Flexible coinductive logic programming

FRANCESCO DAGNINO, DAVIDE ANCONA, ELENA ZUCCA
DIBRIS, University of Genova
(*e-mail:* francesco.dagnino@dibris.unige.it,{davide.ancona,elena.zucca}@unige.it)

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

Recursive definitions of predicates are usually interpreted either inductively or coinductively. Recently, a more powerful approach has been proposed, called *flexible coinduction*, to express a variety of intermediate interpretations, necessary in some cases to get the correct meaning. We provide a detailed formal account of an extension of logic programming supporting flexible coinduction. Syntactically, programs are enriched by *coclauses*, clauses with a special meaning used to tune the interpretation of predicates. As usual, the declarative semantics can be expressed as a fixed point which, however, is not necessarily the least, nor the greatest one, but is determined by the coclauses. Correspondingly, the operational semantics is a combination of standard SLD resolution and coSLD resolution. We prove that the operational semantics is sound and complete with respect to declarative semantics restricted to finite comodels.

KEYWORDS: coinduction, operational semantics, declarative semantics, soundness, completeness

1 Introduction

Standard inductive and coinductive semantics of logic programs sometimes are not enough to properly define predicates on possibly infinite terms (Simon et al. 2007; Ancona 2013).

Consider the logic program in Fig. 1, defining some predicates on lists of numbers represented with the standard Prolog syntax. For simplicity, we consider built-in numbers, as in Prolog.

In standard logic programming, terms are inductively defined, that is, are finite, and predicates are inductively defined as well. In the example program, only finite lists are considered, such as, e.g., `[1|[2|[]]]`, and the three predicates are correctly defined on such lists.

Coinductive logic programming (coLP) (Simon 2006) extends standard logic programming with the ability of reasoning about infinite objects and their properties. Terms are coinductively defined, that is, can be infinite, and predicates are coinductively defined as well. In the example,

$$
\begin{array}{lll}
\mathit{all\_pos}([\,]) & \leftarrow & \\
\mathit{all\_pos}([N|L]) & \leftarrow & N > 0,\ \mathit{all\_pos}(L). \\
\mathit{member}(X,[X|\_]) & \leftarrow & \\
\mathit{member}(X,[Y|L]) & \leftarrow & X \neq Y,\ \mathit{member}(X,L). \\
\mathit{maxElem}([N],N) & \leftarrow & \\
\mathit{maxElem}([N|L],M) & \leftarrow & \mathit{maxElem}(L,M_1),\ M \text{ is } \max(N,M_1).
\end{array}
$$

Fig. 1. An example of logic program: $\mathit{all\_pos}(l)$ succeeds iff l contains only positive numbers, $\mathit{member}(x,l)$ succeeds iff x is in l, $\mathit{maxElem}(l,x)$ succeeds iff x is the greatest number in l.

also infinite lists, such as `[1|[2|[3|[4|...]]]]`, are considered, and the coinductive interpretation of *all_pos* gives the expected meaning on such lists. However, this is not the case for the other two predicates: for *member* the correct interpretation is still the inductive one, as in the coinductive semantics $\mathit{member}(x,l)$ always succeeds for an infinite list l. For instance, for L the infinite list of 0's, $\mathit{member}(1,L)$ has an infinite proof tree where for each node we apply the second clause. Therefore, these two predicates cannot coexist in the same program, as they require two different interpretations.[1]

The predicate *maxElem* shows an even worse situation. The inductive semantics again does not work on infinite lists, but also the coinductive one is not correct: $\mathit{maxElem}(l,n)$ succeeds whenever n is greater than all the elements of l. The expected meaning lies between the inductive and the coinductive semantics, hence, to get it, we need something beyond standard semantics.

Recently, in the more general context of inference systems (Aczel 1977), *flexible coinduction* has been proposed by Dagnino and Ancona et al. (2017; 2017b; 2019), a generalisation able to express a variety of intermediate interpretations. As we recall in Section 2, clauses of a logic program can be seen as meta-rules of an inference system where judgments are ground atoms. Inference rules are ground instances of clauses, and a ground atom is valid if it has a finite proof tree in the inductive interpretation, a possibly infinite proof tree in the coinductive one.

Guided by this abstract view, which provides solid foundations, we develop an extension of logic programming supporting flexible coinduction.

Syntactically, programs are enriched by *coclauses*, which resemble clauses but have a special meaning used to tune the interpretation of predicates. By adding coclauses, we can obtain a declarative semantics intermediate between the inductive and the coinductive one. Standard (inductive) and coinductive logic programming are subsumed by a particular choice of coclauses. Correspondingly, operational semantics is a combination of standard SLD resolution (Lloyd 1987; Apt 1997) and coSLD resolution as introduced by Simon et al. (2006; 2006; 2007). More precisely, as in coSLD resolution, it keeps trace of already considered goals, called *coinductive hypotheses*.[2] However, when a goal unifying with a coinductive hypothesis is found, rather than being considered successful as in coSLD resolution, its standard SLD resolution is triggered in the program where also coclauses are considered. Our main result is that such operational semantics is *sound and complete* with respect to the declarative one restricted to regular proof trees.

An important additional result is that the operational semantics is not incidental, but, as the declarative semantics, turns out to correspond to a precise notion on the inference system denoted by the logic program. Indeed, as detailed in a companion paper of Dagnino (2020), given an inference system, we can always construct another one, with judgments enriched by circular hypotheses, which, interpreted inductively, is equivalent to the *regular* interpretation of the original inference system. In other words, there is a canonical way to derive a (semi-)algorithm to show that a judgment has a regular proof tree, and our operational semantics corresponds to this algorithm. This more abstract view supports the reliability of the approach, and, indeed, the proof of equivalence with declarative semantics can be nicely done in a modular way, that is, by relying on a general result proved by Dagnino (2020).

After basic notions in Section 2, in Section 3 we introduce logic programs with coclauses and

[1] To overcome this issue, *co-logic programming* (Simon et al. 2007) marks predicates as either inductive or coinductive. The declarative semantics, however, becomes quite complex, because stratification is needed.

[2] We prefer to mantain this terminology, inherited from coSLD resolution, even though not corresponding to the proof theoretic sense.

their declarative semantics, and in Section 4 the operational semantics. We provide significant examples in Section 5, the results in Section 6, related work and conclusive remarks in Section 7.

2 Logic programs as inference systems

We recall basic concepts about inference systems (Aczel 1977), and present (standard inductive and coinductive) logic programming (Lloyd 1987; Apt 1997; Simon 2006; Simon et al. 2006; Simon et al. 2007) as a particular instance of this general semantic framework.

Inference systems Assume a set $\mathcal{U}$ called *universe* whose elements j are called *judgements*. An *inference system* $\mathfrak{I}$ is a set of *(inference) rules*, which are pairs $\langle Pr, c\rangle$, also written $\frac{Pr}{c}$, with $Pr \subseteq \mathcal{U}$ set of *premises*, and $c \in \mathcal{U}$ *conclusion*. We assume inference systems to be *finitary*, that is, rules have a finite set of premises. A *proof tree* (a.k.a. *derivation*) in $\mathfrak{I}$ is a tree with nodes (labelled) in $\mathcal{U}$ such that, for each j with set of children Pr, there is a rule $\langle Pr, j\rangle$ in $\mathfrak{I}$. A proof tree for j is a proof tree with root j. The *inference operator* $F_{\mathfrak{I}} : \wp(\mathcal{U}) \to \wp(\mathcal{U})$ is defined by:

$$F_{\mathfrak{I}}(X) = \{j \in \mathcal{U} \mid \langle Pr, j\rangle \in \mathfrak{I} \text{ for some } Pr \subseteq X\}$$

A set $X \subseteq \mathcal{U}$ is *closed* if $F_{\mathfrak{I}}(X) \subseteq X$, *consistent* if $X \subseteq F_{\mathfrak{I}}(X)$, a *fixed point* if $X = F_{\mathfrak{I}}(X)$.

An *interpretation* of an inference system $\mathfrak{I}$ is a set of judgements, that is, a subset of the universe $\mathcal{U}$. The two standard interpretations, the inductive and the coinductive one, can be defined in either model-theoretic or proof-theoretic terms (Leroy and Grall 2009).

- The *inductive interpretation* $\mu[\![\mathfrak{I}]\!]$ is the intersection of all closed sets, that is, the least closed set or, equivalently, the set of judgements with a finite proof tree.
- The *coinductive interpretation* $\nu[\![\mathfrak{I}]\!]$ is the union of all consistent sets, that is, the greatest consistent set, or, equivalently, the set of judgements with an arbitrary (finite or not) proof tree.

By the fixed point theorem (Tarski 1955), both $\mu[\![\mathfrak{I}]\!]$ and $\nu[\![\mathfrak{I}]\!]$ are fixed points of $F_{\mathfrak{I}}$, the least and the greatest one, respectively. We will write $\mathfrak{I} \vdash_\mu j$ for $j \in \mu[\![\mathfrak{I}]\!]$ and $\mathfrak{I} \vdash_\nu j$ for $j \in \nu[\![\mathfrak{I}]\!]$.

Logic programming Assume a *first order signature* $\langle \mathcal{P}, \mathcal{F}, \mathcal{V}\rangle$ with $\mathcal{P}$ set of *predicate symbols* p, $\mathcal{F}$ set of *function symbols* f, and $\mathcal{V}$ countably infinite set of *variable symbols* X (*variables* for short). Each symbol comes with its *arity*, a natural number denoting the number of arguments. Variables have arity 0. A function symbol with arity 0 is a *constant*.

Terms t, s, r are (possibly infinite) trees with nodes labeled by function or variable symbols, where the number of children of a node is the symbol arity[3]. *Atoms* A, B, C are (possibly infinite) trees with the root labeled by a predicate symbol and other nodes by function or variable symbols, again accordingly with the arity. Terms and atoms are *ground* if they do not contain variables, and *finite* (or *syntactic*) if they are finite trees. *(Definite) clauses* have shape $A \leftarrow B_1, \dots, B_n$ with $n \geq 0, A, B_1, \dots, B_n$ finite atoms. A clause where $n = 0$ is called a *fact*. A *(definite) logic program* P is a finite set of clauses.

Substitutions θ, σ are partial maps from variables to terms with a finite domain. We write $t\theta$ for the application of θ to a term t, call $t\theta$ an *instance* of t, and analogously for atoms, set of

[3] For a more formal definition based on paths see, e.g., the work of Ancona and Dovier (2015).

atoms, and clauses. A substitution θ is *ground* if, for all $X \in \mathsf{dom}(\theta)$, $\theta(X)$ is ground, *syntactic* if, for all $X \in \mathsf{dom}(\theta)$, $\theta(X)$ is a finite (syntactic) term.

In order to see a logic program P as an inference system, we fix as universe the *complete Herbrand base* HB_∞, that is, the set of all (finite and infinite) ground atoms[4]. Then, P can be seen as a set of *meta-rules* defining an inference system $\|P\|$ on HB_∞. That is, $\|P\|$ is the set of ground instances of clauses in P, where $A \leftarrow B_1,\ldots,B_n$ is seen as an inference rule $\langle\{B_1,\ldots,B_n\},A\rangle$. In this way, typical notions related to declarative semantics of logic programs turn out to be instances of analogous notions for inference systems. Notably, the (one step) inference operator associated to a program $T_P : \wp(\mathsf{HB}_\infty) \to \wp(\mathsf{HB}_\infty)$, defined by:

$$T_P(I) = \{A \in \mathsf{HB}_\infty \mid (A \leftarrow B_1,\ldots,B_n) \in \|P\|, \{B_1,\ldots,B_n\} \subseteq I\}$$

is exactly $F_{\|P\|}$. An *interpretation* (a set $I \subseteq \mathsf{HB}_\infty$) is a *model* of a program P if $T_P(I) \subseteq I$, that is, it is closed with respect to $\|P\|$. Dually, an interpretation I is a *comodel* of a program P if $I \subseteq T_P(I)$, that is, it is consistent with respect to $\|P\|$. Then, the inductive declarative semantics of P is the least model of P and the coinductive declarative semantics[5] is the greatest comodel of P. These two semantics coincide with the inductive and coinductive interpretations of $\|P\|$, hence we denote them by $\mu[\![P]\!]$ and $\nu[\![P]\!]$, respectively.

3 Coclauses

We introduce logic programs with coclauses and define their declarative semantics. Consider again the example in Fig. 1 where, as discussed in the Introduction, each predicate needed a different kind of interpretation.

As shown in the previous section, the above logic program can be seen as an inference system. In this context, *flexible coinduction* has been proposed (Dagnino 2017; Ancona et al. 2017b; Dagnino 2019), a generalisation able to overcome these limitations. The key notion are *corules*, special inference rules used to control the semantics of an inference system. More precisely, a *generalized inference system*, or *inference system with corules*, is a pair of inference systems $\langle\mathfrak{I}, \mathfrak{I}_{\mathsf{co}}\rangle$, where the elements of $\mathfrak{I}_{\mathsf{co}}$ are called corules. The interpretation of $\langle\mathfrak{I}, \mathfrak{I}_{\mathsf{co}}\rangle$, denoted by $\nu_{\mathsf{fl}}[\![\mathfrak{I},\mathfrak{I}_{\mathsf{co}}]\!]$, is constructed in two steps.

- first, we take the inductive interpretation of the union $\mathfrak{I} \cup \mathfrak{I}_{\mathsf{co}}$, that is, $\mu[\![\mathfrak{I} \cup \mathfrak{I}_{\mathsf{co}}]\!]$,
- then, the union of all sets, consistent with respect to $\mathfrak{I}$, which are subsets of $\mu[\![\mathfrak{I} \cup \mathfrak{I}_{\mathsf{co}}]\!]$, that is, the largest consistent subset of $\mu[\![\mathfrak{I} \cup \mathfrak{I}_{\mathsf{co}}]\!]$.

In proof-theoretic terms, $\nu_{\mathsf{fl}}[\![\mathfrak{I},\mathfrak{I}_{\mathsf{co}}]\!]$ is the set of judgements with an arbitrary (finite or not) proof tree in $\mathfrak{I}$, whose nodes all have a finite proof tree in $\mathfrak{I}\cup\mathfrak{I}_{\mathsf{co}}$. Essentially, by corules we filter out some, undesired, infinite proof trees. Dagnino (2019) proved that $\nu_{\mathsf{fl}}[\![\mathfrak{I},\mathfrak{I}_{\mathsf{co}}]\!]$ is a fixed point of $F_{\mathfrak{I}}$.

To introduce flexible coinduction in logic programming, first we slightly extend the syntax by introducing *(definite) coclauses*, written $A \Leftarrow B_1,\ldots,B_n$, where A, B_1, …, B_n are finite atoms. A coclause where n = 0 is called a *cofact*. Coclauses syntactically resemble clauses, but are used in a special way, like corules for inference systems. More precisely, we have the following definition:

[4] Traditionally (Lloyd 1987), the inductive declarative semantics is restricted to finite atoms. We define also the inductive semantics on the complete Herbrand base in order to work in a uniform context.

[5] Introduced (Simon 2006; Simon et al. 2006) to properly deal with predicates on infinite terms.

Definition 3.1
A *logic program with coclauses* is a pair $\langle P, P_{\mathsf{co}}\rangle$ where P and P_{co} are sets of clauses. Its *declarative semantics*, denoted by $\nu_{\mathsf{fl}}[\![P, P_{\mathsf{co}}]\!]$, is the largest comodel of P which is a subset of $\mu[\![P \cup P_{\mathsf{co}}]\!]$.

In other words, the declarative semantics of $\langle P, P_{\mathsf{co}}\rangle$ is the coinductive semantics of P where, however, clauses are instantiated only on elements of $\mu[\![P \cup P_{\mathsf{co}}]\!]$. Note that this is the interpretation of the generalized inference system $\langle \|P\|, \|P_{\mathsf{co}}\|\rangle$.

Below is the version of the example in Fig. 1, equipped with coclauses.

$$
\begin{array}{lcl}
\mathit{all\_pos}([\,]) & \leftarrow & \\
\mathit{all\_pos}([N|L]) & \leftarrow & N > 0,\ \mathit{all\_pos}(L). \\
\mathit{all\_pos}(\_) & \Leftarrow & \\
\mathit{member}(X,[X|\_]) & \leftarrow & \\
\mathit{member}(X,[Y|L]) & \leftarrow & X \neq Y,\ \mathit{member}(X,L). \\
\mathit{maxElem}([N],N) & \leftarrow & \\
\mathit{maxElem}([N|L],M) & \leftarrow & \mathit{maxElem}(L,M_1),\ M \text{ is } \max(N,M_1). \\
\mathit{maxElem}([N|\_],N) & \Leftarrow &
\end{array}
$$

In this way, all the predicate definitions are correct w.r.t. the expected semantics:

- *all_pos* has coinductive semantics, as the coclause allows any infinite proof trees.
- *member* has inductive semantics, as without coclauses no infinite proof tree is allowed.
- *maxElem* has an intermediate semantics, as the coclause allows only infinite proof trees where nodes have shape $\mathit{maxElem}(l,x)$ with x an element of l.

As the example shows, coclauses allow the programmer to mix inductive and coinductive predicates, and to correctly define predicates which are neither inductive, nor purely coinductive. For this reason we call this paradigm *flexible coinductive logic programming.* Note that, as shown for inference systems with corules (Dagnino 2017; Ancona et al. 2017b; Dagnino 2019), inductive and coinductive semantics are particular cases. Indeed, they can be recovered by special choices of coclauses: the former is obtained when no coclause is specified, the latter when each atom in HB_∞ is an instance of the head of a cofact.

4 Big-step operational semantics

In this section we define an operational counterpart of the declarative semantics of logic programs with coclauses introduced in the previous section.

As in standard coLP (Simon 2006; Simon et al. 2006; Simon et al. 2007), to represent possibly infinite terms we use finite sets of equations between finite (syntactic) terms. For instance, the equation `L = [1,2|L]` represents the infinite list `[1,2,1,2,...]`.

Since the declarative semantics of logic programs with coclauses is a combination of inductive and coinductive semantics, their operational semantics combines standard SLD resolution (Lloyd 1987; Apt 1997) and coSLD resolution (Simon 2006; Simon et al. 2006; Simon et al. 2007). It is presented, rather than in the traditional small-step style, in big-step style, as introduced by Ancona and Dovier (2015). This style turns out to be simpler since coinductive hypotheses (see below) can be kept local. Moreover, it naturally leads to an interpreter, and makes it simpler to prove its correctness with respect to declarative semantics (see the next section).

$$\text{(EMPTY)}\ \frac{}{\langle P,P_{\mathsf{co}}\rangle;S \Vdash \langle \varepsilon;E\rangle \Rightarrow E} \qquad \text{(CO-HYP)}\ \frac{\langle P\cup P_{\mathsf{co}},\emptyset\rangle;\emptyset \Vdash \langle A;E_1\cup E_{A,B}\rangle \Rightarrow E_2 \quad \langle P,P_{\mathsf{co}}\rangle;S \Vdash \langle G_1,G_2;E_2\rangle \Rightarrow E_3}{\langle P,P_{\mathsf{co}}\rangle;S \Vdash \langle G_1,A,G_2;E_1\rangle \Rightarrow E_3}\ \begin{array}{l} B\in S \\ E_1\vdash A=B \\ P_{\mathsf{co}}\neq\emptyset \end{array}$$

$$\text{(STEP)}\ \frac{\langle P,P_{\mathsf{co}}\rangle;S\cup\{A\} \Vdash \langle C_1,\ldots,C_n;E_1\cup E_{A,B}\rangle \Rightarrow E_2 \quad \langle P,P_{\mathsf{co}}\rangle;S \Vdash \langle G_1,G_2;E_2\rangle \Rightarrow E_3}{\langle P,P_{\mathsf{co}}\rangle;S \Vdash \langle G_1,A,G_2;E_1\rangle \Rightarrow E_3}\ \begin{array}{l} \theta \text{ fresh renaming} \\ B\theta\leftarrow C_1\theta,\ldots,C_n\theta\in P \\ E_1\vdash A=B \end{array}$$

Fig. 2. Big-step operational semantics

We introduce some notations. First of all, in this section we assume atoms and terms to be finite (syntactic). A *goal* is a pair $\langle G;E\rangle$, where G is a finite sequence of atoms. A goal is *empty* if G is the empty sequence, denoted ε. An *equation* has shape $s = t$ where s and t are terms, and we denote by E a finite set of equations.

Intuitively, a goal can be seen as a query to the program and the operational semantics has to compute answers (a.k.a. solutions) to such a query. More in detail, the operational semantics, given a goal $\langle G;E_1\rangle$, provides another set of equations E_2, which represents answers to the goal. For instance, given the previous program, for the goal $\langle$`maxElem(L,M)`;$\{$`L = [1,2|L]`$\}\rangle$, the operational semantics returns the set of equations $\{$`L = [1,2|L],M = 2`$\}$.

The judgment of the operational semantics has shape

$$\langle P,P_{\mathsf{co}}\rangle;S \Vdash \langle G;E_1\rangle \Rightarrow E_2$$

meaning that resolution of $\langle G;E_1\rangle$, under the *coinductive hypotheses* S (Simon et al. 2006), succeeds in $\langle P,P_{\mathsf{co}}\rangle$, producing a set of equations E_2. Set $\mathsf{Var}(t)$ the set of variables in a term, and analogously for atoms, set of atoms, and equations. We assume $\mathsf{Var}(S)\subseteq\mathsf{Var}(E_1)$, modelling the intuition that S keeps track of already considered atoms. This condition holds for the initial judgement, and is preserved by rules in Fig. 2, hence it is not restrictive. Resolution starts with no coinductive hypotheses, that is, the top-level judgment has shape $\langle P,P_{\mathsf{co}}\rangle;\emptyset \Vdash \langle G;E_1\rangle \Rightarrow E_2$.

The operational semantics has two flavours:

- If there are no corules ($P_{\mathsf{co}}=\emptyset$), then the judgment models standard SLD resolution, hence the set of coinductive hypotheses is not significant.
- Otherwise, the judgment models *flexible coSLD resolution*, which follows the same schema of coSLD resolution, in the sense that it keeps track in S of the already considered atoms. However, when an atom A in the current goal unifies with a coinductive hypothesis, rather than just considering A successful as in coSLD resolution, standard SLD resolution of A is triggered in the program $P\cup P_{\mathsf{co}}$, that is, also coclauses can be used.

The judgement is inductively defined by the rules in Fig. 2, which rely on some auxiliary (standard) notions. A *solution* of an equation $s = t$ is a *unifier* of t and s, that is, a substitution θ such that $s\theta = t\theta$. A solution of a finite set of equations E is a solution of all the equations in E and E is *solvable* if there exists a solution of E. Two atoms A and B are *unifiable* in a set of equations E, written $E\vdash A=B$, if $A=p(s_1,\ldots,s_n)$, $B=p(t_1,\ldots,t_n)$ and $E\cup\{s_1=t_1,\ldots,s_n=t_n\}$ is solvable, and we denote by $E_{A,B}$ the set $\{s_1=t_1,\ldots,s_n=t_n\}$.

Rule (EMPTY) states that the resolution of an empty goal succeeds. In rule (STEP), an atom A to be resolved is selected, and a clause of the program is chosen such that A unifies with the head of the clause in the current set of equations. Then, resolution of the original goal succeeds if both the

$$
\text{(S-2)}\ \dfrac{
\text{(S-2)}\ \dfrac{
\text{(C)}\ \dfrac{
\text{(S-2)}\ \dfrac{
\text{(S-3])}\ \dfrac{\text{(MAX)}}{\langle\{1,2,3\},\emptyset\rangle;\emptyset \Vdash \langle \mathtt{mE([2|L],M3),M2=max(1,M3)};eq_{\mathrm{L}},M2{=}M\rangle \Rightarrow eq_{\mathrm{L}},M2{=}M,eqs}
}{\langle\{1,2,3\},\emptyset\rangle;\emptyset \Vdash \langle \mathtt{mE(L,M2)};eq_{\mathrm{L}},M2{=}M\rangle \Rightarrow eq_{\mathrm{L}},M2{=}M,eqs}\ \text{(MAX)}
}{\langle\{1,2\},3\rangle;\mathtt{mE(L,M)} \Vdash \langle \mathtt{mE(L,M2),M1=max(2,M2)};eq_{\mathrm{L}}\rangle \Rightarrow eq_{\mathrm{L}},M2{=}M,eqs,M1{=}2}
}{\langle\{1,2\},3\rangle;\mathtt{mE(L,M)} \Vdash \langle \mathtt{mE([2|L],M1),M=max(1,M1)};eq_{\mathrm{L}}\rangle \Rightarrow eq_{\mathrm{L}},M2{=}M,eqs,M1{=}2 \quad \text{(MAX)}}
}{\langle\{1,2\},3\rangle;\emptyset \Vdash \langle \mathtt{mE(L,M)};eq_{\mathrm{L}}\rangle \Rightarrow eq_{\mathrm{L}},M2{=}M,eqs,M1{=}2,M{=}2}
$$

Fig. 3. Example of resolution

body of the selected clause and the remaining atoms are resolved, enriching the set of equations correspondingly. As customary, the selected clause is renamed using fresh variables, to avoid variable clashes in the set of equations obtained after unification. Note that, in the resolution of the body of the clause, the selected atom is added to the current set of coinductive hypotheses. This is not relevant for standard SLD resolution ($P_{\mathsf{co}} = \emptyset$). However, if $P_{\mathsf{co}} \neq \emptyset$, this allows rule (CO-HYP) to handle the case when an atom A that has to be resolved unifies with a coinductive hypothesis in the current set of equations. In this case, standard SLD resolution of such atom in the program $P \cup P_{\mathsf{co}}$ is triggered, and resolution of the original goal succeeds if both such standard SLD resolution of the selected atom and resolution of the remaining goal succeed.

In Fig. 3 we show an example of resolution. We use the shorter syntax `=max`, abbreviate by eq_{L} the equation `L` $=$ `[1,2|L]`, by eqs the equations $M3{=}2, M2{=}2$, by `mE` the predicate `maxElem`, and by (S), (C) the rules (STEP) and (CO-HYP), respectively. When applying rule (STEP), we also indicate the clause/coclause which has been used: we write 1,2,3 for the two clauses and the coclause for the `maxElem` predicate (the first clause is never used in this example). Finally, to keep the example readable and focus on key aspects, we make some simplifications: notably, (MAX) stands for an omitted proof tree solving atoms of shape `_ is max(_,_)`; morever, equations between lists are implicitly applied.

As final remark, note that flexible coSLD resolution nicely subsumes both SLD and coSLD. The former, as already said, is obtained when the set of coclauses is empty, that is, the program is inductive. The latter is obtained when, for all predicate p of arity n, we have a cofact $p(X_1,\ldots,X_n)$.

5 Examples

In this section we discuss some more sophisticated examples.

∞-regular expressions: We define *∞-regular expressions* on an alphabet Σ, a variant of the formalism defined by Löding and Tollkötter (2016) for denoting languages of finite and infinite words, the latter also called ω-words, as follows:

$$r ::= \emptyset \mid \varepsilon \mid a \mid r_1 \cdot r_2 \mid r_1 + r_2 \mid r^\star \mid r^\omega$$

where $a \in \Sigma$. The syntax of standard regular expressions is extended by r^ω, denoting the ω-power of the language A_r denoted by r. That is, the set of words obtained by concatenating infinitely many times words in A_r. In this way, we can denote also languages containing infinite words.

In Fig. 4 we define the predicate *match*, such that *match*(W,R) holds if the finite or infinite word W, implemented as a list, belongs to the language denoted by R. For simplicity, we consider words over the alphabet $\{0,1\}$.

$$
\begin{array}{lll}
concat([\,],W,W) & \leftarrow & \\
concat([B|W_1],W_2,[B|W_3]) & \leftarrow & concat(W_1,W_2,W_3). \\
concat(W_1,W_2,W_1) & \Leftarrow & \\
match([\,],eps) & \leftarrow & \\
match([0],0) & \leftarrow & \\
match([1],1) & \leftarrow & \\
match(W,cat(R_1,R_2)) & \leftarrow & match(W_1,R_1),\ match(W_2,R_2),\ concat(W_1,W_2,W). \\
match(W,plus(R_1,R_2)) & \leftarrow & match(W,R_1). \\
match(W,plus(R_1,R_2)) & \leftarrow & match(W,R_2). \\
match(W,star(R)) & \leftarrow & match\_star(N,W,R). \\
match([\,],omega(R)) & \leftarrow & match([\,],R). \\
match([B|W],omega(R)) & \leftarrow & match([B|W_1],R),\ match(W_2,omega(R)),\ concat(W_1,W_2,W). \\
match(W,omega(R)) & \Leftarrow & \\
match\_star(0,[\,],R) & \leftarrow & \\
match\_star(s(N),W,R) & \leftarrow & match(W_1,R),\ match\_star(N,W_2,R),\ concat(W_1,W_2,W).
\end{array}
$$

Fig. 4. A logic program for ∞-regular expression recognition.

Concatenation of words needs to be defined coinductively, to correctly work on infinite words as well. Note that, when w_1 is infinite, $w_1 w_2$ is equal to w_1.

On operators of regular expressions, *match* can be defined in the standard way (no coclauses). In particular, the definition for expressions of shape $r^\star$ follows the explicit definition of the $\star$-closure of a language: given a language L, a word w belongs to $L^\star$ iff it can be decomposed as $w_1 \dots w_n$, for some $n \geq 0$, where $n = 0$ means w is empty, and $w_i \in L$, for all $i \in 1..n$. This condition is checked by the auxiliary predicate *match_star*.

To define when a word w matches r^ω we have two cases. If w is empty, then it is enough to check that the empty word matches r, as expressed by the first clause, because concatenating infinitely many times the empty word we get again the empty word. Otherwise, we have to decompose w as $w_1 w_2$ where w_1 is not empty and matches r and w_2 matches r^ω as well, as formally expressed by the second clause, To propertly handle infinite words, we need to concatenate infinitely many non-empty words, hence we need to apply the second clause infinitely many times. The coclause allows all such infinite derivations.

An LTL fragment: In Fig. 5 we define the predicate *sat* s.t. $sat(w,\varphi)$ succeeds iff the ω-word w over the alphabet $\{0,1\}$ satisfies the formula φ of the fragment of the Linear Temporal Logic with the temporal operators *until* (**U**) and *always* (**G**) and the predicate *zero* and its negation[6] *one*. Since $sat([B|W],always(Ph))$ succeeds iff all **infinite** suffixes of $[B|W]$ satisfy formula *Ph*, the coinductive interpretation has to be considered, hence a coclause is needed; for instance, $sat(W_0,always(zero))$, with $W_0 = [0|W_0]$, succeeds because the atom $sat(W_0,always(zero))$ in the body of the clause for *always* unifies[7] with the coinductive hypothesis $sat(W_0,always(zero))$ (see rule (CO-HYP) in Figure 2) and the coclause allows it to succeed w.r.t. standard SLD resolution (indeed, atom $sat(W_0,zero)$ succeeds, thanks to the first fact in the logic program).

Differently to *always*, the interpretation of *until* has to be inductive because $until(\varphi_1,\varphi_2)$

[6] Predicates *true* and *false* could be easily defined as well.
[7] Actually, in this case the atom to be resolved and the coinductive hypothesis are syntactically equal.

$$\begin{array}{lcl}
sat\_exists(0,W,Ph) & \leftarrow & sat(W,Ph).\\
sat\_exists(s(N),[B|W],Ph) & \leftarrow & sat\_exists(N,W,Ph).\\
sat\_all(0,W,Ph) & \leftarrow & \\
sat\_all(s(N),[B|W],Ph) & \leftarrow & sat([B|W],Ph),\ sat\_all(N,W,Ph).\\
sat([0|W],zero) & \leftarrow & \\
sat([1|W],one) & \leftarrow & \\
sat([B|W],always(Ph)) & \leftarrow & sat([B|W],Ph),\ sat(W,always(Ph)).\\
sat(W,always(Ph)) & \Leftarrow & \\
sat([B|W],until(Ph_1,Ph_2)) & \leftarrow & sat\_exists(N,[B|W],Ph_2),\ sat\_all(N,[B|W],Ph_1).
\end{array}$$

Fig. 5. A logic program for satisfaction of an LTL fragment: $sat\_exists(N,W,Ph)$ succeeds iff suffix at N of ω-word W satisfies Ph, $sat\_all(N,W,Ph)$ succeeds iff all suffixes of word W at index $< N$ satisfy Ph, $sat(W,Ph)$ succeeds iff ω-word W satisfies Ph.

succeeds iff φ_2 is satisfied after a **finite** number of steps; for this reason, no coclause is given for this operator; for instance, $sat([1,1,0|W_1],until(one,zero))$ with $W_1 = [1|W_1]$ succeeds w.r.t. standard SLD resolution, while $sat(W_1,until(one,zero))$, $sat(W_1,until(always(one),zero))$, and $sat(W_1,until(always(one),always(zero)))$ fail. The clause for $sat([B|W],until(Ph_1,Ph_2))$ follows the standard definition of satisfaction for the **U** operator: there must exist a suffix of $[B|W]$ at index N satisfying Ph_2 ($sat\_exists(N,[B|W],Ph_2)$) s.t. all suffixes of $[B|W]$ at index less than N satisfy Ph_1 ($sat\_all(N,[B|W],Ph_1)$).

An interesting example concerns the goal $sat([1,1|W_0],until(one,always(zero)))$, where the two temporal operators are mixed together: it succeeds as expected, thanks to the two clauses for *until* and the fact that $sat(W_0,always(zero))$ succeeds, as shown above.

Some of the issues faced in this example are also discussed by Gupta et al. (2011).

Big-step semantics modeling infinite behaviour and observations Defining a big-step operational semantics modelling divergence is a difficult task, especially in presence of observations. Ancona et al. (2018; 2020) show how corules can be successfully employed to tackle this problem, by providing big-step semantics able to model divergence for several variations of the lambda-calculus and different kinds of observations. Following this approach, we present in Fig. 6 a similar example, but simpler, to keep it shorter: a logic program with coclauses defining the big-step semantics of a toy language to output possibly infinite sequences[8] of integers. Expressions are regular terms generated by the following grammar:

$$e ::= skip \mid out\ n \mid seq(e_1,e_2)$$

where *skip* is the idle expression, *out n* outputs n, and $seq(e_1,e_2)$ is the sequential composition. The semantic judgement has shape $e \Rightarrow \langle r,\ s\rangle$, represented by the atom $eval(e,r,s)$, where e is an expression, r is either *end* or *div*, for converging or diverging computations, respectively, and s is a possibly infinite sequence of integers. Clauses for *concat* are pretty standard; in this case the definition is purely inductive (hence, no coclause is needed) since the left operand of concatenation is always a finite sequence. Clauses for *eval* are rather straightforward, but sequential composition $seq(e_1,e_2)$ deserves some comment: if the evaluation of e_1 converges, then the com-

[8] For simplicity we consider only integers, but in fact the definition below allows any term as output.

$$\begin{array}{lll}
\mathit{concat}([\,],S,S) & \leftarrow & \\
\mathit{concat}([N|S_1],S_2,[N|S_3]) & \leftarrow & \mathit{concat}(S_1,S_2,S_3). \\
\mathit{eval}(\mathit{skip},\mathit{end},[\,]) & \leftarrow & \\
\mathit{eval}(\mathit{out}(N),\mathit{end},[N]) & \leftarrow & \\
\mathit{eval}(\mathit{seq}(E_1,E_2),R,S) & \leftarrow & \mathit{eval}(E_1,\mathit{end},S_1),\ \mathit{eval}(E_2,R,S_2),\ \mathit{concat}(S_1,S_2,S). \\
\mathit{eval}(\mathit{seq}(E_1,E_2),\mathit{div},S) & \leftarrow & \mathit{eval}(E_1,\mathit{div},S). \\
\mathit{eval}(E,\mathit{div},[\,]) & \Leftarrow & \\
\mathit{eval}(\mathit{seq}(E_1,E_2),\mathit{div},S) & \Leftarrow & \mathit{eval}(E_1,\mathit{end},[N|S_1]),\ \mathit{concat}([N|S_1],S_2,S).
\end{array}$$

Fig. 6. A logic program defining a big-step semantics with infinite behaviour and observations.

putation can continue with the evaluation of e_2, otherwise the overall computation diverges and e_2 is not evaluated.

As opposite to the previous examples, here we do not need just cofacts, but also a coclause; both the cofact and the coclause ensure that for infinite derivations only *div* can be derived. Furthermore, the cofact handles diverging expressions which produce a finite output sequence, as in $\mathit{eval}(E,\mathit{div},[\,])$ or in $\mathit{eval}(\mathit{seq}(\mathit{out}(1),E),\mathit{div},[1])$, with $E = \mathit{seq}(\mathit{skip},E)$ or $E = \mathit{seq}(E,E)$, while the coclause deals with diverging expressions with infinite outputs, as in $\mathit{eval}(E,\mathit{div},S)$ with $E = \mathit{seq}(\mathit{out}(1),E)$ and $S = [1|S]$. The body of the coclause ensures that the left operand of sequential composition converges, thus ensuring a correct productive definition.

6 Soundness and completeness

After formally relating the two approaches, we state soundness of the operational semantics with respect to the declarative one. Then, we show that completeness does not hold in general, and define the *regular* version of the declarative semantics. Finally, we show that the operational semantics is equivalent to this restricted declarative semantics.

Relation between operational and declarative semantics As in the standard case, the first step is to bridge the gap between the two approaches: the former computing equations, the latter defining truth of atoms. This can be achieved through the notions of *answers* to a goal.

Given a set of equations E, $\mathsf{sol}(E)$ is the set of the *solutions* of E, that is, the ground substitutions unifying all the equations in E. Then, $\theta \in \mathsf{sol}(E)$ is an *answer* to $\langle G;E\rangle$ if $\mathsf{Var}(G) \subseteq \mathsf{dom}(\theta)$.

The judgment $\langle P, P_{\mathsf{co}}\rangle;S \Vdash \langle G;E_1\rangle \Rightarrow E_2$ described in Section 4 computes a set of answers to the input goal. Indeed, solutions of the output set of equations are solutions of the input set as well, since the following proposition holds.

Proposition 6.1

1. If $\langle P, P_{\mathsf{co}}\rangle;S \Vdash \langle G;E_1\rangle \Rightarrow E_2$ then $E_1 \subseteq E_2$ and $\mathsf{Var}(G) \subseteq \mathsf{Var}(E_2)$.
2. If $E_1 \subseteq E_2$, then $\mathsf{sol}(E_2) \subseteq \mathsf{sol}(E_1)$.

Proof
(1) Straightforward induction on rules in Figure 2. (2) Trivial. □

On the other hand, we can define which answers are correct in an interpretation:

Definition 6.1
For $I \subseteq \mathsf{HB}_\infty$, the set of answers to $\langle G;E\rangle$ *correct in* I is $\mathsf{ans}(G,E,I) = \{\theta \in \mathsf{sol}(E) \mid G\theta \subseteq I\}$.

Hence, soundness of the operational semantics can be expressed as follows: all the answers computed for a given goal are correct in the declarative semantics.

Theorem 6.1 (*Soundness w.r.t. declarative semantics*)
If $\langle P, P_{\mathsf{co}}\rangle;\emptyset \Vdash \langle G;E\rangle \Rightarrow E'$ holds, then $\mathsf{sol}(E') \subseteq \mathsf{ans}(G,E,\nu_{\mathsf{fl}}[\![P,P_{\mathsf{co}}]\!])$.

Completeness issues The converse of this theorem, that is, all correct answers can be computed, cannot hold in general, since, as shown by Ancona and Dovier (2015), coinductive declarative semantics does not admit any complete procedure[9], hence our model as well, since it generalizes the coinductive one. To explain why completeness does not hold in our case, we can adapt the following example from Ancona and Dovier (2015)[10], where p is a predicate symbol of arity 1, z and s are function symbols of arity 0 and 1 respectively.

$$\begin{array}{lcl} p(X) & \leftarrow & p(s(X)). \\ p(X) & \Leftarrow & \end{array}$$

Let us define $\underline{0} = z$, $\underline{n+1} = s(\underline{n})$ and $\underline{\omega} = s(s(\ldots))$. The declarative semantics is the set $\{p(\underline{x}) \mid x \in \mathbb{N} \cup \{\omega\}\}$. In the operational semantics, instead, only $p(\underline{\omega})$ is considered true. Indeed, all derivations have to apply the rule (CO-HYP), which imposes the equation $X \doteq s(X)$, whose unique solution is $\underline{\omega}$. Therefore, the operational semantics is not complete.

Now the question is the following: can we characterize in a declarative way answers computed by the big-step semantics? In the example, there is a difference between the atoms $p(\underline{\omega})$ and $p(\underline{n})$, with $n \in \mathbb{N}$, because the former has a regular proof tree, namely, a tree with finitely many different subtrees, while the latter has only with non-regular, thus infinite, proof trees.

Following this observation, we prove that the operational semantics is sound and complete with respect to the restriction of the declarative semantics to atoms derivable by regular proof trees. As we will see, this set can be defined in model-theoretic terms, by restricting to finite comodels of the program. Dagnino (2020) defined this restriction for an arbitrary (generalized) inference system. We report here relevant definitions and results.

Regular declarative semantics Let us write $X \subseteq_{\mathit{fin}} Y$ if X is a finite subset of Y. The *regular interpretation* of $\langle \mathfrak{I}, \mathfrak{I}_{\mathsf{co}}\rangle$ is defined as

$$\rho_{\mathsf{fl}}[\![\mathfrak{I},\mathfrak{I}_{\mathsf{co}}]\!] = \bigcup\{X \subseteq_{\mathit{fin}} \mu[\![\mathfrak{I} \cup \mathfrak{I}_{\mathsf{co}}]\!] \mid X \subseteq F_{\mathfrak{I}}(X)\}$$

This definition is like the one of $\nu_{\mathsf{fl}}[\![\mathfrak{I},\mathfrak{I}_{\mathsf{co}}]\!]$, except that we take the union[11] only of those consistent subsets of $\mu[\![\mathfrak{I} \cup \mathfrak{I}_{\mathsf{co}}]\!]$ which are *finite*.The set $\rho_{\mathsf{fl}}[\![\mathfrak{I},\mathfrak{I}_{\mathsf{co}}]\!]$ is a fixed point of $F_{\mathfrak{I}}$ and, precisely, it is the *rational fixed point* (Adámek et al. 2006) of $F_{\mathfrak{I}}$ restricted to $\wp(\mu[\![\mathfrak{I} \cup \mathfrak{I}_{\mathsf{co}}]\!])$, hence we get $\rho_{\mathsf{fl}}[\![\mathfrak{I},\mathfrak{I}_{\mathsf{co}}]\!] \subseteq \nu_{\mathsf{fl}}[\![\mathfrak{I},\mathfrak{I}_{\mathsf{co}}]\!]$.

The proof-theoretic characterization relies on *regular proof trees*, which are proof trees with a

[9] That is, establishing whether an atom belongs to the coinductive declarative semantics is neither decidable nor semi-decidable, even when the Herbrand universe is restricted to the set of rational terms.
[10] Example 10 at page 8.
[11] Which could be an infinite set, hence it is not the same of the greatest finite consistent set.

finite number of subtrees (Courcelle 1983). That is, as proved by Dagnino (2020), $\rho_{\mathsf{fl}}[\![\mathfrak{I},\mathfrak{I}_{\mathsf{co}}]\!]$ is the set of judgments with a regular proof tree in $\mathfrak{I}$ whose nodes all have a finite proof tree in $\mathfrak{I}\cup\mathfrak{I}_{\mathsf{co}}$.

As special case, we get regular semantics of logic programs with coclauses.

Definition 6.2
The *regular declarative semantics* of $\langle P, P_{\mathsf{co}}\rangle$, denoted by $\rho_{\mathsf{fl}}[\![P,P_{\mathsf{co}}]\!]$, is the union of all finite comodels included in $\mu[\![P\cup P_{\mathsf{co}}]\!]$.

As above, $\rho_{\mathsf{fl}}[\![P,P_{\mathsf{co}}]\!]\subseteq \nu_{\mathsf{fl}}[\![P,P_{\mathsf{co}}]\!]$, hence $\mathsf{ans}(G,E,\rho_{\mathsf{fl}}[\![P,P_{\mathsf{co}}]\!])\subseteq \mathsf{ans}(G,E,\nu_{\mathsf{fl}}[\![P,P_{\mathsf{co}}]\!])$.

We state now soundness and completeness of the operational semantics with respect to this semantics. We write $\theta\preceq\sigma$ iff $\mathsf{dom}(\theta)\subseteq\mathsf{dom}(\sigma)$ and, for all $X\in\mathsf{dom}(\theta)$, $\theta(X)=\sigma(X)$. It is easy to see that $\preceq$ is a partial order and, if $\theta\preceq\sigma$ and $\mathsf{Var}(G)\subseteq\mathsf{dom}(\theta)$, then $G\theta=G\sigma$.

Theorem 6.2 (*Soundness w.r.t. regular declarative semantics*)
If $\langle P, P_{\mathsf{co}}\rangle;\emptyset\Vdash\langle G;E\rangle\Rightarrow E'$, and $\theta\in\mathsf{sol}(E')$, then $\theta\in\mathsf{ans}(G,E,\rho_{\mathsf{fl}}[\![P,P_{\mathsf{co}}]\!])$.

Theorem 6.3 (*Completeness w.r.t. regular declarative semantics*)
If $\theta\in\mathsf{ans}(G,E,\rho_{\mathsf{fl}}[\![P,P_{\mathsf{co}}]\!])$, then $\langle P, P_{\mathsf{co}}\rangle;\emptyset\Vdash\langle G;E\rangle\Rightarrow E'$, and $\theta\preceq\sigma$ for some E' and $\sigma\in\mathsf{sol}(E')$.

That is, any answer computed for a given goal is correct in the regular declarative semantics, and any correct answer is included in a computed answer. Theorem 6.2 immediately entails Theorem 6.1 as $\mathsf{ans}(G,E,\rho_{\mathsf{fl}}[\![P,P_{\mathsf{co}}]\!])\subseteq\mathsf{ans}(G,E,\nu_{\mathsf{fl}}[\![P,P_{\mathsf{co}}]\!])$.

Proof technique In order to prove the equivalence of the two semantics, we rely on a property which holds in general for the regular interpretation (Dagnino 2020): we can construct an equivalent inductive characterization. That is, given a generalized inference system $\langle\mathfrak{I}, \mathfrak{I}_{\mathsf{co}}\rangle$ on the universe $\mathscr{U}$, we can construct an inference system $\mathfrak{I}^{\circlearrowleft\mathfrak{I}_{\mathsf{co}}}$ with judgments of shape $H\triangleright j$, for $j\in\mathscr{U}$ and $H\subseteq_{fin}\mathscr{U}$, such that the inductive interpretation of $\mathfrak{I}^{\circlearrowleft\mathfrak{I}_{\mathsf{co}}}$ coincides with the regular interpretation of $\langle\mathfrak{I}, \mathfrak{I}_{\mathsf{co}}\rangle$. The set H, whose elements are called *coinductive hypotheses* , is used to detect cycles in the proof.

In particular, for logic programs with coclauses, we get an inference system with judgments of shape $S\triangleright A$, for S finite set of ground atoms, and A ground atom, defined as follows.

Definition 6.3
Given $\langle P, P_{\mathsf{co}}\rangle$, the inference system $P^{\circlearrowleft P_{\mathsf{co}}}$ consists of the following (meta-)rules:

$$(\textsc{hp})\quad \frac{}{S\triangleright A}\qquad A\in S \text{ and } A\in\mu[\![P\cup P_{\mathsf{co}}]\!]$$

$$(\textsc{rule})\quad \frac{S\cup\{A\}\triangleright B_1\ \ldots\ S\cup\{A\}\triangleright B_n}{S\triangleright A}\qquad (A\leftarrow B_1,\ldots,B_n)\in\|P\|$$

The following proposition states the equivalence with the regular interpretation. The proof is given by Dagnino (2020) in the general case of inference systems with corules.

Proposition 6.2
$P^{\circlearrowleft P_{\mathsf{co}}}\vdash_\mu \emptyset\triangleright A$ iff $A\in\rho_{\mathsf{fl}}[\![P,P_{\mathsf{co}}]\!]$.

Note that the definition of $P^{\circlearrowleft P_{\mathsf{co}}}\vdash_\mu S\triangleright A$ has many analogies with that of the operational semantics in Figure 2. The key difference is that the former handles *ground*, not necessarily finite, atoms, the latter not necessarily ground finite atoms (we use the same metavariables A and S for simplicity). In both cases already considered atoms are kept in an auxiliary set S. In the former, to derive an atom $A \in S$, the side condition requires A to belong to the inductive intepretation of the program $P \cup P_{\mathsf{co}}$. In the latter, when an atom A *unifies* with one in S, standard SLD resolution is triggered in the program $P \cup P_{\mathsf{co}}$.

To summarize, $P^{\circlearrowleft P_{\mathsf{co}}} \vdash_\mu S \triangleright A$ can be seen as an abstract version, at the level of the underlying inference system, of operational semantics. Hence, the proof of soundness and completeness can be based on proving a precise correspondence between these two inference systems, both interpreted inductively. This is very convenient since the proof can be driven in both directions by induction on the defining rules.

The correspondence is formally stated in the following two lemmas.

Lemma 6.1 (*Soundness w.r.t. inductive characterization of regular semantics*)
For all S and $\langle A_1,\dots,A_n;E\rangle$,
if $\langle P, P_{\mathsf{co}}\rangle; S \Vdash \langle A_1,\dots,A_n;E\rangle \Rightarrow E'$ then, for all $\theta \in \mathsf{sol}(E')$ and $i \in 1..n$, $P^{\circlearrowleft P_{\mathsf{co}}} \vdash_\mu S\theta \triangleright A_i\theta$.

Lemma 6.2 (*Completeness w.r.t. inductive characterization of regular semantics*)
For all S, $\langle A_1,\dots,A_n;E\rangle$ and $\theta \in \mathsf{sol}(E)$,
if $P^{\circlearrowleft P_{\mathsf{co}}} \vdash_\mu S\theta \triangleright A_i\theta$, for all $i \in 1..n$, then $\langle P, P_{\mathsf{co}}\rangle; S \Vdash \langle A_1,\dots,A_n;E\rangle \Rightarrow E'$ and $\theta \preceq \sigma$, for some E' and $\sigma \in \mathsf{sol}(E')$.

Soundness follows from Lemma 6.1 and Proposition 6.2, as detailed below.

Proof of Theorem 6.2
Let us assume $\langle P, P_{\mathsf{co}}\rangle; \emptyset \Vdash \langle G;E\rangle \Rightarrow E'$ with $G = A_1,\dots,A_n$, and consider $\theta \in \mathsf{sol}(E')$. By Lemma 6.1, for all $i \in 1..n$, $P^{\circlearrowleft P_{\mathsf{co}}} \vdash_\mu \emptyset \triangleright A_i\theta$ holds, hence, by Proposition 6.2, we get $A_i\theta \in \rho_{\mathsf{fl}}[\![P, P_{\mathsf{co}}]\!]$. Therefore, by Definition 6.2, we get $\theta \in \mathsf{ans}(G, E, \rho_{\mathsf{fl}}[\![P, P_{\mathsf{co}}]\!])$, as needed. □

Analogously, completeness follows from Lemma 6.2 and Proposition 6.2, as detailed below.

Proof of Theorem 6.3
Let $G = A_1,\dots,A_n$ and $\theta \in \mathsf{ans}(G, E, \rho_{\mathsf{fl}}[\![P, P_{\mathsf{co}}]\!])$. Then, for all $i \in 1..n$, we have $A_i\theta \in \rho_{\mathsf{fl}}[\![P, P_{\mathsf{co}}]\!]$ and, by Proposition 6.2, we get $P^{\circlearrowleft P_{\mathsf{co}}} \vdash_\mu \emptyset \triangleright A_i\theta$. Hence, the thesis follows by Lemma 6.2. □

7 Related work and conclusion

We have provided a detailed formal account of an extension of logic programming where programs are enriched by coclauses, which can be used to tune the interpretation of predicates on non-well-founded structures. More in detail, following the same pattern as for standard logic programming, we have defined:

- A declarative semantics (the union of all finite comodels which are subsets of a certain set of atoms determined by coclauses).
- An operational semantics (a combination of standard SLD resolution and coSLD resolution) shown to be sound and complete with respect to the declarative semantics.

As in the standard case, the latter provides a semi-algorithm. Indeed, concrete strategies (such as breadth-first visit of the SLD tree) can be used to ensure that the operational derivation, if any, is found. In this paper we do not deal with this part, however we expect it to be not too different from the standard case.

It has been shown (Ancona and Dovier 2015) that, taking as declarative semantics the coinductive semantics (largest comodel), there is not even a semi-algorithm to check that an atom belongs to that semantics. Hence, there is no hope to find a complete operational semantics. On the other hand, our paper provides, for an extension of logic programming usable in pratice to handle non-well-founded structures, fully-developed foundations and results which are exactly the analogous of those for standard logic programming.

CoLP has been initially proposed by Simon et al. (2006; 2006; 2007) as a convenient sub-paradigm of logic programming to model circularity; it was soon recognized the limitation of its expressive power that does not allow mutually recursive inductive and coinductive predicates, or predicates whose correct interpretation is neither the least, nor the greatest fixed point.

Moura et al. (2013; 2014) and Ancona (2013) have proposed implementations of coLP based on refinements of the Simon's original proposal with the main aim of making them more portable and flexible. Ancona has extended coLP by introducing a *finally* clause, allowing the user to define the specific behavior of a predicate when solved by coinductive hypothesis. Moura's implementation is embedded in a tabled Prolog related to the implementation of Logtalk, and is based on a mechanism similar to *finally* clauses to specify customized behavior of predicates when solved by coinductive hypothesis. While such mechanisms resemble coclauses, the corresponding formalization is purely operational and lacks a declarative semantics and corresponding proof principles for proving correctness of predicate definitions based on them.

Ancona and Dovier (2015) have proposed an operational semantics of coLP based on the big-step approach, which is simpler than the operational semantics initially proposed by Simon et al. and proved it to be sound. They have also formally shown that there is no complete procedure for deciding whether a regular goal belongs to the coinductive declarative semantics, but provided no completeness result restricted to regular derivations, neither mechanisms to extend coLP and make it more flexible.

Ancona et al. (2017a) were the first proposing a principled extension of coLP based on the notion of cofact, with both a declarative and operational semantics; the latter is expressed in big-step style, following the approach of Ancona and Dovier, and is proved to be sound w.r.t. the former. An implementation is provided through a SWI-Prolog meta-interpreter.

Our present work differs from the extension of coLP with cofacts mentioned above for the following novel contributions:

- we consider the more general notion of coclause, which includes the notion of cofact, but is a more expressive extension of coLP;
- we introduce the notion of regular declarative semantics and prove coSLD resolution extended with coclauses is sound and complete w.r.t. the regular declarative semantics;
- we show how generalized inference systems are closely related to logic programs with coclauses and rely on this relationship to carry out proofs in a clean and principled way;
- we extend the implementation[12] of the SWI-Prolog meta-interpreter to support coclauses.

While coSLD resolution and its proposed extensions are limited by the fact that cycles must be detected in derivations to allow resolution to succeed, a stream of work based on the notion of *structural resolution* (Komendantskaya et al. 2016; Komendantskaya et al. 2017) (S-resolution for short) aims to make coinductive resolution more powerful, by allowing to lazily detect infinite derivations which do not have cycles. In particular, recent results (Li 2017; Komendantskaya and

[12] See `https://github.com/davideancona/coLP-with-coclauses`, where also examples of Sect. 5 are available.

Li 2017; Basold et al. 2019) investigate how it is possible to integrate coLP cycle detection into S-resolution, by proposing a comprehensive theory. Trying to integrate S-resolution with coclauses is an interesting topic for future work aiming to make coLP even more flexible.

Another direction for further research consists in elaborating and extending the examples of logic programs with coclauses provided in Section 5, to formally prove their correctness, and experiment their effectiveness with the implemented meta-interpreter.

References

ACZEL, P. 1977. An introduction to inductive definitions. In *Handbook of Mathematical Logic*, J. Barwise, Ed. Studies in Logic and the Foundations of Mathematics, vol. 90. Elsevier, 739 – 782.

ADÁMEK, J., MILIUS, S., AND VELEBIL, J. 2006. Iterative algebras at work. *Mathematical Structures in Computer Scienc 16,* 6, 1085–1131.

ANCONA, D. 2013. Regular corecursion in prolog. *Comput. Lang. Syst. Struct. 39,* 4, 142–162.

ANCONA, D., DAGNINO, F., ROT, J., AND ZUCCA, E. 2020. A big step from finite to infinite computations. *Science of Computer Programming 197*, 102492.

ANCONA, D., DAGNINO, F., AND ZUCCA, E. 2017a. Extending coinductive logic programming with co-facts. In *First Workshop on Coalgebra, Horn Clause Logic Programming and Types, CoALP-Ty'16*, E. Komendantskaya and J. Power, Eds. Electronic Proceedings in Theoretical Computer Science, vol. 258. Open Publishing Association, 1–18.

ANCONA, D., DAGNINO, F., AND ZUCCA, E. 2017b. Generalizing inference systems by coaxioms. In *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017*, H. Yang, Ed. Lecture Notes in Computer Science, vol. 10201. Springer, Berlin, 29–55.

ANCONA, D., DAGNINO, F., AND ZUCCA, E. 2018. Modeling infinite behaviour by corules. In *32nd European Conference on Object-Oriented Programming, ECOOP 2018*, T. D. Millstein, Ed. LIPIcs, vol. 109. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Dagstuhl, 21:1–21:31.

ANCONA, D. AND DOVIER, A. 2015. A theoretical perspective of coinductive logic programming. *Fundamenta Informaticae 140,* 3-4, 221–246.

APT, K. R. 1997. *From logic programming to Prolog*. Prentice Hall International series in computer science. Prentice Hall.

BASOLD, H., KOMENDANTSKAYA, E., AND LI, Y. 2019. Coinduction in uniform: Foundations for corecursive proof search with horn clauses. In *ESOP 2019*, L. Caires, Ed. Lecture Notes in Computer Science, vol. 11423. Springer, 783–813.

COURCELLE, B. 1983. Fundamental properties of infinite trees. *Theoretical Computer Science 25*, 95–169.

DAGNINO, F. 2017. Generalizing inference systems by coaxioms. M.S. thesis, DIBRIS, University of Genova. Best italian master thesis in Theoretical Computer Science 2018.

DAGNINO, F. 2019. Coaxioms: flexible coinductive definitions by inference systems. *Logical Methods in Computer Science 15,* 1.

DAGNINO, F. 2020. Foundations of regular coinduction. Tech. rep., DIBRIS, University of Genova. May. Available at `https://arxiv.org/abs/2006.02887`. Submitted for journal publication.

GUPTA, G., SAEEDLOEI, N., DEVRIES, B. W., MIN, R., MARPLE, K., AND KLUZNIAK, F. 2011. Infinite computation, co-induction and computational logic. In *CALCO 2011 - Algebra and Coalgebra in Computer Science*, A. Corradini, B. Klin, and C. Cîrstea, Eds. Lecture Notes in Computer Science, vol. 6859. Springer, 40–54.

KOMENDANTSKAYA, E. ET AL. 2016. Coalgebraic logic programming: from semantics to implementation. *J. Logic and Computation 26,* 2, 745.

KOMENDANTSKAYA, E. ET AL. 2017. A productivity checker for logic programming. *Post-proc. LOPSTR'16*.

KOMENDANTSKAYA, E. AND LI, Y. 2017. Productive corecursion in logic programming. *Theory Pract. Log. Program. 17,* 5-6, 906–923.

LEROY, X. AND GRALL, H. 2009. Coinductive big-step operational semantics. *Information and Computation 207,* 2, 284–304.

LI, Y. 2017. Structural resolution with coinductive loop detection. In *Post-proceedings of CoALP-Ty'16*, E. Komendantskaya and J. Power, Eds.

LLOYD, J. W. 1987. *Foundations of Logic Programming, 2nd Edition*. Springer.

LÖDING, C. AND TOLLKÖTTER, A. 2016. Transformation between regular expressions and omega-automata. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016*, P. Faliszewski, A. Muscholl, and R. Niedermeier, Eds. LIPIcs, vol. 58. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 88:1–88:13.

MANTADELIS, T., ROCHA, R., AND MOURA, P. 2014. Tabling, rational terms, and coinduction finally together! *TPLP 14,* 4-5, 429–443.

MOURA, P. 2013. A portable and efficient implementation of coinductive logic programming. In *Practical Aspects of Declarative Languages - 15th International Symposium, PADL 2013, Rome, Italy, January 21-22, 2013. Proceedings*. 77–92.

SIMON, L. 2006. Extending logic programming with coinduction. Ph.D. thesis, University of Texas at Dallas.

SIMON, L., BANSAL, A., MALLYA, A., AND GUPTA, G. 2007. Co-logic programming: Extending logic programming with coinduction. In *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007*, L. Arge, C. Cachin, T. Jurdzinski, and A. Tarlecki, Eds. Lecture Notes in Computer Science, vol. 4596. Springer, 472–483.

SIMON, L., MALLYA, A., BANSAL, A., AND GUPTA, G. 2006. Coinductive logic programming. In *Logic Programming, 22nd International Conference, ICLP 2006*, S. Etalle and M. Truszczynski, Eds. Lecture Notes in Computer Science, vol. 4079. Springer, 330–345.

TARSKI, A. 1955. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics 5,* 2, 285–309.