Coinductive Soundness of Corecursive Type Class Resolution

František Farka[1,2], Ekaterina Komendantskaya[3], Kevin Hammond[2], and Peng Fu[3]

[1] University of Dundee, Dundee, Scotland
[2] University of St Andrews, St Andrews, Scotland
{ff32,kh8}@st-andrews.ac.uk
[3] Heriot-Watt University, Edinburgh, Scotland
{ek19,pf7}@hw.ac.uk

Abstract. Horn clauses and first-order resolution are commonly used for the implementation of type classes in Haskell. Recently, several corecursive extensions to type class resolution have been proposed, with the common goal of allowing (co)recursive dictionary construction for those cases when resolution does not terminate. This paper shows, for the first time, that corecursive type class resolution and its recent extensions are coinductively sound with respect to the greatest Herbrand models of logic programs and that they are inductively unsound with respect to the least Herbrand models.

Keywords: Resolution, Coinduction, Herbrand models, Type classes

1 Introduction

The type class mechanism is a popular way of implementing ad-hoc polymorphism and overloading in functional languages. It originated in Haskell [16, 7] and has been further developed in dependently typed languages [6, 3]. For example, it is convenient to define equality for all data structures in a uniform way. In Haskell, this is achieved by introducing the equality class Eq:

```
class Eq x where
  eq :: Eq x ⇒ x → x → Bool
```

and then declaring its instances as needed, e.g. for pairs and integers:

```
instance (Eq x, Eq y) ⇒ Eq (x, y) where
  eq (x1, y1) (x2, y2) = eq x1 x2 && eq y1 y2

instance Eq Int where
  eq x y = primtiveIntEq x y
```

Type class resolution is performed by the compiler and involves checking whether the instance declarations are valid. For example, the following function triggers a check that Eq (Int, Int) is a valid instance of the type class Eq:

```
test :: Eq (Int , Int) ⇒ Bool
test = eq (1 ,2) (1 ,2)
```

It is folklore that type class instance resolution resembles SLD-resolution from logic programming. In particular, an alternative view of the type class instance declarations above would be the following two Horn clauses:

Example 1 (Logic program P_{Pair}).

$$\begin{aligned} \kappa_1 :\ & \mathtt{eq}(x),\ \mathtt{eq}(y) && \Rightarrow \mathtt{eq}(\mathtt{pair}(x, y)) \\ \kappa_2 :\ & && \Rightarrow \mathtt{eq}(\mathtt{int}) \end{aligned}$$

For example, given the query ? $\mathtt{eq}(\mathtt{pair}(\mathtt{int}, \mathtt{int}))$, SLD-resolution terminates successfully with the following sequence of inference steps:

$$\mathtt{eq}(\mathtt{pair}(\mathtt{int}, \mathtt{int})) \to_{\kappa_1} \mathtt{eq}(\mathtt{int}), \mathtt{eq}(\mathtt{int}) \to_{\kappa_2} \mathtt{eq}(\mathtt{int}) \to_{\kappa_2} \emptyset$$

A proof witness (dictionary) is constructed by the Haskell compiler: $\kappa_1\kappa_2\kappa_2$. This is internally treated as an executable function.

Despite the apparent similarity of type class syntax and type class resolution to Horn clauses and SLD-resolution they are not exactly the same. On the syntactic level, type class instance declarations correspond to a restricted form of Horn clauses, namely ones that: (i) do not *overlap* (*i.e.* whose heads do not unify); and (ii) do not contain existential variables (*i.e.* variables that occur in the bodies but not in the heads of the clauses). On the algorithmic level, (iii) type class resolution corresponds to SLD-resolution in which unification is restricted to term-matching. Assuming that there is a clause $B_1, \ldots B_n \Rightarrow A'$, a query ? A' can be resolved with the clause only if A can be matched against A', *i.e.* a substitution σ exists such that $A = \sigma A'$. For comparison, SLD-resolution incorporates unifiers, as well as matchers, *i.e.* it proceeds in resolving the above query and clause also in those cases when $\sigma A = \sigma A'$ holds.

These restrictions derive from a desire to guarantee computation of the principal (most general) type in type class inference. Restrictions (i) and (ii) amount to deterministic inference by resolution, in which only one derivation is possible for every query. Restriction (iii) means that no substitution is applied to a query during the inference, *i.e.* we prove the query in an implicitly universally quantified form. It is a common knowledge that, similarly to SLD-resolution, type class resolution is *inductively sound*, *i.e.* it is sound relative to the least Herbrand models of logic programs [12]. Moreover, it is *universally inductively sound*, *i.e.* if a formula A is proven by type class resolution, every ground instance of A is in the least Herbrand model of the given program. In Section 3, we establish for the first time the universal inductive soundness of type class resolution. Unlike SLD-resolution, type class resolution is *inductively incomplete*, *i.e.* it is incomplete relative to least Herbrand models, even for the class of Horn clauses restricted by the conditions (i) and (ii).

Lämmel and Peyton Jones have suggested [11] an extension to type class resolution that accounts for some non-terminating cases of type class resolution. Consider, for example, the following mutually defined data structures:

```
data OddList a    =  OCons a (EvenList a)
data EvenList a   =  Nil | ECons a (OddList a)
```

and the instance declarations that they give rise to in the Eq class:

```
instance (Eq a, Eq (EvenList a)) ⇒  Eq (OddList a) where
    eq (OCons x xs) (OCons y ys) =  eq x y && eq xs ys

instance (Eq a, Eq (OddList a)) ⇒ Eq (EvenList a) where
    eq Nil          Nil          =  True
    eq (ECons x xs) (ECons y ys) =  eq x y && eq xs ys
    eq _            _            =  False
```

The `test` function below triggers type class resolution in the compiler:

```
test :: Eq (EvenList Int) ⇒ Bool
test = eq Nil Nil
```

Such inference by resolution does not terminate. Consider the Horn clause representation of the type class instance declarations:

Example 2 (Logic program $P_{EvenOdd}$).

$$\begin{aligned}
\kappa_1 &: \texttt{eq}(x),\ \texttt{eq}(\texttt{evenList}(x)) &&\Rightarrow \texttt{eq}(\texttt{oddList}(x))\\
\kappa_2 &: \texttt{eq}(x),\ \texttt{eq}(\texttt{oddList}(x)) &&\Rightarrow \texttt{eq}(\texttt{evenList}(x))\\
\kappa_3 &: &&\Rightarrow \texttt{eq}(\texttt{int})
\end{aligned}$$

The non-terminating resolution trace is given by:

$$\begin{aligned}
&\underline{\texttt{eq}(\texttt{evenList}(\texttt{int}))} \to_{\kappa_2} \texttt{eq}(\texttt{int}),\ \texttt{eq}(\texttt{oddList}(\texttt{int})) \to_{\kappa_3} \texttt{eq}(\texttt{oddList}(\texttt{int}))\\
&\qquad \to_{\kappa_1} \texttt{eq}(\texttt{int}),\ \texttt{eq}(\texttt{evenList}(\texttt{int})) \to_{\kappa_3} \underline{\texttt{eq}(\texttt{evenList}(\texttt{int}))} \to_{\kappa_2} \dots
\end{aligned}$$

A goal eq(evenList(int)) is simplified using the clause κ_2 to goals eq(int) and eq(oddList(int)). First of these is discarded using the clause κ_3. The resolution continus by clauses κ_1 and κ_3 resulting in the original goal eq(evenList(int)). It is easy to see that such process can continue infinitely and that this goal constitutes a cycle, which is underlined.

As suggested by Lämmel and Peyton Jones [11], the compiler can terminate the infinite inference process as soon as it detects the underlined cycle. Moreover, it can construct the corresponding proof witness in a form of a recursive function. For the above example, such a function is given by the fixed point term $\nu\alpha.\kappa_2\kappa_3(\kappa_1\kappa_3\alpha)$, where ν is a fixed point operator. The intuitive reading of such a proof is that an infinite proof of the query ? eq (evenList(int)) exists, and that its shape is fully specified by the recursive proof witness function above – the clauses modify goals as described above in the case of the non-terminating resolution trace and $\nu\alpha$ denotes the resolution repeats recursively at the point where α occurs. We will say that the proof is given by *corecursive type class resolution*.

It has not previously been observed in the literature that corecursive type class resolution is not sound inductively. For example, eq(evenList(int)) is not in the least Herbrand model of the corresponding logic program. However, it is *(universally) coinductively sound*, *i.e.* it is sound relative to the greatest Herbrand models. In particular, $\texttt{eq}(\texttt{evenList}(\texttt{int}))$ is in the greatest Herbrand model of the program. We prove this new result in Section 4. Similarly to the inductive case, corecursive type class resolution is coinductively incomplete. Consider the clause $\kappa_{inf} : \texttt{p}(x) \Rightarrow \texttt{p}(\texttt{f}(x))$. It may be given an interpretation by the greatest (complete) Herbrand models, but corecursive type class resolution does not give rise to infinite proofs for this clause.

As might be expected, the simple method of cycle detection used in corecursive type class resolution does not work for all non-terminating programs. Consider the following example, which gives the definition of a data type Bush (for bush trees), and the corresponding instance declaration of equality class:

```
data Bush a = Nil | Cons a (Bush (Bush a))

instance Eq a, Eq (Bush (Bush a)) ⇒ Eq (Bush a) where
          ...
```

Type class resolution for data type Bush does not terminate, but it does not exhibit cycles, either. Consider the Horn clause translation of the problem:

Example 3 (Logic program P_{Bush}).

$$\begin{aligned} \kappa_1 : &\quad \Rightarrow \texttt{eq}(\texttt{int}) \\ \kappa_2 : &\ \texttt{eq}(x),\ \texttt{eq}(\texttt{bush}(\texttt{bush}(x))) \Rightarrow \texttt{eq}(\texttt{bush}(x)) \end{aligned}$$

The derivation below shows that no cycles arise when we resolve the query ? $\texttt{eq}(\texttt{bush}(\texttt{int}))$ against the program P_{Bush}:

$$\texttt{eq}(\texttt{bush}(\texttt{int})) \to_{\kappa_2} \texttt{eq}(\texttt{int}), \texttt{eq}(\texttt{bush}(\texttt{bush}(\texttt{int})) \to_{\kappa_1} \ldots \to_{\kappa_2}$$
$$\texttt{eq}(\texttt{bush}(\texttt{int})), \texttt{eq}(\texttt{bush}(\texttt{bush}(\texttt{bush}(\texttt{int}))) \to_{\kappa_1} \ldots$$

Fu *et al.* [5] have recently introduced an extension to corecursive type class resolution that allows implicative queries to be proved by corecursion and uses the fixed point proof witness construction. For example, in the above program the Horn formula $\texttt{eq}(x) \Rightarrow \texttt{eq}(\texttt{bush}(x))$ can be (coinductively) proven with the recursive proof witness $\kappa_3 = \nu\alpha.\lambda\beta.\kappa_2\beta(\alpha(\alpha\beta))$. If we add this Horn clause as a third clause to our program, we obtain a proof of $\texttt{eq}(\texttt{bush}(\texttt{int}))$ by applying κ_3 to κ_1. For this case, it is even more challenging to understand whether the proof $\kappa_3\kappa_1$ of $\texttt{eq}(\texttt{bush}(\texttt{int}))$ is indeed sound: inductively, coinductively or in any other sense. In Section 5, we establish, for the first time, coinductive soundness for proofs of such implicative queries, relative to the greatest Herbrand models of logic programs. As a consequence, proofs can be obtained by extending the proof context with coinductively proven Horn clauses (e.g. like κ_3 above) are coinductively sound but inductively unsound. This result completes our study of semantic properties of corecursive type class resolution.

Throughout this paper, we will use the formulation of corecursive type class resolution as given by Fu *et al.* [5]. This extends Howard's simply-typed lambda calculus [8, 4] with a resolution rule and a ν-rule. The resulting calculus is general and accounts for all previously suggested kinds of type class resolution.

Contributions of this paper

By presenting the described results, we answer three research questions:

(1) whether type class resolution and its two recent corecursive extensions [5, 11] are sound relative to the standard (Herbrand model) semantics of logic programming;
(2) whether these new extensions are indeed "corecursive", i.e. whether they are better modelled by the greatest Herbrand model semantics rather than by the least Herbrand model semantics; and
(3) whether the context update technique given in [5] can be brought back to logic programming and can be re-used in its corecursive dialects such as CoLP [14] and CoALP [10] or, even broader, can be incorporated into program transformation techniques [2].

We answer questions (1) and (2) in the affirmative. The answer to question (3) is less straightforward. The way the implicative coinductive lemmata are used in proofs alongside all other Horn clauses in [5] indeed resembles a program transformation method when considered from the logic programming point of view. In reality, different fragments of the calculus given in [5] allow proofs for Horn formulae which, when added to the initial program, may lead to inductively or coinductively unsound extensions. We analyse this situation carefully, throughout all of the technical sections that follow, thereby highlighting which program transformation methods can be soundly borrowed from the existing work on corecursive resolution.

2 Preliminaries

This section describes notation and defines the models that we use in the rest of the paper. As is standard, a first-order signature Σ consists of the set $\mathcal{F}$ of function symbols and the set $\mathcal{P}$ of predicate symbols, all symbols equipped with an *arity*. Constants are function symbols of arity 0. We also assume a countable set $\mathcal{V}$ of variables. Given Σ and $\mathcal{V}$, we have the following standard definitions:

Definition 1 (Syntax of Horn formuale and logic programs).

$$\begin{aligned}
\textit{First-order term} \quad & \textit{Term} ::= \mathcal{V} \mid \mathcal{F}(Term, \ldots, Term) \\
\textit{Atomic formula} \quad & \textit{At} ::= \mathcal{P}(Term, \ldots, Term) \\
\textit{Horn formula (clause)} \quad & \textit{CH} ::= \textit{At}, \ldots, \textit{At} \Rightarrow \textit{At} \\
\textit{Logic program} \quad & \textit{Prog} ::= \textit{CH}, \ldots, \textit{CH}
\end{aligned}$$

We use identifiers t and u to denote terms and A, B, C to denote atomic formulae. We use P with indicies to refer to elements of *Prog*. We say that a term or an atomic formula is *ground* if it contains no variables. We assume that all variables in Horn formulae are implicitly universally quantified. Moreover, the restriction (ii) in Section 1 requires that there are no *existential variables*, *i.e.* given a clause

$B_1, \ldots, B_n \Rightarrow A$, if a variable occurs in B_i, then it occurs in A. We use the common name *formula* to refer to both atomic formulae and to Horn formulae. A *substitution* and the *application* of a substitution to a term or a formula are defined in the usual way. We denote application of a substitution σ to a term t or to an atomic formula A by σt and σA respectively. We denote composition of substitutions σ and τ by $\sigma \circ \tau$. A substitution σ is a *grounding* substitution for a term t if σt is a ground term, and similarly for an atomic formula.

2.1 Models of Logic Programs

Throughout this paper, we use the standard definitions of the least and greatest Herbrand models. Given a signature Σ, the *Herbrand universe* $\mathbf{U}_\Sigma$ is the set of all ground terms over Σ. Given a Herbrand universe $\mathbf{U}_\Sigma$ we define the *Herbrand base* $\mathbf{B}_\Sigma$ as the set of all atoms consisting only of ground terms in $\mathbf{U}_\Sigma$.

Definition 2 (Semantic operator). *Let P be a logic program over signature Σ. The mapping $\mathcal{T}_P : 2^{\mathbf{B}_\Sigma} \to 2^{\mathbf{B}_\Sigma}$ is defined as follows. Let I be a subset of $\mathbf{B}_\Sigma$.*

$$\mathcal{T}_P(I) = \{A \in \mathbf{B}_\Sigma \mid B_1, \ldots B_n \Rightarrow A \textit{ is a ground instance of a clause in } P, \\ \textit{and } \{B_1, \ldots, B_n\} \subseteq I\}$$

The operator gives inductive and coinductive interpretation to a logic program:

Definition 3. *Let P be a logic program.*

– *The* least Herbrand model *is the least set $\mathcal{M}_P \in \mathbf{B}_\Sigma$ such that $\mathcal{M}_P$ is a fixed point of $\mathcal{T}_P$.*
– *The* greatest Herbrand model *is the greatest set $\mathcal{M}'_P \in \mathbf{B}_\Sigma$ such that $\mathcal{M}'_P$ is a fixed point of $\mathcal{T}_P$.*

In [12] the operators $\downarrow$ and $\uparrow$ are introduced, $\mathcal{T}_P \downarrow \omega$ is proven to give the greatest Herbrand model of P, and and $\mathcal{T}_P \uparrow \omega$ the least Herbrand model of P. We will use these constructions in our proofs. The validity of a formula in a model is defined as usual. An atomic formula is *valid* in a model I if and only if for any grounding substitution σ, we have $\sigma F \in I$. A Horn formula $B_1, \ldots, B_n \Rightarrow A$ is valid in I if for any substitution σ, if $\sigma B_1, \ldots, \sigma B_n$ are valid in I then σA is valid in I. We use notation $P \vDash_{ind} F$ to denote that a formula F is valid in $\mathcal{M}_P$ and $P \vDash_{coind} F$ to denote that a formula F is valid in $\mathcal{M}'_P$.

Lemma 1. *Let P be a logic program and let σ be a substitution. The following holds:*

a) If $(\ \Rightarrow A) \in P$ then both $P \vDash_{ind} \sigma A$ and $P \vDash_{coind} \sigma A$
b) If, for all i, $P \vDash_{ind} \sigma B_i$ and $(B_1, \ldots, B_n \Rightarrow A) \in P$ then $P \vDash_{ind} \sigma A$
c) If, for all i, $P \vDash_{coind} \sigma B_i$ and $(B_1, \ldots, B_n \Rightarrow A) \in P$ then $P \vDash_{coind} \sigma A$

The proof can be found in the existing literature (*e.g.* [12]) and follows from the fact that both $\mathcal{M}_P$ and $\mathcal{M}'_P$ are fixed points of the operator $\mathcal{T}_P$.

2.2 Proof Relevant Resolution

In [5], the usual syntax of Horn formulae was embedded into a type-theoretic framework, with Horn formulae seen as types inhabited by proof terms. In this setting, a judgement has the form $\Phi \vdash e : F$, where e is a proof term inhabiting formula F, and Φ is an *axiom environment* containing annotated Horn formulae corresponding to the given logic program. This gives rise to the following syntax, in addition to Definition 1. We assume a set of proof term symbols K, and a set of proof term variables U.

Definition 4 (Syntax of proof terms and axiom environments).

$$\begin{array}{rl} \textit{Proof term} & E ::= K \mid U \mid E\ E \mid \lambda U.E \mid \nu U.E \\ \textit{Axiom environment} & Ax ::= \cdot \mid Ax, (E : CH) \end{array}$$

We use notation κ with indicies to refer to elements of K, α and β with indices to refer to elements of U, e to refer to proof terms in E, and Φ to refer to axiom environments in Ax. Having a judgement $\Phi \vdash e : F$, we call F an *axiom* if $e \in K$, and we call F a *lemma* if $e \notin K$ is a closed term, *i.e.* contains no free variables. A proof term e is in *head normal form* (denoted HNF(e)), if $e = \lambda \underline{\alpha}.\kappa\ \underline{e}$ where $\underline{\alpha}$ and $\underline{e}$ denote (possibly empty) sequences of variables $\alpha_1, \ldots, \alpha_n$ and proof terms $e_1 \ldots e_m$ respectively where n and m are know from context or unimportant. The intention of the above definition is to interpret logic programs, seen as sets of Horn formulae, as types. Example 1 shows how proof term symbols κ_1 and κ_2 can be used to annotate clauses in the given logic program. We capture this intuition in the following formal definition:

Definition 5. *Given a logic program P_A consisting of Horn clauses $H_1, \ldots, H_n$, with each H_i having the shape $B^i_1, \ldots, B^i_k \Rightarrow A^i$, the axiom environment Φ_A is defined as follows. We assume proof term symbols $\kappa_1, \ldots, \kappa_n$, and define, for each H_i, $\kappa_i : B^i_1, \ldots, B^i_k \Rightarrow A^i$.*

Revising Example 1 we can say that it shows the result of translation of the program P_{Pair} into Φ_{Pair} and Φ_{Pair} is an axiom environment for the logic program P_{Pair}. In general, we say that Φ_A is an axiom environment for a logic program P_A if and only if there is a translation of P_A into Φ_A. We drop the index A where it is known or unimportant. The restriction (i) in Section 1 requires that axioms in an axiom environment do not overlap. However, a lemma may overlap with other axioms and lemmata—only axioms are subject to the restriction (i). We refer the reader to [5] for complete exposition of proof-relevant resolution. In the following sections, we will use this syntax to gradually introduce inference rules for proof-relevant corecursive resolution. We start with its "inductive" fragment, *i.e.* the fragment that is sound relative to the least Herbrand models, and then in subsequent sections consider its two coinductive extensions (sound relative to the greatest Herbrand models).

3 Inductive Fragment of Type Class Resolution

In this section, we introduce the inductive fragment of the calculus for the extended type class resolution introduced by Fu *et al.* [5]. We reconstruct the standard theorem of universal inductive soundness for the resolution rule. The resolution rule alone was not sufficient for some of the Fu *et al.*'s examples, It was thus extended with a rule that allowed Horn formulae to be proved, *i.e.* to prove lemmata. Both axioms and lemmata could be used as a part of a environment. In logic programming terms, programs were transformed by adding already proven Horn formulae. We prove the soundness of this method relative to the least Herbrand models, and show that it is not sound relative to the greatest Herbrand models.

Definition 6 (Type class resolution).

$$\textit{if } (e : B_1, \ldots, B_n \Rightarrow A) \in \Phi \frac{\Phi \vdash e_1 : \sigma B_1 \quad \cdots \quad \Phi \vdash e_n : \sigma B_n}{\Phi \vdash e\ e_1 \cdots e_n : \sigma A} \qquad (\textsc{Lp-m})$$

If, for a given atomic formula A, and a given environment Φ, $\Phi \vdash e : A$ is derived using the Lp-m rule we say that A is entailed by Φ and that the proof term e witnesses this entailment. We define derivations and derivation trees resulting from applications of the above rule in the standard way (*cf.* Fu *et al.* [5]).

Example 4. Recall the logic program P_{Pair} in Example 1. The inference steps for eq(pair(int, int)) correspond to the following derivation tree:

$$\frac{\dfrac{}{\Phi_{Pair} \vdash \kappa_2 : \texttt{eq(int)}} \qquad \dfrac{}{\Phi_{Pair} \vdash \kappa_2 : \texttt{eq(int)}}}{\Phi_{Pair} \vdash \kappa_1 \kappa_2 \kappa_2 : \texttt{eq(pair(int,int))}}$$

The above entailment is inductively sound, *i.e.* it is sound with respect to the least Herbrand model of P_{Pair}:

Theorem 1. *Let Φ be an axiom environment for a logic program P, and let $\Phi \vdash e : A$ hold. Then $P \vDash_{ind} A$.*

Proof. By structural induction on the derivation tree and construction of the least Herbrand model, using Lemma 1. □

The rule Lp-m also plays a crucial role in the coinductive fragment of type class resolution, as will be discussed in Sections 4 and 5. Now, we turn to discussion of the other rule present in the work of Fu *et al.* [5]. The rule that allows Horn formulae to be proved is:

Definition 7.

$$\frac{\Phi, (\beta_1 : \ \Rightarrow B_1), \ldots, (\beta_n : \ \Rightarrow B_n) \vdash e : A}{\Phi \vdash \lambda \beta_1, \ldots, \beta_n . e : B_1, \ldots, B_n \Rightarrow A} \qquad (\textsc{Lam})$$

Example 5. To illustrate the use of the LAM rule, consider the following program: Let P consist of two clauses: $A \Rightarrow B$ and $B \Rightarrow C$. Both the least and the greatest Herbrand model of P are empty. Equally, no formula can be derived from the corresponding axiom environment by the LP-M rule. However, we can derive $A \Rightarrow C$ by using a combination of the LAM and LP-M rules. Let $\Phi = (\kappa_1 : A \Rightarrow B), (\kappa_2 : B \Rightarrow C)$. The following is then a derivation tree for a formula $A \Rightarrow C$:

$$\dfrac{\dfrac{\dfrac{\overline{\Phi, (\alpha :\ \Rightarrow A) \vdash \alpha : A}}{\Phi, (\alpha :\ \Rightarrow A) \vdash \kappa_1\alpha : B}}{\Phi, (\alpha :\ \Rightarrow A) \vdash \kappa_2(\kappa_1\alpha) : C}}{\Phi \vdash \lambda\alpha.\kappa_2(\kappa_1\alpha) : A \Rightarrow C}\ \textsc{Lam}$$

When there is no label on right-hand side of an inference step, the inference is by the rule LP-M. We follow this convention throughout the paper.

We can show that the calculus comprising the rules LP-M and LAM is again (universally) inductively sound.

Lemma 2. *Let P be a logic program and let A, B_1, ..., B_n be atomic formulae. If $P, (\ \Rightarrow B_1), \ldots, (\ \Rightarrow B_n) \vDash_{ind} A$ then $P \vDash_{ind} B_1, \ldots, B_n \Rightarrow A$.*

Proof. Assume that $P, (\ \Rightarrow B_1), \ldots, (\ \Rightarrow B_n) \vDash_{ind} A$. From the Definition 2 there is the least n such that for any grounding substitution τ, $(\tau \circ \sigma)A \in \mathcal{T}_{P,(\ \Rightarrow B_1),\ldots,(\ \Rightarrow B_n)} \uparrow n$. Consider and substitution σ and suppose that for all i, $P \vDash \sigma B_i$. From the definition of validity for any grounding τ for all i, $(\tau \circ \sigma)B_i \in \mathcal{M}_P$ hence there is the least m such that $(\tau \circ \sigma)B_i \in \mathcal{T}_P \uparrow m$. From the assumption, for any grounding substitution τ also $(\tau \circ \sigma)A \in \mathcal{T}_P \uparrow (n + m)$ and $P \vDash \sigma A$. Hence $P \vDash_{ind} B_1, \ldots, B_n \Rightarrow A$. □

Theorem 2. *Let Φ be an axiom environment for a logic program P and F a formula. Let $\Phi \vdash e : F$ be by the* LP-M *and* LAM *rules. Then $P \vDash_{ind} F$.*

Proof. By structural induction on the derivation tree using Lemmata 1 & 2. □

Related Program Transformation Methods For Fu *et al.* [5], the main purpose of introducing the rule LAM was to increase expressivity of the proof system. In particular, obtaining an entailment $\Phi \vdash e : H$ of a Horn formula H enabled the environment Φ to be extended with $e : H$, which could be used in future proofs. We show that transforming (the standard, untyped) logic programs in this way is inductively sound. The following theorem follows from Lemma 2:

Theorem 3. *Let Φ be an axiom environment for a logic program P, and let $\Phi \vdash e : F$ for a formula F by the* LP-M *and* LAM *rules. Given a formula F', $P \vDash_{ind} F'$ iff $P, F \vDash_{ind} F'$.*

Note, however, that the above theorem is not as trivial as it looks, in particular, it would not hold coinductively, *i.e.* if we changed $\vDash_{ind}$ to $\vDash_{coind}$ in the statement above. Consider the following proof of what seems to be a trivial formula $A \Rightarrow A$:

Example 6. Using the Lam rule one can prove $\emptyset \vdash \lambda\alpha.\alpha : A \Rightarrow A$:

$$\frac{\dfrac{}{(\alpha : \ \Rightarrow A) \vdash \alpha : A}}{\emptyset \vdash \lambda\alpha.\alpha : A \Rightarrow A}\ \textsc{Lam}$$

Indeed, $\mathcal{M}_{\emptyset} = \emptyset$ and by definition of validity, $\emptyset \vDash_{ind} A \Rightarrow A$. Assume a program consisting of a single formula $A \Rightarrow B$. Both the least and the greatest Herbrand model of this program are empty. However, adding the formula $A \Rightarrow A$ to the program results in the greatest Herbrand model $\{A, B\}$. Thus, $\mathcal{M}'_{P} \neq \mathcal{M}'_{P,(A \Rightarrow A)}$.

4 Universal Coinductive Soundness

The Lp-m rule may result in non-terminating resolution. This can be demonstrated by the program $P_{EvenOdd}$ and the query ? eq(evenList(Int)) from Section 1. Lämmel and Peyton Jones observed [11] that in such cases there is a cycle in the inference that can be detected. This treatment of cycles amounts to coinductive reasoning and results in building a corecursive proof witness—or (co-)recursive dictionary.

Definition 8 (Coinductive type class resolution).

$$\textit{if}\ \mathrm{HNF}(e)\frac{\Phi, (\alpha : \ \Rightarrow A) \vdash e : A}{\Phi \vdash \nu\alpha.e : A} \qquad (\textsc{Nu'})$$

The side condition of Nu' requires the proof witness to be in head normal form. Since, in this section, we are working with a calculus consisting of the rules Lp-m and Nu', there is no way to introduce a λ-abstraction into a proof witness. Therefore, in this section, we restrict ourselves to head normal form terms of the form $\kappa\ \underline{e}$.

Example 7. Recall the program $P_{EvenOdd}$ in Example 2. The originally non-terminating resolution trace for the query ? eq(evenList(int)) is resolved using the Nu' rule as follows:

$$\frac{\dfrac{\dfrac{}{\kappa_3 : \mathtt{eq(int)}}}{\vdash \kappa_3 : \mathtt{eq(int)}} \quad \dfrac{\dfrac{\dfrac{}{\kappa_3 : \mathtt{eq(int)}}}{\vdash \kappa_3 : \mathtt{eq(int)}} \quad \dfrac{\dfrac{}{\alpha : \ \Rightarrow \mathtt{eq(evenList(int))}}}{\vdash \alpha : \mathtt{eq(evenList(int))}}}{\Phi_{EvenOdd}, \alpha : \_ \vdash \kappa_1\kappa_3\alpha : \mathtt{eq(oddList(int))}}}{\dfrac{\Phi_{EvenOdd}, \alpha : \_ \vdash \kappa_2\kappa_3(\kappa_1\kappa_3\alpha) : \mathtt{eq(evenList(int))}}{\Phi_{EvenOdd} \vdash \nu\alpha.\kappa_2\kappa_3(\kappa_1\kappa_3\alpha) : \mathtt{eq(evenList(int))}}\ \textsc{Nu'}}$$

Note that we abbreviate formulae in the environment where these repeat by an underscore and we use this notation in the rest of the paper.

We now come to the discussion of the coinductive soundness of the rule Nu', *i.e.* its soundness relative to the greatest Herbrand models. We note that, not surprisingly (*cf.* [13]), the rule Nu' is inductively unsound. Given a program consisting of just one clause: $\kappa : A \Rightarrow A$, we are able to use the rule NU' to entail A (the derivation will be similar, albeit a lot simpler than in the above example). However, A is not in the least Herbrand model of this program. Similarly, the formula `eq(oddList(int))` proven above is not inductively sound, either. Thus, the coinductive fragment of the extended corecursive resolution is only coinductively sound. When proving the coinductive soundness of the NU' rule, we carefully choose the proof method by which we proceed. Inductive soundness of the LP-M rule was proven by induction on the derivation tree and the construction of the least Herbrand models by iterations of $\mathcal{T}_P$. Here, we give an analogous result, where coinductive soundness is proven by structural coinduction on the iterations of the semantic operator $\mathcal{T}_P$.

In order for the principle of structural coinduction to be applicable in our proof, we must ensure that the construction of the greatest Herbrand model is completed within ω steps of iteration of $\mathcal{T}_P$. This does not hold in general for the greatest Herbrand model construction, as was shown e.g. in [12]. However, it does hold for the restricted shape of Horn clauses we are working with. It was noticed by Lloyd [12] that Restriction (ii) from Introduction implies that $\mathcal{T}_P$ operator converges in at most ω steps. We will capitalise on this fact. The essence of coinductive soundness of the rule NU' is captured by the following lemma:

Lemma 3. *Let P be a logic program, let σ be a substitution, and let A, B_1, $\ldots$, B_n be atomic formulae. If, $\forall i \in \{1, \ldots, n\}$, $P, (\;\Rightarrow \sigma A) \vDash_{coind} \sigma B_i$ and $(B_1, \ldots, B_n \Rightarrow A) \in P$ then $P \vDash_{coind} \sigma A$.*

Proof. Consider now the construction of the greatest Herbrand model for the program P and proceed by coinduction with coinductive hypothesis: for all n, for any grounding substitution τ, $(\tau \circ \sigma)A \in \mathcal{T}_P \downarrow n$. Since, for any τ, $(\tau \circ \sigma)A \in \mathcal{T}_P \downarrow n$ the set $\mathcal{T}_P \downarrow n$ is by definition of the operator $\mathcal{T}_P$ the same as the set $\mathcal{T}_{P,(\;\Rightarrow \sigma A)}$ and from the assumptions of the lemma and monotonicity of $\mathcal{T}_P$ also, for all i, for any grounding substitution τ, $(\tau \circ \sigma)B_i \in \mathcal{T}_P \downarrow n$. Since $B_1, \ldots, B_n \Rightarrow A \in P$ also $(\tau \circ \sigma)A \in \mathcal{T}_P \downarrow (n+1)$. We now apply the coinduction hypotheses to conclude that the same will be true for all subsequent iterations of $\mathcal{T}_P$. But then all instances of σA will be in the greatest Herbrand model of this program. Hence $P \vDash_{coind} \sigma A$ □

Finally, Theorem 4 states universal coinductive soundness of the coinductive type class resolution:

Theorem 4. *Let Φ be an axiom environment for a logic program P and F a formula. Let $\Phi \vdash e : F$ be by the* LP-M *and* NU' *rules. Then $\Phi \vDash_{coind} F$.*

Proof. By structural induction on the derivation tree using Lemmata 1 & 3. □

Choice of Coinductive Models. Perhaps the most unusual feature of the semantics given in this section is the use of greatest Herbrand models rather than *the greatest complete Herbrand models* that is usual in the literature on coinduction in logic programming [12, 10, 14]. *The greatest complete Herbrand models* are obtained as the greatest fixed point of the semantic operator $\mathcal{T}'_P$ on the *complete Herbrand base*, *i.e.* the set of all finite and *infinite* ground atomic formulae formed by the signature of the given program. This construction is preferred in the literature for two reasons. Firstly, $\mathcal{T}'_P$ reaches its greatest fixed point in at most ω steps, whereas $\mathcal{T}_P$ may take more than ω steps in the general case. This is due to compactness of the complete Herbrand base. Moreover, greatest complete Herbrand models give a more natural characterisation for programs like the one given by the clause $\kappa_{inf} : \mathtt{p}(x) \Rightarrow \mathtt{p}(\mathtt{f}(x))$. The greatest Herbrand model of that program is empty, but its greatest complete Herbrand model contains the infinite formula $\mathtt{p}(\mathtt{f}(\mathtt{f}(...)))$. However, restrictions (i) – (iii) imposed by type class resolution mean that the greatest Herbrand models regain those same advantages as complete Herbrand models. It was noticed by Lloyd [12] that restriction (ii) implies that the semantic operator converges in at most ω steps. Restrictions (i) and (iii) imply that proofs by type class resolution have universal interpretation, *i.e.* they hold for all finite instances of queries. Therefore, we never have to talk about programs for which only one infinite instance of a query is valid.

5 Universal Coinductive Soundness of Extended Resolution

The class of problems that can be resolved by coinductive type class resolution is limited to problems where a coinductive hypothesis is in atomic form. Fu *et al.* [5] extended coinductive type class resolution with implicative reasoning and adjusted the rule Nu' such that this restriction of coinductive type class resolution is relaxed:

Definition 9 (Extended coinductive type class resolution).

$$\textit{if}\ \mathrm{HNF}(e)\frac{\Phi, (\alpha : B_1, \ldots, B_n \Rightarrow A) \vdash e : B_1, \ldots, B_n \Rightarrow A}{\Phi \vdash \nu\alpha.e : B_1, \ldots, B_n \Rightarrow A} \qquad (\textsc{Nu})$$

The side condition of the Nu rule requires the proof witness to be in head normal form. However, unlike coinductive type class resolution, extended coinductive type class resolution also uses the Lam rule and a head normal term is of the form $\lambda\underline{\alpha}.\kappa\underline{e}$ for possibly non-empty sequence of proof term variables $\underline{\alpha}$. First, let us note that extended coinductive type class resolution indeed extends the calculus of Section 4:

Proposition 1. *The inference rule* Nu' *is admissible in the extended coinductive type class resolution.*

Further, this is a proper extension. The Nu rule allows queries to be entailed that were beyond the scope of coinductive type class resolution. In Section 1, we demonstrated a derivation for query ? `eq(bush(int))` where no cycles arise and thus the query cannot be resolved by coinductive type class resolution.

Example 8. Recall the program P_{Bush} in Example 3. The query ? $\texttt{eq}(\texttt{bush}(\texttt{int}))$ is resolved as follows:

$$
\dfrac{
\dfrac{}{\Phi_{Bush} \vdash \kappa_1 : \texttt{eq}(\texttt{int})}
\qquad
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{(\beta : \ \Rightarrow \texttt{eq}(x)) \vdash \beta : \texttt{eq}(x)}
\qquad
\dfrac{
\dfrac{
\dfrac{}{(\beta : \ \Rightarrow \texttt{eq}(x)) \vdash \beta : \texttt{eq}(x)}
}{(\alpha : \texttt{eq}(x) \Rightarrow \texttt{eq}(\texttt{bush}(x))), (\beta : \_) \vdash \alpha\beta : \texttt{eq}(\texttt{bush}(x))}
}{(\alpha : \_), (\beta : \_) \vdash \alpha(\alpha\beta) : \texttt{eq}(\texttt{bush}(\texttt{bush}(x)))}
}{\Phi_{Bush}, (\alpha : \_), (\beta : \_) \vdash \kappa_2\beta(\alpha(\alpha\beta)) : \texttt{eq}(\texttt{bush}(x))}
}{\Phi_{Bush}, (\alpha : \_) \vdash \lambda\beta.\kappa_2\beta(\alpha(\alpha\beta)) : \texttt{eq}(x) \Rightarrow \texttt{eq}(\texttt{bush}(x))}\ \textsc{Lam}
}{\Phi_{Bush} \vdash \nu\alpha.\lambda\beta.\kappa_2\beta(\alpha(\alpha\beta)) : \texttt{eq}(x) \Rightarrow \texttt{eq}(\texttt{bush}(x))}\ \textsc{Nu}
}{\Phi_{Bush} \vdash (\nu\alpha.\lambda\beta.\kappa_2\beta(\alpha(\alpha\beta)))\kappa_1 : \texttt{eq}(\texttt{bush}(\texttt{int}))}
$$

Before proceeding with the proof of soundness of extended type class resolution we need to show two intermediate lemmata. The first lemma states that inference by the Nu rule preserves coinductive soundness:

Lemma 4. *Let P be a logic program, let σ be a substitution, and let A, B_1, ..., B_n, C_1, ..., C_m be atomic formulae. If, for all i, $P, B_1, \ldots, B_n, (B_1, \ldots, B_n \Rightarrow \sigma A) \vDash_{coind} \sigma C_i$ and $(C_1, \ldots, C_m \Rightarrow A) \in P$ then $P \vDash_{coind} B_1 \ldots B_n \Rightarrow \sigma A$.*

Proof. Consider the construction of the greatest Herbrand model of the program P and proceed by coinduction with coinductive hypothesis: for all n, $B_1, \ldots, B_n \Rightarrow \sigma A$ is valid in $\mathcal{T}_P \downarrow n$. Assume that, for a groundig substitution τ, for all i, $\tau B_i \in \mathcal{T}_P \downarrow n$. Then also $(\tau \circ \sigma)A \in \mathcal{T}_P \downarrow n$. For the definition of the semantic operator, from its monotonicity, and from assumptions of the lemma it follows that $(\tau \circ \sigma)C_i \in \mathcal{T}_P \downarrow n$. Since $C_1, \ldots, C_n \Rightarrow A \in P$ also $(\tau \circ \sigma)A \in \mathcal{T}_P \downarrow (n+1)$. If the assumption does not hold the from the monotonicity of $\mathcal{T}_P$ it follows that, for all i, $\tau B_i \notin \mathcal{T}_P \downarrow (n+1)$. Therefore, $B_1, \ldots, B_n \Rightarrow \sigma A$ is valid id $\mathcal{T}_P \downarrow (n+1)$. We apply the coinductive hypothesis to conclude that the same holds for all subsequent iterations of $\mathcal{T}_P$. Hence whenever, for a substitution τ, all instances of τB_1 to τB_n are in the greatest Herbrand model then also all instances of $(\tau \circ \sigma)A$ are in the greatest Herbrand models. Hence $P \vDash_{coind} B_1, \ldots, B_n \Rightarrow A$. $\square$

The other lemma that we need in order to prove coinductive soundness of extended type class resolution states that inference using Lam preserves coinductive soundness, *i.e.* we need to show the coinductive counterpart to Lemma 2:

Lemma 5. *Let P be a logic program and A, B_1, ..., B_n atomic formulae. If $P, (\ \Rightarrow B_1), \ldots (\ \Rightarrow B_n) \vDash_{coind} A$ then $P \vDash_{coind} B_1, \ldots, B_n \Rightarrow A$.*

Proof. Assume that, for an arbitrary substitution σ, for all i, σB_i is valid in $\mathcal{M}'_P$. Then, for any grounding substitution τ, from the definition of the semantic operator and from the assumption of the lemma it follows that $(\tau \circ \sigma)A \in \mathcal{M}'_P$. Therefore, σA is valid in $\mathcal{M}'_P$. The substitution σ is choosen arbitrary whence, for any σ, if, for all i, σB_i are valid in P then also σA is valid in P. From the definition of validity it follows that $P \vDash_{coind} B_1, \ldots, B_n \Rightarrow A$. $\square$

Now, the universal coinductive soundness of extended coinductive type class resolution follows straightforwardly:

Theorem 5. *Let Φ be an axiom environment for a logic program P, and let be $\Phi \vdash e : F$ for a formula F by the* Lp-m*,* Lam*, and* Nu *rules. Then $P \vDash_{coind} F$.*

Proof. By induction on the derivation tree using Lemmata 1, 4, & 5. □

Related Program Transformation Methods Let us conclude this section with a discussion of program transformation with Horn formulae that are entailed by the rules Lam and Nu.

By the fact that the rule Nu' is inductively unsound (as discussed in the previous section), it is clear that using program transformation techniques based on lemmata proven by the rules Lam and Nu would be inductively unsound. However, a more interesting result is that adding such program clauses will not change the coinductive soundness of the initial program:

Theorem 6. *Let Φ be an axiom environment for a logic program P, and let $\Phi \vdash e : F$ for a formula F by the* Lp-m*,* Lam *and* Nu *rules such that* HNF(e)*. Given a formula F', $P \vDash_{coind} F'$ iff $(P, F) \vDash_{coind} F'$.*

The above result is possible thanks to the head normal form condition. With it, it is impossible to derive $A \Rightarrow A$ from the empty context by the rule Lam. It is also impossible to make such derivation within the proof term e itself and then derive A by the the rule Nu from $A \Rightarrow A$. The resulting proof term will fail to satisfy the head normal form condition of the rule Nu. Since this condition guards against any such cases, we can be sure that this program transformation method is coinductively sound and hence is safe to use with any coinductive dialect of logic programming, e.g. with CoLP [14].

6 Related Work

The standard approach to type inference for type classes, that corresponds to type class resolution as studied in this paper, was described by Stuckey and Sulzman [15]. Type class resolution was further studied by Lämmel and Peyton Jones [11], who described what we call coinductive type class resolution. The description of extended calculus of Section 5 was first presented by Fu *et al.* [5]. Generally, there is a body of work that focused on allowing for infinite data structures in logic programming. Logic programming with rational trees [1, 9] was studied from both an operational semantics and a declarative semantics point of view. Simon *et al.* [14] introduced co-logic programming (co-LP) that also allows for terms that are rational infinite trees and hence have infinite proofs. Appropriate models of these paradigmata are the least Herbrand model and stratified alternating fixed-point co-Herbrand model respectively. On the other hand, corecursive resolution, as studied in this paper, is more expressive than co-LP: while also allowing infinite proofs, closing of coinductive hypothesis is less constrained.

7 Conclusions and Future Work

In this paper, we have addressed three research questions. First, we provided a uniform analysis of type class resolution in both inductive and coinductive settings and proved its soundness relative to (standard) least and greatest Herbrand models. A feature of this paper is the choice of greatest Herbrand models for coinductive analysis that is allowed by properties of type class resolution. Secondly, we demonstrated on several examples that coinductive resolution is indeed coinductive—that is, it is not sound relative to least Herbrand models. Finally, we addressed the question of whether the methods listed in this paper can be brought back to coinductive dialects of logic programming via soundness preserving program transformations. As future work, we intend to establish the completeness properties of extended type class resolution. Our conjecture is that coinductive completeness of extended type class resolution can be established for a certain fragment of described calculus.

Acknowledgements

This work has been partially supported by the EU Horizon 2020 grant "RePhrase: Refactoring Parallel Heterogeneous Resource-Aware Applications - a Software Engineering Approach" (ICT-644235), by COST Action IC1202 (TACLe), supported by COST (European Cooperation in Science and Technology), and by EPSRC grant EP/K031864/1-2.

References

1. Colmerauer, A.: Equations and inequations on finite and infinite trees. In: FGCS. pp. 85–99 (1984)
2. De Angelis, E., Fioravanti, F., Pettorossi, A., Proietti, M.: Proving correctness of imperative programs by linearizing constrained horn clauses. TPLP 15(4-5), 635–650 (2015)
3. Devriese, D., Piessens, F.: On the bright side of type classes: instance arguments in agda. In: Proc. of ICFP 2011, Tokyo, Japan, September 19-21, 2011. pp. 143–155
4. Fu, P., Komendantskaya, E.: Operational semantics of resolution and productivity in horn clause logic. Formal Aspect of Computing (2016), forthcoming
5. Fu, P., Komendantskaya, E., Schrijvers, T., Pond, A.: Proof relevant corecursive resolution. In: Proc. of FLOPS 2016, Kochi, Japan, March 4-6, 2016 (2016)
6. Gonthier, G., Ziliani, B., Nanevski, A., Dreyer, D.: How to make ad hoc proof automation less ad hoc. In: Proc. of the 16th ACM SIGPLAN international conference on Functional Programming, ICFP 2011, Tokyo, Japan, September 19-21, 2011. pp. 163–175 (2011)
7. Hall, C.V., Hammond, K., Jones, S.L.P., Wadler, P.: Type classes in haskell. ACM Trans. Program. Lang. Syst. 18(2), 109–138 (1996)
8. Howard, W.: The formulae-as-types notion of construction. In: Seldin, J.P., Hindley, J.R. (eds.) To H. B. Curry: Essays on Combinatory Logic, Lambda-Calculus, and Formalism. pp. 479–490. Academic Press, NY, USA (1980)
9. Jaffar, J., Stuckey, P.J.: Semantics of infinite tree logic programming. Theor. Comput. Sci. 46(3), 141–158 (1986)
10. Komendantskaya, E., Johann, P.: Structural resolution: a framework for coinductive proof search and proof construction in horn clause logic. ACM Transcations on Computational Logic submitted (2016)
11. Lämmel, R., Peyton Jones, S.L.: Scrap your boilerplate with class: extensible generic functions. In: Proc. of ICFP 2005, Tallinn, Estonia, September 26-28, 2005. pp. 204–215
12. Lloyd, J.W.: Foundations of Logic Programming, 2nd Edition. Springer (1987)
13. Sangiorgi, D.: On the origins of bisimulation and coinduction. ACM Trans. Program. Lang. Syst. 31(4), 15:1–15:41 (May 2009)
14. Simon, L., Bansal, A., Mallya, A., Gupta, G.: Co-logic programming: Extending logic programming with coinduction. In: Proc. of ICALP 2007, Wroclaw, Poland, July 9-13, 2007. pp. 472–483 (2007)
15. Stuckey, P.J., Sulzmann, M.: A theory of overloading. ACM Trans. Program. Lang. Syst. 27(6), 1216–1269 (2005)
16. Wadler, P., Blott, S.: How to make ad-hoc polymorphism less ad hoc. In: Proc. of POPL '89. pp. 60–76. ACM, New York, NY, USA (1989)

A Ommited Proofs

In this appendix, we state omited proofs in their fullness.

Lemma 1. *Let P be a logic program and let σ be a substitution. The following holds:*

a) If $(\ \Rightarrow A) \in P$ then both $P \vDash_{ind} \sigma A$ and $P \vDash_{coind} \sigma A$
b) If, for all i, $P \vDash_{ind} \sigma B_i$ and $(B_1, \dots, B_n \Rightarrow A) \in P$ then $P\ \vDash_{ind}\ \sigma A$
c) If, for all i, $P \vDash_{coind} \sigma B_i$ and $(B_1, \dots, B_n \Rightarrow A) \in P$ then $P \vDash_{coind} \sigma A$

Proof. a) Let P be a logic program such that $(\ \Rightarrow A) \in P$. By Definition 2 of the semantic operator, for any grounding substitution τ, $\tau A \in \mathcal{T}_P(\mathcal{M}_P)$. Since $\mathcal{M}_P$ is a fixed point of $\mathcal{T}_P$ also $\tau A \in \mathcal{M}_P$ and by definition of validity of a formula, $P \vDash_{ind} A$ and also, for any substitution σ, $P \vDash_{ind} \sigma A$. Since we do not use the fact that $\mathcal{M}_P$ is the least fixed point the proof for coinductive case is identical.

b) Let be P, A, B_1, …, B_n as above. Assume, for all i, $P \vDash_{ind} B_i$ whence, for all i, for any grounding substitution σ, $\sigma B_i \in \mathcal{M}_P$. By Definition 2 of semantic operator, $\sigma A \in \mathcal{T}_P(\mathcal{M}_P)$. Since $\mathcal{M}_P$ is a fixed point also $\sigma A \in \mathcal{M}_P$ and $P \vDash_{ind} \sigma A$.

c) Note that the proof of b) does not make any use of the fact that $\mathcal{M}_P$ is the least fixed point. Therefore use the proofs of and b) *mutatis mutandis.* □

Theorem 1. *Let Φ be an axiom environment for a logic program P, and let $\Phi \vdash e : A$ hold. Then $P \vDash_{ind} A$.*

Proof. By structural induction on the derivation tree.

Base case: Let the derivation be $\dfrac{}{\Phi \vdash \kappa : A}$ for an atomic formula A', a proof term symbol κ, and a substitution σ such that $A = \sigma A'$. By definition of the rule LP-M there is a clause $(\kappa :\ \Rightarrow A') \in \Phi$ and from the Definition 5 of axiom environment also $\ \Rightarrow A' \in P$. From the Lemma 1 part a) follows that $P \vDash_{ind} \sigma A'$.

Inductive case: Let the last step in the derivation tree be $\dfrac{\Phi \vdash e_1 : \sigma B_1 \dots \Phi \vdash e_n : \sigma B_n}{\Phi \vdash \kappa e_1 \dots e_n : \sigma A'}$ for atomic formulae A', B_1, …, B_n, a proof term symbol κ, a substitution σ and proof witneses $e_1, \dots, e_n$ such that $A = \sigma A'$. From the definition of the rule LP-M there is a clause $(\kappa : B_1, \dots B_n \Rightarrow A') \in \Phi$ and from the Definition 5 of axiom environment also $B_1, \dots, B_n \Rightarrow A' \in P$. From the induction assumption, for $i \in \{1, \dots, n\}$, $P \vDash_{ind} \sigma B_i$ and by the Lemma 1 part b), $P \vDash_{ind} \sigma A'$. □

Lemma 2. *Let P be a logic program and let A, B_1, …, B_n be atomic formulae. If $P, (\ \Rightarrow B_1), \dots, (\ \Rightarrow B_n) \vDash_{ind} A$ then $P \vDash_{ind} B_1, \dots, B_n \Rightarrow A$.*

Proof. Assume that $P, (\ \Rightarrow B_1), \dots, (\ \Rightarrow B_n) \vDash_{ind} A$. From the Definition 2 there is the least n such that for any grounding substitution τ, $(\tau \circ \sigma)A \in \mathcal{T}_{P,(\ \Rightarrow B_1),\dots,(\ \Rightarrow B_n)} \uparrow n$. Consider and substitution σ and suppose that for all i, $P \vDash \sigma B_i$. From the definition of validity for any grounding τ for all i,

$(\tau \circ \sigma)B_i \in \mathcal{M}_P$ hence there is the least m such that $(\tau \circ \sigma)B_i \in \mathcal{T}_P \uparrow m$. From the assumption, for any grounding substitution τ also $(\tau \circ \sigma)A \in \mathcal{T}_P \uparrow (n+m)$ and $P \vDash \sigma A$. Hence $P \vDash_{ind} B_1, \ldots, B_n \Rightarrow A$. □

Theorem 2. *Let Φ be an axiom environment for a logic program P and F a formula. Let $\Phi \vdash e : F$ be by the* Lp-m *and* Lam *rules. Then $P \vDash_{ind} F$.*

Proof. By structural induction on the derivation tree.

Base case: Let the derivation be $\dfrac{}{\Phi \vdash \kappa : A}$ for an atomic formula A', a constant symbol κ, and a substitution σ such that $A = \sigma A'$. By definition of the rule Lp-m there is a clause $(\kappa : \ \Rightarrow A') \in \Phi$ and from the Definition 5 of axiom environment also $\ \Rightarrow A' \in P$. From the Lemma 1 part a) follows thet $P \vDash_{ind} \sigma A'$.

Inductive case: Let the last step in the derivation tree be by the rule Lp-m thus of the form $\dfrac{\Phi \vdash e_1 : \sigma B_1 \ldots \Phi \vdash e_n : \sigma B_n}{\Phi \vdash \kappa e_1 \ldots e_n : \sigma A}$ for atomic formulae A', B_1, $\ldots$, B_n, a proof term symbol κ, a substitution σ and proof witnesses $e_1, \ldots, e_n$. From the definition of the rule Lp-m there is a clause $(\kappa : B_1, \ldots B_n \Rightarrow A) \in \Phi$ and from the Definition 5 of axiom environment also $B_1, \ldots, B_n \Rightarrow A' \in P$. From the induction assumption, for $i \in \{1, \ldots, n\}$, $\Phi \vDash_{ind} \sigma B_i$ and by the Lemma 1 part b), $P \vDash_{ind} \sigma A'$.

Let the last step of the derivation be by the rule Lam thus of the form $\dfrac{\Phi, (\beta_1 : \ \Rightarrow B_1), \ldots, (\beta_n : \ \Rightarrow B_n) \vdash e : A}{\Phi \vdash \lambda \beta_1, \ldots, \beta_n.e : B_1, \ldots, B_n \Rightarrow A}$ for atomic formulae A, B_1, $\ldots$, B_n, proof term e, and variables b_1, $\ldots$, b_n. From the induction assumption, $P, (\ \Rightarrow B_1), \ldots, (\ \Rightarrow B_n) \vDash A$ and from the Lemma 2 also $P \vDash_{ind} A$. □

Lemma 3. *Let P be a logic program, let σ be a substitution, and let A, B_1, $\ldots$, B_n be atomic formulae. If, $\forall i \in \{1, \ldots, n\}$, $P, (\ \Rightarrow \sigma A) \vDash_{coind} \sigma B_i$ and $(B_1, \ldots, B_n \Rightarrow A) \in P$ then $P \vDash_{coind} \sigma A$.*

Proof. Consider now the construction of the greatest Herbrand model for the program P and proceed by coinduction with coinductive hypothesis: for all n, for any grounding substitution τ, $(\tau \circ \sigma)A \in \mathcal{T}_P \downarrow n$. Since, for any τ, $(\tau \circ \sigma)A \in \mathcal{T}_P \downarrow n$ the set $\mathcal{T}_P \downarrow n$ is by definition of the operator $\mathcal{T}_P$ the same as the set $\mathcal{T}_{P,(\ \Rightarrow \sigma A)}$ and from the assumptions of the lemma and monotonicity of $\mathcal{T}_P$ also, for all i, for any grounding substitution τ, $(\tau \circ \sigma)B_i \in \mathcal{T}_P \downarrow n$. Since $B_1, \ldots, B_n \Rightarrow A \in P$ also $(\tau \circ \sigma)A \in \mathcal{T}_P \downarrow (n+1)$. We now apply the coinduction hypotheses to conclude that the same will be true for all subsequent iterations of $\mathcal{T}_P$. But then all instances of σA will be in the greatest Herbrand model of this program. Hence $P \vDash_{coind} \sigma A$ □

Theorem 4. *Let Φ be an axiom environment for a logic program P and F a formula. Let $\Phi \vdash e : F$ be by the* Lp-m *and* Nu' *rules. Then $\Phi \vDash_{coind} F$.*

Proof. By structural induction on the derivation tree.

Base case: Let the derivation be with empty assumptions. Then it is by the rule Lam and of the form $\dfrac{}{\Phi \vdash \kappa : \sigma A}$ for an atomic formula A, a constant symbol κ, and a substitution σ. By definition of the rule LP-M there is a clause $(\kappa : \ \Rightarrow A) \in \Phi$. By Lemma 1 c), $\Phi \vDash_{coind} \sigma A$.

Inductive case: Let the last step be by the rule LP-M and of the form $\dfrac{\Phi \vdash e_1 : \sigma B_1 \dots \Phi \vdash e_n : \sigma B_n}{\Phi \vdash \kappa e_1 \dots e_n : \sigma A}$ for an atomic formulae A, B_1, ..., B_n a constant symbol κ, a substitution σ and proof witnesses e_1, ..., e_n. By definition of the rule LP-M there is a clause $(\kappa : B_1, \dots B_n \Rightarrow A) \in \Phi$.

By the induction assumption, for $i \in \{1, \dots, n\}$, $\Phi \vDash_{coind} B_i$ and by Lemma 1 d), $\Phi \vDash_{coind} \sigma A$.

Let the last step be by the rule NU' and of the form $\dfrac{\Phi, (\alpha : \ \Rightarrow A) \vdash e : A}{\Phi \vdash \nu\alpha.e : A}$ for an atomic formula A, a variable α and a proof witness e in the head normal form. W.l.o.g. let $e = \kappa e_1 \dots e_n$. Therefore there is (the previous? depends on h.n.f.) inference step of the form $\dfrac{\Phi \vdash e_1 : \sigma B'_1 \dots \Phi \vdash e_n : \sigma B'_n}{\Phi \vdash \kappa e_1 \dots e_n : \sigma A'}$ for $\sigma A' = A$ and $(\kappa : B'_1, \dots B'_n \Rightarrow A') \in \Phi$. By the induction assumption, for all i, $\Phi, (\alpha : \ \Rightarrow A) \vDash B_i$. By Lemma 3, $\Phi \vDash_{coind} A$. □

Proposition 1. *The inference rule* NU' *is admissible in the extended coinductive type class resolution.*

Proof. Let Φ be an environment, let A be an atomic formula and let $\Phi, (\alpha : \ \Rightarrow A) \vdash e : A$ where e is in head normal form. Then by the LAM rule $\Phi, (\alpha : \ \Rightarrow A) \vdash \lambda \underline{\beta}.e : \ \Rightarrow A$ where $\underline{\beta}$ is an empty sequence of variables. Therefore $\Phi, (\alpha : \ \Rightarrow A) \vdash e : \ \Rightarrow A$. Since e is in head normal form by the NU rule also $\Phi \vdash \nu\alpha.e : A$. □

Lemma 4. *Let P be a logic program, let σ be a substitution, and let A, B_1, ..., B_n, C_1, ..., C_m be atomic formulae. If, for all i, $P, B_1, \dots, B_n, (B_1, \dots, B_n \Rightarrow \sigma A) \vDash_{coind} \sigma C_i$ and $(C_1, \dots, C_m \Rightarrow A) \in P$ then $P \vDash_{coind} B_1 \dots B_n \Rightarrow \sigma A$.*

Proof. Consider the construction of the greatest Herbrand model of the program P and proceed by coinduction with coinductive hypothesis: for all n, $B_1, \dots, B_n \Rightarrow \sigma A$ is valid in $\mathcal{T}_P \downarrow n$. Assume that, for a groundig substitution τ, for all i, $\tau B_i \in \mathcal{T}_P \downarrow n$. Then also $(\tau \circ \sigma)A \in \mathcal{T}_P \downarrow n$. For the definition of the semantic operator, from its monotonicity, and from assumptions of the lemma it follows that $(\tau \circ \sigma)C_i \in \mathcal{T}_P \downarrow n$. Since $C_1, \dots, C_n \Rightarrow A \in P$ also $(\tau \circ \sigma)A \in \mathcal{T}_P \downarrow (n+1)$. If the assumption does not hold the from the monotonicity of $\mathcal{T}_P$ it follows that, for all i, $\tau B_i \notin \mathcal{T}_P \downarrow (n+1)$. Therefore, $B_1, \dots, B_n \Rightarrow \sigma A$ is valid id $\mathcal{T}_P \downarrow (n+1)$. We apply the coinductive hypothesis to conclude that the same holds for all subsequent iterations of $\mathcal{T}_P$. Hence whenever, for a substitution τ, all instances of τB_1 to τB_n are in the greatest Herbrand model then also all instances of $(\tau \circ \sigma)A$ are in the greatest Herbrand models. Hence $P \vDash_{coind} B_1, \dots, B_n \Rightarrow A$. □

Lemma 5. *Let P be a logic program and A, B_1, ..., B_n atomic formulae. If $P, (\ \Rightarrow B_1), \dots (\ \Rightarrow B_n) \vDash_{coind} A$ then $P \vDash_{coind} B_1, \dots, B_n \Rightarrow A$.*

Proof. Assume that, for an arbitrary substitution σ, for all i, σB_i is valid in $\mathcal{M}'_P$. Then, for any grounding substitution τ, from the definition of the semantic operator and from the assumption of the lemma it follows that $(\tau \circ \sigma)A \in \mathcal{M}'_P$. Therefore, σA is valid in $\mathcal{M}'_P$. The substitution σ is choosen arbitrary whence, for any σ, if, for all i, σB_i are valid in P then also σA is valid in P. From the definition of validity it follows that $P \vDash_{coind} B_1, \ldots, B_n \Rightarrow A$. $\square$

Theorem 5. *Let Φ be an axiom environment for a logic program P, and let be $\Phi \vdash e : F$ for a formula F by the* Lp-m, Lam, *and* Nu *rules. Then $P \vDash_{coind} F$.*

Proof. By structural induction on the derivation tree.

Base case: Let the derivation be with empty assumptions. Then it is by the rule Lam and of the form $\dfrac{}{\Phi \vdash \kappa : \sigma A}$ for an atomic formula A, a constant symbol κ, and a substitution σ. By definition of the rule Lp-m there is a clause $(\kappa : \ \Rightarrow A) \in \Phi$. By Lemma 1 c), $\Phi \vDash_{coind} \sigma A$.

Inductive case: Let the last step be by the rule Lp-m and of the form $\dfrac{\Phi \vdash e_1 : \sigma B_1 \ldots \Phi \vdash e_n : \sigma B_n}{\Phi \vdash \kappa e_1 \ldots e_n : \sigma A}$ for an atomic formulae A, B_1, …, B_n a constant symbol κ, a substitution σ and proof witnesses e_1, …, e_n. By definition of the rule Lp-m there is a clause $(\kappa : B_1, \ldots B_n \Rightarrow A) \in \Phi$.

By the induction assumption, for $i \in \{1, \ldots, n\}$, $\Phi \vDash_{coind} B_i$ and by Lemma 1 d), $\Phi \vDash_{coind} \sigma A$.

Let the last step of the derivation be by the rule Lam. Then it is of the form $\dfrac{\Phi, (\beta_1 : \ \Rightarrow B_1), \ldots, (\beta_n : \ \Rightarrow B_n) \vdash e : A}{\Phi \vdash \lambda\beta_1, \ldots, \beta_n.e : B_1, \ldots, B_n \Rightarrow A}$ for atomic formulae A, B_1, …, B_n, proof term e, and variables $b_1, \ldots, b_n$. By the induction assumption, $\Phi, (\beta_1 : \ \Rightarrow B_1), \ldots, (\beta_n : \ \Rightarrow B_n) \vDash_{coind} A$ and by Lemma 5 also $\Phi \vDash_{coind} B_1, \ldots, B_n \Rightarrow A$.

Let the last step be by the rule Nu and of the form $\dfrac{\Phi, (\alpha : B_1, \ldots, B_n \Rightarrow A) \vdash e : B_1, \ldots B_n \Rightarrow A}{\Phi \vdash \nu\alpha.e : B_1, \ldots, B_n \Rightarrow A}$ for an atomic formulae A, B_1, …, B_n, a variable α and a proof witness e in the head normal form. W.l.o.g. let $e = \lambda\beta_1 \ldots \beta_n.\kappa e_1 \ldots e_m$. Therefore there is inference step of the form

$$\dfrac{\begin{array}{c}\Phi, (\beta_1 : \ \Rightarrow B_1), \ldots, (\beta_n : \ \Rightarrow B_n), (\alpha : B_1, \ldots, B_n \Rightarrow A) \vdash e_1 : \sigma C'_1 \\ \ldots \\ \Phi, (\beta_1 : \ \Rightarrow B_1), \ldots, (\beta_n : \ \Rightarrow B_n), (\alpha : B_1, \ldots, B_n \Rightarrow A) \vdash e_m : \sigma C'_m\end{array}}{\Phi, (\beta_1 : \ \Rightarrow B_1), \ldots, (\beta_n : \ \Rightarrow B_n), (\alpha : B_1, \ldots, B_n \Rightarrow A) \vdash \kappa e_1 \ldots e_n : \sigma A'}$$

for $\sigma A' = A$ and $(\kappa : C'_1, \ldots C'_m \Rightarrow A') \in \Phi$. By the induction assumption, for all i, $\Phi, (\beta_1 : B_1), \ldots, (\beta_n : B_n), (\alpha : B_1, \ldots, B_n \Rightarrow A) \vDash C_i$. By Lemma 4 $\Phi \vDash_{coind} B_1, \ldots, B_n \Rightarrow A$. $\square$

B Reviews

Review 1

Overall evaluation: **2** (accept)

Reviewer's confidence: **4** (high)

Direct acceptance to LNCS proceedings: yes

Review: The paper introduces a proof calculus, called corecursive type class resolution, for proving coinductive properties for Haskell type classes encoded as a particular class of logical programs. The soundness of the proof system is proved.

The paper is well written and the reported results are consistent. The investigated problem is clearly defined and well motivated by illustrative examples. I didn't deeply check the proofs, but the results seems plausible.

The paper fits the conference topics since it investigates how the co-logic programming paradigm can be used to solve an important problem from functional programming, namely type class resolution. Therefore I think that it deserves to be accepted and presented at LOPSTR 2016.

Nothing to answer/comment on in the above.

Detailed comments:

- ~~p. 1: "we study coinductive properties of proofs" - I think that "coinductive properties of type classes" is studied in the paper.~~ The above text got reformulated
- ~~p. 5: explicitly mention the Horn clause $A \Leftarrow B_1, ..., B_n$ in the restriction. Mention also that FV(A) denotes the set of free variables of A.~~ The restriction was reformulated
- ~~p. 6, Def. 2: The a mapping $\to$ Then a mapping~~ Done
- ~~p.6, in the definition for the validity of a Horn formula: I guess that it should be "for any substitution σ" instead of "for σ a substitution"~~ Done
- ~~p.6, Lemma 1: Mention that σ is a substitution and give a reference where a proof can be found.~~ We refer to Lloyd [12] for proof
- p. 9, 12: the rules (Mu') and (Mu) are similar to the circularity rule from the circular coinduction proof system, which allows to use the property you have to prove as a hypothesis in a sound way. I guess that the guarded condition HNF(e) plays a role similar to that of the freezing operator in circular coinduction, namely to forbid the unsound use of the coinductive hypothesis. Here is a reference for the circular coinduction:

  Grigore Rosu and Dorel Lucanu. *Circular coinduction — a proof theoretical foundation.* In CALCO 2009, volume 5728 of Lecture Notes in Computer Science, pages 127–144. Springer, 2009.

- ~~I suggest to use ν instead of μ in the definition of the proof terms; usually, μ is used for the least fixpoint and ν for the greatest fixpoint. The coinductive reasoning is related to the greatest fixpoint.~~ Done
- ~~p. 10: "We note that the rule Mu' is inductively unsound." This is not so surprising. According to~~

  Davide Sangiorgi. An Introduction to Bisimulation and Coinduction. Cambridge University Press, 2012. A preliminary version of Chapter 2 from which we adapt paterial can be found at `http://www.cs.unibo.it/~sangio/DOC_public/corsoFL.pdf`.

- ~~the least fixpoint includes elements built with finite proff trees and the greatest fixpoint includes elements built with both finite and infinite proof trees. The cycles discarded by (Mu') could be part of infinite inferences.~~ I added a reference to Sangiorgi [13] (different paper than the paper mention by the referee)
- ~~p. 11: greatest greatest → greatest~~ done

Response to Review 1 The reviewer provided detailed comments on formulations of several sentences in our paper and noticed some typographical errors, *e.g.* repeated words. We would like to thank the reviewer for these comments. We made the suggested changes in several places and we reformulated the text in the other also due to the comments of other reviewers. We do not discus these comments in a further detail.

However, the reviewer suggested two further references. In the first reference,

Grigore Rosu and Dorel Lucanu. *Circular coinduction — a proof theoretical foundation.* In CALCO 2009, volume 5728 of Lecture Notes in Computer Science, pages 127–144. Springer, 2009.

authors present a proof system for circular coinduction. In the referenced paper, Rosu and Lucanu state that they do not prove soundness with respect to any model. Although we believe that the reviewers suggestion on similarity of guardedness condition and freezing operator is correct we do not think this paper is the right place to discuss it as we are interested here in soundness with respect to Herbrand models.

The other reference the reviewer provides is the

Davide Sangiorgi. *An Introduction to Bisimulation and Coinduction.* Cambridge University Press, 2012. A preliminary version of Chapter 2 from which we adapt paterial can be found at `http://www.cs.unibo.it/~sangio/DOC_public/corsoFL.pdf`.

We already cite one of the Sangiorgi's papers [13] and we include this reference where the reviewer suggests to include the above one as we believe it servers the purpose the reviewer intended.

Review 2

Overall evaluation: **1** (weak accept)

Reviewer's confidence: **3** (medium)

Direct acceptance to LNCS proceedings: -

Review: This work advances recent work by the last author at FLOPS 2016. Said previous work focused on enhancing corecursive type-class resolution by providing a general approach to proof-style resolution which specifically included an idea of adding horn formula resolvents to the proof environment. This new work focuses on showing (proving) that the extensions of the previous work are (co)inductively sound relative to the (least) greatest Herbrand models of logic programs.

The new paper very much overlaps with the previous paper, but I don't think that this needs to be a problem. The original aspects seem to consist of, specifically, the soundness-related theorems and sections 3.1, 4.1, and 5.1. The research on transformation techniques relates very well to LOPSTR.

The amount of reuse of material from the previous publication is very significant. Thus, I am not fully convinced. The authors don't carefully enough pronounce their contribution and the significance of it—in relation to their previous work.

Detailed comments:

- .2, l.-3: Is there a type class-relevant counterpart for the $\Rightarrow q(f(x))$ example? I removed the example
- ~~p. 3: Regarding the resolution trace for eq(evenList(int)): The style is not clear. For instance, there are two subsequent terms that mention oddList.~~ I added a para of explanation
- p. 3-4: The development along these pages seems to suggest a point at which lambdas for proof terms become necessary. They suddenly arise with the bush example; some explanation as to when lambdas are needed is appreciated.
- ~~p. 4: citation [11] is mentioned for its "recent corecursive extension". This is not clear—given that [11] deals with logic programming (dealing very well with SYB) with no immediate contribution to type-class resolution. Please clarify.~~ It was a typo, the coorect citation is Lämmel and Peyton Jones [11]
- ~~p. 5; l. -7: The B_i and the A in the restriction (ii) come out of nowhere.~~ We reformulated the restriction
- p. 7: Regarding "However, a lemma may overlap with other axioms and lemmata." This was not clear.
- p. 7: The notion of sequence for proof terms was a bit puzzling. There is only application. Likewise, the notion of sequence of lambda variables has not been explained, but, arguably, it is more established.

– The section structure throughout sections 3-5 is strange with just one subsection 3.1, 4.1, 5.1. Why is that and how can you better clarify your contributions?

Typos:

– ~~Commas around "e.g." and "i.e." are preferred.~~ We use british grammar, commas depend on paritucular occurence of "*e.g.* and" "*i.e.*"
– ~~p. 5: "I this section".~~ Done.
– ~~p. 6: "any σ grounding subsitution".~~ Done.
– ~~p. 13: "preservers".~~ Done.
– ~~p. 13: "by discussion" → "by a discussion".~~ Done.
– ~~p. 14: "On more general level".~~ Not in the text any more.

Response to Review 2 The reviewer raised a question of novelty of material in the paper and its overlap with Fu's *et al.* paper [5]. We changed closing of the introduction of our paper to state more clearly that our paper studies soundness of the calculus of extended corecursive type class resolution, which is not present in Fu *et al.*, and also soundness of preexisting calculi. The novelty of our paper are the theorems of coinductive soundness of the calculi in Section 4 and Section 5 and soundness of program transformation methods.

We address the detailed comments the reviewer made as follows: The reviewer noted that some explanation why lambdas arise with the `Bush` example is appreciated. We include a reference that clarifies why such an extension of resolution mechanism is necessary.

The reviewer noted that the sentence "However, a lemma may overlap with other axioms and lemmata" was not clear. We clarified our definitions of an axiom and a lemma in the context of axiom environment and we clarified the definition of overlapping in the restrictions in the Section 1.

Furthermore, the reviewer pointed out several formulations that were not clear and some typographical errors. We clarified suggested parts of the text and we fixed the typographical errors. We do not discuss these in further detail.

Review 3

Overall evaluation: **1** (weak accept)

Reviewer's confidence: **4** (high)

Direct acceptance to LNCS proceedings: -

Review: This paper proposes a nice application of the theory of Logic Programming to Type Theory. In particular, it considers some recent extensions of type class resolution that allow certain forms of co-inductive proofs, and proves various soundness and unsoundness results with respect to the least and the greatest (standard) Herbrand models.

On the negative side, I think there is a problem with the paper, namely Theorem 6 seems incorrect. The same counterexample given at the end of Section 3.1 should hold also for the statement of Theorem 6, since the only difference between Theorem 6 and the situation analyzed in Section 3.1 is the presence of the MU rule, which does not play any role in the counterexample. Anyway, Theorem 6 is just a side result, so even if it is incorrect, it can be removed (together with the whole Section 5.1 which is just about this result) without affecting the rest of the paper.

Another problem is that I find a bit disappointing that the question of completeness is not addressed. More precisely, it is dismissed at the end of Page 2 using the fact that type class resolution performs instantiation instead of full resolution, thus it won't be able to prove non-trivial existential formulas. This is true, but obviously one should consider whether these extensions are powerful enough to derive universal formulas when they are true in the least (resp. greatest) Herbrand model. I suspect that it should not be too difficult to analyze at least the case of atomic formulas, and that the answer should be yeas in the inductive (least model) case, and no in the co-inductive case.

Despite the above problems, I still think that it is a nice paper, and I am in favor for accepting it.

MINOR REMARKS FOR THE AUTHORS

- ~~In the introduction, after Exampe 1, ? eq(int, int) should be ? eq(pair(int,int))~~ Typo, done.
- Page 3: Please explain what the notation $\mu\alpha.\kappa_2\kappa_3(\kappa_1\kappa_3\alpha)$ represents. Are κ_1, κ_2 and κ_3 the "witness" of the logic programming clauses? And what is the result of, for instance, applying κ_3 to α ?
- ~~Page 11, Second line of Section 4.1: greatest greatest Herbrand model~~ Done.

Response to Review 3 The reviewer suggested there is a problem with the statement of the Theorem 6. Indeed, a necessary condition was missing in the statement of the theorem in the previously submitted version of the paper. We amended the statement of the theorem and extended the explanation to make the use of this condition clear. Furthermore, the reviewer noted that we do not address the question of completeness. He provided several conjectures of completeness results in inductive and coinductive case. We believe that the question of completeness is an interesting one and that his conjectures are in concordance with the future work we project in the Section 7. However, we do not think that the page limit of the paper allows for proper analysis of the issue and we omit any partial analysis in favour of more thorough discussion of soundness results.

The reviewer also made some minor remarks. We incorporated these into the text and we do not discuss them further.

C Completeness

Example 9. The calculus of Lp-m and Lam is not complete in the inductive case. Consider the following program with a function symbol `f`:

$$\begin{aligned} \kappa_1 : &\ \Rightarrow \mathtt{A(f)} \\ \kappa_2 : &\ \Rightarrow \mathtt{B(f)} \end{aligned}$$

The least Herbrand model $\mathcal{M}_P = \{A(f), B(f)\}$. Therefore $P \vDash_{ind} \mathtt{B}(x) \Rightarrow \mathtt{A}(x)$. However, any proof of $\mathtt{B}(x) \Rightarrow \mathtt{A}(x)$ needs to show that:

$$\dfrac{\dfrac{\dots}{(P, \alpha : \ \Rightarrow \mathtt{B}(x)) \vdash e : \mathtt{A}(x)}}{P \vdash \lambda\alpha.e : \mathtt{B}(x) \Rightarrow \mathtt{A}(x)}\ \textsc{Lam}$$

where e is a proof term. Since $\mathtt{A}(x)$ is not an implication and cannot proceed by the rule Lam. However, neither $\mathtt{B}(x)$, nor any of heads of the clauses in P does match the goal $\mathtt{A}(x)$. Therefore the proof cannot proceed by the rule Lp-m.

The same example show incompletness of the calculus of Lp-m, Lam, and Nu in the coinductive case. The greatest Herbrand model $\mathcal{M}'_P$ is the same and although the rule Nu allows to begin the the above proof with another step, it does not instantiate the goal $\mathtt{A}(x)$ anyhow and therefore neither of the clauses κ_1 nor κ_2 will be allicable in later steps. Therefore there is no way how to construct an proof term e such that HNF(e) in order to make coinductively sound conclusion.

As for the completness for atomic formulate, consider the following example:

Example 10. Let Σ be signature with one predicate symbol `A` and two function symbols `f` and `g`, the first unary and the other nullary.

$$\begin{aligned} \kappa_1 : &\ \Rightarrow \mathtt{A(f(}x\mathtt{))} \\ \kappa_2 : &\ \Rightarrow \mathtt{A(g)} \end{aligned}$$

The least Herbrand model $\mathcal{M}_P = \{A(g), A(f(g)), A(f(f(g))), \dots\}$. Therefore $P \vDash_{ind} \mathtt{A}(x)$. However, neither the clause κ_1 nor the clause κ_2 math $\mathtt{A}(x)$ and there is no way how to proceed with the proof

$$\dfrac{\dots}{P \vdash e : \mathtt{A}(x)}\ \textsc{Lp-m}$$

A similar argument holds in the coinductive case.