TPLP **20** (6): 990–1005, 2020.
doi:10.1017/S1471068420000423

The New Normal: We Cannot Eliminate Cuts in Coinductive Calculi, But We Can Explore Them

Ekaterina Komendantskaya
Heriot-Watt University, Edinburgh, Scotland, UK
(*e-mail:* ek19@hw.ac.uk)

Dmitry Rozplokhas
Jet Brains Research, St Petersburgh, Russia
(*e-mail:* rozplokhas@gmail.com)

Henning Basold
Leiden University, the Netherlands
(*e-mail:* henning@basold.eu)

submitted 10 August 2020; revised 11 August 2020; accepted 12 August 2020

Abstract

In sequent calculi, cut elimination is a property that guarantees that any provable formula can be proven analytically. For example, Gentzen's classical and intuitionistic calculi **LK** and **LJ** enjoy cut elimination. The property is less studied in coinductive extensions of sequent calculi. In this paper, we use coinductive Horn clause theories to show that cut is not eliminable in a coinductive extension of **LJ**, a system we call **CLJ**. We derive two further practical results from this study. We show that CoLP by Gupta et al. gives rise to cut-free proofs in **CLJ** with fixpoint terms, and we formulate and implement a novel method of coinductive theory exploration that provides several heuristics for discovery of cut formulae in **CLJ**.

KEYWORDS: Sequent Calculus, Horn Clauses, Coinduction, Cut Elimination, Theory Exploration.

1 Introduction

Cut elimination is one of the central properties of interest for sequent calculi (Gentzen 1969), and more generally, proof theory. Informally, whenever we want to prove a formula φ relative to a given theory Γ, we can use cut to first prove another formula ψ, and then show that ψ implies φ:

$$\frac{\Gamma \vdash \psi \qquad \Gamma, \psi \vdash \varphi}{\Gamma \vdash \varphi}\ \text{CUT}$$

The *cut elimination property* holds if every proof of a sequent that uses a cut, can be transformed into a cut-free proof. Cut elimination serves as a form of completeness result for the calculus: cut-free proofs can be constructed analytically by simply following the structure of formulae, eliminating any need to discover a cut formula. For first-order logic, the most famous example of a calculus with eliminable cut is Gentzen's system **LJ**.

Recently, coinduction became a prominent proof method, and has been incorporated

into a number of proof systems, see e.g. (Brotherston and Simpson 2011) for cyclic proof systems or (Gupta et al. 2007) for coinductive logic programming (CoLP). Informally speaking, coinductive extensions of proof systems give finitary methods to prove formulae that would otherwise require an infinite proof. Usually, it comes in the shape of a fixpoint rule:

$$\boxed{\frac{\Gamma, \varphi \vdash \varphi}{\Gamma \vdash \varphi}\ \textsc{co-fix}}$$

The rule allows one to add the formula φ, that would otherwise cause infinite derivations, directly to the set of assumptions, and thus close the proof coinductively in a finite number of steps. Sometimes φ is called a *coinduction hypothesis*. The CO-FIX rule usually comes with certain guardedness or productivity conditions. These vary from system to system, but always serve to guarantee soundness of the rule.

Only recently, the relation between these two principles, cut elimination and coinduction, attracted special attention of the proof-theoretic community. A series of papers (Saotome et al. 2020; Kimura et al. 2020) showed that cut is not eliminable in a cyclic first-order Separation logic. In this paper, we show that this problem is more general: Adding a coinduction rule to a first-order proof system destroys the property of cut elimination. We show this for the Gentzen's intuitionistic sequent calculus **LJ**, although a similar argument works for Gentzen's **LK**, and any sequent calculus for a logic with implication and universal and existential quantification. We call **LJ** augmented with the cofix rule *Coinductive* ***LJ***, or simply **CLJ**, and show that cut is not eliminable in **CLJ**.

The system **CLJ** is very similar to *coinductive uniform proofs* (CUP) (Basold et al. 2019), only that CUP does not feature a cut rule. CUP is a coinductive extension of uniform proofs, a fragment of the Gentzen's sequent calculus introduced to model the derivations obtained by first-order resolution in Prolog (Miller et al. 1991; Miller and Nadathur 2012). As it turns out, CUP is sound with respect to the largest Herbrand models of logic programs (Basold et al. 2019).

We apply our result in two ways. Firstly, we show that derivations in CoLP (Gupta et al. 2007) in fact correspond to cut-free proofs in **CLJ**. This gives a proof-theoretic characterisation to the well-known results of incompleteness of CoLP. Moreover, our characterisation of CoLP's loops by fixpoint terms may pave the way for future embeddings of CoLP in richer theorem provers.

Secondly, seeing that we cannot hope to prove all theorems of interest analytically, we propose to establish a stronger infrastructure for *theory exploration* in coinductive first-order theories. Similarly to the *Boyer-Moore Waterfall Model* (Boyer and Moore 1979), the methodology consists of four steps: (1) use a suitable coinductive sequent calculus (e.g. **CLJ** without cut or CUP) to prove analytically as much as possible; (2) use first-order resolution to explore the loops in derivations and suggest suitable coinductive lemmas; (3) use the calculus to prove the discovered lemmas and discard those that cannot be proven; (4) use the proven lemmas as cut formulae to complete previously failed proofs.

We present an implementation of this method, that comprises an implementation of CUP, several coinductive theory exploration methods from the literature, including CoLP and the method of Fu et al. (2016), as well as one novel theory exploration method. The implementation is available on Github[1]. These results are of interest to either logic

[1] `https://github.com/CoUniform/theory-exploration`

https://doi.org/10.1017/S1471068420000423 Published online by Cambridge University Press

$$\dfrac{\dfrac{\dfrac{\dfrac{}{p(x) \equiv p(x)}}{\{p(x)\} + \emptyset + \{\forall x.\, p(x)\} \vdash p(x)}\,\textbf{(Axiom)} \qquad \dfrac{\dfrac{\dfrac{}{p(f(x)) \equiv p(f(x))}}{\emptyset + \{p(f(x))\} + \emptyset \vdash p(f(x))}\,\textbf{(Axiom)}}{\emptyset + \{\forall x.\, p(x)\} + \emptyset \vdash p(f(x))}\,(\forall\textbf{-L-G})}{\{p(f(x)) \to p(x)\} + \emptyset + \{\forall x.\, p(x)\} \vdash p(x)}\,(\to\textbf{-L-T})}{\Gamma_T + \emptyset + \{\forall x.\, p(x)\} \vdash p(x)}\,(\forall\textbf{-L-T})}{\Gamma_T + \emptyset + \{\forall x.\, p(x)\} \vdash \forall x.\, p(x)}\,(\forall\textbf{-R})}{\spadesuit}\,\textbf{(CO-FIX)}$$

$$\dfrac{\dfrac{\spadesuit}{\Gamma_T + \emptyset + \emptyset \vdash \forall x.\, p(x)} \qquad \dfrac{\dfrac{}{p(a) \equiv p(a)}\,\textbf{(Axiom)}}{\Gamma_T + \{p(a)\} + \emptyset \vdash p(a)}\,(\forall\textbf{-L-G})}{\Gamma_T + \emptyset + \emptyset \vdash p(a)}\,\textbf{(Cut)}$$

Fig. 1: *A coinductive proof in **CLJ** with cut.*

programmers who need to reason about richer coinductive properties than CoLP already handles, or the developers of other theorem provers that feature coinduction.

We can illustrate this paper's results by means of three examples.

Cut Non-Eliminability. Consider the following logic program Γ_T:

$$\kappa_u : \forall x.\, p(f(x)) \to p(x)\,,$$

and the goal formula $p(a)$ for some constant a. We may attempt to prove $p(a)$ by means of an infinite tree that follows the rules of the system **LJ**:

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma_T \vdash p(f(a))}\,(\forall\textbf{-L}) \qquad \dfrac{}{p(a) \equiv p(a)}\,\textbf{(Axiom)}}{\Gamma_T, p(f(a)) \to p(a) \;\vdash p(a)}\,(\to\textbf{-L})}{\Gamma_T, \Gamma_T \vdash p(a)}\,(\forall\textbf{-L})}{\Gamma_T \vdash p(a)}\,\textbf{(C-L)}$$

In fact, $p(a)$ is not directly (analytically) provable in **LJ**. However, if we proved the lemma $\forall x.\, p(x)$, we could derive $p(a)$ as an instance. Such a proof for $p(a)$ in our system **CLJ** is shown in Figure 1. Sequents in **CLJ** have contexts that consist of three parts that are separated by "+": the logic program Γ_T, a context with ordinary proof assumptions (see the application of the rule **(→-L-T)**), and one which holds coinduction hypotheses (see the application of the rule **(CO-FIX)**). This splitting of contexts allows us to ensure *guardedness*, and therefore soundness of coinductive proofs. The proof proceeds by introducing $\forall x.\, p(x)$ through the cut rule into the proof of $p(a)$ in the lower part of Figure 1. We then proceed to prove $\forall x.\, p(x)$ by using the **(CO-FIX)**-rule, and we therefore call this formula a coinduction hypothesis.

In Section 3, we will use this example to prove cut non-eliminability in **CLJ**. That is, we will show that it is impossible to give a cut-free proof for $\Gamma_T + \emptyset + \emptyset \vdash p(a)$. It is worth noting that coinductive inference for $p(a)$ also cannot be accomplished in CoLP (Gupta et al. 2007), and this logic program has been used to show incompleteness of CoLP.

https://doi.org/10.1017/S1471068420000423 Published online by Cambridge University Press

Understanding the Proof-Theoretic Power of CoLP. Looking with proof-theoretic spectacles at CoLP, we notice that CoLP requires circular unifiers seen as fixpoint terms to represent rational terms but does not require the cut rule. For example, consider the logic program $P_{\mathbf{stream}0}$ that defines the stream of zeros:

$$\kappa_{\mathbf{stream}0} : \forall x.\, \mathbf{stream}(x) \to \mathbf{stream}(\mathrm{scons}(0, x))$$

CoLP finds a loop in the resolution trace $\mathbf{stream}(x) \overset{x/\mathrm{scons}(0,x)}{\to} \mathbf{stream}(x) \to \ldots$, and generates a circular unifier $x = \mathrm{scons}(0, x)$ as a finitary representation of the stream. The Prolog query $\mathbf{stream}(x)$ corresponds to the goal $\exists t.\ \mathbf{stream}(t)$ in **CLJ**. In order to obtain a proof for $P_{\mathbf{stream}} + \emptyset + \emptyset \vdash \exists t.\ \mathbf{stream}(t)$ in **CLJ**, we will need to instantiate the existential variable t with the term $s := \mathrm{fix}\, x.\, \mathrm{scons}(0, x)$. Note the use of a fixpoint at the term level as an alternative representation for circular unifiers. We can then prove $P_{\mathbf{stream}} + \emptyset + \emptyset \vdash \mathbf{stream}(s)$ by **(CO-FIX)** with $\mathbf{stream}(s)$ as coinduction hypothesis. More generally, all CoLP proofs yield cut-free proofs in **CLJ**, as we will show in Section 4.

Going Beyond State of the Art. The above results allow us to look at the picture more generally, and notice that proofs of some propositions in coinductive first-order Horn clause theories in fact require proving coinduction lemmas that are formulated in a richer language. Already in our simple example, $\forall x.\, p(x)$ is a goal in hereditary Harrop logic, rather than Horn clause logic because universal goals cannot be proven in Prolog. One can find examples when *higher-order* coinductive lemmas are needed to complete proofs arising from logic programs. Take, for example, the logic program $P_{\mathbf{from}}$ that defines streams of successive natural numbers, e.g., $0, s(0), s(s(0)), \cdots$:

$$\kappa_{\mathbf{from}} : \forall x\, y.\, \mathbf{from}(s(x), y) \to \mathbf{from}(x, \mathrm{scons}(x, y))$$

To prove the goal $\exists t.\, \mathbf{from}(0, t)$, we have to find a finitary representation of the (infinite) term $\mathrm{scons}(0, \mathrm{scons}(s(0), \cdots)$. This is not possible with circular unifiers, but rather with *higher-order* fixpoint terms. Moreover, we also have to generalise our goal, which leads to the coinduction lemma $\forall x.\, \mathbf{from}(x, \mathrm{fix}\, f.\, \lambda x.\, \mathrm{scons}(x, f(s\, x)))$. From this lemma, we are able to obtain $\exists t.\, \mathbf{from}(0, t)$ as a corollary.

In order to prove lemmas at this level of generality, one could use λ-Prolog (Miller and Nadathur 2012) that features both higher-order terms and hereditary Harrop clauses. CUP (Basold et al. 2019) shows that a coinductive extension of λ-Prolog is sound relative to the greatest Herbrand models. However, CUP itself has no capacity to *search* for lemmas that can serve as coinduction hypotheses, it can only *prove* one correct if it is already found. In Section 5, we contribute several theory exploration techniques. Coinductive theory exloration for the example Γ_T from above has already been introduced in Fu et al. (2016). Our implementation incorporates this method, the CoLP-style search for fixpoint terms, and one novel extension that also searches for higher-order coinduction hypotheses, as required for the example $P_{\mathbf{from}}$.

2 Background: Fixpoint Terms and Horn Clause theories

We will only work with first-order Horn clause theories in this paper. However, in presence of coinduction, even these theories may require formulae with higher-order fixpoint terms,

https://doi.org/10.1017/S1471068420000423 Published online by Cambridge University Press

$$\frac{c : \tau \in \Sigma}{\Gamma \vdash c : \tau} \qquad \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \qquad \frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M\, N : \tau}$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x.\, M : \sigma \to \tau} \qquad \frac{\Gamma, x : \tau \vdash M : \tau}{\Gamma \vdash \mathrm{fix}\, x.\, M : \tau}$$

$$\frac{(p : \tau_1 \to \cdots \to \tau_n \to o) \in \Pi \qquad \Gamma \vdash M_1 : \tau_1 \qquad \cdots \qquad \Gamma \vdash M_n : \tau_n}{\Gamma \Vdash p\, M_1 \cdots M_n}$$

$$\frac{\Gamma \Vdash \varphi \qquad \Gamma \Vdash \psi \qquad \square \in \{\wedge, \vee, \to\}}{\Gamma \Vdash \varphi \square \psi} \qquad \frac{\Gamma, x : \tau \Vdash \varphi}{\Gamma \Vdash \forall x : \tau.\, \varphi} \qquad \frac{\Gamma, x : \tau \Vdash \varphi}{\Gamma \Vdash \exists x : \tau.\, \varphi}$$

Fig. 2: ***Top:*** *Well-formed Terms.* ***Bottom:*** *Well-formed Formulae.*

as we saw in the introduction. This motivates the use of simply typed λ- and fixpoint-terms (Barendregt et al. 2013; Basold et al. 2019). For Horn clause theory definitions, we follow closely the notation used in Uniform proofs (Miller and Nadathur 2012).

We define the sets $\mathbb{T}$ of *types* and $\mathbb{P}$ of *proposition types* by the following grammars, where ι and o are the *base type* and *base proposition type*.

$$\mathbb{T} \ni \sigma, \tau ::= \iota \mid \sigma \to \tau \qquad\qquad \mathbb{P} \ni \rho ::= o \mid \sigma \to \rho, \quad \sigma \in \mathbb{T}$$

A *term signature* Σ is a set of pairs $c : \tau$, where $\tau \in \mathbb{T}$, and a *predicate signature* is a set Π of pairs $p : \rho$ with $\rho \in \mathbb{P}$. The elements in Σ and Π are called *term symbols* and *predicate symbols*, respectively. Given term and predicate signatures Σ and Π, we refer to the pair (Σ, Π) as *signature*. Let Var be a countable set of variables, the elements of which we denote by $x, y, \ldots$ We call a finite list Γ of pairs $x : \tau$ of variables and types a *context*. The set Λ_Σ of *(well-typed) terms* over Σ is the collection of all M with $\Gamma \vdash M : \tau$ for some context Γ and type $\tau \in \mathbb{T}$, where $\Gamma \vdash M : \tau$ is defined in Figure 2. A term is called *closed* if $\vdash M : \tau$, otherwise it is called *open*. We say that φ is a *(well-formed) formula* in context Γ, if $\Gamma \Vdash \varphi$ is inductively derivable from the rules in Figure 2.

It is customary in logic programming to write the arguments to symbols as tuples like, for example, in $f(t_1, t_2)$. Our definition uses juxtaposition instead for simplicity, that is, we would write this term as $f\, t_1\, t_2$. Throughout this paper, we will, however, often employ the logic programming style for the benefit of the reader.

We will use a standard β- and fix-reduction relation on terms, see (Basold et al. 2019). The equivalence closure of the reduction relation (convertibility) is denoted by $\equiv$.

The *order* of a type $\tau \in \mathbb{T}$ is given as usual by $\mathrm{ord}(\iota) = 0$ and $\mathrm{ord}(\sigma \to \tau) = \max\{\mathrm{ord}(\sigma) + 1, \mathrm{ord}(\tau)\}$. If $\mathrm{ord}(\tau) \leq 1$, then the arity of τ is given by $\mathrm{ar}(\iota) = 0$ and $\mathrm{ar}(\iota \to \tau) = \mathrm{ar}(\tau) + 1$. A signature Σ is called *first-order*, if for all $f : \tau \in \Sigma$ we have $\mathrm{ord}(\tau) \leq 1$; similarly for Π. We let the arity of f then be $\mathrm{ar}(\tau)$ and denote it by $\mathrm{ar}(f)$.

The *guarded base terms* over a first-order signature Σ are given by the following rules.

$$\frac{x : \tau \in \Gamma \qquad \mathrm{ord}(\tau) \leq 1}{\Gamma \vdash_g x : \tau} \qquad \frac{f : \tau \in \Sigma}{\Gamma \vdash_g f : \tau} \qquad \frac{\Gamma \vdash_g M : \sigma \to \tau \qquad \Gamma \vdash_g N : \sigma}{\Gamma \vdash_g M\, N : \tau}$$

$$\frac{f : \sigma \in \Sigma \qquad \mathrm{ord}(\tau) \leq 1 \qquad \Gamma, x : \tau, y_1 : \iota, \ldots, y_{\mathrm{ar}(\tau)} : \iota \vdash_g M_i : \iota \qquad 1 \leq i \leq \mathrm{ar}(f)}{\Gamma \vdash_g \mathrm{fix}\, x.\, \lambda \vec{y}.\, f\, \vec{M} : \tau}$$

General *guarded terms* are generated by the following grammar.

$$G ::= M \text{ (with } \vdash_g M : \tau \text{ for some type } \tau) \mid c \in \Sigma \mid x \in \mathrm{Var} \mid G\, G \mid \lambda x.\, G$$

https://doi.org/10.1017/S1471068420000423 Published online by Cambridge University Press

Finally, M is a *first-order* term over Σ with $\Gamma \vdash M : \tau$ if $\mathrm{ord}(\tau) \leq 1$ and the types of all variables occurring in Γ are of order 0.

Note that an important aspect of guarded terms is that no free variable occurs under a fix-operator. *Guarded base terms* should be seen as specific fixpoint terms that we will be able to unfold into potentially infinite trees. *Guarded terms* close guarded base terms under operations of the simply typed λ-calculus. Basold et al. (2019) provides examples and further discussion of guarded terms. In what follows, we will use the following sets of well-typed terms: the set Λ^{-}_{Σ} of all *simple terms*, i.e. terms that do not involve fix; the set $\Lambda^{G,1}_{\Sigma}$ of guarded first-order terms; the set $\Lambda^{-,1}_{\Sigma}$ of simple first-order terms.

Definition 2.1 (*Atoms*)
A formula φ of the shape $p\, M_1 \cdots M_n$ is an *atom* and a

- *first-order atom*, if p and all the terms M_i are first-order;
- *guarded atom*, if all terms M_i are guarded; and
- *simple atom*, if all terms M_i are simple.

The sets of first-order, guarded and simple atoms are denoted by At_1, At^g_ω and At^s_ω. We denote intersections of these sets by $\mathrm{At}^g_1 = \mathrm{At}_1 \cap \mathrm{At}^g_\omega$ and $\mathrm{At}^s_1 = \mathrm{At}_1 \cap \mathrm{At}^s_\omega$.

Definition 2.2 (*D- and G-formulae, Logic Programs, Coinduction Hypothesis*)
Let D and G be generated by the following grammar.

$$D ::= \mathrm{At}^g_\omega \mid G \to D \mid D \wedge D \mid \forall x : \tau.\, D$$
$$G ::= \mathrm{At}^g_\omega \mid G \wedge G \mid G \vee G \mid \exists x : \tau.\, G \mid D \to G \mid \forall x : \tau.\, G$$

A D-formula of the shape $\forall \vec{x}.\, A_1 \wedge \cdots \wedge A_n \to A_0$ is called *H-formula* or *Horn clause* if $A_k \in \mathrm{At}^s_1$. Finally, a *logic program* (or *program*) P is a set of H-formulae.

A formula φ is a *coinduction hypothesis* if φ simultaneously is a D- and a G-formula.

D- and G-formulae are also known as *definite clauses* and *goal clauses* in the logic programming literature. The above syntax of D and G-formulae in fact presents an extension of Horn clause syntax to hereditary Harrop formulae (that allow universal and implicative goals). Coming back to our running example of Γ_T formulated in Section 1, we see that Γ_T was given by a Horn clause. However, the proof of a goal $p(a)$ required to prove $\forall x.\, p(x)$ first, which is a goal of hereditary Harrop logic.

3 Coinductive Sequent Calculus CLJ; Proof of Cut Non-Elimination

We start with introducing **CLJ**, a coinductive dialect of the Gentzen's intuitionistic sequent calculus **LJ** (Gentzen 1969). The rules in Figure 3 follow the standard formulation of **LJ** (Sorensen and Urzyczyn 2006) (including notation Γ, ψ for $\Gamma \cup \{\psi\}$), except for the following three differences. Firstly, we restrict ourselves to logic programs for Γ_T, and we allow only G-formulae in Γ_A and Γ_C. As a result, we omit some **LJ** rules for existential and disjunctive formulae on the left. Secondly, we introduce the rule **(CO-FIX)** in its standard formulation, see e.g. (Basold et al. 2019). Finally, we ensure guardedness of coinduction in **CLJ** by splitting the context into logic programs Γ_T, intermediate proof assumptions Γ_A, and coinduction assumptions Γ_C. Applying the rule **(CO-FIX)** is the only way of introducing a coinduction assumption in Γ_C. But, to complete a proof that

https://doi.org/10.1017/S1471068420000423 Published online by Cambridge University Press

$$\frac{\varphi' \in \Gamma_T \cup \Gamma_A \qquad \varphi \equiv \varphi'}{\Gamma_T + \Gamma_A + \Gamma_C \vdash \varphi}\ \textbf{(Axiom)} \qquad \frac{\Gamma_T + \Gamma_A + \Gamma_C \vdash \varphi_1 \qquad \Gamma_T + \Gamma_A + \Gamma_C \vdash \varphi_2}{\Gamma_T + \Gamma_A + \Gamma_C \vdash \varphi_1 \wedge \varphi_2}\ (\wedge\textbf{-R})$$

$$\frac{\Gamma_T, \psi_i + \Gamma_A + \Gamma_C \vdash \varphi \qquad i \in \{1,2\}}{\Gamma_T, \psi_1 \wedge \psi_2 + \Gamma_A + \Gamma_C \vdash \varphi}\ (\wedge\textbf{-L-T}) \qquad \frac{\Gamma_T + \Gamma_A, \psi_i + \Gamma_C \vdash \varphi \qquad i \in \{1,2\}}{\Gamma_T + \Gamma_A, \psi_1 \wedge \psi_2 + \Gamma_C \vdash \varphi}\ (\wedge\textbf{-L-G})$$

$$\frac{\Gamma_T + \Gamma_A + \Gamma_C \vdash \varphi \qquad x \notin FV(\Gamma_T \cup \Gamma_A \cup \Gamma_C)}{\Gamma_T + \Gamma_A + \Gamma_C \vdash \forall x.\, \varphi}\ (\forall\textbf{-R}) \qquad \frac{\Gamma_T + \Gamma_A + \Gamma_C \vdash \varphi\,[N/x]}{\Gamma_T + \Gamma_A + \Gamma_C \vdash \exists x.\, \varphi}\ (\exists\textbf{-R})$$

$$\frac{\Gamma_T, \psi\,[N/x] + \Gamma_A + \Gamma_C \vdash \varphi}{\Gamma_T, \forall x.\, \psi + \Gamma_A + \Gamma_C \vdash \varphi}\ (\forall\textbf{-L-T}) \qquad \frac{\Gamma_T + \Gamma_A, \psi\,[N/x] + \Gamma_C \vdash \varphi}{\Gamma_T + \Gamma_A, \forall x.\, \psi + \Gamma_C \vdash \varphi}\ (\forall\textbf{-L-G})$$

$$\frac{\Gamma_T + \Gamma_A, \psi + \Gamma_C \vdash \varphi}{\Gamma_T + \Gamma_A + \Gamma_C \vdash \psi \to \varphi}\ (\to\textbf{-R}) \qquad \frac{\Gamma_T, \psi + \Gamma_A + \Gamma_C \vdash \varphi \qquad \Gamma_T + \Gamma_A, \Gamma_C + \emptyset \vdash \xi}{\Gamma_T, \xi \to \psi + \Gamma_A + \Gamma_C \vdash \varphi}\ (\to\textbf{-L-T})$$

$$\frac{\Gamma_T + \Gamma_A, \psi + \Gamma_C \vdash \varphi \qquad \Gamma_T + \Gamma_A + \Gamma_C \vdash \xi}{\Gamma_T + \Gamma_A, \xi \to \psi + \Gamma_C \vdash \varphi}\ (\to\textbf{-L-G})$$

$$\frac{\Gamma_T + \Gamma_A + \Gamma_C, \varphi \vdash \varphi}{\Gamma_T + \Gamma_A + \Gamma_C \vdash \varphi}\ \textbf{(CO-FIX)} \qquad \frac{\Gamma_T + \Gamma_A + \Gamma_C \vdash \psi \qquad \Gamma_T + \Gamma_A, \psi + \Gamma_C \vdash \varphi}{\Gamma_T + \Gamma_A + \Gamma_C \vdash \varphi}\ \textbf{(Cut)}$$

Fig. 3: *The rules for **CLJ**, standard structural rules are assumed.*

starts with **(CO-FIX)**, we can never use formulae from Γ_C. The only rule that allows us to shift the coinduction hypotheses from Γ_C to Γ_A and thus make them usable in proofs is the rule **(→-L-T)**. Inuitively, this means we can only use a coinduction assumption after we "resolved" our current goal against some clause from Γ_T.

For this section only, it is sufficient to take a much smaller fragment of **CLJ**, and restrict ourselves to only simple first-order atoms in Γ_A and Γ_C. In later sections, it will be made clear how and why higher-order and fixpoint terms can be useful.

LJ has four structural rules: weakening, exchange and contraction on the left, and weakening on the right. We omit the latter, as we extend the **(Axiom)** rule in a way that renders right weakening opaque. To mimic **LJ**, we need to add the remaining three structural rules **(WL-T)**, **(XL-T)** and **(CL-T)** for Γ_T:

$$\frac{\Gamma_T + \Gamma_A + \Gamma_C \vdash \varphi}{\Gamma_T, \psi + \Gamma_A + \Gamma_C \vdash \varphi} \qquad \frac{\Gamma_T, \phi, \psi, \Gamma'_T + \Gamma_A + \Gamma_C \vdash \varphi}{\Gamma_T, \psi, \phi, \Gamma'_T + \Gamma_A + \Gamma_C \vdash \varphi} \qquad \frac{\Gamma_T, \psi, \psi + \Gamma_A + \Gamma_C \vdash \varphi}{\Gamma_T, \psi + \Gamma_A + \Gamma_C \vdash \varphi}$$

and similarly for Γ_A. We assume these 6 rules additionally to those in Figure 3.

We do not state soundness of **CLJ** here, as soundness of a very similar proof system CUP relative to the greatest Herbrand models of logic programs was already proven in (Basold et al. 2019). Here, our main goal is to prove cut non-elimnation in **CLJ**. We use the example of Section 1 to show this.

Theorem 3.1 (*Cut is not eliminable in **CLJ***)
Any proof of $\{\forall x.\, p(f(x)) \to p(x)\} + \emptyset + \emptyset \vdash p(a)$ uses the **(Cut)** rule.

Proof. To prove the theorem we will construct a set $\mathcal{S}$ of *bad sequents* in a proof tree for $\{\forall x.\, p(f(x)) \to p(x)\} + \emptyset + \emptyset \vdash p(a)$, such that the following conditions hold:

1. The rule **(Axiom)** does not belong to $\mathcal{S}$;
2. For every instance of any rule except **(Cut)**, if the conclusion belongs to $\mathcal{S}$ then at least one premise belongs to $\mathcal{S}$;
3. Sequent $\{\forall x.\, p(f(x)) \to p(x)\} + \emptyset + \emptyset \vdash p(a)$ belongs to $\mathcal{S}$.

https://doi.org/10.1017/S1471068420000423 Published online by Cambridge University Press

If these three conditions hold, then there are no finite proofs without cut for any sequent in $\mathcal{S}$, including the sequent from the theorem statement.

Let us now construct $\mathcal{S}$. It consists of sequents of the form $\Gamma_T + \Gamma_A + \Gamma_C \vdash p(N)$ (with an arbitrary term $N \in \Lambda_\Sigma^{-,1}$) such that:

- $\Gamma_T \subseteq \{p(t) \mid t \in \Lambda_\Sigma^{-,1} : t \neq f^i(N) \ \ \forall i \geq 0\} \cup \{\forall x.\, p(f(x)) \to p(x)\} \cup \{p(f(t)) \to p(t) \mid t \in \Lambda_\Sigma^{-,1}\}$,
- $\Gamma_A \subseteq \{p(t) \mid t \in \Lambda_\Sigma^{-,1} : t \neq f^i(N) \ \ \forall i \geq 0\}$,
- $\Gamma_C \subseteq \{p(t) \mid t \in \Lambda_\Sigma^{-,1} : t \neq f^i(N) \ \ \forall i > 0\}$.

So, we allow in premises only formulae of the form $p(t)$ with t different from N with f applied any number of times, we also allow succedent in the set of unguarded premises (note $>$ instead of $\geq$ there) and the given clause $\forall x.\, p(f(x)) \to p(x)$ in the set of theory assumptions (uninstatiated or instantiated with an arbitrary term).

We now only need to check that the conditions for a set of *bad sequents* hold.

(1) Obvious, as we explicitly forbade the succedent from the guarded assumptions.

(2) There are very few rules except **(Cut)** that we can apply to a sequent of this form. We can apply **(∀-L-T)**, **(CO-FIX)** or the structural rules, which will keep us in $\mathcal{S}$ simply by its definition. The only non-trivial case is if we apply the **(→-L-T)**-rule to use an assumption $p(f(M)) \to p(M)$ with some term M. We will consider two subcases here:

(2.1) $M \neq f^i(N)$ for all $i \geq 0$. Then the premise

$$\Gamma_T, p(M) + \Gamma_A + \Gamma_C \vdash p(N)$$

belongs to $\mathcal{S}$, as in this subcase $p(M)$ satisfies the condition for assumptions from Γ_T.

(2.2) $M = f^k(N)$ for some $k \geq 0$. Then we can show that the other premise

$$\Gamma_T + \Gamma_A, \Gamma_C + \emptyset \vdash p(f(M))$$

belongs to $\mathcal{S}$. We can rewrite it as

$$\Gamma_T + \Gamma_A, \Gamma_C + \emptyset \vdash p(f^{k+1}(N)).$$

As all assumptions of the form $p(t)$ safisfy $t \neq f^i(N) \ \ \forall i > 0$, because the conclusion belongs to $\mathcal{S}$, they therefore satisfy $t \neq f^{i+k+1}(N) \ \ \forall i \geq 0$.

(3) Obvious. □

Note that, because of its simplicity, this result will be replicable in many sequent calculi like, for instance, the classical system **LK** (Troelstra and Schwichtenberg 2000; Sorensen and Urzyczyn 2006).

4 CoLP Derivations as Cut-free Proofs

Intuitively, the loop detection method of CoLP (Gupta et al. 2007) amounts to finding atoms A and B in an SLD-derivation such that A and B unify. This, possibly circular, unifier gives rise to a possibly infinite atom given by a *rational tree* (Courcelle 1983). It may seem plausible to conjecture that CoLP's set of all provable atoms corresponds to the set of all rational trees in the program's model, but this conjecture is disproven by our example of the logic program Γ_T and the goal $p(a)$, that can be represented by a rational tree, but cannot be proven in CoLP. This section proposes an alternative characterisation of provability in CoLP as a set of atoms provable in cut-free **CLJ**. Providing a different perspective on this result, Dagnino et al. (2020) have recently shown that CoLP covers

https://doi.org/10.1017/S1471068420000423 Published online by Cambridge University Press

all regular infinite SLD-trees. The regular proofs of Dagnino et al. (2020) correspond to finite cut-free **CLJ** proofs in which the coinduction hypothesis/goal encapsulates the structure of the entire infinite regular proof.

To establish our result, we need to allow first-order guarded fixpoint terms in goals and in (coinductive) assumptions in Γ_T, Γ_A and Γ_C. The main technical idea of this section is to show how circular unifiers of CoLP convert into first-order fixpoint terms. This conversion delivers us the theoretical result we seek, and may also open the way for using CoLP within richer coinductive theorem provers.

Substitution σ is a finitely supported function from variables to simple first-order terms (i.e. terms in $\Lambda_\Sigma^{-,1}$). As usual, a substitution σ can be extended to a function from $\Lambda_\Sigma^{-,1}$ to $\Lambda_\Sigma^{-,1}$ by taking $(f\ t_1 \dots t_n)[\sigma] = f\ t_1[\sigma] \dots t_n[\sigma]$, whenever f is a constant in Σ. If σ_1 and σ_2 are substitutions, then their *composition* $\sigma_1 \circ \sigma_2$ is defined by $(\sigma_1 \circ \sigma_2)(x) = \sigma_2(x)[\sigma_1]$. A substitution σ is a *unifier* for $t, u \in \Lambda_\Sigma^{-,1}$, if $t[\sigma] = u[\sigma]$, it is a *matcher* if $t[\sigma] = u$. We say a substitution $\sigma = [t/x]$ is *circular* if x appears among the free variables of t. For example, $[\mathrm{scons}(0, x)/x]$ is a circular substitution.

In order to represent circular substitutions as fixpoint terms, we need to extend the notion of substitution to *fix-substitution*, which is defined as a finitely supported function from variables to guarded first-order terms, i.e. terms in $\Lambda_\Sigma^{G,1}$. We will denote fix-substitutions by $\delta, \delta_0, \delta_1, \dots$ to distinguish them from simple first-order substitutions. Fix-substitutions extend to functions $\Lambda_\Sigma^{G,1} \to \Lambda_\Sigma^{G,1}$ by capture-avoiding substitution.

A fix-substitution δ is a *fixpoint unifier* for $t, u \in \Lambda_\Sigma^{G,1}$, if $t[\delta] \equiv u[\delta]$, where we recall $\equiv$ to be conversion with fix- and β-reduction (see (Basold et al. 2019)).

We first show that, given a circular substitution $\sigma = [f\,\vec{t}/x]$, we can obtain a fix-substitution $\delta = \left[\mathrm{fix}\, x.\, f\,\vec{t}/x\right]$. For example, the circular substitution $[\mathrm{scons}(0, x)/x]$ gives rise to the fix-substitution $[\mathrm{fix}\, x.\, \mathrm{scons}(0, x)/x]$. Finding such substitutions in the general case requires some additional machinery, as the following example shows.

Example 4.1 (*Circular substitutions do not result in circular unifiers*)
For the two atoms $p(f(x, y), g(x, y))$ and $p(x, y)$, let $\sigma_1 = [f(x, y)/x]$ and $\sigma_2 = [g(x, y)/y]$. We would like to define a unifier by $\sigma = \sigma_2 \circ \sigma_1$. However, the composition will result in $\sigma = [f(x, g(x, y))/x,\ g(x, y)/y]$, which is not quite the unifier $[f(x, y)/x,\ g(x, y)/y]$ that we expect. For this reason, the circular substitutions are not composed in CoLP, but are simply taken as sets of equations, like $\{x = f(x, y)\,, y = g(x, y)\}$.

We need a notion of composition for circular substitutions, in order to have proper circular unifiers as part of the language. And this is where we make use of fixpoint terms.

Definition 4.1 (*Unifying equations*)
Given $t, u \in \Lambda_\Sigma^{-,1}$, a set $\mathcal{U}_{t,u}$ of *unifying equations* is defined inductively as follows:

1. if $t = x$ for some $x \in \mathrm{Var}$, then $\mathcal{U}_{t,u} = \{x = u\}$,
2. if $u = x$ for some $x \in \mathrm{Var}$, then $\mathcal{U}_{t,u} = \{x = t\}$,
3. if $t = f\, t_1 \cdots t_n$ and $u = f\, u_1 \cdots u_n$, then $\mathcal{U}_{t,u} = \bigcup_{k=1}^{n} \mathcal{U}_{t_k, u_k}$, and
4. $\mathcal{U}_{t,u} = \emptyset$ otherwise.

Two simple first-order atoms $A = p\, t_1 \cdots t_m$ and $B = p\, u_1 \cdots u_m$ have as set of unifying equations $\mathcal{U}_{A,B} = \bigcup_{k=1}^{m} \mathcal{U}_{t_k, u_k}$.

https://doi.org/10.1017/S1471068420000423 Published online by Cambridge University Press

Clearly, if $\mathcal{U}_{t,u}$ is empty, then t and u are not unifiable. If the set of unifying equations contains at most one equation for each variable, we say that it is *linear unifying*.

The mentioned set $\{x = f(x,y), y = g(x,y)\}$ is linear unifying for $p(f(x,y), g(x,y))$ and $p(x,y)$. We refer an interested reader to (Courcelle 1983; Gupta et al. 2007) for a more detailed study of properties of unifying equations. Notably, every system of such equations has the most general unifier that is rational.

Definition 4.2 (*Circular Unifier*)
Let $A, B \in \mathrm{At}_1^s$ have a set of linear unifying equations $\mathcal{U}_{A,B} = \bigcup_{i=1}^{n}\{x_i = t_i\}$. We can define a sequence of fix-substitutions $\delta_0, \delta_1, \ldots, \delta_n$, such that δ_k unifies the first k equations, as follows:

$$\delta_0 = \mathrm{id}$$

$$\delta_{i+1} = \begin{cases} \delta_i, & \text{if } t_{i+1}[\delta_i] = x_{i+1} \\ [t_{i+1}[\delta_i]/\ x_{i+1}] \circ \delta_i, & \text{if } x_{i+1} \notin FV(t_{i+1}[\delta_i]) \\ [\mathrm{fix}\, x_{i+1}.\, t_{i+1}[\delta_i]/\ x_{i+1}] \circ \delta_i, & \text{if } t_{i+1}[\delta_i] \neq x_{i+1},\ x_{i+1} \in FV(t_{i+1}[\delta_i]) \end{cases}$$

Then the fix-substitution δ_n is called the *circular unifier* for A and B.

Example 4.2 (*Circular Unifiers*)
Given the set $\mathcal{U} = \{x = f(y), y = g(x)\}$ for the atoms $p(f(y), (g(x))$ and $p(x,y)$, the circular unifier will be $\delta = [\mathrm{fix}\, y.\, g(f(y))/y] \circ [f(y)/x]$, which amounts to $[f(\mathrm{fix}\, y.\, g\,(f(y)))/x, \mathrm{fix}\, y.\, g(f(y))/y]$.

We continue with the equations $\{x = f(x,y), y = g(x,y)\}$, and atoms $p\,(f(x,y), g(x,y))$ and $p(x,y)$ from Example 4.1. From Definition 4.2, we obtain the desired circular unifier $[\mathrm{fix}\, y.\, g\,(\mathrm{fix}\, z.\, f(z,y), y)/y] \circ [\mathrm{fix}\, x.\, f(x,y)/x]$, which in turn is equal to the substitution $[\mathrm{fix}\, x.\, f(x, \mathrm{fix}\, y.\, g(\mathrm{fix}\, z.\, f(z,y), y))/\ x, \mathrm{fix}\, y.\, g(\mathrm{fix}\, z.\, f(z,y), y)/\ y]$.

The following lemma shows that circular unifiers are fixpoint unifiers.

Lemma 4.1 (*Circular unifier is a fixpoint unifier*)
Let $A, B \in \mathrm{At}_1^s$ and let σ be their circular unifier. Then, $A[\sigma] \equiv B[\sigma]$.

We can now use circular unifiers to generate coinduction hypotheses.

Example 4.3 (*Coinduction Hypothesis from Circular Unifiers*)
Taking $P_{\mathbf{stream}0}$ and the goal $\mathbf{stream}\,(\mathrm{scons}(0, x'))$, CoLP finds $\{x' = \mathrm{scons}(0, x')\}$ as circular unifier. This corresponds to the coinduction hypothesis $\mathbf{stream}\,(\mathrm{fix}\, x.\, \mathrm{scons}(0, x))$.

Simon et al. (2006) have shown that the method of loop detection is sound relative to the complete Herbrand models of logic programs. CUP, a cut-free fragment of **CLJ** was also shown to be sound relative to the complete Herbrand models(Basold et al. 2019). We only need to show that we form fixpoint terms from loops correctly.

Theorem 4.1 (*CoLP proofs in cut-free **CLJ***)
Let Γ_T be a logic program and $A \in \mathrm{At}_1^s$. If CoLP returns a proof and a circular substitution θ for Γ_T and A that is given by a set $\mathcal{U}$ of linear unifying equations, then:

- there exists a circular unifier δ for $\mathcal{U}$,
- and there is a cut-free proof for $\Gamma_T + \emptyset + \emptyset \vdash \exists \vec{x}.\, A$.

https://doi.org/10.1017/S1471068420000423 Published online by Cambridge University Press

Proof. The first property follows from the construction of Definition 4.2 and Lemma 4.1. The second property is also proven constructively, by constricting a **CLJ** proof in which, as the first step, the existential variables $\vec{x}$ are substituted as in δ, and then the proof for $\Gamma_T + \emptyset + \emptyset \vdash A[\delta]$ proceeds by **(CO-FIX)**, taking $A[\delta]$ as coinduction hypothesis. The proof is completed by following the same resolution steps (emulated by a combination of **(∀-L-T)**, **(∧-L-T)**, **(→-L-T)**, **(Axiom)**) as in the given CoLP derivation, applying the coinduction hypothesis where loop detection was applied by CoLP (using **(Axiom)**). □

Taking, for example, the logic program $P_{\mathbf{stream}0}$ and the input formula $\mathbf{stream}\, x$, and having obtained $\mathbf{stream}\,(\mathrm{fix}\, x.\, \mathrm{scons}(0, x))$ from CoLP's circular unifier, we will be able to prove $P_{\mathbf{stream}0} + \emptyset + \emptyset \vdash \mathbf{stream}\,(\mathrm{fix}\, x.\, \mathrm{scons}(0, x))$ by coinduction.

We provide implementation of the method of turning CoLP-style circular unifiers into **CLJ** (or CUP) proofs[1].

5 Coinductive Theory Exploration and Implementation

Coinductive proofs in first-order logic are, in general, not recursively enumerable. We thus have to resort to smaller, cut-free, fragments of coinductive theories, as in CoLP or CUP, for automated proving. As a consequence, we can only hope for heuristics to find suitable cut formulae (and coinduction hypotheses) in the general case.

We present here a new method of *coinductive theory exploration* for **CLJ**, and provide its implementation.[1] We automate cut-free proof search in **CLJ** (equivalently in CUP). That is, given a logic program P and a goal G, we can (semi)decide whether $P+\emptyset+\emptyset \vdash G$ holds. If the automated search fails, a theory exploration method is invoked. It analyses proof-patterns and in particular loops that arose in the failed proof of G. It generalises this information in a form of a candidate coinduction hypothesis CH. The tool then tries to prove $P + \emptyset + \emptyset \vdash CH$ by coinduction. If the proof fails, CH is discarded. If the proof succeeds, the tool re-attempts to prove $P + CH + \emptyset \vdash G$.

Our implementation incorporates three kinds of methods. Firstly, we benefit from CoLP's method of searching for circular unifiers, whenever such exist. Secondly, we implement the method of Fu et al. (2016) that worked for cases when CH was limited to H-formulae (without fixpoint or λ-terms). Finally, we implement a completely novel heuristic that covers the case when CH is a G-formula with (guarded) higher-order fixpoint terms. This method is restricted to logic programs that define non-periodic streams, such as $P_{\mathbf{from}}$ or the program that defines the stream of Fibonacci numbers in Example 5.2. However, our implementation is done in a modular way and will admit novel heuristics and extensions in the future.

From the technical point of view, our implementation benefits from using S-resolution by Komendantskaya and Li (2017) instead of SLD-resolution, when it comes to exploring recursive proof patterns. S-resolution helps to separate out the term-matching and unification components of computations, by doing term-matching steps eagerly, and unification steps lazily. Figure 4 shows term-matching steps as vertical transitions and unification steps as horizontal transitions. Each vertical block, also called a *rewriting tree*, shows clearly reductions of the stream constructor. This is a useful property, as it helps to see the relation between the constructor and other arguments.

Formally, a *rewriting tree* is defined by a map from a tree domain to At_1^s. For definitions of infinite trees as maps from infinite tree domains see e.g. (Courcelle 1983). We write ω

https://doi.org/10.1017/S1471068420000423 Published online by Cambridge University Press

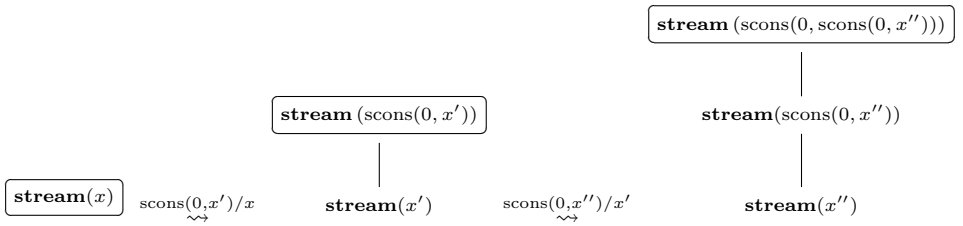

Fig. 4: *Rewriting tree transitions for* $P_{\mathbf{stream}0}$*. Boxes show tree roots.*

for the set of non-negative integers and ω^* for the set of all finite lists over ω. Lists are denoted by $(i, \ldots, j)$ where $i, \ldots, j \in \omega$. The empty list is denoted ϵ. If $u, v \in \omega^*$, then $(u, v) \in \omega^*$ is the concatenation of u and v. If $u \in \omega^*$ and $i \in \omega$, then (u, i) denotes the list $(u, (i))$. Finally, $u > v$ if $u = (v, v')$ for some non-empty v'. A set $L \subseteq \omega^*$ is a *(finitely branching) tree domain* provided:

- $\forall u \in \omega^*. \forall j \in \omega.$ if $(u, j) \in L$ then $u \in L$ and $\forall i < j. (u, i) \in L$; and
- the set $\{i \in \omega \mid (u, i) \in L\}$ is finite for all $u \in L$.

A non-empty tree domain always contains ϵ, which we call its *root*.

Definition 5.1 (*Rewriting tree*)
A rewriting tree for $A \in \mathrm{At}_1^s$ and a logic program P is a map $T\colon L \to \mathrm{At}_1^s$ satisfying:

- $T(\epsilon) = A$, and
- $(u, i) \in L$ and $T(u, i) = B_i[\sigma]$, if there is $(\forall \vec{x}. B_1 \wedge \cdots \wedge B_i \wedge \cdots \wedge B_n \to B) \in P$ and $T(u) = B[\sigma]$. If $n = 0$, we write $T(u, i) = \Box$.

In the above definition, we assume the standard method of *renaming variables apart* used to avoid circular unification.

Given the rewriting tree T (for P and $A \in \mathrm{At}_1^s$), such that some leaf $T(u)$ unifies with the head of a clause in P via a substitution θ, we can construct a rewriting tree T_1 for P and $A[\theta]$. We write $T \overset{\theta}{\rightsquigarrow} T_1$ to denote this tree transition. Figures 4 and 5 show such transitions. We say that a logic program is *productive* (Komendantskaya and Li 2017) if it admits only finite rewriting trees, thus requiring tree transitions for any infinite computation. $P_{\mathbf{stream}0}$ and $P_{\mathbf{from}}$ are productive programs, whereas Γ_T is not. For the rest of this section, we will be working only with productive programs (as all stream definitions give rise to such). Our implementation[1] also covers coinductive theory exploration for infinite rewriting trees, following the method of Fu et al. (2016).

The new heuristic for programs defining non-periodic streams is based on three ideas:

Idea 1: Non-periodic streams can be described by higher-order fixpoint terms. Usually, definitions of non-periodic streams rely on iterating some function that modifies its arguments recursively, and thus computes the stream members that do not unify among each other. In the case of $P_{\mathbf{from}}$, the map s modifies, say, 0 to $s(0)$, $s(s(0))$, and so on. Thus, definitions of such streams involve construction of a fixpoint of a function, rather than of a term variable. We explore this connection between non-periodic stream patterns and higher-order recursive functions.

We assume for the remainder of this section that the goal of our proof is an atom $A \in \mathrm{At}_1^s$ that is built of a predicate that defines some infinite stream, that is,

$$A = p_{\mathbf{stream}}\, t_1^{in} \cdots t_j^{in}\, x^{out}\,,$$

https://doi.org/10.1017/S1471068420000423 Published online by Cambridge University Press

and the program that defines $p_{\mathbf{stream}}$ is productive. Moreover, x^{out} is the *output* argument in the process of computation of streams, the terms $t_1^{in}, \ldots, t_j^{in}$ contain no variables and provide the inputs for the stream construction. For example, in the goal $\mathbf{from}(0, y)$, 0 is the input and y is the output.

We thus exclude programs like P_{double}:

$$\kappa_{double} : \forall x\, y\, z_1\, z_2.\, double(s(x), s(s(y)), z_1, z_2) \to double(x, y, \mathrm{scons}(x, z_1), \mathrm{scons}(y, z_2))$$

that defines two streams of numbers. This restriction is made in order to reduce the notational clutter. The method we present should generalise well to these cases, modulo keeping track of term positions.

Finally, we require that all clauses in the given program are linear, that is, contain at most one recursive call (all examples given so far are linear).

Definition 5.2 (*Higher-order fixpoint stream definition*)
Given a logic program P, and an n-ary predicate $p_{\mathbf{stream}}$ in P that defines a stream s with the function (stream constructor) scons in its last argument, we say s^{fix} given by $\mathrm{fix}\, f.\, \lambda x_1 \cdots x_{n-1}.\, \mathrm{scons}\, x_1\, (f\, t_1^? \cdots t_{n-1}^?)$ is a *higher-order fixpoint definition of* s if there exist $t_1^?, \ldots, t_{n-1}^? \in \mathrm{At}_1^s$ such that

$$P + \emptyset + \emptyset \vdash \forall x_1 \cdots x_{n-1}.\, p_{\mathbf{stream}}\, x_1 \cdots x_{n-1}\, s^{fix}.$$

In this case we call $\forall x_1 \cdots x_{n-1}.\, p_{\mathbf{stream}}\, x_1 \cdots x_{n-1}\, s^{fix}$ the *candidate coinduction hypothesis* for P and $p_{\mathbf{stream}}$.

We can now see that coinductive theory exploration for higher-order fixpoint stream definitions amounts to search for suitable $t_1^?, \ldots, t_{n-1}^? \in \Lambda_\Sigma^{-,1}$; these terms contain the functions that will be iterated by fix. We next define a possible heuristic for this search.

Idea 2: Resolution by term matching helps to find and analyse irregular recursive proof patterns. This idea has been explored in detail by Fu et al. (2016) in the context of infinite rewriting trees. We follow that line of work and use the Paterson condition to find irregular recursive patterns in rewriting trees:

Definition 5.3 (*Paterson Condition (Sulzmann et al. 2007)*)
Let $\Sigma(A)$, $\mathrm{FVar}(A)$ denote the multiset of term symbols and the multiset of free variables in A. The *Paterson condition* is satisfied by an H-formula $\forall \vec{x}.\, (B_1 \wedge \cdots \wedge B_n \to A)$ if $(\Sigma(B_i) \cup \mathrm{FVar}(B_i)) \subset (\Sigma(A) \cup \mathrm{FVar}(A))$ for each B_i. The pair of simple first-order atoms $\langle A, B \rangle$ is called a *critical pair*, if $\forall \vec{x}.\, B \to A$ does not satisfy the Paterson condition.

Irregular proof traces usually give rise to critical pairs. To use this fact, we say a rewriting tree T is an *irregular rewriting tree* if, each leaf $T(u)$ is either a $\Box$ or forms a critical pair $\langle T(\epsilon), T(u) \rangle$ with the root $T(\epsilon)$. In Figure 5, the second tree is an irregular rewriting tree, but Figure 4 has none.

Idea 3: We need anti-unification to turn irregular recursive patterns into higher-order fixpoints. As Figure 5 shows, just having irregular rewriting trees does not solve the problem of finding higher-order stream definitions. Given a sequence of rewriting tree transitions, we need to be able to abstract from concrete constants to general recursive

https://doi.org/10.1017/S1471068420000423 Published online by Cambridge University Press

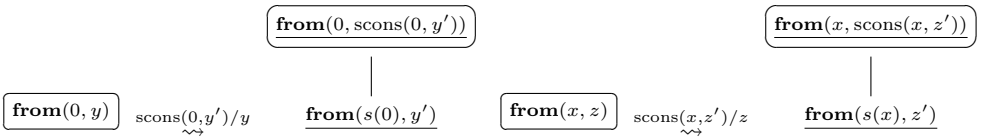

Fig. 5: *Left: Transition of two rewriting trees for the goal formula* **from**$(0, y)$. *Underlined are the critical pairs. Right: abstract representation of the second tree on the left, and a transition for the abstract tree.*

patterns. We implement our own version of the algorithm of anti-unification by (Plotkin 1970) to obtain abstract representations of rewriting trees.

Let $M, N \in \Lambda_{\Sigma}^{-,1}$ be two simple first-order terms, possibly with free variables. We write $M \leq N$ if there is a substitution σ, such that $M[\sigma] = N$. A term A is a *generalisation* of M and N, if $A \leq M$ and $A \leq N$. The following lemma shows that the order and term generalisation are sensible:

Lemma 5.1
The order $\leq$ makes $\Lambda_{\Sigma}^{-,1}$ a poset. Moreover, for any two terms M and N, the set $\{A \in \Lambda_{\Sigma}^{-,1} \mid A \text{ generalises } M \text{ and } N\}$ is filtered, that is, for all generalisations A and B there is a generalisation C with $A \leq C$ and $B \leq C$.

Since the set of generalisations is filtered and bounded, there is a maximal generalisation.

Definition 5.4 (*Anti-Unifier (Plotkin 1970)*)
The *anti-unifier* of two terms M and N is the maximal (or least general) generalisation of M and N, and will be denoted by $M \sqcap N$. This extends in the obvious way to the anti-unifier of atoms.

For example, $p(a) \sqcap p(b) = p(x)$.

Definition 5.5 (*Abstract Representation of a Rewriting Tree*)
Let T be a rewriting tree. Suppose that $\langle T(\epsilon), T(v_1)\rangle, \ldots, \langle T(\epsilon), T(v_n)\rangle$ are all critical pairs, where $T(v_1), \ldots, T(v_n)$ are leaves of T. Let us define $A \in \mathrm{At}_1^s$ to be the anti-unifier $T(\epsilon) \sqcap (\prod_{i=1}^{n} T(v_i))$. The *abstract representation* T' of T is defined as:

- $T'(\epsilon) = A$
- $T'(u, i) = B_i[\sigma]$ if $T'(u) = B[\sigma]$ and $(B_1, ..., B_n \to B) \in P$. When $n = 0$, we write $T'(u, i) = \Box$.
- $T'(u)$ is undefined if $u > v_i$ for some $T(v_i)$ $(1 \leq i \leq n)$, i.e. $T'(v_1), \ldots, T'(v_n)$ are leaves of T'.

It is easy to see that there exists an abstract representation for each irregular rewriting tree. In Figure 5 the third tree is the abstract representation of the second tree. It abstracts away from concrete terms to more general recursive patterns. However, it is really the fourth tree obtained by transition from the third tree that is of interest. We formalise the above intuition as follows. When a proof search

- starts with a program P, a goal A and a rewriting tree T for P and A,
- finds an irregular rewriting tree T' and its corresponding abstract tree T''
- and then proceeds constructing tree transitions from T'',

we will say the search is done in an *abstract search domain for P and A*. Figure 5 shows rewriting trees in an abstract search domain for $P_{\mathbf{from}}$ and **from**$(0, y)$.

The next definition uses Ideas 1, 2, 3 to formulate the novel method of search for higher-order fixpoint terms that capture irregular proof patterns:

https://doi.org/10.1017/S1471068420000423 Published online by Cambridge University Press

Definition 5.6 (*Heuristic Search for Coinduction Hypotheses for Irregular Streams*)
Let T be an irregular rewriting tree in an abstract search domain for P, with the root $A = p\,x_1\,\cdots\,x_{n-1}\,t_n$, where p defines a stream s. Let A and a leaf $T(v) = p\,t_1\,\cdots\,t_{n-1}\,x_n$ form a critical pair. Then a candidate higher-order fixpoint definition of s is obtained by taking $t_i^? = t_i$ (as in Definition 5.2).

Example 5.1 (*Candidate Coinduction Hypothesis*)
The final tree in Figure 5 gives rise to a critical pair. Applying Definitions 5.2 and 5.6, we obtain the candidate fixpoint term $f_{\mathbf{from}} := \mathrm{fix}\, f.\, \lambda x.\mathrm{scons}(x, f(s(x))$ and the candidate coinduction hypothesis $\forall x.\, \mathbf{from}(x, f_{\mathbf{from}})$. Then we prove $P_{\mathbf{from}} + \emptyset + \emptyset \vdash \forall x.\, \mathbf{from}(x, f_{\mathbf{from}})$. The original goal, $\exists z.\, \mathbf{from}(0, z)$ is then obtained by an application of **(Cut)**, i.e. we prove $P_{\mathbf{from}} + (\forall x.\, \mathbf{from}(x, f_{\mathbf{from}}) + \emptyset \vdash \exists z.\, \mathbf{from}(0, z)$ by instantiating z with $f_{\mathbf{from}}\, 0$.

Example 5.2 (*More Complex Coinduction Hypotheses*)
Taking the program P_{fib} computing pseudo-Fibonacci sequence

$$\kappa_{fib} : \forall xy.\, fib(y, x + y, z) \to fib(x, y, \mathrm{scons}(x, z))$$

and a goal $\exists z.\, fib(0, 1, z)$, we obtain an abstract representation of a rewriting tree with the root $fib(x, y, \mathrm{scons}(x, z))$, and the leaf $fib(y, (x+y), z)$. The corresponding candidate stream definition f_{fib} is given by $\mathrm{fix}\, f.\, \lambda x\, y.\, \mathrm{scons}(x, f(y, x + y))$, and the coinduction hypothesis is $\forall x\, y.\, fib(x, y, f_{fib})$.

6 Conclusions, Related and Future Work

This paper contributes to previous attempts to give proof-theoretic and constructive interpretation to logic and answer-set programming: (Miller et al. 1991; Miller and Nadathur 2012; Fu and Komendantskaya 2016; Schubert and Urzyczyn 2018; Basold et al. 2019). Here, our goal was two-fold. Firstly, we showed that cut is not eliminable in a coinductive first-order sequent calculus. Secondly, we analysed the current state of the art in coinductive logic programming (given by CoLP) in the proof-theoretic terms, exposing that CoLP derivations in fact correspond to cut-free proofs in **CLJ**. Both of these results led to a conclusion that any further progress in coinductive logic programming is only possible by introducing richer heuristics of coinductive theory exploration. With this in mind, we proposed a composite method, similar to the famous *Boyer-Moore Waterfall Model* (Boyer and Moore 1979), which incorporates automated proofs in **CLJ**, as well as several existing and one novel heuristics searching for suitable coinduction hypotheses. We provided a prototype implementation.[1]

The novel theory exploration heuristic that we provided serves mainly as an illustration of the range of methods (S-resolution, anti-unification, higher-order fixpoint terms) that can be employed in the future for a systematic synthesis of coinduction hypotheses for proofs in Horn clause theories. We hope to investigate further extensions in the future.

Coinduction is now implemented in major theorem provers, like Coq, Agda, Abella, Isabelle/HOL (Blanchette et al. 2017), and term-rewriting systems (Endrullis et al. 2015). The methods we described here will be applicable in many of these. For example, we supply Coq implementation of all our running examples on the implememtation page.[1]

https://doi.org/10.1017/S1471068420000423 Published online by Cambridge University Press

References

Barendregt, H., Dekkers, W., and Statman, R. 2013. *Lambda Calculus with Types*. Cambridge University Press, Cambridge ; New York.

Basold, H., Komendantskaya, E., and Li, Y. 2019. Coinduction in uniform: Foundations for corecursive proof search with horn clauses. In *ESOP 2019*. 783–813.

Blanchette, J. et al. 2017. Foundational nonuniform (co)datatypes for higher-order logic. In *LICS'17*. IEEE Computer Society, 1–12.

Boyer, R. S. and Moore, J. S. 1979. *A Computational Logic*. ACM Monograph Series. Academic Press.

Brotherston, J. and Simpson, A. 2011. Sequent calculi for induction and infinite descent. *JLC 21,* 6, 1177–1216.

Courcelle, B. 1983. Fundamental properties of infinite trees. *TCS 25*, 95–169.

Dagnino, F., Ancona, D., and E.Zucca. 2020. Flexible coinductive logic programming. *TPLP*.

Endrullis, J., Hansen, H. H., Hendriks, D., Polonsky, A., and Silva, A. 2015. A coinductive framework for infinitary rewriting and equational reasoning. In *RTA'15*. 143–159.

Fu, P. and Komendantskaya, E. 2016. Operational semantics of resolution and productivity in Horn clause logic. *Formal Aspects of Computing*.

Fu, P., Komendantskaya, E., Schrijvers, T., and Pond, A. 2016. Proof relevant corecursive resolution. In *FLOPS'16*. Springer, 126–143.

Gentzen, G. 1969. Investigations into logical deduction. In *The Collected Papers of Gerhard Gentzen*, M. Szabo, Ed. Studies in Logic and the Foundations of Mathematics, vol. 55. Elsevier, 68 – 131.

Gupta, G., Bansal, A., Min, R., Simon, L., and Mallya, A. 2007. Coinductive logic programming and its applications. In *Logic Programming*, V. Dahl and I. Niemelä, Eds. Springer Berlin Heidelberg, Berlin, Heidelberg, 27–44.

Kimura, D., Nakazawa, K., Terauchi, T., and Unno, H. 2020. Failure of cut-elimination in cyclic proofs of separation logic. *Computer Software 37*, 39–52.

Komendantskaya, E. and Li, Y. 2017. Productive corecursion in logic programming. *J. TPLP (ICLP'17 post-proc.) 17,* 5-6, 906–923.

Miller, D. and Nadathur, G. 2012. *Programming with Higher-order logic*. Cambridge University Press.

Miller, D., Nadathur, G., Pfenning, F., and Scedrov, A. 1991. *Uniform Proofs as a Foundation for Logic Programming*. Annals of Pure and Applied Logic, vol. 51. Elsevier, 125–157.

Plotkin, G. D. 1970. A note on inductive generalization. *Machine intelligence*.

Saotome, K., Nakazawa, K., and Kimura, D. 2020. Restriction on cut in cyclic proof system for symbolic heaps. In *FLOPS'20*.

Schubert, A. and Urzyczyn, P. 2018. First-order answer set programming as constructive proof search. *Theory Pract. Log. Program. 18,* 3-4, 673–690.

Simon, L., Mallya, A., Bansal, A., and Gupta, G. 2006. Coinductive logic programming. In *ICLP*. 330–345.

Sorensen, M. H. and Urzyczyn, P. 2006. *Lectures on the Curry-Howard Isomorphism*. Studies in Logic. Elsevier.

Sulzmann, M., Duck, G. J., Jones, S. L. P., and Stuckey, P. J. 2007. Understanding functional dependencies via constraint handling rules. *J. Funct. Program. 17,* 1, 83–129.

Troelstra, A. S. and Schwichtenberg, H. 2000. *Basic Proof Theory*, 2nd ed. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge.

https://doi.org/10.1017/S1471068420000423 Published online by Cambridge University Press