Coinductive Soundness of Corecursive Type Class Resolution

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

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

Abstract. Horn clauses and first-order resolution are commonly used to implement *type classes* in Haskell. Several corecursive extensions to type class resolution have recently been proposed, with the goal of allowing (co)recursive dictionary construction where resolution does not terminate. This paper shows, for the first time, that corecursive type class resolution and its 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. We establish incompleteness results for various fragments of the proof system.

Keywords: Resolution · Coinduction · Herbrand models · Type classes · Haskell · Horn clauses

1 Introduction

Type classes can be used to implement ad-hoc polymorphism and overloading in functional languages. The approach 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 any necessary instances of the class, 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 Haskell compiler and involves checking whether all the instance declarations are valid. For example, the following function triggers a check that `Eq (Int, Int)` is a valid instance of 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. The type class instance declarations above could, for example, be viewed as the following two Horn clauses:

Example 1 (Logic program P_{Pair}).

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

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

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

The proof witness $\kappa_1\kappa_2\kappa_2$ (called a "dictionary" in Haskell) is constructed by the Haskell compiler. This is treated internally 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, however, identical. At a 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 that (ii) do not contain existential variables (*i.e.* variables that occur in the bodies but not in the heads of the clauses). At an algorithmic level, (iii) type class resolution corresponds to SLD-resolution in which unification is restricted to term-matching. Assuming there is a clause $B_1, \ldots B_n \Rightarrow A'$, then a query ? A' can be resolved with this clause only if A can be matched against A', *i.e.* if a substitution σ exists such that $A = \sigma A'$. In comparison, SLD-resolution incorporates *unifiers*, as well as *matchers*, *i.e.* it also proceeds to resolve the above query and clause in all the cases where $\sigma A = \sigma A'$ holds.

These restrictions guarantee that type class inference computes the *principal* (most general) type. 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 inference, *i.e.* we prove the query in an implicitly universally quantified form. It is common knowledge that (as with SLD-resolution) type class resolution is *inductively sound*, *i.e.* that it is sound relative to the least Herbrand models of logic programs [12]. Moreover, in Section 3 we establish, for the first time, that it is also *universally inductively sound*, *i.e.* that if a formula A is proved by type class resolution, every ground instance of A is in the least Herbrand model of the given program. In contrast to SLD-resolution, however, type class resolution is *inductively incomplete*, *i.e.* it is incomplete relative to least Herbrand models, even for the class of Horn clauses that is restricted by 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 (by 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)
```

which give rise to the following instance declarations for 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 Haskell compiler:

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

However, inference by resolution does not terminate in this case. Consider the Horn clause representation of the type class instance declarations:

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

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

The non-terminating resolution trace is given by:

$$\underline{\mathtt{eq}(\mathtt{evenList}(\mathtt{int}))} \rightarrow_{\kappa_2} \mathtt{eq}(\mathtt{int}),\ \mathtt{eq}(\mathtt{oddList}(\mathtt{int})) \rightarrow_{\kappa_3} \mathtt{eq}(\mathtt{oddList}(\mathtt{int})) \rightarrow_{\kappa_1} \mathtt{eq}(\mathtt{int}),\ \mathtt{eq}(\mathtt{evenList}(\mathtt{int})) \rightarrow_{\kappa_3} \underline{\mathtt{eq}(\mathtt{evenList}(\mathtt{int}))} \rightarrow_{\kappa_2} \ldots$$

A goal $\mathtt{eq}(\mathtt{evenList}(\mathtt{int}))$ is simplified using the clause κ_2 to goals $\mathtt{eq}(\mathtt{int})$ and $\mathtt{eq}(\mathtt{oddList}(\mathtt{int}))$. The first of these is discarded using the clause κ_3. Resolution continues using κ_1 and κ_3, resulting in the original goal $\mathtt{eq}(\mathtt{evenList}(\mathtt{int}))$. It is easy to see that such a process could continue infinitely and that this goal constitutes a *cycle* (underlined above).

As suggested by Lämmel and Peyton Jones [11], the compiler can obviously terminate the infinite inference process as soon as it detects the underlined cycle. Moreover, it can also construct the corresponding proof witness in a form of a recursive function. For the example above, 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. We say that the proof is given by *corecursive type class resolution*.

Corecursive type class resolution is not inductively sound. For example, the formula $\mathtt{eq}(\mathtt{evenList}(\mathtt{int}))$ is not in the least Herbrand model of the corresponding logic program. However, as we prove in Section 4, it is *(universally) coinductively sound*, *i.e.* it is sound relative to the greatest Herbrand models. For example, $\mathtt{eq}(\mathtt{evenList}(\mathtt{int}))$ is in the greatest Herbrand model of the program $P_{EvenOdd}$. Similarly to the inductive case, corecursive type class resolution is coinductively incomplete. Consider the clause $\kappa_{inf} : \mathtt{p}(x) \Rightarrow \mathtt{p}(\mathtt{f}(x))$. This clause may be given an interpretation by the greatest (complete) Herbrand models. However, corecursive type class resolution does not yield infinite proofs.

Unfortunately, this simple method of cycle detection does not work for all non-terminating programs. Consider the following example, which defines a data type `Bush` (for bush trees), and its corresponding instance for `Eq`:

```
data Bush a = Nil | Cons a (Bush (Bush a))
instance Eq a, Eq (Bush (Bush a)) ⇒ Eq (Bush a) where { ... }
```

Here, type class resolution does not terminate. However, it does not exhibit cycles either. Consider the Horn clause translation of the problem:

Example 3 (Logic program P_{Bush}).

$$\kappa_1 : \quad \Rightarrow \mathtt{eq}(\mathtt{int})$$
$$\kappa_2 : \ \mathtt{eq}(x),\ \mathtt{eq}(\mathtt{bush}(\mathtt{bush}(x))) \Rightarrow \mathtt{eq}(\mathtt{bush}(x))$$

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

$$\mathtt{eq}(\mathtt{bush}(\mathtt{int})) \rightarrow_{\kappa_2} \mathtt{eq}(\mathtt{int}), \mathtt{eq}(\mathtt{bush}(\mathtt{bush}(\mathtt{int})) \rightarrow_{\kappa_1} \ldots \rightarrow_{\kappa_2}$$
$$\mathtt{eq}(\mathtt{bush}(\mathtt{int})), \mathtt{eq}(\mathtt{bush}(\mathtt{bush}(\mathtt{bush}(\mathtt{int}))) \rightarrow_{\kappa_1} \ldots$$

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 recursive proof witness construction. Implicative queries require the language of proof terms to be extended with λ-abstraction. 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 to our program, we obtain a proof of $\mathtt{eq}(\mathtt{bush}(\mathtt{int}))$ by applying κ_3 to κ_1. In 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: whether 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. Namely, we determine that proofs that are obtained by extending the proof context with coinductively proven Horn clauses (such as κ_3 above) are coinductively sound but inductively unsound. This result completes our study of the semantic properties of corecursive type class resolution. Sections 3 and 5 summarise our arguments concerning the inductive and coinductive incompleteness of corecursive type class resolution.

Contributions. 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 reapplied 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, however, 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 the technical sections that follow. In this way, we highlight which program transformation methods can be soundly borrowed from existing work on corecursive resolution. We will use the formulation of corecursive type class resolution given by Fu *et al.* [5]. This extends Howard's simply-typed λ-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.

2 Preliminaries

This section describes our notation and defines the models that we will 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 of which possess 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).

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

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, restriction (ii) from 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 also occurs in A. We use the common term *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 $\mathbf{B}_\Sigma$.*

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

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 $\mathcal{T}_P$.*

Lloyd [12] introduces the operators $\downarrow$ and $\uparrow$ and proves that $\mathcal{T}_P \downarrow \omega$ gives the greatest Herbrand model of P, and that $\mathcal{T}_P \uparrow \omega$ gives the least Herbrand model of P. We will use these constructions in our own 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 the 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 $P \vDash_{coind} \sigma A$

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 that correspond to the given logic program. This gives rise to the following syntax, in addition to that of 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).

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

We use the notation κ with indices 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. Given 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.* it contains no free

variables. A proof term e is in *guarded head normal form* (denoted gHNF(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 known from the context or are unimportant. The intention of the above definition is to interpret logic programs, seen as sets of Horn formulae, as types. Example 1 shows how the 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 $B_1^i, \ldots, B_k^i \Rightarrow A^i$, 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, $\kappa_i : B_1^i, \ldots, B_k^i \Rightarrow A^i$.*

Revisiting 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. Restriction (i) from 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 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 (which are both sound with respect to the greatest Herbrand models).

3 Inductive Soundness of Type Class Resolution

This section describes the inductive fragment of the calculus for the extended type class resolution that was introduced by Fu *et al.* [5]. We reconstruct the standard theorem of universal inductive soundness for the resolution rule. We consider an extended version of type class resolution, working with queries given by Horn formulae, rather than just atomic formulae. We show that the resulting proof system is inductively sound, but coinductively unsound; we also show that it is incomplete. Based on these results, we discuss the program transformation methods that can arise.

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 $\texttt{eq}(\texttt{pair}(\texttt{int},\ \texttt{int}))$ correspond to the following derivation tree:

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

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. We now discuss the other rule that is present in the work of Fu *et al.* [5], i.e the rule that allows us to prove Horn formulae:

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 formulae 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 $A \Rightarrow C$:

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

When there is no label on the right-hand side of an inference step, inference uses the Lp-m rule. 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. By induction on construction of $\mathcal{M}_P$. □

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. □

Inductive Completeness and Incompleteness of the Proof System LP-M + LAM**.** In principle, one can consider two different variants. Extending the standard results of [12], our first formulation is:

Inductive Completeness-1: *if a ground atomic formula F is in $\mathcal{M}_P$, then $\Phi_P \vdash e : F$ is in the* LP-M + LAM *proof system.*

Such a result can be proved, as in [12], by straightforward induction on the construction of $\mathcal{M}_P$. Such a proof will be based solely on the properties of the rule LP-M and on the properties of the semantic operator $\mathcal{T}_P$ that is used to construct the least Herbrand models. An alternative formulation of the completeness result, this time involving implicative formulae and hence the rule LAM in the proof, would be:

Inductive Completeness-2: *if $\mathcal{M}_P \models_{ind} F$ then $\Phi_P \vdash e : F$ is in the* LP-M + LAM *proof system.*

However, this result would not hold for either system LP-M or LP-M + LAM. Consider the following examples.

Example 6. Let Σ be a signature consisting of a unary predicate symbol $\mathtt{A}$, a unary function symbol $\mathtt{f}$ and a constant function symbol g. Let P_6 be a program given by the following axiom environment:

$$\kappa_1 : \ \Rightarrow \mathtt{A}(\mathtt{f}(x))$$

$$\kappa_2 : \ \Rightarrow \mathtt{A}(\mathtt{g})$$

The least Herbrand model of P_6 is $\mathcal{M}_{P_6} = \{A(g), A(f(g)), A(f(f(g))), \dots\}$. Therefore, $P \models_{ind} \mathtt{A}(x)$. However, neither κ_1 nor κ_2 matches $\mathtt{A}(x)$ and there is thus no way to construct a proof term e satisfying:

$$\frac{\cdots}{P \vdash e : \mathtt{A}(x)}\ \text{LP-M}$$

We demonstrate the incompleteness of the proof system LP-M + LAM through the following example:

Example 7. Let Σ be a signature consisting of the unary predicate symbols $\mathtt{A}$ and $\mathtt{B}$, and a constant function symbol $\mathtt{f}$. Consider a program P_7 given by the following axiom environment:

$$\kappa_1 : \ \Rightarrow \mathtt{A}(\mathtt{f})$$

$$\kappa_2 : \ \Rightarrow \mathtt{B}(\mathtt{f})$$

The least Herbrand model is $\mathcal{M}_{P_7} = \{A(f), B(f)\}$. Therefore $P \models_{ind} \mathtt{B}(x) \Rightarrow \mathtt{A}(x)$. However, any proof of $\mathtt{B}(x) \Rightarrow \mathtt{A}(x)$ needs to show that:

$$\frac{\dfrac{\cdots}{(P, \alpha : \ \Rightarrow \mathtt{B}(x)) \vdash e : \mathtt{A}(x)}}{P \vdash \lambda\alpha.e : \mathtt{B}(x) \Rightarrow \mathtt{A}(x)}\ \text{LAM}$$

where e is a proof term. This proof will not succeed since no axiom or hypothesis matches $\mathtt{A}(x)$.

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 the formula $A \Rightarrow A$:

Example 8. Using the Lam rule, one can prove $\emptyset \vdash \lambda\alpha.\alpha : A \Rightarrow A$:

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

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 Coinductive Soundness of Corecursive Type Class Resolution

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 may be 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—*i.e.* a *(co-)recursive dictionary.*

Definition 8 (Coinductive type class resolution).

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

The side condition of Nu' requires the proof witness to be in guarded 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 guarded head normal form terms of the form $\kappa\ \underline{e}$.

Example 9. 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:

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

Note that we abbreviate repeated formulae in the environment using an underscore. We will use this notation in the rest of the paper.

We can now discuss the coinductive soundness of the NU' rule, *i.e.* its soundness relative to the greatest Herbrand models. We note that, not surprisingly (*cf.* [13]), the NU' rule 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 of this will be similar to, albeit a lot simpler than, that in the above example). However, A is not in the least Herbrand model of this program. Similarly, the formula `eq(oddList(int))` that was proved above is also not inductively sound. Thus, the coinductive fragment of the extended corecursive resolution is only coinductively sound. When proving the coinductive soundness of the NU' rule, we must 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 through the construction of the least Herbrand models by iterations of $\mathcal{T}_P$. Here, we give an analogous result, where coinductive soundness is proved 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 Section 1 implies that the $\mathcal{T}_P$ operator converges in at most ω steps. We will capitalise on this fact. The essence of the coinductive soundness of 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, ..., 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 $P \vDash_{coind} \sigma A$.*

The proof of the lemma is similar to the proof of the Lemma 4 in the next section and we do not state it here. 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 the greatest Herbrand models rather than the greatest *complete* Herbrand models. The latter is more common in the literature on coinduction in logic programming [10, 12, 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. However, its greatest complete Herbrand model contains the infinite formula $\mathtt{p}(\mathtt{f}(\mathtt{f}(...))$. 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 a universal interpretation, *i.e.* that they hold for all finite instances of queries. Therefore, we never need to talk about programs for which only one infinite instance of a query is valid.

5 Coinductive Soundness of Extended Corecursive Type Class 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).

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

The side condition of the Nu rule requires the proof witness to be in guarded head normal form. However, unlike coinductive type class resolution, extended coinductive type class resolution also uses the Lam rule and a guarded head normal term is of the more general form $\lambda\underline{\alpha}.\kappa\underline{e}$ for a 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.*

Furthermore, 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 ? $\mathtt{eq}(\mathtt{bush}(\mathtt{int}))$ where no cycles arise and thus the query cannot be resolved by coinductive type class resolution.

Example 10. Recall the program P_{Bush} we defined in the Example 3. The query ? $\mathtt{eq}(\mathtt{bush}(\mathtt{int}))$ is resolved as follows:

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

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 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 grounding 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, it follows from the monotonicity of the operator itself, and from the assumptions made by the lemma 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 then 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 if $\mathcal{T}_P \downarrow (n+1)$. 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 model. Hence $P \vDash_{coind} B_1, \ldots, B_n \Rightarrow A$. □

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 $P \vDash_{coind} B_1, \ldots, B_n \Rightarrow A$.*

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. □

Coinductive Incompleteness of the Proof System LP-M + LAM + NU**.** In Section 3, we considered two ways of stating inductive completeness of type class resolution. We state the corresponding result for the coinductive case here. As both the notions of completeness are shown not to hold we discuss them in the reversed order than the inductive completeness, first the more general case and then the more restricted one:

Coinductive Completeness-2: *if* $\mathcal{M}'_P \vDash_{coind} F$ *then* $\Phi_P \vdash e : F$ *in the* LP-M + LAM + NU *proof system.*

Recall Examples 6 and 7, and the programs P_6 and P_7. We demonstrated that, in general, there are formulae that are valid in $\mathcal{M}_P$ but do not have a proof in P. The same two examples will serve our purpose here. For example, the greatest Herbrand model of the program P_6 is $\mathcal{M}'_P = \mathcal{M}_P = \{A(g), A(f(g)), A(f(f(g))), \ldots\}$. Therefore, for an atomic formula $\mathtt{A}(x)$, $P \vDash_{coind} \mathtt{A}(x)$. However, it is impossible to construct a proof:

$$\frac{\cdots}{P \vdash e : \mathtt{A}(x)}\ \text{LP-M}$$

The rules LP-M and LAM are not applicable for the same reasons as in the inductive case and the rule NU is not applicable since $\mathtt{A}(x)$ is not a Horn formula.

Moreover, a more restricted formulation in the traditional style of Lloyd [12] does not improve the situation:

Coinductive Completeness-1: *if a ground atomic formula* F *is in* $\mathcal{M}'_P$*, then* $\Phi_P \vdash e : F$ *in the* LP-M + LAM + NU *proof system.* Such a result does not hold, since there exist coinductive logic programs that define corecursive schemes that cannot be captured in this proof system. Consider the following example [5]:

Example 11. Let Σ be a signature with a binary predicate symbol D, a unary function symbol s and a constant function symbol z. Consider a program P_{11} with the signature Σ given by the following axiom environment:

$$\kappa_1 :\ \mathtt{D}(x, \mathtt{s}(y)) \Rightarrow \mathtt{D}(\mathtt{s}(x), y)$$

$$\kappa_2 :\ \mathtt{D}(\mathtt{s}(x), \mathtt{z}) \Rightarrow \mathtt{D}(\mathtt{z}, x)$$

Let us denote a term $\mathtt{s}(\mathtt{s}(\ldots \mathtt{s}(x) \ldots))$ where the symbol $\mathtt{s}$ is applied i-times as $\mathtt{s}^i(x)$. By observing the construction of $\mathcal{M}'_P$ we can see that, for all i, if $\mathtt{D}(\mathtt{z}, \mathtt{s}^i(x))$ then $\mathtt{D}(\mathtt{s}^i(x), \mathtt{z}) \in \mathcal{M}'_P$ and also $\mathtt{D}(\mathtt{z}, \mathtt{s}^{i-1}(\mathtt{x})) \in \mathcal{M}'_P$. Therefore $\mathtt{D}(\mathtt{z}, \mathtt{z}) \in \mathcal{M}'_P$. However, there is no proof of $\mathtt{D}(\mathtt{z}, \mathtt{z})$ since any number of proof steps resulting from the use of LP-M generates yet another ground premise that is different from all previous premises. Consequently, the proof cannot be closed by NU. Also, no lemma that would allow for a proof can be formulated; an example of such a lemma would be the above $\mathtt{D}(\mathtt{z}, \mathtt{s}^i(x)) \Rightarrow \mathtt{D}(\mathtt{z}, \mathtt{s}^{i-1}(\mathtt{x}))$. This is a higher order formula and cannot be expressed in a first order Horn clause logic.

Related Program Transformation Methods. We conclude this section with a discussion of program transformation with Horn formulae that are entailed by the rules LAM and NU. From the fact that the NU' rule is inductively unsound, it is clear that using program transformation techniques based on the lemmata that were proved by the LAM and NU rules would also 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* gHNF(e). *Given a formula F', $P \vDash_{coind} F'$ iff $(P, F) \vDash_{coind} F'$.*

The above result is possible thanks to the guarded head normal form condition, since it is then impossible to use a clause $A \Rightarrow A$ that was derived from an empty context by the rule LAM. It is also impossible to make such a derivation within the proof term e itself and to then derive A by the NU rule from $A \Rightarrow A$. The resulting proof term will fail to satisfy the guarded head normal form condition that is required by NU. Since this condition guards against any such cases, we can be sure that this program transformation method is coinductively sound and hence that it 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, corresponding 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 here call *coinductive type class resolution*. The description of the extended calculus of Section 5 was first presented by Fu *et al.* [5]. Generally, there is a body of work that focuses 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 that have infinite proofs.Corecursive resolution, as studied in this paper, is more expressive than co-LP: while also allowing infinite proofs, and closing of coinductive hypotheses is less constrained in our approach.

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. Secondly, we demonstrated, through 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 reapplied to coinductive dialects of logic programming *via* soundness preserving program transformations.

As future work, we intend to extend our analysis of Horn-clause resolution to Horn clauses with existential variables and existentially quantified goals. We believe that such resolution accounts to type inference for other language constructs than type classes, namely type families and algebraic data types.

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

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 (2011)
4. Fu, P., Komendantskaya, E.: Operational semantics of resolution and productivity in horn clause logic. Formal Aspects of Computing pp. 1–22 (2016)
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 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 (2015)
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 (2005)
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)