# Unified Classical Logic Completeness
## A Coinductive Pearl

Jasmin Christian Blanchette[1], Andrei Popescu[1,2], and Dmitriy Traytel[1]

[1] Fakultät für Informatik, Technische Universität München, Germany
[2] Institute of Mathematics Simion Stoilow of the Romanian Academy, Bucharest, Romania

**Abstract.** Codatatypes are absent from many programming languages and proof assistants. We make a case for their importance by revisiting a classic result: the completeness theorem for first-order logic established through a Gentzen system. The core of the proof establishes an abstract property of possibly infinite derivation trees, independently of the concrete syntax or inference rules. This separation of concerns simplifies the presentation. The abstract proof can be instantiated for a wide range of Gentzen and tableau systems as well as various flavors of first-order logic. The corresponding Isabelle/HOL formalization demonstrates the recently introduced support for codatatypes and the Haskell code generator.

## 1 Introduction

Gödel's completeness theorem [11] is a major result about first-order logic (FOL). It forms the basis of a wide range 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 syntactically consistent (i.e., no contradiction arises from it), then it has a model. 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, mathematicians belong to the semantic school, whereas computer scientists tend to take the other side of the argument. The completeness theorem, which combines syntax and semantics, is also disputed, with the result that each school has its own proof. In his review of Gallier's *Logic for Computer Science* [10], Pfenning, a fellow "syntactician," notes the following [27]:

> 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 completeness proof based on Hilbert systems follows the *Henkin style*: It employs a heavy bureaucratic apparatus to establish facts about deduction and conservative language extensions, culminating with a 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* (perhaps more properly called Beth–Hintikka–Schütte–Kanger style [19]). 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 emphasizes the tension of the argument: The deduction system exhaustively attempts 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. An otherwise extremely rigorous textbook such as Bell and Machover's [1] becomes slightly informal when defining 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 $n$.

At best, the trees are defined rigorously (e.g., as prefix-closed sets), but the actual reasoning relies on the intuitive understanding 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 lack of apparatus for reasoning about infinite structures is reflected in the realm of mechanical theorem proving. While many provers include mechanisms for defining freely generated datatypes, few have dedicated support for potentially infinite structures such as lazy trees. On the other hand, functional programmers routinely manipulate lazy data structures. Notably, lazy evaluation is the primary strategy in Haskell.

This paper presents a rigorous Beth–Hintikka-style proof of the completeness theorem, based on a Gentzen system. Besides the use of codatatypes, the other main novel aspect of the proof is its modularity. The core tree construction argument is isolated from the proof system and concrete formula syntax (Section 3). The abstract proof can be instantiated for a wide range of Gentzen and tableau systems as well as various flavors of FOL—e.g., with or without predicates, equality, or sorts (Sections 4 and 5). This modularization replaces the textbook proofs by analogy. The core of the argument amounts to reasoning about a functional program over lazy data structures.

The proof is formalized in Isabelle/HOL [26] (Section 6). The tree construction makes use of a new definitional package for codatatypes [35], which automates the derivation of characteristic theorems from specifications of the constructors. Through Isabelle's code generator [13], 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.

**Conventions.** Isabelle/HOL is a proof assistant based on classical higher-order logic (HOL) with Hilbert choice, the axiom of infinity, and rank-1 polymorphism. HOL notations are a mixture of functional programming and mathematical syntaxes.

In this paper, HOL 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 $\alpha$ are identified with sets.

## 2 A Gentzen System for First-Order Logic

We fix a first-order language: a countably infinite set $\mathsf{var}$ of variables $x$, $y$, $z$ and countable sets $\mathsf{fsym}$ and $\mathsf{psym}$ of function symbols $f$ and predicate symbols $p$ together with assignments $\mathsf{ar} : \mathsf{fsym} \to \mathsf{nat}$ and $\mathsf{ar} : \mathsf{psym} \to \mathsf{nat}$ 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 $\varphi$, $\psi$ are defined as follows:

$$\textbf{datatype}\ \mathsf{fmla} = \mathsf{Atm}\ \mathsf{atom} \mid \mathsf{Neg}\ \mathsf{fmla} \mid \mathsf{Conj}\ \mathsf{fmla}\ \mathsf{fmla} \mid \mathsf{All}\ \mathsf{var}\ \mathsf{fmla}$$

Thus, a formula may be an atom, a negation, a conjunction, or a universal quantification.

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 $\varphi$ by a structure $\mathscr{S}$ with respect to a variable valuation $\xi : \mathsf{var} \to S$ are defined in the standard way. For terms:

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

For atoms: $\mathscr{S} \models_{\xi} p(t_1, \ldots, t_n)$ iff $P_p\ \big([\![t_1]\!]^{\mathscr{S}}_{\xi}, \ldots, [\![t_n]\!]^{\mathscr{S}}_{\xi}\big)$. For formulas:

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

The following substitution lemma relates the notions of satisfaction and capture-avoiding substitution $\varphi[t/x]$ of a term $t$ for a variable $x$ in a formula $\varphi$:

**Lemma 1.** *$\mathscr{S} \models_{\xi} \varphi[t/x]$ iff $\mathscr{S} \models_{\xi[x \leftarrow [\![t]\!]^{\mathscr{S}}_{\xi}]} \varphi$.*

A sequent is a pair $\Gamma \rhd \Delta$ of finite formula sets. Satisfaction is extended to sequents: $\mathscr{S} \models_{\xi} \Gamma \rhd \Delta$ iff $(\forall \varphi \in \Gamma.\ \mathscr{S} \models_{\xi} \varphi) \Rightarrow (\exists \psi \in \Delta.\ \mathscr{S} \models_{\xi} \psi)$.

The proof system on sequents is defined inductively as follows, where $\Gamma, \varphi$ abbreviates the set $\Gamma \cup \{\varphi\}$ and $\Gamma, \varphi, \psi$ abbreviates $\Gamma \cup \{\varphi, \psi\}$:

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

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

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

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). The completeness theorem

$$\vdash \Gamma \rhd \Delta \;\vee\; (\exists \mathscr{S}, \xi.\; \mathscr{S} \not\models_{\xi} \Gamma \rhd \Delta)$$

states that any sequent $\Gamma \rhd \Delta$ either is provable (denoted by $\vdash$) or has a countermodel, i.e., a structure $\mathscr{S}$ and a valuation $\xi$ that falsify it.

## 3 Abstract Completeness

The proof of the completeness theorem is divided in two parts. The first part, described in this section, focuses on the core of the completeness argument in an abstract, syntax-free manner. This level captures the tension between the existence of a proof or of of an abstract notion of countermodel; the latter is introduced via what we call an escape path—an infinite sequence of states that "escapes" the proof attempt. The tension is distilled in a completeness result: Either there exists a finite derivation tree or there exists an infinite derivation tree with a suitable escape path. The second part maps the abstract escape path to a concrete, proof-system-specific countermodel. Section 4 performs this connection for the Gentzen system of Section 2.

**Rule Systems.** We abstract away the syntax of formulas and sequents and the specific rules of the proof system. We fix countable sets $\mathsf{state}$ and $\mathsf{rule}$ for sets and rules. We assume that the meaning of the rules is given by an effect relation $\mathsf{eff} : \mathsf{rule} \to \mathsf{state} \to \mathsf{state}\ \mathsf{fset} \to \mathsf{bool}$, where $\alpha\ \mathsf{fset}$ denotes the set of finite subsets of $\alpha$. The reading of $\mathsf{eff}\ r\ s\ ss$ is as follows: Starting from state $s$, applying rule $r$ expands $s$ into the states $ss$.

A state represents a formal statement in the logic. The triple $\mathscr{R} = (\mathsf{state}, \mathsf{rule}, \mathsf{eff})$ forms a *rule system*.

*Example 1.* The Gentzen system from Section 2 can be presented as a rule system. The set $\mathsf{state}$ is the set of sequents, and $\mathsf{rule}$ consists of the following: a rule $\textsc{Ax}_a$ for each atom $a$; rules $\textsc{NegL}_\varphi$ and $\textsc{NegR}_\varphi$ for each formula $\varphi$; rules $\textsc{ConjL}_{\varphi,\psi}$ and $\textsc{ConjR}_{\varphi,\psi}$ for each pair of formulas $\varphi$ and $\psi$; a rule $\textsc{AllL}_{x,\varphi,t}$ for each variable $x$, formula $\varphi$, and term $t$; and a rule $\textsc{AllR}_{x,\varphi}$ for each variable $x$ and formula $\varphi$.

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

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

**Derivation Trees.** Possibly infinite trees are represented by the following codatatype:

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

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

**Fig. 1.** A proof

This definition introduces a constructor $\mathsf{Node} : \alpha \to (\alpha\ \mathsf{tree})\ \mathsf{fset} \to \alpha\ \mathsf{tree}$ and two selectors $\mathsf{lab} : \alpha\ \mathsf{tree} \to \alpha$, $\mathsf{sub} : \alpha\ \mathsf{tree} \to (\alpha\ \mathsf{tree})\ \mathsf{fset}$. They 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 applied an infinite number of times.

A *step* combines the current state and the rule to be applied: $\mathsf{step} = \mathsf{state} \times \mathsf{rule}$. Derivation trees are defined as trees labeled by steps, $\mathsf{dtree} = \mathsf{step}\ \mathsf{tree}$, where the root's label $(s, r)$ represents the proved goal $s$ and the first (backward) applied rule $r$. The well-formed derivation trees are captured by the predicate $\mathsf{wf} : \mathsf{dtree} \to \mathsf{bool}$ defined by the coinductive rule[3]

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

Thus, the predicate wf is the greatest (weakest) solution to

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

The term $\mathsf{image}\ f\ A$ denotes the image of set $A$ through function $f$, and fst is the left projection operator.

The first assumption requires that the rule $r$ from the root be applied to obtain the subtrees' labels. The second assumption requires that wellformedness hold for the immediate subtrees. The coinductive nature of the definition ensures that these properties hold for arbitrarily deep subtrees of $T$, even if $T$ has infinite paths.

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

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

A *proof* of a state $s$ is a finite well-formed derivation tree with the state $s$ at its root. An infinite well-formed derivation tree represents a failed proof attempt.

*Example 2.* Given the instantiation of Example 1, 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] Double lines distinguish coinductive rules from their inductive counterparts.

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

**Fig. 2.** A failed proof attempt

**Escape Paths.** 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 $\alpha$ with the constructor $\mathsf{SCons}$ and the selectors $\mathsf{shead}$ and $\mathsf{stail}$:

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

The predicate $\mathsf{ipath} : \mathsf{step}\ \mathsf{stream} \to \mathsf{dtree} \to \mathsf{bool}$, which ascertains whether a stream of steps is an infinite path in a tree, is defined coinductively:

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

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

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

The following lemma is easy to prove by coinduction.

**Lemma 2.** *For any $\sigma$ and $T$, if $\mathsf{wf}\ T$ and $\mathsf{ipath}\ \sigma\ T$, then $\mathsf{epath}\ \sigma$.*

*Example 3.* The stream

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

where $s \cdot \sigma = \mathsf{SCons}\ s\ \sigma$ and $s^{\infty} = s \cdot s \cdot \ldots$ is an escape path for the tree of Figure 2.

Since the trees are finitely branching, König's lemma applies. Its proof occasions a warm-up corecursive definition.

**Lemma 3.** *If $T$ is infinite, there exists an infinite path $\sigma$ in $T$.*

*Proof.* By the contrapositive of $\textsc{Fin}$, if $\mathsf{Node}\ (s,r)\ Ts$ is infinite, there exists an infinite subtree $T \in Ts$. Let $\mathsf{f} : \{T \in \mathsf{dtree}.\ \neg\, \mathsf{finite}\ T\} \to \{T \in \mathsf{dtree}.\ \neg\, \mathsf{finite}\ T\}$ be a function witnessing this fact—i.e., $\mathsf{f}\ T$ is an immediate infinite subtree of $T$. The desired infinite path $\mathsf{p} : \{T \in \mathsf{dtree}.\ \neg\, \mathsf{finite}\ T\} \to \mathsf{step}\ \mathsf{stream}$ can be defined by primitive corecursion over the codatatype of streams: $\mathsf{p}\ T = \mathsf{SCons}\ (\mathsf{lab}\ T)\ (\mathsf{p}\ (\mathsf{f}\ T))$. The predicate $\mathsf{ipath}\ (\mathsf{p}\ T)\ T$ holds by straightforward coinduction on the definition of $\mathsf{ipath}$. $\square$

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

**Fig. 3.** A derivation tree with a countermodel path

**Countermodel Paths.** A countermodel path is a structure that witnesses the unprovability of a state $s$. Any escape path starting in $s$ is a candidate for a countermodel path, given that it indicates a way to apply the proof rules without reaching any result. For it to be a genuine countermodel path, all possible proofs must have been attempted. More specifically, whenever a rule becomes enabled along the escape path, it is eventually applied later in the sequence. For FOL with sequents as states, such paths can be used to produce actual countermodels by interpreting as true (resp. false) all statements made along the path on the left (resp. right) of the sequents.

A rule $r$ is *enabled* in a state $s$ if it has an effect (i.e., $\exists ss.\ \mathsf{eff}\ r\ s\ ss$). This is written $\mathsf{enabled}\ r\ s$. For any rule $r$, stream $\sigma$, and predicate $P : \alpha\ \mathsf{stream} \to \mathsf{bool}$:

- $\mathsf{taken}_r\ \sigma$ iff $r$ is taken at the start of the stream (i.e., $\mathsf{shead}\ \sigma = (s, r)$ for some $s$);
- $\mathsf{enabledAt}_r\ \sigma$ iff $r$ is enabled at the beginning of the stream (i.e., if $\mathsf{shead}\ \sigma = (s, r')$, then $\mathsf{enabled}\ r\ s$);
- $\mathsf{ev}\ P\ \sigma$ (“eventually $P$”) iff $P$ is true for some suffix of $\sigma$;
- $\mathsf{alw}\ P\ \sigma$ (“always $P$”) iff $P$ is true for all suffixes of $\sigma$.

A stream of steps $\sigma$ is *saturated* if, at each point, any enabled rule is taken at a later point: $\forall r \in \mathsf{rule}.\ \mathsf{alw}\ (\lambda\sigma'.\ \mathsf{enabledAt}_r\ \sigma' \Rightarrow \mathsf{ev}\ \mathsf{taken}_r\ \sigma')\ \sigma$. A *countermodel path* for a state $s$ is a saturated escape path $\sigma$ starting at $s$ (i.e., $\mathsf{shead}\ \sigma = (s, r)$ for some $r$).

*Example 4.* The escape path given in Example 3 is not saturated, because the rule $\textsc{AllL}_{x,p(x),z}$ is enabled starting from the first position but never taken.

*Example 5.* The escape path associated with 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 only enabled rules along the path are of the form $\textsc{AllL}_{x,p(x),\_}$, and each is always eventually taken.

**Completeness.** For the proof of completeness, we assume that the set of rules satisfies the following properties:

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

The above conditions are local properties of the rules’ effect, not global properties of the proof system. This makes them easy to verify for particular systems.

Saturation is a stronger condition than the standard properties of fairness and justice [9]. Fairness would require the rules to be continuously enabled to guarantee that they are eventually taken. The property of justice is stronger in that it would require the rules to be enabled infinitely often, but not necessarily continuously. Saturation goes further: If a rule is ever enabled, it will certainly be chosen at a later point. Saturation may seem too strong for the task at hand; however, in the presence of persistence, the notions of fairness, justice, and saturation all coincide.

**Theorem 4.** *Given a rule system that fulfills availability and persistence, every state admits a proof or a countermodel path.*

*Proof.* The proof uses the following combinators:

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

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

Let $s$ be a state. 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 will also belong to fair. Given $\rho \in \mathsf{fair}$ and $s \in \mathsf{state}$, we assume $\mathsf{sdropWhile}\ (\lambda r.\ \neg\, \mathsf{enabled}\ r\ s)\ \rho$ has the form $\mathsf{SCons}\ r\ \rho'$, making $r$ the first enabled rule in $\rho$. Such a rule exists because, by availability, at least one rule is enabled in $s$ and, by fairness, all the rules occur in $\rho$. Since $\mathsf{enabled}\ r\ s$, we can pick a state set $ss$ such that $\mathsf{eff}\ r\ s\ ss$. We define $\mathsf{mkTree} : \mathsf{fair} \to \mathsf{state} \to \mathsf{dtree}$ corecursively as $\mathsf{mkTree}\, \rho\ s = \mathsf{Node}\ (s, r)\ (\mathsf{image}\ (\mathsf{mkTree}\, \rho')\ ss)$.

We prove that, for all $\rho \in \mathsf{fair}$ and $s$, the derivation tree $\mathsf{mkTree}\, \rho\ s$ 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 state $s$ and $\mathsf{ipath}\ (\mathsf{mkTree}\, \rho\ s)\ \sigma$, then $r$ appears along $\sigma$ (i.e., there exists a state $s'$ such that $(s', r)$ is in $\sigma$). This follows by induction on the position of $r$ in $\rho$, $\mathsf{pos}\ r\, \rho$—formally, the length of the shortest list $\rho_0$ such that $\rho = \rho_0\ @\ \mathsf{SCons}\ r\ \_$, where @ denotes concatenation. Let $r'$ be the first rule from $\rho$ enabled at state $s$. If $r = r'$, then $\mathsf{mkTree}\, \rho\ s$ has label $(s, r)$ already. Otherwise, $\rho$ has the form $\rho_1\ @\ [r']\ @\ \rho'$, with $r$ not in $\rho_1$, hence $\mathsf{pos}\ r\, \rho' < \mathsf{pos}\ r\, \rho$. From the definitions of ipath and mkTree, it follows that $\mathsf{ipath}\ (\mathsf{mkTree}\, \rho'\ s')\ (\mathsf{stail}\ \sigma)$ holds for some $s' \in ss$ such that $\mathsf{eff}\ r\ s'\ ss$. By the induction hypothesis, $r$ appears along $\mathsf{stail}\ \sigma$, hence along $\sigma$, as desired. In particular, $T_0 = \mathsf{mkTree\ fenum}\ s$ is well-formed and all its infinite paths are saturated.

Finally, if $T_0$ is finite, it is the desired finite derivation tree. Otherwise, by Lemma 3 (König) it has an infinite path. This path is necessarily saturated; by Lemma 2, it is the desired countermodel path. □

Theorem 4 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.

If we are not interested in witnessing the proof attempt closely, Theorem 4 can be established more directly, by constructing the fair path without going through an intermediate fair tree. The key observation is that if a state $s$ has no proof and $\mathsf{eff}\ r\ s\ ss$, there must exist a state $s' \in ss$ that has no proof (otherwise we would compose by $r$ the proofs of all $s'$ into a proof of $s$). 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 $\mathsf{mkPath} : \mathsf{fair} \to \{s \in \mathsf{state}.\ s \text{ has no proof}\} \to \mathsf{step\ stream}$ following the same idea as for the previous tree construction (function $\mathsf{mkTree}$): $\mathsf{mkPath}\ \rho\ s = \mathsf{SCons}\ (s, r)\ (\mathsf{mkPath}\ \rho'\ (\mathsf{pick}\ r\ s\ ss))$, 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.

**Omitting the Availability Assumption.** The above result 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 5.** *A rule system* $\mathscr{R} = (\mathsf{state}, \mathsf{rule}, \mathsf{eff})$ *that fulfills persistence can be transformed into an equivalent rule system* $\mathscr{R}_{\mathrm{idle}} = (\mathsf{state}, \mathsf{rule}_{\mathrm{idle}}, \mathsf{eff}_{\mathrm{idle}})$ *that fulfills both persistence and availability, with* $\mathsf{rule}_{\mathrm{idle}} = \mathsf{rule} \cup \{\textsc{Idle}\}$ *and* $\mathsf{eff}_{\mathrm{idle}}$ *behaving like* $\mathsf{eff}$ *on* $\mathsf{rule}$ *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 state. The modified system is equivalent to the original one because IDLE does not alter the state. □

**Theorem 6.** *Given a rule system* $\mathscr{R}$ *that fulfills persistence, every state admits a proof over* $\mathscr{R}$ *or a countermodel path over* $\mathscr{R}_{\mathrm{idle}}$.

*Proof.* We first apply Theorem 4 to the system $\mathscr{R}_{\mathrm{idle}}$ to obtain that every state admits either a proof or a countermodel path, both in this system. And since $\mathscr{R}$ and $\mathscr{R}_{\mathrm{idle}}$ are equivalent, any proof of $\mathscr{R}_{\mathrm{idle}}$ yields one of $\mathscr{R}$. □

## 4 Concrete Completeness

The abstract completeness proof is parameterized by a rule system. This section concretizes the result for the Gentzen system from Section 2 to derive the standard completeness theorem. Example 1 recast it as a rule system; we must verify that it fulfills persistence and interpret abstract countermodel paths as actual FOL countermodels.

The Gentzen rules are persistent because they preserve the context surrounding the eigenformulas. For example, an application of $\textsc{Ax}_a$ (which affects only the atom $a$) leaves any potential enabledness of $\textsc{AllL}_{x,\varphi,t}$ (which affects only formulas with $\mathsf{All}$ at the top) unchanged; moreover, $\textsc{Ax}_a$ does not overlap with $\textsc{Ax}_b$ for $a \neq b$. The only subtlety concerns $\textsc{AllR}_{x,\varphi}$, which requires the existence of a fresh $y$. But since the sequents are finite, we can always find a fresh variable in the infinite set $\mathsf{var}$.

On the other hand, availability does not hold for the proof system; for example, the sequent $p(x) \rhd q(x)$ has no enabled rule. Hence, we need Theorem 6 and its $\textsc{Idle}$ rule.

**Lemma 7.** *If* $\Gamma \rhd \Delta$ *admits a countermodel path, there exist a structure* $\mathscr{S}$ *and a valuation* $\xi : \mathsf{var} \to S$ *such that* $\mathscr{S} \not\models_\xi \Gamma \rhd \Delta$.

*Proof.* Let $\sigma$ be a countermodel path for $\Gamma \rhd \Delta$ (i.e., a saturated escape path having $\Gamma \rhd \Delta$ as the state from the head). Let $\tilde{\Gamma}$ be the union of the left-hand sides of sequents occurring in $\sigma$, and let $\tilde{\Delta}$ be the union of the corresponding right-hand sides. Clearly, $\Gamma \subseteq \tilde{\Gamma}$ and $\Delta \subseteq \tilde{\Delta}$. The pair $(\tilde{\Gamma}, \tilde{\Delta})$ can be shown to be well-behaved with respect to all the connectives and quantifiers in the following sense:

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

These properties follow from the saturation of $\sigma$ with respect to the corresponding rules. The proofs are routine. For example, if $\mathsf{All}\ x\, \varphi \in \tilde{\Gamma}$ and $t$ is a term, $\textsc{AllL}_{x,\varphi,t}$ is enabled in $\sigma$ and hence eventually taken, ensuring that $\varphi[t/x] \in \tilde{\Gamma}$.

We construct the concrete countermodel $\mathscr{S} = (S, F, P)$ as follows. We let the domain $S$ be the set $\mathsf{term}$ and $\xi$ be the embedding of variables into terms. For each $n$-ary $f$ and $p$ and each $t_1, \ldots, t_n \in S$, we define $F_f\ (t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$ and $P_p\ (t_1, \ldots, t_n) \Leftrightarrow p(t_1, \ldots, t_n) \in \tilde{\Gamma}$.

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

**Theorem 8.** *For any sequent* $\Gamma \rhd \Delta$, *we have* $\vdash \Gamma \rhd \Delta \,\vee\, (\exists \mathscr{S}, \xi.\ \mathscr{S} \not\models_\xi \Gamma \rhd \Delta)$.

*Proof.* From Theorem 6 and Lemma 7. □

## 5 Further Concrete Instances

Theorem 6 is applicable to classic 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 $\mathrm{G}_=$ (the latter for FOL with equality) from Gallier [10] and the systems G1, G2, G3, GS1, GS2, and GS3 from Troelstra and Schwichtenberg [36]. Persistence is easy to check. The syntax-independent part of the argument is provided by Theorem 6, while an ad hoc step analogous to Lemma 7 is required to build a concrete countermodel.

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 [14]—including those presented in Bell and Machover [1] and in Fitting [8]—are usually persistent when regarded as derivation systems. After an application of Theorem 6, the argument for interpreting the abstract model is similar to that for Gentzen systems (Lemma 7).

Regrettably, abstract completeness is not directly applicable beyond classical logic. It is generally not clear how to extract a specific model from a nonstandard logic from an abstract (proof-theoretic) model. Another issue is that standard sequent systems for nonclassical variations of FOL such as modal or intuitionistic logics do not satisfy persistence. A typical right rule for the modal operator $\Box$ ("must") is as follows [36]:

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

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 [25] organizes modal logics as labeled transition systems, for which Kripke completeness is derived. 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 only resilient facts are ever assumed about the context. The resulting proof system satisfies persistence, enabling Theorem 6. The Kripke countermodel construction is roughly as for classic FOL Gentzen systems.

## 6 Formalization and Implementation

The definitions, lemmas, and theorems presented in Sections 2 to 4 are formalized in the proof assistant Isabelle/HOL. The instantiation step of Section 4 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 [4]). The formal development is publicly available [5].

The necessary codatatypes and corecursive definitions are realized using a recently introduced definitional package [35]. 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.

For generating code, we make the additional assumption that the effect relation is a partial function $\mathsf{eff}' : \mathsf{rule} \to \mathsf{state} \to (\mathsf{state}\ \mathsf{fset})\ \mathsf{option}$, where the Isabelle datatype $\alpha\ \mathsf{option}$ enriches a copy of $\alpha$ with a special value None. From this function, we build the relational eff as the partial function's graph. Isabelle's code generator [12, 13] can then produce Haskell code for the computable part of our completeness proof: the abstract prover mkTree, defined corecursively in the proof of Theorem 4. The code is reproduced below:

```
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 (state, rule, eff), the code for mkTree explicitly takes eff as a parameter.

Although the 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 [23] and generates functional code in Haskell, OCaml, Scala, or Standard ML. Partial correctness is guaranteed irrespective 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 (state, rule, eff), a fair rule enumeration $\rho$, and a state $s$ representing the formula to prove, $\mathsf{mkTree}\ \mathsf{eff}\ \rho\ s$ yields a finite derivation tree of $s$ if $s$ is provable in the system; otherwise, it produces an infinite fair derivation tree whose infinite paths are all countermodel paths. These guarantees involve only partial correctness of ground term evaluation.[4]

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

## 7 Related Work

This paper joins a series of pearls aimed at reclaiming mathematical concepts and results for coinductive methods, including streams [29, 33], regular expressions [30, 32], and automata [31]. Some developments pass the ultimate test of formalization, usually

[4] There are subtle "moral correctness" aspects [7] concerning the transport of statements between Isabelle's and Haskell's type systems that are yet to be established for codatatypes, with Isabelle's domain package [16] as a potential mediator.

in Agda and Coq, the codatatype-aware proof assistants par excellence: Eratosthenes' sieve [3], real number basics [6], and temporal logic for red-blue trees [24].

So why write yet another formalized pearl involving coinduction? First, because we finally could—with the new codatatype package, Isabelle has caught up with its rivals in this area. Second, because, although codatatypes are a good match for the completeness theorem, there seems to be no proof in the literature that takes advantage of this.

While there are many accounts of the completeness theorem for FOL and related logics, most of them prefer 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 [25] gives a revealing account in the context of modal logic, quoting reviews that were favorable to Kripke's completeness result [20] but critical of his informal argument based on infinite tableau trees.[5] Kaplan [18] 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 [10]. 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 [14] first performs an implicit switch from finite to possibly infinite tableaux, and then claims that tableau chain suprema exist by wrongly invoking Zorn's lemma [14, Definition 3.16].[6]

The completeness theorem has been mechanized before in proof assistants. Schlöder and Koepke, in Mizar [34], formalize a Henkin-style argument for possibly uncountable languages. Building on an early insight by Krivine [21] concerning the expressibility of the completeness proof in intuitionistic second-order logic, Ilik [17] analyzes Henkin-style arguments for classic and intuitionistic logic with respect to standard and Kripke models and formalizes them in Coq (without employing codatatypes).

At least three proofs were developed using HOL-based provers. Harrison [15], in HOL Light, and Berghofer [2], in Isabelle, formalize Henkin-style arguments. Ridge and Margetson [22, 28], in Isabelle, employ proof trees constructed as graphs of nodes each carrying its level as a natural number. Their work has the merits of analyzing the computational content of proofs in the style of Gallier [10] and discussing an OCaml implementation. Our formalization improves over this work in a similar way in which our presentation improves over Gallier's: The newly introduced codatatype and corecursion support in Isabelle provides the right 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 more abstract, applying to several variants of FOL and beyond.

[5] And Kripke's degree of rigor in this early paper is not far from today's state of the art in proof theory; see, e.g., Troelstra and Schwichtenberg [36].

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

## 8 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. Gallier's pseudo-Pascal code is inspiring, but we find "pseudo-Haskell," i.e., Isabelle/HOL with codatatypes, superior to combine computational intuition and mathematical rigor.

Codatatypes are the key to formulate an account that is both rigorous and abundant in algorithmic content. 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 paper, like its predecessor [4], 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 paper. Blanchette is supported by the Deutsche Forschungsgemeinschaft (DFG) project Hardening the Hammer (grant Ni 491/14-1). Popescu is supported by the DFG project Security Type Systems and Deduction (grant Ni 491/13-2) as part of the program Reliably Secure Software Systems ($RS^3$, priority program 1496). Traytel is 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: Klein, G., Nipkow, T., Paulson, L. (eds.) Archive of Formal Proofs. http://afp.sf.net/entries/FOL-Fitting.shtml (2007)
3. Bertot, Y.: Filters on coinductive streams, an application to Eratosthenes' sieve. In: Urzyczyn, P. (ed.) TLCA 2005. LNCS, vol. 3461, pp. 102–115. Springer (2005)
4. Blanchette, J.C., Popescu, A.: Mechanizing the metatheory of Sledgehammer. In: Fontaine, P., Ringeissen, C., Schmidt, R.A. (eds.) FroCoS 2013. LNAI, vol. 8152, pp. 245–260. Springer (2013)
5. Blanchette, J.C., Popescu, A., Traytel, D.: Formal development associated with this paper. http://www21.in.tum.de/~traytel/compl_devel.zip (2014)
6. Ciaffaglione, A., Gianantonio, P.D.: A certified, corecursive implementation of exact real numbers. Theor. Comput. Sci. 351(1), 39–51 (2006)
7. Danielsson, N.A., Hughes, J., Jansson, P., Gibbons, J.: Fast and loose reasoning is morally correct. In: Morrisett, J.G., Jones, S.L.P. (eds.) POPL 2006. pp. 206–217. ACM (2006)
8. Fitting, M.: First-Order Logic and Automated Theorem Proving, Second Edition. Graduate Texts in Computer Science, Springer (1996)
9. Francez, N.: Fairness. Texts and Monographs in Computer Science, Springer (1986)
10. Gallier, J.H.: Logic for Computer Science: Foundations of Automatic Theorem Proving. Computer Science and Technology, Harper & Row (1986)
11. Gödel, K.: Über die Vollständigkeit des Logikkalküls. Ph.D. thesis, Universität Wien (1929)
12. Haftmann, F.: Code Generation from Specifications in Higher-Order Logic. Ph.D. thesis, Technische Universität München (2009)

13. Haftmann, F., Nipkow, T.: Code generation via higher-order rewrite systems. In: Blume, M., Kobayashi, N., Vidal, G. (eds.) FLOPS 2010. LNCS, vol. 6009, pp. 103–117. Springer (2010)
14. Hähnle, R.: Tableaux and related methods. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. I, pp. 100–178. Elsevier (2001)
15. Harrison, J.: Formalizing basic first order model theory. In: Grundy, J., Newey, M.C. (eds.) TPHOLs '98. LNCS, vol. 1479, pp. 153–170. Springer (1998)
16. Huffman, B.: A purely definitional universal domain. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 260–275. Springer (2009)
17. Ilik, D.: Constructive Completeness Proofs and Delimited Control. Ph.D. thesis, École Polytechnique (2010)
18. Kaplan, D.: Review of Kripke (1959) [20]. J. Symb. Log. 31, 120–122 (1966)
19. Kleene, S.C.: Mathematical Logic. John Wiley & Sons (1967)
20. Kripke, S.: A completeness theorem in modal logic. J. Symb. Log. 24(1), 1–14 (1959)
21. Krivine, J.L.: Une preuve formelle et intuitionniste du théorème de complétude de la logique classique. Bull. Symb. Log. 2(4), 405–421 (1996)
22. Margetson, J., Ridge, T.: Completeness theorem. In: Klein, G., Nipkow, T., Paulson, L. (eds.) Archive of Formal Proofs. `http://afp.sf.net/entries/Completeness.shtml` (2004)
23. Mayr, R., Nipkow, T.: Higher-order rewrite systems and their confluence. Theor. Comput. Sci. 192(1), 3–29 (1998)
24. 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: Yang, H. (ed.) APLAS 2011. LNCS, vol. 7078, pp. 353–368. Springer (2011)
25. Negri, S.: Kripke completeness revisited. In: Primiero, G., Rahman, S. (eds.) Acts of Knowledge: History, Philosophy and Logic: Essays Dedicated to Göran Sundholm, pp. 247–282. College Publications (2009)
26. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)
27. Pfenning, F.: Review of "Jean H. Gallier: Logic for Computer Science, Harper & Row, New York 1986" [10]. J. Symb. Log. 54(1), 288–289 (1989)
28. Ridge, T., Margetson, J.: A mechanically verified, sound and complete theorem prover for first order logic. In: Hurd, J., Melham, T.F. (eds.) TPHOLs 2005. LNCS, vol. 3603, pp. 294–309. Springer (2005)
29. Roşu, G.: Equality of streams is a $\Pi_2^0$-complete problem. In: Reppy, J.H., Lawall, J.L. (eds.) ICFP '06. ACM (2006)
30. Roşu, G.: An effective algorithm for the membership problem for extended regular expressions. In: Seidl, H. (ed.) FoSSaCS 2007. LNCS, vol. 4423, pp. 332–345. Springer (2007)
31. Rutten, J.J.M.M.: Automata and coinduction (an exercise in coalgebra). In: Sangiorgi, D., de Simone, R. (eds.) CONCUR '98. LNCS, vol. 1466, pp. 194–218. Springer (1998)
32. Rutten, J.J.M.M.: Regular expressions revisited: A coinductive approach to streams, automata, and power series. In: Backhouse, R.C., Oliveira, J.N. (eds.) MPC 2000. LNCS, vol. 1837, pp. 100–101. Springer (2000)
33. Rutten, J.J.M.M.: Elements of stream calculus (an extensive exercise in coinduction). Electr. Notes Theor. Comput. Sci. 45, 358–423 (2001)
34. Schlöder, J.J., Koepke, P.: The Gödel completeness theorem for uncountable languages. Formalized Mathematics 20(3), 199–203 (2012)
35. Traytel, D., Popescu, A., Blanchette, J.C.: Foundational, compositional (co)datatypes for higher-order logic: Category theory applied to theorem proving. In: LICS 2012, pp. 596–605. IEEE (2012)
36. Troelstra, A.S., Schwichtenberg, H.: Basic Proof Theory, Second Edition. Cambridge University Press (2000)