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, 6] and has been further developed in dependently typed languages [5, 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). For example, given a clause $\Rightarrow \mathtt{q}(\mathtt{f}(x))$ and a query ? $\mathtt{q}(x)$, SLD-resolution is able to find a proof (instantiating x with $\mathtt{f}(x)$), but type class resolution fails.

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 $P_{EvenOdd}$).

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

The non-terminating resolution trace is given by:

$$\underline{\mathsf{eq}(\mathsf{evenList}(\mathtt{int}))} \to_{\kappa_2} \mathsf{eq}(\mathtt{int}),\ \mathsf{eq}(\mathsf{oddList}(\mathtt{int})) \to_{\kappa_3} \mathsf{eq}(\mathsf{oddList}(\mathtt{int}))$$
$$\to_{\kappa_1} \mathsf{eq}(\mathtt{int}),\ \mathsf{eq}(\mathsf{evenList}(\mathtt{int})) \to_{\kappa_3} \underline{\mathsf{eq}(\mathsf{evenList}(\mathtt{int}))} \to_{\kappa_2} \dots$$

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 ? $\mathsf{eq}\ (\mathsf{evenList}(\mathtt{int}))$ exists, and that its shape is fully specified by the recursive proof witness function above. 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, $\mathsf{eq}(\mathsf{evenList}(\mathtt{int}))$ 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, $\mathsf{eq}(\mathsf{evenList}(\mathtt{int}))$ 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 $\kappa_{inf} : \mathrm{p}(x) \Rightarrow \mathrm{p}(\mathtt{f}(x))$. 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}).

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

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

$$\begin{aligned} \mathtt{eq}(\mathtt{bush}(\mathtt{int})) \to_{\kappa_2} \mathtt{eq}(\mathtt{int}), \mathtt{eq}(\mathtt{bush}(\mathtt{bush}(\mathtt{int})) \to_{\kappa_1} \ldots \to_{\kappa_2} \\ \mathtt{eq}(\mathtt{bush}(\mathtt{int})), \mathtt{eq}(\mathtt{bush}(\mathtt{bush}(\mathtt{bush}(\mathtt{int}))) \to_{\kappa_1} \ldots \end{aligned}$$

Fu *et al.* [4] 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 $\mathtt{eq}(x) \Rightarrow \mathtt{eq}(\mathtt{bush}(x))$ 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 in our program, we obtain a proof of $\mathtt{eq}(\mathtt{bush}(\mathtt{int}))$ 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 $\mathtt{eq}(\mathtt{bush}(\mathtt{int}))$ 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.* [4]. THis extends Howard's simply-typed lambda calculus [7] 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 [4, 10] 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 [4] can be brought back to logic programming and can be re-used in its corecursive dialects such as CoLP [14] and CoALP [9] 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 [4] indeed resembles a program transformation method when considered from the logic programming point of view. In reality, different fragments of the calculus given in [4] 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).

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

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 above 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 of the lemma can be found in the existing literature [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 [4], 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 and proof terms respectively. 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. We refer the reader to [4] 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.* [4]. 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).

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

If, for a given atomic formula A, and a given environment $\varPhi$, $\varPhi \vdash e : A$ is derived using the Lp-m rule we say that A is entailed by $\varPhi$ 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.* [4]).

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:

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

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

Theorem 1. *Let $\varPhi$ be an axiom environment for a logic program P, and let $\varPhi \vdash e : A$ 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.* [4]. The rule that allows Horn formulae to be proved is:

Definition 7.

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

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 $\varPhi = (\kappa_1 : A \Rightarrow B), (\kappa_2 : B \Rightarrow C)$. The following is then a derivation tree for a formula $A \Rightarrow C$:

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

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. Let σ be an arbitrary substitution. Assume that, for all i, $P \vDash_{ind} \sigma B_i$. By the definition of $\mathcal{T}_{P,(\ \Rightarrow B_1),\ldots(\ \Rightarrow B_n)}$, for all i and for any τ grounding substitution, $(\tau \circ \sigma)B_i \in \mathcal{T}_{P,(\ \Rightarrow B_1),\ldots(\ \Rightarrow B_n)} \uparrow 1$ and from the above condition $P, (\ \Rightarrow B_1), \ldots (\ \Rightarrow B_n) \vDash_{ind} A$ and the universal quantification of formulae there is m such that $A \in \mathcal{T}_{P,(\ \Rightarrow B_1),\ldots(\ \Rightarrow B_n)} \uparrow (m+1)$. Hence from the assumption $\Phi \vDash_{ind} A$. By definition of validity, $P \vDash_{ind} B_1, \ldots, B_n \Rightarrow A$. □

Theorem 2. *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. 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.* [4], 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 $\emptyset \vdash \lambda\alpha.\alpha : A \Rightarrow A$:

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

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).

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

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:

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

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 greatest Herbrand models. We note that, not surprisingly (*cf.* [13]), the rule Nu' is inductively unsound. Given a program consisting of just one clause $A \Rightarrow A$, 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 can only be 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 $\mathcal{T}_P$. Note that the principle of structural coinduction is applicable in our proof since the operator $\mathcal{T}_P$ converges in ω steps. This property of $\mathcal{T}_P$ holds only for programs without existential variables, see [12]. This condition is, of course, satisfied for the "type class resolution" fragment that we consider in this paper.

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

Proof. The proof of coinductive soundness of the rule Lp-m can be reconstructed by analogy with the proof of its inductive soundness from the previous section. Thus, we consider only the coinductive soundness of the rule Nu', *i.e.* the case when e has the form $\nu\alpha.e'$, with e' being a formula in the head normal form. This condition implies that e' is a proof term consisting of α and some $\kappa_i, \ldots, \kappa_j$, where $\kappa_i, \ldots, \kappa_j$ are axioms in Φ. This means that the derivation tree leading to the proof of F involved the derivation steps applying $\kappa_i, \ldots, \kappa_j$ to the coinductive hypothesis $(\alpha : \ \Rightarrow F)$ in the environment.

Our proof proceeds by coinduction, with coinductive hypotheses: $\forall n$, if $\Phi \vdash e : F$ holds by derivation using the rules Lp-m and Nu' (the latter applied last), then F is in $\mathcal{T}_P \downarrow (n)$. Consider the construction of the greatest Herbrand model for the program P. The set $\mathcal{T}_P \downarrow (0)$ by definition includes all formulae from $\mathbf{B}_\Sigma$, including all ground instances of F. By the above derivation for F, we may conclude that (a possibly more general form of) F is the head of some clause κ_j. Thus, by the definition of $\mathcal{T}_P \downarrow$, (all ground instances of) F will be contained in $\mathcal{T}_P \downarrow (1)$ (note that by construction $\mathcal{T}_P \downarrow (1) \subseteq \mathcal{T}_P \downarrow (0)$). The same reasoning holds for all clause heads for $\kappa_i, \ldots, \kappa_j$, whose ground instances, too, will be in $\mathcal{T}_P \downarrow (1)$. By the derivation for $\Phi \vdash e : F$, we know that it is sufficient to have F and the formulae contained in the heads of $\kappa_i, \ldots, \kappa_j$ to entail F by Lp-m rule. But then, by the definition of $\mathcal{T}_P \downarrow$, (ground instances of) F will be contained in $\mathcal{T}_P \downarrow (2)$. We now apply the coinduction hypotheses to conclude that the same construction can be repeated for (all ground instances of) F in $\mathcal{T}_P \downarrow (n)$, for all $n > 2$. Thus, (all ground instances of) F will be in $\mathcal{T}_P \downarrow (\omega)$, *i.e.* in the greatest Herbrand model of this program. □

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, 9, 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 p(f(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.* [4] 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).

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

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 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 ? eq(bush(int)) is resolved as follows:

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

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 3. *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. Assume the coinductive hypothesis, that whenever B_1 to B_n are valid in $\mathcal{T}_P \downarrow n$ then also σA is valid in $\mathcal{T}_P \downarrow n$. By definition, $\mathcal{T}_P \downarrow 0 = \mathbf{B}_\Sigma$ and the set $\mathcal{T}_P \downarrow 0$ contains all ground instances of σA and B_1 to B_n. We know from the above conditions that having all instances of σA and B_1 to B_n in the model there is a sequence of iterations of $\mathcal{T}_P$ that infers, for all i, all instances of σC_i, *i.e.* for any substitution τ if, for almost all n, $\{(\tau \circ \sigma)A, \tau B_1, \ldots, \tau B_n\} \subseteq \mathcal{T}_P \downarrow n$ there is m such that, for all i, $(\tau \circ \sigma)C_i \in \mathcal{T}_P \downarrow (n+m)$ and since $(\kappa : C_1, \ldots, C_m \Rightarrow A) \in P$ then $(\tau \circ \sigma)A \in \mathcal{T}_P \downarrow (n+m+1)$. We apply the coinductive hypothesis to conclude that the same holds for almost 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 4. *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. As for the previous lemma, consider the construction of the greatest Herbrand model of the program P. By definition, $\mathcal{T}_P \downarrow 0$ contains all ground instances of A. We know from the above conditions that having all the instances of A and B_1 to B_n in the model there is a sequence of iterations of $\mathcal{T}_P$ that infers all instances of A. We apply the coinductive hypothesis A and by the same argument as above $P \vDash_{coind} B_1, \ldots, B_n \Rightarrow A$. $\square$

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, 3, & 4. $\square$

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.* [4]. Generally, there is a body of work that focused on allowing for infinite data structures in logic programming. Logic programming with rational trees [1, 8] 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 irrational 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 "'Coalgebraic Logic Programming for Type Inference".

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., Schrijvers, T., Pond, A.: Proof relevant corecursive resolution. In: Proc. of FLOPS 2016, Kochi, Japan, March 4-6, 2016 (2016)
5. 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)
6. 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), http://doi.acm.org/10.1145/227699.227700
7. 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)
8. Jaffar, J., Stuckey, P.J.: Semantics of infinite tree logic programming. Theor. Comput. Sci. 46(3), 141–158 (1986)
9. 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)
10. Lämmel, R.: Scrap your boilerplate: prologically! In: Porto, A., López-Fraguas, F.J. (eds.) Proc. of POPL 2009, September 7-9, 2009, Coimbra, Portugal. pp. 7–12. ACM (2009)
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)