A theoretical perspective of Coinductive Logic Programming

Davide Ancona
University of Genova
davide.ancona@unige.it

Agostino Dovier
University of Udine
agostino.dovier@uniud.it

Abstract

In this paper we study the semantics of Coinductive Logic Programming and clarify its intrinsic computational limits, which prevent, in particular, the definition of a complete, computable, operational semantics. We propose a new operational semantics that allows a simple correctness result and the definition of a simple meta-interpreter. We compare, and prove the equivalence, with the operational semantics defined and used in other papers on this topic.

1 Introduction

Induction is a powerful mathematical principle for reasoning about finite entities; for this reason, it is at the basis of computer science and, more in particular, of programming languages and their semantics, where induction is modeled by the notion of least fixed point of (complete, or at least monotonic) functions defined over a (possibly complete) lattice.

On the other hand, there are situations naturally arising both in mathematics and computer science, where it is necessary to reason about infinite or circular entities. For instance, equivalence of ill-founded sets in set theory [1] cannot be adequately captured by the extensionality principle used in standard set theory, because proving the equality of two ill-founded sets, like those defined by the equations $X = \{X\}$, $Y = \{Y, \{Y\}\}$, would lead to circularity. This problem can be solved with Aczel's AFA axiom and the subsequent coinductive notion of bisimulation.

Bisimulation and other definitions and properties involving infinite or cyclic entities can be formalized and proved with the coinductive principle. The minimum representation of an ill-founded set S can be obtained by computing the maximum bisimulation between a set S and S itself with a greatest fixpoint algorithm [25, 16, 14]. In the area of planning, the semantics of Action Description Languages in presence of static causal laws is given in terms of a "maximum" solution of a cyclic equation [17, 15].

Graphs are one of the most commonly used circular structures in computer science, but there are also other well-known formalisms where the notion of circularity is dominant, like automata (of several kinds), recursive types, grammars, process algebras, and temporal logics. Bisimulation is at the basis of the equivalence relations in these formalisms. Coinduction is also employed for formal reasoning about non-terminating computations, to prove properties of systems that have to run forever.

There are at least three main reasons that make Prolog a suitable programming language for supporting coinductive programming [31, 30, 22]: its remarkably clean semantics, its built-in support for rational terms, and its declarative nature. Imperative languages lack abstraction mechanisms for defining and manipulating circular structures that avoid explicit use of references; for instance, in an object-oriented language, creating two objects pointing to each other through two fields requires at least three statements, because the field of the firstly created object can be correctly assigned only after the second object has been created. In Prolog the definition of two circular values is more concise, and, hence, less error prone. More importantly, in object-oriented languages implementing an equality test that ensures referencial transparency is not a trivial task when objects contain cycles, since the test may not terminate for naive solutions.

In purely declarative languages, coinductive definitions on cyclic values cannot be easily turned into functions, as happens with inductively defined data types. Lazy functional languages are not exceptional in this respect, when strict operations are involved; for instance, equality between infinite lists can be easily specified coinductively, but the corresponding Haskell function directly obtainable from this specification fails to terminate for all pairs of infinite lists, even when such lists are only circular (and, hence, finitely representable).

Coinductive Logic Programming (briefly, co-LP) shares with LP the same syntax, but the least fixpoint semantics is replaced by the greatest fixpoint one, and, consequently, the standard Herbrand model is extended

to include also infinite terms (that is, finitely branching trees with infinite depth).

In this paper we provide a formal account of co-LP [31, 29] to try to clarify what are the main potentials, but also the limits of this new computational paradigm.

A first contribution of this paper is the definition of a small-step operational semantics which is closer to the standard operational semantics of Logic Programming. Indeed, we extend and generalize the notion of SLD derivation, by associating sets of coinductive hypotheses with goal atoms, and, by adding, besides the standard SLD transition rule, a rule for dealing with coinduction. The extension is natural, since in our semantics each derivation obtained by applying only the SLD rule can be trivially reduced to a valid SLD derivation by simply forgetting the coinductive hypotheses associated with goal atoms.

Besides this direct relation with the standard LP operational semantics, this semantics is simpler than the one originally given by Gupta et al. [31, 29], and allowed us to provide a more concise proof of soundness w.r.t. the model-theoretic semantics, by exploiting previous results from Jaffar and Stuckey [18].

A second contribution consists in the definition of an alternative big-step operational semantics inspired by previous work on co-LP [2], which is more suitable for implementation purposes. Indeed, it provides a simple specification of the implementation of co-LP as supported, for instance, by the `coinduction` library[1] of SWI-Prolog. We formally prove that the small-step and big-step operational semantics are equivalent; by virtue of this equivalence, and of the soundness results proved for the small-step semantics, we can directly derive that the provided big-step operational semantics (and, hence, every implementation based on it) is sound w.r.t. the model-theoretic semantics of co-LP.

Finally, a third, and maybe more foundational contribution, consists in proving that establishing whether a given atom belongs to the coinductive semantics of a definite clause program is neither decidable nor semi-decidable, even when the Herbrand universe is restricted to the set of rational terms. This implies the incompleteness of *any* computational approach to co-LP.

The paper is organized as follows: in Section 2 we recall the main notions on finite and infinite terms, in Section 3 we overview the main properties of the T_P operator. In Section 4 we analyze the computability of the sets relevant for the semantics, then in Section 5 we prove the intrinsic computational incompleteness of co-LP and provide a new operational semantics for co-LP, based on a simpler coinductive resolution rule, and prove its correctness. Conclusions are drawn in Section 6, whereas the appendix defines an alternative operational semantics, given in big-step style, and proves the equivalence of the two presented semantics with that given by Gupta et al. [31, 29].

2 Preliminaries

Terms and Formulas. A first-order signature is a 3-tuple $\langle \Pi, \mathcal{F}, \mathcal{V} \rangle$ where Π is a set of predicate symbols, $\mathcal{F}$ is a set of function and constant symbols, and $\mathcal{V}$ is a denumerable set of variable symbols. For any $\diamondsuit \in \mathcal{F} \cup \mathcal{P}$, $\mathsf{ar}(\diamondsuit) \geq 0$ (arity) denotes the number of arguments of the symbol $\diamondsuit$. We write $\diamondsuit/n$ to mean $\mathsf{ar}(\diamondsuit) = n$. For any $x \in \mathcal{V}$, $\mathsf{ar}(x) = 0$. If $f \in \mathcal{F}$ and $\mathsf{ar}(f) = 0$, f is called a constant symbol.

$\mathbb{N}^*$ will denote the set of finite sequences of integer numbers (strings), and "$\cdot$" will denote the strings concatenation operator. A tree will be represented as the set of the paths from its root, precisely, a *tree* T is a non-empty subset of $\mathbb{N}^*$ such that if $n \in \mathbb{N}$ and $u \cdot n \in T$ then $u \in T$ and $(\forall m \in \{0, \ldots, n-1\})(u \cdot m \in T)$. The *root* of the tree is identified by the empty path ε.[2] The children of the node u are $u \cdot 0, u \cdot 1, \ldots$, from left to right. If $u \in T$ and $u \cdot 0 \notin T$ then u is a *leaf*. A *term* t is a pair $\langle T, \rho \rangle$ where T is a tree and ρ is a function $\rho : T \longrightarrow \mathcal{F} \cup \mathcal{V}$ such that for each $u \in T$: $|\{u \cdot n \in T : n \in \mathbb{N}\}| = \mathsf{ar}(\rho(u))$. Terms are trees with nodes labeled by ρ with signature symbols in a way consistent with their arities. A term $\langle T, \rho \rangle$ is *finite* if and only if T is a finite set.

Example 1 *Let $\mathcal{F} = \{\mathbf{z}/0, s/1, f/2\}$, and let X, Y be two variables. Then, $T_1 = \{\varepsilon, 0, 1, 0{\cdot}0, 0{\cdot}1, 1{\cdot}0, 1{\cdot}0{\cdot}0\}$ and $\rho_1 = \{(\varepsilon, f), (0, f), (1, s), (0{\cdot}0, \mathbf{z}), (0{\cdot}1, X), (1{\cdot}0, s), (1{\cdot}0{\cdot}0, Y)\}$ identify a term, say t_1 ($t_1 = f(f(\mathbf{z}, X), s(s(Y)))$ using the standard notation of finite terms)—see Fig. 1. $T_2 = \{0\}^* = \{\varepsilon, 0, 0 \cdot 0, 0 \cdot 0 \cdot 0, 0 \cdot 0 \cdot 0 \cdot 0, \ldots\}$ and ρ_2 defined as $\rho_2(x) = s$ for all $x \in T_2$ identify an infinite term. We will refer to this term as Ω in this paper. Using an extension of the standard notation for finite terms, $\Omega = s(s(s(\cdots))))$—see Fig. 1.*

When clear from the context, we simply identify a term using the standard notation (which allows, of course, only an approximation of infinite terms).

[1] `http://www.swi-prolog.org/pldoc/doc/swi/library/coinduction.pl`

[2] Non-emptiness and prefix closure ensure that $\varepsilon \in T$ for all T.

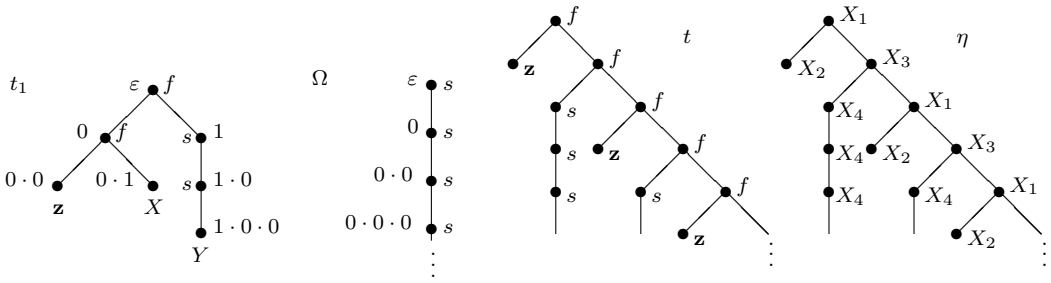

Figure 1: Rational terms described in Examples 1 and 2

A *term equation* is an object of the form $\ell = r$ where ℓ and r are finite terms. A *system of term equations* is a set of term equations. A finite system of term equations E is in *solved form* if $E = \{X_1 = r_1, \ldots, X_n = r_n\}$, where $X_1, \ldots, X_n$ are pairwise distinct variables, and for all $i \in \{1, \ldots, n\}$, if r_i is a variable, then X_i does not occur in $r_1, \ldots, r_n$ (and, in particular, r_i is different from X_i).

Given two trees T_1 and T_2 and a path $u \in T_2$, T_1 is a subtree of T_2 rooted at u (briefly, $T_1 \trianglelefteq_u T_2$) if $T_1 = \{v \in \mathbb{N}^* : uv \in T_2\}$. A term $\langle T_1, \rho_1\rangle$ is a subterm of a term $\langle T_2, \rho_2\rangle$ rooted at u if $T_1 \trianglelefteq_u T_2$, and $(\forall v \in T_1)(\rho_1(v) = \rho_2(uv))$. In this case we write $\langle T_1, \rho_1\rangle = \langle T_2, \rho_2\rangle|_u$.

A term is called *rational* if it has a finite number of different subterms. All finite terms are rational. It is well-known from unification theory that a given finite system of term equations either is unsatisfiable or it has (at least) one rational tree solution θ that can be expressed by a solved form system $\{X_1 = r_1, \ldots, X_n = r_n\}$; that is, θ is a *tree substitution* (see the formal definition below) mapping the variable X_i to the tree r_i for all $i \in \{1 \ldots n\}$.

Also the converse is true, namely, if $t = \langle T, \rho\rangle$ is a rational term, and $\{t_1, \ldots, t_n\}$ is the finite set of its different subterms (without loss of generality, let $t_1 = t$), then there is a solved form system of equations on n variables that admits the unique solution θ such that for $i = 1, \ldots, n$ $\theta(X_i) = t_i$. For further reading on this material, we refer to [21, 13].

Example 2 *Let $\mathcal{F} = \{\mathbf{z}/0, s/1, f/2\}$. The infinite tree t in Fig. 1 has exactly 4 subtrees identified by the function η (Fig. 1–right). The solved form system of equations associated is $\{X_1 = f(X_2, X_3), X_2 = \mathbf{z}, X_3 = f(X_4, X_1), X_4 = s(X_4)\}$.*

An atomic formula, or *atom*, is a pair $\langle T, \rho\rangle$ where T is a tree and ρ is a function $\rho : T \longrightarrow \Pi \cup \mathcal{V} \cup \mathcal{F}$ such that $\rho(\varepsilon) = p \in \Pi$, $\mathsf{ar}(p) \notin T$, and $(\forall i \in \{0, \ldots, \mathsf{ar}(p) - 1\})(i \in T \wedge \langle T, \rho\rangle|_i$ is a term). An atom is *finite* if T is a finite set.

As implicitly admitted by Prolog, an atom can be simply viewed as a term whose root is labeled by a predicate symbol. With $\mathsf{FV}(t)$ we denote the set of variables occurring in a term or in an atom t. A term/atom is *ground* if $\mathsf{FV}(t) = \emptyset$.

A *definite clause* is a disjunction of one finite atom with a finite number, possibly zero, of negations of finite atoms. A *definite clause program* is a finite set of definite clauses.

Universes. Let P be a definite clause program. Let $\mathcal{F}_P$ be the set of function and constant symbols in P. The *Herbrand Universe* U_P is the set of finite (ground) terms one can build with $\mathcal{F}_P$. As done in [19, §3] if there exists no $f/0$ in $\mathcal{F}_P$, we add one constant symbol (e.g., $\mathbf{z}/0$) to $\mathcal{F}_P$ in order to form ground terms. Such a set is finite (and isomorphic to $\mathcal{F}_P$) if and only if for all $f \in \mathcal{F}_P$ $\mathsf{ar}(f) = 0$.

The *complete Herbrand Universe* co-U_P [19, §25] is the set of finite and infinite terms one can build with $\mathcal{F}_P$. The set co-U_P contains finite terms, infinite rational terms, and non-rational terms. co-U_P is finite (and equal to U_P) if and only if for all $f \in \mathcal{F}_P$ $\mathsf{ar}(f) = 0$.

Example 3 *Assume $\mathcal{F}_P = \{\mathbf{z}/0, s/1\}$. As a shorthand, we will use $\underline{0}$ for $\mathbf{z}$ and $\underline{n+1}$ for $s(\underline{n})$. Therefore: $U_P = \{\mathbf{z}, s(\mathbf{z}), s(s(\mathbf{z})), s(s(s(\mathbf{z}))), \ldots\} = \{\underline{0}, \underline{1}, \underline{2}, \underline{3}, \ldots\}$. This signature allows just one infinite term: the term $\Omega = s(s(s(\cdots)))$ (c.f. Example 1). The only subterm of Ω is the term itself ($\Omega = s(\Omega)$), therefore co-$U_P = \{\Omega, \underline{0}, \underline{1}, \underline{2}, \underline{3}, \ldots\}$.*

Assume now $\mathcal{F}_P = \{\lambda/0, [\cdot\,|\,\cdot]/2\}$. We will interpret $[\cdot\,|\,\cdot]$ as the usual Prolog list constructor and use λ for the empty list and adopt the usual shorthand, when possible. A partial view of U_P is the following:

$$U_P = \{\ \lambda, [\lambda], [\lambda, \lambda], [\lambda, \lambda, \lambda], \ldots, [[\lambda]], [\lambda, [\lambda]], [[\lambda], \lambda], \ldots\ \}$$

co-U_P *contains either rational or non-rational infinite terms. As a (local) shorthand, let us use here* 0 *for* λ *and* 1 *for* $[\lambda]$ *(i.e.* $[\lambda \mid \lambda]$*). Then, for instance, the infinite lists* $[0,0,0,0,0,0,0,0,0,0,\ldots]$ *and* $[0,1,0,1,0,1,0,1,0,1,\ldots]$ *that are solutions to the term equations* $X = [0|X]$ *and* $X = [0,1|X]$*, respectively, are two rational terms in* co-U_P*. Let* $b_1.b_2b_3b_4b_5b_6\ldots$ *(*$1.01101\ldots$*) be the (infinite) binary representation of* $\sqrt{2}$ *(*$b_i \in \{0,1\}$*). The infinite list* $\mathsf{sqrt2} = [b_1, b_2, b_3, b_4, b_5, b_6, \ldots]$ *belongs to* co-U_P*, and it has an infinite number of different subterms, otherwise* $\sqrt{2}$ *would be a rational number.*

While U_P is finite or denumerable, if $\mathcal{F}$ contains at least two unary symbols or at least one constant symbol and one symbol of arity greater than 1 (briefly, $\mid \mathcal{F} \mid \geq 2$, $\sum_{f \in \mathcal{F}} \mathsf{ar}(f) \geq 2$), then co-U_P is not denumerable (generalize the reasoning made in Example 3 for defining $\mathsf{sqrt2}$).

A *tree substitution* (ground substitution in [18]) θ is a mapping from a finite subset of variables, $\mathsf{dom}(\theta)$ to co-U_P. The application of θ replaces each leaf labeled by $X \in \mathsf{dom}(\theta)$ in t with the subterm $\theta(X)$ (we omit here a more formal definition). The notion of application can be extended to atoms and formulas in the usual way.

Let P be a definite clause program. With B_P, the *Herbrand base* of P we denote the sets of all ground atoms, namely atoms based on the predicate symbols in P and on the terms in U_P. Sets $I \subseteq B_P$ are called *interpretations*. $\langle B_P, \subseteq \rangle$ is a complete lattice, where $\bot = \emptyset$, $\top = B_P$, least upper bound (lub) is computed by $\cup$ and greatest lower bound (glb) is computed by $\cap$. We define:

$$\begin{array}{rcl} \mathsf{ground}(P) & = & \{(A \leftarrow B_1, \ldots, B_n)\theta : \quad A \leftarrow B_1, \ldots, B_n \in P, \text{ and } \theta \text{ is a tree substitution} \\ & & \qquad\qquad \theta : \mathsf{FV}(A \leftarrow B_1, \ldots, B_n) \longrightarrow U_P\} \end{array}$$

In the context of the complete universe, an atom is a finite or an infinite tree whose root is labeled by a predicate symbol p and its $\mathsf{ar}(p)$ children are the roots of (finite and infinite) terms from co-U_P. The *complete Herbrand base* co-B_P is the set of all ground atoms based on the predicate symbols in P and the finite and infinite terms in co-U_P. Sets $I \subseteq \mathsf{co}$-B_P are called co-*interpretations*. Finally, let:

$$\begin{array}{rcl} \mathsf{co\text{-}ground}(P) & = & \{(A \leftarrow B_1, \ldots, B_n)\theta : \quad A \leftarrow B_1, \ldots, B_n \in P, \text{ and } \theta \text{ is a tree substitution} \\ & & \qquad\qquad \theta : \mathsf{FV}(A \leftarrow B_1, \ldots, B_n) \longrightarrow \mathsf{co}\text{-}U_P\} \end{array}$$

If $S \subseteq \mathsf{co\text{-}ground}(P)$, with $\Upsilon(S) \subseteq S$ we denote the subset of S *restricted to rational terms*.

SLD derivation and its variants. In the classical definition of SLD derivation, given a definite clause program P and a goal[3] $G = A_1, \ldots, A_n$, the rewriting rule $\vdash$ that leads G to G' is defined as follows: choose $A_i = p(s_1, \ldots, s_k)$ in G, and $p(t_1, \ldots, t_k) \leftarrow B_1, \ldots, B_m$ a renaming with fresh variables of a clause in P; if $\{s_1 = t_1, \ldots, s_k = t_k\}$ is solvable and has m.g.u. θ, then $G' = \theta(A_1, \ldots, A_{i-1}, B_1, \ldots, B_m, A_{i+1}, \ldots, A_n)$. This notion is generalized by the following one, where systems of equations are explicitly allowed to deal with (possibly infinite) rational terms. The rewriting rule $\vdash_\infty$ that leads a goal $G = \langle E \,\square\, A_1, \ldots, A_n \rangle$, where E is a solvable set of equations and A_i are atoms, to a goal G' is defined as follows: choose $A_i = p(s_1, \ldots, s_k)$ in G, and $p(t_1, \ldots, t_k) \leftarrow B_1, \ldots, B_m$ a renaming with fresh variables of a clause in P; if $E' = E \cup \{s_1 = t_1, \ldots, s_k = t_k\}$ is solvable (in co-U_P), then $G' = \langle E' \,\square\, A_1, \ldots, A_{i-1}, B_1, \ldots, B_m, A_{i+1}, \ldots, A_n \rangle$. Since θ corresponds to exactly the same solutions of $\{s_1 = t_1, \ldots, s_n = t_n\}$, the definitions of $\vdash$ and $\vdash_\infty$ are equivalent for definite programs (assuming to use or not occurs-check consistently in the two cases).

The notion of $\vdash_\infty$ can be extended to derivations and trees as usual. A *derivation* is a (finite or infinite) sequence of applications of $\vdash_\infty$; a *ground derivation* is a (finite or infinite) sequence of applications of $\vdash_\infty$, where a ground instantiation of the program is considered at each step. A goal is *successful* if it is of the form $\langle E \,\square\, \varepsilon \rangle$ (ε is the empty list of atoms). A derivation is *successful* if it is finite and its last goal is successful. A derivation is *failed* if it is finite, its last goal is non-empty, and the selected atom of such a goal does not unify with the head of any clause in P. A *selection rule* R chooses the literal A_i in a goal $G = \langle E \,\square\, A_1, \ldots, A_n \rangle$ given the whole derivation that led to G. In Prolog, R simply selects A_1 (leftmost atom). Given a selection rule R, the *SLD* tree for a goal G is the tree representing all the possible derivations starting from G according with R (choices are related to the different clauses defining the predicate of A_i). If all derivations of the *SLD* tree for G are failed, then it is a *finitely failed SLD-tree*.

Example 4 (SLD derivation with infinite terms) *Let* P *be the following definite clause program:*

$$\mathtt{num}(\mathtt{z}). \qquad \mathtt{num}(s(X)) \leftarrow \mathtt{num}(X).$$

[3]For the sake of notational simplicity, we just consider the list of atoms of a goal, not its logical meaning.

Using the standard SLD rule we would have the successful derivation

$$\mathtt{num}(s(s(\mathbf{z}))) \vdash \mathtt{num}(s(\mathbf{z})) \vdash \mathtt{num}(\mathbf{z}) \vdash \varepsilon$$

Using $\vdash_\infty$ *we would have the following equivalent successful derivation:*

$$\langle \emptyset \,\square\, \mathtt{num}(s(s(\mathbf{z}))) \rangle \vdash_\infty \langle \{X = s(\mathbf{z})\} \,\square\, \mathtt{num}(X) \rangle \vdash_\infty \langle \{X = s(\mathbf{z}), X = s(X_1)\} \,\square\, \mathtt{num}(X_1) \rangle \vdash_\infty \langle \{X = s(\mathbf{z}), X = s(X_1), X_1 = 0\} \,\square\, \varepsilon \rangle$$

The system has mgu $X/s(\mathbf{z}), X_1/0$. *Using* $\vdash_\infty$ *we can also get a derivation for* $\mathtt{num}(\omega)$*:*

$$\begin{array}{l} \langle \{X = s(X)\} \,\square\, \mathtt{num}(X) \rangle \vdash_\infty \langle \{X = s(X), X = s(X_1)\} \,\square\, \mathtt{num}(X_1) \rangle \vdash_\infty \\ \langle \{X = s(X), X = s(X_1), X_1 = s(X_2)\} \,\square\, \mathtt{num}(X_2) \rangle \vdash_\infty \cdots \end{array}$$

The derivation is infinite. Observe that $\{X = s(X), X = s(X_1), X_1 = s(X_2), X_2 = s(X_3), \ldots\}$ *is solvable and equivalent to* $\{X = s(X), X_1 = X, X_2 = X, X_3 = X, \ldots\}$.

Rule $\vdash_\infty$ is similar to the rule proposed in [18] to deal with Prolog II programs [12], but E admits here only equations (hence, inequations are not allowed), and a single atom is selected for each derivation step, while all atoms are selected together in [18]. We refer to this rule as $\vdash_\pi$. The notions of derivation and tree apply to this rule, as well.

Proposition 1 *Let P be a program without inequations and $G = \langle E \,\square\, A \rangle$ be a goal. Then*

1. *There is a $\vdash_\pi$-successful derivation for G if and only if there is a $\vdash_\infty$-successful derivation for G.*
2. *The SLD tree for G with the $\vdash_\pi$-rule finitely fails if and only if there is a selection rule R such that the SLD tree for G using the $\vdash_\infty$-rule finitely fails.*

The proof of (1) follows from the independence of the computation rule [19]. For (2) it is sufficient to define R that simulates, using repeated single steps applications, the [18]-rule.

We remark, however, that using a given selection rule R it might be the case that a goal G has an infinite $\vdash_\infty$ derivation while $\vdash_\pi$-rule leads to a finitely failed SLD tree. For instance, this happens using the leftmost selection rule in the program $p \leftarrow p. \qquad q \leftarrow p, r.$ with the goal $\langle \emptyset \,\square\, q \rangle$.

Some computability notions used in the paper. We assume basic computability notions (see e.g., [26]). The relevant definitions and results are reported here for reader's convenience. Let us denote, as usual, $K = \{x \in \mathbb{N} : M_x(x)\downarrow\}$ and $\bar{K} = \mathbb{N} \setminus K$, where M_x is the x-th Turing machine and $M_x(x)\downarrow$ means that the computation of the Turing machine M_x on input x terminates in a finite number of steps. K is recursively enumerable (r.e.) but not recursive and $\bar{K}$ is productive (hence, non r.e. in a very strong way). Given two sets A and B, $A \leq B$ (and $\bar{A} \leq \bar{B}$) if there is a total recursive function f such that for all $x \in \mathbb{N}$ $x \in A$ iff $f(x) \in B$. A r.e. set B is *r.e. complete* if for all r.e. set A it holds that $A \leq B$ (due to transitivity of $\leq$ and completeness of K, it is sufficient to prove that $K \leq B$). A is complete iff A is r.e. and $\bar{A}$ is productive (Myhill). If $\bar{K} \leq A$ (or equivalently $K \leq \bar{A}$) then A is productive, as well. The class Π^1_1 is one of the lower classes in the analytical hierarchy of sets [26, §16]. Let $\mathbb{F}$ be the set of total functions from $\mathbb{N}$ to $\mathbb{N}$. A set S is Π^1_1 if there is a recursive relation $R \subseteq \mathbb{F} \times \mathbb{N}^2$ such that $S = \{x \in \mathbb{N} : (\forall f)(\exists y)(R(f,(y,x)))\}$. Observe that $\bar{S} = \{x \in \mathbb{N} : (\exists f)(\forall y)(\neg R(f,(y,x)))\}$. S is Π^1_1 *complete* if any Π^1_1 set can be reduced to it. Clearly, if S is Π^1_1 complete, neither S nor $\bar{S}$ are r.e..

3 Least and Greatest Fixpoints

Finite Terms Universe. Given a definite clause program P, the $T_P : \wp(B_P) \longrightarrow \wp(B_P)$ operator between sets of ground atomic formulas is defined as follows.

$$T_P(I) \quad = \quad \{a : (a \leftarrow b_1, \ldots, b_n) \in \mathsf{ground}(P) \wedge \{b_1, \ldots, b_n\} \subseteq I\} \tag{1}$$

It is well-known that I is a (Herbrand) model of P if and only if $T_P(I) \subseteq I$. In particular, this holds for any I which is a fixpoint of T_P, i.e., such that $T_P(I) = I$. It is also well-known that T_P is monotonic ($\forall X, Y \in \wp(B_P)$ $(X \subseteq Y \Rightarrow T_P(X) \subseteq T_P(Y))$) and upward continuous (for each infinite sequence $I_0 \subseteq I_1 \subseteq I_2 \subseteq \cdots$: $T_P\left(\bigcup_{i \geq 0} I_i\right) = \bigcup_{i \geq 0} T_P(I_i)$) but not downward continuous (downward continuous means that for each infinite

sequence $I_0 \supseteq I_1 \supseteq I_2 \supseteq \cdots$: $T_P\left(\bigcap_{i\geq 0} I_i\right) = \bigcap_{i\geq 0} T_P(I_i)$). Let us recall the definitions of iterated T_P (we assume the notion of (successor/limit) ordinal [19]):

$$\begin{cases} T_P \uparrow 0 = \emptyset & T_P \downarrow 0 = B_P \\ T_P \uparrow \alpha = T_P(T_P \uparrow (\alpha-1)) & T_P \downarrow \alpha = T_P(T_P \downarrow (\alpha-1)) \quad \text{if } \alpha \text{ is a successor ordinal} \\ T_P \uparrow \alpha = \bigcup_{\beta<\alpha} T_P \uparrow \beta & T_P \downarrow \alpha = \bigcap_{\beta<\alpha} T_P \downarrow \beta \quad \text{if } \alpha \text{ is a limit ordinal} \end{cases}$$

Being monotonic, the *Knaster-Tarksi theorem* ensures that T_P admits a least and a greatest fixpoint, denoted as $lfp(T_P)$ and $gfp(T_P)$, respectively. Being upward continuous, from *Kleene's fixpoint Theorem*, it follows that: $lfp(T_P) = T_P \uparrow \omega$, where ω is the first limit ordinal. $lfp(T_P)$ coincides with the minimum Herbrand model of P, which itself is equal to the set of computed answers. The equality $lfp(T_P) = T_P \uparrow \omega$ means that if an atom belongs to this set we are able to prove this fact with a finite (although of a-priori unbounded length) proof.

Instead, $gfp(T_P) = T_P \downarrow \alpha$ for some ordinal α. The fact that T_P is not downward continuous means that α can be an ordinal "larger" than ω. "*This asymmetry is one of the most curious phenomena in the theory of logic programming*" [8]. The asymmetry is clearly pointed out by the following classical example (from [19, page 37]):

Example 5 *Let P be the following program:* $q(s(X)) \leftarrow q(X).$ $\quad$ $p(\mathbf{z}) \leftarrow q(X).$
Let us observe that: $T_P \downarrow 0 = \{p(\underline{0}), p(\underline{1}), p(\underline{2}), \ldots, q(\underline{0}), q(\underline{1}), q(\underline{2}), \ldots\}$, then for $i > 0$ it holds that $T_P \downarrow i = \{p(\underline{0}), q(\underline{i}), q(\underline{i+1}), q(\underline{i+2}), \ldots\}$. The unique atom that belongs to all these sets, and therefore to their intersection $T_P \downarrow \omega$, is $p(\underline{0})$. Thus, we have that $T_P \downarrow \omega = \{p(\underline{0})\}$ but it is not a fix-point. In fact, $T_P(\{p(\underline{0})\}) = \emptyset = T_P(\emptyset)$. For computing $gfp(T_P) = \emptyset$ a transfinite computation $T_P \downarrow \omega + 1$ is needed.

The example can be generalized to show that $\omega + 1$ cannot be enough (c.f. Theorem 1).

Infinite Terms Universe. Let us analyze the co-Herbrand Universe. One can define the T_P in the same way as in Equation 1, provided B_P is replaced by co-B_P and co-ground(P) is considered. To avoid ambiguity, let us refer to this operator as T_P^{co}. Iteration of T_P^{co} is defined in the same way as for T_P.

The "pure Prolog" case. The properties of T_P^{co} are studied for definite clause programs in [19, §26]. In particular, it is stated that: $lfp(T_P^{\mathsf{co}}) = T_P^{\mathsf{co}} \uparrow \omega$ and $gfp(T_P^{\mathsf{co}}) = T_P^{\mathsf{co}} \downarrow \omega$. The second property can be derived from the compactness of the metric space $\langle \mathsf{co}\text{-}U_P, \delta \rangle$, where δ is the standard metric between finitely branching trees with possible infinite depth.

Example 6 *Let us consider again the program in Example 5. Since $\Omega = s(\Omega)$, assuming $q(\Omega)$, then $q(s(\Omega)) = q(\Omega)$ is justified by the first clause (while the behavior of other atoms remains the same). In this case $T_P^{\mathsf{co}} \downarrow \omega = \{q(\Omega), p(\underline{0})\}$ is also a fixpoint ($p(\underline{0})$ is justified by the second clause since we have $q(\Omega)$).*

The Prolog II case. Let us assume a weak form of negation in the language: we allow the use of a $\neq$ (constraint) predicate between terms. Let us analyze the following example.

Example 7 *Let P be the following program.* $p(\mathbf{z}) \leftarrow q(X).$ $\quad$ $q(s(X)) \leftarrow q(X), X \neq s(X).$
Being $\Omega = s(\Omega)$, $q(\Omega)$ is no longer supported by the second clause, and therefore: $T_P^{\mathsf{co}} \downarrow \omega = \{p(\underline{0})\}$. Since $T_P^{\mathsf{co}}(T_P^{\mathsf{co}} \downarrow \omega) = \emptyset$, again the fixpoint is not reached in ω steps.

This means that as soon as the language has the capability of expressing inequality (as it happens in Constraint Logic Programming), the asymmetry between upward and downward computations emerges also considering infinite term universes.

Remark 1 *In the standard minimum model semantics, once $\mathcal{F}$ is known, the $\neq$ predicate above can be defined by a definite clause program without using negation. E.g., if $\mathcal{F} = \{\mathbf{z}/0, s/1\}$:*
$\mathbf{z} \neq s(X).$ $\quad$ $s(X) \neq \mathbf{z}.$ $\quad$ $s(X) \neq s(Y) \leftarrow X \neq Y.$ *We can also state when two terms are equal (using only one fact and unification): $X = X$. However, when computing $T_P^{\mathsf{co}} \downarrow \alpha$, $\Omega \neq \Omega$ is supported by the last clause. Therefore, both $\Omega = \Omega$ and $\Omega \neq \Omega$ belong to $gfp(T_P^{\mathsf{co}})$.*

4 Partitioning the Herbrand Base

The sets B_P and co-B_P can be partitioned into four subsets, according to the properties of the T_P, T_P^{co} operators, respectively. Figure 2 shows this partition for B_P (the case of co-B_P is analogous whenever T_P is replaced by T_P^{co}—although for definite clause programs $T_P^{\mathsf{co}} \downarrow \omega = gfp(T_P^{\mathsf{co}})$). In the remainder of the section we characterize these sets operationally and from the computability point of view. For the finite tree case, see also [8], for the infinite tree see [18, §6].

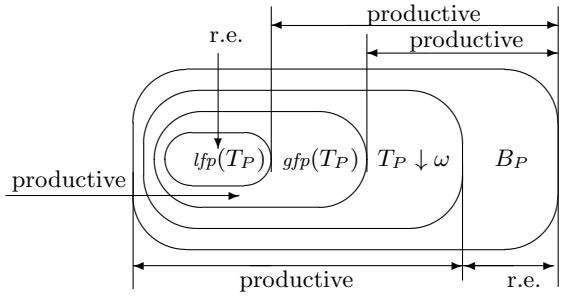

Figure 2: Venn Diagram of B_P and co-B_P induced by the T_P operator

The finite tree case. Let us focus on the $\vdash$ derivation rule. It is well-known that:

- The Success Set $SS = T_P \uparrow \omega = lfp(T_P)$ is the set of ground atoms that admit a successful SLD derivation. SS is the standard semantics of a logic program and coincides with the minimum Herbrand model.
- The Finite Failure set $FF = B_P \setminus T_P \downarrow \omega$ is the set of ground atoms for which there exists a finitely failed SLD-tree (note, however, that not all SLD trees for a given atom are required to fail finitely, depending on the selection rule).
- $IS = gfp(T_P) \setminus T_P \uparrow \omega$ is the set of ground atoms that admit neither a successful derivation, nor a finitely failed SLD-tree, but they admit an infinite ground derivation (i.e. an SLD derivation of unbounded length, where only ground instances of program clauses are used).
- $IF = T_P \downarrow \omega \setminus gfp(T_P)$ is the set of ground atoms that admit neither a successful derivation, nor a finitely failed SLD-tree, nor an infinite ground derivation, but they admit an infinite SLD derivation.

We also recall that: $B_P \setminus T_P \uparrow \omega$ is the set of atoms A for which there is no successful derivation (CWA rule for inferring $\neg A$ from P), and $B_P \setminus gfp(T_P)$ is the set of atoms A that belongs to *no* Herbrand model of the completion of P (Herbrand rule for inferring $\neg A$ from P).

Example 8 *Let us consider the definite clause program P:*

$$\mathtt{num}(\mathbf{z}). \qquad \mathtt{num}(s(X)) \leftarrow \mathtt{num}(X).$$
$$\mathtt{p}(\mathbf{z}) \leftarrow \mathtt{num}(X), \mathtt{q}(X). \qquad \mathtt{p}(s(X)) \leftarrow \mathtt{num}(X), \mathtt{p}(s(X)). \qquad \mathtt{q}(s(X)) \leftarrow q(X).$$

Then: $\bullet \mathtt{num}(\underline{n}) \in T_P \uparrow \omega$ *for all* $n \in \mathbb{N}$ $\quad \bullet \mathtt{q}(\underline{n}) \in B_P \setminus T_P \downarrow \omega$ *for all* $n \in \mathbb{N}$
$\bullet \mathtt{p}(\underline{n}) \in gfp(T_P) \setminus T_P \uparrow \omega$ *for all* $n \in \mathbb{N}$, $n > 0$ $\quad \bullet \mathtt{p}(\underline{0}) \in T_P \downarrow \omega \setminus gfp(T_P)$.

Theorem 1 *Given a definite clause program P, in general:*

1. *$T_P \uparrow \omega$ is r.e. complete*
2. *$B_P \setminus T_P \downarrow \omega$ is r.e. complete (and $T_P \downarrow \omega$ is a productive set)*
3. *$B_P \setminus gfp(T_P)$ is Π^1_1 complete (thus, $gfp(T_P)$ and $B_P \setminus gfp(T_P)$ are not r.e.)*

This result is due to Blair [9]. With "in general" we mean that there can be cases when the sets are simpler (e.g., when there are no function symbols in the program P), but that there are cases where the situation is hard. In particular, as far as point (3) is concerned, Blair in [9] shows that there is a program P such that $gfp(T_P)$ is reached after $\omega + \omega$ iterations and that $B_P \setminus gfp(T_P)$ is Π^1_1-complete.

The infinite tree case. Let us focus now on the infinite term universe. The just given characterization of the subsets of B_P in terms of $\vdash$-derivations is the same reported for co-B_P by Jaffar and Stuckey in [18] in the universe of finite and infinite terms for their derivation rule $\vdash_\pi$ (c.f., Proposition 1). We recall that, in the absence of $\neq$ constraints, $T_P^{\mathsf{co}} \downarrow \omega = gfp(T_P^{\mathsf{co}})$ (hence, the set IF is empty).

Example 9 *Let us consider the program P of Example 8 and analyze the partition of the complete Herbrand Base. We have that $T_P^{\mathsf{co}} \uparrow \omega = T_P \uparrow \omega$. Moreover, $T_P^{\mathsf{co}} \downarrow \omega \setminus gfp(T_P^{\mathsf{co}}) = \emptyset$ and $gfp(T_P^{\mathsf{co}}) \setminus lfp(T_P^{\mathsf{co}}) = \{\mathtt{p}(\underline{n}) : n \in \mathbb{N}\} \cup \{\mathtt{num}(\Omega), \mathtt{p}(\Omega), \mathtt{q}(\Omega)\}$.*

Example 10 *An interesting and simple example is the following (the constant* $\mathbf{z}$ *is added to* $\mathcal{F}$*):*

$$p(X) \leftarrow p(s(X)).$$

In this case $T_P^{\mathsf{co}} \uparrow \omega = \emptyset = B_P \setminus T_P^{\mathsf{co}} \downarrow \omega = T_P^{\mathsf{co}} \downarrow \omega \setminus \mathit{gfp}(T_P^{\mathsf{co}})$*. The only non-empty Venn region is* $\mathit{gfp}(T_P^{\mathsf{co}}) \setminus \mathit{lfp}(T_P^{\mathsf{co}})$ *which coincides with* $\mathit{gfp}(T_P^{\mathsf{co}}) = \mathsf{co}\text{-}B_P = \{p(\Omega), p(\underline{0}), p(\underline{1}), p(\underline{2}), p(\underline{3}), \ldots\}$*. Note that in this case* $\Upsilon(\mathsf{co}\text{-}B_P) = \mathsf{co}\text{-}B_P$*, hence* $\Upsilon(\mathit{gfp}(T_P^{\mathsf{co}})) = \mathit{gfp}(T_P^{\mathsf{co}}) = \mathit{gfp}(T_P^{\mathsf{rat}})$ *(where* T_P^{rat} *denotes* T_P *restricted to* $\Upsilon(\mathsf{co}\text{-}B_P)$*); however, only* $p(\Omega)$ *has an infinite "rational" ground derivation.*

Example 11 *Let us consider the program* P

$$\mathtt{inf\_list}(X, [X|L]) \leftarrow \mathtt{inf\_list}(s(X), L). \qquad p(\mathbf{z}) \leftarrow \mathtt{inf\_list}(\mathbf{z}, L).$$

Obviously, $\mathit{lfp}(T_P^{\mathsf{co}}) = \emptyset$*. We have that*

$$\begin{array}{l}\langle \emptyset \,\square\, p(\mathbf{z})\rangle \vdash_\infty \langle \emptyset \,\square\, \mathtt{inf\_list}(\mathbf{z}, L)\rangle \vdash_\infty \langle \{L = [\mathbf{z}|L_1]\} \,\square\, \mathtt{inf\_list}(s(\mathbf{z}), L_1)\rangle \vdash_\infty \\ \langle \{L = [\mathbf{z}|L_1], L_1 = [s(\mathbf{z})|L_2]\} \,\square\, \mathtt{inf\_list}(s(s(\mathbf{z})), L_2)\rangle \vdash_\infty \cdots \vdash_\infty \\ \langle \{L = [\underline{0}|L_1], L_1 = [\underline{1}|L_2], \ldots, L_n = [\underline{n}|L_{n+1}]\} \,\square\, \mathtt{inf\_list}(\underline{n+1}, L_{n+1})\rangle\end{array}$$

The set $\mathit{lfp}(T_P)$ *is empty, while the infinite, non-rational atom* $\mathtt{inf\_list}(\underline{0}, [\underline{0}, \underline{1}, \underline{2}, \ldots])$*, as well as the atom* $p(\mathbf{z})$ *belong to* $\mathit{gfp}(T_P) \setminus \mathit{lfp}(T_P) = \mathit{gfp}(T_P)$*. Similarly, for any term* t *of the signature, we have that* $\mathtt{inf\_list}(t, [t, s(t), s(s(t)), s(s(s(t))), \ldots]) \in \mathit{gfp}(T_P) \setminus \mathit{lfp}(T_P) = \mathit{gfp}(T_P)$*. Being* $\Omega = s(\Omega)$*, in particular,* $\mathtt{inf\_list}(\Omega, [\Omega, \Omega, \ldots]) \in \mathit{gfp}(T_P) \setminus \mathit{lfp}(T_P) = \mathit{gfp}(T_P)$*, therefore* $\mathtt{inf\_list}(\Omega, [\Omega, \Omega, \ldots])$ *and* $p(\mathbf{z})$ *are the only atoms in that set. All other rational atoms belong to* $\mathsf{co}\text{-}B_P \setminus T^{\mathsf{co}} \downarrow \omega$*. However, whereas for* $\mathtt{inf\_list}(\Omega, [\Omega, \Omega, \Omega, \ldots])$ *there exists an infinite but "rational" ground derivation, for* $p(\mathbf{z})$ *there are no "rational" ground derivations; furthermore* $\mathtt{inf\_list}(\Omega, [\Omega, \Omega, \ldots]) \in \mathit{gfp}(T_P^{\mathsf{rat}})$*, but* $p(\mathbf{z}) \notin \mathit{gfp}(T_P^{\mathsf{rat}})$*. Finally,* $T_P^{\mathsf{rat}} \downarrow \omega \setminus \mathit{gfp}(T_P^{\mathsf{rat}}) = \{p(\mathbf{z})\}$*; this is related to the fact that* $\Upsilon(\mathsf{co}\text{-}B_P)$ *with the standard distance* δ *fails to be a complete metric space.*

Let us end this section with a computability result on the rational part of these sets.

Theorem 2 *Given a definite clause program* P*, in general:*

1. $\Upsilon(T_P^{\mathsf{co}} \uparrow \omega)$ *is r.e. complete*
2. $\Upsilon(\mathsf{co}\text{-}B_P \setminus \mathit{gfp}(T_P^{\mathsf{co}}))$ *is r.e. complete (hence,* $\Upsilon(\mathit{gfp}(T_P^{\mathsf{co}}))$ *is productive).*

Proof. The "positive" part of the theorem is a consequence of [18]: checking whether a rational ground atom A belongs to $T_P^{\mathsf{co}} \uparrow \omega$ or to $\mathsf{co}\text{-}B_P \setminus T_P^{\mathsf{co}} \downarrow \omega$ are in fact semi-decidable properties. In the former case it is sufficient to detect a successful $\vdash_\infty$-derivation; in the latter case it amounts to verifying that there exists $n \in \mathbb{N}$ s.t. A does not belong to $T_P^{\mathsf{co}} \downarrow n$: this can be achieved by checking that the SLD tree obtained with a selection rule that mimics the $\vdash_\pi$-selection rule (c.f. Proposition 1) finitely fails.

Starting from a goal with rational terms, only rational terms are generated during a finite (successful or not) computation. We omit the Υ symbol in the rest of the proof for readability.

To prove the r.e. completeness of the two sets, we reduce the set K to each of them. Let us consider the recursive total function:

$$\varphi(x, y) \quad = \quad \begin{cases} 0 & \text{If } M_x(x) \not\downarrow \text{ in} \leq y \text{ steps} \\ 1 & \text{If } M_x(x) \downarrow \text{ in} \leq y \text{ steps} \end{cases}$$

By Turing completeness of definite clause programming under the *lfp* semantics, we can write a program defining a predicate h (halting) such that:

$$\begin{array}{ll} \leftarrow h(\underline{x}, \underline{y}, \underline{z}) & \text{has a finite successful derivation if } \varphi(x, y) = z,\ z \in \{0, 1\} \\ \leftarrow h(\underline{x}, \underline{y}, \underline{z}) & \text{has a finite failure tree if } (z \in \{0, 1\} \text{ and } \varphi(x, y) \neq z) \text{ or } z \notin \{0, 1\} \end{array}$$

Let P_φ be the program defining h containing the predicate definition: $\mathtt{num}(\mathbf{z}). \quad \mathtt{num}(s(X)) \leftarrow \mathtt{num}(X).$

1) We define a program P_1 and show that $K \leq T_{P_1}^{\mathsf{co}} \uparrow \omega$. Let us extend the program P_φ with the clause (assume, w.l.o.g., that r does not appear in P_φ):

$$r(X) \leftarrow \mathtt{num}(N), h(X, N, s(\mathbf{z})).$$

Let P_1 be the program obtained. We prove that $x \in K$ iff $r(\underline{x}) \in T^{\mathsf{co}}_{P_1} \uparrow \omega$.

If $x \in K$, then there is a number t such that $\leftarrow h(\underline{x}, \underline{t}, s(\mathbf{z}))$ has a successful derivation. Therefore $\leftarrow r(\underline{x})$ has a successful derivation, and hence $r(\underline{x}) \in T^{\mathsf{co}}_{P_1} \uparrow \omega$.

If $x \in \bar{K}$ the computation $\leftarrow r(\underline{x})$ has no successful derivations, hence, $r(\underline{x}) \notin T^{\mathsf{co}}_{P_1} \uparrow \omega$.

2) We define a program P_2 and show that $\bar{K} \leq \mathit{gfp}(T^{\mathsf{co}}_{P_2})$ (i.e., $x \in \bar{K}$ iff $q(\underline{x}) \in \mathit{gfp}(T^{\mathsf{co}}_{P_2})$, equivalent to prove that $K \leq \mathsf{co}\text{-}B_{P_2} \setminus \mathit{gfp}(T^{\mathsf{co}}_{P_2})$). P_2 is P_φ extended with the following definitions of the two predicates p, q (assume, w.l.o.g., that q and p do not appear in P_φ):

$$q(X) \quad \leftarrow \quad p(X, \mathbf{z}). \qquad p(X, Y) \quad \leftarrow \quad h(X, Y, \mathbf{z}), p(X, s(Y)).$$

If $x \in \bar{K}$, $h(\underline{x}, \underline{y}, \underline{0})$ has a successful derivation for all $y \in \mathbb{N}$ and therefore there is an infinite ground derivation for $q(\underline{x})$: $q(\underline{x}) \in \mathit{gfp}(T^{\mathsf{co}}_{P_2})$.

If $x \in K$, let t be the number of steps such that $M_x(x)$ terminates: the subgoal $\leftarrow h(\underline{x}, \underline{t}, \underline{0})$ fails, hence all derivations for the overall goal are failing: $q(\underline{x}) \notin \mathit{gfp}(T^{\mathsf{co}}_{P_2})$. □

Let us observe that if the language includes the built-in $\neq$ predicate on terms, as in Prolog II, then, in general, $T^{\mathsf{co}}_P \downarrow \omega \neq \mathit{gfp}(T^{\mathsf{co}}_P)$ and the complement of *gfp* is no longer r.e. (it follows immediately from Blair's result, Theorem 1).

Let us briefly reason on non-rational atoms. Their cardinality is more than denumerable. A denumerable subset of them can be "defined" by a program (e.g., Example 11, where we have only ground infinite computations for $A = \mathtt{inf\_list}(0, [\underline{0}, \underline{1}, \underline{2}, \ldots]))$: $A \in \mathit{gfp}(T^{\mathsf{co}}_P)$ but there is no way to verify it. In other cases all terms are implicitly referred, like when we have a fact $p(X)$. In general we are not able to formalize the test $A \in S$ for a non-rational term. Notions of computation on non-rational terms require different starting points and are outside the scope of this paper.

5 co-Logic Programming

Coinductive Logic Programming (or simply co-LP) is introduced in [31]. A co-LP program, syntactically, is a *definite clause program*. The difference w.r.t. "standard" Logic Programming is in the semantics. In "standard" Logic Programming, the semantics is based on $\mathit{lfp}(T_P)$ a r.e. complete set, in general (c.f. Theorem 1). For r.e. sets membership can be finitely verified. This set coincides with the minimum Herbrand Model, which is, in turn, exactly the set of logical consequences of the program P. The semantics of co-LP, instead is based on the $\mathit{gfp}(T^{\mathsf{co}}_P)$. Thus, we consider the complete Herbrand Universe and Base and the operator T^{co}_P:

Definition 1 (co-LP model-theoretical semantics) *Let P be a definite clause program.*

- *Let $a \in \mathsf{co}\text{-}B_P$. We say that $P \models^{\mathsf{co}} a$ if and only if $a \in \mathit{gfp}(T^{\mathsf{co}}_P)$.*
- *Let A be a finite atom. We say that $P \models^{\mathsf{co}} A$ if and only if for all tree substitutions $\gamma : \mathsf{FV}(A) \longrightarrow \mathsf{co}\text{-}U_P$, it holds that $P \models^{\mathsf{co}} A\gamma$.*
- *$\models^{\mathsf{co}}$ is extended to first-order formulas ($\neg, \vee, \exists$) as usual.*

The following theorem proves the negative result suggested by Example 10 and 11: there is no complete procedure for deciding whether $P \models^{\mathsf{co}} a$, even when a is rational (that is, it can be represented in a finite way). This is suggested, for instance, by the program P in Example 10, and the rational atom $p(\underline{0})$: we have $P \models^{\mathsf{co}} p(\underline{0})$, but the corresponding derivation is infinite and contain infinitely different rational terms.

Theorem 3 (Inherent incompleteness of co-LP) *Even restricting to rational terms, no complete procedure exists for establishing whether $P \models^{\mathsf{co}} a$.*

Proof. The rational part of $\mathit{gfp}(T^{\mathsf{co}}_P)$ is, in general, productive (Theorem 2). This means that there is no computable procedure capable of verifying whether a rational atom belongs to $\Upsilon(\mathit{gfp}(T^{\mathsf{co}}_P))$ (even when it is the case) that works for all atoms, otherwise $\Upsilon(\mathit{gfp}(T^{\mathsf{co}}_P))$ would be r.e., a contradiction. □

Anyway, any productive set has an infinite r.e. subset. A procedure would aim at identifying such a subset(s), thus computing an under-approximation of $\mathit{gfp}(T^{\mathsf{co}}_P)$ (possibly, including strictly $\mathit{lfp}(T^{\mathsf{co}}_P)$). The procedure $\vdash^{\mathsf{co}}$ defined below and the original procedure in [31] aim at identifying such a set. However, if the procedures finitely fails, then they correctly state that the atom does not belong to $\mathit{gfp}(T^{\mathsf{co}}_P)$.

A computable procedure $\vdash_{\mathsf{co}}$ that approximates $\models_{\mathsf{co}}$ should behave as follows. Given a goal G it should return "no" or a non empty set of computed answer systems of equations (briefly, c.a.s.) $\{\theta_1, \theta_2, \ldots\}$ for $\mathsf{FV}(G)$, such that for all $i \in \{1, 2, \ldots\}$ $P \models_{\mathsf{co}} G\theta_i$. As for rule $\vdash_\infty$, since infinite trees must be considered, the c.a.s. needs to be implicitly stored. Therefore, each θ_i can be a finite set of equations identifying some rational tree, or, theoretically, can be an infinite set of equations identifying a possibly non-rational tree, with variables in it.

The procedure in [31] is based on a notion of rewriting between pairs of trees and states, where trees are dynamic data structures resembling the execution of a resolution procedure (like SLD resolution) and a state is a set of equations. We re-formulate the same concept with an alternative notion at a (slightly) higher level using a notion of goal with hypotheses (a similar notion is introduced, in a different context in [10, 11]). In Appendix A we prove the equivalence of the two proposals.

A *hypothetical goal* is a list of pairs (A, S) where A is an atom and S is a set of atoms (hypotheses), together with a system of equations. Given a goal $G = \langle E \,\square\, A_1, \ldots, A_u\rangle$, where E is a set of equations (needed to define rational terms) and A_i are program defined atoms, we define the corresponding hypothetical goal $\alpha(G) = \langle (A_1, \emptyset), \ldots, (A_u, \emptyset) \,\square\, E\rangle$.

Definition 2 (co-LP operational semantics (revisited)) *Given a definite clause program P and a hypothetical goal $G = \langle (A_1, S_1), \ldots, (A_u, S_u) \,\square\, E\rangle$, the rewriting rule $\vdash_{\mathsf{co}}$ that leads G to G' (briefly $G \vdash_{\mathsf{co}} G'$) is defined as follows: choose (A_i, S_i) in G, let $A_i = p(s_1, \ldots, s_n)$. G' is computed in one of the two following ways:*

1. *let $p(t_1, \ldots, t_n) \leftarrow B_1, \ldots, B_m$ be a renaming of a clause in P with fresh variables, and let $E' = E \cup \{s_1 = t_1, \ldots, s_n = t_n\}$ be solvable. Let $S' = S_i \cup \{p(s_1, \ldots, s_n)\}$. Then:*
   $G' = \langle (A_1, S_1), \ldots, (A_{i-1}, S_{i-1}), (B_1, S'), \ldots, (B_m, S'), (A_{i+1}, S_{i+1}), \ldots, (A_u, S_u) \,\square\, E'\rangle$
2. *let $p(t_1, \ldots, t_n) \in S_i$ be such that $E' = E \cup \{s_1 = t_1, \ldots, s_n = t_n\}$ is solvable. Then:*
   $G' = \langle (A_1, S_1), \ldots, (A_{i-1}, S_{i-1}), (A_{i+1}, S_{i+1}), \ldots, (A_u, S_u) \,\square\, E'\rangle$

In this case we say that $G \vdash_{\mathsf{co}} G'$. A derivation for a goal G is a maximal sequence $G_1, G_2, G_3, \ldots$ s.t. $G_1 = \alpha(G)$ and $G_i \vdash_{\mathsf{co}} G_{i+1}$ for $i > 0$. A derivation is successful *if it is a finite sequence $G_1, \ldots, G_k$ s.t. $G_k = \langle \varepsilon \,\square\, E\rangle$ and E is a solvable system of equations. In this case, the computed answer system is the part of the m.g.u. of E relevant for the variables in G_1. A derivation is* failing *if it is a finite sequence $G_1, \ldots, G_k$ which is not successful (hence, there is no G' such that $G_k \vdash_{\mathsf{co}} G'$).*

We will write $\vdash_{\mathsf{co}\,1}$ ($\vdash_{\mathsf{co}\,2}$) if the first (second) rule is applied. The first rule is the standard resolution rule; moreover, the atom removed by resolution is added to the hypotheses of the subgoal. The second rule exploits previous hypotheses for justifying a goal by co-induction. In a sense, rule (1) leads to $lfp(T_P)$, rule (2) aims to capture (some) atoms belonging to $gfp(T_P) \setminus lfp(T_P)$ that are co-inductively supported.

Example 12 *Let us consider the program of Example 5 and the goal $\leftarrow p(\mathbf{z})$. The following is a successful derivation:*

$$\begin{array}{l}\langle (p(\mathbf{z}), \emptyset) \,\square\, \emptyset\rangle \\ \vdash_{\mathsf{co}\,1} \langle (q(X_0), \{p(0)\}) \,\square\, \emptyset\rangle \\ \vdash_{\mathsf{co}\,1} \langle (q(X_1), \{p(0), q(X_0)\}) \,\square\, \{X_0 = s(X_1)\}\rangle \\ \vdash_{\mathsf{co}\,2} \langle \varepsilon \,\square\, \{X_0 = s(X_1), X_0 = X_1\}\rangle\end{array}$$

The system $E = \{X_0 = s(X_1), X_0 = X_1\}$ is solvable and its m.g.u. is $\theta = \{X_0 = X_1, X_1 = s(X_1)\}$. The computed answer system is simply the empty set, being the initial ground goal.

Example 13 *Let P_1 consist of the following clause $p([\mathbf{z}, s(\mathbf{z})|X]) \leftarrow p(X)$ and let us analyze the following two successful derivations for the goal $\leftarrow p(X)$.*

$$\begin{array}{l}\langle (p(X), \emptyset) \,\square\, \emptyset\rangle \\ \vdash_{\mathsf{co}\,1} \langle (p(X_1), \{p(X)\}) \,\square\, \{X = [\mathbf{z}, s(\mathbf{z})|X_1]\} \\ \vdash_{\mathsf{co}\,2} \langle (\varepsilon \,\square\, \{X = X_1, X = [\mathbf{z}, s(\mathbf{z})|X_1]\}\rangle\end{array}$$

which yields the computed answer system $\{X = [\mathbf{z}, s(\mathbf{z})|X]\}$ and:

$$\begin{array}{l}\langle (p(X), \emptyset) \,\square\, \emptyset\rangle \\ \vdash_{\mathsf{co}\,1} \langle (p(X_1), \{p(X)\}) \,\square\, \{X = [\mathbf{z}, s(\mathbf{z})|X_1]\}\rangle \\ \vdash_{\mathsf{co}\,1} \langle (p(X_2), \{p(X), p(X_1)\}) \,\square\, \{X = [\mathbf{z}, s(\mathbf{z})|X_1], X_1 = [\mathbf{z}, s(\mathbf{z})|X_2]\}\rangle \\ \vdash_{\mathsf{co}\,2} \langle \varepsilon \,\square\, \{X = X_2, X = [\mathbf{z}, s(\mathbf{z})|X_1], X_1 = [\mathbf{z}, s(\mathbf{z})|X_2]\}\rangle\end{array}$$

which yields the computed answer system $\{X = [\mathbf{z}, s(\mathbf{z}), \mathbf{z}, s(\mathbf{z})|X]\}$. An infinite number of successful computations (actually, with the same answer, in this special case) can be computed.

Example 14 *Let P_2 consist of the following two clauses $p([\mathbf{z}|X]) \leftarrow p(X)$. $p([s(\mathbf{z})|X]) \leftarrow p(X)$. Let us denote with c_1 and c_2 the two clauses and let us analyze the following successful derivation for the goal $\leftarrow p(X_0)$ where we add the information on the clause selected in a $\vdash_{\mathsf{co}\,1}$ step:*

$$\langle (p(X_0), \emptyset) \,\square\, \emptyset \rangle$$
$$\vdash_{\mathsf{co}\,1}^{c_1} \langle (p(X_1), \{p(X_0)\}) \,\square\, \{X_0 = [\mathbf{z}|X_1]\} \rangle$$
$$\vdash_{\mathsf{co}\,1}^{c_2} \langle (p(X_2), \{p(X_0), p(X_1)\}) \,\square\, \{X_0 = [\mathbf{z}|X_1], X_1[s(\mathbf{z})|X_2]\} \rangle$$
$$\vdash_{\mathsf{co}\,1}^{c_1} \langle (p(X_3), \{p(X_0), p(X_1), p(X_2)\}) \,\square\, \{X_0 = [\mathbf{z}|X_1], X_1[s(\mathbf{z})|X_2], X_2 = [\mathbf{z}|X_3]\} \rangle$$
$$\vdash_{\mathsf{co}\,1}^{c_1} \langle (p(X_4), \{p(X_0), p(X_1), p(X_2), p(X_3)\}) \,\square\, \{X_0 = [\mathbf{z}|X_1], X_1[s(\mathbf{z})|X_2], X_2 = [\mathbf{z}|X_3], X_3 = [\mathbf{z}|X_4]\} \rangle$$
$$\vdash_{\mathsf{co}\,1}^{c_2} \langle (p(X_5), \{p(X_0), p(X_1), p(X_2), p(X_3), p(X_4)\}) \,\square\, \{X_0 = [\mathbf{z}|X_1], X_1[s(\mathbf{z})|X_2], X_2 = [\mathbf{z}|X_3], X_3 = [\mathbf{z}|X_4], X_4 = [s(\mathbf{z})|X_5]\} \rangle$$
$$\vdash_{\mathsf{co}\,2}^{c_2} \langle \varepsilon \,\square\, \{X_0 = [\mathbf{z}|X_1], X_1[s(\mathbf{z})|X_2], X_2 = [\mathbf{z}|X_3], X_3 = [\mathbf{z}|X_4], X_4 = [s(\mathbf{z})|X_5], X_2 = X_5\} \rangle$$

which yields the computed answer system $\{X = [\mathbf{z}, s(\mathbf{z})|X_2], X_2 = [\mathbf{z}, \mathbf{z}, s(\mathbf{z})|X_2]\}$. In general, $p(X)$ with X bound to any infinite lists of the form $X = [a_1, a_2, \ldots, a_m, b_1, \ldots, b_n, b_1, \ldots, b_n, b_1, \ldots, b_n, \ldots]$ where $m \geq 0, n > 0$ and $a_i \in \{\underline{0}, \underline{1}\}, b_j \in \{\underline{0}, \underline{1}\}$ with a finite possibly empty prefix and an infinite periodic suffix can be computed by a $\vdash_{\mathsf{co}}$ derivation.

Theorem 4 (Correctness) *Let P be a definite clause program. If there is a $\vdash_{\mathsf{co}}$ derivation for $G_0 = \langle (A, \emptyset) \,\square\, E \rangle$ with c.a.s. θ, then $P \models_{\mathsf{co}} A\gamma$ for every term substitution γ solution of $E\theta$.*

Proof. We define *derivation trees* whose nodes are labeled by atoms, related to a non failing derivation as follows: an edge (x, y) occurs if node x is labeled by atom p which was selected for application of rule $\vdash_{\mathsf{co}\,1}$ and y is labeled by an atom q_i of the body of the clause used. Leaves of the tree are marked either by true (when a fact has been used for removing the atom using rule $\vdash_{\mathsf{co}\,1}$) or by co (when the coinductive rule $\vdash_{\mathsf{co}\,2}$ has been used).

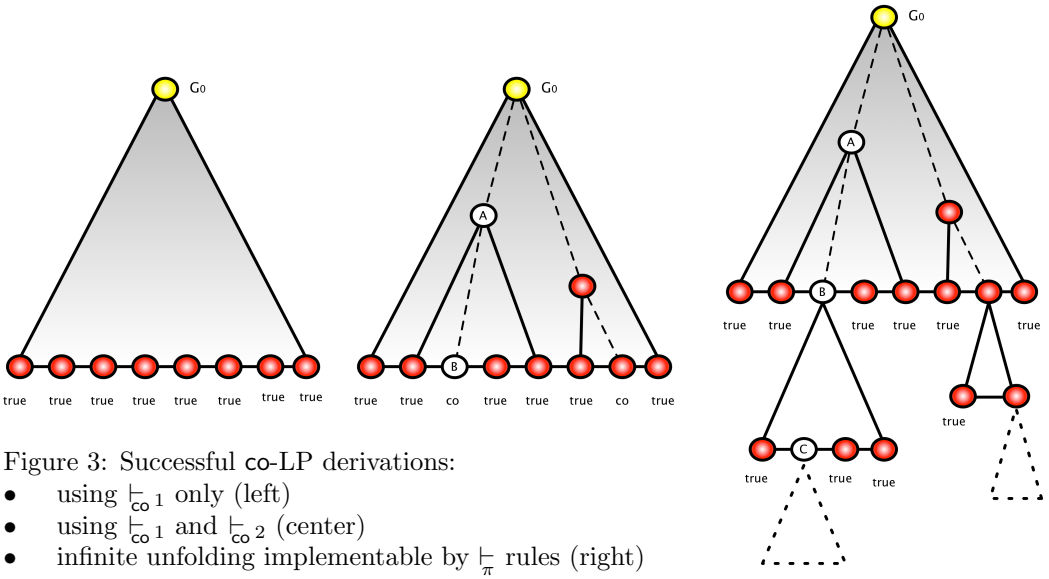

Figure 3: Successful co-LP derivations:
- using $\vdash_{\mathsf{co}\,1}$ only (left)
- using $\vdash_{\mathsf{co}\,1}$ and $\vdash_{\mathsf{co}\,2}$ (center)
- infinite unfolding implementable by $\vdash_{\pi}$ rules (right)

Let us assume that in a successful derivation only rule $\vdash_{\mathsf{co}\,1}$ is applied. Then the derivation tree is of the form reported in Figure 3 (left). By a breadth-first visit we can build a $\vdash_{\pi}$ successful derivation of the goal G_0, therefore $A\gamma \in T_P^{\mathsf{co}} \uparrow \omega$ by correctness of Prolog II procedure [18] (c.f., also, Proposition 1).

If, instead, rule $\vdash_{\mathsf{co}\,2}$ is employed at least once, we are in a situation of the kind reported in Figure 3 (center). The proof we propose is similar to that of the *pumping lemma*: by unfolding we will show that we are able to produce an infinite $\vdash_{\pi}$ derivation and therefore: $A\gamma \in T_P^{\mathsf{co}} \downarrow \omega$ [18].

For simplicity, let us consider a leftmost selection rule and let us consider the first leaf marked by co encountered in a depth-first search (c.f., Figure 3—node B).

The derivation can be split in three parts.

1. A first part of the proof (possibly of length zero), starting from $G_0 = \langle (A, \emptyset) \,\square\, E \rangle$ in which only $\vdash_{\mathsf{co}\,1}$ is used that ends where the atom $p(\vec{t})$, later used for the first co-inductive hypothesis, is selected (c.f., Figure 3—node A).

Namely, we have $\langle (A, \emptyset) \,\square\, E\rangle \vdash_{\mathsf{co}\,1} \cdots \vdash_{\mathsf{co}\,1} \langle (p(\vec{t}), H), \ldots \,\square\, E_k\rangle$.
2. A part of the proof starting from the previous described goal in which only $\vdash_{\mathsf{co}\,1}$ rule is applied until the first occurrence of rule $\vdash_{\mathsf{co}\,2}$ is applied (node B):

$$\langle (p(\vec{t}), H), \ldots \,\square\, E_k\rangle \vdash_{\mathsf{co}\,1} \langle (B_1, H \cup \{p(\vec{t})\}), \ldots, (B_n, H \cup \{p(\vec{t})\}) \ldots \,\square\, E_k \cup \{\vec{t} = \vec{a}\}\rangle \vdash_{\mathsf{co}\,1}$$
$$\cdots \vdash_{\mathsf{co}\,1} \langle (p(\vec{s}), H \cup \{p(\vec{t})\} \cup H'), \vec{B} \,\square\, E_k \cup \{\vec{t} = \vec{a}\} \cup E'\rangle \vdash_{\mathsf{co}\,2} \langle \vec{B} \,\square\, E_k \cup \{\vec{t} = \vec{a}, \vec{s} = \vec{t}\} \cup E'\rangle$$

where in the first derivation step the (renamed) clause $p(\vec{a}) \leftarrow B_1, \ldots, B_n$ is used.
3. The rest of the proof, that leads to $\langle \varepsilon \,\square\, E_k \cup \{\vec{t} = \vec{a}, \vec{s} = \vec{t}\} \cup E' \cup E''\rangle$. In this last part of the proof, either $\vdash_{\mathsf{co}\,1}$ or $\vdash_{\mathsf{co}\,2}$ can be used.
Let us focus on the part (2) of the derivation. For simplicity of notation, let us assume that it is of length 2, namely that the atom B_1 is of the form $q(\vec{d})$ and that at the first derivation the renamed rule $q(\vec{e}) \leftarrow p(\vec{s}), \ldots$ is used. This means that the above part of the derivation is of the form:

$$\begin{array}{ll}
\langle (p(\vec{t}), H), \ldots \,\square\, E_k\rangle \vdash_{\mathsf{co}\,1} & (p(\vec{a}) \leftarrow q(\vec{d}), \ldots) \\
\langle (q(\vec{d}), H \cup \{p(\vec{t})\}), \ldots \,\square\, E_k \cup \{\vec{t} = \vec{a}\}\rangle \vdash_{\mathsf{co}\,1} & (q(\vec{e}) \leftarrow p(\vec{s}), \ldots) \\
\langle (p(\vec{s}), H \cup \{p(\vec{t}), q(\vec{d})\}), \ldots \,\square\, E_k \cup \{\vec{t} = \vec{a}, \vec{d} = \vec{e}\}\rangle \vdash_{\mathsf{co}\,2} & \\
\langle \ldots \,\square\, E_k \cup \{\vec{t} = \vec{a}, \vec{d} = \vec{e}, \vec{s} = \vec{t}\}\rangle &
\end{array}$$

We prove that the application of the $\vdash_{\mathsf{co}\,2}$ rule can be replaced by the application of rule $\vdash_{\mathsf{co}\,1}$ with a renamed version of $p(\vec{a}) \leftarrow q(\vec{d}), \ldots$ and then $\vdash_{\mathsf{co}\,1}$ with a renamed version of $q(\vec{e}) \leftarrow p(\vec{s}), \ldots$ and so on forever. Let θ be the renaming substitution that transforms $p(\vec{a}) \leftarrow q(\vec{d}), \ldots$ into its new renamed version (θ introduces fresh variables). We know from application of rule $\vdash_{\mathsf{co}\,2}$ that $\{\vec{t} = \vec{a}, \vec{d} = \vec{e}, \vec{s} = \vec{t}\} \cup E_k$ is solvable, therefore, by equality axioms, also $\{\vec{s} = \vec{a}, \vec{d} = \vec{e}, \vec{s} = \vec{t}\}$ is solvable. Since θ is a variable renaming with fresh variables, we have that $\{\vec{s} = \vec{a}\theta, \vec{t} = \vec{a}, \vec{d} = \vec{e}\} \cup E_k$ is solvable. Therefore rule $\vdash_{\mathsf{co}\,1}$ can be applied again (and the new selected atom is $q(\vec{d}\theta)$. Again, since $\vec{d} = \vec{e}$, positive fresh variables of the rule $q(\vec{e}) \leftarrow p(\vec{s}), \ldots$ can be applied and this can be repeated forever.

This is graphically represented by the repetition of subtrees in Figure 3 (right). The same idea can be repeated to all other leaves marker by co. By a breadth-first visit of the tree we can build the infinite $\vdash_{\pi}$-derivation. □

In Theorem 3 we have proved the *incompleteness* of any procedure for computing $\models_{\overline{\mathsf{co}}}$, even restricting to rational terms. Let us see a typical example of a rational atom that is not computed.

From Example 10 (a program with the unique clause $p(X) \leftarrow p(s(X))$.) we learned that $p(\mathbf{z}) \in gfp(T_P^{\mathsf{co}})$. However, no finite derivation exists for it, even though all involved goals are rational (the infinite sequence contains rational terms that are all distinct, hence $\vdash_{\mathsf{co}\,2}$ rule can never be applied). The derivation will be of the form

$$\begin{array}{l}
\langle (p(\mathbf{z}), \emptyset) \,\square\, \emptyset)\rangle \\
\vdash_{\mathsf{co}\,1} \langle (p(s(X)), \{p(\mathbf{z})\}) \,\square\, \{X = \mathbf{z}\})\rangle \\
\vdash_{\mathsf{co}\,1} \langle (p(s(X_1)), \{p(\mathbf{z}), p(s(X))\}) \,\square\, \{X = \mathbf{z}, X_1 = s(X)\})\rangle \\
\vdash_{\mathsf{co}\,1} \cdots
\end{array}$$

The hypothetical goal $\langle (p(X), \emptyset) \,\square\, \{X = s(X)\}\rangle$, instead, admits a $\vdash_{\mathsf{co}}$-derivation.

5.1 Implementation Notes

A meta-interpreter which faithfully mimics rules $\vdash_{\mathsf{co}}$ can be downloaded from `clp.dimi.uniud.it`. Another possible implementation based on a big-step operational semantics is presented in the contribution [2]. We prove the equivalence of these two semantics in Appendix B. An implementation based on the big-step semantics has the advantage that co-inductive hypotheses are handled more efficiently since they are shared among atoms.

Two papers have been recently published in the time spanning the period between the appearance of the conference version and the completion of the revision of the journal version of this work, witnessing the interest toward this sub-paradigm of Logic Programming. Moura [24] presents a portable implementation of co-LP very close to [2] (both based on refinements of the Simon's original proposal [31]), which has been embedded in a tabled Prolog [20], exploiting tabling for storing hypothesis. The work of Moura relates to the implementation of Logtalk, an open source object-oriented logic programming language supporting co-LP; both [20] and [2] propose a more flexible implementation than that provided in [31], with similar mechanisms to allow the user to specify customized behavior in case of the application of the coinductive rule. In [2], the problem of finite/infinite failure of coinductive predicates is also tackled.

Except for these differences, all proposed implementations of the operational semantics (including that implemented by Simon et al. and in the `coinduction` library of SWI-Prolog) are based on the big-step semantics presented here; however, to our knowledge, no correctness proof of the big-step operational semantics, and, hence, of the implementation of co-LP has been provided before.

6 Conclusions

co-LP is an interesting emerging sub-paradigm of logic programming which is suitable for naturally modeling circularity and which can be fruitfully applied to several kinds of applications ranging over type inference, subtyping between recursive types, and, more generally, static analysis and symbolic execution of programs [5, 4, 6, 7, 3], verification of real time systems [27], bisimulation [14] and model checking [28], and SAT solvers [23].

In this paper we revisit and deepen some aspects of the foundations of co-LP [31, 29]; in particular, we provide two simpler but equivalent operational semantics (small- and big-step, respectively), whose proof of correctness can be directly derived from early results from Jaffar and Stuckey [18]. The small-step semantics is simpler than that originally defined [31], and is based on a natural extension of the notion of SLD derivation. The big-step semantics provides a simple specification of an interpreter of co-LP, as that supported by the `coinduction` library in SWI-Prolog. From the proof of equivalence of the small-step and big-step operational semantics, and the proof of soundness of the small-step semantics, we have directly derived the correctness of the big-step operational semantics, and, hence, of every implementation based on it.

Furthermore, some intrinsic computability and expressivity limits of pure co-LP have been formally proved. Concerning computability, there exists no complete procedure for computing $\models_{\mathsf{co}}$, even when only rational terms are considered (that is, $\Upsilon(gfp(T_P^{\mathsf{co}}))$ is productive).

Acknowledgments

The authors wish to thank Andrea Formisano, Alberto Policriti, and Peter Stuckey for the useful discussions on the research pursued in this paper and the anonymous reviewers that allowed us to improve the presentations and removing several typos. The work is partially supported by ¡NδAM GNCS-14 and GNCS-15 projects, and by MIUR DISCO - Distribution, Interaction, Specification, Composition for Object Systems.

References

[1] Aczel, P.: *Non-well-founded sets*, CSLI Lecture Notes, 14, Stanford University, Center for the Study of Language and Information, 1988.

[2] Ancona, D.: Regular corecursion in Prolog, *Computer Languages, Systems & Structures*, **39**(4), 2013, 142–162.

[3] Ancona, D., Corradi, A.: Sound and Complete Subtyping between Coinductive Types for Object-Oriented Languages, *ECOOP 2014 - Object-Oriented Programming - 28th European Conference, Uppsala, Sweden, July 28 - August 1, 2014. Proceedings*, 2014.

[4] Ancona, D., Corradi, A., Lagorio, G., Damiani, F.: Abstract compilation of object-oriented languages into coinductive CLP(X): can type inference meet verification?, *FoVeOOS 2010 Revised Selected Papers* (B. Beckert, C. Marché, Eds.), 6528, Springer, 2010.

[5] Ancona, D., Lagorio, G.: Coinductive type systems for object-oriented languages, *ECOOP 2009 - Object-Oriented Programming* (S. Drossopoulou, Ed.), 5653, Springer, 2009.

[6] Ancona, D., Lagorio, G.: Idealized coinductive type systems for imperative object-oriented programs, *RAIRO - Theoretical Informatics and Applications*, **45**(1), 2011, 3–33.

[7] Ancona, D., Lagorio, G.: Static single information form for abstract compilation, *Theoretical Computer Science (IFIP TCS 2012)* (J. C. Baeten, T. Ball, F. S. de Boer, Eds.), 7604, Springer, 2012.

[8] Apt, K. R.: *Introduction to Logic Programming*, Technical Report TR-87-35, Department of Computer Sciences, The University of Texas at Austin, 1988.

[9] Blair, H. A.: The Recursion-Theoretic Complexity of the Semantics of Predicate Logic as a Programming Language, 54, 1982.

[10] Bonatti, P. A.: Resolution for Skeptical Stable Model Semantics, *J. Autom. Reasoning*, **27**(4), 2001, 391–421.

[11] Bonatti, P. A., Pontelli, E., Son, T. C.: Credulous Resolution for Answer Set Programming, *AAAI* (D. Fox, C. P. Gomes, Eds.), AAAI Press, 2008.

[12] Colmerauer, A.: Equations and Inequations on Finite and Infinite Trees, *FGCS*, 1984.

[13] Courcelle, B.: Fundamental Properties of Infinite Trees, *Theor. Comput. Sci.*, **25**, 1983, 95–169.

[14] Dovier, A.: Set Graphs VI: Logic Programming and Bisimulation, *Proceedings of the 29th Italian Conference on Computational Logic, Torino, Italy, June 16-18, 2014*. (L. Giordano, V. Gliozzi, G. L. Pozzato, Eds.), 1195, CEUR-WS.org, 2014.

[15] Dovier, A., Formisano, A., Pontelli, E.: Multivalued action languages with constraints in CLP(FD), *TPLP*, **10**(2), 2010, 167–235.

[16] Dovier, A., Piazza, C., Policriti, A.: An efficient algorithm for computing bisimulation equivalence, *Theor. Comput. Sci.*, **311**(1-3), 2004, 221–256.

[17] Gelfond, M., Lifschitz, V.: Action Languages, *Electron. Trans. Artif. Intell.*, **2**, 1998, 193–210.

[18] Jaffar, J., Stuckey, P. J.: Semantics of Infinite Tree Logic Programming, *Theoretical Computer Science*, **46**, 1986, 141–158.

[19] Lloyd, J. W.: *Foundations of Logic Programming, 2nd Edition*, Springer, 1987.

[20] Mantadelis, T., Rocha, R., Moura, P.: Tabling, Rational Terms, and Coinduction Finally Together!, *TPLP*, **14**(4-5), 2014, 429–443.

[21] Martelli, A., Montanari, U.: An Efficient Unification Algorithm, *ACM Trans. Program. Lang. Syst.*, **4**(2), 1982, 258–282.

[22] Min, R., Bansal, A., Gupta, G.: Towards Predicate Answer Set Programming via Coinductive Logic Programming, *AIAI* (L. S. I. et al., Ed.), 296, Springer, 2009.

[23] Min, R., Gupta, G.: Coinductive Logic Programming and its Application to Boolean SAT, *FLAIRS Conference*, 2009.

[24] Moura, P.: A Portable and Efficient Implementation of Coinductive Logic Programming, *Practical Aspects of Declarative Languages - 15th International Symposium, PADL 2013, Rome, Italy, January 21-22, 2013. Proceedings* (K. F. Sagonas, Ed.), 7752, Springer, 2013, ISBN 978-3-642-45283-3.

[25] Paige, R., Tarjan, R. E.: Three Partition Refinement Algorithms, *SIAM J. Comput.*, **16**(6), 1987, 973–989.

[26] Rogers, Jr, H.: *Theory of Recursive Functions and Effective Computability*, The MIT Press, 1987.

[27] Saeedloei, N., Gupta, G.: Verifying Complex Continuous Real-Time Systems with Coinductive CLP(R), *Proc. of LATA 2010*, Lecture Notes in Computer Science, Springer, 2010.

[28] Saeedloei, N., Gupta, G.: A logic-based modeling and verification of CPS, *SIGBED Review*, **8**(2), 2011, 31–34.

[29] Simon, L.: *Extending logic programming with coinduction*, Ph.D. Thesis, University of Texas at Dallas, 2006.

[30] Simon, L., Bansal, A., Mallya, A., Gupta, G.: Co-Logic Programming: Extending Logic Programming with Coinduction, *ICALP* (L. Arge, C. Cachin, T. Jurdzinski, A. Tarlecki, Eds.), 4596, 2007.

[31] Simon, L., Mallya, A., Bansal, A., Gupta, G.: Coinductive Logic Programming, *Proc. of International Conference on Logic Programming* (S. Etalle, M. Truszczynski, Eds.), 4079, Springer, 2006.

A Equivalence with the original co-LP operational semantics

We will prove the equivalence of $\vdash_{\mathsf{co}}$ with the transition rule proposed by Simon et al. in the original co-LP formulation [31], summarized below.

A *state* is a pair (T, E) where T is a finite tree with nodes labeled by atoms and E is a set of term equations. Let P be a definite clause program; a [31]-transition between states $(T, E) \xrightarrow{J} (T', E')$ occurs when there is a leaf node N in T labeled by an atom $p(s_1, \ldots, s_n)$ and either of the following points holds:

1. There is a clause $c = p(t_1, \ldots, t_n) \leftarrow B_1, \ldots, B_m \in P$ (renamed with fresh variables) such that $E' = E \cup \{s_1 = t_1, \ldots, s_n = t_n\}$ is solvable. In this case the label J of the transition is c. Two cases are considered:
   (a) $m = 0$ (c a fact): T' is obtained from T by removing the leaf N and recursively its ancestors as long as the nodes found have as unique children the last node removed.
   (b) $m > 0$: then T' is obtained from T by adding m children $N_1, \ldots, N_m$ to N labeled by atoms $B_1, \ldots, B_m$, respectively.
2. There is an ancestor N' of N found in $m > 0$ steps in the path from N to the root of T, labeled by $p(t_1, \ldots, t_n)$ and such that $E' = E \cup \{s_1 = t_1, \ldots, s_n = t_n\}$ is solvable. Then T' is obtained from T by removing N, plus, recursively, all its ancestors as soon as the nodes found have as unique children the last node removed. In this case $J = \nu(m)$.

The notion of [31]-transition is generalized to the notion of [31]-computation (sequence of transitions). A [31]-computation is successful if a state with empty tree T and a solvable equation system E is found. Giving a node N in T, we denote as $\mathsf{anc}(N)$ the set of the labels of the nodes of all the ancestors of N in T (i.e. all the nodes encountered in the path from N to the root, excluding N itself).

Lemma 1 *Let P be a definite clause program, $A_1, \ldots, A_u$ be atoms, E_0 be a set of equations, and $h \geq 0$. Let T_0 be a tree whose leaves are $L_1, \ldots, L_u$, and they are labeled by $A_1, \ldots, A_u$.*
There is a [31]-computation of h steps from a state (T_0, E_0), leading to (T_h, E_h) such that the leaves $L'_1, \ldots, L'_v$ of T_h are labeled by $C_1, \ldots, C_v$ ($v \geq 0$) if and only if *there is a $\vdash_{\mathsf{co}}$-derivation of h steps for $\langle (A_1, \mathsf{anc}(L_1))), \ldots, (A_u, \mathsf{anc}(L_u)) \,\square\, E_0 \rangle$ leading to $\langle (C_1, \mathsf{anc}(L'_1)), \ldots, (C_v, \mathsf{anc}(L'_v)) \,\square\, E_h \rangle$.*

Proof. The proof is carried on by induction on the number of steps $h \geq 0$.
Base. For $h = 0$ the proof follows trivially.
Step. Let $h > 0$.
($\longrightarrow$) Let us assume that there is a [31]-computation

$$(T_0, E_0) \xrightarrow{J_1} (T_1, E_1) \xrightarrow{J_2} \cdots \xrightarrow{J_h} (T_h, E_h)$$

Let us assume that atom $A_i = p(s_1, \ldots, s_n)$ labeling the leaf L_i is selected for the application of the first transition. To simplify the notation, we assume that $i = 1$. Three cases, according to the transition rule, should be considered.

1a. $J_1 = p(t_1, \ldots, t_n) \leftarrow B_1, \ldots, B_m$, $m = 0$ (i.e., J_1 is a renaming of a *fact* in P). This means that $E_1 = E_0 \cup \{s_1 = t_1, \ldots, s_n = t_n\}$ is solvable, and the leaves of T_1 are simply those of T_0 deprived of L_1. From (T_1, E_1), the state (T_h, E_h) is obtained in $h - 1$ steps, hence we can apply the inductive hypothesis starting from

$$\langle (A_2, S_2), \ldots, (A_u, S_u) \,\square\, E_0 \cup \{s_1 = t_1, \ldots, s_n = t_n\} \rangle$$

and the proof follows immediately.

1b. $J_1 = p(t_1, \ldots, t_n) \leftarrow B_1, \ldots, B_m$, $m > 0$: this means that $E_1 = E_0 \cup \{s_1 = t_1, \ldots, s_n = t_n\}$ is solvable. L_1 is replaced as leaf by nodes $N_1, \ldots, N_m$ labeled by $B_1, \ldots, B_m$ in T_1 which has as leaves also nodes $A_2, \ldots, A_u$. Let us observe that for $j = 1, \ldots, m$ it holds that $\mathsf{anc}(N_j) = S_1 \cup \{A_1\}$. From (T_1, E_1), the state (T_h, E_h) is obtained in $h - 1$ steps, hence we can apply the inductive hypothesis starting from

$$\langle (B_1, S_1 \cup \{A_1\}), \ldots, (B_m, S_1 \cup \{A_1\}), (A_2, S_2), \ldots, (A_u, S_u) \,\square\, E_0 \cup \{s_1 = t_1, \ldots, s_n = t_n\} \rangle$$

The proof follows by noticing that the goal above can be obtained by applying rule ($\vdash_{\mathsf{co}} 1$) to the first atom in $\langle (A_1, S_1), \ldots, (A_u, S_u) \,\square\, E_0 \rangle$.

2. This is the co-inductive step; using the notation of [31] the name of the rule is $J_1 = \nu(m)$. This means that there exists $p(t_1, \ldots, t_n)$ among the ancestors of L_1 (hence $p(t_1, \ldots, t_n) \in \mathsf{anc}(L_1)$) such that $E_1 = E_0 \cup \{s_1 = t_1, \ldots, s_n = t_n\}$ is solvable. T_1 is obtained from T_0 by removing L_1 and, possibly some of its ancestors; however, no new leaves are added. Since $p(t_1, \ldots, t_n) \in \mathsf{anc}(L_1)$, also rule $(\vdash_{\mathsf{co}} 2)$ is applicable to $\langle (A_1, S_1), \ldots, (A_u, S_u) \,\square\, E_0 \rangle$. The proof follows by inductive hypothesis in this case, as well.

($\longleftarrow$) Let us consider a tree T_0 with leaves $L_1, \ldots, L_u$ labeled by atoms $A_1, \ldots, A_u$ and let us assume that there is a $\vdash_{\mathsf{co}}$-computation of h steps for $\langle (A_1, \mathsf{anc}(L_1))), \ldots, (A_u, \mathsf{anc}(L_u)) \,\square\, E_0 \rangle$ leading to $F = \langle (C_1, R_1), \ldots, (C_v, R_v) \,\square\, E_h \rangle$ (ancestor sets are computed in T_0). Let us assume that atom $A_i = p(s_1, \ldots, s_n)$ is selected for the application of the first transition, that can be of type $\vdash_{\mathsf{co}} 1$ or $\vdash_{\mathsf{co}} 2$. To simplify the notation, we assume that $i = 1$.

$\vdash_{\mathsf{co}} 1$ This means that there is a renaming of a rule of P of the form $p(t_1, \ldots, t_n) \leftarrow B_1, \ldots, B_m$ and that $E_1 = E_0 \cup \{s_1 = t_1, \ldots, s_n = t_n\}$ is solvable. Therefore there is a $\vdash_{\mathsf{co}}$ derivation of $h - 1$ steps from $\langle (B_1, \mathsf{anc}(L_1) \cup \{A_1\}), \ldots, (B_m, \mathsf{anc}(L_1) \cup \{A_1\}), (A_2, \mathsf{anc}(L_2)), \ldots, (A_u, \mathsf{anc}(L_u)) \,\square\, E_1 \rangle$ to the goal F.

By inductive hypothesis, there is a [31]-computation of $h - 1$ steps from a tree whose leaves are nodes $L_2, \ldots, L_u$ plus the m new nodes $N_1, \ldots, N_m$ (possibly $m = 0$) such that $R_1, \ldots, R_v$ are the labels of the ancestors of the leaves of the tree T_h of the final state. Let T_1 be the tree obtained by adding nodes $N_1, \ldots, N_m$ and edges $(L_1, N_1), \ldots, (L_m, N_m)$ to T_0. Since transition rule [31]-1a/1b leads from $\langle T_0, E_0 \rangle$ to $\langle T_1, E_1 \rangle$, the proof follows.

$\vdash_{\mathsf{co}} 2$ This means that there is an atom $p(t_1, \ldots, t_n) \in \mathsf{anc}(L_1)$ such that $E_1 = E_0 \cup \{s_1 = t_1, \ldots, s_n = t_n\}$ is solvable and there is a $\vdash_{\mathsf{co}}$ derivation of $h - 1$ steps from $\langle (A_2, \mathsf{anc}(L_2)), \ldots, (A_u, \mathsf{anc}(L_u)) \,\square\, E_1 \rangle$ to the goal F.

By inductive hypothesis, there is a [31]-computation of $h - 1$ steps from a tree whose leaves are nodes $L_2, \ldots, L_u$ such that $R_1, \ldots, R_v$ are the labels of the ancestors of the leaves of the tree T_h of the final state. Let T_1 be the tree obtained by removing node L_1 plus all its successors until a non-leaf is found. As a particular case, if all nodes disappear in this way, then L_1 is the unique leaf of T_0, hence $h = 1$ and F is the goal $\langle \varepsilon, E_1 \rangle$. Since transition rule [31]-3 leads from $\langle T_0, E_0 \rangle$ to $\langle T_1, E_1 \rangle$, the proof follows immediately.

□

Theorem 5 *Given a definite clause program P, an atom A, and a set of equations E, there is a successful [31]-computation starting from the state $(\{\varepsilon\}, E)$, where the root ε is labeled by A and E is a set of equations if and only if there is a successful $\vdash_{\mathsf{co}}$ derivation for $\langle (A, \emptyset), E \rangle$.*

Proof. Immediate by Lemma 1. □

B A big-step operational semantics

The operational semantics of co-LP defined in Section 5 is given in the same style of the operational semantics of LP, with rewriting rules describing the single computation steps leading to the final result of a goal resolution. According to the standard classification in formal programming language semantics, this corresponds to what is called a small-step semantics; here we present an equivalent, but more abstract, big-step semantics, where the overall result of a goal resolution is expressed by a set of inference rules from which an interpreter for co-LP can be directly derived [2], with an implementation similar to that supported by the `coinduction` library supported by SWI-Prolog.

$$\text{(empty)}\ \frac{}{S \vdash_P \langle \varepsilon \,\square\, E\rangle \Rightarrow E}$$

$$\text{(ind)}\ \frac{\begin{array}{l} p(t_1,\ldots,t_n) \leftarrow A_1,\ldots,A_m \text{ renaming of a clause in } P \text{ with fresh variables} \\ E_1 \cup \{s_1 = t_1,\ldots,s_n = t_n\} \text{ solvable} \\ S \cup \{p(s_1,\ldots,s_n)\} \vdash_P \langle A_1,\ldots,A_m \,\square\, E_1 \cup \{s_1 = t_1,\ldots,s_n = t_n\}\rangle \Rightarrow E_2 \\ S \vdash_P \langle G_1, G_2 \,\square\, E_2\rangle \Rightarrow E_3 \end{array}}{S \vdash_P \langle G_1, p(s_1,\ldots,s_n), G_2 \,\square\, E_1\rangle \Rightarrow E_3}$$

$$\text{(co-ind)}\ \frac{\begin{array}{l} p(t_1,\ldots,t_n) \in S \qquad E_1 \cup \{s_1 = t_1,\ldots,s_n = t_n\} \text{ solvable} \\ S \vdash_P \langle G_1, G_2 \,\square\, E_1 \cup \{s_1 = t_1,\ldots,s_n = t_n\}\rangle \Rightarrow E_2 \end{array}}{S \vdash_P \langle G_1, p(s_1,\ldots,s_n), G_2 \,\square\, E_1\rangle \Rightarrow E_2}$$

$$\text{(main)}\ \frac{\emptyset \vdash_P \langle G \,\square\, \emptyset\rangle \Rightarrow E}{\vdash_P G \Rightarrow E}$$

The big-step semantics is expressed by two predicates; the main predicate, $\vdash_P G \Rightarrow E$, takes a program P, a goal G, and a set of term equations E; if $\vdash_P G \Rightarrow E$ holds, then E is solvable, and $P \models_{\mathsf{co}} A\gamma$ holds for all atoms A in G, and all substitutions γ solutions of E (hence, E corresponds to a c.a.s. for the goal G in the program P).

Predicate $\vdash_P G \Rightarrow E$ is defined by the unique inference rule (main), which uses the auxiliary predicate $S \vdash_P \langle G \,\square\, E\rangle \Rightarrow E'$ in its premise; such a predicate takes a set of atoms S corresponding to the coinductive hypotheses, a program P, a goal G, and two sets of term equations E and E'; if $S \vdash_P \langle G \,\square\, E\rangle \Rightarrow E'$ holds, then for all substitutions γ solutions of E, $P \cup S\gamma \models_{\mathsf{co}} A\gamma\gamma'$ holds for all atoms A in G, and all substitutions γ' solutions of E'.

More operationally, E corresponds to the substitution computed before resolving G, whereas E' corresponds to the substitution computed after resolving G; S is the set of coinductive hypotheses that can be used when resolving G.

Inference rule (empty) deals with the base case consisting of the empty goal, whereas rules (ind) and (co-ind) are the counterpart of rewrite rules $\vdash_{\mathsf{co}\,1}$ and $\vdash_{\mathsf{co}\,2}$, respectively.

Let us consider the program P of Example 5 and the goal $\leftarrow p(\mathbf{z})$. The following is a successful derivation tree, with $E = \{X_0 = s(X_1), X_0 = X_1\}$:

$$\text{(main)}\ \frac{\text{(ind)}\ \dfrac{\text{(co-ind)}\ \dfrac{\{p(\mathbf{z}), q(X_0)\} \vdash_P \langle \varepsilon \,\square\, \{X_0 = s(X_1), X_0 = X_1\}\rangle \Rightarrow E}{\{p(\mathbf{z}), q(X_0)\} \vdash_P \langle q(X_1) \,\square\, \{X_0 = s(X_1)\}\rangle \Rightarrow E}\ \nabla_1}{\{p(\mathbf{z})\} \vdash_P \langle q(X_0) \,\square\, \emptyset\rangle \Rightarrow E}\ \nabla_2}{\emptyset \vdash_P \langle p(\mathbf{z}) \,\square\, \emptyset\rangle \Rightarrow E}}{\vdash_P p(\mathbf{z}) \Rightarrow E}$$

where $\nabla_1 = \text{(empty)}\ \dfrac{}{\{p(\mathbf{z})\} \vdash_P \langle \varepsilon \,\square\, E\rangle \Rightarrow E}$ and $\nabla_2 = \text{(empty)}\ \dfrac{}{\emptyset \vdash_P \langle \varepsilon \,\square\, E\rangle \Rightarrow E}$

While the operational semantics $\vdash_{\mathsf{co}}$ defined in Section 5, associates a distinct copy of the coinductive hypotheses with each atom in the goal, here they are shared by each atom. Equivalence of the two semantics is stated by the following theorems. We first prove that the big-step operational semantics is sound w.r.t. the rewriting semantics $\vdash_{\mathsf{co}}$: if, according to the big-step semantics, a goal G succeeds with a set of equations E, then there exists a successful derivation starting from G and ending in $\langle \varepsilon \,\square\, E\rangle$, Then we prove the converse property, that is, the rewriting semantics $\vdash_{\mathsf{co}}$ is sound w.r.t. the big-step one: if there exists a successful derivation starting from G and ending in $\langle \varepsilon \,\square\, E\rangle$, then G succeeds with a set of equations E according to the big-step semantics.

The following theorem is instrumental to proving soundness of the big-step semantics w.r.t. the rewriting semantics $\vdash_{\mathsf{co}}$; it shows that whenever a goal G succeeds according to the big-step semantics, then either G is already empty, or there exists a $\vdash_{\mathsf{co}}$ derivation step for G. The proof proceeds by case analysis on the first applied inference rule.

Theorem 6 *If $S \vdash_P \langle A_1,\ldots,A_n \,\square\, E_1\rangle \Rightarrow E_2$, then either $n = 0$ and $E_1 = E_2$, or either of the facts below holds:*

- *there exist $k \in \{1,\ldots,n\}$ and atoms $B_1,\ldots,B_m$, and E_1' s.t. $\langle (A_1,S),\ldots,(A_n,S) \,\square\, E_1\rangle \vdash_{\mathsf{co}\,1}$ $\langle (A_1,S),\ldots,(A_{k-1},S),(B_1,S'),\ldots,(B_m,S'),(A_{k+1},S),\ldots,(A_n,S) \,\square\, E_1'\rangle$;*

- *there exist* $k \in \{1,\ldots,n\}$ *and* E_1' *s.t.*
  $\langle (A_1,S),\ldots,(A_n,S)\,\square\, E_1\rangle \vdash_{\mathsf{co}\,2} (A_1,S),\ldots,(A_{k-1},S),(A_{k+1},S),\ldots,(A_n,S)\,\square\, E_1'\rangle$.

Proof. By case analysis on the inference rule applied for deriving $S \vdash_P \langle A_1,\ldots,A_n\,\square\, E_1\rangle \Rightarrow E_2$.

If the first applied inference rule is (empty), then $n = 0$ and $E_1 = E_2$.

If the first applied inference rule is (ind), then the premises of such a rule must hold, therefore there exists $k \in \{1,\ldots,n\}$ s.t. $A_k = p(s_1,\ldots,s_h)$ and there exists a renaming $p(t_1,\ldots,t_h) \leftarrow B_1,\ldots,B_m$ of a clause in P s.t. $E_1 \cup \{s_1 = t_1,\ldots,s_h = t_h\}$ is solvable, hence rewriting rule $\vdash_{\mathsf{co}\,1}$ is applicable and we get the step
$\langle (A_1,S),\ldots,(A_n,S)\,\square\, E_1\rangle \vdash_{\mathsf{co}\,1}$
$\langle (A_1,S),\ldots,(A_{k-1},S),(B_1,S'),\ldots,(B_m,S'),(A_{k+1},S),\ldots,(A_n,S)\,\square\, E_1'\rangle$,
where $S' = S \cup \{p(s_1,\ldots,s_h)\}$ and $E_1' = E_1 \cup \{s_1 = t_1,\ldots,s_h = t_h\}$.

If the first applied inference rule is (co-ind), then the premises of such a rule must hold, therefore there exists $k \in \{1,\ldots,n\}$ s.t. $A_k = p(s_1,\ldots,s_h)$ and there exists an atom $p(t_1,\ldots,t_h) \in S$ s.t. $E_1 \cup \{s_1 = t_1,\ldots,s_h = t_h\}$ is solvable, hence the rewriting rule $\vdash_{\mathsf{co}\,2}$ is applicable and we get the step
$\langle (A_1,S),\ldots,(A_n,S)\,\square\, E_1\rangle \vdash_{\mathsf{co}\,2} \langle (A_1,S),\ldots,(A_{k-1},S),(A_{k+1},S),\ldots,(A_n,S)\,\square\, E_1'\rangle$, where $E_1' = E_1 \cup \{s_1 = t_1,\ldots,s_h = t_h\}$. □

The next lemma is used in the proof of soundness of the big-step semantics w.r.t. the rewriting semantics $\vdash_{\mathsf{co}}$. It states that the derivation of a sub-goal is independent from its surrounding context, and can be easily proved by induction on the length of the derivation.

Lemma 2 *There exists a successful* $\vdash_{\mathsf{co}}$ *derivation for* $\langle (A_1,S),\ldots,(A_n,S)\,\square\, E_1\rangle$ *ending in* $\langle \varepsilon\,\square\, E_2\rangle$*, if and only if there exists a* $\vdash_{\mathsf{co}}$ *derivation from*
$\langle (B_1,S_1),\ldots,(B_k,S_k),(A_1,S),\ldots,(A_n,S),(B_{k+1},S_{k+1}),\ldots,(B_{k+m},S_{k+m})\,\square\, E_1\rangle$ *to*
$\langle (B_1,S_1),\ldots,(B_{k+m},S_{k+m})\,\square\, E_2\rangle$.

Proof. Directly by induction on the length of the derivation. □

What follows is the main theorem proving soundness of the big-step semantics w.r.t. the rewriting semantics $\vdash_{\mathsf{co}}$. It uses Theorem 6 and Lemma 2, and is proved by induction on the inference rules of the big-step semantics, and by case analysis on them.

Theorem 7 *If* $S \vdash_P \langle A_1,\ldots,A_n\,\square\, E_1\rangle \Rightarrow E_2$*, then there exists a successful* $\vdash_{\mathsf{co}}$ *derivation for* $\langle (A_1,S),\ldots,(A_n,S)\,\square\, E_1\rangle$ *ending in* $\langle \varepsilon\,\square\, E_2\rangle$.

Proof. by induction on the inference rules of the big-step semantics, and by case analysis on them.

If the first applied inference rule is (empty), then by Theorem 6 $n = 0$ and $E_1 = E_2$, and $\langle \varepsilon\,\square\, E_2\rangle$ is a successful $\vdash_{\mathsf{co}}$ derivation for $\langle (A_1,S),\ldots,(A_n,S)\,\square\, E_1\rangle$ ending in $\langle \varepsilon\,\square\, E_2\rangle$.

If the first applied inference rule is (ind), then by Theorem 6 there exist $k \in \{1,\ldots,n\}$ and E_1' s.t. $\langle (A_1,S),\ldots,(A_n,S)\,\square\, E_1\rangle \vdash_{\mathsf{co}\,1}$
$\langle (A_1,S),\ldots,(A_{k-1},S),(B_1,S'),\ldots,(B_m,S'),(A_{k+1},S),\ldots,(A_n,S)\,\square\, E_1'\rangle$,
where $S' = S \cup \{p(s_1,\ldots,s_h)\}$ and $E_1' = E_1 \cup \{s_1 = t_1,\ldots,s_h = t_h\}$; furthermore, by hypothesis and by rule (ind), $S' \vdash_P \langle B_1,\ldots,B_m\,\square\, E_1'\rangle \Rightarrow E_1''$, and $S \vdash_P \langle A_1,\ldots,A_{k-1},A_{k+1},\ldots,A_n\,\square\, E_1''\rangle \Rightarrow E_2$ hold, hence, by inductive hypothesis, there exist successful $\vdash_{\mathsf{co}}$ derivations for $\langle (B_1,S'),\ldots,(B_m,S')\,\square\, E_1'\rangle$, and $\langle (A_1,S)\ldots,(A_{k-1},S),(A_{k+1},S)\,\square\, E_1'',\ldots,(A_n,S_n)\rangle$, ending in $\langle \varepsilon\,\square\, E_1''\rangle$ and $\langle \varepsilon\,\square\, E_2\rangle$, respectively. Finally, we can conclude by Lemma 2, and transitivity of $\vdash_{\mathsf{co}}$ derivation.

If the first applied inference rule is (co-ind), then by Theorem 6 there exist $k \in \{1,\ldots,n\}$ and E_1' s.t. $\langle (A_1,S),\ldots,(A_n,S)\,\square\, E_1\rangle \vdash_{\mathsf{co}\,2} \langle (A_1,S),\ldots,(A_{k-1},S),(A_{k+1},S),\ldots,(A_n,S)\,\square\, E_1'\rangle$; furthermore, $S \vdash_P A_1,\ldots,A_{k-1},A_{k+1},\ldots,A_n$ E_2 holds by hypothesis and by rule (co-ind), hence, by inductive hypothesis, there exists a successful $\vdash_{\mathsf{co}}$ derivation for
$\langle (A_1,S)\ldots,(A_{k-1},S),(A_{k+1},S),\ldots,(A_n,S_n)\,\square\, E_1'\rangle$, ending in $\langle \varepsilon\,\square\, E_2\rangle$. Finally, we can conclude by transitivity of $\vdash_{\mathsf{co}}$ derivation. □

Since the big-step semantics is formulated in terms of the main predicate $\vdash_P G \Rightarrow E$, we need a final claim for proving soundness of the big-step semantics w.r.t. the rewriting semantics $\vdash_{\mathsf{co}}$. However, since the definition of $\vdash_P G \Rightarrow E$ directly depends on the auxiliary predicate $S \vdash_P \langle G\,\square\, E\rangle \Rightarrow E'$, the proof is a simple consequence of Theorem 7.

Corollary 1 *If* $\vdash_P A_1,\ldots,A_n \Rightarrow E$*, then there exists a successful* $\vdash_{\mathsf{co}}$ *derivation for*
$\langle (A_1,\emptyset),\ldots,(A_n,\emptyset)\,\square\,\emptyset\rangle$ *ending in* $\langle \varepsilon\,\square\, E\rangle$.

Proof. Any derivation tree for $\vdash_P A_1, \ldots, A_n \Rightarrow E$ can only be obtained by firstly applying inference rule (main), therefore $\emptyset \vdash_P \langle A_1, \ldots, A_n \,\square\, \emptyset\rangle \Rightarrow E$ holds and we can conclude by Theorem 7. □

The following lemma generalizes the property of the independence of the computation rule in LP operational semantics [19] to the case of co-LP. It is instrumental to the proof of soundness of the rewriting semantics $\vdash_{\mathsf{co}}$ w.r.t. the big-step one, and is proved by induction on the number of rewriting steps.

Lemma 3 *If there exists a successful $\vdash_{\mathsf{co}}$ derivation of length l for $\langle (A_1, S_1), \ldots, (A_n, S_n) \,\square\, E_1\rangle$ ending in $\langle \varepsilon \,\square\, E_2\rangle$, then for all $i \in \{1, \ldots, n\}$ there exist E_1', S, and $B_1, \ldots, B_m$ s.t.*
$\langle (A_1, S_1), \ldots, (A_n, S_n) \,\square\, E_1\rangle \vdash_{\mathsf{co}}$
$\langle (A_1, S_1), \ldots, (A_{i-1}, S_{i-1}), (B_1, S), \ldots, (B_m, S), (A_{i+1}, S_{i+1}), (A_n, S_n) \,\square\, E_1'\rangle$
and $\langle (A_1, S_1), \ldots, (A_{i-1}, S_{i-1}), (B_1, S), \ldots, (B_m, S), (A_{i+1}, S_{i+1}), (A_n, S_n) \,\square\, E_1'\rangle$ has a successful derivation of length $l - 1$ ending in $\langle \varepsilon \,\square\, E_2\rangle$.

Proof. By induction on the number of rewriting steps before selecting atom A_i. The proof relies on the following easily provable facts:

- after a derivation step all atoms that have not been selected and their associated coinductive hypotheses remain unchanged;
- if $E_1 \cup E_2$ is solvable, then E_2 and $E_2 \cup E_1$ are solvable as well.

□

The following theorem states that the rewriting semantics $\vdash_{\mathsf{co}}$ is sound w.r.t. the big-step one; the proof is by induction on the length of the $\vdash_{\mathsf{co}}$ derivation.

Theorem 8 *If there exists a successful $\vdash_{\mathsf{co}}$ derivation for $\langle (A_1, S), \ldots, (A_n, S) \,\square\, E_1\rangle$ ending in $\langle \varepsilon \,\square\, E_2\rangle$, then $S \vdash_P \langle A_1, \ldots, A_n \,\square\, E_1\rangle \Rightarrow E_2$ holds.*

Proof. By induction on the length of the $\vdash_{\mathsf{co}}$ derivation.

If the derivation has length 0, then $n = 0$, and $E_1 = E_2$, therefore we can get a derivation tree by applying inference rule (empty).

If the derivation has length greater than 0, then the first step can be obtained by applying either rewriting rule $\vdash_{\mathsf{co}\,1}$ or $\vdash_{\mathsf{co}\,2}$.

If rule $\vdash_{\mathsf{co}\,1}$ is applied, then we have
$\langle (A_1, S), \ldots, (A_n, S) \,\square\, E_1\rangle \vdash_{\mathsf{co}\,1}$
$\langle (A_1, S), \ldots, (A_{i-1}, S), (B_1, S'), \ldots, (B_m, S'), (A_{i+1}, S), \ldots, (A_n, S) \,\square\, E_1'\rangle$
with $E_1' = E_1 \cup \{s_1 = t_1, \ldots, s_h = t_h\}$ solvable, $A_i = p(s_1, \ldots, s_h)$, and $p(t_1, \ldots, t_h) \leftarrow B_1, \ldots, B_m$ be a renaming of a clause in P with fresh variables. Then by Lemmas 3 and 2, there exist successful derivations of length strictly less than n from $\langle (B_1, S'), \ldots, (B_m, S') \,\square\, E_1'\rangle$ to $\langle \varepsilon \,\square\, E_1''\rangle$, and from $\langle (A_1, S), \ldots, (A_{i-1}, S), (A_{i+1}, S), \ldots, (A_n, S) \,\square\, E_1''\rangle$ to $\langle \varepsilon \,\square\, E_2\rangle$.

By inductive hypothesis $S \vdash_P \langle B_1, \ldots, B_m \,\square\, E_1'\rangle \Rightarrow E_1''$ and $S \vdash_P \langle A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n \;\square\, E_1''\rangle \Rightarrow E_2$ hold, therefore we can conclude by applying inference rule (ind).

If rule $\vdash_{\mathsf{co}\,2}$ is applied, then we have
$\langle (A_1, S), \ldots, (A_n, S) \,\square\, E_1\rangle \vdash_{\mathsf{co}\,2} \langle (A_1, S), \ldots, (A_{i-1}, S), (A_{i+1}, S), \ldots, (A_n, S) \,\square\, E_1'\rangle$
with $E_1' = E_1 \cup \{s_1 = t_1, \ldots, s_h = t_h\}$ solvable, $A_i = p(s_1, \ldots, s_h)$, and $p(t_1, \ldots, t_n) \in S_i$. By inductive hypothesis $S \vdash_P \langle A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n \,\square\, E_1'\rangle \Rightarrow E_2$ holds, therefore we can conclude by applying inference rule (co-ind). □

Finally, the next corollary of Theorem 8 proves the soundness of the rewriting semantics $\vdash_{\mathsf{co}}$ w.r.t. the big-step one expressed in terms of the main predicate $\vdash_P G \Rightarrow E$.

Corollary 2 *If there exists a successful $\vdash_{\mathsf{co}}$ derivation for $\langle (A_1, \emptyset), \ldots, (A_n, \emptyset) \,\square\, \emptyset\rangle$ ending in $\langle \varepsilon \,\square\, E\rangle$, then $\vdash_P A_1, \ldots, A_n \Rightarrow E$ holds.*

Proof. By Theorem 8 $\emptyset \vdash_P \langle A_1, \ldots, A_n \,\square\, \emptyset\rangle \Rightarrow E$, hence we can conclude by inference rule (main). □