co-LP: Back to the Roots

Davide Ancona
University of Genova, DIBRIS, ITALY

Agostino Dovier
University of Udine, DIMI, ITALY

submitted 10 April 2013; accepted 23 May 2013

Abstract

Recently, several papers dealing with co-inductive logic programming have been proposed, dealing with pure Prolog and constraint logic programming, with and without negation. In this paper we revisit and use, as much as possible, some fundamental results developed in the Eighties to analyze the foundations, and to clarify the possibilities but also the intrinsic theoretical limits of this programming paradigm.

1 Introduction

When we attend our first courses in Mathematics and in Programming we learn a powerful, yet intuitive, reasoning principle: induction. We use it for several mathematical proofs, including correctness proofs of our programs. Induction is at the basis of recursive programming. We fix the input/output behavior for some initial cases, while larger cases are defined upon the output of smaller ones. It is not surprising, therefore, that the semantics of programming languages, and in particular of Prolog, is based on induction or, put it in other words, on the least fixpoint of an operator.

On the other hand, there are mathematical and programming contexts where circular phenomena arise naturally. For instance, in set theory the problem of establishing whether two ill-founded sets are equivalent (Aczel 1988). The extensionality principle, that governs set equality in standard set theory, is not adequate to prove the equality of the sets that are solutions of the following equations: $X = \{X\}$, $Y = \{Y, \{Y\}\}$, since it would lead to circularity. Aczel's AFA axiom and the subsequent coinductive notion of bisimulation allow a correct solution to this problem. Bisimulation and other similar coinductive definitions are at the basis of the analysis of circular properties. Computing the maximum bisimulation between a set S and S itself, that leads to the minimum representation of S, is implemented as a greatest fixpoint algorithm (Paige and Tarjan 1987; Dovier et al. 2004).

In computer science maybe the most preeminent example of circularity are graphs, that are employed in many application domains; but there are also other formalisms that permeate computer science where the notion of circularity is dominant, like automata (of several kinds), grammars, process algebras, and temporal logics. Bisimulation is at the basis of the equivalence relations in these formalisms.

Prolog has been chosen as a promising language to embed coinductive reasoning (Simon et al. 2006; Simon et al. 2007; Min et al. 2009) for at least three main reasons: its incredibly clean semantics, its natural support for rational terms, and its declarative nature. In imperative languages

circular structures are implemented in the same way as inductive structures through an explicit use of references, whereas in purely declarative languages, without a native support for circularity and coinduction structures like graphs cannot be implemented in a high level way (that is, transparent to references), as happens for inductively defined structures.

The basic idea of Coinductive Logic Programming (briefly, co-LP) is to keep the same syntax, while the least fixpoint semantics is replaced by the greatest fixpoint semantics. Moreover, since coinductive definitions are naturally used for dealing with infinite terms, the semantics is computed over the universe of infinite trees rather than on the usual Herbrand universe.

In this paper we revisit and deepen some aspects of the foundations of co-LP (Simon 2006) to try to clarify what are the main properties of this kind of semantics in logic programming.

A first contribution of our study is a formulation of a coinductive resolution rule which is simpler than the corresponding one originally given by Gupta et al (Simon et al. 2006); the proof of correctness we provide is simple, since it is mainly based on results from Jaffar and Stuckey (Jaffar and Stuckey 1986). We also formally prove that no complete procedure can exist, in general, for establishing whether a given atom belongs to the coinductive semantics of a definite clause program, even when only rational terms are considered.

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 we propose a resolution rule able to approximate the semantics. Finally, conclusions are drawn in Section 6.

2 Preliminaries

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

A *tree* T is a non-empty subset of $\mathbb{N}^*$ such that if $u \cdot n \in T$ 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 string[1] ε. 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 $\rho : T \longrightarrow \mathscr{F} \cup \mathscr{V}$ such that for each $u \in T$, $|\{ux \in T : x \in \mathbb{N}\}| = \mathsf{ar}(\rho(u))$. 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 2.1 Let $\mathscr{F} = \{0/0, s/1, f/2\}$, 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 $\rho_1 = \{(\varepsilon, f), (0, f), (1, s), (0\cdot 0, 0), (0\cdot 1, X), (1\cdot 0, s), (1\cdot 0\cdot 0, Y)\}$ identify a term, say t_1 ($t_1 = f(f(0, X), s(s(Y)))$ using the standard notation of finite terms).
$T_2 = \{\varepsilon, 0, 0\cdot 0, 0\cdot 0\cdot 0, 0\cdot 0\cdot 0\cdot 0, \ldots\}$ 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, $\Omega = s(s(s(\cdots))))$. ☑

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

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 $i \in \{1, \ldots, n\}$, $X_i \neq r_i$ and if r_i is a variable, then X_i does not occur in $r_1, \ldots, r_n$.

Given two trees T_1 and T_2 and a string $u \in T_2$, 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 said *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 equations either is unsatisfiable or it has (at least) one rational tree solution which can be expressed by a solved form system. 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 (w.l.o.g., let $t_1 = t$), 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 (Martelli and Montanari 1982; Courcelle 1983).

An atomic formula, or *atom*, is a pair $\langle T, \rho \rangle$ where T is a tree and ρ is a function $\rho : T \longrightarrow \Pi \cup \mathscr{V} \cup \mathscr{F}$, and $\rho(\varepsilon) = p \in \Pi$, $\mathsf{ar}(p) \notin T$, and $(\forall i \in \{0, \ldots, \mathsf{ar}(p) - 1\})(i \in T \wedge \langle \{v \in \mathbb{N}^* \mid i \cdot v \in T\}, \rho_i \rangle$ is a term), where $\rho_i(u) \stackrel{\mathsf{def}}{=} \rho(i \cdot u)$. 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 $\mathscr{F}_P$ be the set of functional symbols in P; if there exists no $f/0$ in $\mathscr{F}_P$, then $0/0$ is added to $\mathscr{F}_P$. The *Herbrand Universe* U_P is the set of finite (ground) terms one can build with $\mathscr{F}_P$. Such a set is finite (and isomorphic to $\mathscr{F}_P$) if and only if for all $f \in \mathscr{F}_P$ $\mathsf{ar}(f) = 0$.

The *complete Herbrand Universe* $\mathsf{co}\text{-}U_P$ (Lloyd 1987, §25) is the set of finite and infinite terms one can build with $\mathscr{F}_P$. The set $\mathsf{co}\text{-}U_P$ contains finite terms, infinite rational terms, and non rational terms. $\mathsf{co}\text{-}U_P$ is finite (and equal to U_P) if and only if for all $f \in \mathscr{F}_P$ $\mathsf{ar}(f) = 0$.

Example 2.2 Assume $\mathscr{F}_P = \{0/0, s/1\}$. As a shorthand, we will use $\underline{0}$ for 0 and $\underline{n+1}$ for $s(\underline{n})$. Therefore: $U_P = \{0, s(0), s(s(0)), s(s(s(0))), \ldots\} = \{\underline{0}, \underline{1}, \underline{2}, \underline{3}, \ldots\}$. This signature allows just one infinite term: the term $\Omega = s(s(s(\cdots)))$ (c.f. Example 2.1). This term is equal to any of its subterms, in particular $\Omega = s(\Omega)$, therefore $\mathsf{co}\text{-}U_P = \{\Omega, \underline{0}, \underline{1}, \underline{2}, \underline{3}, \ldots\}$.

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

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

$\mathsf{co}\text{-}U_P$ contains either rational or non rational infinite terms. As a shorthand, let us use 0 for λ and 1 for $[\lambda]$ (i.e. $[\lambda\,|\,\lambda]$). 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 $\mathsf{co}\text{-}U_P$. 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 $\mathsf{co}\text{-}U_P$, and it has an infinite number of different subterms, otherwise $\sqrt{2}$ would be a rational number in $\mathbb{R}$. ☑

While U_P is finite or denumerable, if $\mathscr{F}$ contains at least two unary symbols or at least one constant symbol and one symbol of arity greater than 1 (briefly, $|\mathscr{F}| \geqslant 2, \sum_{f \in \mathscr{F}} \mathsf{ar}(f) \geqslant 2$), then co-$U_P$ is not denumerable (generalize the reasoning made in Example 2.2 for defining sqrt2).

A *tree substitution* (ground substitution in (Jaffar and Stuckey 1986)) θ is a mapping from a finite subset of variables, $\mathsf{dom}(\theta)$ to co-U. 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:

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

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 $I \subseteq$ co-B_P are called co-*interpretations*. Finally, we also define:

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

If $S \subseteq \mathsf{co\text{-}ground}(P)$, with $\Upsilon(S) \subseteq S$ we denote its subset built only on rational terms.

SLD derivation and its variants. In the classical definition of SLD derivation, given a definite clause program P and a goal $G = A_1, \ldots, A_n$,[2] 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 system 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 $G' = \langle E' \,\square\, A_1, \ldots, A_{i-1}, B_1, \ldots, B_m, A_{i+1}, \ldots, A_n \rangle$.

Rule $\vdash_\infty$ is a special case of the rule proposed in (Jaffar and Stuckey 1986) to deal with Prolog II programs (Colmerauer 1984), where in E either equations or inequations are admitted. As such, it inherits all the results from (Jaffar and Stuckey 1986), when inequations are not explicitly used in the Program. Comparing $\vdash$ with $\vdash_\infty$, since θ has exactly the same solutions of $\{s_1 = t_1, \ldots, s_n = t_n\}$, the two definitions are equivalent for definite programs (assuming to use or not use 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 application of $\vdash_\infty$; a *ground derivation* is a (finite or infinite) sequence of application 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*

[2] For the sake of simplicity, we simply consider the list of atoms of a goal, not its logical meaning.

if it is finite and its last goal is successful. A derivation is *failing* if it is finite, its last goal is not successful, and rule $\vdash_\infty$ is not applicable to it. A selection rule R chooses the literal A_i in a goal $G = \langle E \Box A_1, \ldots, A_n \rangle$ given the whole derivation that led to G. In Prolog, R simply selects A_1. 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 failing, then it is a *finite failure tree*.

Example 2.3 (SLD derivation with infinite terms) Let P be the following definite clause program: $\mathtt{num}(0).$ $\quad \mathtt{num}(s(X)) \leftarrow \mathtt{num}(X).$ Using standard SLD rule we would have the successful derivation $\mathtt{num}(s(s(0))) \vdash \mathtt{num}(s(0)) \vdash \mathtt{num}(0) \vdash \varepsilon$. Using $\vdash_\infty$ we would have the following equivalent successful derivation: $\langle \emptyset \Box \mathtt{num}(s(s(0))) \rangle \vdash_\infty \langle \{X = s(0)\} \Box \mathtt{num}(X) \rangle \vdash_\infty$

$$\langle \{X = s(0), X = s(X_1)\} \Box \mathtt{num}(X_1) \rangle \vdash_\infty \langle \{X = s(0), X = s(X_1), X_1 = 0\} \Box \varepsilon \rangle$$

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

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

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 $\{X = s(X), X_1 = X, X_2 = X, X_3 = X, \ldots\}$. ☑

Some computability notions used in the paper. We assume basic computability notions (see e.g., (Rogers 1987)). 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 $\overline{K} = \mathbb{N} \setminus K$, 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 $\overline{K}$ is productive (hence, non r.e. in a very strong way). $A \leqslant B$ (and $\overline{A} \leqslant \overline{B}$) 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 $A \leqslant B$ (due to transitivity of $\leqslant$ and completeness of K, it is sufficient to prove that $K \leqslant B$). A is complete iff A is r.e. and $\overline{A}$ is productive (Myhill). If $\overline{K} \leqslant A$ (or equivalently $K \leqslant \overline{A}$) then A is productive, as well. The class Π^1_1 is one of the lower classes in the analytical hierarchy of sets (Rogers 1987, §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 $\overline{S} = \{x \in \mathbb{N} : (\exists f)(\forall y)(\neg R(f,(y,x)))\}$. 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 $\overline{S}$ 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 $((\forall X, Y \in \wp(\mathscr{B}_P))(X \subseteq Y \rightarrow T_P(X) \subseteq T_P(Y)))$ and upward continuous (for each infinite sequence $I_0 \subseteq I_1 \subseteq I_2 \subseteq \cdots$: $T_P\left(\bigcup_{i \geqslant 0} I_i\right) = \bigcup_{i \geqslant 0} T_P(I_i)$) but not downward continuous (for

each infinite sequence $I_0 \supseteq I_1 \supseteq I_2 \supseteq \cdots$: $T_P\left(\bigcap_{i\geqslant 0} I_i\right) = \bigcap_{i\geqslant 0} T_P(I_i)$). Let us recall the definitions of iterated T_P (we assume the notion of (successor/limit) ordinal (Lloyd 1987)):

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

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*" (Apt 1988). The asymmetry is clearly pointed out by the following classical example (from (Lloyd 1987, page 37)):

Example 3.1 Let P be the following program: $q(s(X)) \leftarrow q(X).$ $\quad p(0) \leftarrow q(X).$
We have that $T_P \downarrow \omega = \{p(0)\}$ but it is not a fix-point. In fact, $T_P(\{p(0)\}) = \emptyset = T_P(\emptyset)$. For computing $gfp(T_P) = \emptyset$ a transfinite computation $T_P \downarrow \omega + 1$ is needed.

The example can be generalized to show that $\omega + 1$ can be not enough (c.f. Theorem 4.1).

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 (Lloyd 1987, §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 $\langle \text{co-}U_P, \delta \rangle$.

Example 3.2 Let us consider again the program in Example 3.1. Since $\Omega = s(\Omega)$, assuming $q(\Omega)$, then $q(s(\Omega)) = q(\Omega)$ is justified by the first clause. In this case $T_P^{\mathsf{co}} \downarrow \omega = \{q(\Omega), p(0)\}$ is also a fixpoint. ☑

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

Example 3.3 Let P be the following program. $p(0) \leftarrow q(X).$ $\quad q(s(X)) \leftarrow q(X), X \neq s(X).$
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 3.1 In the standard minimum model semantics, once $\mathscr{F}$ is known, the $\neq$ predicate above can be defined by a definite clause program without using negation. E.g., if $\mathscr{F} = \{0/0, s/1\}$: $0 \neq s(X).$ $\quad s(X) \neq 0.$ $\quad s(X) \neq s(Y) \leftarrow X \neq Y.$ 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 Venn Diagrams

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 1 shows this partition (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 (Apt 1988), for the infinite tree see (Jaffar and Stuckey 1986, §6).

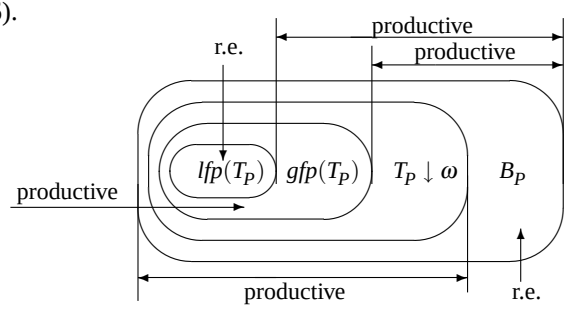

Fig. 1. Venn Diagram

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

- The set $T_P \uparrow \omega$ is the set of ground atoms that admit a successful SLD derivation. It is the standard semantics of a logic program and coincides with the minimum Herbrand model.
- The set $B_P \setminus T_P \downarrow \omega$, also called the finite failure set, is the set of ground atoms that have only failing derivations, and the number of these derivations is finite.
- $IS = gfp(T_P) \setminus T_P \uparrow \omega$ is the set of ground atoms that do not admit a successful ground derivation but they admit (at least) one infinite ground derivation.
- $IF = T_P \downarrow \omega \setminus gfp(T_P)$ is the set of ground atoms for which there are no successful derivations, there are no infinite ground derivations, and there exist an infinite number of failing (hence, finite) ground derivations.

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 4.1 Let us consider the definite clause program P:

$\mathtt{num}(0).$ $\quad \mathtt{num}(s(X)) \leftarrow \mathtt{num}(X).$

$\mathtt{p}(0) \leftarrow \mathtt{num}(X), \mathtt{q}(X).$ $\quad \mathtt{p}(s(X)) \leftarrow \mathtt{num}(X), \mathtt{p}(s(X)).$ $\quad \mathtt{q}(s(X)) \leftarrow q(X).$

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 $n \in \mathbb{N}$, $n > 0$ $\quad \bullet \mathtt{p}(0) \in T_P \downarrow \omega \setminus gfp(T_P)$. ☑

Theorem 4.1 (*(Blair 1982)*) 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.).

With "in general" we mean that there can be cases when the four 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 (Blair 1982) 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 (Jaffar and Stuckey 1986) in the universe of finite and infinite terms for their derivation rule, which coincides with $\vdash_\infty$ if $\neq$ constraints between terms are not used. We recall that, in the absence of $\neq$ constraints, $T_P^{\mathsf{co}} \downarrow \omega = gfp(T_P^{\mathsf{co}})$ (hence, the set IS is empty).

Example 4.2 Let us consider the program P of Example 4.1 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 $gfp(T_P^{\mathsf{co}}) \setminus lfp(T_P^{\mathsf{co}}) = \{\mathtt{num}(\Omega), \mathtt{p}(0), \mathtt{p}(\Omega), \mathtt{q}(\Omega)\}$. ☑

Example 4.3 An interesting and simple example is the following (the constant 0 is added to $\mathscr{F}$):

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

In this case $T_P^{\mathsf{co}} \uparrow \omega = \emptyset = B_P \setminus T_P^{\mathsf{co}} \downarrow \omega = T_P^{\mathsf{co}} \downarrow \omega \setminus gfp(T_P^{\mathsf{co}})$. The only non-empty Venn region is $gfp(T_P^{\mathsf{co}}) \setminus lfp(T_P^{\mathsf{co}})$ which coincides with $gfp(T_P^{\mathsf{co}}) = \mathsf{co}\text{-}B_P = \{p(\Omega), p(\underline{0}), p(\underline{1}), p(\underline{2}), p(\underline{3}), \ldots\}$. Note that in this case $\Upsilon(\mathsf{co}\text{-}B_P) = \mathsf{co}\text{-}B_P$, hence $\Upsilon(gfp(T_P^{\mathsf{co}})) = gfp(T_P^{\mathsf{co}}) = gfp(T_P^{\mathsf{rat}})$ (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 4.4 Let us consider the program P

$$\mathtt{inf\_list}(X, [X|L]) \leftarrow \mathtt{inf\_list}(s(X), L). \qquad p(0) \leftarrow \mathtt{inf\_list}(0, L).$$

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

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

The infinite, non rational atom $\mathtt{inf\_list}(\underline{0}, [\underline{0}, \underline{1}, \underline{2}, \ldots])$, as well as the atom $p(0)$ belong to $gfp(T_P) \setminus lfp(T_P)$. Similarly, for any term t of the signature, we have that $\mathtt{inf\_list}(t, [t, s(t), s(s(t)), s(s(s(t))), \ldots]) \in gfp(T_P) \setminus lfp(T_P)$. Being $\Omega = s(\Omega)$, in particular, $\mathtt{inf\_list}(\Omega, [\Omega, \Omega, \ldots]) \in gfp(T_P) \setminus lfp(T_P)$, hence $p(0)$ and this one are the unique rational atoms in that set. All other rational atoms belong to $\mathsf{co}\text{-}B_P \setminus gfp(T^{\mathsf{co}} \downarrow \omega)$. However, whereas for $\mathtt{inf\_list}(\Omega, [\Omega, \Omega, \Omega, \ldots])$ there exists an infinite but "rational" ground derivation, for $p(0)$ there are no "rational" ground derivations; furthermore $\mathtt{inf\_list}(\Omega, [\Omega, \Omega, \ldots]) \in gfp(T_P^{\mathsf{rat}})$, but $p(0) \notin gfp(T_P^{\mathsf{rat}})$. ☑

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

Theorem 4.2 Given a definite clause program P, in general:

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

Proof The "positive" part of the theorem is a consequence of (Jaffar and Stuckey 1986): checking whether a rational 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 verify that the number of its derivations is finite and all of them are failing. 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:

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

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

$\leftarrow h(\underline{x},\underline{y},\underline{z})$ has a finite successful derivation if $\varphi(x,y) = z$, $z \in \{0,s(0)\}$
$\leftarrow h(\underline{x},\underline{y},\underline{z})$ has a finite failure tree if ($z \in \{0,s(0)\}$ and $\varphi(x,y) \neq z$) or $z \notin \{0,s(0)\}$

Let P_φ be the program defining h and including the following predicate definition:

$$\mathtt{num}(0). \qquad\qquad \mathtt{num}(s(X)) \leftarrow \mathtt{num}(X).$$

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

$$r(X) \leftarrow \mathtt{num}(N), h(X,N,s(0)).$$

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

If $x \in K$, then there is a number t such that $\leftarrow h(\underline{x},\underline{t},s(0))$ has a successful derivation. Therefore $\leftarrow r(\underline{x})$ has a successful derivation, and hence $r(\underline{x}) \in T_{P_1}^{\mathsf{co}} \uparrow \omega$.

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

2) We define a program P_2 and show that $\overline{K} \leqslant gfp(T_{P_2}^{\mathsf{co}})$ (i.e., $x \in \overline{K}$ iff $gfp(T_{P_2}^{\mathsf{co}})$, equivalent to $K \leqslant \mathsf{co}\text{-}B_{P_2} \setminus gfp(T_{P_2}^{\mathsf{co}})$). 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):

$$q(X) \quad \leftarrow \quad p(X,0). \qquad\qquad p(X,Y) \quad \leftarrow \quad h(X,Y,0), p(X,s(Y)).$$

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

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: $q(\underline{x}) \notin gfp(T_{P_2}^{\mathsf{co}})$. □

Let us observe that if the language includes the built-in $\neq$ predicate on terms, as in Prolog II, then, in general, $T_P^{\mathsf{co}} \downarrow \omega \neq gfp(T_P^{\mathsf{co}})$ and the complement of *gfp* is no longer r.e. (it follows immediately from Blair's result, Theorem 4.1).

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 4.4, where we have only ground infinite computations for $A = \mathtt{inf\_list}(0,[\underline{0},\underline{1},\underline{2},\ldots])$): $A \in gfp(T_P^{\mathsf{co}})$ 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 (Simon et al. 2006). 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

$lfp(T_P)$ a r.e. complete set, in general (c.f. Theorem 4.1). 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 $gfp(T_P^{\mathsf{co}})$. Thus, we consider the complete Herbrand Universe and Base and the operator T_P^{co}:

Definition 5.1 (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 $a \in gfp(T_P^{\mathsf{co}})$.
- 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.

Corollary 5.1 (*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 $gfp(T_P^{\mathsf{co}})$ is, in general, productive (Theorem 4.2). This means that there is no computable procedure capable of verifying whether an atom belongs to $\Upsilon(gfp(T_P^{\mathsf{co}}))$ that works for all atoms (otherwise $\Upsilon(gfp(T_P^{\mathsf{co}}))$ would be r.e., a contradiction). □

Anyway, any productive set has an infinite r.e. subset. A procedure would aim to identifying such a subset(s), thus computing an approximation of $gfp(T_P^{\mathsf{co}})$ (possibly, including strictly $lfp(T_P^{\mathsf{co}})$). The procedure $\vdash_{\mathsf{co}}$ defined below and the procedure in (Simon et al. 2006) aim at identify such a set. However, if the procedures finitely fails, they correctly state that the atom does not belong to $gfp(T_P^{\mathsf{co}})$.

A computation 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 (Simon et al. 2006) 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 (Bonatti 2001; Bonatti et al. 2008)). An *hypothetical goal* is a list of pairs $(A \Box S)$ where A is an atom and S is a set of atoms (hypotheses), together with a system of equations. Given a goal $G = E \Box A_1, \ldots, A_n$, 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 $\alpha(G) = \langle (A_1, \emptyset), \ldots, (A_n, \emptyset) \Box E \rangle$.

Definition 5.2 (co-*LP operational semantics (revisited)*) Given a definite clause program P and an hypothetical goal $G = \langle (A_1, S_1), \ldots, (A_n, S_n) \Box E \rangle$, 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: $G' = \langle (A_1, S_1), \ldots, (A_{i-1}, S_{i-1}), (B_1, S'), \ldots, (B_m, S'), (A_{i+1}, S_{i+1}), \ldots, (A_n, S_n) \Box E' \rangle$

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:
   $G' = \langle (A_1,S_1),\ldots,(A_{i-1},S_{i-1}),(A_{i+1},S_{i+1}),\ldots,(A_n,S_n) \Box E' \rangle$

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. $G_k = \langle \varepsilon \Box E \rangle$ 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) aim to capture (some) atoms belonging to $gfp(T_P) \setminus lfp(T_P)$ that are co-inductively supported.

Example 5.1 Let us consider the program of Example 3.1 and the goal $\leftarrow p(0)$. The following is a successful derivation: $\langle (p(0),\emptyset) \Box \emptyset \rangle \vdash_{\mathsf{co}1} \langle (q(X_0),\{p(0)\}) \Box \emptyset \rangle \vdash_{\mathsf{co}1}$
$\langle (q(X_1),\{p(0),q(X_0)\}) \Box \{X_0 = s(X_1)\} \rangle \vdash_{\mathsf{co}2} \langle \varepsilon \Box \{X_0 = s(X_1), X_0 = X_1\} \rangle$.
$E = \{X_0 = s(X_1), X_0 = X_1\}$ 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 goal ground. ☑

Example 5.2 Let P_1 consist of the following clause $p([0,1|X]) \leftarrow p(X)$ and let us analyze the following two successful derivations for the goal $\leftarrow p(X)$.

$$\langle (p(X),\emptyset) \Box \emptyset \rangle \vdash_{\mathsf{co}1} \langle (p(X_1),\{p(X)\}) \Box \{X = [0,1|X_1]\} \vdash_{\mathsf{co}2} \langle (\varepsilon \Box \{X = X_1, X = [0,1|X_1]\} \rangle$$

which yields the computed answer system $\{X = [0,1|X]\}$.

$$\begin{array}{l} \langle (p(X),\emptyset) \Box \emptyset \rangle \vdash_{\mathsf{co}1} \langle (p(X_1),\{p(X)\}) \Box \{X = [0,1|X_1]\} \rangle \vdash_{\mathsf{co}1} \\ \langle (p(X_2),\{p(X),p(X_1)\}) \Box \{X = [0,1|X_1], X_1 = [0,1|X_2]\} \rangle \vdash_{\mathsf{co}2} \\ \langle \varepsilon \Box \{X = X_2, X = [0,1|X_1], X_1 = [0,1|X_2]\} \rangle \end{array}$$

which yields the computed answer system $\{X = [0,1,0,1|X]\}$. An infinite number of successful computations (actually, with the same answer, in this special case) can be computed. ☑

Theorem 5.1 (*Correctness*) Let P be a definite clause program. If there is a $\vdash_{\mathsf{co}}$ derivation for $\langle (A,\emptyset) \Box E \rangle$ with c.a.s. θ, then $P \models_{\mathsf{co}} A\theta\gamma$ for every term substitution γ solution of $E\theta$.

Proof If in the derivation only rule $\vdash_{\mathsf{co}1}$ is applied, then $AE\gamma \in T_P^{\mathsf{co}} \uparrow \omega$ by correctness of Prolog II procedure (Jaffar and Stuckey 1986).

If, instead, rule $\vdash_{\mathsf{co}2}$ is employed at least once, the proof is similar to that of the *pumping lemma*: we show that from a finite successful derivation we are able to produce an infinite derivation using only rule $\vdash_{\mathsf{co}1}$, that, again by (Jaffar and Stuckey 1986) would prove that $AE\gamma \in T_P^{\mathsf{co}} \downarrow \omega$.

Let us consider a successful $\vdash_{\mathsf{co}}$ derivation for $\langle (A,\emptyset) \Box E \rangle$ in which $\vdash_{\mathsf{co}2}$ is employed at least once. Let us split the derivation in three parts:
1. A first part of the proof (possibly of length zero), starting from $\langle (A,\emptyset) \Box E \rangle$ 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. Namely, we have $\langle (A,\emptyset) \Box E \rangle \vdash_{\mathsf{co}1} \cdots \vdash_{\mathsf{co}1} \langle (p(\vec{t}),H),\ldots \Box E_k \rangle$.
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:

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

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 $\langle \varepsilon \Box E_k \cup \{\vec{t} = \vec{a}, \vec{s} = \vec{t}\} \cup E' \cup E'' \rangle$. 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:

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

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 $\vec{d} = \vec{e}$, a renaming with fresh variables of the rule $q(\vec{e}) \leftarrow p(\vec{s}), \ldots$ can be applied and this can be repeated forever.

This proof schema can be repeated for any length of the intermediate part of the derivation. □

We have already proved the *incompleteness* of any procedure for computing $\models_{\mathsf{co}}$, even restricting to rational terms. Let us see a typical example of rational atom that is not computed. From Example 4.3 we learned that $p(0) \in gfp(T_P^{\mathsf{co}})$. 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).

A meta-interpreter consistent with the $\vdash_{\mathsf{co}}$ definition is reported in (Ancona 2012).

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 and, more in general, static analysis and symbolic execution of programs (Ancona and Lagorio 2009; Ancona et al. 2011; Ancona and Lagorio 2011; Ancona and Lagorio 2012), verification of real time systems (Saeedloei and Gupta 2010), model checking, and SAT solvers (Min and Gupta 2009).

In this paper we revisit and deepen some aspects of the foundations of co-LP (Simon 2006); in particular, we provide a simpler but equivalent operational semantics, whose proof of correctness can be directly derived from early results from Jaffar and Stuckey (Jaffar and Stuckey 1986).

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).

References

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

ANCONA, D. 2012. Regular corecursion in prolog. In *SAC*, S. Ossowski and P. Lecca, Eds. ACM, 1897–1902.

ANCONA, D., CORRADI, A., LAGORIO, G., AND DAMIANI, F. 2011. Abstract compilation of object-oriented languages into coinductive CLP(X): can type inference meet verification? In *Formal Verification of Object-Oriented Software International Conference, FoVeOOS 2010, Paris, France, June 28-30, 2010, Revised Selected Papers*, B. Beckert and C. Marché, Eds. Lecture Notes in Computer Science, vol. 6528. Springer.

ANCONA, D. AND LAGORIO, G. 2009. Coinductive type systems for object-oriented languages. In *ECOOP 2009 - Object-Oriented Programming*, S. Drossopoulou, Ed. Lecture Notes in Computer Science, vol. 5653. Springer, 2–26.

ANCONA, D. AND LAGORIO, G. 2011. Idealized coinductive type systems for imperative object-oriented programs. *RAIRO - Theoretical Informatics and Applications 45*, 1, 3–33.

ANCONA, D. AND LAGORIO, G. 2012. Static single information form for abstract compilation. In *Theoretical Computer Science (IFIP TCS 2012)*, J. C. Baeten, T. Ball, and F. S. de Boer, Eds. Lecture Notes in Computer Science, vol. 7604. Springer, 10–27.

APT, K. R. 1988. Introduction to Logic Programming. Tech. Rep. TR-87-35, Department of Computer Sciences, The University of Texas at Austin.

BLAIR, H. A. 1982. The recursion-theoretic complexity of the semantics of predicate logic as a programming language.

BONATTI, P. A. 2001. Resolution for skeptical stable model semantics. *J. Autom. Reasoning 27*, 4, 391–421.

BONATTI, P. A., PONTELLI, E., AND SON, T. C. 2008. Credulous resolution for answer set programming. In *AAAI*, D. Fox and C. P. Gomes, Eds. AAAI Press, 418–423.

COLMERAUER, A. 1984. Equations and inequations on finite and infinite trees. In *FGCS*. 85–99.

COURCELLE, B. 1983. Fundamental properties of infinite trees. *Theor. Comput. Sci. 25*, 95–169.

DOVIER, A., PIAZZA, C., AND POLICRITI, A. 2004. An efficient algorithm for computing bisimulation equivalence. *Theor. Comput. Sci. 311*, 1-3, 221–256.

JAFFAR, J. AND STUCKEY, P. J. 1986. Semantics of infinite tree logic programming. *Theoretica Computer Science 46*, 141–158.

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

MARTELLI, A. AND MONTANARI, U. 1982. An efficient unification algorithm. *ACM Trans. Program. Lang. Syst. 4*, 2, 258–282.

MIN, R., BANSAL, A., AND GUPTA, G. 2009. Towards predicate answer set programming via coinductive logic programming. In *AIAI*, L. S. Iliadis, I. Maglogiannis, G. Tsoumakas, I. P. Vlahavas, and M. Bramer, Eds. IFIP Advances in Information and Communication Technology, vol. 296. Springer, 499–508.

MIN, R. AND GUPTA, G. 2009. Coinductive logic programming and its application to boolean sat. In *FLAIRS Conference*.

PAIGE, R. AND TARJAN, R. E. 1987. Three partition refinement algorithms. *SIAM J. Comput. 16*, 6, 973–989.

ROGERS, JR, H. 1987. *Theory of Recursive Functions and Effective Computability*. The MIT Press.

SAEEDLOEI, N. AND GUPTA, G. 2010. Verifying complex continuous real-time systems with coinductive CLP(R). In *Proc. of LATA 2010*. Lecture Notes in Computer Science. Springer.

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

SIMON, L., BANSAL, A., MALLYA, A., AND GUPTA, G. 2007. Co-logic programming: Extending logic

programming with coinduction. In *ICALP*, L. Arge, C. Cachin, T. Jurdzinski, and A. Tarlecki, Eds. Lecture Notes in Computer Science, vol. 4596. 472–483.

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

Acknowledgments

The authors wish to thank Andrea Formisano, Alberto Policriti, and Peter Stuckey for the useful discussions on the research pursued in this paper. The authors acknowledge the ICLP reviewers who provided constructive comments.