

Middlesex University Research Repository

An open access repository of

Middlesex University research

<http://eprints.mdx.ac.uk>

Blanchette, Jasmin Christian, Popescu, Andrei and Traytel, Dmitriy (2017) Soundness and completeness proofs by coinductive methods. *Journal of Automated Reasoning*, 58 (1). pp. 149-179. ISSN 0168-7433

Final accepted version (with author's formatting)

This version is available at: <http://eprints.mdx.ac.uk/21796/>

Copyright:

Middlesex University Research Repository makes the University's research available electronically.

Copyright and moral rights to this work are retained by the author and/or other copyright owners unless otherwise stated. The work is supplied on the understanding that any use for commercial gain is strictly forbidden. A copy may be downloaded for personal, non-commercial, research or study without prior permission and without charge.

Works, including theses and research projects, may not be reproduced in any format or medium, or extensive quotations taken from them, or their content changed in any way, without first obtaining permission in writing from the copyright holder(s). They may not be sold or exploited commercially in any format or medium without the prior written permission of the copyright holder(s).

Full bibliographic details must be given when referring to, or quoting from full items including the author's name, the title of the work, publication details where relevant (place, publisher, date), pagination, and for theses or dissertations the awarding institution, the degree type awarded, and the date of the award.

If you believe that any material held in the repository infringes copyright law, please contact the Repository Team at Middlesex University via the following email address:

eprints@mdx.ac.uk

The item will be removed from the repository while any claim is being investigated.

See also repository copyright: re-use policy: <http://eprints.mdx.ac.uk/policies.html#copy>

Soundness and Completeness Proofs by Coinductive Methods

Jasmin Christian Blanchette · Andrei Popescu ·
Dmitriy Traytel

the date of receipt and acceptance should be inserted later

Abstract Codatatypes are absent from many programming languages and theorem provers, reflecting their absence from the standard repertoire of most computer scientists and logicians. We make a case for their usefulness by showing how they can be employed to produce compact, high-level proofs of key results in logic: the soundness and completeness of proof systems for variations of first-order logic. For the classical completeness result, we first establish an abstract property of possibly infinite derivation trees. The abstract proof can be instantiated for a wide range of Gentzen and tableau systems as well as various flavors of first-order logic. Soundness is often easy to prove, but it becomes interesting as soon as one allows infinite proofs of first-order formulas. This forms the subject of several cyclic proof systems for first-order logic augmented with inductive predicate definitions studied in the literature. All the discussed results are formalized using Isabelle/HOL's recently introduced support for codatatypes and corecursion. The development illustrates some unique features of Isabelle/HOL's new coinductive specification language such as nesting through non-free types and mixed recursion–corecursion.

1 Introduction

Gödel's completeness theorem [20] is a major result about first-order logic (FOL). It forms the basis of results and techniques in various areas, including mathematical logic, automated deduction, and program verification. It can be stated as follows: If a set of formulas is satisfied by all structures, then it has a proof. The theorem enjoys many accounts in the literature

Jasmin Christian Blanchette
Inria Nancy & LORIA, Villers-lès-Nancy, France
Max-Planck-Institut für Informatik, Saarbrücken, Germany
E-mail: jasmin.blanchette@{inria.fr,mpi-inf.mpg.de}

Andrei Popescu
Department of Computer Science, School of Science and Technology, Middlesex University, UK
E-mail: a.popescu@mdx.ac.uk

Dmitriy Traytel
Institute of Information Security, Department of Computer Science, ETH Zurich, Switzerland
E-mail: traytel@inf.ethz.ch

that generalize and simplify the original proof; indeed, a textbook on mathematical logic would be incomplete without a proof of this fundamental theorem.

Formal logic has always been a battleground between semantic and syntactic methods. Generally, mathematicians belong to the semantic school, whereas computer scientists tend to take the other side of the argument. The completeness theorem, which combines syntax and semantics, is also disputed, with the result that each school has its own proof. In his review of Gallier’s *Logic for Computer Science* [19], Pfenning, a fellow “syntactician,” notes the following [38]:

All too often, proof-theoretic methods are neglected in favor of shorter, and superficially more elegant semantic arguments. [In contrast, in Gallier’s book] the treatment of the proof theory of the Gentzen system is oriented towards computation with proofs. For example, a pseudo-Pascal version of a complete search procedure for first-order cut-free Gentzen proofs is presented.

In the context of completeness, the “superficially more elegant semantic arguments” are proofs that rely on Hilbert systems. These systems have several axioms but only one or two deduction rules, providing minimal support for presenting the structure of proofs or for modeling proof search. A proof of completeness based on Hilbert systems follows the *Henkin style*: It employs a heavy bureaucratic apparatus to establish facts about deduction and conservative language extensions, culminating in a highly nonconstructive step: an application of Zorn’s lemma to extend any syntactically consistent set of formulas to a maximally consistent one, from which a model is produced.

In contrast, a proof of completeness based on more elaborate Gentzen or tableau systems follows the *Beth–Hintikka style* [28]. It performs a search that builds either a finite deduction tree yielding a proof (or refutation, depending on the system) or an infinite tree from which a countermodel (or model) can be extracted. Such completeness proofs have an intuitive content that stresses the tension of the argument: The deduction system systematically tries to prove the goal; a failure yields, at the limit, a countermodel.

The intuitive appeal of the Beth–Hintikka approach comes at a price: It requires reasoning about infinite derivation trees and infinite paths. Unfortunately, convenient means to reason about infinite (or lazy) data structures are lacking in mainstream mathematics. For example, an otherwise extremely rigorous textbook such as Bell and Machover’s [1] becomes surprisingly informal when defining and using possibly infinite refutation tableau trees:

A tableau is a set of elements, called nodes, partially ordered and classified into levels as explained below. With each node is associated a finite set of formulas. We shall usually identify a given node with its associated set of formulas; this is somewhat imprecise (since in fact the same set of formulas can be associated with different nodes) but will not cause confusion.

Each node belongs to a unique level, which is labeled by some natural number. There is just one node of level 0, called the initial node of the tableau. Each node at level $n + 1$ is a successor of a unique node, which must be of level n .

In textbooks, at best the trees are defined rigorously (e.g., as prefix-closed sets), but the reasoning is performed informally, disregarding the original definition and relying on the intuitive notion of trees, as Gallier does. One could argue that trees are intuitive and do not need a formal treatment, but the same holds for the syntax of formulas, which is treated very rigorously in most of the textbooks.

The main contribution of this article is a rigorous Beth–Hintikka-style proof of the completeness theorem, based on a Gentzen system. The potentially infinite trees are captured

by codatatypes (also called coinductive datatypes or final coalgebras) [26]. Another novel aspect of the proof is its modularity: The core tree construction argument is isolated from the proof system and concrete formula syntax, with the concrete syntactic details concealed behind an abstract Herbrandness assumption (Section 3). The Herbrandness assumption can be verified in concrete cases (by performing the standard Herbrand model construction) for a wide range of Gentzen and tableau systems for FOL, various flavors of FOL (e.g., with or without predicates, equality, or sorts), and even modal logics with explicit-world Gentzen systems (Section 4). This modularization replaces the textbook proofs by analogy. The core of the argument amounts to reasoning about a functional program over lazy data structures.

A second contribution of this article is an application of the same coinductive machinery (infinite trees and streams and corecursive functions between them) to some interesting recent results from the automated deduction literature: the soundness of infinite (cyclic) proofs for FOL with inductive definitions and related logics, studied by Brotherston et al. [11–14]. For formalizing these results, we follow rather closely the abstract constructions of Brotherston et al. [14], except that we use coinduction and corecursion to streamline the development. The presentation follows the same path as for completeness: The main result is stated abstractly (Section 5) and instantiated by concrete examples (Section 6).

The proofs of the abstract results are formalized in Isabelle/HOL (Section 7). The infinite trees and paths constructions rely on a new definitional package for codatatypes [5, 10], which automates the derivation of characteristic theorems from high-level specifications of types and functions. Through Isabelle’s code generator [22], the corecursive construction gives rise to a Haskell program that implements a semidecision procedure for validity instantiable with various proof systems, yielding verified sound and complete provers.

The completeness proof has been applied to the formalization of optimized translations between sorted and unsorted FOL [4, 6]. The soundness proofs of these translations rest on the Löwenheim–Skolem theorem, a corollary of a slightly generalized version of the completeness theorem. The previous formal proofs of the completeness theorem, including two in Isabelle, support a more restrictive logic than many-sorted FOL (Section 8).

An earlier version of this article was presented at the IJCAR 2014 conference in Vienna, Austria, under a different title [8]. The article considerably extends the conference paper with infinite-proof soundness (Sections 5 and 6) as a second application of coinductive methods. It also provides more details about the Beth–Hintikka-style proof of the completeness theorem (Sections 3 and 4).

Conventions. Isabelle/HOL [36] is a proof assistant based on classical higher-order logic (HOL) with Hilbert choice, the axiom of infinity, and rank-1 polymorphism. It is the logic of Gordon’s original HOL system [21] and of its many successors. HOL notations are similar to those of functional programming languages, but they also include many traditional symbols and syntaxes from mathematics, notably to denote simply typed sets. We refer to Nipkow and Klein [35, Part 1] for a modern introduction. In this article, the logic is viewed not as a formal system but rather as a framework for expressing mathematics, much like set theory is employed by working mathematicians. In keeping with the standard semantics of HOL, types α are identified with sets.

2 Preliminaries on First-Order Logic

The soundness and completeness results we formalize in this article apply to an array of variations of first-order logic and beyond (modal logics, separation logic, etc.). To give some

concrete grounding for the forthcoming abstract development, we recall the basic (unsorted) first-order logic and its extension with inductive predicates.

2.1 Classical First-Order Logic

We fix a first-order language: a countably infinite set var of variables x, y, z and countable sets fsym and psym of function symbols f and predicate symbols p together with an assignment $\text{ar} : \text{fsym} \uplus \text{psym} \rightarrow \text{nat}$ of numeric arities. Terms $t \in \text{term}$ are symbolic expressions built inductively from variables by application of function symbols $f \in \text{fsym}$ to tuples of arguments whose lengths respect the arities: $f(t_1, \dots, t_{\text{ar} f})$. Atoms $a \in \text{atom}$ are symbolic expressions of the form $p(t_1, \dots, t_{\text{ar} p})$, where $p \in \text{psym}$ and $t_1, \dots, t_{\text{ar} p} \in \text{term}$.

Formulas φ, ψ may be atoms, negations, conjunctions, or universal quantifications. They are defined as follows:

```

datatype fmla = Atm atom
                | Neg fmla
                | Conj fmla fmla
                | All var fmla

```

As usual, we define the (syntactic) implication of two formulas φ_1, φ_2 by $\text{Imp } \varphi_1 \varphi_2 = \text{Neg } (\text{Conj } \varphi_1 (\text{Neg } \varphi_2))$.

To distinguish between the first-order language of study and the HOL metalanguage, we use the constructor names `Neg`, `Conj`, and `All` for the former, keeping the traditional symbols \neg , \wedge , and \forall for the latter. We often write a instead of `Atm` a , thus pretending that atoms are included in formulas.

A structure $\mathcal{S} = (S, (F_f)_{f \in \text{fsym}}, (P_p)_{p \in \text{psym}})$ for the given language consists of a carrier set S , together with a function $F_f : S^n \rightarrow S$ for each n -ary $f \in \text{fsym}$ and a predicate $P_p : S^n \rightarrow \text{bool}$ for each n -ary $p \in \text{psym}$. The notions of interpretation of a term t and satisfaction of a formula φ by a structure \mathcal{S} with respect to a variable valuation $\xi : \text{var} \rightarrow S$ are defined in the standard way. For terms (by structural recursion):

$$\llbracket x \rrbracket_{\xi}^{\mathcal{S}} = \xi x \qquad \llbracket f(t_1, \dots, t_n) \rrbracket_{\xi}^{\mathcal{S}} = F_f (\llbracket t_1 \rrbracket_{\xi}^{\mathcal{S}}, \dots, \llbracket t_n \rrbracket_{\xi}^{\mathcal{S}})$$

For atoms:

$$\mathcal{S} \models_{\xi} p(t_1, \dots, t_n) \quad \text{iff} \quad P_p (\llbracket t_1 \rrbracket_{\xi}^{\mathcal{S}}, \dots, \llbracket t_n \rrbracket_{\xi}^{\mathcal{S}})$$

For formulas (by structural recursion):

$$\begin{aligned} \mathcal{S} \models_{\xi} \text{Atm } a & \quad \text{iff} \quad \mathcal{S} \models_{\xi} a & \quad \mathcal{S} \models_{\xi} \text{Conj } \varphi \psi & \quad \text{iff} \quad \mathcal{S} \models_{\xi} \varphi \wedge \mathcal{S} \models_{\xi} \psi \\ \mathcal{S} \models_{\xi} \text{Neg } \varphi & \quad \text{iff} \quad \mathcal{S} \not\models_{\xi} \varphi & \quad \mathcal{S} \models_{\xi} \text{All } x \varphi & \quad \text{iff} \quad \forall a \in S. \mathcal{S} \models_{\xi[x \leftarrow a]} \varphi \end{aligned}$$

Above, $\xi[x \leftarrow a]$ denotes the valuation that sends x to a and all other variables y to ξy .

We define the notion of a structure \mathcal{S} satisfying a formula φ , written $\mathcal{S} \models \varphi$, to mean satisfaction with respect to all valuations: $\forall \xi. \mathcal{S} \models_{\xi} \varphi$.

A sequent is a pair $\Gamma \triangleright \Delta$ of finite formula sets. Satisfaction is extended to sequents: $\mathcal{S} \models \Gamma \triangleright \Delta$ iff $(\forall \varphi \in \Gamma. \mathcal{S} \models \varphi) \Rightarrow (\exists \psi \in \Delta. \mathcal{S} \models \psi)$. We can think of $\Gamma \triangleright \Delta$ as an implication between the conjunction of the formulas of Γ and the disjunction of the formulas of Δ . If $\mathcal{S} \models \Gamma \triangleright \Delta$, we also call \mathcal{S} a *model* of $\Gamma \triangleright \Delta$. By contrast, if $\mathcal{S} \not\models \Gamma \triangleright \Delta$, we call \mathcal{S} a *countermodel* of $\Gamma \triangleright \Delta$.

A standard proof system on sequents is defined inductively as follows, where the notation Γ, φ abbreviates the set $\Gamma \cup \{\varphi\}$:

$$\begin{array}{c}
\frac{}{\Gamma, \text{Atm } a \triangleright \Delta, \text{Atm } a} \text{AX} \quad \frac{\Gamma \triangleright \Delta, \varphi}{\Gamma, \text{Neg } \varphi \triangleright \Delta} \text{NEGL} \quad \frac{\Gamma, \varphi \triangleright \Delta}{\Gamma \triangleright \Delta, \text{Neg } \varphi} \text{NEGR} \\
\frac{\Gamma, \varphi, \psi \triangleright \Delta}{\Gamma, \text{Conj } \varphi \psi \triangleright \Delta} \text{CONJL} \quad \frac{\Gamma \triangleright \Delta, \varphi \quad \Gamma \triangleright \Delta, \psi}{\Gamma \triangleright \Delta, \text{Conj } \varphi \psi} \text{CONJR} \\
\frac{\Gamma, \text{All } x \varphi, \varphi[t/x] \triangleright \Delta}{\Gamma, \text{All } x \varphi \triangleright \Delta} \text{ALLL} \quad \frac{\Gamma \triangleright \Delta, \varphi[y/x]}{\Gamma \triangleright \Delta, \text{All } x \varphi} \text{ALLR} \quad (y \text{ fresh})
\end{array}$$

In a proof, the rules are applied from bottom to top. One chooses a formula from either side of the sequent, the eigenformula, and applies a rule according to the topmost connective or quantifier. For a given choice of eigenformula, at most one rule is applicable. The aim of applying the rules is to prove the sequent by building a finite derivation tree whose branches are closed by an axiom (AX). Provability of a sequent in this system is denoted by prefixing \vdash to the sequent (e.g., $\vdash \Gamma \triangleright \Delta$).

The soundness theorem states that any provable sequent is satisfied by all structures:

$$\vdash \Gamma \triangleright \Delta \Rightarrow (\forall \mathcal{S}. \mathcal{S} \models \Gamma \triangleright \Delta)$$

The completeness theorem states the converse—namely, any sequent that is satisfied by all structures is provable:

$$(\forall \mathcal{S}. \mathcal{S} \models \Gamma \triangleright \Delta) \Rightarrow \vdash \Gamma \triangleright \Delta$$

Section 3 presents abstract versions of these classic soundness and completeness theorems.

2.2 First-Order Logic with Inductive Predicates

Another logic we consider is first-order logic with inductive predicates (IFOL). In addition to the first-order language given by var , fsym , psym and $\text{ar} : \text{fsym} \uplus \text{psym} \rightarrow \text{nat}$, we fix a subset of predicate symbols $\text{ipsym} \subseteq \text{psym}$ which are called *inductive*. Moreover, we fix, for each $p \in \text{ipsym}$, a set ind_p of Horn clauses specifying p , so that each element of ind_p has the form $\text{Imp}(\text{Conj } \psi_1 \dots \psi_m \varphi_1 \dots \varphi_n) \varphi$, where

- Conj denotes the conjunction of multiple formulas;¹
- φ is an atom of the form $p(t_1, \dots, t_{\text{ar } p})$;
- each φ_i is an atom of the form $p'(t_1, \dots, t_{\text{ar } p'})$ for some $p' \in \text{ipsym}$;
- each ψ_j is a formula not containing any $p' \in \text{ipsym}$.

The above set of restrictions is aimed to ensure monotonicity of the Horn clauses interpreted as inductive definitions. To simplify the exposition, we choose a rather strong restriction, but more flexible ones are possible [11].

A first-order structure $\mathcal{S} = (S, (F_f)_{f \in \text{fsym}}, (P_p)_{p \in \text{psym}})$ is said to be *inductive* (with respect to ipsym and $(\text{ind}_p)_{p \in \text{ipsym}}$) if the interpretation of each $p \in \text{ipsym}$ is indeed inductive, i.e., the family $(P_p)_{p \in \text{ipsym}}$ constitutes the least fixpoint of all the inductive Horn clauses interpreted in \mathcal{S} ; in other words, if $(P_p)_{p \in \text{ipsym}}$ is the least family of predicates (with respect to component-wise implication) such that $\mathcal{S} \models \chi$ for all $\chi \in \bigcup_{p \in \text{psym}} \text{ind}_p$.

Example 1 Assume $\text{fsym} = \{0, \text{Suc}\}$, $\text{ipsym} = \text{psym} = \{\text{even}, \text{odd}\}$, and

¹ Given formulas ψ_1, \dots, ψ_k , we let $\text{Conj } \psi_1 \dots \psi_k$ denote $\text{Conj } \psi_1 (\text{Conj } \psi_2 (\dots \psi_n) \dots)$. In particular, when $k = 0$ it denotes the “true” formula \top , defined in a standard way, e.g., as $\text{Imp } a \ a$ for some atom a .

- ind_{even} consists of the following clauses:
 - $\text{even}(0)$;
 - $\text{Imp}(\text{even}(x))(\text{even}(\text{Suc}(\text{Suc}(x))))$;
- ind_{odd} consists of the following clauses:
 - $\text{odd}(\text{Suc}(0))$;
 - $\text{Imp}(\text{odd}(x))(\text{odd}(\text{Suc}(\text{Suc}(x))))$.

A structure $\mathcal{S} = (S, F_0, F_{\text{Suc}}, P_{\text{even}})$ for this language is inductive iff, for fixed $(S, F_0, F_{\text{Suc}}, P_{\text{even}}$ and P_{odd} is the least (i.e., strongest) pair of unary predicates on S such that

- $P_{\text{even}}(F_0)$;
- $\forall n \in S. P_{\text{even}}(n) \Rightarrow P_{\text{even}}(F_{\text{Suc}}(F_{\text{Suc}}(n)))$;
- $P_{\text{odd}}(F_{\text{Suc}}(F_0))$;
- $\forall n \in S. P_{\text{odd}}(n) \Rightarrow P_{\text{odd}}(F_{\text{Suc}}(F_{\text{Suc}}(n)))$.

In particular, if (S, F_0, F_{Suc}) is the standard set of natural numbers with zero and successor, then P_{even} and P_{odd} must be the standard evenness and oddness predicates.

Obviously, we have more admissible rules for inductive structures than for arbitrary structures. The following rules are admissible for even, where $\Gamma[t/x]$ denotes the (capture-avoiding) substitution of the term t for the variable x in all formulas of Γ :

$$\frac{}{\Gamma \triangleright \Delta, \text{even}(0)} \text{EVEN}_0 \quad \frac{\Gamma \triangleright \Delta, \text{even}(x)}{\Gamma \triangleright \Delta, \text{even}(\text{Suc}(\text{Suc}(x)))} \text{EVEN}_{\text{Suc}}$$

$$\frac{\Gamma[0/x] \triangleright \Delta[0/x] \quad \Gamma[\text{Suc}(\text{Suc}(y))/x], \text{even}(y) \triangleright \Delta[\text{Suc}(\text{Suc}(y))/x]}{\Gamma, \text{even}(x) \triangleright \Delta} \text{EVEN}_{\text{split}} \quad (\text{y fresh})$$

The “direct” rules EVEN_0 and EVEN_{Suc} (corresponding to the two cases in the inductive specification of even) are admissible since all inductive structures satisfy the Horn clauses for even. Moreover, the “inversion” rule $\text{EVEN}_{\text{split}}$ is admissible by the least fixpoint assumption—the rule expresses that, if $\text{even}(x)$ is known, then it must have been obtained by an application of one of the inductive clauses for even. Similar rules hold for odd:

$$\frac{}{\Gamma \triangleright \Delta, \text{odd}(\text{Suc}(0))} \text{ODD}_0 \quad \frac{\Gamma \triangleright \Delta, \text{odd}(x)}{\Gamma \triangleright \Delta, \text{odd}(\text{Suc}(\text{Suc}(x)))} \text{ODD}_{\text{Suc}}$$

$$\frac{\Gamma[\text{Suc}(0)/x] \triangleright \Delta[\text{Suc}(0)/x] \quad \Gamma[\text{Suc}(\text{Suc}(y))/x], \text{odd}(y) \triangleright \Delta[\text{Suc}(\text{Suc}(y))/x]}{\Gamma, \text{odd}(x) \triangleright \Delta} \text{ODD}_{\text{split}} \quad (\text{y fresh})$$

In general, for all inductive predicates $p \in \text{ipsym}$, the following rules are sound:

$$\frac{\Gamma \triangleright \Delta, \text{prems}(\chi)}{\Gamma \triangleright \Delta, \text{concl}(\chi)} p_\chi \quad \frac{(\Gamma[\bar{t}/\bar{x}], \text{prems}(\chi') \triangleright \Delta[\bar{t}/\bar{x}])_{\chi \in \text{ind}_p}}{\Gamma, p(\bar{x}) \triangleright \Delta} p_{\text{split}}$$

There is one p_χ rule for each $p \in \text{ipsym}$ and $\chi \in \text{ind}_p$, with $\text{prems}(\chi)$ denoting the premises of χ and $\text{concl}(\chi)$ its conclusion. In addition, there is one p_{split} rule for each $p \in \text{ipsym}$ such that

- χ' is a fresh-variable copy of χ , with all its free variables fresh for $\Gamma \cup \Delta$;
- $p(\bar{t})$ is $\text{concl}(\chi')$ —with \bar{t} thus being a tuple of terms $(t_1, \dots, t_{\text{ar } p})$;
- \bar{x} is a tuple of distinct variables $(x_1, \dots, x_{\text{ar } p})$.

The Gentzen system for IFOL consists of the FOL rules from Section 2.1 extended with the above rules p_χ , p_{split} and the following substitution rule:

$$\frac{\Gamma \triangleright \Delta}{\Gamma[t/x] \triangleright \Delta[t/x]} \text{SUBST}_{t,x}$$

Beyond the additional admissible rules, a crucial insight originating with Brotherston [11] is that, for inductive structures, a certain type of proof circularity can also be permitted. It allows the proof trees to have as leaves not only axioms, but also backward links to other sequents in the tree. This insight is useful for automating induction [11]. Brotherston et al. [14] give an abstract, logic-independent argument for why allowing such circularities is sound. In Section 5, we show that this argument can be naturally formalized using a coinductive machinery similar to the one we use for classic completeness.

3 Abstract Soundness and Completeness

In this section, we develop an abstract framework that allows us to state and prove soundness and completeness for an unspecified syntax and class of structures and satisfaction relation. The framework is obtained by distilling the corresponding concrete results for FOL.

For soundness, we simply assume the rules to be locally sound for the models, which immediately yields global soundness. Completeness is significantly more elaborate. The proof is divided in two parts. The first part, captured in this section, focuses on the core of the completeness argument in an abstract, syntax-free manner. This level captures the tension between the existence of a proof or of a countermodel-producing path, introduced through what we call an escape path—an infinite sequence of rule applications that “escapes” the proof attempt. The tension is distilled in the following result: Either there exists a finite derivation tree or there exists an infinite derivation tree containing a suitable escape path. The second part maps the escape path to a concrete, proof-system-specific countermodel employing a construction due to Herbrand. At the abstract level, we assume a “Herbrand function” that produces countermodels from escape paths. In Section 4, we instantiate this function for the Gentzen system presented in Section 2.1.

3.1 Sequents and Structures

We abstract away from the syntax of formulas and sequents and the specific rules of the proof system. We fix countable sets `sequent` and `rule` for sequents and rules. Our abstract sequents represent the formal statements in the logic, which can take either the form of concrete sequents or other forms.

Moreover, we abstract away from the concrete form of structures, assuming a fixed class `structure` and a satisfaction relation

$$|= : \text{structure} \rightarrow \text{sequent} \rightarrow \text{bool}$$

where $S \models s$ indicates that S satisfies (is a model of) s . We write $\models s$ to indicate that s is satisfied by all models in `structure`: $\forall S \in \text{structure}. S \models s$.

3.2 Rule Systems

We assume that the meaning of the rules is given by an effect relation

$$\text{eff} : \text{rule} \rightarrow \text{sequent} \rightarrow \text{sequent fset} \rightarrow \text{bool}$$

where $\alpha \text{ fset}$ denotes the set of finite subsets of α . The reading of $\text{eff } r \ s \ ss$ is as follows: Starting from sequent s , applying rule r expands s into the sequents ss . We can think of sequents as proof goals, each goal being replaced by zero or more subgoals by applying a rule. The triple $\mathcal{R} = (\text{sequent}, \text{rule}, \text{eff})$ forms a *rule system*.

Example 2 The Gentzen system from Section 2.1 can be presented as a rule system. The set sequent is the set of sequents, and rule consists of the following: a rule AX_a for each atom a ; rules NEGL_φ and NEGR_φ for each formula φ ; rules $\text{CONJL}_{\varphi,\psi}$ and $\text{CONJR}_{\varphi,\psi}$ for each pair of formulas φ and ψ ; a rule $\text{ALLL}_{x,\varphi,t}$ for each variable x , formula φ , and term t ; and a rule $\text{ALLR}_{x,\varphi}$ for each variable x and formula φ .

The eigenformula is part of the rule. Hence we have a countably infinite number of rules. The effect is defined as follows, where we use semicolons (;) to separate set elements:

$$\begin{aligned} \text{eff } \text{AX}_a (\Gamma, \text{Atm } a \triangleright \Delta, \text{Atm } a) & \emptyset \\ \text{eff } \text{NEGR}_\varphi (\Gamma \triangleright \Delta, \text{Neg } \varphi) & \{\Gamma, \varphi \triangleright \Delta\} \\ \text{eff } \text{NEGL}_\varphi (\Gamma, \text{Neg } \varphi \triangleright \Delta) & \{\Gamma \triangleright \Delta, \varphi\} \\ \text{eff } \text{CONJL}_{\varphi,\psi} (\Gamma, \text{Conj } \varphi \ \psi \triangleright \Delta) & \{\Gamma, \varphi, \psi \triangleright \Delta\} \\ \text{eff } \text{CONJR}_{\varphi,\psi} (\Gamma \triangleright \Delta, \text{Conj } \varphi \ \psi) & \{\Gamma \triangleright \Delta, \varphi; \Gamma \triangleright \Delta, \psi\} \\ \text{eff } \text{ALLL}_{x,\varphi,t} (\Gamma, \text{All } x \ \varphi \triangleright \Delta) & \{\Gamma, \text{All } x \ \varphi, \varphi[t/x] \triangleright \Delta\} \\ \text{eff } \text{ALLR}_{x,\varphi} (\Gamma \triangleright \Delta, \text{All } x \ \varphi) & \{\Gamma \triangleright \Delta, \varphi[y/x]\} \quad \text{where } y \text{ is fresh for } \Gamma \text{ and } \text{All } x \ \varphi \end{aligned}$$

3.3 Derivation Trees

Possibly infinite trees with nodes labeled by elements in a set α are represented by the following codatatype:

$$\text{codatatype } \alpha \text{ tree} = \text{Node } (\text{lab} : \alpha) (\text{sub} : (\alpha \text{ tree}) \text{ fset})$$

This definition introduces a constructor $\text{Node} : \alpha \rightarrow (\alpha \text{ tree}) \text{ fset} \rightarrow \alpha \text{ tree}$ and two selectors $\text{lab} : \alpha \text{ tree} \rightarrow \alpha$, $\text{sub} : \alpha \text{ tree} \rightarrow (\alpha \text{ tree}) \text{ fset}$. Trees have the form $\text{Node } a \ Ts$, where a is the tree's *label* and Ts is the finite set of its (immediate) *subtrees*. The **codatatype** keyword indicates that, unlike for inductive datatypes, this tree formation rule may be applied an infinite number of times to create infinite objects.

A *step* combines the current sequent and the rule to be applied: $\text{step} = \text{sequent} \times \text{rule}$. Derivation trees are defined as trees labeled by steps, $\text{dtree} = \text{step tree}$, in which the root's label (s, r) represents the proved goal s and the first (backward) applied rule r . The well-formed derivation trees are captured by the predicate $\text{wf} : \text{dtree} \rightarrow \text{bool}$ defined by the coinductive rule

$$\frac{\text{eff } r \ s \ (\text{image } (\text{fst} \circ \text{lab}) \ Ts) \quad \forall T \in Ts. \text{wf } T}{\text{wf } (\text{Node } (s, r) \ Ts)} \text{WF}$$

The term $\text{image } f \ A$ denotes the image of set A through function f , and fst is the left projection operator (i.e., $\text{fst } (x, y) = x$). The first assumption requires that the rule r from the root be applied to obtain the subtrees' labels. The second assumption requires that wellformedness hold for the immediate subtrees. The coinductive nature of the definition ensures that these properties hold for arbitrarily deep subtrees of T , even if T has infinite paths.

Double lines distinguish coinductive rules from their inductive counterparts. Thus, the predicate wf is the greatest (weakest) solution to the equivalence

$$\text{wf } (\text{Node } (s, r) \ Ts) \Leftrightarrow \text{eff } r \ s \ (\text{image } (\text{fst} \circ \text{lab}) \ Ts) \wedge (\forall T \in Ts. \text{wf } T)$$

$$\frac{\frac{\frac{}{\forall x. p(x), p(y) \triangleright p(y)}{AX_{p(y)}}}{\forall x. p(x) \triangleright p(y)} \quad \frac{\frac{}{\forall x. p(x), p(z) \triangleright p(z)}{AX_{p(z)}}}{\forall x. p(x) \triangleright p(z)} \quad \frac{}{\forall x. p(x) \triangleright p(y) \wedge p(z)} \text{CONJR}_{p(y), p(z)}}{\frac{}{\forall x. p(x), p(y) \triangleright p(z)} \text{ALLL}_{x, p(x), y}}{\forall x. p(x) \triangleright p(z)} \text{ALLL}_{x, p(x), z}}$$

Fig. 1 A proof

$$\frac{\frac{\frac{}{\forall x. p(x), p(y) \triangleright p(y)}{AX_{p(y)}}}{\forall x. p(x) \triangleright p(y)} \quad \frac{\frac{}{\forall x. p(x), p(y) \triangleright p(z)}{AX_{p(z)}}}{\forall x. p(x) \triangleright p(z)} \quad \frac{}{\forall x. p(x) \triangleright p(y) \wedge p(z)} \text{CONJR}_{p(y), p(z)}}{\frac{}{\forall x. p(x), p(y) \triangleright p(z)} \text{ALLL}_{x, p(x), y}}{\forall x. p(x) \triangleright p(z)} \text{ALLL}_{x, p(x), y}} \quad \frac{}{\forall x. p(x) \triangleright p(y) \wedge p(z)} \text{CONJR}_{p(y), p(z)}$$

Fig. 2 A failed proof attempt

which is also the greatest solution to the implication

$$\text{wf} (\text{Node } (s, r) \text{ } Ts) \Rightarrow \text{eff } r \text{ } s (\text{image } (\text{fst} \circ \text{lab}) \text{ } Ts) \wedge (\forall T \in Ts. \text{wf } T)$$

To establish a fact of the form $\forall T. P T \rightarrow \text{wf } T$ with $P : \text{dtree} \rightarrow \text{bool}$, a proof by *coinduction on the definition of wf* proceeds by simply showing that P is also a solution of the same implication:

$$P (\text{Node } (s, r) \text{ } Ts) \Rightarrow \text{eff } r \text{ } s (\text{image } (\text{fst} \circ \text{lab}) \text{ } Ts) \wedge (\forall T \in Ts. P T)$$

3.4 Proofs

The finite derivation trees—the trees that would result from an inductive datatype definition with the same constructors—can be carved out of the codatatype dtree using the predicate finite defined inductively (i.e., as a least fixpoint) by the rule

$$\frac{\forall T \in Ts. \text{finite } T}{\text{finite } (\text{Node } (s, r) \text{ } Ts)} \text{FIN}$$

Note the difference between inductive and coinductive definitions: An inductive definition repeatedly produces instances of the predicate, whereas a coinductive definition specifies tests that must be passed indefinitely by a particular value to be accepted. Indeed, the traditional view is that induction proceeds by constructing, coinduction by destructing.

A *proof* of a sequent s is a finite well-formed derivation tree with the sequent s at its root:

$$\text{proof } T \text{ } s \Leftrightarrow \text{finite } T \wedge \text{wf } T \wedge \text{fst } (\text{lab } T) = s$$

An infinite well-formed derivation tree represents a failed proof attempt.

Example 3 Given the instantiation of Example 2, Figure 1 shows a finite derivation tree for the sequent $\text{All } x (p(x) \triangleright \text{Conj } (p(y)) (p(z)))$ written using the familiar syntax for logical symbols. Figure 2 shows an infinite tree for the same sequent.

3.5 Soundness

We assume that the rules are locally sound.

$$\text{Local Soundness: } \forall r s ss. \text{eff } r s ss \wedge (\forall s' \in ss. \models s') \Rightarrow \models s$$

The soundness theorem follows by induction on the finiteness of trees representing proofs.

Theorem 4 Assume the rule system fulfills Local Soundness with respect to the structures. Then every sequent that has a proof is satisfied by all structures. Formally:

$$\forall s. (\exists T. \text{proof } T s) \Rightarrow \models s$$

3.6 Infinite Paths and König's Lemma

An infinite path in a derivation tree can be regarded as a way to “escape” the proof. To represent infinite paths independently of trees, we introduce the codatatype of streams over a type α with the constructor `SCons` and the selectors `shead` and `stail`:

$$\text{codatatype } \alpha \text{ stream} = \text{SCons } (\text{shead: } \alpha) (\text{stail: } \alpha \text{ stream})$$

The coinductive predicate `ipath : dtree \rightarrow step stream \rightarrow bool` determines whether a stream of steps is an infinite path in a tree:

$$\frac{T \in Ts \quad \text{ipath } T \text{ steps}}{\text{ipath } (\text{Node } (s, r) Ts) (\text{SCons } (s, r) \text{ steps})} \text{IPATH}$$

Since the trees are finitely branching, König's lemma applies. Its proof allows us to show a first simple corecursive definition.

Lemma 5 If the tree T is infinite, there exists an infinite path `steps` in T .

Proof By the contrapositive of `FIN`, if `Node (s, r) Ts` is infinite, there exists an infinite subtree $T \in Ts$. Let $f : \{T \in \text{dtree} \mid \neg \text{finite } T\} \rightarrow \{T \in \text{dtree} \mid \neg \text{finite } T\}$ be a function witnessing this fact—i.e., $f T$ is an immediate infinite subtree of T . The desired infinite path $p : \{T \in \text{dtree}. \neg \text{finite } T\} \rightarrow \text{step stream}$ can be defined by primitive corecursion over the codatatype of streams: $p T = \text{SCons } (\text{lab } T) (p (f T))$. Equivalently, in terms of the selectors:

$$\text{shead } (p T) = \text{lab } T \qquad \text{stail } (p T) = p (f T)$$

Thus, `ipath (p T) T` by straightforward coinduction on the definition of `ipath`. \square

The following extension of König's lemma applies to well-formed derivation trees, allowing one to construct an infinite path satisfying a unary and a binary invariant. Its statement involves the “always” predicate defined coinductively for streams of any set β , namely, `alw : (β stream \rightarrow bool) \rightarrow β stream \rightarrow bool`, where `alw P xs` states that the predicate P holds for all suffixes of `xs`:

$$\frac{P xs \quad \text{alw } P (\text{stail } xs)}{\text{alw } P xs} \text{ALW}$$

Lemma 6 Fix the set α and the predicates $I : \text{sequent} \times \alpha \rightarrow \text{bool}$ and $P : \text{sequent} \times \alpha \rightarrow \text{rule} \rightarrow \text{sequent} \times \alpha \rightarrow \text{bool}$. Assume that

$$\forall r s ss a. \text{eff } r s ss \wedge I(s, a) \Rightarrow (\exists s' a'. s' \in ss \wedge I(s', a') \wedge P(s, a) r(s', a'))$$

Then, if the tree T is well-formed, there exists a stream $steps \in (\text{step} \times \alpha)$ stream such that its first projection is an infinite path in T (formally, $\text{ipath } T (\text{smap } \text{fst } steps)$) and

$$\begin{aligned} \text{alw } (\lambda steps'. \text{let } ((s, r), a) = \text{shead } steps' \text{ and } ((s', _), a') = \text{shead } (\text{stail } steps') \\ \text{in } I(s, a) \wedge P(s, a) r(s', a')) \\ steps \end{aligned}$$

Above, the assumption is that, for any sequent s , rule r , and element $a \in \alpha$ such that the predicate I holds for (s, a) , there exists a premise sequent s' along the backward application of r and an element a' such that I again holds for (s', a') and the predicate P holds for (s, a) , r , and (s', a') . The conclusion is that there exists an infinite path in the tree along which I and P always hold.

The proof is similar to that of Lemma 5, but taking a slightly more complex function f , namely $f : B \rightarrow B$, where $B = \{(T, a) \in \text{dtree} \times \alpha \mid I(\text{fst } (\text{lab } T), a)\}$, such that $P(\text{fst } (\text{lab } T), a) r(\text{fst } (\text{lab } T'), a')$ holds whenever $(T, a) \in B$ and $(T', a') = f(T, a)$. The lemma's assumption ensures such a choice of f is possible.

3.7 Escape Paths

An *escape path* is a stream of steps that can form an infinite path in a derivation tree. It is defined coinductively as the predicate $\text{epath} : \text{step stream} \rightarrow \text{bool}$, which requires that every element in the given stream be obtained by applying an existing rule and choosing one of the resulting sequents:

$$\frac{\text{eff } r s ss \quad s' \in ss \quad \text{epath } (\text{SCons } (s', r') steps)}{\text{epath } (\text{SCons } (s, r) (\text{SCons } (s', r') steps))} \text{EPATH}$$

The following lemma is easy to prove by coinduction on the definition of epath .

Lemma 7 For any stream $steps$ and tree T , if $\text{wf } T$ and $\text{ipath } steps T$, then $\text{epath } steps$.

Example 8 The stream

$$(\forall x. p(x) \triangleright p(y) \wedge p(z)) \cdot (\forall x. p(x) \triangleright p(z)) \cdot (\forall x. p(x), p(y) \triangleright p(z))^\infty$$

where $s \cdot steps = \text{SCons } s steps$ and $s^\infty = s \cdot s \cdot \dots$, is an escape path for the tree of Figure 2.

3.8 Countermodel Paths

A countermodel path is a sequence of steps that witnesses the unprovability of a sequent s . Any escape path starting in s is a candidate for a countermodel path, given that it indicates a way to apply the proof rules without reaching any result. For it to be a genuine countermodel path, all possible proofs must have been attempted. More specifically, whenever a rule becomes enabled along the escape path, it is eventually applied later in the sequence. For FOL with its standard sequents, such paths can be used to produce actual countermodels

Remark 11 The saturation condition on step streams from Section 3.8 is a stronger condition than the standard properties of fairness and justice [18]. Fairness would require the rules to be continuously enabled to guarantee that they are eventually taken. The property of justice is stronger in that it would require the rules to be enabled infinitely often, but not necessarily continuously. Saturation goes further: If a rule is ever enabled, it will certainly be chosen at a later point. Saturation may seem too strong for the task at hand; however, in the presence of Persistence, the notions of fairness, justice, and saturation all coincide.

In addition, we assume a function $\text{herbrand} : \text{sequent} \rightarrow \text{step stream} \rightarrow \text{structure}$ that maps countermodel paths to actual countermodels:

$$\text{Herbrandness: } \forall s \text{ steps. countermodelPath } s \text{ steps} \Rightarrow \text{herbrand } s \text{ steps} \not\models s$$

A countermodel path provides an argument against provability. That this argument fully complements provability is essentially the content of the completeness theorem in its abstract form:

Lemma 12 Assume the rule system fulfills Availability and Persistence. Then every sequent admits a proof or a countermodel path. Formally:

$$\forall s. (\exists T. \text{proof } T \ s) \vee (\exists \text{steps. countermodelPath } s \ \text{steps})$$

Proof The proof uses the following combinators:

- $\text{stake} : \alpha \text{ stream} \rightarrow \text{nat} \rightarrow \alpha \text{ list}$ maps ρ and n to the list of the first n elements of ρ ;
- $\text{smap} : (\alpha \rightarrow \beta) \rightarrow \alpha \text{ stream} \rightarrow \beta \text{ stream}$ maps f to every element of the stream;
- $\text{nats} : \text{nat stream}$ denotes the stream of natural numbers: $0 \cdot 1 \cdot 2 \cdot 3 \cdot \dots$;
- $\text{flat} : (\alpha \text{ list}) \text{ stream} \rightarrow \alpha \text{ stream}$ maps a stream of finite nonempty lists to the stream obtained by concatenating those lists;
- $\text{sdropWhile} : (\alpha \rightarrow \text{bool}) \rightarrow \alpha \text{ stream} \rightarrow \alpha \text{ stream}$ removes the maximal prefix of elements that fulfill a given predicate from a given stream (or returns an irrelevant default value if the predicate holds for all elements of the stream).

We start by constructing a stream of rules fenum in a fair fashion, so that every rule occurs infinitely often in fenum . Let enum be a stream such that its elements cover the entire set rule (which is required to be countable). Take $\text{fenum} = \text{flat} (\text{smap} (\text{stake } \text{enum}) (\text{stail } \text{nats}))$. Thus, if $\text{enum} = r_1 \cdot r_2 \cdot r_3 \cdot \dots$, then $\text{fenum} = r_1 \cdot r_1 \cdot r_2 \cdot r_1 \cdot r_2 \cdot r_3 \cdot \dots$.

Let s be a sequent. Using fenum , we build a derivation tree T_0 labeled with s such that all its infinite paths are saturated. Let fair be the subset of rule stream consisting of the fair streams. Clearly, any suffix of an element in fair also belongs to fair . In particular, fenum and all its suffixes belong to fair . Given $\rho \in \text{fair}$ and $s \in \text{sequent}$, $\text{sdropWhile} (\lambda r. \neg \text{enabled } r \ s) \rho$ has the form $\text{SCons } r \ \rho'$, making r the first enabled rule in ρ . Such a rule exists because, by Availability, at least one rule is enabled in s and, by fairness, all the rules occur in ρ . Since $\text{enabled } r \ s$, we can pick a sequent set ss such that $\text{eff } r \ s \ ss$. We define $\text{mkTree} : \text{fair} \rightarrow \text{sequent} \rightarrow \text{dtree}$ corecursively as $\text{mkTree } \rho \ s = \text{Node} (s, r) (\text{image } (\text{mkTree } \rho') \ ss)$.

We prove that, for all $\rho \in \text{fair}$ and s , the derivation tree $\text{mkTree } \rho \ s$ is well formed and all its infinite paths are saturated. Wellformedness is obvious because at each point the continuation is built starting with the effect of a rule. For saturation, we show that if rule r is enabled at sequent s and $\text{ipath} (\text{mkTree } \rho \ s) \ \text{steps}$, then r appears along steps (i.e., there exists a sequent s' such that (s', r) is in steps). This follows by induction on the position of r in ρ , $\text{pos } r \ \rho$ —formally, the length of the shortest list ρ_0 such that $\rho = \rho_0 @ \text{SCons } r \ _$, where $@$ denotes concatenation. Let r' be the first rule from ρ enabled at sequent s . If $r = r'$,

then $\text{mkTree } \rho s$ has label (s, r) already. Otherwise, ρ has the form $\rho_1 @ [r'] @ \rho'$, with r not in ρ_1 , hence $\text{pos } r \rho' < \text{pos } r \rho$. From the definitions of ipath and mkTree , it follows that $\text{ipath } (\text{mkTree } \rho' s')$ (stail *steps*) holds for some $s' \in ss$ such that $\text{eff } r s' ss$. By the induction hypothesis, r appears along stail *steps*, hence along *steps* as desired. In particular, $T_0 = \text{mkTree fenum } s$ is well formed and all its infinite paths are saturated.

Finally, if T_0 is finite, it is the desired proof of s . Otherwise, by Lemma 5 (König) it has an infinite path. This path is necessarily saturated; by Lemma 7, it is the desired countermodel path. \square

Lemma 12 captures the abstract essence of arguments from the literature, although this is sometimes hard to grasp under the thick forest of syntactic details and concrete strategies for fair enumeration: A fair tree is constructed, which attempts a proof; in case of failure, the tree exhibits a saturated escape path. By Herbrandness, we immediately obtain completeness:

Theorem 13 Assume the rule system that fulfills Availability, Persistence, and Herbrandness. Then every sequent that is satisfied by all structures admits a proof:

$$\forall s. \models s \Rightarrow (\exists T. \text{proof } T s)$$

Proof Given s such that $\models s$, assume by absurdity that there exists no T such that $\text{proof } T s$. By Lemma 12, we obtain *steps* such that $\text{countermodelPath } s \text{ steps}$. Then, by Herbrandness, we have $\text{herbrand } s \text{ steps} \not\models s$, which contradicts $\models s$. \square

Remark 14 If we are not interested in witnessing the proof attempt closely, Lemma 12 can be established more directly by building the fair path without going through an intermediate fair tree. The key observation is that if a sequent s has no proof and $\text{eff } r s ss$, there must exist a sequent $s' \in ss$ that has no proof. (Otherwise, we could compose the proofs of all s' into a proof of s by applying rule r .) Let $\text{pick } r s ss$ denote such an s' . We proceed directly to the construction of a saturated escape path as a corecursive predicate $\text{mkPath} : \text{fair} \rightarrow \{s \in \text{sequent}. s \text{ has no proof}\} \rightarrow \text{step stream}$ following the same idea as for the previous tree construction (function mkTree): $\text{mkPath } \rho s = \text{SCons } (s, r) (\text{mkPath } \rho' (\text{pick } r s ss))$, where again $\text{SCons } r \rho' = \text{sdropWhile } (\lambda r. \neg \text{enabled } r s) \rho$ and ss is such that $\text{eff } r s ss$. Fairness of $\text{mkPath } \rho s$ follows by a similar argument as before for fairness of the tree.

3.10 Omitting the Availability Assumption

The abstract completeness result (Lemma 12) assumes Availability and Persistence. Among these assumptions, Persistence is essential: It ensures that the constructed fair path is saturated, meaning that every rule available at any point is eventually applied. Availability can be added later to the system without affecting its behavior by introducing a special “idle” rule.

Lemma 15 A rule system $\mathcal{R} = (\text{sequent}, \text{rule}, \text{eff})$ that fulfills Persistence can be transformed into an equivalent rule system $\mathcal{R}_{\text{idle}} = (\text{sequent}, \text{rule}_{\text{idle}}, \text{eff}_{\text{idle}})$ that fulfills both Persistence and Availability, with $\text{rule}_{\text{idle}} = \text{rule} \cup \{\text{IDLE}\}$ and eff_{idle} behaving like eff on rule and $\text{eff}_{\text{idle}} \text{ IDLE } s ss \Leftrightarrow ss = \{s\}$.

Proof Availability for the modified system follows from the continuous enabledness of IDLE. Persistence follows from the Persistence of the original system together with the property that IDLE is continuously enabled and does not alter the sequent. The modified system is equivalent to the original one because IDLE does not alter the sequent. \square

Now we can rephrase Theorem 13 in the context where we only assume Persistence and a slight variation of the Herbrandness condition, where all the concepts refer as before to the rule system \mathcal{R} , except for $\text{countermodelPath}_{\text{idle}}$ which refers to $\mathcal{R}_{\text{idle}}$:

$$\text{Herbrandness}_{\text{idle}}: \forall s \text{ steps. } \text{countermodelPath}_{\text{idle}} s \text{ steps} \Rightarrow \text{herbrand } s \text{ steps} \not\equiv s$$

Theorem 16 Assume the rule system \mathcal{R} fulfills Persistence and $\text{Herbrandness}_{\text{idle}}$. Then every sequent that has a proof in \mathcal{R} is satisfied by all structures.

Proof We first apply Lemma 12 to the system $\mathcal{R}_{\text{idle}}$ to obtain that every sequent admits either a proof or a countermodel path, both in this system. And since \mathcal{R} and $\mathcal{R}_{\text{idle}}$ are equivalent, any proof of $\mathcal{R}_{\text{idle}}$ yields one of \mathcal{R} . Conversely, any proof of $\mathcal{R}_{\text{idle}}$ can be transformed into a proof of \mathcal{R} by omitting all applications of IDLE. We thus proved that every sequent admits a proof over \mathcal{R} or a countermodel path over $\mathcal{R}_{\text{idle}}$. It remains to apply $\text{Herbrandness}_{\text{idle}}$, just like we did with Herbrandness for Theorem 16. \square

Remark 17 The addition of IDLE is inspired by, and similarly motivated as, that of idle transitions to Kripke structures in the context of temporal logic, where it is technically convenient to consider only infinite paths.

4 Concrete Instances of Soundness and Completeness

The abstract soundness is based on the Local Soundness assumption, which is easy to verify for all the considered instances—therefore, below we focus on completeness.

4.1 Classical First-Order Logic

The abstract completeness proof is parameterized by a rule system. This section concretizes the result for the Gentzen system from Section 2.1 to derive the standard completeness theorem. Example 2 recast it as a rule system; we must verify that it fulfills the Persistence and Herbrandness conditions.

The Gentzen rules are syntax-directed in that they operate on formulas having specific connectives or quantifiers at the top. This is what guarantees Persistence. For example, an application of AX_a (which affects only the atom a) leaves any potential enabledness of $ALLL_{x,\varphi,t}$ (which affects only formulas with All at the top) unchanged, and vice versa; moreover, AX_a does not overlap with AX_b for $a \neq b$. A minor subtlety concerns $ALLR_{x,\varphi}$, which requires the existence of a fresh y in order to be enabled. The Persistence proof goes through because the sequents are finite, so we can always find a fresh variable in the countably infinite set var . On the other hand, Availability does not hold for the proof system; for example, the sequent $p(x) \triangleright q(x)$ has no enabled rule. Hence, we need Theorem 16 and its IDLE rule.

To infer the standard completeness theorem from Theorem 16, it suffices to define a suitable function herbrand . Let steps be a countermodel path for $\Gamma \triangleright \Delta$ (i.e., a saturated escape path with $\Gamma \triangleright \Delta$ as the first sequent). Let $\tilde{\Gamma}$ be the union of the left-hand sides of sequents occurring in σ , and let $\tilde{\Delta}$ be the union of the corresponding right-hand sides. Clearly, $\Gamma \subseteq \tilde{\Gamma}$ and $\Delta \subseteq \tilde{\Delta}$.

We define $\text{herbrand}(\Gamma \triangleright \Delta) \text{ steps}$ to be $\mathcal{S} = (S, F, P)$, where

- S is the set of terms, term;
- for each n -ary f and p and each $t_1, \dots, t_n \in S$:

- $F_f(t_1, \dots, t_n) = f(t_1, \dots, t_n)$;
- $P_p(t_1, \dots, t_n) \Leftrightarrow p(t_1, \dots, t_n) \in \tilde{\Gamma}$.

Lemma 18 herbrand $(\Gamma \triangleright \Delta)$ *steps* is a countermodel for $\Gamma \triangleright \Delta$, in that there exists a valuation $\xi : \text{var} \rightarrow S$ such that $\mathcal{S} \not\models_{\xi} \Gamma \triangleright \Delta$.

Proof First, the pair $(\tilde{\Gamma}, \tilde{\Delta})$ from the definition of herbrand can be shown to be well behaved with respect to all the connectives and quantifiers in the following sense:

1. For all atoms a , $\text{Atm } a \notin \tilde{\Gamma} \cap \tilde{\Delta}$.
2. If $\text{Neg } \varphi \in \tilde{\Gamma}$, then $\varphi \in \tilde{\Delta}$.
3. If $\text{Neg } \varphi \in \tilde{\Delta}$, then $\varphi \in \tilde{\Gamma}$.
4. If $\text{Conj } \varphi \psi \in \tilde{\Gamma}$, then $\varphi \in \tilde{\Gamma}$ and $\psi \in \tilde{\Gamma}$.
5. If $\text{Conj } \varphi \psi \in \tilde{\Delta}$, then $\varphi \in \tilde{\Delta}$ or $\psi \in \tilde{\Delta}$.
6. If $\text{All } x \varphi \in \tilde{\Gamma}$, then $\varphi[t/x] \in \tilde{\Gamma}$ for all t .
7. If $\text{All } x \varphi \in \tilde{\Delta}$, there exists a variable y such that $\varphi[y/x] \in \tilde{\Delta}$.

A pair $(\tilde{\Gamma}, \tilde{\Delta})$ satisfying these properties is sometimes called a Hintikka set [1, 19]. These properties follow from the saturation of σ with respect to the corresponding rules. The proofs are routine. For example:

1. If $\text{Atm } a \in \tilde{\Gamma} \cap \tilde{\Delta}$, the rule Ax_a is enabled in σ and hence, by saturation, it is eventually taken. This is impossible since this is a rule without premises, whose application would prevent the continuation of the infinite sequence σ .
6. If $\text{All } x \varphi \in \tilde{\Gamma}$ and t is a term, $\text{ALLL}_{x,\varphi,t}$ is enabled in σ and hence eventually taken, ensuring that $\varphi[t/x] \in \tilde{\Gamma}$.

Let ξ be the embedding of variables into terms. To prove $\mathcal{S} \not\models_{\xi} \Gamma \triangleright \Delta$, it suffices to show that $\forall \varphi \in \tilde{\Gamma}. \mathcal{S} \models_{\xi} \varphi$ and $\forall \varphi \in \tilde{\Delta}. \mathcal{S} \not\models_{\xi} \varphi$. These two facts follow together by induction on the depth of φ . In the base case, if $\text{Atm } a \in \tilde{\Gamma}$, then $\mathcal{S} \models_{\xi} \text{Atm } a$ follows directly from the definition of \mathcal{S} ; moreover, if $\text{Atm } a \in \tilde{\Delta}$, then by property 1 $\text{Atm } a \notin \tilde{\Gamma}$, hence again $\mathcal{S} \not\models_{\xi} \text{Atm } a$ follows from the definition of \mathcal{S} . The only nontrivial inductive case is All , which requires Lemma 19. Assume $\text{All } x \varphi \in \tilde{\Gamma}$. By property 6, we have $\varphi[t/x] \in \tilde{\Gamma}$ for any t . Hence, by the induction hypothesis, $\mathcal{S} \models_{\xi} \varphi[t/x]$. By Lemma 19, $\mathcal{S} \models_{\xi[x \leftarrow t]} \varphi$ for all t ; that is, $\mathcal{S} \models_{\xi} \text{All } x \varphi$. The second fact, concerning $\tilde{\Delta}$, follows similarly from property 7. \square

Above, we used the substitution lemma, which relates the notions of satisfaction and substitution:

Lemma 19 $\mathcal{S} \models_{\xi} \varphi[t/x]$ iff $\mathcal{S} \models_{\xi[x \leftarrow [t]_{\xi}]} \varphi$.

We have thus obtained:

Corollary 20 A sequent is provable by a (finite) proof in the Gentzen system of classical FOL iff it is satisfied by all FOL structures.

Remark 21 The rule ALLL stores, in the left context, a copy of the universal formula $\text{All } x \varphi$ when applied backward. This is crucial for concrete completeness since a fair enumeration should try all the t instances of the universally quantified variable x , which requires Availability of $\text{All } x \varphi$ even after its use. If we labeled ALLL as $\text{ALLL}_{x,\varphi}$ instead of $\text{ALLL}_{x,\varphi,t}$, thereby delegating the choice of t to the nondeterminism of eff , the system would still be persistent as required by the abstract completeness proof, but Lemma 18 (and hence concrete completeness) would not hold—property 6 from the lemma’s proof would fail.

4.2 Further Instances

Theorem 16 is applicable to classical FOL Gentzen systems from the literature, in several variants: with sequent components represented as lists, multisets or sets, one-sided or two-sided, and so on. This includes the systems G' , $GCNF'$, G , and G_- from Gallier [19] and the systems $G1$, $G2$, $G3$, $GS1$, $GS2$, and $GS3$ from Troelstra and Schwichtenberg [47]. Persistence is easy to check. The syntax-independent part of the argument is provided by Theorem 16, while an ad hoc step analogous to Lemma 18 is required to build a concrete countermodel from a countermodel path to complete the proof.

Several FOL refutation systems based on tableaux or resolution are instances of the abstract theorem, providing that we read the abstract notion of “proof” as “refutation” and “countermodel” as “model.” Nondestructive tableaux [23]—including those presented in Bell and Machover [1] and in Fitting [17]—are usually persistent when regarded as derivation systems. After an application of Theorem 16, the ad hoc argument for interpreting the abstract model is similar to that for Gentzen systems (Lemma 18).

Regrettably, abstract completeness is not directly applicable beyond classical logic. It is generally not clear how to extract a specific model from a nonstandard logic from an abstract (proof-theoretic) model. Another issue is that standard sequent systems for nonclassical variations of FOL such as modal or intuitionistic logics do not satisfy Persistence. A typical right rule for the modal operator \Box (“must”) is as follows [47]:

$$\frac{\Box \Gamma \triangleright \Diamond \Delta, \varphi}{\Box \Gamma \triangleright \Diamond \Delta, \Box \varphi} \text{ MUSTR}$$

To be applicable, the rule requires that all the formulas in the context surrounding the eigenformula have \Box or \Diamond at the top. Other rules may remove these operators, or introduce formulas that do not have them, thus disabling MUSTR.

Recent work targeted at simplifying completeness arguments [34] organizes modal logics as labeled transition systems, for which Kripke completeness is derived. (The technique also applies to the completeness of intuitionistic logic with respect to Kripke semantics.) In the proposed systems, the above rule becomes

$$\frac{\Gamma, w R w' \triangleright \Delta, w' : \varphi}{\Gamma \triangleright \Delta, w : \Box \varphi} \text{ MUSTR}' \quad (w' \text{ fresh})$$

The use of labels for worlds (w, w') and the bookkeeping of the accessibility relation R makes it possible to recast the rule so that either no facts (as above) or only resilient facts are ever assumed about the surrounding context. The resulting proof system satisfies Persistence, enabling Theorem 16. The Kripke countermodel construction is roughly as for classical FOL Gentzen systems.

5 Abstract Infinite-Proof Soundness

In the previous sections, completeness is established by analyzing the interplay between the existence of *finite* proof trees (representing valid proofs) and certain *infinite* proof trees (used to produce countermodels).

In this section, we look into when can *infinite* proof trees be accepted as valid proofs. In this process, we give a coinductive account of the abstract development of Brotherston

$$\frac{\frac{\text{even}(0) \triangleright \text{odd}(\text{Suc}(0))}{\text{ODD}_0} \quad \frac{\frac{\text{Link } L}{\text{even}(y) \triangleright \text{odd}(\text{Suc}(y))} \text{SUBST}_{y,x}}{\text{even}(y) \triangleright \text{odd}(\text{Suc}(\text{Suc}(\text{Suc}(y))))} \text{ODD}_{\text{Suc}}}{\text{even}(x) \triangleright \text{odd}(\text{Suc}(x))} \text{EVEN}_{\text{split}}$$

Fig. 4 A cyclic proof tree

$$\frac{\frac{\frac{\text{even}(0) \triangleright \text{odd}(\text{Suc}(0))}{\text{ODD}_0} \quad \frac{\frac{\text{even}(x) \triangleright \text{odd}(\text{Suc}(x))}{\text{even}(y) \triangleright \text{odd}(\text{Suc}(y))} \text{SUBST}_{y,x}}{\text{even}(y) \triangleright \text{odd}(\text{Suc}(\text{Suc}(\text{Suc}(y))))} \text{ODD}_{\text{Suc}}}{\text{even}(x) \triangleright \text{odd}(\text{Suc}(x))} \text{EVEN}_{\text{split}}}{\text{even}(x) \triangleright \text{odd}(\text{Suc}(x))}$$

Fig. 5 The infinite proof tree corresponding to the cyclic proof tree of Figure 4

et al. [14], slightly more general since we work with arbitrary infinite proofs and not just cyclic proofs.

We first recall some motivation and intuition about cyclic proofs, in the context of the IFOL logic from Section 2.2. It is clear that the rules of the IFOL Gentzen are (locally) sound for inductive structures. Hence, any finite proof tree built using these rules represents a valid proof, in that its root sequent is known to be satisfied by all inductive structures. On the other hand, for the language of Example 1 from Section 2.2, consider the cyclic tree of Figure 4, where one of the leaves is not an axiom, but rather a link L “back” to the root of the tree (decorated with L). A cyclic proof indicates that, when L is reached, the (backward) proof continues with the sequent that L points to. Thus, the intended proof corresponding to the cyclic tree from Figure 4 is the infinite tree from Figure 5. We first cover infinite proof trees in general and then come back to cyclic proof trees.

5.1 Soundness of Infinite Proof Trees

As before, we fix the countable sets `sequent` and `rule` for sequents and rules, the class `structure`, and the satisfaction relation $\models : \text{structure} \rightarrow \text{sequent} \rightarrow \text{bool}$, writing $\models s$ for $\forall S \in \text{structure}. S \models s$. In addition, we fix the following:

- a set marker of items called *markers*;
- a *marking function* $\delta : \{(s, r, s') \in \text{sequent} \times \text{rule} \times \text{sequent} \mid \exists ss. \text{eff } s r ss \wedge s' \in ss\} \rightarrow (\text{marker} \times \text{bool} \times \text{marker}) \text{ set}$;
- an ordinal `ord`, with $<$ and \leq denoting its strict and nonstrict order relations;
- a *height function* $\sigma : \text{marker} \times \text{structure} \rightarrow \text{ord}$.

The marking function associates a set of triples (M, b, M') with every backward application (s, r, s') of every rule. In (s, r, s') , the rule r is thought of as being applied to s and then one of the resulting sequents, s' , being chosen. In (M, b, M') , the Boolean value b being `False` means “stay” and b being `True` means “decrease.” The “stay” and “decrease” directives refer to how the height function σ should evolve along this rule application: Given a sequent s , a countermodel S for it and a rule r that yields a set of sequents ss when applied to s , there should exist a sequent $s' \in ss$ and a countermodel S' of s' such that, for all $(M, b, M') \in \delta(s, r, s')$, when moving from (M, S) to (M', S') :

- the height should at least stay the same (or decrease) when b says “stay”;
- the height should decrease when b says “decrease.”

Formally, we postulate the following condition:

Descent: For all S and s such that $S \not\models s$ and all r, ss such that $\text{eff } s \ r \ ss$, there exist S' and $s' \in ss$ such that $S' \not\models s'$ and $\text{descent } (s, S) \ r \ (s', S')$.

Above, $\text{descent} : \text{sequent} \times \text{structure} \rightarrow \text{rule} \rightarrow \text{sequent} \times \text{structure} \rightarrow \text{bool}$ is the following predicate:

$$\begin{aligned} \text{descent } (s, S) \ r \ (s', S') &\Leftrightarrow \\ (\forall M \ b \ M'. (M, b, M') \in \delta (s, r, s') \Rightarrow & \\ (b = \text{False} \wedge \sigma (M', S') \leq \sigma (M, S)) \vee (b = \text{True} \wedge \sigma (M', S') < \sigma (M, S))) & \end{aligned}$$

Remark 22 The Descent condition is a strengthening of the Local Soundness condition from Section 3.5: Removing the “and $\text{descent } (s, S) \ r \ (s', S')$ ” part yields the contrapositive of Local Soundness. Essentially, Descent is a form of Local Soundness with the additional requirement that the marking function’s directives must be respected.

Under the Descent assumption, one can identify certain “good” infinite proof trees, which can be accepted as valid proofs along with the finite ones. First, we need the notion of a stream of Booleans and a stream of markers following an infinite path. The predicate $\text{follow} : \text{bool stream} \rightarrow \text{marker stream} \rightarrow \text{step stream} \rightarrow \text{bool}$ is defined coinductively:

$$\frac{(M, b, \text{shead } Ms) \in \delta (s, r, \text{fst } (\text{shead } steps)) \quad \text{follow } bs \ Ms \ steps}{\text{follow } (\text{SCons } b \ bs) (\text{SCons } M \ Ms) (\text{SCons } (s, r) \ steps)} \text{ FOLLOW}$$

Thus, $\text{follow } bs \ Ms \ steps$ states that, for any consecutive steps $(s, r), (s', _)$ in $steps$, the corresponding markers M, M' in Ms and the Boolean corresponding to the first of these, b in bs , have the property that $(M, b, M') \in \delta (s, r, s')$; in other words, the streams bs and Ms represent choices of the sets of markers for $steps$.

We define a tree to be good if each of its infinite paths have some streams of Booleans and markers that eventually follow it (i.e., follow a suffix of it), in such a way that the Booleans indicate infinite decrease:

$$\text{good } T \Leftrightarrow (\forall steps. \text{ipath } T \ steps \Rightarrow \text{ev } (\lambda steps'. \exists bs \ Ms. \text{follow } bs \ Ms \ steps' \wedge \text{infDecr } bs) \ steps)$$

where the infinite decrease of a Boolean stream simply means True (“decrease”) occurring infinitely often:

$$\text{infDecr} = \text{alw } (\text{ev } (\lambda bs. \text{shead } bs = \text{True}))$$

A (potentially) infinite proof of a sequent, or *iproof*, is then defined as a good, well-formed tree having that sequent at the root:

$$\text{iproof } T \ s \Leftrightarrow \text{wf } T \wedge \text{good } T \wedge \text{fst } (\text{lab } T) = s$$

Since finite trees (have no infinite paths and hence) are trivially good, (finite) proofs are particular cases of iproofs. Our goal is to show that iproofs are also sound:

Theorem 23 Assume the rule system fulfills Descent. Then every sequent that has an iproof is satisfied by all structures:

$$\forall s. (\exists T. \text{iproof } T \ s) \Rightarrow \models s$$

Proof Fix s and T such that $\text{iproof } T \ s$ and assume by absurdity that $\not\models s$, meaning there exists S such that $S \not\models s$. To obtain a contradiction, we proceed as follows:

1. Applying Lemma 6 (Section 3.6) for $\alpha = \text{structure}$, $I = \lambda(s, S)$. $S \not\models s$ and $P = \text{descent}$ to the Descent assumption, we obtain a stream stepSs of step–structure pairs (i.e., of sequent–rule–structure triples) $((s, r), S)$, such that, at each point in the stream, the structure S is a countermodel for the sequent s and descent holds for any two consecutive elements. Formally, we have $\text{alwCmodDesc } \text{stepSs}$ (“always countermodel and descending”), where alwCmodDesc is defined as

$$\text{alw } (\lambda \text{stepSs. let } ((s, r), S) = \text{shead } \text{stepSs} \text{ and } ((s', _), S') = \text{shead } (\text{stail } \text{stepSs}) \\ \text{in } S \not\models s \wedge \text{descent } (s, S) \ r \ (s', S'))$$

and the step components of stepSs form an infinite path in T : $\text{ipath } T \ (\text{smap } \text{fst } \text{stepSs})$.

2. Using the goodness of T , we obtain the streams bs and Ms that follow a suffix steps' of $\text{smap } \text{fst } \text{stepSs}$:

$$\exists \text{steps}' \ \text{steps}'' \ bs \ Ms. \ \text{smap } \text{fst } \text{stepSs} = \text{steps}'' @ \text{steps}' \wedge \text{follow } bs \ Ms \ \text{steps}' \wedge \text{infDecr } bs$$

where $@$ denotes concatenation of a list and a stream.

3. Taking the suffix steps' of stepSs that corresponds to the suffix steps' of its step component $\text{smap } \text{fst } \text{stepSs}$ (formally, defining stepSs' to be $\text{stake } (\text{length } \text{steps}') \ \text{stepSs}$), we end up with the streams stepSs' , bs and Ms such that
 - follow $bs \ Ms$ ($\text{smap } \text{fst } \text{stepSs}'$);
 - $\text{infDecr } bs$;
 - $\text{alwCmodDesc } \text{stepSs}'$ (because alw is invariant under suffix).
4. Let zip denote the pair-zipping of two streams, defined by primitive corecursion as $\text{zip } (\text{SCons } x \ xs) \ (\text{SCons } y \ ys) = \text{SCons } (x, y) \ (\text{zip } xs \ ys)$. Taking $ks = \text{smap } \sigma \ (\text{zip } (\text{smap } \text{fst } \text{stepSs}') \ Ms)$ (which is a stream of ord elements), we have that ks satisfies the following properties:
 - It is always nonstrictly decreasing: $\text{alw } (\lambda ks'. \text{shead } ks' \geq \text{shead } (\text{stail } ks')) \ ks$.
 - It is infinitely often strictly decreasing: $\text{alw } (\text{ev } (\lambda ks'. \text{shead } (\text{stail } ks') > \text{shead } ks')) \ ks$.
 This follows easily by conduction using the gathered properties for stepSs' , bs , and Ms .
5. From ks , we obtain the substream ks' that is always strictly decreasing:

$$\text{alw } (\lambda ks'. \text{shead } ks' > \text{shead } (\text{stail } ks')) \ ks$$

which is in contradiction with the well-foundedness of the order $<$ on the ordinal ord . \square

One detail not explained in the above proof is the construction of ks' , a stream of strictly decreasing items, from ks , a stream of nonstrictly decreasing but always eventually strictly decreasing items. Informally, if ks has the form $k_0 \cdot k_1 \cdot \dots$ with $k_0 = k_1 = \dots = k_n > k_{n+1} = k_{n+2} = \dots = k_{n+m} > k_{n+m+1} = \dots$, then ks' will be $k_0 \cdot k_{n+1} \cdot k_{n+m+1} \cdot \dots$; that is, ks' will only contain the “jump” elements. Formally, this construction can be compactly described as $ks' = \text{bfilter } (>, ks)$, where $> : \text{ord} \rightarrow \text{ord} \rightarrow \text{bool}$ is the dual strict order on ord . The general-purpose polymorphic “binary filter” combinator, $\text{bfilter} : D_\alpha \rightarrow \alpha \ \text{stream}$, has as domain the subset D_α of $(\alpha \rightarrow \text{bool}) \times \alpha$ consisting of pairs (P, xs) such that, for every element x in xs , there exists a later element y in xs such that $P \ x \ y$:

$$D_\alpha = \{(P, xs) \mid \text{alw } (\lambda xs'. \text{ev } (\lambda xs''. P \ (\text{shead } xs') \ (\text{shead } xs'')) \ (\text{stail } xs')) \ xs\}$$

The `bfilter` function is defined as follows, where for readability we write `bfilterP` instead of `bfilter (P, _)`:

$$\begin{aligned} \text{bfilter}_P xs &= (\text{if } P (\text{shead } xs) (\text{shead } (\text{stail } xs)) \\ &\quad \text{then } \text{SCons } (\text{shead } xs) (\text{bfilter}_P (\text{stail } xs)) \\ &\quad \text{else } \text{bfilter}_P (\text{SCons } (\text{shead } xs) (\text{stail } (\text{stail } xs))) \end{aligned}$$

Thus, `bfilterP` collects from a stream all the pairs of elements proximally related by P and forms a new stream with them. Note that its definition is neither purely recursive nor purely corecursive. Whereas the call on the `then` branch is corecursive (showing how to produce the first element in the result stream, `shead xs`), the call on the `else` branch does not exhibit any explicit productivity. However, the overall function is well-defined (and indeed productive) because, thanks to the choice of the domain, at each point there will only be a finite number of consecutive calls on the `else` branch, just until the encountered element will be in relation P with `shead xs`. In summary, the *recursive* call on the `else` branch concurs with the *corecursive* call on the `then` branch for the well-definedness of this function.

Thanks to recent infrastructure development [10], Isabelle is willing to accept this mixed recursive–corecursive definition, in the face of evidence for the terminating recursiveness of the `else` branch calls: If $(P, xs) \in D_\alpha$, then in particular $\text{ev } (\lambda xs''. P (\text{shead } xs) (\text{shead } xs''))$ holds for `stail xs`; hence, if we write xs' for the input to the recursive call, `SCons (shead xs) (stail (stail xs))`, we have that `shead xs' = shead xs` and `stail xs' = stail (stail xs)`. Hence:

$$\text{num}_P (\text{shead } xs') (\text{stail } xs') = \text{num}_P (\text{shead } xs) (\text{stail } (\text{stail } xs)) < \text{num}_P (\text{shead } xs) (\text{stail } xs)$$

where `numP z ys` is the number of elements that need to be consumed from `ys` before reaching some `y` such that $P z y$. The use of such sophisticated corecursive machinery for this not so complex example may seem excessive. But note that `bfilter` is a general-purpose combinator, defined once and usable for arbitrary predicates P .

For the above example, namely, $P = >$ and $k_0 = k_1 = \dots = k_n > k_{n+1} = k_{n+2} = \dots = k_{n+m} > k_{n+m+1} = \dots$, the execution of `bfilter` proceeds as follows:

$$\begin{aligned} &\text{bfilter}_P (k_0 \cdot k_1 \cdot \dots \cdot k_n \cdot k_{n+1} \cdot k_{n+2} \cdot \dots \cdot k_{n+m} \cdot k_{n+m+1} \cdot \dots) \\ &= \text{bfilter}_P (k_0 \cdot k_2 \cdot \dots \cdot k_n \cdot k_{n+1} \cdot k_{n+2} \cdot \dots \cdot k_{n+m} \cdot k_{n+m+1} \cdot \dots) \\ &\quad \vdots \\ &= \text{bfilter}_P (k_0 \cdot k_{n+1} \cdot k_{n+2} \cdot \dots \cdot k_{n+m} \cdot k_{n+m+1} \cdot \dots) \\ &= k_0 \cdot \text{bfilter}_P (k_{n+1} \cdot k_{n+2} \cdot \dots \cdot k_{n+m} \cdot k_{n+m+1} \cdot \dots) \\ &\quad \vdots \\ &= k_0 \cdot \text{bfilter}_P (k_{n+1} \cdot k_{n+m} \cdot k_{n+m+1} \cdot \dots) \\ &= k_0 \cdot k_{n+1} \cdot \text{bfilter}_P (k_{n+m+1} \cdot \dots) \\ &\quad \vdots \end{aligned}$$

5.2 From Cyclic Trees to Infinite Trees

Having the general result for the soundness of infinite trees in place, let us come back to the notion of cyclic tree, which formed our original motivation. In addition to the sets `sequent` and `rule`, we assume a set `link` of links. The set of cyclic derivation trees is introduced as a datatype:

$$\begin{aligned} \text{datatype } \text{cdtree} &= \text{Node } \alpha ((\alpha \text{ cdtree}) \text{ fset}) \\ &\quad | \text{Link link} \end{aligned}$$

Thus, there are two differences between the derivation trees we used so far (*dtree*) and the cyclic derivation trees: the latter are restricted to be finite and are allowed to have, in addition to usual nodes, also links to other cyclic derivation trees. To animate the links, we fix a function $\text{pointsTo} : \text{link} \rightarrow \text{cdtree}$ (showing, for each link, where it points to), for which we assume the following property:

Good Links: The target of pointsTo is never a link: $\forall L L'. \text{pointsTo } L \neq \text{Link } L'$.

Each cyclic derivation tree T yields a derivation tree $\text{treeOf } T$ as follows: T is traversed as if recursively producing $\text{treeOf } T$ by the application of the same rules, and when reaching a $\text{Link } L$ one moves to the pointed cyclic tree $\text{pointsTo } L$. Because $\text{pointsTo } L$ can be arbitrarily large (in particular, larger than T), the definition of $\text{treeOf} : \text{cdtree} \rightarrow \text{dtree}$ cannot proceed recursively on the domain, cdtree . However, it can naturally proceed corecursively on the codomain, dtree :

$$\begin{aligned} \text{treeOf } (\text{Node } (s, r) \ Ts) &= \text{Node } (s, r) \ (\text{fimage } \text{treeOf } Ts) \\ \text{treeOf } (\text{Link } L) &= \text{treeOf } (\text{pointsTo } L) \end{aligned}$$

Strictly speaking, the definition is not purely corecursive: In the Link case, it is not apparent what is the first layer of the result tree. However, since $\text{treeOf } (\text{pointsTo } L)$ is guaranteed to not have the form $\text{Link } L'$, hence start with the Node constructor, we know that after exactly two calls we reach the Node case—so Isabelle has no problem accepting this definition, again as a particular case of recursion–corecursion mixture.

The root sequent of a cyclic derivation tree, $\text{seqOf} : \text{cdtree} \rightarrow \text{sequent}$, is defined as expected, following the link when necessary:

$$\begin{aligned} \text{seqOf } (\text{Node } (s, r) \ Ts) &= (s, r) \\ \text{seqOf } (\text{Link } L) &= \text{seqOf } (\text{pointsTo } L) \end{aligned}$$

Even though cyclic derivation trees are finite entities, they can exhibit infinite behavior through the links. Therefore, we must define their wellformedness $\text{cwf} : \text{cdtree} \rightarrow \text{bool}$ not inductively, but coinductively, by the following rules:

$$\frac{\text{eff } r \ s \ (\text{image } \text{seqOf } Ts) \quad \forall T \in Ts. \ \text{cwf } T}{\text{cwf } (\text{Node } (s, r) \ Ts)} \text{ CWF-NODE} \quad \frac{\text{cwf } (\text{pointsTo } L)}{\text{cwf } (\text{Link } L)} \text{ CWF-LINK}$$

The CWF-NODE rule is essentially the same as for derivation trees, whereas CWF-LINK asks the trees pointed by links to be well-formed as well.

Now we can define the notion of being a cyclic proof for a sequent:

$$\text{cproof } T \ s \Leftrightarrow \text{cwf } T \wedge \text{good } (\text{treeOf } L) \wedge \text{seqOf } T = s$$

Goodness being a property stated in terms of infinite traces, it naturally belongs to the (infinite) derivation tree of a cyclic derivation tree. It is easy to see that the root sequence of a cyclic tree T is the same as its generated derivation tree $\text{treeOf } T$, and to prove by induction that wellformedness of T implies wellformedness of $\text{treeOf } T$. With Theorem 23, this immediately gives our desired soundness theorem for cyclic derivation trees:

Theorem 24 Assume the rule system fulfills Descent and the function pointsTo fulfills Good Links. Then every sequent that has a cyclic proof is satisfied by all structures:

$$\forall s. (\exists T. \text{cproof } T \ s) \Rightarrow \models s$$

6 Concrete Instances of Infinite-Proof Soundness

Here we consider instances of the abstract development establishing soundness of infinite proofs. We discuss in detail our running example, FOL with inductive definitions. Then we briefly mention other possible instances.

6.1 First-Order Logic with Inductive Predicates

We consider IFOL from Section 2.2 and its associated Gentzen system. The abstract structures (elements of the set structure) are not instantiated to mere concrete structures, but rather to pairs (\mathcal{S}, ξ) where $\mathcal{S} = (S, (F_f)_{f \in \text{fsym}}, (P_p)_{p \in \text{psym}})$ is a structure and $\xi : \text{var} \rightarrow S$ is a variable valuation. We instantiate the satisfaction relation, $(\mathcal{S}, \xi) \models \varphi$, to $\mathcal{S} \models_{\xi} \varphi$.

To instantiate Theorems 23 and 24, we must define the notion of marker (forming the set marker), the ordinal ord, and the functions $\delta : \{(s, r, s') \in \text{sequent} \times \text{rule} \times \text{sequent} \mid \exists ss. \text{eff } s \ r \ ss \wedge s' \in ss\} \rightarrow (\text{marker} \times \text{bool} \times \text{marker})$ set and $\sigma : \text{marker} \times \text{structure} \rightarrow \text{ord}$ and verify the Descent property for them.

We take marker to be the set of inductive predicate atoms: $\{p(\bar{t}) \mid p \in \text{ipsym}\}$. When defining δ on (s, r, s') , we distinguish three cases:

- r is any rule not having the form $\text{SUBST}_{t,x}$ or p_{split} . Then, assuming $s = (\Gamma \triangleright \Delta)$, we set

$$\delta(s, r, s') = \{(M, \text{False}, M) \mid M \in \text{marker} \cap \Gamma\}$$

- r has the form $\text{SUBST}_{t,x}$. Then s has the form $\Gamma[t/x] \triangleright \Delta[t/x]$, and we set

$$\delta(s, r, s') = \{(M[t/x], \text{False}, M) \mid M \in \text{marker} \cap \Gamma\}$$

- r has the form p_{split} for some $p \in \text{ipsym}$. Then s has the form $\Gamma, p(\bar{x}) \triangleright \Delta$ and s' has the form $\Gamma[\bar{t}/\bar{x}], \text{prems}(\chi') \triangleright \Delta[\bar{t}/\bar{x}]$ for some $\chi \in \text{ind}_p$. We set

$$\delta(s, r, s') = \{(M, \text{False}, M[\bar{t}/\bar{x}]) \mid M \in \text{marker} \cap \Gamma\} \cup \{(p(\bar{x}), \text{True}, \psi) \mid \psi \in \text{marker} \cap \text{prems}(\chi')\}$$

We take ord to be nat, the set of natural numbers ordered by the natural order.² Let $\mathcal{S} = (S, (F_f)_{f \in \text{fsym}}, (P_p)_{p \in \text{psym}})$ be a fixed inductive structure. We define the family of interpretations $(P_{p,n} : S^{\text{ar } p} \rightarrow S)_{p \in \text{ipsym}, n \in \text{nat}}$ recursively as follows:

- $P_{p,0} \bar{a} \Leftrightarrow \text{False}$;
- $P_{p,n+1} \bar{a} \Leftrightarrow P_{p,n} \bar{a} \vee (\exists \chi \in \text{ind}_p. \exists \xi. \bar{a} = \xi(\text{varsOf}(\text{concl}(\chi))) \wedge \mathcal{S}_n \models_{\xi} \text{prems}(\chi))$, where $\mathcal{S}_n = (S, (F_f)_{f \in \text{fsym}}, (P_{p,n})_{p \in \text{psym}})$ and $\text{varsOf}(\text{concl}(\chi))$ is the tuple \bar{x} appearing in the conclusion $p(\bar{x})$ of χ .

The above predicates are nothing but the finitary approximations of the predicates $(P_p : S^{\text{ar } p} \rightarrow S)_{p \in \text{ipsym}}$. Thanks to the inductiveness of \mathcal{S} , it is well-known that the approximations converge to their target:

Lemma 25 $\forall p \in \text{ipsym}. \forall \bar{a} \in S^{\text{ar } p}. P_p \bar{a} = (\exists n. P_{p,n} \bar{a})$.

We are now ready to define $\sigma(M, (\mathcal{S}, \xi))$. We distinguish two cases:

² This is acceptable here, since we employ finitary Horn clauses and the language is countable. Different assumptions may require larger ordinals.

- Assume $\mathcal{S} \models_{\xi} M$, and assume M has the form $p(\bar{t})$ with $p \in \text{ipsym}$. Then we have $P_p(\llbracket \bar{t} \rrbracket_{\xi}^{\mathcal{S}})$. We set $\sigma(M, (\mathcal{S}, \xi))$ to be the smallest n such that $P_{p,n}(\llbracket \bar{t} \rrbracket_{\xi}^{\mathcal{S}})$ (which exists by Lemma 25).
- Assume $\mathcal{S} \not\models_{\xi} M$. Then the return result of $\sigma(M, (\mathcal{S}, \xi))$ is irrelevant (we can put anything here).

Finally, we verify the Descent condition. Let (\mathcal{S}, ξ) and s be such that $\mathcal{S} \not\models_{\xi} s$, and let r and ss be such that $\text{eff } s \ r \ ss$. We need to provide $s' \in ss$ and (\mathcal{S}', ξ') such that $\mathcal{S}' \not\models_{\xi'} s'$ and descent $(s, (\mathcal{S}, \xi)) \ r \ (s', (\mathcal{S}', \xi'))$ holds. We will actually take \mathcal{S}' to be \mathcal{S} , so we only need to provide s' and ξ' .

We split the discussion into cases, according to the definition of δ :

- r is any rule not having the form $\text{SUBST}_{t,x}$ or p_{split} . We further distinguish some sub-cases:
 - r is AX. This is impossible, since AX is sound for all (inductive) structures, which contradicts our hypothesis $\mathcal{S} \not\models_{\xi} s$.
 - r is a single-premise rule involving no freshness side condition, i.e., is one of NEGL, NEGR, CONJR, ALLL and p_{χ} . Then we take s' to be the single premise and $\xi' = \xi$.
 - r is CONJL. Then we take $\xi' = \xi$ and s' to be one of the two premises, say, s_i , such that $\mathcal{S} \not\models_{\xi} s_i$ (which is known to exist by the soundness of CONJL).
 - r is ALLR. Then $s = (\Gamma \triangleright \Delta, \text{All } x \ \varphi)$, and we take s' to be the only premise of r , namely, $s' = \Gamma \triangleright \Delta, \varphi[y/x]$, where y is known to be fresh. Since $\mathcal{S} \not\models_{\xi} \text{All } x \ \varphi$, we obtain $a \in S$ such that $\mathcal{S} \not\models_{\xi[x \leftarrow a]} \varphi$; by the freshness of y , this also implies $\mathcal{S} \not\models_{\xi[y \leftarrow a]} \varphi[y/x]$. We take $\xi' = \xi[y \leftarrow a]$.
- r has the form $\text{SUBST}_{t,x}$. Then s has the form $\Gamma[t/x] \triangleright \Delta[t/x]$. We take $\xi' = \xi[x \leftarrow \llbracket t \rrbracket_{\xi}^{\mathcal{S}}]$ and s to be the only premise of r .
- r has the form p_{split} for some $p \in \text{ipsym}$. Then s has the form $\Gamma, p(\bar{x}) \triangleright \Delta$. Since $\mathcal{S} \models_{\xi} p(\bar{x})$, we have $P_p(\xi \bar{x})$, and let m be the least number such that $P_{p,m}(\xi \bar{x})$. By the definition of $P_{p,-}$, we have that m has the form $n + 1$ and there exist $\chi \in \text{ind}_p$ and the valuation ξ'' such that $\xi \bar{x} = \xi'' \bar{x}$ and $\mathcal{S}_n \models_{\xi''} \text{prems}(\chi)$.³ We take s' to be the premise of r corresponding to χ , namely $s' = (\Gamma[\bar{t}/\bar{x}], \text{prems}(\chi) \triangleright \Delta[\bar{t}/\bar{x}])$. Finally, we define ξ' as follows:

$$\xi' z = \begin{cases} \xi z, & \text{if } z \text{ appears free in } \Gamma \cup \Delta \\ \xi'' z, & \text{if } z \text{ appears free in } \text{prems}(\chi') \\ \text{anything,} & \text{otherwise} \end{cases}$$

In all cases, it is routine to check (when necessary with the help of the substitution Lemma 19) that $\mathcal{S}' \not\models_{\xi'} s'$ and descent $(s, (\mathcal{S}, \xi)) \ r \ (s', (\mathcal{S}', \xi'))$. When checking the latter for $r = p_{\text{split}}$, we need to notice the following:

- For all we have $M \in \text{marker} \cap \Gamma, \llbracket M[\bar{t}/\bar{x}] \rrbracket_{\xi'}^{\mathcal{S}} = \llbracket M \rrbracket_{\xi}^{\mathcal{S}}$.
- For all $q(\bar{t}) \in \text{marker} \cap \text{prems}(\chi')$, n is the smallest number such that $P_{q,n}(\llbracket \bar{t} \rrbracket_{\xi'}^{\mathcal{S}})$.

Corollary 26 Assume that, in the Genzten system of IFOL, a sequent $\Gamma \triangleright \Delta$ either has an iproof or has a cyclic proof and pointsTo fulfills Good Links. Then $\Gamma \triangleright \Delta$ is satisfied by all pairs (\mathcal{S}, ξ) : $\mathcal{S} \models_{\xi} \Gamma \triangleright \Delta$. Hence, $\mathcal{S} \models \Gamma \triangleright \Delta$.

³ The definition of $P_{p,-}$ works with the original clauses $\chi \in \text{ind}_p$, whereas here we apply it to the “copies” χ' of χ guaranteed to have their variables fresh for Γ and Δ , as stipulated in the p_{split} rule. This is unproblematic, since it is easy to verify that the definition of $P_{p,-}$ is invariant under bijective renaming of variables in the clauses χ .

Example 27 The infinite tree T of Figure 5 is an iproof, i.e., is well-formed and satisfies the goodness predicate. To see the latter, note that the only infinite path in this tree is $steps = ((s_1, r_1) \cdot (s_2, r_2) \cdot (s_3, r_3))^\omega$, where

- $s_1 = (\text{even}(x) \triangleright \text{odd}(\text{Suc}(x))), r_1 = \text{EVEN}_{\text{split}};$
- $s_2 = (\text{even}(y) \triangleright \text{odd}(\text{Suc}(\text{Suc}(\text{Suc}(y))))) , r_2 = \text{ODD}_{\text{Suc}};$
- $s_3 = (\text{even}(y) \triangleright \text{odd}(\text{Suc}(y))), r_3 = \text{SUBST}_{y,x}.$

If we let $bs = (\text{True} \cdot \text{False} \cdot \text{False})^\omega$ and $Ms = (\text{even}(x) \cdot \text{even}(y) \cdot \text{even}(y))^\omega$, we have that follow bs Ms steps. Indeed, bs and Ms “follow” steps from the beginning, since $\delta(s_1, r_1, s_2) = \{(\text{even}(x), \text{True}, \text{even}(y))\}$, $\delta(s_2, r_2, s_3) = \{(\text{even}(y), \text{False}, \text{even}(y))\}$, and $\delta(s_3, r_3, s_1) = \{(\text{even}(y), \text{False}, \text{even}(x))\}$.

Moreover, for the cyclic tree T' in Figure 4, taking pointsTo L to be T' , we obtain that T' is a well-formed cyclic proof and $\text{treeOf } T' = T$.

For accepting a cyclic or infinite proof in IFOL as a good (and hence valid) proof, any infinite path should infinitely often apply the split rule to the (persistent) instance of the same inductive predicate p .⁴ This can be seen from the fact that a triple $(_, \text{True}, _)$, meaning “decrease,” only appears in the definition of δ for the split rule. In the above example, the predicate p that ensures goodness along the tree’s sole infinite path is even.

6.2 Other Instances

Variations of Gentzen systems for FOL, as discussed in Section 4.2, can in principle be extended to IFOL, and the abstract infinite-proof soundness result would apply. Other instances include extensions of modal logic with inductive definitions. In such logics, the structures can be viewed as tuples $\mathcal{S} = ((S_w)_{w \in W}, (F_{f,w})_{f \in \text{fsym}, w \in W}, (P_{p,w})_{p \in \text{psym}, w \in W})$, where W is a set of “worlds,” perhaps endowed with additional structure (algebraic, order-theoretic, etc.). The inductiveness condition for predicates can be stated similarly to that of IFOL, but can also spread across different worlds. (The split rule can be adapted accordingly.)

Separation logic can be regarded as a variation of the above, where the structure carrier S_w is fixed to some S and the worlds are heaps, i.e., partial functions from a fixed set of locations to S . (In separation logic terminology, the worlds are the heaps, whereas the valuations ξ are the stacks.) Two such instances are described by Brotherston et al. [14], one for entailment [13] and one for termination proofs [12].

7 Formalization and Implementation

The definitions, lemmas, and theorems presented in Sections 3 and 5, pertaining to the presented abstract soundness and completeness results, have been formalized in the proof assistant Isabelle/HOL. The instantiation step of Section 4.1 is formalized for a richer version of FOL, with sorts and interpreted equality, as required by our motivating application (efficient encodings of sorts in unsorted FOL [6]). The formal development is publicly available [7,9].

The necessary codatatypes and corecursive definitions are realized using a recently introduced definitional package [5] for (co)datatypes with support for mutual and nested

⁴ It is known [14] that goodness is decidable for cyclic trees of logics for which rule application is decidable, such as IFOL.

(co)recursion [46] and mixed recursive–corecursive function definitions [10]. The tree codatatype illustrates the support for corecursion through permutative data structures (with non-free constructors) such as finite sets, a feature that is not available in any other proof assistant. The formalization is close to this article’s presentation, with a few differences originating from Isabelle’s lack of support for dependent types.

For generating code, we make the additional assumption that the effect relation is deterministic, hence corresponds to a partial function $\text{eff}' : \text{rule} \rightarrow \text{sequent} \rightarrow (\text{sequent fset})$ option, where the Isabelle datatype α option enriches a copy of α with a special value `None`.⁵ From this function, we build the relational eff as the partial function’s graph. Isabelle’s code generator [22] can then produce Haskell code for the computable part of our completeness proof—the abstract prover `mkTree`, defined corecursively in the proof of Theorem 13:

```
data Stream a = SCons a (Stream a)
newtype FSet a = FSet [a]
data Tree a = Node a (FSet (Tree a))

fmap f (FSet xs) = FSet (map f xs)

sdropWhile p (SCons a  $\sigma$ ) =
  if p a then sdropWhile p  $\sigma$  else SCons a  $\sigma$ 

mkTree eff  $\rho$  s =
  Node (s, r) (fmap (mkTree eff  $\rho'$ ) (fromJust (eff r s)))
  where SCons r  $\rho'$  = sdropWhile (\r -> not (isJust (eff r s)))  $\rho$ 
```

Finite sets are represented as lists. The functions `isJust` : α option \rightarrow bool and `fromJust` : α option \rightarrow α are the Haskell-style discriminator and selector for option. Since the Isabelle formalization is parametric over rule systems (sequent , rule , eff), the code for `mkTree` explicitly takes eff as a parameter.

Although Isabelle’s code generator was not designed with codatypes in mind, it is general enough to handle them. Internally, it reduces Isabelle specifications to higher-order rewrite systems [32] and generates functional code in Haskell, OCaml, Scala, or Standard ML. Partial correctness is guaranteed regardless of the target language’s evaluation strategy. However, for the guarantee to be non-vacuous for corecursive definitions, one needs a language with a lazy evaluation strategy, such as Haskell.

The verified contract of the program reads as follows: Given an available and persistent rule system (sequent , rule , eff), a fair rule enumeration ρ , and a sequent s representing the formula to prove, `mkTree eff ρ s` either yields a finite derivation tree of s or produces an infinite fair derivation tree whose infinite paths are all countermodel paths. These guarantees involve only partial correctness of ground term evaluation.

The generated code is a generic countermodel-producing semidecision procedure parameterized by the the proof system. Moreover, the fair rule enumeration parameter ρ can be instantiated to various choices that may perform better than the simple scheme described in Section 3.

8 Related Work

This article joins a series of pearls aimed at reclaiming mathematical concepts and results for coinductive methods, including streams [40, 44], regular expressions [41, 43], and au-

⁵ In the proof system from Example 2, eff is not deterministic due to the rule ALLR. It can be made deterministic by refining the rule with a systematic choice of the fresh variable y .

tomata [42]. Some developments pass the ultimate test of formalization, usually in Agda and Coq, the codatatype-aware proof assistants par excellence: the sieve of Eratosthenes [3], real number basics [15], and temporal logic for red–blue trees [33].

So why write yet formalized manifesto for coinduction and corecursion? First, because we finally could—with the new codatatype package, Isabelle has caught up with its rivals in this area, and has even superseded them in some respects: for example, the flexible and sound mixture of recursion and corecursion allowed in function definitions is unique to Isabelle. Second, because, although codatypes are a good match for the completeness and the infinite-proof soundness theorems (as we hope to have shown), there seems to be no proof in the literature that takes advantage of this.

There are many accounts of the completeness theorem for FOL and related logics, including Petria’s very abstract account [37] within institution-independent model theory [16]. However, the vast majority of these accounts (including Petria’s) favor the more mathematical Henkin style, which obfuscates the rich structure of proof and failure. This preference has a long history. It is positively motivated by the ability to support uncountable languages. More crucially, it is negatively motivated by the lack of rigor perceived in the alternative: “geometric” reasoning about infinite trees. Negri [34] gives a revealing account in the context of modal logic, quoting reviews that were favorable to Kripke’s completeness result [29] but critical of his informal argument based on infinite tableau trees.⁶ Kaplan [27] remarks that “although the author extracts a great deal of information from his tableau constructions, a completely rigorous development along these lines would be extremely tedious.”

A few textbooks venture in a proof-theoretic presentation of completeness, notably Gallier’s [19]. Such a treatment highlights not only the structure, but also the algorithmic content of the proofs. The price is usually a lack of rigor, in particular a gap between the definition of derivation trees and its use in the completeness argument. This lack of rigor should not be taken lightly, as it may lead to serious ambiguities or errors: In the context of a tableau completeness proof development, Hähnle [23] first performs an implicit transition from finite to possibly infinite tableaux, and then claims that tableau chain suprema exist by wrongly invoking Zorn’s lemma [23, Definition 3.16].⁷

Orthogonally, we wanted to isolate and reuse the abstract core of the argument involving potentially infinite derivation trees and countermodel paths. Except for syntactic details, the different accounts are after the same goal, and they reach it in a variety of more or less colorful, if not noisy, ways; most of them do acknowledge that their approach is similar to previous ones, but cannot refer to a given abstract result that addresses this goal. Consequently, they have to repeat a variation of the same argument. For example, Gallier’s monograph [19] repeats the argument four times, for logics of increasing complexity: propositional logic, FOL with no function symbols or equality, FOL with function symbols but no equality, and finally full FOL containing everything; Bell and Machover [1] employ a different fair tree generation strategy, to the same effect; for a world-instrumented system for modal logic, Negri [34] employs yet another strategy.

Unlike the infinite-proof soundness theorem (which represents a newer line of research), the completeness theorem has been mechanized before in proof assistants. Schlöder and Koepke, in Mizar [45], formalize a Henkin-style argument for possibly uncountable languages. Building on an early insight by Krivine [30] concerning the expressibility of the completeness proof in intuitionistic second-order logic, Ilik [25] analyzes Henkin-style ar-

⁶ And Kripke’s degree of rigor in this early article is not far from today’s state of the art in proof theory; see, e.g., Troelstra and Schwichtenberg [47].

⁷ This is the only error we found in this otherwise excellent chapter on tableaux.

guments for classical and intuitionistic logic with respect to standard and Kripke models and formalizes them in Coq (without employing codatatypes).

At least three proofs were developed using HOL-based systems. Harrison [24], in HOL Light, and Berghofer [2], in Isabelle, formalize Henkin-style arguments. Ridge and Margetson [31, 39], in Isabelle, employ proof trees constructed as graphs of nodes that carry their levels as natural numbers. This last work has the merits of analyzing the computational content of proofs in the style of Gallier [19] and discussing an OCaml implementation. Our formalization relates to this work in a similar way to which our presentation relates to Gallier’s: The newly introduced support for codatatypes and corecursion in Isabelle provides suitable abstraction mechanisms for reasoning about infinite trees, avoiding boilerplate for tree manipulation based on numeric indexing. Moreover, codatatypes are mapped naturally to Haskell types, allowing Isabelle’s code generator to produce certified Haskell code. Finally, our proof is abstract and applies to several variants of FOL and beyond.

9 Conclusion

The completeness theorem is a fundamental result about classical logic. Its proof is presented in many variants in the literature. Few of these presentations emphasize the algorithmic content, and none of them uses codatatypes. Despite the variety of approaches proposed in textbooks and formalizations, we found them lacking in rigor or readability. Gallier’s pseudo-Pascal code is inspiring, but we prefer “pseudo-Haskell,” i.e., Isabelle/HOL with codatatypes, to combine computational intuition and full mathematical rigor.

In our view, coinduction is the key to formulate an account that is both mathematically rigorous and abundant in algorithmic content. This applies to both of our case studies: classic completeness and infinite-proof soundness. The definition of the abstract prover `mkTree` is stated rigorously, is accessible to functional programmers, and replaces pages of verbose descriptions.

The advantages of machine-checked metatheory are well known from programming language research, where new results are often formalized and proof assistants are used in the classroom. This article reported on some steps we have taken to apply the same methods to formal logic and automated reasoning.

Acknowledgment. Tobias Nipkow made this work possible. Mark Summerfield and the anonymous reviewers suggested many textual improvements to earlier versions of this article. Blanchette was partially supported by the Deutsche Forschungsgemeinschaft (DFG) project Hardening the Hammer (grant NI 491/14-1). Popescu was partially supported by the DFG project Security Type Systems and Deduction (grant NI 491/13-2) as part of the program Reliably Secure Software Systems (RS³, priority program 1496). Traytel was supported by the DFG program Program and Model Analysis (PUMA, doctorate program 1480). The authors are listed alphabetically.

References

1. Bell, J.L., Machover, M.: *A Course in Mathematical Logic*. North-Holland (1977)
2. Berghofer, S.: First-order logic according to Fitting. In: G. Klein, T. Nipkow, L. Paulson (eds.) *Archive of Formal Proofs*. <http://afp.sf.net/entries/FOL-Fitting.shtml> (2007)
3. Bertot, Y.: Filters on coinductive streams, an application to Eratosthenes’ sieve. In: P. Urzyczyn (ed.) *TLCA 2005, LNCS*, vol. 3461, pp. 102–115. Springer (2005)

4. Blanchette, J.C., Böhme, S., Popescu, A., Smallbone, N.: Encoding monomorphic and polymorphic types. In: N. Piterman, S. Smolka (eds.) TACAS 2013, *LNCS*, vol. 7795, pp. 493–507. Springer (2013)
5. Blanchette, J.C., Hölzl, J., Lochbihler, A., Panny, L., Popescu, A., Traytel, D.: Truly modular (co)datatypes for Isabelle/HOL. In: G. Klein, R. Gamboa (eds.) ITP 2014, *LNCS*. Springer (2014)
6. Blanchette, J.C., Popescu, A.: Mechanizing the metatheory of Sledgehammer. In: P. Fontaine, C. Ringeisen, R.A. Schmidt (eds.) FroCoS 2013, *LNCS*, vol. 8152, pp. 245–260. Springer (2013)
7. Blanchette, J.C., Popescu, A., Traytel, D.: Abstract completeness. In: G. Klein, T. Nipkow, L. Paulson (eds.) Archive of Formal Proofs. http://afp.sf.net/entries/Abstract_Completeness.shtml (2014)
8. Blanchette, J.C., Popescu, A., Traytel, D.: Unified classical logic completeness—A coinductive pearl. In: S. Demri, D. Kapur, C. Weidenbach (eds.) IJCAR 2014, *LNCS*, vol. 8562, pp. 46–60. Springer (2014)
9. Blanchette, J.C., Popescu, A., Traytel, D.: Formal development associated with this paper. <http://people.inf.ethz.ch/traytel/compl-journal-devel.tgz> (2015)
10. Blanchette, J.C., Popescu, A., Traytel, D.: Foundational extensible corecursion: A proof assistant perspective. In: K. Fisher, J.H. Reppy (eds.) ICFP 2015, pp. 192–204. ACM (2015)
11. Brotherston, J.: Cyclic proofs for first-order logic with inductive definitions. In: B. Beckert (ed.) TABLEAUX 2005, *LNCS*, vol. 3702, pp. 78–92. Springer (2005)
12. Brotherston, J., Bornat, R., Calcagno, C.: Cyclic proofs of program termination in separation logic. In: POPL, pp. 101–112 (2008)
13. Brotherston, J., Distefano, D., Petersen, R.L.: Automated cyclic entailment proofs in separation logic. In: CADE, pp. 131–146 (2011)
14. Brotherston, J., Gorogiannis, N., Petersen, R.L.: A generic cyclic theorem prover. In: R. Jhala, A. Igarashi (eds.) APLAS, *LNCS*, vol. 7705, pp. 350–367. Springer (2012)
15. Ciaffaglione, A., Gianantonio, P.D.: A certified, corecursive implementation of exact real numbers. *Theor. Comput. Sci.* **351**(1), 39–51 (2006)
16. Diaconescu, R.: *Institution-independent Model Theory*. Birkhauser (2008)
17. Fitting, M.: *First-Order Logic and Automated Theorem Proving, Second Edition*. Graduate Texts in Computer Science. Springer (1996)
18. Francez, N.: *Fairness. Texts and Monographs in Computer Science*. Springer (1986)
19. Gallier, J.H.: *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Computer Science and Technology. Harper & Row (1986)
20. Gödel, K.: über die vollständigkeit des logikkalküls. Ph.D. thesis, Universität Wien (1929)
21. Gordon, M.J.C., Melham, T.F. (eds.): *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press (1993)
22. Haftmann, F., Nipkow, T.: Code generation via higher-order rewrite systems. In: M. Blume, N. Kobayashi, G. Vidal (eds.) FLOPS 2010, *LNCS*, vol. 6009, pp. 103–117. Springer (2010)
23. Hähnle, R.: Tableaux and related methods. In: A. Robinson, A. Voronkov (eds.) *Handbook of Automated Reasoning*, vol. I, pp. 100–178. Elsevier (2001)
24. Harrison, J.: Formalizing basic first order model theory. In: J. Grundy, M.C. Newey (eds.) TPHOLs '98, *LNCS*, vol. 1479, pp. 153–170. Springer (1998)
25. Ilik, D.: *Constructive completeness proofs and delimited control*. Ph.D. thesis, École Polytechnique (2010)
26. Jacobs, B., Rutten, J.: A tutorial on (co)algebras and (co)induction. *Bull. Eur. Assoc. Theor. Comput. Sci.* **62**, 222–259 (1997)
27. Kaplan, D.: Review of Kripke (1959) [29]. *J. Symb. Log.* **31**, 120–122 (1966)
28. Kleene, S.C.: *Mathematical Logic*. John Wiley & Sons (1967)
29. Kripke, S.: A completeness theorem in modal logic. *J. Symb. Log.* **24**(1), 1–14 (1959)
30. Krivine, J.L.: Une preuve formelle et intuitionniste du théorème de complétude de la logique classique. *Bull. Symb. Log.* **2**(4), 405–421 (1996)
31. Margetson, J., Ridge, T.: Completeness theorem. In: G. Klein, T. Nipkow, L. Paulson (eds.) Archive of Formal Proofs. <http://afp.sf.net/entries/Completeness.shtml> (2004)
32. Mayr, R., Nipkow, T.: Higher-order rewrite systems and their confluence. *Theor. Comput. Sci.* **192**(1), 3–29 (1998)
33. Nakata, K., Uustalu, T., Bezem, M.: A proof pearl with the fan theorem and bar induction: Walking through infinite trees with mixed induction and coinduction. In: H. Yang (ed.) APLAS 2011, *LNCS*, vol. 7078, pp. 353–368. Springer (2011)
34. Negri, S.: Kripke completeness revisited. In: G. Primiero, S. Rahman (eds.) *Acts of Knowledge: History, Philosophy and Logic: Essays Dedicated to Göran Sundholm*, pp. 247–282. College Publications (2009)
35. Nipkow, T., Klein, G.: *Concrete Semantics: A Proof Assistant Approach*. Springer (2014). <http://www.in.tum.de/~nipkow/Concrete-Semantics>

36. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, *LNCS*, vol. 2283. Springer (2002)
37. Petria, M.: An institutional version of gödel's completeness theorem. In: Algebra and Coalgebra in Computer Science, Second International Conference, CALCO 2007, Bergen, Norway, August 20-24, 2007, Proceedings, pp. 409–424 (2007)
38. Pfenning, F.: Review of “Jean H. Gallier: Logic for Computer Science, Harper & Row, New York 1986” [19]. *J. Symb. Log.* **54**(1), 288–289 (1989)
39. Ridge, T., Margetson, J.: A mechanically verified, sound and complete theorem prover for first order logic. In: J. Hurd, T.F. Melham (eds.) TPHOLs 2005, *LNCS*, vol. 3603, pp. 294–309. Springer (2005)
40. Roşu, G.: Equality of streams is a Π_2^0 -complete problem. In: J.H. Reppy, J.L. Lawall (eds.) ICFP '06. ACM (2006)
41. Roşu, G.: An effective algorithm for the membership problem for extended regular expressions. In: H. Seidl (ed.) FoSSaCS 2007, *LNCS*, vol. 4423, pp. 332–345. Springer (2007)
42. Rutten, J.J.M.M.: Automata and coinduction (an exercise in coalgebra). In: D. Sangiorgi, R. de Simone (eds.) CONCUR '98, *LNCS*, vol. 1466, pp. 194–218. Springer (1998)
43. Rutten, J.J.M.M.: Regular expressions revisited: A coinductive approach to streams, automata, and power series. In: R.C. Backhouse, J.N. Oliveira (eds.) MPC 2000, *LNCS*, vol. 1837, pp. 100–101. Springer (2000)
44. Rutten, J.J.M.M.: Elements of stream calculus (an extensive exercise in coinduction). *Electr. Notes Theor. Comput. Sci.* **45**, 358–423 (2001)
45. Schlöder, J.J., Koepke, P.: The Gödel completeness theorem for uncountable languages. *Formalized Mathematics* **20**(3), 199–203 (2012)
46. Traytel, D., Popescu, A., Blanchette, J.C.: Foundational, compositional (co)datatypes for higher-order logic: Category theory applied to theorem proving. In: LICS 2012, pp. 596–605. IEEE (2012)
47. Troelstra, A.S., Schwichtenberg, H.: Basic Proof Theory, Second Edition. Cambridge University Press (2000)