HAL
archives-ouvertes.fr

Soundness and Completeness Proofs by Coinductive Methods

Jasmin Christian Blanchette, Andrei Popescu, Dmitriy Traytel

▶ **To cite this version:**

Jasmin Christian Blanchette, Andrei Popescu, Dmitriy Traytel. Soundness and Completeness Proofs by Coinductive Methods. Journal of Automated Reasoning, Springer Verlag, 2017, 58 (1), pp.149 - 179. <10.1007/s10817-016-9391-3>. <hal-01643157>

HAL Id: hal-01643157
https://hal.inria.fr/hal-01643157

Submitted on 21 Nov 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Noname manuscript No.
(will be inserted by the editor)

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 We show how codatatypes 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 for various flavors of first-order logic. Soundness 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 [23] 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 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 speaking, mathematicians belong to the semantic school, whereas computer scien-

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

tists 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* [22], Pfenning, a fellow "syntactician," notes the following [41]:

> 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* [31]. 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) [29]. 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). This 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 (including cyclic) proofs for FOL with inductive definitions and related logics, studied by Brotherston et al. [12–16]. For formalizing these results, we follow rather closely the abstract constructions of Brotherston et al. [16], 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 definitions of infinite trees and paths rely on a new definitional package for codatatypes [6, 11], which automates the derivation of characteristic theorems from high-level specifications of types and functions. Through Isabelle's code generator [25], 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, 7]. 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 [9]. 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 [39] 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 [24] 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 [38, 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 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 $\mathsf{ar} : \mathsf{fsym} \uplus \mathsf{psym} \to \mathsf{nat}$ of numeric arities. Terms $t \in \mathsf{term}$ are symbolic expressions built inductively from variables by application of function symbols $f \in \mathsf{fsym}$ to tuples of arguments whose lengths respect the arities: $f(t_1, \ldots, t_{\mathsf{ar}\, f})$. Atoms $a \in \mathsf{atom}$ are symbolic expressions of the form $p(t_1, \ldots, t_{\mathsf{ar}\, p})$, where $p \in \mathsf{psym}$ and $t_1, \ldots, t_{\mathsf{ar}\, p} \in \mathsf{term}$.

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

$$\begin{array}{rcl} \textbf{datatype}\ \mathsf{fmla} & = & \mathsf{Atm}\ \mathsf{atom} \\ & | & \mathsf{Neg}\ \mathsf{fmla} \\ & | & \mathsf{Conj}\ \mathsf{fmla}\ \mathsf{fmla} \\ & | & \mathsf{All}\ \mathsf{var}\ \mathsf{fmla} \end{array}$$

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

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 $\mathsf{Atm}\ a$, thus pretending that atoms are included in formulas.

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

$$[\![x]\!]^{\mathscr{S}}_{\xi} = \xi\ x \qquad\qquad [\![f(t_1, \ldots, t_n)]\!]^{\mathscr{S}}_{\xi} = F_f\ \big([\![t_1]\!]^{\mathscr{S}}_{\xi}, \ldots, [\![t_n]\!]^{\mathscr{S}}_{\xi}\big)$$

For atoms:

$$\mathscr{S} \models_{\xi} p(t_1, \ldots, t_n) \quad \text{iff} \quad P_p\ \big([\![t_1]\!]^{\mathscr{S}}_{\xi}, \ldots, [\![t_n]\!]^{\mathscr{S}}_{\xi}\big)$$

For formulas (by structural recursion):

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

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

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

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

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

$$\frac{}{\Gamma,\ \mathsf{Atm}\ a \rhd \Delta,\ \mathsf{Atm}\ a}\ \textsc{Ax} \qquad \frac{\Gamma \rhd \Delta, \varphi}{\Gamma,\ \mathsf{Neg}\ \varphi \rhd \Delta}\ \textsc{NegL} \qquad \frac{\Gamma, \varphi \rhd \Delta}{\Gamma \rhd \Delta,\ \mathsf{Neg}\ \varphi}\ \textsc{NegR}$$

$$\frac{\Gamma, \varphi, \psi \rhd \Delta}{\Gamma, \mathsf{Conj}\,\varphi\,\psi \rhd \Delta}\ \textsc{ConjL} \qquad \frac{\Gamma \rhd \Delta, \varphi \quad \Gamma \rhd \Delta, \psi}{\Gamma \rhd \Delta, \mathsf{Conj}\,\varphi\,\psi}\ \textsc{ConjR}$$

$$\frac{\Gamma, \mathsf{All}\,x\,\varphi, \varphi[t/x] \rhd \Delta}{\Gamma, \mathsf{All}\,x\,\varphi \rhd \Delta}\ \textsc{AllL} \qquad \frac{\Gamma \rhd \Delta, \varphi[y/x]}{\Gamma \rhd \Delta, \mathsf{All}\,x\,\varphi}\ \textsc{AllR}\ (y \text{ fresh})$$

Since here sequents are sets, we do not need the structural rules (weakening, contraction, and exchange).

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., $\vdash \Gamma \rhd \Delta$).

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

$$\vdash \Gamma \rhd \Delta \Rightarrow (\forall \mathscr{S}.\, \mathscr{S} \models \Gamma \rhd \Delta)$$

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

$$(\forall \mathscr{S}.\, \mathscr{S} \models \Gamma \rhd \Delta) \Rightarrow \vdash \Gamma \rhd \Delta$$

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 ($\mathrm{FOL}_{\mathsf{ind}}$). In addition to the first-order language given by var, fsym, psym, and $\mathsf{ar} : \mathsf{fsym} \uplus \mathsf{psym} \to \mathsf{nat}$, we fix a subset of predicate symbols $\mathsf{ipsym} \subseteq \mathsf{psym}$ which are called *inductive*. Moreover, we fix, for each $p \in \mathsf{ipsym}$, a set ind_p of Horn clauses specifying p, so that each element of ind_p is of the form $\mathsf{Imp}\,(\mathsf{Conj}\,\psi_1\ \ldots\ \psi_m\,\varphi_1\ \ldots\ \varphi_n)\,\varphi$, where

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

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 [12].

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

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

- $\mathsf{ind}_{\mathsf{even}}$ consists of the following clauses:

[1] Given formulas $\psi_1, \ldots, \psi_k$, we let $\mathsf{Conj}\,\psi_1\ \ldots\ \psi_k$ denote $\mathsf{Conj}\,\psi_1\,(\mathsf{Conj}\,\psi_2\,(\ldots\psi_n)\ldots)$. In particular, when $k = 0$ it denotes the "true" formula $\top$, defined in a standard way, e.g., as $\mathsf{Imp}\,a\,a$ for some atom a.

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

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

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

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

Obviously, there are 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:

$$\frac{}{\Gamma\ \rhd \Delta, \mathsf{even}(0)}\ \textsc{Even}_0 \qquad \frac{\Gamma \rhd \Delta, \mathsf{even}(x)}{\Gamma \rhd \Delta, \mathsf{even}(\mathsf{Suc}(\mathsf{Suc}(x)))}\ \textsc{Even}_{\mathsf{Suc}}$$

$$\frac{\Gamma[0/x] \rhd \Delta[0/x] \quad \Gamma[\mathsf{Suc}(\mathsf{Suc}(y))/x], \mathsf{even}(y) \rhd \Delta[\mathsf{Suc}(\mathsf{Suc}(y))/x]}{\Gamma, \mathsf{even}(x) \rhd \Delta}\ \textsc{Even}_{\mathrm{split}}\ (y \text{ fresh})$$

The *direct* rules $\textsc{Even}_0$ and $\textsc{Even}_{\mathsf{Suc}}$ (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 $\textsc{Even}_{\mathrm{split}}$ is admissible by the least fixpoint assumption—the rule expresses that, if $\mathsf{even}(x)$ is known, then it must have been obtained by an application of one of the inductive clauses for even. Similar rules hold for odd:

$$\frac{}{\Gamma \rhd \Delta, \mathsf{odd}(\mathsf{Suc}(0))}\ \textsc{Odd}_0 \qquad \frac{\Gamma \rhd \Delta, \mathsf{odd}(x)}{\Gamma \rhd \Delta, \mathsf{odd}(\mathsf{Suc}(\mathsf{Suc}(x)))}\ \textsc{Odd}_{\mathsf{Suc}}$$

$$\frac{\Gamma[\mathsf{Suc}(0)/x] \rhd \Delta[\mathsf{Suc}(0)/x] \quad \Gamma[\mathsf{Suc}(\mathsf{Suc}(y))/x], \mathsf{odd}(y) \rhd \Delta[\mathsf{Suc}(\mathsf{Suc}(y))/x]}{\Gamma, \mathsf{odd}(x) \rhd \Delta}\ \textsc{Odd}_{\mathrm{split}}\ (y \text{ fresh})$$

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

$$\frac{(\Gamma \rhd \Delta, \varphi)_{\varphi\, \in \mathsf{prems}(\chi)}}{\Gamma \rhd \Delta, \mathsf{concl}(\chi)}\ p_\chi \qquad \frac{(\Gamma[\bar{t}/\bar{x}],\ \mathsf{prems}(\chi') \rhd \Delta[\bar{t}/\bar{x}])_{\chi \in \mathsf{ind}_p}}{\Gamma,\ p(\bar{x}) \rhd \Delta}\ p_{\mathrm{split}}$$

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

- χ' is a variant of χ where all the free variables are fresh for $\Gamma \cup \Delta$;
- $p(\bar{t})$ is $\mathsf{concl}(\chi')$—with $\bar{t}$ thus being a tuple of terms $(t_1, \ldots, t_{\mathsf{ar}\,p})$;
- $\bar{x}$ is a tuple of distinct variables $(x_1, \ldots, x_{\mathsf{ar}\,p})$.

The Gentzen system for $\mathrm{FOL}_{\mathsf{ind}}$ consists of the FOL rules from Section 2.1 extended with the above rules p_χ, p_{split} and the following substitution rule:

$$\frac{\Gamma \rhd \Delta}{\Gamma[t/x] \rhd \Delta[t/x]}\ \textsc{Subst}_{t,x}$$

This rule is designed to complement the direct rule p_χ, for the purpose of applying the Horn clauses of p to particular instances.

The inversion rule p_{split} is not as powerful as it could be, due to the restriction that the eigenformula, $p(\overline{x})$, must be a predicate symbol applied to variables. In a $\text{FOL}_{\mathsf{ind}}$ variant with equality, this could be strengthened to speak about arbitrary terms instead of variables [13, Section 4.1].

Beyond the additional admissible rules, a crucial insight of Brotherston [12] is that, for inductive structures, a certain type of proof circularity is permissible. It allows the proof trees to have as leaf nodes not only axioms, but also backward links to other sequents in the tree. This insight is useful for automating induction [12]. Brotherston et al. [16] 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

Before studying applications, we first 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 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 and 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 introduced 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 $\mathsf{sequent}$ 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 $\mathsf{structure}$ and a satisfaction relation

$$\models : \mathsf{structure} \to \mathsf{sequent} \to \mathsf{bool}$$

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: $\forall S \in \mathsf{structure}.\ S \models s$.

3.2 Rule Systems

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

$$\mathsf{eff} : \mathsf{rule} \to \mathsf{sequent} \to \mathsf{sequent\ fset} \to \mathsf{bool}$$

where α fset denotes the set of finite subsets of α. The reading of 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 $\mathscr{R} = (\mathsf{sequent}, \mathsf{rule}, \mathsf{eff})$ 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 $\textsc{Ax}_a$ for each atom a; rules $\textsc{NegL}_\varphi$ and $\textsc{NegR}_\varphi$ for each formula φ; rules $\textsc{ConjL}_{\varphi,\psi}$ and $\textsc{ConjR}_{\varphi,\psi}$ for each pair of formulas φ and ψ; a rule $\textsc{AllL}_{x,\varphi,t}$ for each variable x, formula φ, and term t; and a rule $\textsc{AllR}_{x,\varphi}$ 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:

$\mathsf{eff}\ \textsc{Ax}_a\ (\Gamma, \mathsf{Atm}\ a \rhd \Delta, \mathsf{Atm}\ a)\ \emptyset$
$\mathsf{eff}\ \textsc{NegR}_\varphi\ (\Gamma \rhd \Delta, \mathsf{Neg}\ \varphi)\ \{\Gamma, \varphi \rhd \Delta\}$
$\mathsf{eff}\ \textsc{NegL}_\varphi\ (\Gamma, \mathsf{Neg}\ \varphi \rhd \Delta)\ \{\Gamma \rhd \Delta, \varphi\}$
$\mathsf{eff}\ \textsc{ConjL}_{\varphi,\psi}\ (\Gamma, \mathsf{Conj}\ \varphi\ \psi \rhd \Delta)\ \{\Gamma, \varphi, \psi \rhd \Delta\}$
$\mathsf{eff}\ \textsc{ConjR}_{\varphi,\psi}\ (\Gamma \rhd \Delta, \mathsf{Conj}\ \varphi\ \psi)\ \{\Gamma \rhd \Delta, \varphi;\ \Gamma \rhd \Delta, \psi\}$
$\mathsf{eff}\ \textsc{AllL}_{x,\varphi,t}\ (\Gamma, \mathsf{All}\ x\ \varphi \rhd \Delta)\ \{\Gamma, \mathsf{All}\ x\ \varphi, \varphi[t/x] \rhd \Delta\}$
$\mathsf{eff}\ \textsc{AllR}_{x,\varphi}\ (\Gamma \rhd \Delta, \mathsf{All}\ x\ \varphi)\ \{\Gamma \rhd \Delta, \varphi[y/x]\}$ where y is fresh for Γ and $\mathsf{All}\ x\ \varphi$

3.3 Derivation Trees

Finitely branching, possibly infinite trees with nodes labeled by elements in a set α are represented by the following codatatype:

$$\textbf{codatatype}\ \alpha\ \mathsf{tree} = \mathsf{Node}\ (\mathsf{lab}\colon \alpha)\ (\mathsf{sub}\colon (\alpha\ \mathsf{tree})\ \mathsf{fset})$$

This definition introduces a constructor $\mathsf{Node} : \alpha \to (\alpha\ \mathsf{tree})\ \mathsf{fset} \to \alpha\ \mathsf{tree}$ and two selectors $\mathsf{lab} : \alpha\ \mathsf{tree} \to \alpha$ and $\mathsf{sub} : \alpha\ \mathsf{tree} \to (\alpha\ \mathsf{tree})\ \mathsf{fset}$. Trees have the form $\mathsf{Node}\ a\ Ts$, 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 iterated an infinite number of times to create infinitely deep objects.

Remark 3 Inductive datatypes can also store infinite objects, provided the constructors are infinitely branching. However, the infiniteness of these objects manifests itself in breadth only. The following examples illustrate this distinction:

$$\begin{aligned} \textbf{datatype}\ \alpha\ \mathsf{tree}_1 &= \mathsf{Leaf}\ \alpha \mid \mathsf{Node}\ (\mathsf{nat} \to \alpha\ \mathsf{tree}_1) \\ \textbf{codatatype}\ \alpha\ \mathsf{tree}_2 &= \mathsf{Leaf}\ \alpha \mid \mathsf{Node}\ (\mathsf{nat} \to \alpha\ \mathsf{tree}_2) \end{aligned}$$

Due to infinite branching over the type nat, the elements of the datatype $\alpha\ \mathsf{tree}_1$ can be infinite, and indeed of unbounded depth—however, they will never have a genuinely infinite depth, as witnessed by an infinite path. By contrast, the elements of the codatatype $\alpha\ \mathsf{tree}_2$ can have infinite paths (but need not to).

A *step* combines the current sequent and the rule to be applied. Derivation trees are defined as trees labeled by steps:

$$\mathsf{step} = \mathsf{sequent} \times \mathsf{rule} \qquad\qquad \mathsf{dtree} = \mathsf{step\ tree}$$

We think of the root's label (s, r) as representing the proved goal s and the first (backwards) applied rule r. The well-formed derivation trees are captured by the predicate $\mathsf{wf} : \mathsf{dtree} \to \mathsf{bool}$ defined by the coinductive rule

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

The term $\mathsf{image}\ f\ A$ denotes the image of set A through function f, and fst is the left projection operator (i.e., $\mathsf{fst}\ (x, y) = x$). The first assumption requires that the rule r from the root be applied to obtain the subtrees' labels. The second assumption requires that wellformedness holds for the immediate subtrees. The coinductive interpretation of the definition ensures that the iteration of this rule can cover infinite trees; this would not be the case with an inductive interpretation.

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

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

which is also the greatest solution to the implication

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

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

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

3.4 Proofs

The finite derivation trees can be carved out of the codatatype dtree using the predicate finite defined inductively (i.e., as a least fixpoint) by the rule

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

Indeed, the inductive interpretation is the right one for defining finite, since we want to enforce the well-founded iteration of the rule. (By contrast, a coinductive interpretation would classify all trees as "finite," which is certainly not desirable.)

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

$$\mathsf{proof}\ T\ s \iff \mathsf{finite}\ T \wedge \mathsf{wf}\ T \wedge \mathsf{fst}\ (\mathsf{lab}\ T) = s$$

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

$$\dfrac{\dfrac{\dfrac{}{\forall x.\, p(x),\ p(y) \rhd p(y)}\ \textsc{Ax}_{p(y)}}{\forall x.\, p(x) \rhd p(y)}\ \textsc{AllL}_{x,p(x),y} \qquad \dfrac{\dfrac{\dfrac{}{\forall x.\, p(x),\ p(z) \rhd p(z)}\ \textsc{Ax}_{p(z)}}{\forall x.\, p(x) \rhd p(z)}\ \textsc{AllL}_{x,p(x),z}}{\forall x.\, p(x) \rhd p(y) \wedge p(z)}\ \textsc{ConjR}_{p(y),\,p(z)}$$

Fig. 1 A proof

$$\dfrac{\dfrac{\dfrac{}{\forall x.\, p(x),\ p(y) \rhd p(y)}\ \textsc{Ax}_{p(y)}}{\forall x.\, p(x) \rhd p(y)}\ \textsc{AllL}_{x,p(x),y} \qquad \dfrac{\dfrac{\dfrac{\vdots}{\forall x.\, p(x),\ p(y) \rhd p(z)}\ \textsc{AllL}_{x,p(x),y}}{\forall x.\, p(x),\ p(y) \rhd p(z)}\ \textsc{AllL}_{x,p(x),y}}{\forall x.\, p(x) \rhd p(z)}\ \textsc{AllL}_{x,p(x),y}}{\forall x.\, p(x) \rhd p(y) \wedge p(z)}\ \textsc{ConjR}_{p(y),\,p(z)}$$

Fig. 2 A failed proof attempt

Example 4 Given the instantiation of Example 2, Figure 1 shows a finite derivation tree for the sequent $\mathsf{All}\ x\ (p(x)) \rhd \mathsf{Conj}\ (p(y))\ (p(z))$ 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.

$$\textit{Local Soundness}\colon\ \forall r\ s\ ss.\ \mathsf{eff}\ r\ s\ ss \wedge (\forall s' \in ss.\ \models s') \Rightarrow \models s$$

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

Theorem 5 Assume the rule system fulfills Local Soundness.Then every sequent that has a proof is satisfied by all structures. Formally:

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

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:

$$\textbf{codatatype}\ \alpha\ \mathsf{stream} = \mathsf{SCons}\ (\mathsf{shead}\colon \alpha)\ (\mathsf{stail}\colon \alpha\ \mathsf{stream})$$

The coinductive predicate $\mathsf{ipath} : \mathsf{dtree} \to \mathsf{step\ stream} \to \mathsf{bool}$ determines whether a stream of steps is an infinite path in a tree:

$$\dfrac{T \in Ts \qquad \mathsf{ipath}\ T\ \sigma}{\mathsf{ipath}\ (\mathsf{Node}\ (s, r)\ Ts)\ (\mathsf{SCons}\ (s, r)\ \sigma)}\ \textsc{IPath}$$

Our notion of a tree being finite can be shown to coincide with the more standard notion of having a finite number of nodes. Hence the following result is a variant of König's lemma. Its proof allows us to show a first simple corecursive definition.

Lemma 6 If the tree T is infinite (i.e., non-finite), there exists an infinite path σ in T.

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

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

Thus, $\mathsf{ipath}\ (\mathsf{p}\ T)\ T$ by straightforward coinduction on the definition of ipath. □

Remark 7 Essentially the same proof would hold if we allowed our trees to be infinitely branching, by replacing finite sets with countable sets in the definition of the tree codatatype. This may seem counterintuitive if we think of the standard formulation of König's lemma, but finite would no longer mean having a finite number of nodes—it would simply mean "well founded."

The following extension of König's lemma applies to well-formed derivation trees, allowing one to construct an infinite path satisfying a unary invariant on its steps and a binary invariant on pairs of neighbored steps. The latter additionally takes the transition rule between the neighbored steps into account. The lemma's statement involves the "always" predicate defined coinductively for streams over any set β, namely, $\mathsf{alw} : (\beta\ \mathsf{stream} \to \mathsf{bool}) \to \beta\ \mathsf{stream} \to \mathsf{bool}$, where $\mathsf{alw}\ P\ xs$ states that the predicate P holds for all suffixes of xs:

$$\frac{P\ xs \qquad \mathsf{alw}\ P\ (\mathsf{stail}\ xs)}{\mathsf{alw}\ P\ xs}\ \textsc{Alw}$$

Lemma 8 Fix the set α and the predicates $I : \mathsf{sequent} \times \alpha \to \mathsf{bool}$ and $P : \mathsf{sequent} \times \alpha \to \mathsf{rule} \to \mathsf{sequent} \times \alpha \to \mathsf{bool}$. Assume that

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

If the tree T is well formed, there exists a stream $\rho \in (\mathsf{step} \times \alpha)\ \mathsf{stream}$ such that its first projection is an infinite path in T (formally, $\mathsf{ipath}\ T\ (\mathsf{smap}\ \mathsf{fst}\ \rho)$) and ρ satisfies the predicate

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

In the above lemma, 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 6, but it requires a slightly more complex function f, namely $f : B \to B$, where $B = \{(T, a) \in \mathsf{dtree} \times \alpha \mid I\ (\mathsf{fst}\ (\mathsf{lab}\ T), a)\}$, such that $P\ (\mathsf{fst}\ (\mathsf{lab}\ T),\ a)\ r\ (\mathsf{fst}\ (\mathsf{lab}\ T'),\ a')$ 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 $\mathsf{epath} : \mathsf{step\ stream} \to \mathsf{bool}$, which requires that every element in the given stream be obtained by applying an existing rule and choosing one of the resulting sequents:

$$\frac{\mathsf{eff}\ r\ s\ ss \qquad s' \in ss \qquad \mathsf{epath}\ (\mathsf{SCons}\ (s', r')\ \sigma)}{\mathsf{epath}\ (\mathsf{SCons}\ (s, r)\ (\mathsf{SCons}\ (s', r')\ \sigma))}\ \textsc{EPath}$$

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

Lemma 9 For any stream σ and tree T, if $\mathsf{wf}\ T$ and $\mathsf{ipath}\ \sigma\ T$, then $\mathsf{epath}\ \sigma$.

Example 10 The stream

$$\forall x.\ p(x) \rhd p(y) \wedge p(z),\ \textsc{ConjR}_{p(y),\,p(z)}) \cdot (\forall x.\ p(x) \rhd p(z),\ \textsc{AllL}_{x,p(x),y}) \cdot (\forall x.\ p(x), p(y) \rhd p(z),\ \textsc{AllL}_{x,p(x),y})^{\infty}$$

where $(s, r) \cdot \sigma = \mathsf{SCons}\ (s, r)\ \sigma$ and $(s, r)^{\infty} = (s, r) \cdot (s, r) \cdot \ldots$, 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 at 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 by interpreting all statements along the path on the left of the sequents as true, and all statements on the right as false.

Formally, a rule r is *enabled* in a sequent s if it has an effect (i.e., $\mathsf{eff}\ r\ s\ ss$ for some ss). This is written $\mathsf{enabled}\ r\ s$. For any rule r and stream σ:

- $\mathsf{taken}_r\ \sigma$ states that r is taken at the start of the stream (i.e., $\mathsf{shead}\ \sigma = (s, r)$ for some s);
- $\mathsf{enabledAt}_r\ \sigma$ states that r is enabled at the beginning of the stream (i.e., if $\mathsf{shead}\ \sigma = (s, r')$, then $\mathsf{enabled}\ r\ s$);

Recall that, given any set α, predicate $P : \alpha\ \mathsf{stream} \to \mathsf{bool}$, and stream $xs \in \alpha\ \mathsf{stream}$, the predicate $\mathsf{alw}\ P\ xs$ ("always P") states that P is true for all suffixes of xs. Dually, we take $\mathsf{ev}\ P\ \sigma$ ("eventually P") to mean that P is true for some suffix of xs.

A stream σ is *saturated* if, at each point, any enabled rule is taken at a later point:

$$\mathsf{saturated}\ \sigma \iff (\forall r \in \mathsf{rule}.\ \mathsf{alw}\ (\lambda\sigma'.\ \mathsf{enabledAt}_r\ \sigma' \Rightarrow \mathsf{ev}\ \mathsf{taken}_r\ \sigma')\ \sigma)$$

A *countermodel path* for a sequent s is a saturated escape path σ starting at s:

$$\mathsf{countermodelPath}\ s\ \sigma \iff \mathsf{epath}\ \sigma \wedge \mathsf{saturated}\ \sigma \wedge \mathsf{fst}\ (\mathsf{shead}\ \sigma) = s$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\forall x.\, p(x),\, p(t_1),\, p(t_2),\, p(t_3) \rhd q(y)}\,\textsc{All}\mathrm{L}_{x,p(x),t_4}}{\forall x.\, p(x),\, p(t_1),\, p(t_2) \rhd q(y)}\,\textsc{All}\mathrm{L}_{x,p(x),t_3}}{\forall x.\, p(x),\, p(t_1) \rhd q(y)}\,\textsc{All}\mathrm{L}_{x,p(x),t_2}}{\forall x.\, p(x) \rhd q(y)}\,\textsc{All}\mathrm{L}_{x,p(x),t_1}$$

Fig. 3 A derivation tree with a countermodel path

Example 11 The escape path presented in Example 10 is not saturated, since the rule $\textsc{All}\mathrm{L}_{x,p(x),z}$ is enabled starting from the first position but never taken.

Example 12 The escape path in the tree of Figure 3 is a countermodel path for $\forall x.\, p(x) \rhd q(y)$, assuming that each possible term occurs infinitely often in the sequence $t_1, t_2, \ldots$. The enabled rules along the escape path are all of the form $\textsc{All}\mathrm{L}_{x,p(x),\_}$, and they are all always eventually taken.

3.9 Completeness

For the proof of completeness, we assume that the set of rules fulfills two properties:

- *Availability*: For each sequent, at least one rule is enabled (i.e., $\forall s.\, \exists r.\, \mathsf{enabled}\ r\ s$).
- *Persistence*: For each sequent, if a rule is enabled but not taken, it remains enabled (i.e., $\forall s\, r\, r'\, s'\, ss.\, \mathsf{enabled}\ r'\ s \wedge r' \neq r \wedge \mathsf{eff}\ r\ s\ ss \wedge s' \in \mathsf{set}\ ss \Rightarrow \mathsf{enabled}\ r'\ s'$).

(We will later remove the first condition with Theorem 18.) The above conditions are local properties of the effects of rules, not global properties of the proof system. This makes them easy to verify for particular systems.

Remark 13 The saturation condition on streams of steps from Section 3.8 is stronger than the standard properties of fairness and justice [21]. Fairness would require the rules to be continuously enabled to guarantee that they are eventually taken. 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, a rule enabled at some point and never taken will be continuously (in particular, infinitely often) enabled; hence the notions of fairness, justice, and saturation all coincide.

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

$$\textit{Herbrandness}:\ \forall s\, \sigma.\, \mathsf{countermodelPath}\ s\ \sigma \Rightarrow \mathsf{herbrand}\ s\ \sigma \not\models s$$

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

Lemma 14 Assume the rule system fulfills Availability and Persistence. Then every sequent admits a proof or a countermodel path. Formally:

$$\forall s.\ (\exists T.\, \mathsf{proof}\ T\ s) \vee (\exists \sigma.\, \mathsf{countermodelPath}\ s\ \sigma)$$

Proof The proof uses the following combinators:

- stake : α stream $\to$ nat $\to \alpha$ list maps ρ and n to the list of the first n elements of ρ;
- smap : $(\alpha \to \beta) \to \alpha$ stream $\to \beta$ stream maps a function to every element of the stream;
- nats : nat stream denotes the stream of natural numbers: $0 \cdot 1 \cdot 2 \cdot 3 \cdot \ldots$;
- flat : $(\alpha$ list$)$ stream $\to \alpha$ stream maps a stream of finite nonempty lists to the stream obtained by concatenating those lists;
- sdropWhile : $(\alpha \to$ bool$) \to \alpha$ stream $\to \alpha$ 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).

We start by constructing a fair stream of rules fenum—i.e., every rule occurs infinitely often in fenum. Let enum be a stream whose elements cover the entire set rule, which is required to be countable. Take fenum = flat (smap (stake enum) (stail nats)). Thus, if enum $= r_1 \cdot r_2 \cdot r_3 \cdot \ldots$, then fenum $= r_1 \cdot r_1 \cdot r_2 \cdot r_1 \cdot r_2 \cdot r_3 \cdot \ldots$.

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 $\rho \in$ fair and $s \in$ sequent, sdropWhile $(\lambda r.\ \neg$ enabled $r\ s)\ \rho$ has the form SCons $r\ \rho'$, making r the first enabled rule in ρ. Such a rule exists because, by Availability, at least one rule is enabled at s and, by fairness, all the rules occur in ρ. Since enabled $r\ s$, we can pick a set of sequents ss such that eff $r\ s\ ss$. We define mkTree : fair $\to$ sequent $\to$ dtree corecursively as

$$\mathsf{mkTree}\ \rho\ s = \mathsf{Node}\ (s, r)\ (\mathsf{image}\ (\mathsf{mkTree}\ \rho')\ ss)$$

We prove that, for all $\rho \in$ fair and s, the derivation tree 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 ipath (mkTree $\rho\ s$) σ, then r appears along σ (i.e., there exists a sequent s' such that (s', r) is in σ). This follows by induction on the position of r in ρ, pos $r\ \rho$—formally, the length of the shortest list ρ_0 such that $\rho = \rho_0$ @ SCons r _, where @ denotes concatenation.

Let r' be the first rule from ρ enabled at sequent s. If $r = r'$, then mkTree $\rho\ s$ has label (s, r) already. Otherwise, ρ has the form ρ_1 @ $[r']$ @ ρ', with r not in ρ_1, hence pos $r\ \rho' <$ pos $r\ \rho$. From the definitions of ipath and mkTree, it follows that ipath (mkTree $\rho'\ s'$) (stail σ) holds for some $s' \in ss$ such that eff $r\ s'\ ss$. By the induction hypothesis, r appears along stail σ, hence along σ as desired. In particular, $T_0 =$ 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 6 (König) it has an infinite path. This path is necessarily saturated; by Lemma 9, it is the desired countermodel path. □

Lemma 14 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 15 Fix a rule system and assume that it fulfills Availability, Persistence, and Herbrandness. Then every sequent that is satisfied by all structures admits a proof:

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

Proof Given s such that $\models s$, assume by absurdity that there exists no T such that $\mathsf{proof}\ T\ s$. By Lemma 14, we obtain σ such that $\mathsf{countermodelPath}\ s\ \sigma$. Then, by Herbrandness, we have $\mathsf{herbrand}\ s\ \sigma \not\models s$, which contradicts $\models s$. □

Remark 16 If we are not interested in witnessing the proof attempt closely, Lemma 14 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 $\mathsf{eff}\ r\ s\ ss$, 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 $\mathsf{pick}\ r\ s\ ss$ denote such an s'. We proceed directly to the construction of a saturated escape path as a corecursive predicate $\mathsf{mkPath} : \mathsf{fair} \to \{s \in \mathsf{sequent}.\ s \text{ has no proof}\} \to \mathsf{step\ stream}$ following the same idea as for the previous tree construction (function mkTree):

$$\mathsf{mkPath}\ \rho\ s = \mathsf{SCons}\ (s, r)\ (\mathsf{mkPath}\ \rho'\ (\mathsf{pick}\ r\ s\ ss))$$

where again $\mathsf{SCons}\ r\ \rho' = \mathsf{sdropWhile}\ (\lambda r.\ \neg\ \mathsf{enabled}\ r\ s)\ \rho$ and ss is such that $\mathsf{eff}\ r\ s\ ss$. Fairness of $\mathsf{mkPath}\ \rho\ s$ follows by a similar argument as before for fairness of the tree.

3.10 Omitting the Availability Assumption

The abstract completeness result (Lemma 14) 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 17 A rule system $\mathscr{R} = (\mathsf{sequent}, \mathsf{rule}, \mathsf{eff})$ that fulfills Persistence can be transformed into the rule system $\mathscr{R}_{\mathrm{idle}} = (\mathsf{sequent}, \mathsf{rule}_{\mathrm{idle}}, \mathsf{eff}_{\mathrm{idle}})$ that fulfills both Persistence and Availability, with $\mathsf{rule}_{\mathrm{idle}} = \mathsf{rule} \cup \{\textsc{Idle}\}$ and $\mathsf{eff}_{\mathrm{idle}}$ behaving like eff on rule and $\mathsf{eff}_{\mathrm{idle}}\ \textsc{Idle}\ s\ ss \Leftrightarrow ss = \{s\}$.

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. □

Now we can rephrase Theorem 15 to assume Persistence and a slight variation of the Herbrandness condition. All the concepts refer as before to the rule system $\mathscr{R}$, except for $\mathsf{countermodelPath}_{\mathrm{idle}}$, which refers to $\mathscr{R}_{\mathrm{idle}}$:

$$\textit{Herbrandness}_{\mathrm{idle}}\colon\ \forall s\ \sigma.\ \mathsf{countermodelPath}_{\mathrm{idle}}\ s\ \sigma \Rightarrow \mathsf{herbrand}\ s\ \sigma \not\models s$$

Theorem 18 Let $\mathscr{R}$ be a rule system that fulfills Persistence and $\text{Herbrandness}_{\mathrm{idle}}$. Then every sequent that is satisfied by all structures admits a proof in $\mathscr{R}$.

Proof We apply Lemma 14 to the system $\mathscr{R}_{\mathrm{idle}}$ to obtain that every sequent admits either a proof or a countermodel path, both in this system. Since $\mathscr{R}_{\mathrm{idle}}$ is an extension of $\mathscr{R}$ with a single rule, any proof in $\mathscr{R}$ corresponds to a proof in $\mathscr{R}_{\mathrm{idle}}$. Conversely, any proof in $\mathscr{R}_{\mathrm{idle}}$ can be transformed into a proof in $\mathscr{R}$ by omitting all applications of Idle. We thus proved that every sequent admits a proof in $\mathscr{R}$ or a countermodel path over $\mathscr{R}_{\mathrm{idle}}$. We can then apply $\text{Herbrandness}_{\mathrm{idle}}$, just as we did with Herbrandness for Theorem 18. □

Remark 19 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 recasts 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 $\textsc{Ax}_a$ (which affects only the atom a) leaves any potential enabledness of $\textsc{All}\mathrm{L}_{x,\varphi,t}$ (which affects only formulas with All at the top) unchanged, and vice versa; moreover, $\textsc{Ax}_a$ does not overlap with $\textsc{Ax}_b$ for $a \neq b$. A minor subtlety concerns $\textsc{All}\mathrm{R}_{x,\varphi}$, which requires the existence of a fresh y in order to be enabled. Persistence holds 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 example, the sequent $p(x) \rhd q(x)$ has no enabled rule. Hence, we need Theorem 18 and its IDLE rule.

To infer the standard completeness theorem from Theorem 18, it suffices to define a suitable function $\mathsf{herbrand}$. Let σ be a countermodel path for $\Gamma \rhd \Delta$ (i.e., a saturated escape path starting at $\Gamma \rhd \Delta$). Let $\widetilde{\Gamma}$ be the union of the left-hand sides of sequents occurring in σ, and let $\widetilde{\Delta}$ be the union of the corresponding right-hand sides. Clearly, $\Gamma \subseteq \widetilde{\Gamma}$ and $\Delta \subseteq \widetilde{\Delta}$. We define $\mathsf{herbrand}\ (\Gamma \rhd \Delta)\ \sigma$ to be $\mathscr{S} = (S, F, P)$, where

- S is the set of terms, term;
- for each n-ary f and p and each $t_1, \ldots, t_n \in S$:
  - $F_f\,(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$;
  - $P_p\,(t_1, \ldots, t_n) \Leftrightarrow p(t_1, \ldots, t_n) \in \widetilde{\Gamma}$.

In what follows, we employ the substitution lemma, which relates the notions of satisfaction and substitution:

Lemma 20 $\mathscr{S} \models_\xi \varphi[t/x]$ iff $\mathscr{S} \models_{\xi[x \leftarrow [\![t]\!]^{\mathscr{S}}_\xi]} \varphi$.

Lemma 21 The structure $\mathsf{herbrand}\ (\Gamma \rhd \Delta)\ \sigma$ is a countermodel for $\Gamma \rhd \Delta$, meaning that there exists a valuation $\xi : \mathsf{var} \to S$ such that $\mathscr{S} \not\models_\xi \Gamma \rhd \Delta$.

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

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

A pair $(\widetilde{\Gamma}, \widetilde{\Delta})$ fulfilling these properties is sometimes called a Hintikka set [1, 22]. These properties follow from the saturation of σ with respect to the corresponding rules. The proofs are routine. For example:

1. If $\mathsf{Atm}\, a \in \widetilde{\Gamma} \cap \widetilde{\Delta}$, 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 $\mathsf{All}\, x\, \varphi \in \widetilde{\Gamma}$ and t is a term, $\mathrm{ALLL}_{x,\varphi,t}$ is enabled in σ and hence eventually taken, ensuring that $\varphi[t/x] \in \widetilde{\Gamma}$.

Let ξ be the embedding of variables into terms. To prove $\mathscr{S} \not\models_\xi \Gamma \rhd \Delta$, it suffices to show that $\forall \varphi \in \widetilde{\Gamma}.\ \mathscr{S} \models_\xi \varphi$ and $\forall \varphi \in \widetilde{\Delta}.\ \mathscr{S} \not\models_\xi \varphi$. These two facts follow together by induction on the depth of φ. In the base case, if $\mathsf{Atm}\, a \in \widetilde{\Gamma}$, then $\mathscr{S} \models_\xi \mathsf{Atm}\, a$ follows directly from the definition of $\mathscr{S}$; moreover, if $\mathsf{Atm}\, a \in \widetilde{\Delta}$, then by property 1 $\mathsf{Atm}\, a \notin \widetilde{\Gamma}$, hence again $\mathscr{S} \not\models_\xi \mathsf{Atm}\, a$ follows from the definition of $\mathscr{S}$. The only nontrivial inductive case is All, which requires Lemma 20. Assume $\mathsf{All}\, x\, \varphi \in \widetilde{\Gamma}$. By property 6, we have $\varphi[t/x] \in \widetilde{\Gamma}$ for any t. Hence, by the induction hypothesis, $\mathscr{S} \models_\xi \varphi[t/x]$. By Lemma 20, $\mathscr{S} \models_{\xi[x \leftarrow t]} \varphi$ for all t; that is, $\mathscr{S} \models_\xi \mathsf{All}\, x\, \varphi$. The second fact, concerning $\widetilde{\Delta}$, follows similarly from property 7. □

We have thus obtained:

Corollary 22 A sequent is provable by a (finite) proof in the Gentzen system of classical FOL iff it is satisfied by all FOL structures.

Remark 23 The rule ALLL stores, in the left context, a copy of the universal formula $\mathsf{All}\, x\, \varphi$ when applied backwards. 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 $\mathsf{All}\, x\, \varphi$ even after its use. If we labeled ALLL as $\mathrm{ALLL}_{x,\varphi}$ instead of $\mathrm{ALLL}_{x,\varphi,t}$, 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 21 (and hence concrete completeness) would not hold—property 6 from the lemma's proof would fail.

4.2 Further Instances

Theorem 18 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 [22] and G1, G2, G3, GS1, GS2, and GS3 from Troelstra and Schwichtenberg [51]. Persistence is easy to check. The syntax-independent part of the argument is provided by Theorem 18, while an ad hoc step analogous to Lemma 21 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 [26]—including those presented in Bell and Machover [1] and in Fitting [20]—are usually persistent when regarded as derivation systems. After an application of Theorem 18, the ad hoc argument for interpreting the abstract model is similar to that for Gentzen systems (Lemma 21).

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 have Persistence. A typical right rule for the modal operator $\Box$ ("must") is as follows [51]:

$$\frac{\Box\,\Gamma \rhd \Diamond\,\Delta, \varphi}{\Box\,\Gamma \rhd \Diamond\,\Delta, \Box\,\varphi}\ \textsc{MustR}$$

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 [37] 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

$$\frac{\Gamma, w\,R\,w' \rhd \Delta, w' : \varphi}{\Gamma \rhd \Delta, w : \Box\,\varphi}\ \begin{array}{l}\textsc{MustR}'\\ (w' \text{ fresh})\end{array}$$

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 fulfills Persistence, enabling Theorem 18. 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 the question: When can *infinite* proof trees be accepted as valid proofs?

We give a coinductive account of the abstract development of Brotherston et al. [16], in a slightly more general form since we work with arbitrary infinite proofs, which may be acyclic. We start by recalling some motivation and intuition about cyclic proofs, in the context of the $\mathrm{FOL}_{\mathsf{ind}}$ logic from Section 2.2. It is clear that the rules of the $\mathrm{FOL}_{\mathsf{ind}}$ Gentzen system 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. But other proofs are permissible too.

For the language of Example 1 from Section 2.2, consider the cyclic tree of Figure 4, where one of the leaf nodes 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.

5.1 Soundness of Infinite Proof Trees

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

$$\dfrac{\dfrac{}{\mathsf{even}(0) \rhd \mathsf{odd}(\mathsf{Suc}(0))}\,\textsc{Odd}_0 \quad \dfrac{\dfrac{\mathsf{Link}\ L}{\mathsf{even}(y) \rhd \mathsf{odd}(\mathsf{Suc}(y))}\,\textsc{Subst}_{y,x}}{\mathsf{even}(y) \rhd \mathsf{odd}(\mathsf{Suc}(\mathsf{Suc}(\mathsf{Suc}(y))))}\,\textsc{Odd}_{\mathsf{Suc}}}{L :: \mathsf{even}(x) \rhd \mathsf{odd}(\mathsf{Suc}(x))}\,\textsc{Even}_{\mathrm{split}}$$

Fig. 4 A cyclic proof tree

$$\dfrac{\dfrac{}{\mathsf{even}(0) \rhd \mathsf{odd}(\mathsf{Suc}(0))}\,\textsc{Odd}_0 \quad \dfrac{\dfrac{\dfrac{\vdots}{\mathsf{even}(x) \rhd \mathsf{odd}(\mathsf{Suc}(x))}}{\mathsf{even}(y) \rhd \mathsf{odd}(\mathsf{Suc}(y))}\,\textsc{Subst}_{y,x}}{\mathsf{even}(y) \rhd \mathsf{odd}(\mathsf{Suc}(\mathsf{Suc}(\mathsf{Suc}(y))))}\,\textsc{Odd}_{\mathsf{Suc}}}{\mathsf{even}(x) \rhd \mathsf{odd}(\mathsf{Suc}(x))}\,\textsc{Even}_{\mathrm{split}}$$

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

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

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 height 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 $(M, b, M') \in \mathsf{mark}\ (s, r, s')$, 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 $\mathsf{eff}\ s\ r\ ss$, there exist S' and $s' \in ss$ such that $S' \not\models s'$ and $\mathsf{descent}\ (s, S)\ r\ (s', S')$.

In the above, $\mathsf{descent} : \mathsf{sequent} \times \mathsf{structure} \to \mathsf{rule} \to \mathsf{sequent} \times \mathsf{structure} \to \mathsf{bool}$ is defined as follows:

$$\begin{array}{l}\mathsf{descent}\ (s, S)\ r\ (s', S') \iff \\ (\forall M\ b\ M'.\ (M, b, M') \in \mathsf{mark}\ (s, r, s') \Rightarrow \\ \qquad (b = \mathsf{False} \wedge \mathsf{height}\ (M', S') \leq \mathsf{height}\ (M, S)) \vee \\ \qquad (b = \mathsf{True} \wedge \mathsf{height}\ (M', S') < \mathsf{height}\ (M, S)))\end{array}$$

Remark 24 The Descent condition is a strengthening of the Local Soundness condition from Section 3.5: Removing the "and $\mathsf{descent}\ (s, S)\ r\ (s', S')$" 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, we can identify certain "good" infinite proof trees that 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 follow : bool stream $\to$ marker stream $\to$ step stream $\to$ bool is defined coinductively:

$$\frac{(M, b, \mathsf{shead}\ Ms) \in \mathsf{mark}\ (s, r, \mathsf{fst}\ (\mathsf{shead}\ \sigma)) \qquad \mathsf{follow}\ bs\ Ms\ \sigma}{\mathsf{follow}\ (\mathsf{SCons}\ b\ bs)\ (\mathsf{SCons}\ M\ Ms)\ (\mathsf{SCons}\ (s, r)\ \sigma)}\ \textsc{Follow}$$

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

We define a tree to be good if each of its infinite paths has 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:

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

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

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

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

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

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

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

Proof Fix s and T such that 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 8 (Section 3.6) for $\alpha = \mathsf{structure}$, $I = \lambda(s, S).\ S \not\models s$ and $P = \mathsf{descent}$ to the Descent assumption, we obtain a stream δ 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 alwCmodDesc δ ("always countermodel and descending"), where alwCmodDesc is defined as

$$\begin{aligned}&\mathsf{alw}\ (\lambda\delta.\ \mathsf{let}\ ((s, r), S) = \mathsf{shead}\ \delta\ \mathsf{and}\ ((s', \_), S') = \mathsf{shead}\ (\mathsf{stail}\ \delta)\\ &\qquad \mathsf{in}\ S \not\models s \wedge \mathsf{descent}\ (s, S)\ r\ (s', S'))\end{aligned}$$

   and the step components of δ form an infinite path in T: ipath T (smap fst δ).
2. Using the goodness of T, we obtain the streams bs and Ms that follow a suffix σ' of smap fst δ:

$$\exists \sigma'\ \sigma''\ bs\ Ms.\ \mathsf{smap\ fst}\ \delta = \sigma'' @ \sigma' \wedge \mathsf{follow}\ bs\ Ms\ \sigma' \wedge \mathsf{infDecr}\ bs$$

   where @ denotes concatenation of a list and a stream.

3. Taking the suffix δ' of δ that corresponds to the suffix σ' of its step component $\mathsf{smap}\ \mathsf{fst}\ \delta$ (formally, defining δ' to be $\mathsf{stake}\ (\mathsf{length}\ \sigma')\ \delta$), we end up with the streams δ', bs and Ms such that
   - $\mathsf{follow}\ bs\ Ms\ (\mathsf{smap}\ \mathsf{fst}\ \delta')$;
   - $\mathsf{infDecr}\ bs$;
   - $\mathsf{alwCmodDesc}\ \delta'$ (because alw is invariant under suffix).
4. Let zip denote the pair-zipping of two streams, defined by primitive corecursion as $\mathsf{zip}\ (\mathsf{SCons}\ x\ xs)\ (\mathsf{SCons}\ y\ \ ys) = \mathsf{SCons}\ (x,y)\ (\mathsf{zip}\ xs\ ys)$. Taking $ks = \mathsf{smap}\ \mathsf{height}\ (\mathsf{zip}\ (\mathsf{smap}\ \mathsf{fst}\ \delta')\ Ms)$ (a stream of ord elements), we have that
   - ks is always nonstrictly decreasing: $\mathsf{alw}\ (\lambda ks'.\ \mathsf{shead}\ ks' \geq \mathsf{shead}\ (\mathsf{stail}\ ks'))\ ks$;
   - ks is infinitely often strictly decreasing: $\mathsf{alw}\ (\mathsf{ev}\ (\lambda ks'.\ \mathsf{shead}\ (\mathsf{stail}\ ks') > \mathsf{shead}\ ks'))\ ks$.

   This follows easily by coinduction using the gathered properties for δ', bs, and Ms.
5. From ks, we obtain the substream ks' that is always strictly decreasing:

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

   which is in contradiction with the wellfoundedness of the order $<$ on the ordinal ord. □

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 is of the form $k_0 \cdot k_1 \cdot \ldots$ with $k_0 = k_1 = \cdots = k_n > k_{n+1} = k_{n+2} = \cdots = k_{n+m} > k_{n+m+1} = \cdots$, then ks' will be $k_0 \cdot k_{n+1} \cdot k_{n+m+1} \cdot \ldots$; that is, ks' will only contain the "jump" elements. Formally, this construction can be compactly described as $ks' = \mathsf{bfilter}\ (>, ks)$, where $> : \mathsf{ord} \to \mathsf{ord} \to \mathsf{bool}$ is the dual strict order on ord. The general-purpose polymorphic "binary filter" combinator, $\mathsf{bfilter} : D_\alpha \to \alpha\ \mathsf{stream}$, has as domain the subset D_α of $(\alpha \to \alpha \to \mathsf{bool}) \times \alpha\ \mathsf{stream}$ consisting of pairs (P, xs) such that, for every element x in xs, there exists an element y occurring later in xs such that $P\ x\ y$:

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

The bfilter function is defined as follows, where $\mathsf{bfilter}_P$ abbreviates $\mathsf{bfilter}\ (P, \_)$:

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

Thus, $\mathsf{bfilter}_P$ collects from a stream all the pairs of elements proximally related by P and forms a new stream with them. 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, $\mathsf{shead}\ xs$), 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 be at most a finite number of consecutive calls on the 'else' branch, until an element satisfying $P\ (\mathsf{shead}\ xs)$ is met. In summary, the *recursive* call on the 'else' branch concurs with the *corecursive* call on the 'then' branch for the welldefinedness of this function.

Thanks to recent infrastructure development [11], Isabelle accepts this mixed recursive–corecursive definition, after the user has shown that the recursion in the 'else' branch calls terminates: If $(P, xs) \in D_\alpha$, then in particular $\mathsf{ev}\ (\lambda xs''.\ P\ (\mathsf{shead}\ xs)\ (\mathsf{shead}\ xs''))$ holds for $\mathsf{stail}\ xs$. If we write xs' for the input to the recursive call, $\mathsf{SCons}\ (\mathsf{shead}\ xs)\ (\mathsf{stail}\ (\mathsf{stail}\ xs))$, we have that $\mathsf{shead}\ xs' = \mathsf{shead}\ xs$ and $\mathsf{stail}\ xs' = \mathsf{stail}\ (\mathsf{stail}\ xs)$. Hence:

$$\begin{aligned}\mathsf{num}_P\ (\mathsf{shead}\ xs')\ (\mathsf{stail}\ xs') &= \mathsf{num}_P\ (\mathsf{shead}\ xs)\ (\mathsf{stail}\ (\mathsf{stail}\ xs)) \\ &< \mathsf{num}_P\ (\mathsf{shead}\ xs)\ (\mathsf{stail}\ xs)\end{aligned}$$

where $\mathsf{num}_P\ z\ ys$ 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 example may seem excessive. But note that bfilter is a general-purpose combinator, defined once and usable for arbitrary predicates P.

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

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

5.2 From Cyclic Trees to Infinite Trees

With the general result for the soundness of infinite trees in place, let us come back to the notion of cyclic tree. 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:

$$\begin{aligned}
\textbf{datatype}\ \mathsf{cdtree} \ =\ &\mathsf{Node}\ \mathsf{step}\ (\mathsf{cdtree}\ \mathsf{fset})\\
\mid\ &\mathsf{Link}\ \mathsf{link}
\end{aligned}$$

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, links to other cyclic derivation trees. To animate the links, we fix a function $\mathsf{pointsTo} : \mathsf{link} \to \mathsf{cdtree}$ specifying, for each link, where it points to, with the following requirement:

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

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

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

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 $\mathsf{treeOf}\ (\mathsf{pointsTo}\ L)$ is guaranteed to not have the form $\mathsf{Link}\ L'$, 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 an instance of recursion–corecursion.

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

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

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

$$\frac{\mathsf{eff}\;r\;s\;(\mathsf{image}\;\mathsf{seqOf}\;Ts) \qquad \forall T \in Ts.\;\mathsf{cwf}\;T}{\mathsf{cwf}\;(\mathsf{Node}\;(s,r)\;Ts)}\;\textsc{Cwf-Node} \qquad \frac{\mathsf{cwf}\;(\mathsf{pointsTo}\;L)}{\mathsf{cwf}\;(\mathsf{Link}\;L)}\;\textsc{Cwf-Link}$$

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:

$$\mathsf{cproof}\;T\;s \iff \mathsf{cwf}\;T \wedge \mathsf{good}\;(\mathsf{treeOf}\;L) \wedge \mathsf{seqOf}\;T = s$$

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 $\mathsf{treeOf}\;T$, and we can prove by induction that wellformedness of T implies wellformedness of $\mathsf{treeOf}\;T$. With Theorem 25, this immediately gives our desired soundness theorem for cyclic derivation trees.

Theorem 26 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:

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

The above theorem is a slight generalization of the result by Brotherston et al. [16]. It is more general in that it does not require the cyclic proof trees to be regular. If we wanted to model precisely the regular trees, we would need to further restrain pointsTo; however, such restrictions do not affect soundness.

6 Concrete Instances of Infinite-Proof Soundness

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

Recall $\mathrm{FOL}_{\mathsf{ind}}$ from Section 2.2 and its associated Gentzen system. The abstract structures (elements of the set structure) are not instantiated by mere concrete structures, but rather by pairs $(\mathscr{S},\xi)$ where $\mathscr{S} = \big(S,\,(F_f)_{f\,\in\,\mathsf{fsym}},\,(P_p)_{p\,\in\,\mathsf{psym}}\big)$ is a structure and $\xi : \mathsf{var} \to S$ is a variable valuation. We instantiate the satisfaction relation, $(\mathscr{S},\xi) \models \varphi$, with $\mathscr{S} \models_\xi \varphi$.

To instantiate Theorems 25 and 26, we must define the notion of marker (forming the set marker), the ordinal ord, and the functions $\mathsf{mark} : \{(s,r,s') \in \mathsf{sequent} \times \mathsf{rule} \times \mathsf{sequent} \mid$

$\exists ss.\ \mathsf{eff}\ s\ r\ ss \wedge s' \in ss\} \to (\mathsf{marker} \times \mathsf{bool} \times \mathsf{marker})\ \mathsf{set}$ and $\mathsf{height} : \mathsf{marker} \times \mathsf{structure} \to \mathsf{ord}$ and verify the Descent property.

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

- Rule r is not of the form $\textsc{Subst}_{t,x}$ or p_{split}. Then, assuming $s = (\Gamma \rhd \Delta)$, we set

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

- Rule r has the form $\textsc{Subst}_{t,x}$. Then s has the form $\Gamma[t/x] \rhd \Delta[t/x]$, and we set

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

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

$$\begin{aligned}\mathsf{mark}\ (s, r, s') = {} & \{(M, \mathsf{False}, M[\bar{t}/\bar{x}]) \mid M \in \mathsf{marker} \cap \Gamma\} \cup \\ & \{(p(\bar{x}), \mathsf{True}, \psi) \mid \psi \in \mathsf{marker} \cap \mathsf{prems}(\chi')\}\end{aligned}$$

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

- $P_{p,0}\ \bar{a} \Leftrightarrow \mathsf{False}$;
- $P_{p,n+1}\ \bar{a} \Leftrightarrow P_{p,n}\ \bar{a} \vee (\exists \chi \in \mathsf{ind}_p.\ \exists \xi.\ \bar{a} = \xi\ (\mathsf{varsOf}(\mathsf{concl}(\chi))) \wedge \mathscr{S}_n \models_\xi \mathsf{prems}(\chi))$, where $\mathscr{S}_n = \big(S,\ (F_f)_{f \in \mathsf{fsym}},\ (P_{p,n})_{p \in \mathsf{psym}}\big)$ and $\mathsf{varsOf}(\mathsf{concl}(\chi))$ 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 $(P_p : S^{\mathsf{ar}\ p} \to S)_{p \in \mathsf{ipsym}}$. Thanks to the inductiveness of $\mathscr{S}$, it is well known that the approximations converge to their target:

Lemma 27 For all $p \in \mathsf{ipsym}$ and $\bar{a} \in S^{\mathsf{ar}\ p}$, we have $P_p\ \bar{a}$ iff $\exists n.\ P_{p,n}\ \bar{a}$.

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

- Assume $\mathscr{S} \models_\xi M$, and assume M has the form $p(\bar{t})$ for $p \in \mathsf{ipsym}$. Then $P_p(\llbracket \bar{t} \rrbracket_\xi^{\mathscr{S}})$. We let $\mathsf{height}\ (M, (\mathscr{S}, \xi))$ be the smallest n such that $P_{p,n}(\llbracket \bar{t} \rrbracket_\xi^{\mathscr{S}})$ (which exists by Lemma 27).
- Assume $\mathscr{S} \not\models_\xi M$. Then the value of $\mathsf{height}\ (M, (\mathscr{S}, \xi))$ is irrelevant, and we can put anything here.

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

We distinguish several cases, following the definition of mark:

- Rule r is not of the form $\textsc{Subst}_{t,x}$ or p_{split}. We further distinguish some subcases:
  - r is AX. This is impossible, since AX is sound for all (inductive) structures, which contradicts the hypothesis $\mathscr{S} \not\models_\xi s$.
  - 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 $\xi' = \xi$.

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

  - r is CONJL. Then we take $\xi' = \xi$ and s' to be one of the two premises, say, s_i, such that $\mathscr{S} \not\models_\xi s_i$ (which is known to exist by the soundness of CONJL).
  - r is ALLR. Then $s = (\Gamma \rhd \Delta, \mathsf{All}\ x\ \varphi)$, and we take s' to be the only premise of r, namely, $s' = \Gamma \rhd \Delta, \varphi[y/x]$, where y is known to be fresh. Since $\mathscr{S} \not\models_\xi \mathsf{All}\ x\ \varphi$, we obtain $a \in S$ such that $\mathscr{S} \not\models_{\xi[x \leftarrow a]} \varphi$; by the freshness of y, this also implies $\mathscr{S} \not\models_{\xi[y \leftarrow a]} \varphi[y/x]$. We let $\xi' = \xi[y \leftarrow a]$.
- Rule r has the form $\text{SUBST}_{t,x}$. Then s has the form $\Gamma[t/x] \rhd \Delta[t/x]$. We set $\xi' = \xi[x \leftarrow [\![t]\!]^{\mathscr{S}}_\xi]$ and s to the only premise of r.
- Rule r has the form p_{split} for some $p \in \mathsf{ipsym}$. Then s has the form $\Gamma, p(\bar{x}) \rhd \Delta$. Since $\mathscr{S} \models_\xi p(\bar{x})$, 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 $P_{p,\_}$, $m > 0$ and there exist $\chi \in \mathsf{ind}_p$ and the valuation ξ'' such that $\xi\, \bar{x} = \xi''\, \bar{x}$ and $\mathscr{S}_{m-1} \models_{\xi''} \mathsf{prems}(\chi')$.[3] We take s' to be the premise of r corresponding to χ, namely $s' = (\Gamma[\bar{t}/\bar{x}],\ \mathsf{prems}(\chi') \rhd \Delta[\bar{t}/\bar{x}])$. Finally, we define ξ' as follows:

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

For all the cases, it is routine to check (when necessary with the help of Lemma 20) that $\mathscr{S}' \not\models_{\xi'} s'$ and $\mathsf{descent}\ (s, (\mathscr{S}, \xi))\ r\ (s', (\mathscr{S}', \xi'))$. When checking the latter for $r = p_{\text{split}}$, we rely on the following observations:

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

Corollary 28 Assume that, in the Genzten system of $\text{FOL}_{\mathsf{ind}}$, a sequent $\Gamma \rhd \Delta$ either has an iproof or has a cyclic proof and $\mathsf{pointsTo}$ fulfills Good Links. Then $\Gamma \rhd \Delta$ is satisfied by all pairs $(\mathscr{S}, \xi)$, i.e., $\mathscr{S} \models_\xi \Gamma \rhd \Delta$. Hence, $\mathscr{S} \models \Gamma \rhd \Delta$.

Example 29 The infinite tree T of Figure 5 is an iproof, i.e., is well formed and satisfies the goodness predicate. To see the latter, observe that the only infinite path in this tree is $\sigma = ((s_1, r_1) \cdot (s_2, r_2) \cdot (s_3, r_3))^\omega$, where

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

If we let $bs = (\mathsf{True} \cdot \mathsf{False} \cdot \mathsf{False})^\omega$ and $Ms = (\mathsf{even}(x) \cdot \mathsf{even}(y) \cdot \mathsf{even}(y))^\omega$, we have that $\mathsf{follow}\ bs\ Ms\ \sigma$. Indeed, bs and Ms "follow" σ from the start, since

$$\begin{aligned} \mathsf{mark}\ (s_1, r_1, s_2) &= \{(\mathsf{even}(x), \mathsf{True}, \mathsf{even}(y))\} \\ \mathsf{mark}\ (s_2, r_2, s_3) &= \{(\mathsf{even}(y), \mathsf{False}, \mathsf{even}(y))\} \\ \mathsf{mark}\ (s_3, r_3, s_1) &= \{(\mathsf{even}(y), \mathsf{False}, \mathsf{even}(x))\} \end{aligned}$$

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

[3] The definition of $P_{p,\_}$ works with the original clauses $\chi \in \mathsf{ind}_p$, 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.

For accepting a cyclic or infinite proof in $\mathrm{FOL}_{\mathsf{ind}}$ 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 $(\_, \mathsf{True}, \_)$, meaning "decrease," only appears in the definition of mark for the split rule. In the above example, the predicate p that ensures goodness along the tree's unique 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 $\mathrm{FOL}_{\mathsf{ind}}$, 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 $\mathscr{S} = \big((S_w)_{w\in W},\ (F_{f,w})_{f\,\in\,\mathsf{fsym},w\in W},\ (P_{p,w})_{p\,\in\,\mathsf{psym},w\in W}\big)$, 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 $\mathrm{FOL}_{\mathsf{ind}}$, 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. [16], one for entailment [15] and one for termination proofs [14].

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 [7]). The formal development is publicly available [8, 10].

The necessary codatatypes and corecursive definitions are realized using a recently introduced definitional package for (co)datatypes [6] with support for mutual and nested (co)recursion [50] and mixed recursive–corecursive function definitions [11]. 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, and hence corresponds to a partial function $\mathsf{eff}' : \mathsf{rule} \to \mathsf{sequent} \to (\mathsf{sequent}\ \mathsf{fset})\ \mathsf{option}$, where the Isabelle datatype $\alpha\ \mathsf{option}$ 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 [25] can then produce Haskell code for the computable part of our completeness proof—the abstract prover mkTree, defined corecursively in the proof of Theorem 15:

[4] Goodness is decidable for cyclic trees in logics where rule application is decidable, such as $\mathrm{FOL}_{\mathsf{ind}}$ [16].

[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.

```
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 σ) =
  if p a then sdropWhile p σ else SCons a σ

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

Finite sets are represented as lists. The functions $\mathsf{isJust} : \alpha\ \mathsf{option} \to \mathsf{bool}$ and $\mathsf{fromJust} : \alpha\ \mathsf{option} \to \alpha$ 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 codatatypes in mind, it is general enough to handle them. Internally, it reduces Isabelle specifications to higher-order rewrite systems [35] 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. However, more research is needed to understand how our framework or a refinement of it can accommodate state-of-the-art proof search procedures, such as conflict-driven clause learning [5]. For example, one would rather want to specify ρ lazily (using unification) during, and not before, the evaluation of mkTree.

8 Related Work

This article joins a series of pearls aimed at reclaiming mathematical concepts and results for coinductive methods, including streams [43, 47], regular expressions [44, 46], and automata [45]. 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 [18], and temporal logic for red–blue trees [36].

So why write yet another 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 mixture of recursion and corecusion allowed in function definitions is unique to Isabelle. Second, because although codatatypes 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 [40] within institution-independent model theory [19]. However, most of these accounts 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 [37] gives a revealing account in the context of modal logic, quoting reviews that were favorable to Kripke's completeness result [32] but critical of his informal argument based on infinite tableau trees.[6] Kaplan [30] 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 [22]. 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 [26] 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 [26, 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 [22] 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; Bell and Machover [1] employ a different fair tree generation strategy, to the same effect; for a world-instrumented system for modal logic, Negri [37] employs yet another strategy.

Regrettably, one of the completeness results not covered by our abstract completeness is connected to our other case study: logics admitting infinite proofs. Brotherston and Simpson [17] show that completeness for $\mathrm{FOL}_{\mathsf{ind}}$ (with equality) can be achieved by allowing infinite proofs satisfying the Goodness condition from Section 5.1. This nonstandard completeness result is not captured by our framework from Section 3, which requires finite proofs.

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 [49], formalize a Henkin-style argument for possibly uncountable languages. Building on an early insight by Krivine [33] concerning the expressibility of the completeness proof in intuitionistic second-order logic, Ilik [28] analyzes Henkin-style arguments for classical and intuitionistic logic with respect to standard and Kripke models and formalizes them in Coq (without employing codatatypes).

At least four proofs were developed using HOL-based systems. Harrison [27], in HOL Light, and Berghofer [2], in Isabelle, opted for Henkin-style arguments. Berghofer's work was recently extended by Schlichtkrull [48] to prove the completeness of first-order reso-

[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 [51].

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

lution. Ridge and Margetson [34, 42], also 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 [22] 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 a wide range of FOL variants 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. The reviewers read the submitted paper carefully and made useful and insightful comments and suggestions. Blanchette was partially supported by the Deutsche Forschungsgemeinschaft (DFG) project Hardening the Hammer (grant NI 491/14-1). Popescu was partially supported by the EPSRC project Verification of Web-based Systems (VOWS, grant EP/N019547/1) and by the DFG project Security Type Systems and Deduction (grant NI 491/13-3). 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://www.isa-afp.org/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., Fleury, M., Weidenbach, C.: A verified SAT solver framework with learn, forget, restart, and incrementality. In: N. Olivetti, A. Tiwari (eds.) IJCAR 2016, *LNCS*, vol. 9706. Springer (2016)
6. 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*, vol. 8558, pp. 93–110. Springer (2014)
7. Blanchette, J.C., Popescu, A.: Mechanizing the metatheory of Sledgehammer. In: P. Fontaine, C. Ringeissen, R.A. Schmidt (eds.) FroCoS 2013, *LNCS*, vol. 8152, pp. 245–260. Springer (2013)
8. Blanchette, J.C., Popescu, A., Traytel, D.: Abstract completeness. In: G. Klein, T. Nipkow, L. Paulson (eds.) Archive of Formal Proofs. `http://www.isa-afp.org/entries/Abstract_Completeness.shtml` (2014)
9. 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)
10. Blanchette, J.C., Popescu, A., Traytel, D.: Formal development associated with this paper. `http://people.inf.ethz.ch/trayteld/compl-journal-devel.tgz` (2015)
11. 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)
12. 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)
13. Brotherston, J.: Sequent calculus proof systems for inductive definitions. Ph.D. thesis, University of Edinburgh (2006)
14. Brotherston, J., Bornat, R., Calcagno, C.: Cyclic proofs of program termination in separation logic. In: G.C. Necula, P. Wadler (eds.) POPL 2008, pp. 101–112. ACM (2008)
15. Brotherston, J., Distefano, D., Petersen, R.L.: Automated cyclic entailment proofs in separation logic. In: N. Bjørner, V. Sofronie-Stokkermans (eds.) CADE-23, *LNCS*, vol. 6803, pp. 131–146. Springer (2011)
16. Brotherston, J., Gorogiannis, N., Petersen, R.L.: A generic cyclic theorem prover. In: R. Jhala, A. Igarashi (eds.) APLAS 2012, *LNCS*, vol. 7705, pp. 350–367. Springer (2012)
17. Brotherston, J., Simpson, A.: Complete sequent calculi for induction and infinite descent. In: LICS 2007, pp. 51–62. IEEE Computer Society (2007)
18. Ciaffaglione, A., Gianantonio, P.D.: A certified, corecursive implementation of exact real numbers. Theor. Comput. Sci. **351**(1), 39–51 (2006)
19. Diaconescu, R.: Institution-Independent Model Theory. Studies in Universal Logic. Birkhäuser (2008)
20. Fitting, M.: First-Order Logic and Automated Theorem Proving, 2nd edn. Graduate Texts in Computer Science. Springer (1996)
21. Francez, N.: Fairness. Texts and Monographs in Computer Science. Springer (1986)
22. Gallier, J.H.: Logic for Computer Science: Foundations of Automatic Theorem Proving. Computer Science and Technology. Harper & Row (1986)
23. Gödel, K.: Über die Vollständigkeit des Logikkalküls. Ph.D. thesis, Universität Wien (1929)
24. Gordon, M.J.C., Melham, T.F. (eds.): Introduction to HOL: A Theorem Proving Environment for Higher Order Logic. Cambridge University Press (1993)
25. 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)
26. Hähnle, R.: Tableaux and related methods. In: A. Robinson, A. Voronkov (eds.) Handbook of Automated Reasoning, vol. I, pp. 100–178. Elsevier (2001)
27. 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)
28. Ilik, D.: Constructive completeness proofs and delimited control. Ph.D. thesis, École polytechnique (2010)
29. Jacobs, B., Rutten, J.: A tutorial on (co)algebras and (co)induction. Bull. Eur. Assoc. Theor. Comput. Sci. **62**, 222–259 (1997)
30. Kaplan, D.: Review of Kripke (1959) [32]. J. Symb. Log. **31**, 120–122 (1966)
31. Kleene, S.C.: Mathematical Logic. John Wiley & Sons (1967)
32. Kripke, S.: A completeness theorem in modal logic. J. Symb. Log. **24**(1), 1–14 (1959)
33. Krivine, J.L.: Une preuve formelle et intuitionniste du théorème de complétude de la logique classique. Bull. Symb. Log. **2**(4), 405–421 (1996)
34. Margetson, J., Ridge, T.: Completeness theorem. In: G. Klein, T. Nipkow, L. Paulson (eds.) Archive of Formal Proofs. `http://www.isa-afp.org/entries/Completeness.shtml` (2004)
35. Mayr, R., Nipkow, T.: Higher-order rewrite systems and their confluence. Theor. Comput. Sci. **192**(1), 3–29 (1998)
36. 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)

37. 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)
38. Nipkow, T., Klein, G.: Concrete Semantics: With Isabelle/HOL. Springer (2014)
39. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, *LNCS*, vol. 2283. Springer (2002)
40. Petria, M.: An institutional version of Gödel's completeness theorem. In: CALCO 2007, pp. 409–424 (2007)
41. Pfenning, F.: Review of "Jean H. Gallier: Logic for Computer Science, Harper & Row, New York 1986" [22]. J. Symb. Log. **54**(1), 288–289 (1989)
42. 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)
43. Roşu, G.: Equality of streams is a Π^0_2-complete problem. In: J.H. Reppy, J.L. Lawall (eds.) ICFP '06. ACM (2006)
44. 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)
45. 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)
46. 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)
47. Rutten, J.J.M.M.: Elements of stream calculus (an extensive exercise in coinduction). Electr. Notes Theor. Comput. Sci. **45**, 358–423 (2001)
48. Schlichtkrull, A.: Formalization of the resolution calculus for first-order logic. In: J.C. Blanchette, S. Merz (eds.) ITP 2016, *LNCS*, vol. 9807. Springer (2016)
49. Schlöder, J.J., Koepke, P.: The Gödel completeness theorem for uncountable languages. Formalized Mathematics **20**(3), 199–203 (2012)
50. 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 Computer Society (2012)
51. Troelstra, A.S., Schwichtenberg, H.: Basic Proof Theory, 2nd edn. Cambridge University Press (2000)