this framework to the following logics, proving strong completeness for each of them:

- A propositional tableau system.
- A propositional sequent calculus.
- A natural deduction system for first-order logic.
- An axiomatic system for modal logic.
- A natural deduction system for hybrid logic.

Except for the tableau system, where the Hintikka sets model the rules of the calculus rather than the semantics of the logic, all examples derive the Hintikka sets from the semantics and MCS-induced models in the style of the framework. I will use the term *formula* broadly. In the hybrid logic example, we actually work with labelled formulas which are formulas paired with a nominal.

8.1.1 Related work

The previous chapters in this thesis contain a lot of related work when it comes to formalized completeness proofs. The work below deserves a special mention.

Blanchette et al. [10] have formalized an abstract framework for analytic completeness proofs in Isabelle/HOL. Their work formalizes infinite derivation trees in the abstract and proves various useful properties about them. It can be easily used to create executable provers, but targets tableaux and sequent calculus more than axiomatic and natural deduction systems.

Fitting employs *abstract consistency properties* in his completeness proofs for tableaux, resolution and axiomatic systems. These are a different way of working with an abstract notion of consistency from which the model existence theorem can be proved. Fitting gives the abstract consistency properties for propositional and first-order logic and shows that the various proof systems live up to the requirements. My trick for deriving Hintikka sets can be seen as a different way of *decomposing* derivational consistency into conditions that are easier to connect with the semantics.

Schlöder and Koepke [8] used Mizar to formalize a completeness theorem for first-order logic for uncountably large languages.

8.2 Maximal Consistent Sets

Consistency of a set of formulas S wrt. a logical calculus typically means that we cannot refute, derive a contradiction from, any subset of S . Maximality of S means that any formula consistent with S is contained in S . The constructions and proofs below are adapted from Chang and Keisler [1] with two main differences. One, they work in set theory, where I work in the weaker higher-order logic (HOL), which for instance, cannot express the type of all ordinals. Two, to ensure saturation, they extend the language they are working with, which from a HOL perspective means changing its type. Instead, I assume that there are enough parameters from the beginning (Berghofer [6] uses the same trick in his Isabelle/HOL formalization).

It is intuitively simple to extend a consistent set S to a maximal consistent one. Take the countable case as a demonstration. First, we assume that we have an exhaustive enumeration of formulas p_n for each n . Next, we build a sequence of consistent sets S_n for each n with $S_0 = S$ as our starting point. To form S_{n+1}

we simply take p_n together with S_n , if this is consistent, or S_n alone if not. The MCS is then the infinite union $\bigcup_n S_n$. Henkin's idea [5] to ensure saturation is equally simple. Saturation of S means that for every existential formula in S , S also contains a witness of that formula. Adding these witnesses at the end can be difficult, since we need to show that there are still consistent witnesses available. Instead, whenever we add an existential formula, we also add suitable witnesses for that formula.

To formalize this process in the abstract and prove that the result indeed is a maximal consistent saturated set, we need to make some assumptions. Isabelle's locale mechanism is excellent for this. It allows us to work in an abstract context with e.g. a predicate *consistent* that obeys certain assumptions. Interested logicians can then instantiate the locale with their own concrete consistency predicate, prove that it fulfills the requirements and obtain specialized versions of all the definitions and proofs that I have given abstractly.

8.2.1 Ordinals and Cardinals in Isabelle/HOL

I rely on Isabelle/HOL's support for ordinals and cardinals, developed by Blanchette et al. [9], to support uncountable languages. Here, an ordinal is simply a set of pairs that represents a wellorder relation. Being a wellorder, every non-empty subset of an ordinal has a least element. The set of elements related by the wellorder is called its *Field*. Due to the axiom of choice, we can impose a wellorder on any set and thus think of it as an ordinal. The cardinal of a set is an ordinal for that set that is no greater than any other ordinal for that set. Here, ordinal r is no greater than ordinal r' , $r \leq_o r'$, when r embeds into r' (there is a map which both respects and reflects the order). Again due to the axiom of choice, there is a cardinal for any set X , which we denote $|X|$.

Since an ordinal (and thus cardinal) is simply a wellorder, we can relate the elements of its *Field*. The function *under* r n gives the set of elements smaller than n in the wellorder r , while its strict counterpart *underS* r n excludes n itself. Any wellorder has a least element, which means any ordinal has a zero element. The successor of an element n , is the smallest element $m > n$ by the wellorder. If an element in an ordinal is the successor of some other element, we call it a successor element. Finally, limit elements are those elements that are not successor elements, for instance ω wrt. the natural numbers, which is greater than any natural number but not the direct successor of any of them. Limit ordinals are ordinals with the property that every element has a successor. Any cardinal over an infinite *Field* is a limit ordinal.

Wellorder recursion is an elimination principle for elements of a wellorder, akin to

Table 8.1: Assumptions for enumerating formulas of type $'a$.

- A relation over the formulas: $r :: 'a \text{ rel}$
- The relation is a wellorder: $\text{Well-order } r$
- The relation is an infinite cardinal: $\text{Cinfinite } r$

primitive recursion on natural numbers. More precisely, when defining a function $f\ n$ by wellorder recursion, we need to provide three cases for the type of n : when it is a zero, successor and limit element, respectively. Zero elements are the starting point, so $f\ 0$ must be defined outright. When defining $f\ (n + 1)$, for successor elements $n + 1$, we get to assume $f\ n$. When defining $f\ n$ for limit elements, we can make use of $f\ m$ for any $m < n$ that is strictly smaller by the wellorder. Wellorder induction provides the complementary proof technique.

The Isabelle/HOL cardinal library provides recursion over wellorders, *worecZSL*, split into the three cases: zero (Z), successor (S) and limit (L) elements.

8.2.2 Enumeration

We need an enumeration of the formulas we want to build MCSs of. We want to support uncountably large languages, so we cannot simply assume a surjective function from natural numbers to formulas, and higher-order logic is not strong enough to express a function from ordinals to formulas. Instead, we assume a wellorder relation over formulas (one can always be imposed). This provides enough structure that we can use the formulas themselves as numbers. We can then use wellorder recursion to construct each set S_n in our sequence, where n is a formula serving double duty as a number (and the target of the recursion). Wellorder induction provides the complementary proof technique, allowing us to prove results about each set S_n .

To prove maximality we require that every formula has a successor (since formula n only gets added at stage S_{n+1}). Therefore, our wellorder must be an infinite cardinal so that it is a limit ordinal and has this property (an infinite cardinal be found for any infinite set). In total, we can work with anything satisfying table 8.1. Note that infinite cardinals are always wellorders, but table 8.1 is written such that it corresponds to the existing locale *wo-rel* with an additional infinity assumption.

Table 8.2: Assumptions for building maximal consistent sets.

- A consistency predicate: $consistent :: 'a \text{ set} \Rightarrow \text{bool}$
- Subsets preserve consistency:
 $\bigwedge S S'. consistent\ S \Longrightarrow S' \subseteq S \Longrightarrow consistent\ S'$
- Inconsistencies are finite:
 $\bigwedge S. \neg consistent\ S \Longrightarrow \exists S' \subseteq S. finite\ S' \wedge \neg consistent\ S'$

Table 8.3: Additional assumptions for building saturated MCSs.

- A function that returns the parameters in a formula:
 $params :: 'a \Rightarrow 'i \text{ set}$
- A function that returns witnesses for a formula (fresh wrt. the given set):
 $witness :: 'a \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set}$
- Formulas contain finitely many parameters: $\bigwedge p. finite\ (params\ p)$
- Sets of witnesses contain finitely many parameters:
 $\bigwedge p S. finite\ (\bigcup q \in witness\ p\ S. params\ q)$
- Witnesses preserve consistency (when free parameters are available):
 $\bigwedge p S. consistent\ (\{p\} \cup S)$
 $\Longrightarrow infinite\ (UNIV - (\bigcup q \in S. params\ q))$
 $\Longrightarrow consistent\ (witness\ p\ S \cup \{p\} \cup S)$

8.2.3 Lindenbaum's Lemma

Table 8.2 lists the basic assumptions we need, besides table 8.1, to build maximal consistent sets (\bigwedge is Isabelle's metalogical universal quantifier). We need a consistency predicate where subsets of consistent sets are consistent and inconsistent sets have a finite inconsistent subset. This last assumption is usually satisfied by the finiteness of proofs. Table 8.3 lists the extra assumptions necessary to ensure saturation. In order to add witnesses, we need a function that gives us the parameters in a formula and a function that generates witnesses for a formula, given a set of formulas whose parameters cannot be used. For non-existential formulas, this *witness* function returns an empty set. To ensure that we do not use up all the parameters, we need that formulas and sets of witnesses contain only finitely many parameters. Finally, to prove consistency, we need that witnesses, built from unused parameters, preserve consistency.

8.2.4 Lindenbaum's Extension

As described above, we construct a sequence of sets S_n from an initial set S . For n we use the elements of our wellorder: formulas, serving a double purpose as numbers. In the following, we are working inside this abstract wellorder, r , which we assume to be a limit ordinal such that every element has a successor. This is especially apparent in the Isabelle definitions where we often have to reference the *Field* of the wellorder explicitly.

DEFINITION 8.1 (CONSTRUCTING S_n) Assume an initial set S . We use wellorder recursion to construct S_n by cases on n . When $n = 0$, a zero element, $S_0 = S$. When it is $n + 1$, a successor element, $S_{n+1} = \text{witness } n \ S_n \cup \{n\} \cup S_n$ when $\{n\} \cup S_n$ is consistent and $S_{n+1} = S_n$ otherwise. When n is a limit element, $S_n = \bigcup_{m < n} S_m$ is built from the sets of strictly smaller elements.

In Isabelle this becomes:

definition *extend* $S \ n \equiv \text{worecZSL } r \ S \ \text{extend} S \ \text{extend} L \ n$

definition *extendS* $n \ \text{prev} \equiv \text{if consistent } (\{n\} \cup \text{prev}) \ \text{then witness } n \ \text{prev} \cup \{n\} \cup \text{prev} \ \text{else prev}$

definition *extendL* $\text{rec } n \equiv \bigcup m \in \text{under} S \ r \ n. \ \text{rec } m$

The above definition is admissible, since the limit case only relies on strictly smaller elements.

DEFINITION 8.2 (CONSTRUCTING $\bigcup_n S_n$) The maximal consistent saturated set for S is now $\bigcup_n S_n$.

definition *Extend* $S \equiv \bigcup n \in \text{Field } r. \ \text{extend } S \ n$

LEMMA 8.3 (PROPERTIES OF THE EXTENSION) *The initial set S is included in any extension S_n (when n is in the Field of our wellorder). This means that when the Field is not empty, S is included in $\bigcup_n S_n$. For $m \leq n$, $S_m \subseteq S_n$.*

shows $n \in \text{Field } r \implies S \subseteq \text{extend } S \ n$

shows $\text{Field } r \neq \{\} \implies S \subseteq \text{Extend } S$

shows $m \in \text{under } r \ n \implies \text{extend } S \ m \subseteq \text{extend } S \ n$

PROOF. Either directly or by wellorder induction. □

8.2.5 Consistency

The trickiest part of proving consistency is ensuring that we have fresh parameters available for the witnesses. First note that any parameter in a set S_n comes either from the initial set S , from a formula $m < n$ or from the witnesses of a formula $m < n$. To ease notation, let $\text{paramss } S \equiv \bigcup p \in S. \text{ params } p$ be all the parameters in the set S .

LEMMA 8.4 (CONSISTENCY OF S_n) *Assume that S is consistent and that $|\text{UNIV} - (\text{paramss } S)|$, the cardinality of unused parameters, is at least the cardinality of our wellorder of formulas. Then S_n is consistent.*

assumes consistent S and $r \leq_o |\text{UNIV} - \text{paramss } S|$
shows consistent (extend S n)

PROOF. By wellorder induction on n . $S_0 = S$ is consistent by assumption.

For $n + 1$, note first that the cardinality of formulas seen so far is strictly smaller than the total number of formulas in our wellorder: $|\{m \mid m < n\}| < r$. Since the parameters of a formula or a set of witnesses is finite, the cardinality of all the parameters added so far, call it X , is strictly smaller than r . By assumption it is then strictly smaller than $|\text{UNIV} - (\text{paramss } S)|$. This means that $|\text{UNIV} - (\text{paramss } S \cup X)|$, the cardinality of parameters that were not there initially and have not been added, is at least r . Since every parameter in S_n comes from either $\text{paramss } S$ or X , there must be infinitely many unused parameters in S_n . By assumption, adding witnesses thus preserves consistency.

For n a limit ordinal, assume towards a contradiction that S_n is inconsistent. Then a finite subset of $\bigcup_{m < n} S_m$ is inconsistent. But the finite subset is built from finitely many elements, so we have a finite subset $M \subseteq \{m \mid m < n\}$ such that $\bigcup_{m \in M} S_m$ is inconsistent. Since M is finite, and $S_i \subseteq S_j$ for $i \leq j$, we must have an upper bound $k < n$ such that $(\bigcup_{m \in M} S_m) \subseteq S_k$. But S_k is consistent by the induction hypothesis, so it cannot contain an inconsistent $\bigcup_{m \in M} S_m$. \square

LEMMA 8.5 ($\bigcup_n S_n$ IS CONSISTENT) *If S is consistent and the cardinality of unused parameters in S is at least r , then $\bigcup S_n$ is consistent.*

assumes consistent S and $r \leq_o |\text{UNIV} - \text{paramss } S|$
shows consistent (Extend S)

PROOF. Assume towards a contradiction that $\bigcup_n S_n$ is inconsistent. Then a finite subset $S' \subseteq \bigcup_n S_n$ is inconsistent. Since S' is finite, there must be an m such that $S' \subseteq S_m$. By lemma 8.4, S_m is consistent, causing a contradiction. \square

8.2.6 Relative Maximality

We are now ready to prove maximality wrt. our *Field*.

DEFINITION 8.6 (*r*-MAXIMALITY) A set S is r -maximal if it contains every formula in the *Field* of r that is consistent with S .

definition $\text{maximal}' S \equiv \forall p \in \text{Field } r. \text{ consistent } (\{p\} \cup S) \longrightarrow p \in S$

LEMMA 8.7 ($\bigcup_n S_n$ IS *r*-MAXIMAL) The set $\bigcup_n S_n$ is always r -maximal.

shows $\text{maximal}' (\text{Extend } S)$

PROOF. Assume p is in the *Field* of r and consistent with $\bigcup_n S_n$. Then it is consistent with S_p . Moreover, since r is assumed to be a limit ordinal, the successor $p + 1$ is also in the *Field* of r . In combination, $p \in S_{p+1} \subseteq \bigcup_n S_n$. \square

8.2.7 Relative Saturation

Saturation is as simple to define and prove as maximality.

DEFINITION 8.8 (*r*-SATURATION) A set S is r -saturated if, for every formula p in both S and the *Field* of r , S contains witnesses for p .

definition $\text{saturated}' S \equiv \forall p \in S. p \in \text{Field } r \longrightarrow (\exists S'. \text{ witness } p \ S' \subseteq S)$

LEMMA 8.9 ($\bigcup_n S_n$ IS *r*-SATURATED) When $\bigcup_n S_n$ is consistent, it is also r -saturated.

assumes $\text{consistent} (\text{Extend } S)$

shows $\text{saturated}' (\text{Extend } S)$

PROOF. Like the proof of lemma 8.7, but noting that the witnesses of a formula get added alongside the formula. \square

8.2.8 Concrete Limit Ordinals

So far, we have worked with an abstract infinite cardinal over some arbitrary *Field*. This allows us to build MCSs over only a subset of formulas, in case we need to do that. However, we typically work with the full universe of formulas.

In this case we can replace the assumptions in table 8.1 with a single assumption that the universe of formulas is infinite. We then instantiate r as the cardinality of this universe, which is trivially a wellorder and an infinite cardinal. Under this assumption that the wellorder r contains every formula, r -maximality and r -saturation reduce to simpler definitions.

DEFINITION 8.10 (MAXIMALITY) A set S is maximal if it contains every formula that is consistent with S .

definition *maximal* $S \equiv \forall p. \text{consistent}(\{p\} \cup S) \longrightarrow p \in S$

DEFINITION 8.11 (SATURATION) A set S is saturated if, for every formula p in S , S contains witnesses for p .

definition *saturated* $S \equiv \forall p \in S. \exists S'. \text{witness } p \ S' \subseteq S$

We get the following theorem in conclusion.

THEOREM 8.12 ($\bigcup_n S_n$ IS MAXIMAL, CONSISTENT, SATURATED)

Assume tables 8.2 and 8.3 and that the universe of formulas is infinite. If S is consistent and the cardinality of unused parameters in S is at least the cardinality of all formulas, then $\bigcup_n S_n$ is a maximal consistent saturated set.

assumes *consistent* S **and** $|UNIV :: 'a \text{ set}| \leq o \ |UNIV - \text{paramss } S|$
shows *consistent* (*Extend* S)
 and *maximal* (*Extend* S)
 and *saturated* (*Extend* S)

PROOF. Follows from lemmas 8.4, 8.7 and 8.9 and the simplifying assumption that the *Field* of r contains every formula. \square

If the logic does not require saturation, we can trivially satisfy table 8.3: by letting both *params* and *witness* return empty sets, the assumptions about them become trivial. For logics without existential formulas, we just need to fulfill the assumptions in table 8.2 and prove that the universe of formulas is infinite.

THEOREM 8.13 ($\bigcup S_n$ IS A MAXIMAL CONSISTENT SET)

Assume table 8.2 and that the universe of formulas is infinite. If S is consistent, then $\bigcup_n S_n$ is a maximal consistent set.

assumes *consistent* S
shows *consistent* (*Extend* S) **and** *maximal* (*Extend* S)

PROOF. Special case of theorem 8.12. \square

Table 8.4: Assumptions about refutational proof systems (wrt. table 8.2).

- A refutation predicate: $\text{refute} :: 'a \text{ list} \Rightarrow \text{bool}$
- That is invariant under all structural rules:
 $\bigwedge A B. \text{refute } A \implies \text{set } A \subseteq \text{set } B \implies \text{refute } B$
- Where consistency means that no list of elements can be refuted:
 $\bigwedge S. \text{consistent } S = (\nexists S'. \text{set } S' \subseteq S \wedge \text{refute } S')$

8.3 Refutations, Derivations and MCSs

I now briefly consider abstract proof systems and how they relate to maximal consistent sets. That is, what knowing a little bit more about the abstract consistency predicate tells us. In particular, for the proof of the truth lemma it is useful to know when a formula is included in an MCS and when it is absent. I assume proof systems that work with lists, since lists can easily be converted to sets or multisets, for calculi that are implemented using these structures instead.

8.3.1 Refutations

Table 8.4 assumes a predicate that tells us whether a list of formulas is refutable and asks that any permutation, contraction and weakening of a refutable list is refutable. Finally, it stipulates that consistency of a set means that no list of elements can be refuted (the Isabelle function *set* turns a list into a set). Note that Isabelle lists are inductive structures built from an empty list $[]$ and a prepend operation $\#$. This makes them finite by definition.

THEOREM 8.14 (REFUTABILITY AND MCSs) *Assume the basic MCS assumptions in table 8.2 and the refutational assumptions in table 8.4. Formula p is absent from a maximal consistent set S if and only if there is a list of elements S' from S such that p, S' is refutable.*

assumes *consistent S and maximal S*

shows $p \notin S \longleftrightarrow (\exists S'. \text{set } S' \subseteq S \wedge \text{refute } (p \# S'))$

PROOF. Assume $p \notin S$. By maximality, adding it yields a refutable list of elements and by the structural rules, we can rearrange this to put p at the front. For the other direction, assume p, S' is refutable for some list of elements S' from S . Then $p \notin S$ as this would contradict the consistency of S . \square

Table 8.5: Assumptions about derivational proof systems (wrt. table 8.2).

Basic assumptions:

- A derivation predicate: $derive :: 'a\ list \Rightarrow 'a \Rightarrow bool$
- A falsity element: $fls :: 'a$
- Where the derivation predicate is invariant under all structural rules:
 $\bigwedge A\ B\ p.\ derive\ A\ p \implies set\ A \subseteq set\ B \implies derive\ B\ p$
- And consistency means we cannot derive falsity from any list of elements:
 $\bigwedge S.\ consistent\ S = (\nexists S'.\ set\ S' \subseteq S \wedge derive\ S'\ fls)$

Additional assumptions:

- We can derive any assumption:
 $\bigwedge A\ p.\ p \in set\ A \implies derive\ A\ p$
- We have the Cut rule:
 $\bigwedge A\ B\ p\ q.\ derive\ A\ p \implies derive\ (p \# B)\ q \implies derive\ (A @ B)\ q$

8.3.2 Derivations

Table 8.5 assumes a predicate that tells us whether a formula can be derived from a list of assumptions and asks that any permutation, contraction and weakening of the list of assumptions does not change derivability. It also assumes a falsity element and stipulates that consistency of a set means that we cannot derive falsity from any list of elements. Additionally, we may assume that we can derive any assumption and that we have the Cut rule: if we can derive p from A and q from p, B then we can *cut* out p and derive q directly from A, B (in Isabelle, $@$ denotes list concatenation).

The first theorem mirrors its refutational counterpart.

THEOREM 8.15 (DERIVABILITY OF FALSITY AND MCSs) *Assume both the basic MCS assumptions in table 8.2 and the basic derivational assumptions in table 8.5. Formula p is absent from a maximal consistent set S if and only if there is a list of elements S' from S such that we can derive falsity from p, S' .*

assumes *consistent S and maximal S*

shows $p \notin S \iff (\exists S'.\ set\ S' \subseteq S \wedge derive\ (p \# S')\ fls)$

PROOF. Analogous to the proof of theorem 8.14. □

The second theorem relies on having the Cut rule.

THEOREM 8.16 (DERIVABILITY AND MCSs) *Assume the basic MCS assumptions in table 8.2 and all the derivational assumptions in table 8.5. Formula p is included in a maximal consistent set S if and only if we can derive p from some list of elements from S .*

assumes *consistent S and maximal S*

shows $p \in S \iff (\exists S'. \text{ set } S' \subseteq S \wedge \text{derive } S' p)$

PROOF. Assume $p \in S$. Since we can derive any assumption, we can just pick a list of elements from S that includes p .

Assume we can derive p from some list of elements A from S . We want to show that $\{p\} \cup S$ is consistent, in which case $p \in S$ follows from maximality. Assume towards a contradiction that we can derive falsity from p and a list of elements B from S . Then by the Cut rule, we can derive falsity from A, B . But A, B is a list of elements from S , contradicting its consistency. \square

8.4 Truth Lemma

I will now demonstrate a technique for deriving Hintikka sets from a logic's semantics and the models induced by the maximal consistent (saturated) sets. This technique works for compositional semantics, where the truth value of a formula is defined by the truth values of its subformulas (but potentially in a different context). Note that Smullyan [3] defines Hintikka sets to be *saturated downwards* only. My (derived) Hintikka sets are fully saturated, downwards and upwards, which Smullyan calls *saturated sets*. Since my Hintikka sets are also Hintikka sets by Smullyan's definition and I already use *saturation* for something else, I use the term *Hintikka sets*. I first demonstrate the technique informally, using propositional logic as an example, before explaining the Isabelle/HOL formalization Isabelle/HOL and applying it to a range of examples. I use the term *truth lemma*, but *model existence theorem* would also be appropriate.

8.4.1 Introduction

We want to prove a truth lemma: any maximal consistent (saturated) set S induces a model that satisfies all formulas in S . As a stepping stone, we want to use Hintikka sets to classify MCSs not by the consistency predicate, but by conditions on subformulas. We thereby split the proof into two smaller steps:

proving that any MCS is a Hintikka set and that any Hintikka set induces a model that satisfies the formulas it contains.

8.4.1.1 Example Language

Take the following language for formulas p, q over propositional variables P :

$$p ::= P \mid \perp \mid p \rightarrow p$$

Our models are interpretations I that assign either true or false to each propositional variable and our semantics is as follows:

$$\begin{aligned} I \models P & \iff I \models P \\ I \models \perp & \iff \text{False} \\ I \models p \rightarrow q & \iff I \models p \longrightarrow I \models q \end{aligned}$$

8.4.1.2 Manual Hintikka Sets

We can manually define Hintikka sets H for our logic with the conditions:

$$\begin{aligned} \perp & \notin H \\ p \rightarrow q \in H & \iff p \in H \longrightarrow q \in H \end{aligned}$$

This makes it easy to show that $p \in H \iff I^H \models p$ where $I^H \models P = (P \in H)$ is the canonical model. We would also have to fix a proof system and show that any MCS satisfies the above Hintikka conditions, but we leave this for later.

8.4.1.3 Derived Hintikka Sets

To begin, note that we can take a compositional semantics like the one above and *punch a hole in it* by replacing all recursive calls with calls to some relation. The resulting non-recursive semantics allows us to capture the idea that Hintikka sets are defined by conditions on subformulas.

For our example, we obtain a new predicate *semics* (the word *semantics* with a hole in it) over the relation *rel*:

$$\begin{aligned} \text{semics } I \text{ rel } P & \iff I \models P \\ \text{semics } I \text{ rel } \perp & \iff \text{False} \\ \text{semics } I \text{ rel } (p \rightarrow q) & \iff \text{rel } I \text{ } p \longrightarrow \text{rel } I \text{ } q \end{aligned}$$

Table 8.6: Assumptions for proving a truth lemma (wrt. tables 8.2 and 8.3).

- A semantics with a hole in it:
 $semics :: 'model \Rightarrow ('model \Rightarrow 'fm \Rightarrow bool) \Rightarrow 'fm \Rightarrow bool$
- A semantics: $semantics :: 'model \Rightarrow 'fm \Rightarrow bool$
- The MCS-induced models: $models-from :: 'a set \Rightarrow 'model set$
- A relation between MCSs, models and formulas:
 $rel :: 'a set \Rightarrow 'model \Rightarrow 'fm \Rightarrow bool$
- Filling the hole in *semics* with *semantics* gives the semantics:
 $semantics M p \longleftrightarrow semics M semantics p$
- Hintikka sets give rise to satisfying models:
 $\bigwedge H M p. \forall M \in models-from H. \forall p. semics M (rel H) p \longleftrightarrow rel H M p$
 $\implies M \in models-from H \implies semantics M p \longleftrightarrow rel H M p$
- Maximal consistent saturated sets are Hintikka sets:
 $\bigwedge H. consistent H \implies maximal H \implies saturated H \implies$
 $\forall M \in models-from H. \forall p. semics M (rel H) p \longleftrightarrow rel H M p$

We noticed above that the model induced by an MCS/Hintikka set for our logic is $I^H P = (P \in H)$. The truth lemma we want to prove should hold for formulas in the MCS, so we should take $rel^H I^H p = (p \in H)$. The following equation then simplifies to the manual Hintikka set above:

$$semics I^H rel^H p \longleftrightarrow rel^H I^H p$$

In the propositional symbol case it reduces to the trivial $P \in H \longleftrightarrow P \in H$. In the falsity case we get the condition $False \longleftrightarrow \perp \in H$ which is equivalent to $\perp \notin H$. In the implication case we get $p \in H \longrightarrow q \in H \longleftrightarrow p \rightarrow q \in H$.

For propositional logic, the MCS shows up only once in the canonical model I^H , but for modal logic, the canonical model consists of a set of MCSs, where we need to know that all of them are Hintikka sets. This motivates the following extension to the equation, where $\mathbb{M}(H)$ are the models induced by the MCS H :

$$\forall M \in \mathbb{M}(H). \quad semics M rel^H p \longleftrightarrow rel^H M p$$

In the examples, I demonstrate that this *Hintikka equation* works for first-order logic, modal logic and hybrid logic too.

8.4.2 Formalization in Isabelle/HOL

Table 8.6 formalizes the informal process above. We need a semantics with a hole in it, *semics*, and one without, *semantics*. Then we need the models arising from an MCS, *models-from*, and a relation between MCS, models and formulas, *rel*. The semantics with a hole in it should be faithful to the intact one. If a set H behaves Hintikka-like wrt. all models arising from it, i.e. *rel* holds on a formula exactly when it holds on its subformulas, then *rel* should correspond with the semantics on H . Finally, all maximal consistent saturated sets should behave Hintikka-like with respect to the models they induce.

Note that the type ' a ' is what the MCSs consist of, while ' fm ' is the formula type supported by the semantics. These can differ, as for the hybrid logic calculus below where we build MCSs of labelled formulas, but evaluate plain formulas.

THEOREM 8.17 (TRUTH LEMMA) *Assume table 8.6, a maximal consistent saturated set H and a model M based on H . A formula p is true under M exactly when $rel\ H\ M\ p$.*

assumes *consistent H*
and *maximal H*
and *saturated H*
and $M \in models-from\ H$
shows $semantics\ M\ p \longleftrightarrow rel\ H\ M\ p$

PROOF. By assumption, the MCS forms a Hintikka-like set wrt. *rel*, which, by assumption, has a model. \square

It is evident from the proof above, that I am using the Isabelle locale differently from earlier: here it is simply used for structuring a proof. The use of the locale has a technical advantage, however: it allows Isabelle to write out the Hintikka equation for us, after we have provided its components. The method works for MCSs without saturation by simply omitting the assumptions of saturation.

8.5 Examples

In this section I introduce the languages and proof systems along with their consistency predicates, before I dive into each logic and prove strong completeness using the framework above.

8.5.1 Languages and Proof Systems

Table 8.7 lists the languages of the five logics I have instantiated the framework with. \ddagger serves as formal markers for propositional symbols P and likewise for \cdot and nominals i, k . For first-order logic, \ddagger marks predicate symbols P with a list of argument terms ts , $\#$ marks variables n and \dagger marks function symbols f with argument terms ts . Here, the variables are natural numbers used as de Bruijn indices. The modal logic is really a multimodal logic with an indexed necessity operator, but I will not emphasize this point further. The types of propositional symbols, predicate symbols, function symbols and nominals are all parametric and can be instantiated to anything from strings to predicates on real numbers.

Table 8.8 lists the consistency predicates for the five examples. In all cases, it says that a set is consistent when no list of elements from that set can be refuted (tableau) or used to derive a contradiction (the rest). The definitions rely on the concrete proof systems in tables 8.10 to 8.14. It is easy to prove by rule induction, that each proof system is sound for that logic's semantics. Note that the hybrid logic proof system uses labelled formulas, pairs of a nominal and a formula, rather than plain formulas. The framework supports this.

Table 8.9 lists the witness functions for first-order and hybrid logic. They use Hilbert's choice operator (*SOME*) to pick fresh parameters for the witnesses.

It is not hard to prove for each example that it lives up to the assumptions in table 8.2 and, for first-order logic and hybrid logic, those in table 8.3. Note that the *params* of a first-order logic formula is its set of function symbols and the *params* of a hybrid logic formula is its set of nominals. We can therefore build maximal consistent (saturated) sets for each logic. Likewise, the tableau system in table 8.10 meets the requirements for refutational systems in table 8.4 and the rest of the proof systems meet all the derivational requirements in table 8.5. This allows us to use theorems 8.14 to 8.15 about the presence and absence of formulas in maximal consistent sets. The next sections cover how to prove the truth lemma in each case.

Table 8.7: The languages of the five examples.

Logic	Syntax
Prop. Logic (Tableau)	$\ddagger P \mid \neg p \mid p \longrightarrow p$
Prop. Logic (Seq. Calc.)	$\ddagger P \mid \perp \mid p \longrightarrow p$
First-Order Logic	$\ddagger P \ ts \mid \perp \mid p \longrightarrow p \mid \forall p \quad (\text{and } \#n \mid \dagger f \ ts)$
Modal Logic	$\ddagger P \mid \perp \mid p \longrightarrow p \mid \Box i \ p$
Hybrid Logic	$\ddagger P \mid \cdot i \mid \perp \mid p \longrightarrow p \mid \Diamond p \mid @i \ p$

Table 8.8: Consistency predicates for the five examples (cf. tables 8.10 to 8.14).

Logic	Consistency predicate
Prop. Logic (Tableau)	$\text{consistent } S \equiv \nexists S'. \text{ set } S' \subseteq S \wedge \vdash_T S'$
Prop. Logic (S. C.)	$\text{consistent } S \equiv \nexists S'. \text{ set } S' \subseteq S \wedge S' \vdash_S [\perp]$
First-Order Logic	$\text{consistent } S \equiv \nexists S'. \text{ set } S' \subseteq S \wedge S' \vdash_{\forall} \perp$
Modal Logic	$\text{consistent } S \equiv \nexists S'. \text{ set } S' \subseteq S \wedge S' \vdash_{\Box} \perp$
Hybrid Logic	$\text{consistent } S \equiv \nexists S' a. \text{ set } S' \subseteq S \wedge S' \vdash_{@} (a, \perp)$

Table 8.9: Witnesses for first-order and hybrid logic.

Logic	Witness function
FOL	$\text{witness } (\neg (\forall p)) S = \{\neg \langle \star(\text{SOME } a. a \notin \text{params}'(\{p\} \cup S)) \rangle p\}$
HL	$\text{witness } (i, \Diamond p) S = (\text{let } k = (\text{SOME } k. k \notin \text{nominals } (\{(i, p)\} \cup S)) \text{ in } \{(k, p), (i, \Diamond (\cdot k))\})$

8.5.2 Propositional Tableau

Table 8.10 gives the tableau rules as an inductive predicate ($\vdash_T - :: 'p \text{ fm list} \Rightarrow \text{bool}$) that states whether a (conjunctive) list of formulas can be refuted.

The semantics for this example are based on interpretations I of the propositional variables. Semantic brackets turn the interpretation into a predicate on formulas:

$$\begin{aligned}
\llbracket I \rrbracket (\nexists P) &\longleftrightarrow I P \\
\llbracket I \rrbracket (\neg p) &\longleftrightarrow \neg \llbracket I \rrbracket p \\
\llbracket I \rrbracket (p \longrightarrow q) &\longleftrightarrow \llbracket I \rrbracket p \longrightarrow \llbracket I \rrbracket q
\end{aligned}$$

We define the Hintikka sets manually, with the goal of having them model irrefutable lists. Each proof rule, other than *Weak*, has a dual Hintikka condition:

$$\begin{aligned}
\text{AxiomH: } &\bigwedge P. \nexists P \in H \Longrightarrow \neg \nexists P \in H \Longrightarrow \text{False} \\
\text{NegIH: } &\bigwedge p. \neg \neg p \in H \Longrightarrow p \in H \\
\text{ImpPH: } &\bigwedge p q. p \longrightarrow q \in H \Longrightarrow \neg p \in H \vee q \in H \\
\text{ImpNH: } &\bigwedge p q. \neg (p \longrightarrow q) \in H \Longrightarrow p \in H \wedge \neg q \in H
\end{aligned}$$

For the model lemma we use the interpretation $hmodel H \equiv \lambda P. \nexists P \in H$.

LEMMA 8.18 (HINTIKKA MODEL) *The Hintikka set H is satisfiable.*

assumes *Hintikka H*

shows $(p \in H \longrightarrow \llbracket hmodel H \rrbracket p) \wedge (\neg p \in H \longrightarrow \neg \llbracket hmodel H \rrbracket p)$

PROOF. By structural induction on the formula. □

Table 8.10: Propositional tableau proof system.

<i>Axiom:</i>	$\vdash_T \ddagger P \# \neg \ddagger P \# A$
<i>NegI:</i>	$\vdash_T p \# A \implies \vdash_T \neg \neg p \# A$
<i>ImpP:</i>	$\vdash_T \neg p \# A \implies \vdash_T q \# A \implies \vdash_T (p \longrightarrow q) \# A$
<i>ImpN:</i>	$\vdash_T p \# \neg q \# A \implies \vdash_T \neg (p \longrightarrow q) \# A$
<i>Weaken:</i>	$\vdash_T A \implies \text{set } A \subseteq \text{set } B \implies \vdash_T B$

Table 8.11: Propositional sequent calculus proof system.

<i>Axiom:</i>	$p \# A \vdash_S p \# B$
<i>FlsL:</i>	$\perp \# A \vdash_S B$
<i>FlsR:</i>	$A \vdash_S \perp \# B \implies A \vdash_S B$
<i>ImpL:</i>	$A \vdash_S p \# B \implies q \# A \vdash_S B \implies (p \longrightarrow q) \# A \vdash_S B$
<i>ImpR:</i>	$p \# A \vdash_S q \# B \implies A \vdash_S (p \longrightarrow q) \# B$
<i>Cut:</i>	$A \vdash_S p \# B \implies p \# C \vdash_S D \implies A @ C \vdash_S B @ D$
<i>WeakenL:</i>	$A \vdash_S B \implies \text{set } A \subseteq \text{set } A' \implies A' \vdash_S B$
<i>WeakenR:</i>	$A \vdash_S B \implies \text{set } B \subseteq \text{set } B' \implies A \vdash_S B'$

Table 8.12: First-order logic natural deduction proof system.

<i>Assm:</i>	$p \in \text{set } A \implies A \vdash_{\forall} p$
<i>FlsE:</i>	$A \vdash_{\forall} \perp \implies A \vdash_{\forall} p$
<i>ImpI:</i>	$p \# A \vdash_{\forall} q \implies A \vdash_{\forall} p \longrightarrow q$
<i>ImpE:</i>	$A \vdash_{\forall} p \longrightarrow q \implies A \vdash_{\forall} p \implies A \vdash_{\forall} q$
<i>UniI:</i>	$A \vdash_{\forall} \langle \star a \rangle p \implies a \notin \text{params } (p \# A) \implies A \vdash_{\forall} \forall p$
<i>UniE:</i>	$A \vdash_{\forall} \forall p \implies A \vdash_{\forall} \langle t \rangle p$
<i>Clas:</i>	$(p \longrightarrow q) \# A \vdash_{\forall} p \implies A \vdash_{\forall} q$
<i>Weak:</i>	$A \vdash_{\forall} p \implies q \# A \vdash_{\forall} p$

params returns the set of all function symbols in a set of formulas.

$\langle t \rangle p$ instantiates the formula p with the term t . $\star a$ is shorthand for $\dagger a$ [].

Table 8.13: Modal logic system K proof system.

<i>A1:</i>	<i>tautology</i> $p \implies \vdash_{\Box} p$
<i>A2:</i>	$\vdash_{\Box} \Box i (p \longrightarrow q) \longrightarrow \Box i p \longrightarrow \Box i q$
<i>R1:</i>	$\vdash_{\Box} p \implies \vdash_{\Box} p \longrightarrow q \implies \vdash_{\Box} q$
<i>R2:</i>	$\vdash_{\Box} p \implies \vdash_{\Box} \Box i p$

tautology is true for propositional tautologies.

Table 8.14: Hybrid logic labelled natural deduction proof system.

<i>Assm</i> :	$(i, p) \in \text{set } A \implies A \vdash_{\mathbb{Q}} (i, p)$
<i>Ref</i> :	$A \vdash_{\mathbb{Q}} (i, \cdot i)$
<i>Nom</i> :	$A \vdash_{\mathbb{Q}} (i, \cdot k) \implies A \vdash_{\mathbb{Q}} (i, p) \implies A \vdash_{\mathbb{Q}} (k, p)$
<i>FlsE</i> :	$A \vdash_{\mathbb{Q}} (i, \perp) \implies A \vdash_{\mathbb{Q}} (k, p)$
<i>ImpI</i> :	$(i, p) \# A \vdash_{\mathbb{Q}} (i, q) \implies A \vdash_{\mathbb{Q}} (i, p \longrightarrow q)$
<i>ImpE</i> :	$A \vdash_{\mathbb{Q}} (i, p \longrightarrow q) \implies A \vdash_{\mathbb{Q}} (i, p) \implies A \vdash_{\mathbb{Q}} (i, q)$
<i>SatI</i> :	$A \vdash_{\mathbb{Q}} (i, p) \implies A \vdash_{\mathbb{Q}} (k, @i p)$
<i>SatE</i> :	$A \vdash_{\mathbb{Q}} (i, @k p) \implies A \vdash_{\mathbb{Q}} (k, p)$
<i>DiaI</i> :	$A \vdash_{\mathbb{Q}} (i, \Diamond (\cdot k)) \implies A \vdash_{\mathbb{Q}} (k, p) \implies A \vdash_{\mathbb{Q}} (i, \Diamond p)$
<i>DiaE</i> :	$A \vdash_{\mathbb{Q}} (i, \Diamond p) \implies k \notin \text{nominals } (\{(i, p), (j, q)\} \cup \text{set } A) \implies$ $(k, p) \# (i, \Diamond (\cdot k)) \# A \vdash_{\mathbb{Q}} (j, q) \implies A \vdash_{\mathbb{Q}} (j, q)$
<i>Clas</i> :	$(i, p \longrightarrow q) \# A \vdash_{\mathbb{Q}} (i, p) \implies A \vdash_{\mathbb{Q}} (i, p)$
<i>Weak</i> :	$A \vdash_{\mathbb{Q}} (i, p) \implies (k, q) \# A \vdash_{\mathbb{Q}} (i, p)$

nominals returns the nominals in a set of labelled formulas.

It is harder to see that MCSs are Hintikka sets.

LEMMA 8.19 (MCSs ARE HINTIKKA) *If H is a maximal consistent set, then it satisfies all Hintikka conditions.*

assumes consistent H and maximal H

shows Hintikka H

PROOF. *AxiomH* is fulfilled since the list $[\ddagger P, \neg \ddagger P]$ is refutable with the *Axiom* rule, so any set with both elements must be inconsistent, but H is consistent.

NegIH asks us to assume $\neg \neg p \in H$ and prove $p \in H$. If $p \notin H$ then by theorem 8.14, we can refute p, H' where H' is a list of elements from H . With *NegI*, we can then refute $\neg \neg p, H'$. This contradicts the assumption $\neg \neg p \in H$.

ImpPH and *ImpNH* have similar proofs. Assume that the premise formula is in H and assume towards a contradiction that the conclusion formula is not. Use theorem 8.14 to obtain a refutation with the conclusion formula and use the proof rules to refute the premise formula instead. \square

The proof above demonstrates the utility of theorem 8.14: it connects the MCS with the proof system in exactly the way we need.

Strong completeness follows in the usual way:

THEOREM 8.20 (THE TABLEAU SYSTEM IS STRONGLY COMPLETE)

If p is valid under assumptions X , then we can refute $\neg p$ and a list of elements

from X .

assumes $\forall M :: 'p \text{ model. } (\forall q \in X. \llbracket M \rrbracket q) \longrightarrow \llbracket M \rrbracket p$
shows $\exists A. \text{ set } A \subseteq X \wedge \vdash_T \neg p \# A$

PROOF. Assume towards a contradiction that there is no such list of elements refutable in conjunction with $\neg p$. Then the set $\{\neg p\} \cup X$ is consistent and can be extended to an MCS by theorem 8.13. This gives us a model for $\neg p$ and all of X , contradicting the validity of p under X . \square

We made no assumptions about the type $'p$ of propositional symbols in the above theorem. This means that we have shown completeness regardless of what we instantiate the type with: strings, real numbers, sets of sets of real numbers, etc. are all possibilities.

8.5.3 Propositional Sequent Calculus

Table 8.11 gives the rules of the two-sided sequent calculus in the form of an inductive predicate ($- \vdash_S - :: 'p \text{ fm list} \Rightarrow 'p \text{ fm list} \Rightarrow \text{bool}$) that tells us whether a (conjunctive) list of premises implies a (disjunctive) list of conclusions.

The semantics are again based on interpretations I of the propositional variables. The language is slightly different this time. Semantic brackets still turn an interpretation into a predicate on formulas:

$$\begin{aligned} \llbracket I \rrbracket \perp & \longleftrightarrow \text{False} \\ \llbracket I \rrbracket (\ddagger P) & \longleftrightarrow I P \\ \llbracket I \rrbracket (p \longrightarrow q) & \longleftrightarrow \llbracket I \rrbracket p \longrightarrow \llbracket I \rrbracket q \end{aligned}$$

This time, we derive the Hintikka sets from the general equation:

$$\forall M \in \mathbb{M}(H). \text{ semics } M P^H p \longleftrightarrow P^H M p$$

The semantics with a hole in it looks like this:

$$\begin{aligned} \text{semics } I \text{ rel } \perp & \longleftrightarrow \text{False} \\ \text{semics } I \text{ rel } (\ddagger P) & \longleftrightarrow I P \\ \text{semics } I \text{ rel } (p \longrightarrow q) & \longleftrightarrow \text{rel } I p \longrightarrow \text{rel } I q \end{aligned}$$

We use the same MCS-induced model as for tableau, $hmodel H \equiv \lambda P. \ddagger P \in H$, and instantiate the relation with $\text{rel } H - p = (p \in H)$. Inserting these into the general equation gives us:

$$\forall M \in \{hmodel H\}. \text{ semics } M (\text{rel } H) p \longleftrightarrow \text{rel } H M p$$

A little bit of simplification yields the following Hintikka equation:

$$\text{semics } (hmodel\ H) \ (rel\ H) \ p \longleftrightarrow p \in H$$

We need to show that we can satisfy formulas in sets H that satisfy this equation.

LEMMA 8.21 (HINTIKKA MODEL) *Assume H satisfies the Hintikka equation. Any formula p is in H if and only if it is satisfied by $hmodel\ H$.*

assumes $\bigwedge p. \text{semics } (hmodel\ H) \ (rel\ H) \ p \longleftrightarrow p \in H$

shows $p \in H \longleftrightarrow \llbracket hmodel\ H \rrbracket p$

PROOF. By structural induction on p .

For \perp we need to prove $\perp \notin H$. This is exactly what our assumption gives us.

For $\nexists P$, we need to prove $\nexists P \in H$ if and only if $\nexists P \in H$, which is trivial.

For $p \longrightarrow q$, we need to prove $p \longrightarrow q \in H$ if and only if $\llbracket hmodel\ H \rrbracket p$ implies $\llbracket hmodel\ H \rrbracket q$. The assumption gives us that $p \in H \longrightarrow q \in H$ if and only if $p \longrightarrow q \in H$. Here, we can use the induction hypothesis to rewrite $p \in H$ to $\llbracket hmodel\ H \rrbracket p$ and $q \in H$ to $\llbracket hmodel\ H \rrbracket q$, giving us exactly what we need. \square

In practice, Isabelle/HOL can prove this lemma automatically and its formulation is given to us when we instantiate the locale described by table 8.6. The equation is directed in the way it is to avoid making Isabelle's simplifier loop.

We now need to prove the other half of the truth lemma.

LEMMA 8.22 (MCSSs ARE HINTIKKA) *Assume H is a maximal consistent set. Then the Hintikka equation holds for H .*

assumes *consistent H and maximal H*

shows $\text{semics } (hmodel\ H) \ (rel\ H) \ p \longleftrightarrow p \in H$

PROOF. By cases on the formula p .

For \perp , we need to show $\perp \notin H$. This follows from the consistency of H .

For $\nexists P$, we need to show $\nexists P \in H \longleftrightarrow \nexists P \in H$ which is trivial.

For $p \longrightarrow q$, we need to show that $p \in H \longrightarrow q \in H$ if and only if $p \longrightarrow q \in H$. We show both directions.

Assume $p \in H \longrightarrow q \in H$. We proceed by cases on whether $p \in H$. When $p \in H$, we have $q \in H$. By theorem 8.16 in one direction, we can then derive q

from some list of formulas from H . Then we can also derive $p \longrightarrow q$ from that list using the *ImpR* and *WeakenL* rules. By theorem 8.16 in the other direction, we must therefore have $p \longrightarrow q \in H$. When $p \notin H$, by theorem 8.15, we must be able to derive \perp from p and some list of elements from H . But then we can also derive $p \longrightarrow q$ from this list using the *FlsR*, *ImpR* and *WeakenR* rules. By theorem 8.16, $p \longrightarrow q$ must be in H .

Assume $p \longrightarrow q \in H$ and $p \in H$. By theorem 8.16 and the *Cut* rule, $q \in H$. \square

This proof shows how useful theorems 8.15 and 8.16 are.

THEOREM 8.23 (THE SEQUENT CALC. IS STRONGLY COMPLETE)

If the formula p is valid under assumptions X , then we can derive p from a list of formulas from X .

assumes $\forall M :: 'p \text{ model. } (\forall q \in X. \llbracket M \rrbracket q) \longrightarrow \llbracket M \rrbracket p$

shows $\exists A. \text{ set } A \subseteq X \wedge A \vdash_S [p]$

PROOF. Analogous to the proof of theorem 8.20. \square

8.5.4 First-Order Logic

Table 8.12 gives the natural deduction rules in the form of an inductive predicate $(- \vdash_{\forall} - :: ('f, 'p) \text{ fm list} \Rightarrow ('f, 'p) \text{ fm} \Rightarrow \text{bool})$ that tells us whether a formula can be derived from a list of assumptions.

Having a list of assumptions is somewhat strict compared to having a multiset or similar. The only structural rule that I have included directly is weakening with a formula at the front of the list (*Weak*). From this, it is easy to prove that we can also permute and contract elements using the following trick with the *ImpI* and *ImpE* rules.

LEMMA 8.24 (STRUCTURAL RULES) *If we can derive p from assumptions A then we can also derive p from any list B that contains all elements in A .*

shows $A \vdash_{\forall} p \Longrightarrow \text{set } A \subseteq \text{set } B \Longrightarrow B \vdash_{\forall} p$

PROOF. By induction on the list of assumptions A for an arbitrary p .

In the base case, we can derive p from no assumptions. By repeated invocations of the *Weak* rule (a nested induction on B) we can derive p from B as desired.

In the inductive case, we can derive p from the list q, A for some formula q and the induction hypothesis says that anything we can derive from A alone, we can derive from B . We can derive $q \longrightarrow p$ from A using the *ImpI* rule, so we can derive $q \longrightarrow p$ from B . Using *Assm* we can also derive q from B , since B is a superset of q, A . By the *ImpE* rule, we can then derive p from B as desired. \square

The semantics of our first-order logic is based on an implicit domain and a triple of denotations: a variable denotation E , a function denotation F and a predicate denotation G . I use semantic parentheses to turn a variable and function denotation into a function from terms to the domain:

$$\begin{aligned} \llbracket (E, F) \rrbracket (\#n) &= E\ n \\ \llbracket (E, F) \rrbracket (\dagger f\ ts) &= F\ f\ (\text{map } \llbracket (E, F) \rrbracket\ ts) \end{aligned}$$

In the function case, the definition uses *map* over lists to recursively evaluate the subterms. I use semantic brackets to turn the triple of denotations into a predicate on formulas:

$$\begin{aligned} \llbracket (E, F, G) \rrbracket \perp &\longleftrightarrow \text{False} \\ \llbracket (E, F, G) \rrbracket (\dagger P\ ts) &\longleftrightarrow G\ P\ (\text{map } \llbracket (E, F) \rrbracket\ ts) \\ \llbracket (E, F, G) \rrbracket (p \longrightarrow q) &\longleftrightarrow \llbracket (E, F, G) \rrbracket p \longrightarrow \llbracket (E, F, G) \rrbracket q \\ \llbracket (E, F, G) \rrbracket (\forall p) &\longleftrightarrow (\forall x. \llbracket (x \circ E, F, G) \rrbracket p) \end{aligned}$$

In the quantifier case, $x \circ E$ adds the element x to the front of the denotation E such that variable 0 maps to x and any variable $n + 1$ maps to $E\ n$, taking into account that we crossed a binder.

The semantics with a hole in it becomes, as expected, the following:

$$\begin{aligned} \text{semics } (E, F, G) \text{ rel } \perp &\longleftrightarrow \text{False} \\ \text{semics } (E, F, G) \text{ rel } (\dagger P\ ts) &\longleftrightarrow G\ P\ (\text{map } \llbracket (E, F) \rrbracket\ ts) \\ \text{semics } (E, F, G) \text{ rel } (p \longrightarrow q) &\longleftrightarrow \text{rel } (E, F, G) p \longrightarrow \text{rel } (E, F, G) q \\ \text{semics } (E, F, G) \text{ rel } (\forall p) &\longleftrightarrow (\forall x. \text{rel } (x \circ E, F, G) p) \end{aligned}$$

For the MCS-induced model, we use the term model based on the variable constructor $\#$ and function constructor \dagger :

$$\text{hmodel } H \equiv (\#, \dagger, \lambda P\ ts. \dagger P\ ts \in H)$$

Terms evaluate to themselves under this denotation: $\llbracket (\#, \dagger) \rrbracket t = t$. This allows us to evaluate predicates by simply looking them up in the MCS.

To write up the Hintikka equation, we need a relation to plug the hole in the semantics with. We could take $\text{rel } H\ (E, F, G) p = (p \in H)$, but this would not

work, as we are throwing away information. In the first-order logic semantics, we cannot understand a formula in isolation but need to take the variable denotation into account since it carries information from the binders we have met so far. In the term model, the variable denotation is a function from variables to terms, so we can apply it as a substitution. The correct relation becomes:

$$\text{rel } H (E, -, -) p = (\text{sub-fm } E p \in H)$$

The function *sub-fm* simply applies a substitution to a formula in the proper, capture-avoiding way.

The Hintikka equation becomes:

$$\text{semics } (hmodel H) (\text{rel } H) p \longleftrightarrow \text{sub-fm } \# p \in H$$

It is easy to prove that $\text{sub-fm } \# p = p$, which simplifies the right-hand side to $p \in H$. We have seen by now that this equation works for falsity, predicates and implication. For the universal quantifier it reduces to:

$$(\forall t. \langle t \rangle p \in H) \longleftrightarrow (\forall p) p \in H$$

That is, a universally quantified formula appears in the Hintikka set if and only if all its instances do too. This is perfect for the Hintikka model lemma.

LEMMA 8.25 (HINTIKKA MODEL) *Assume H satisfies the Hintikka equation. Any formula p is in H if and only if it is satisfied by $hmodel H$.*

assumes $\bigwedge p. \text{semics } (hmodel H) (\text{rel } H) p \longleftrightarrow p \in H$

shows $p \in H \longleftrightarrow \llbracket hmodel H \rrbracket p$

PROOF. By structural induction on p . □

Maximal consistent saturated sets satisfy the Hintikka equation.

LEMMA 8.26 (MCSSs ARE HINTIKKA) *Assume H is a maximal consistent saturated set. Then the Hintikka equation holds for H .*

assumes *consistent H and maximal H and saturated H*

shows $\text{semics } (hmodel H) (\text{rel } H) p \longleftrightarrow p \in H$

PROOF. By cases on the formula p . Falsity, predicates and implication are as in the proof of lemma 8.22 for sequent calculus. We prove both directions of the quantifier case.

Assume first that $\forall t. \langle t \rangle p \in H$. Assume towards a contradiction that $(\forall p) p \notin H$. Then $\{\neg \forall p\} \cup H$ must be consistent, so by maximality $\{\neg \forall p\} \in H$. By saturation, this existential formula has a witness which is also in H : $\neg \langle \star a \rangle p \in H$

for some a . But by our assumption, $\langle t \rangle p \in H$ for all terms t , including $\star a$. This contradicts the consistency of H .

Assume next that $(\forall p) \in H$. We need to show $\forall t. \langle t \rangle p \in H$. It is easy to prove that when $\forall p$ is in a consistent set, it is consistent to also add any instance $\langle t \rangle p$ since the instance can already be derived from the quantified formula using the *UniE* rule. From here, maximality proves the thesis. \square

Strong completeness follows as earlier, but with extra concerns about cardinalities. Since we rely on saturated MCSs, we need to ensure we have enough parameters available. Theorem 8.12 asks that the cardinality of unused parameters is at least the cardinality of the universe of formulas, but what exactly is the cardinality of the universe of formulas? Note that our first-order logic syntax can be written down, linearly, as a list of symbols. The set of lists over some type has the same cardinality as the type itself, since lists are finite. We need finitely many symbols for parentheses and infix operators, so what actually contributes to the cardinality is function symbols, predicate symbols and de Bruijn indices (natural numbers). Natural numbers have the lowest infinite cardinality, so assuming we have infinitely many function symbols, we can disregard the natural numbers. This means that the cardinality of our formulas is bounded by the sum of the cardinalities of the function and predicate symbols.

THEOREM 8.27 (THE FOL CALCULUS IS STRONGLY COMPLETE) *Assume p is valid under assumptions X , that the universe of function symbols is infinite and the cardinality of unused parameters in X is at least the sum of the cardinalities of function and predicate symbols. Then we can derive p from a list of formulas from X .*

assumes $\forall M :: ('f\ tm, 'f, 'p)\ model. (\forall q \in X. \llbracket M \rrbracket q \longrightarrow \llbracket M \rrbracket p$
and *infinite* ($UNIV :: 'f\ set$)
and $|UNIV :: 'f\ set| + c\ |UNIV :: 'p\ set| \leq o\ |UNIV - params'\ X|$
shows $\exists A. set\ A \subseteq X \wedge A \vdash_{\forall} p$

PROOF. Analogous to the proof of theorem 8.20. \square

Weak completeness allows us to state the cardinality assumption in a slightly neater fashion. The predicate $valid\ p \equiv \forall M :: ('f\ tm, -, -)\ model. \llbracket M \rrbracket p$ simply abbreviates validity in the correct domain.

COROLLARY 8.28 (THE FOL CALCULUS IS COMPLETE) *If p is a valid formula, the universe of function symbols is infinite and there are at least as many function symbols as predicate symbols, then we can derive p .*

assumes *valid p*
 and *infinite (UNIV :: 'f set)*
 and $|UNIV :: 'p \text{ set}| \leq_o |UNIV :: 'f \text{ set}|$
shows $\Box \vdash_{\forall} p$

PROOF. With an empty set of assumptions, to apply theorem 8.27 we simply need to prove that the universe of function symbols is at least as large as the sum of function and predicate symbols. This holds when there are no more predicate symbols than function symbols. \square

We can do away with the cardinality assumptions completely by using the same type for function and predicate symbols. Then we just need it to be infinite.

COROLLARY 8.29 *If p is valid and the universe of function/predicate symbols is infinite, then we can derive p.*

fixes $p :: ('f, 'f) \text{ fm}$
assumes *valid p and infinite (UNIV :: 'f set)*
shows $\Box \vdash_{\forall} p$

8.5.5 Modal Logic

Table 8.13 gives the axiomatic proof system in the form of an inductive predicate ($\vdash_{\Box} - :: ('i, 'p) \text{ fm} \Rightarrow \text{bool}$) that tells us whether a formula can be derived. In practice, I use lists of assumptions to mimic a natural deduction system:

$A \vdash_{\Box} p \equiv \vdash_{\Box} A \leadsto p$

Here, the arrow, \leadsto , builds a chain of implications from the list on the left to the formula on the right.

The semantics is based on Kripke models consisting of a set of worlds, an accessibility relation for each agent and a valuation of propositional symbols at each world:

datatype $('i, 'p, 'w) \text{ model} =$
 $\text{Model } (\mathcal{W}: 'w \text{ set}) (\mathcal{R}: 'i \Rightarrow 'w \Rightarrow 'w \text{ set}) (\mathcal{V}: 'w \Rightarrow 'p \Rightarrow \text{bool})$

I use an explicit set of worlds to avoid subtypes later, where we will use the set of all MCSs as the set of worlds.

Formulas are evaluated in a context consisting of a model and a world:

$$\begin{aligned}
(M, w) \models \perp & \longleftrightarrow \text{False} \\
(M, w) \models \nexists P & \longleftrightarrow \mathcal{V} M w P \\
(M, w) \models p \longrightarrow q & \longleftrightarrow (M, w) \models p \longrightarrow (M, w) \models q \\
(M, w) \models \Box i p & \longleftrightarrow (\forall v \in \mathcal{W} M \cap \mathcal{R} M i w. (M, v) \models p)
\end{aligned}$$

For the necessity operator, \Box , we make sure to only quantify over worlds that are both in the model and reachable by the agent. We could also assume wellformed models, where agents can only reach worlds that are actually in the model, but this trick saves us that worry.

The semantics with a hole in it becomes the following:

$$\begin{aligned}
\text{semics } (M, w) \text{ rel } \perp & \longleftrightarrow \text{False} \\
\text{semics } (M, w) \text{ rel } (\nexists P) & \longleftrightarrow \mathcal{V} M w P \\
\text{semics } (M, w) \text{ rel } (p \longrightarrow q) & \longleftrightarrow \text{rel } (M, w) p \longrightarrow \text{rel } (M, w) q \\
\text{semics } (M, w) \text{ rel } (\Box i p) & \longleftrightarrow (\forall v \in \mathcal{W} M \cap \mathcal{R} M i w. \text{rel } (M, v) p)
\end{aligned}$$

We base the MCS-induced models on the usual *canonical* model [2]:

- The set of worlds consists of all MCSs (*mcss*).
- World W is reachable from V by agent i if everything that i finds necessary at V is in W : $\{p. \Box i p \in V\} \subseteq W$.
- Proposition P holds at world V if $P \in V$.

Since we have a whole set of MCSs this time, we want Hintikka-like qualities for all them. Our *semics* function takes a complete context of model and world, so we can achieve this by pairing the canonical model with every MCS (ignoring the concrete MCS that was given to us):

$$\text{models-from} = \lambda-. \{(canonical, V) \mid V. V \in mcss\}$$

When plugging the hole in the *sem(ant)ics*, we again ignore the given set H and instead consider the world in the current context:

$$\text{rel} - (-, w) p = (p \in w)$$

The Hintikka equation becomes, after a bit of simplification:

$$\forall V \in mcss. \text{semics } (canonical, V) (\text{rel } H) p \longleftrightarrow p \in V$$

This equation reduces to the usual cases for propositions, falsity and implication. For the necessity operator, it reduces to:

$$\forall V \in mcss. (\forall W \in mcss. \{q. \Box i q \in V\} \subseteq W \longrightarrow p \in W) \longleftrightarrow (\Box i p) \in V$$

This states that a boxed formula $\Box i p$ occurs in a world/MCS V exactly when p occurs in all the worlds reachable from V with the canonical accessibility relation. The equation is derived from the semantics, so the model lemma is trivial.

LEMMA 8.30 (HINTIKKA MODEL) *Given the Hintikka equation, the formula p is satisfied at V in the canonical model exactly when $p \in V$.*

assumes $\bigwedge (V :: ('i, 'p) \text{ fm set}) p. V \in \text{mcss} \implies$
 $\text{semics } (\text{canonical}, V) (\text{rel } H) p \longleftrightarrow p \in V$
shows $V \in \text{mcss} \implies (\text{canonical}, V) \models p \longleftrightarrow p \in V$

PROOF. By structural induction on the formula. □

Any maximal consistent set satisfies the Hintikka equation.

LEMMA 8.31 (MCSSs ARE HINTIKKA) *Assume V is a maximal consistent set. Then the Hintikka equation holds for V .*

assumes $V \in \text{mcss}$
shows $\text{semics } (\text{canonical}, V) (\text{rel } H) p \longleftrightarrow \text{rel } H (\text{canonical}, V) p$

PROOF. By cases on the formula p . Falsity, predicates and implication are as in the proof of lemma 8.22 for sequent calculus. The modal case follows the proof by Fagin et al. [2]. □

We easily obtain strong completeness.

THEOREM 8.32 (SYSTEM K IS STRONGLY COMPLETE) *If p is valid under assumptions X , then we can derive p from a list of formulas from X .*

assumes $\forall M :: ('i, 'p, ('i, 'p) \text{ fm set}) \text{ model}. \forall w \in \mathcal{W} M.$
 $(\forall q \in X. (M, w) \models q) \longrightarrow (M, w) \models p$
shows $\exists A. \text{ set } A \subseteq X \wedge A \vdash_{\Box} p$

PROOF. Analogous to the proof of theorem 8.20. □

8.5.6 Hybrid Logic

Table 8.14 gives the natural deduction proof system in the form of an inductive predicate $(- \vdash_{\text{@}} - :: ('i, 'p) \text{ lbd list} \Rightarrow ('i, 'p) \text{ lbd} \Rightarrow \text{bool})$ that tells us whether a labelled formula can be derived from a list of (labelled) assumptions. This system is inspired by Braüner's calculus [7]. The trick in lemma 8.24 to prove all the structural rules admissible from *Weak*, *ImpI* and *ImpE* applies here too, with

some additional use of *SatI* and *SatE* to shift nominals between the metalogical labels and the object logic.

The semantics is again based on Kripke models, but this time we implicitly use the type $'w$ as the set of worlds, since the canonical model we build will be over a full type, rather than a subtype:

datatype ($'w, 'p$) *model* =
Model ($R: 'w \Rightarrow 'w \text{ set}$) ($V: 'w \Rightarrow 'p \Rightarrow \text{bool}$)

Formulas are evaluated in a context consisting of a model, an assignment which maps nominals to worlds, and a world:

$$\begin{aligned}
 (M, g, w) \models \perp & \iff \text{False} \\
 (M, g, w) \models \ddagger P & \iff V M w P \\
 (M, g, w) \models \cdot i & \iff w = g i \\
 (M, g, w) \models (p \longrightarrow q) & \iff (M, g, w) \models p \longrightarrow (M, g, w) \models q \\
 (M, g, w) \models \Diamond p & \iff (\exists v \in R M w. (M, g, v) \models p) \\
 (M, g, w) \models @i p & \iff (M, g, g i) \models p
 \end{aligned}$$

For the modal operator, here \Diamond , we no longer need to worry about reaching worlds outside the model, since we use the full type $'w$. Other than that, we have two new cases compared to modal logic. A nominal i is true at a world w when the assignment g maps i to w , that is, when i is naming the world we are currently at. The satisfaction operator, $@i$ shifts evaluation to the world named by nominal i .

The semantics with a hole in it becomes the following:

$$\begin{aligned}
 \text{semics } (M, g, w) \text{ rel } \perp & \iff \text{False} \\
 \text{semics } (M, g, w) \text{ rel } (\ddagger P) & \iff V M w P \\
 \text{semics } (M, g, w) \text{ rel } (\cdot i) & \iff w = g i \\
 \text{semics } (M, g, w) \text{ rel } (p \longrightarrow q) & \iff \text{rel } (M, g, w) p \longrightarrow \text{rel } (M, g, w) q \\
 \text{semics } (M, g, w) \text{ rel } (\Diamond p) & \iff (\exists v \in R M w. \text{rel } (M, g, v) p) \\
 \text{semics } (M, g, w) \text{ rel } (@ i p) & \iff \text{rel } (M, g, g i) p
 \end{aligned}$$

For the MCS-induced models, we need to give a type of worlds, a reachability relation, a valuation and an assignment. We use representatives of equivalence classes of nominals as worlds (i.e. simply nominals). This is justified by the *Ref* and *Nom* rules which ensure that the following relation between nominals is an equivalence when H is a maximal consistent set:

definition $hequiv H i k \equiv (i, \cdot k) \in H$

Say that a formula p is derivable *at a nominal* i when (i, p) can be derived. Then, for equivalent nominals i and k , any formula derivable at i is also derivable at k .

The canonical assignment needs to assign to each nominal its corresponding world (also a nominal). It simply takes the smallest element of the set of equivalent nominals and uses it as representative:

definition $assign\ H\ i \equiv minim\ (|UNIV|)\ \{k. hequiv\ H\ i\ k\}$

This definition is possible due to the axiom of choice, with which we can wellorder any set and wellorders have minimum elements. Informally, I write $[i]$ for the representative of i 's equivalence class, with H being clear from the context.

Reachability is based on this assignment and the possibility operator:

definition $reach\ H\ i \equiv \{assign\ H\ k\ |k. (i, \Diamond (\cdot k)) \in H\}$

From world i , we can reach all worlds $[k]$ where $(i, \Diamond k) \in H$ witnesses the reachability.

The valuation is much simpler:

definition $val\ H\ i\ P \equiv (i, \Vdash P) \in H$

It is parameterized by the MCS H and simply says that proposition P holds at world i when this is witnessed by H .

This time, *canonical* gives the full context to evaluate a formula under:

definition $canonical\ H\ i \equiv (Model\ (reach\ H)\ (val\ H),\ assign\ H,\ assign\ H\ i)$

We just need one more definition to write up the Hintikka equation:

definition $rel\ H\ (-, -, i)\ p \equiv ((i, p) \in H)$

Once again, we cannot understand a formula out of context, so we pair it with the current world, a nominal.

Since we only have one model per MCS, we can simplify the Hintikka equation:

$$semics\ (canonical\ H\ i)\ (rel\ H)\ p \longleftrightarrow ([i], p) \in H$$

Let us once again look at the new cases and what the equation reduces to. For a nominal k , we get that the equivalence classes should be the same, exactly when this is witnessed by H .

$$[i] = [k] \longleftrightarrow ([i], k) \in H$$

For $\Diamond p$, we get that it occurs at $[i]$ in H , exactly when p can be found at some nominal k reachable from $[i]$:

$$(\exists k \in \text{reach } H [i]. (k, p) \in H) \longleftrightarrow ([i], \Diamond p) \in H$$

Finally, $\@k p$ occurs at $[i]$ in H , exactly when p occurs at $[k]$ in H :

$$([k], p) \longleftrightarrow ([i], \@k p) \in H$$

We reach the Hintikka model lemma.

LEMMA 8.33 (HINTIKKA MODEL) *If H satisfies the Hintikka equation for all nominals and formulas, then formula p is satisfied at i in the canonical model exactly when $([i], p) \in H$.*

assumes $\bigwedge i p. \text{semics } (\text{canonical } H \ i) \ (\text{rel } H) \ p \longleftrightarrow \text{rel } H \ (\text{canonical } H \ i) \ p$
shows $(\text{canonical } H \ i \models p) \longleftrightarrow \text{rel } H \ (\text{canonical } H \ i) \ p$

PROOF. By structural induction on the formula. □

We now need to show that a maximal consistent saturated set satisfies the Hintikka equation for all nominals and formulas.

LEMMA 8.34 (MCSS ARE HINTIKKA) *Assume H is a maximal consistent saturated set. Then the Hintikka equation holds for H , for any i and p .*

assumes *consistent H and maximal H and saturated H*
shows $\text{semics } (\text{canonical } H \ i) \ (\text{rel } H) \ p \longleftrightarrow \text{rel } H \ (\text{canonical } H \ i) \ p$

PROOF. By cases on the formula p .

The nominal case holds since the relation is an equivalence relation.

For the diamond case, we prove both directions. I omit the formal marker \cdot in front of nominals. In the first direction, assume that there is a nominal k reachable from $[i]$ such that $(k, p) \in H$. By reachability, we must then have some k such that $([i], \Diamond k) \in H$ and $([k], p) \in H$. By the nature of the equivalence classes, we must directly have $(k, p) \in H$ for that k . By the derivability theorem 8.16, we must have some list of elements A from H , from which we can derive both $([i], \Diamond k)$ and (k, p) . By the *DiaI* rule, we can then derive $([i], \Diamond p)$ from A . By theorem 8.16 again, this proves that it is in H as desired. In the second direction, assume that $([i], \Diamond p) \in H$. By saturation we must then have some k such that $([i], \Diamond k) \in H$ and $(k, p) \in H$. This time, we use the nature of equivalence classes to switch from $(k, p) \in H$ to $([k], p) \in H$. This now matches the definition of reachability, so we have some k reachable from $[i]$ with $(k, p) \in H$.

For the satisfaction operator, we can directly notice that $([k], p) \in H$ if and only if $([i], @k p) \in H$ from theorem 8.16, the *SatI* and *SatE* rules and the nature of equivalence classes. \square

Like for the first-order logic syntax, we notice that the cardinality of our universe of formulas is bounded by the cardinalities of propositional and nominal symbols.

THEOREM 8.35 (THE HL CALCULUS IS STRONGLY COMPLETE)

Assume p is valid under assumptions X , that the universe of nominals is infinite and the cardinality of unused nominals in X is at least the sum of the cardinalities of nominals and propositional symbols. Then we can derive (i, p) (for any i) from a list of labelled formulas from X .

assumes $\forall M :: ('i, 'p) \text{ model. } \forall g w.$
 $(\forall (k, q) \in X. (M, g, g k) \models q) \longrightarrow (M, g, w) \models p$
and *infinite* ($UNIV :: 'i \text{ set}$)
and $|UNIV :: 'i \text{ set}| + c |UNIV :: 'p \text{ set}| \leq o |UNIV - \text{nominals } X|$
shows $\exists A. \text{set } A \subseteq X \wedge A \vdash_{@} (i, p)$

PROOF. Analogous to the proof of theorem 8.20. \square

Again, weak completeness allows us to state the cardinality assumption in a slightly neater fashion. The predicate *valid* $p \equiv \forall (M :: ('i, 'p) \text{ model}) g w. (M, g, w) \models p$ simply abbreviates validity in the correct domain.

COROLLARY 8.36 (THE HL CALCULUS IS COMPLETE)

If p is a valid formula, the universe of nominals is infinite and there are at least as many nominals as propositional symbols, then we can derive (i, p) for any i .

assumes *valid* p
and *infinite* ($UNIV :: 'i \text{ set}$)
and $|UNIV :: 'p \text{ set}| \leq o |UNIV :: 'i \text{ set}|$
shows $\square \vdash_{@} (i, p)$

We can do away with the cardinality assumptions completely by using the same type for nominals and propositional symbols. We just need it to be infinite.

COROLLARY 8.37 *If p is valid and the universe of nominals / propositional symbols is infinite, then we can derive (i, p) for any i .*

fixes $p :: ('i, 'i) \text{ fm}$
assumes *valid* p **and** *infinite* ($UNIV :: 'i \text{ set}$)
shows $\square \vdash_{@} (i, p)$

8.6 Conclusion

We have seen how to build maximal consistent saturated sets for any logic that satisfies a reasonable set of assumptions. Moreover, we have seen how to connect MCSs with refutational and derivational proof systems, respectively, to obtain useful lemmas about when formulas are absent from and present in MCSs. I have instantiated the abstract framework with five different logics, demonstrating its usability in practice. The Hintikka equation that I have provided guides the completeness proof: if we know the canonical model, we automatically find out what we need to prove about the MCSs to prove consistency.

Some future work remains. I would like to provide a similar solution to deriving Hintikka sets for tableaux, for which they were first invented. Due to its refutational nature, the manual Hintikka sets rely on downwards saturation only and heavy use of object logic negation. Maybe there is a way to *punch a hole* in the calculus, rather than the semantics, to obtain a useful Hintikka equation. In a similar vein, I rely heavily on the structural rules, but Berghofer's formalization [6] of Fitting's book [4] does not include any such rules for its natural deduction system. It would be interesting to see if the examples above can be similarly simplified, perhaps by refining the theorems about proof systems and maximal consistent sets. Other than this, it seems obvious to prove the completeness of more logics using the framework. I am not aware that anyone has formalized a transfinite version of Lindenbaum's lemma in Isabelle before. My contribution enables us to formalize the completeness of logics with inherently uncountable languages.

References

- [1] Chen C. Chang and H. Jerome Keisler. *Model theory, Third Edition*. Vol. 73. Studies in logic and the foundations of mathematics. North-Holland, 1992. ISBN: 978-0-444-88054-3.
- [2] Ronald Fagin et al. *Reasoning About Knowledge*. MIT Press, 1995. ISBN: 9780262562003. DOI: [10.7551/mitpress/5803.001.0001](https://doi.org/10.7551/mitpress/5803.001.0001).
- [3] Raymond M. Smullyan. *First-order logic*. Dover Publications, 1995.
- [4] Melvin Fitting. *First-Order Logic and Automated Theorem Proving, Second Edition*. Graduate Texts in Computer Science. Springer, 1996. ISBN: 978-1-4612-7515-2. DOI: [10.1007/978-1-4612-2360-3](https://doi.org/10.1007/978-1-4612-2360-3).
- [5] Leon Henkin. "The discovery of my completeness proofs". In: *Bulletin of Symbolic Logic* 2.2 (1996), pp. 127–158. DOI: [10.2307/421107](https://doi.org/10.2307/421107).

- [6] Stefan Berghofer. “First-Order Logic According to Fitting”. In: *Archive of Formal Proofs* (Aug. 2007). <https://isa-afp.org/entries/FOL-Fitting.html>, Formal proof development. ISSN: 2150-914x.
- [7] Torben Braüner. *Hybrid Logic and its Proof-Theory*. 1st ed. Springer, 2011. ISBN: 978-94-007-0001-7. DOI: [10.1007/978-94-007-0002-4](https://doi.org/10.1007/978-94-007-0002-4).
- [8] Julian J. Schlöder and Peter Koepke. “The Gödel Completeness Theorem for Uncountable Languages”. In: *Formalized Mathematics* 20.3 (2012), pp. 199–203. DOI: [10.2478/v10037-012-0023-z](https://doi.org/10.2478/v10037-012-0023-z).
- [9] Jasmin Christian Blanchette, Andrei Popescu, and Dmitriy Traytel. “Cardinals in Isabelle/HOL”. In: *Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*. Ed. by Gerwin Klein and Ruben Gamboa. Vol. 8558. Lecture Notes in Computer Science. Springer, 2014, pp. 111–127. DOI: [10.1007/978-3-319-08970-6_8](https://doi.org/10.1007/978-3-319-08970-6_8).
- [10] Jasmin Christian Blanchette, Andrei Popescu, and Dmitriy Traytel. “Soundness and Completeness Proofs by Coinductive Methods”. In: *Journal of Automated Reasoning* 58.1 (2017), pp. 149–179. DOI: [11.1007/s10817-016-9391-3](https://doi.org/10.1007/s10817-016-9391-3).

Included Publications

Asta Halkjær From. “Formalizing Henkin-Style Completeness of an Axiomatic System for Propositional Logic”. In: *European Summer School in Logic, Language, and Information - ESSLLI 2019 & 2020 Student Sessions, Selected Papers*. Ed. by Alexandra Pavlova and Mina Young Pedersen. Lecture Notes in Computer Science. Springer, 2023. Forthcoming.

Asta Halkjær From. “A Succinct Formalization of the Completeness of First-Order Logic”. In: *27th International Conference on Types for Proofs and Programs, TYPES 2021, June 14-18, 2021, Leiden, The Netherlands (Virtual Conference)*. Ed. by Henning Basold, Jesper Cockx, and Silvia Ghilezan. Vol. 239. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 8:1–8:24. DOI: [10.4230/LIPIcs.TYPES.2021.8.4230](https://doi.org/10.4230/LIPIcs.TYPES.2021.8.4230).

Asta Halkjær From and Frederik Krogsdal Jacobsen. “Verifying a Sequent Calculus Prover for First-Order Logic with Functions in Isabelle/HOL”. In: *13th International Conference on Interactive Theorem Proving, ITP 2022, August 7-10, 2022, Haifa, Israel*. Ed. by June Andronick and Leonardo de Moura. Vol. 237. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 13:1–13:22. DOI: [10.4230/LIPIcs.ITP.2022.13](https://doi.org/10.4230/LIPIcs.ITP.2022.13).

Asta Halkjær From. “Synthetic Completeness for a Terminating Seligman-Style Tableau System”. In: *26th International Conference on Types for Proofs and Programs, TYPES 2020, March 2-5, 2020, University of Turin, Italy*. Ed. by Ugo de'Liguoro, Stefano Berardi, and Thorsten Altenkirch. Vol. 188. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 5:1–5:17. DOI: [10.4230/LIPIcs.TYPES.2020.5.4230](https://doi.org/10.4230/LIPIcs.TYPES.2020.5.4230).

Asta Halkjær From. “Formalized Soundness and Completeness of Epistemic and Public Announcement Logic”. In: *Special issue: Selected papers from WoLLIC 2021, the 27th Workshop on Logic, Language, Information, and Computation*. Ed. by Alexandra Silva, Renata Wassermann, and Ruy J. G. B. de Queiroz. Journal of Logic and Computation. Forthcoming.

Jannis Limperg and Asta Halkjær From. “Aesop: White-Box Best-First Proof Search for Lean”. In: *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2023, Boston, MA, USA, January 16-17, 2023*. Ed. by Robbert Krebbers et al. ACM, 2023, pp. 253–266. DOI: [10.1145/3573105.3575671](https://doi.org/10.1145/3573105.3575671).

Other Publications

- [1] Asta Halkjær From. “Hybrid logic in the Isabelle Proof Assistant: Benefits, Challenges and the Road Ahead”. In: *Short Papers: Advances in Modal Logic (AiML)*. Ed. by Nicola Olivetti and Rinke Verbrugge. 2020, pp. 23–27.
- [2] Asta Halkjær From, Patrick Blackburn, and Jørgen Villadsen. “Formalizing a Seligman-Style Tableau System for Hybrid Logic (Short Paper)”. In: *Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part I*. Ed. by Nicolas Peltier and Viorica Sofronie-Stokkermans. Vol. 12166. Lecture Notes in Computer Science. Springer, 2020, pp. 474–481. DOI: [10.1007/978-3-030-51074-9_27](https://doi.org/10.1007/978-3-030-51074-9_27).
- [3] Asta Halkjær From and Jørgen Villadsen. “A Concise Sequent Calculus for Teaching First-Order Logic”. Isabelle Workshop 2020. <https://sketis.net/isabelle/isabelle-workshop-2020>. 2020.
- [4] Asta Halkjær From, Jørgen Villadsen, and Patrick Blackburn. “Isabelle/HOL as a Meta-Language for Teaching Logic”. In: *Proceedings 9th International Workshop on Theorem Proving Components for Educational Software, ThEdu@IJCAR 2020, Paris, France, 29th June 2020*. Ed. by Pedro Quaresma, Walther Neuper, and João Marcos. Vol. 328. EPTCS. 2020, pp. 18–34. DOI: [10.4204/EPTCS.328.2](https://doi.org/10.4204/EPTCS.328.2).
- [5] Asta Halkjær From. “Formalized Soundness and Completeness of Epistemic Logic”. In: *Logic, Language, Information, and Computation - 27th International Workshop, WoLLIC 2021, Virtual Event, October 5-8, 2021, Proceedings*. Ed. by Alexandra Silva, Renata Wassermann, and Ruy J. G. B. de Queiroz. Vol. 13038. Lecture Notes in Computer Science. Springer, 2021, pp. 1–15. DOI: [10.1007/978-3-030-88853-4_1](https://doi.org/10.1007/978-3-030-88853-4_1).
- [6] Asta Halkjær From, Agnes Moesgård Eschen, and Jørgen Villadsen. “Formalizing Axiomatic Systems for Propositional Logic in Isabelle/HOL”. In: *Intelligent Computer Mathematics - 14th International Conference, CICM 2021, Timisoara, Romania, July 26-31, 2021, Proceedings*. Ed. by Fairouz Kamareddine and Claudio Sacerdoti Coen. Vol. 12833. Lecture Notes in Computer Science. Springer, 2021, pp. 32–46. DOI: [10.1007/978-3-030-81097-9_3](https://doi.org/10.1007/978-3-030-81097-9_3).
- [7] Asta Halkjær From, Frederik Krogsdal Jacobsen, and Jørgen Villadsen. “SeCaV: A Sequent Calculus Verifier in Isabelle/HOL”. In: *Proceedings 16th Logical and Semantic Frameworks with Applications, LSFA 2021*,

- Buenos Aires, Argentina (Online), 23rd - 24th July, 2021. Ed. by Mauricio Ayala-Rincón and Eduardo Bonelli. Vol. 357. EPTCS. 2021, pp. 38–55. DOI: [10.4204/EPTCS.357.4](https://doi.org/10.4204/EPTCS.357.4).
- [8] Asta Halkjær From, Anders Schlichtkrull, and Jørgen Villadsen. “A Sequent Calculus for First-Order Logic Formalized in Isabelle/HOL”. In: *Proceedings of the 36th Italian Conference on Computational Logic, Parma, Italy, September 7-9, 2021*. Ed. by Stefania Monica and Federico Bergenti. Vol. 3002. CEUR Workshop Proceedings. CEUR-WS.org, 2021, pp. 107–121. URL: <http://ceur-ws.org/Vol-3002/paper7.pdf>.
- [9] Asta Halkjær From and Jørgen Villadsen. “Teaching Automated Reasoning and Formally Verified Functional Programming in Agda and Isabelle/HOL”. In: *10th International Workshop on Trends in Functional Programming in Education (TFPIE 2021) — Presentation Only/Online Papers*. 2021, pp. 1–20. URL: https://wiki.tfpie.science.ru.nl/images/a/a6/TFPIE_AHF_JV.pdf.
- [10] Jørgen Villadsen, Asta Halkjær From, and Patrick Blackburn. “Teaching Intuitionistic and Classical Propositional Logic Using Isabelle”. In: *Proceedings 10th International Workshop on Theorem Proving Components for Educational Software, ThEdu@CADE 2021, (Remote) Carnegie Mellon University, Pittsburgh, PA, United States, 11 July 2021*. Ed. by João Marcos, Walther Neuper, and Pedro Quaresma. Vol. 354. EPTCS. 2021, pp. 71–85. DOI: [10.4204/EPTCS.354.6](https://doi.org/10.4204/EPTCS.354.6).
- [11] Agnes Moesgård Eschen, Asta Halkjær From, and Jørgen Villadsen. “More Formalized Axiomatic Systems for Propositional Logic in Isabelle/HOL”. In: *Logic and Artificial Intelligence* (2022). Forthcoming.
- [12] Asta Halkjær From. “On Termination for Hybrid Tableaux”. Isabelle Workshop 2022. https://files.sketis.net/Isabelle_Workshop_2022/Isabelle_2022_paper_13.pdf. 2022.
- [13] Asta Halkjær From, Simon Tobias Lund, and Jørgen Villadsen. “A Case Study in Computer-Assisted Meta-reasoning”. In: *Distributed Computing and Artificial Intelligence, Volume 2: Special Sessions 18th International Conference*. Ed. by Sara Rodríguez González et al. Vol. 332. Lecture Notes in Networks and Systems. Cham: Springer International Publishing, 2022, pp. 53–63. ISBN: 978-3-030-86887-1. DOI: [10.1007/978-3-030-86887-1_5](https://doi.org/10.1007/978-3-030-86887-1_5).
- [14] Jørgen Villadsen et al. “Interactive Theorem Proving for Logic and Information”. In: *Natural Language Processing in Artificial Intelligence — NLPinAI 2021*. Ed. by Roussanka Loukanova. Vol. 999. Studies in Computational Intelligence. Cham: Springer International Publishing, 2022, pp. 25–48. ISBN: 978-3-030-90138-7. DOI: [10.1007/978-3-030-90138-7_2](https://doi.org/10.1007/978-3-030-90138-7_2).

Archive of Formal Proofs

Asta Halkjær From. “Synthetic Completeness”. In: *Archive of Formal Proofs* (Jan. 2023). https://isa-afp.org/entries/Synthetic_Completeness.html, Formal proof development. ISSN: 2150-914x.

Asta Halkjær From and Jørgen Villadsen. “Soundness and Completeness of Implicational Logic”. In: *Archive of Formal Proofs* (Sept. 2022). https://isa-afp.org/entries/Implicational_Logic.html, Formal proof development. ISSN: 2150-914x.

Asta Halkjær From. “A Naive Prover for First-Order Logic”. In: *Archive of Formal Proofs* (Mar. 2022). https://isa-afp.org/entries/FOL_Seq_Calc3.html, Formal proof development. ISSN: 2150-914x.

Asta Halkjær From and Frederik Krogsdal Jacobsen. “A Sequent Calculus Prover for First-Order Logic with Functions”. In: *Archive of Formal Proofs* (Jan. 2022). https://isa-afp.org/entries/FOL_Seq_Calc2.html, Formal proof development. ISSN: 2150-914x.

Asta Halkjær From. “Soundness and Completeness of an Axiomatic System for First-Order Logic”. In: *Archive of Formal Proofs* (Sept. 2021). https://isa-afp.org/entries/FOL_Axiomatic.html, Formal proof development. ISSN: 2150-914x.

Asta Halkjær From. “Public Announcement Logic”. In: *Archive of Formal Proofs* (June 2021). https://isa-afp.org/entries/Public_Announcement_Logic.html, Formal proof development. ISSN: 2150-914x.

Asta Halkjær From. “Formalizing a Seligman-Style Tableau System for Hybrid Logic”. In: *Archive of Formal Proofs* (Dec. 2019). https://isa-afp.org/entries/Hybrid_Logic.html, Formal proof development. Updated Jun. 2020. ISSN: 2150-914x.

Asta Halkjær From. “Epistemic Logic: Completeness of Modal Logics”. In: *Archive of Formal Proofs* (Oct. 2018). https://isa-afp.org/entries/Epistemic_Logic.html, Formal proof development. Updated Apr. 2021. ISSN: 2150-914x.