

A coalgebraic approach to unification semantics of logic programming [★]

Roberto Bruni¹, Ugo Montanari¹, and Giorgio Mossa²

¹ Dipartimento di Informatica, Università di Pisa, Italy

² Dipartimento di Matematica, Università di Pisa, Italy

Abstract. In the version of logic programming (LP) based on interpretations where variables occur in atoms, a goal reduction via unification can be seen as a transition labelled by the most general unifier. Categorically, it is thus natural to model a logic program as a coalgebra. In the paper we represent: (i) goals as the substitutive monoid freely generated by the predicate symbols; (ii) the LTS as the structured coalgebra defined by the SOS rules implicit in the LP semantics; (iii) the bisimulation semantics of a logic program as its image on the final coalgebra.

1 Introduction

Logic programming is a paradigm based on first order Horn clauses and SLD resolution. Its fundamental ingredient is to be equipped both with an operational semantics, based on goal reduction via resolution, and with a declarative semantics. The second is defined both in terms of satisfaction (à la Tarski) of the clauses on (standard) interpretations of the Herbrand base (minimal model), and in terms of the least fix point of a transformation on interpretations.

The three approaches are proved equivalent, thus allowing for the famous paradigm Algorithm = Logic + Control. We refer to [18] for the programming motivation and to [22] for the underlying theory.

In the classical version, all the semantics are given as *ground* interpretations, i.e. as sets of *true* atoms without variables. However, it is possible to extend the basic approach to interpretations and refutations containing also atoms with variables. Then also the meaning of a goal changes: it considers also refutations of non-ground atoms. It is defined as the set of all the answer substitutions (possibly with variables) computed by its refutations. It is easy to see that the corresponding operational semantics is more informative, namely finer.

For instance, the goal $P(x)$ is assigned the same semantics in the case of ground interpretations for the two programs

$$\{P(x) :- \Box\} \text{ and } \{P(x) :- \Box, P(a) :- \Box\}$$

[★] Research supported by the MIUR PRINs 201784YSZ5 *ASPRA: Analysis of program analyses* and by University of Pisa PRA.2018.66 *DECLWARE: Metodologie dichiarative per la progettazione e il deployment di applicazioni*.

namely $\{(t/x) \mid t \text{ is a term}\}$ (typically an infinite set), while the semantics are different when atoms with variables are considered, namely $\{(y/x)\}$ (a singleton) and $\{(y/x), (a/x)\}$ (a two-element set) respectively. Here answer substitution (y, x) is defined up to variable renaming \simeq , e.g. $(y/x) \simeq (z/x)$.

While from a logical point of view the classical semantics is satisfactory, since it includes all the (ground) logical consequences of the clauses, the extended semantics is more convenient from a programming point of view: for instance, for the query $P(x)$ the former program cannot select any specific answer, while the latter does. Thus the two programs should be regarded as different. As for the classical version, also for the extended theory there are declarative semantics based on suitable notions of S-interpretation, S-minimal model and S-transformation which correspond to the extended operational semantics [12].

It is easy to see that the extended operational semantics can be seen as a labelled transition system (LTS), where states are goals, transitions are goal reductions and labels are the substitutions computed by unification in the corresponding reductions. The computed answer substitution corresponding to a path is obtained by composing the substitutions of its transitions, while the semantics of a goal G is the set of substitutions, projected on the variables of G , computed by all the paths from G to the empty goal. In addition, both states and transitions of the LTS have a simple algebraic structure.

The clean structure we outlined here has inspired several authors to take advantage of the universal constructions of category theory to embed operational non-ground logic programming into other well-studied mathematical structures. A general approach for equipping transition systems with an algebraic semantics has been applied to logic programming in the seminal papers [10,8]. Generalizations to richer structures have been studied in [13,2]. Coalgebras turn out to be particularly fit [15,17,16,4,5], as it should be expected since they have been particularly successful in modelling LTS with all kinds of structures. The meta model of tile systems has been adopted in [7]. An important design point is to choose the category where the coalgebra lives. If the coalgebra can be lifted from category **Set** to a category of algebras (obtaining a structured coalgebra/bialgebra) then arrows become at the same time bisimulations and homomorphisms, thus guaranteeing that bisimilarity is a congruence.

A critical point about categorical LTS for logic programming is about correctly modelling unification, in particular with the co/bialgebraic approach. The universal property of most general unifiers (mgu's) is essential: while restriction to mgu's is not necessary, since any unifier will derive correct answers, the number of unifiers is often infinite, and unification matches completeness and efficiency. Most general unifiers are perfectly representable in category theory: they arise from the pushout construction in the category of substitutions. However to match unification with the homomorphism requirements of coalgebras is difficult. To the authors' knowledge, only [7] and [4] succeed in this task.

In this paper, we define a structured coalgebra for operational non-ground logic programming, which correctly considers only reductions with mgu's. Our construction works as follows. First, we define goals as the algebra of substitutive

monoids (SM) freely generated by the set of predicate symbols. The algebraic specification of SM is obtained as the tensor product of the specifications of monoids and of substitutions. The latter specification has natural numbers as sorts (the number of variables) and contains all the substitutions as unary operations, with axioms for substitution composition and unity. The tensor product construction automatically inserts all the exchange axioms, e.g. between the monoidal operation (i.e. the logical conjunction) and substitutions. Second, as base category for the coalgebra we choose the functor category of SM-algebras, but without axioms. Third, we model the LTS as the structured coalgebra defined by the SOS rules implicit in the logic programming semantics. Finally, given a logic program, its image in the final coalgebra yields its bisimulation semantics.

We emphasize the simplicity of our construction for assigning a natural algebraic structure to goals: the ordinary logic representation as conjunction of atomic goals becomes the standard form of the terms of our initial algebra, equipped “automatically” by exchange axioms.

The above construction makes sure that the structural coalgebra we defined fully corresponds to the operational semantics of non-ground logic programming. However the situation is different when we consider abstract semantics. In fact, for logic programming, as mentioned above, the semantics of a goal G is the set of substitutions, projected on the variables of G , computed by all the paths from G to the empty goal. This situation corresponds to a language semantics of LTS (the set of labels of the paths to final states), with the additional difference that the monoid of the labels (the substitutions) is not free. In the case of coalgebras, the abstract semantics of a LTS is represented by its image on the terminal coalgebra, where all *bisimilar* states are identified. It is easy to see that the categorical, bisimilar abstract semantics is finer than the non-ground semantics.

We are not aware of coalgebraic approaches which yield the classical non-ground semantics. On the other hand, bisimilarity semantics may be natural and convenient when considering LP-based process description languages. In this context, we are not interested in logic computations as refutations of goals for problem solving or artificial intelligence, but we consider LP as a goal rewriting mechanism. We consider logic subgoals as concurrent communicating processes that evolve according to the rules defined by the clauses and that use unification as the fundamental interaction primitive. A presentation of this kind of use of logic programming can be found in [7,19].

Structure of the paper In Section 2 we fix the notation and recall the operational semantics of logic programming (SLD derivation) together with some preliminaries about coalgebraic representations of labelled transition systems. In Section 3 we characterize the algebra of goals as the initial model for the theory of substitutive monoids. In Section 4 we define the coalgebra for SLD derivation and prove that it admits a terminal object. Some final remarks are in Section 5.

2 Background

In the next subsections we will recall the basic notions of logic programming and structured coalgebras that we will need.

2.1 Operational semantics with non-ground atoms à la Levi-Palamidessi-Falaschi-Martelli.

Definition 1 (Signatures). A logic signature is a pair of sets of symbols (Σ, Π) and an arity function $\text{ar}: \Sigma + \Pi \rightarrow \mathbb{N}$ associating with each symbol in Σ and Π a natural number, called its arity. The symbols in Σ are the operation symbols and the symbols in Π the predicates.

The set of operation symbols Σ with the relative restriction of the arity function form what is called an algebraic signature.

In general, given an algebraic signature Σ and a set of variables X , we will denote by $T_\Sigma(X)$ the set of terms with variables in X and by $T_\Sigma = T_\Sigma(\emptyset)$ the set of ground terms.

Definition 2 (Atoms, goals). Atomic formulas, or atoms for short, are expressions of the form $P(t_1, \dots, t_n)$ where $P \in \Pi$ is a predicate symbol with arity n , and the t_i 's are just terms in the set $T_\Sigma(X)$, for some set variables X . A goal G is a finite conjunction of atomic formulas over the same set of variables.

In what follows we will identify goals with finite lists of atoms, so we will write $G \equiv A_1 \dots A_k$ for the goal made of the conjunction of the atoms A_i 's. We will use the symbol \square to denote the empty conjunction of atoms, the empty goal.

Definition 3 (Substitutions). Given two sets X and Y a substitution from (terms over variables in) X to (terms over variables in) Y is a function of the form $\sigma: X \rightarrow T_\Sigma(Y)$.

Remark 1. These functions have a natural action on terms: for every $\sigma: X \rightarrow T_\Sigma(Y)$ and every term $t \in T_\Sigma(X)$ we have the term $\sigma(t) \in T_\Sigma(Y)$ obtained by replacing the variable occurrences in t with their images via σ . Similarly, for each atom A and each goal G in the set of variables X we have the atom and goal $\sigma(A)$ and $\sigma(G)$, in the set of variables Y . These are obtained respectively by A and G replacing the occurrences of the variables with their images via σ .

Substitutions form the morphisms of a category whose objects are the sets of variables. Given a pair of substitutions $\sigma: X \rightarrow T_\Sigma(Y)$ and $\tau: Y \rightarrow T_\Sigma(Z)$ their composition is the substitution $\tau \circ \sigma: X \rightarrow T_\Sigma(Z)$ defined as the function sending every variable $x \in X$ into the term $\tau(\sigma(x))$, the result of applying τ to the term $\sigma(x)$. Identity substitutions are given by the functions $\text{id}_X: X \rightarrow T_\Sigma(X)$ sending each variable in itself, seen as an element of $T_\Sigma(X)$.

Definition 4 (Substitutions preorder). Given two substitutions σ and σ' we say that σ is more general than σ' if there is a substitution τ such that $\sigma' = \tau \circ \sigma$.

Table 1: Inference rules for SLD-resolution.

$\frac{H :- B \in \mathbb{P} \quad \sigma = \text{mgu}(A, \rho(H))}{\mathbb{P} \models A \Rightarrow_{\sigma} \sigma(\rho(B))}$	Atomic goal
<p>where ρ is a variable renaming such that $\rho(H)$ and $\rho(B)$ have no variable in common with A</p>	
$\frac{\mathbb{P} \models G \Rightarrow_{\sigma} F}{\mathbb{P} \models G', G \Rightarrow_{\sigma} \sigma(G'), F} \quad \frac{\mathbb{P} \models G \Rightarrow_{\sigma} F}{\mathbb{P} \models G, G' \Rightarrow_{\sigma} F, \sigma(G')}$	Conjunctive goal
<p>where the goal G' and F have no variable in common</p>	

This relation induces a preorder on the substitutions.

Definition 5 (Most general unifier I). *Given two terms/atoms/goals t_1 and t_2 over the same set of variables, a unifier for them is a substitution σ such that $\sigma(t_1) = \sigma(t_2)$. A most general unifier for t_1 and t_2 is a unifier that is most general in the sense of definition 4.*

Remark 2. The *mgu*'s are unique up isomorphism, meaning that if σ and σ' are two *mgu*'s for the same terms there is a unique substitution γ such that $\sigma' = \gamma \circ \sigma$ and this γ is an isomorphism in the above mentioned category of substitutions.

Next we introduce logic programs and their operational semantics.

Definition 6 (Horn clauses, logic programs). *A Horn clause is an expression of the form $H :- B$ where H is an atom and B is a goal called respectively the head and the body of the clause. Without loss of generality we assume the head and body are formulas over the same set of variables. A finite set of Horn clauses is called a logic program.*

A Horn clause is basically a formula stating that the head is a logical consequence of the body, i.e., that every valuation satisfying the body satisfies the head too. In particular any clause whose body is the empty goal \square is a formula stating that the head is satisfied by all valuations.

Logic programs are able to express computations; they do so via the (operational) SLD-reduction semantics. This semantics is described in Table 1 via inference rules that describe the transitions $G \Rightarrow_{\sigma} F$ of a *labelled transition system* whose states are goals and whose labels are substitutions.

Each sequent of the form $\mathbb{P} \models G \Rightarrow_{\sigma} F$ asserts that the program \mathbb{P} produces a computation step from a state G to a state F with an observation σ . These computation steps represent the classical behavior of logic program systems, in which at each point of a computation the system selects an atom from the current goal, it tries to unify the selected atom with the head of a clause in the

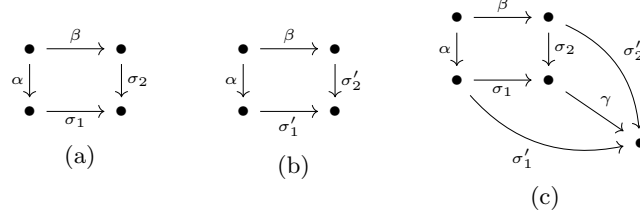


Fig. 1: Mgu's via pushout

program (up to a renaming with fresh names), and then it updates the current goal, replacing the atom with the body of the clause and then applying the computed unifier.

From a logic point of view the sequent $\mathbb{P} \models G \Rightarrow_\sigma F$ states that every substitution γ which makes the goal $\gamma(F)$ a consequence of the axioms in \mathbb{P} makes also $\gamma(\sigma(G))$ a consequence of \mathbb{P} .

We stress the fact that in logic programming we are always interested in unifying goals and clauses with distinct names, which is the reason for the renaming in the *atomic goal rule* in Table 1. It is possible to avoid this variable renaming changing the definitions of unifiers and mgu's.

Definition 7 (Most general unifier II). *Given a pair of terms/atoms/goals t_1 and t_2 , a unifier for them is a pair of substitutions (σ_1, σ_2) such that $\sigma_1(t_1) = \sigma_2(t_2)$. Given two unifiers $\sigma = (\sigma_1, \sigma_2)$ and $\sigma' = (\sigma'_1, \sigma'_2)$ we say that σ is more general than σ' if there is a substitution γ such that $\sigma'_i = \gamma \circ \sigma_i$ for $i = 1, 2$. As before, the relation of being more general induces a preorder on the unifiers. A most general unifier (mgu) for the terms t_1 and t_2 is a unifier which is most general among all the possible unifiers.*

The notion of unifier and mgu admit a nice categorical description.

Remark 3. Every atom $P(t_1, \dots, t_n)$ can be written in the form $\sigma_t(P(x_1, \dots, x_n))$ where $\{x_1, \dots, x_n\}$ is a canonical set of variables, $\sigma_t: \{x_1, \dots, x_n\} \rightarrow T_\Sigma(X)$ is the unique substitution such that $\sigma_t(x_i) = t_i$ for each i (where X is the set of variables over which $P(t_1, \dots, t_n)$ is defined).

Remark 3 provides a canonical decomposition of an atom as the result of a substitution to a *canonical predicate*, namely $P(x_1, \dots, x_n)$. As a matter of notation, in what follows we write just P for $P(x_1, \dots, x_n)$.

Remark 4 (Mgu's via pushout). Two atoms $\alpha(P)$ and $\beta(Q)$ unify only when $P = Q$, and in this case a unifier for them is a pair of substitutions $\sigma = (\sigma_1, \sigma_2)$ such that $\sigma_1 \circ \alpha = \sigma_2 \circ \beta$, i.e. such that the square in Fig. 1a commutes in the category of substitutions. Such a square is an mgu if for any other commuting square of the form in Fig. 1b there is a, necessarily unique, γ which makes

commute the diagram in Fig. 1c. This is equivalent to say that $(\alpha, \beta, \sigma_1, \sigma_2)$ is a pushout square. So, unifiers are commutative squares in the form above and mgu's are those squares that are pushouts.

Using this characterization of unification via pushouts we can modify the *Atomic goal* rule of Table 1 with the following rule:

$$\frac{\alpha(P) : -B \in \mathbb{P} \quad (\alpha, \beta, \sigma_1, \sigma_2) \text{ is a pushout}}{\mathbb{P} \models \beta(P) \Rightarrow_{\sigma_2} \sigma_1(B)}.$$

In this way the LTS is not changed, but there is no need of creating new variables.

2.2 LTSs as coalgebras.

Definition 8 (F -coalgebra). *Given a category \mathbf{C} and an endofunctor $F : \mathbf{C} \rightarrow \mathbf{C}$ a coalgebra for the functor F , or F -coalgebra for short, is a pair $\langle X, \alpha \rangle$ with X an object of \mathbf{C} and $\alpha : X \rightarrow F(X)$. Given two coalgebras $\langle X, \alpha \rangle$ and $\langle Y, \beta \rangle$ a cohomomorphism h from $\langle X, \alpha \rangle$ to $\langle Y, \beta \rangle$, written as $h : \langle X, \alpha \rangle \rightarrow \langle Y, \beta \rangle$, is given by a morphism $h : X \rightarrow Y$ such that the following diagram commutes.*

$$\begin{array}{ccc} X & \xrightarrow{h} & Y \\ \alpha \downarrow & & \downarrow \beta \\ F(X) & \xrightarrow{F(h)} & F(Y) \end{array}$$

F -coalgebras and relative cohomomorphisms form the category $\mathbf{Coalg}(F)$.

Coalgebras provide an elegant way to encode different notions of dynamic systems in a categorical framework. As an example we can consider the case of LTSs. We recall that a labeled transition system is a triple $\langle S, L, \longrightarrow \rangle$ where S is a set of states, L is a set of labels, and $\longrightarrow \subseteq S \times L \times S$ is a transition relation. Every LTS $\langle S, L, \longrightarrow \rangle$ can be regarded as a coalgebra for the endofunctor $\mathcal{P}_L : \mathbf{Set} \rightarrow \mathbf{Set}$ defined on objects as $\mathcal{P}_L(X) = \mathcal{P}(L \times X)$ where \mathcal{P} is the *powerset functor*. If $\langle S, L, \longrightarrow \rangle$ is an LTS, the relation $\longrightarrow \subseteq S \times L \times S$ gives the function $p : S \rightarrow \mathcal{P}_L(S)$ that sends every $s \in S$ into the set $p(s) = \{(l, t) \mid s \xrightarrow{l} t\}$.

One of the most interesting thing about coalgebras is that they give an abstract semantics for dynamic systems in term of final objects: if $\langle X, h \rangle$ is a coalgebra, we say that two states, i.e. two elements of X , are *bisimilar* if they have the same image via the unique cohomomorphism $t : \langle X, h \rangle \rightarrow \langle T, \tau \rangle$ into a final coalgebra $\langle T, \tau \rangle$. This definition of *bisimilarity* generalizes the classical one for transition systems, meaning that two states are bisimilar in the classical sense if and only if they are bisimilar in the coalgebraic sense.

The advantage of coalgebras over classical LTS is that the states' space now can be an object of a generic category, not just a set. Hence coalgebras allow to work with states that have an additional structure. This justifies the name *structured coalgebras* for coalgebras over categories of structures, in particular we will be interested in coalgebras over categories of algebras.

3 Algebraic Structures

In this section we introduce an algebraic structure over the goals of logic programming. The main result is to provide a characterization of this *algebra of goals* as the initial model for *the theory of substitutive monoids*. This theory is many-sorted and its sorts represents the number of variables that can occur in a goal. Each sort is equipped with a binary operation and a constant, representing the conjunction and the empty goal respectively, and there are unary typed operations that correspond to substitution operations of logic programming.

Instead of presenting the theory of substitutive monoids directly, we first recall the *theory of monoids*, then, we introduce the *theory of substitutions*, and finally we take the tensor product of these theories and add some constants. This way the different operations of substitutive monoids are presented to the reader in a gradual way and the distributive axioms between the two algebras are introduced automatically by the tensor product construction.

3.1 Monoids, substitutions and substitutive monoids

Definition 9 (The theory of monoids). *We let $\Gamma_{\mathbf{Mon}} = (S_{\mathbf{Mon}}, \Sigma_{\mathbf{Mon}}, E_{\mathbf{Mon}})$ be the algebraic theory of monoids having a unique sort M , a binary operation $\cdot : M \times M \rightarrow M$ and a constant \square . The axioms of $E_{\mathbf{Mon}}$ are the following*

$$\begin{aligned} x \cdot (y \cdot z) &= (x \cdot y) \cdot z \text{ for all } x, y, z \text{ (associativity);} \\ x \cdot \square &= \square \cdot x = x \text{ for all } x \text{ (unit).} \end{aligned}$$

Algebras of this theory are monoids, i.e. algebraic structures with a binary associative operation \cdot whose unit is \square .

The monoidal structure captures the algebraic operation of goal conjunction. Indeed, for every set of variables X , the goals having free variables in X form a monoid with conjunction and the unit of the monoid is the empty goal.

Definition 10 (The theory of substitutions). *Given a signature Σ , we let $\Gamma_{\mathbf{Sub}} = (S_{\mathbf{Sub}}, \Sigma_{\mathbf{Sub}}, E_{\mathbf{Sub}})$ be the algebraic theory of substitutions over Σ . The set of sorts is the countable set $S_{\mathbf{Sub}} = \{\underline{n} : n \in \mathbb{N}\}$. For every substitution $\sigma : \{x_1, \dots, x_n\} \rightarrow T_{\Sigma}(\{x_1, \dots, x_n\})$ we have an operation symbol $\underline{\sigma} \in \Sigma_{\mathbf{Sub}}$ with arity $\underline{\sigma} : \underline{n} \rightarrow \underline{m}$ and we have the following axioms*

$$\begin{aligned} \tau(\underline{\sigma}(x)) &= \tau \circ \underline{\sigma}(x) \text{ for all } \sigma : \underline{n} \rightarrow \underline{m} \text{ and } \tau : \underline{m} \rightarrow \underline{k} \text{ and any } x; \\ \underline{\text{id}}_{\underline{n}}(x) &= x \text{ for any identity substitution } \text{id}_{\underline{n}} : \underline{n} \rightarrow \underline{n} \text{ and any } x. \end{aligned}$$

The algebras for this theory are rather simple, they are basically **Set**-valued functors (not necessarily cartesian ones) from the (*opposite of the*) *Lawvere theory over the signature Σ* , that is functors from the category of finite sets of variables and substitutions between them, in the sense of logic.

While the monoidal operation captures goal conjunction, the substitution operations capture the operation of variable instantiation, which is fundamental for

logic programming. Again the goals provide examples of this substitutive structure in which the operations of Γ_{Sub} are interpreted with their corresponding substitutions.

As seen above, goals mix together monoidal and substitutive structures, but these structures interact with each other in a specific way. Their interaction gives rise to a different algebraic structure which is captured by the so called *tensor product of the two theories*.

Using the machinery of algebraic theories we can build a new theory as the tensor product of Γ_{Mon} and Γ_{Sub} , which we call the *theory of substitutive monoids* and denote by Γ_{SM} . We will not describe the construction in details (see e.g. [14]), instead we describe the resulting theory.

Definition 11 (The theory of substitutive monoids). *The theory of substitutive monoids $\Gamma_{\text{SM}} = (S_{\text{SM}}, \Sigma_{\text{SM}}, E_{\text{SM}})$ has sorts those of Γ_{Sub} , that is $S_{\text{SM}} = S_{\text{Sub}}$, each operation $\sigma: \underline{n} \rightarrow \underline{m}$ in Σ_{Sub} is also an operation of Σ_{SM} with the same arity and for every sort $\underline{n} \in S_{\text{SM}}$ we have a binary typed operation $\cdot_{\underline{n}}: \underline{n} \times \underline{n} \rightarrow \underline{n}$ and a constant $\square_{\underline{n}}: \underline{n}$. The axioms in E_{SM} contain those in E_{Sub} , with the addition, for every sort \underline{n} , of the following monoid axioms*

$$x \cdot_{\underline{n}} (y \cdot_{\underline{n}} z) = (x \cdot_{\underline{n}} y) \cdot_{\underline{n}} z \quad x \cdot_{\underline{n}} \square_{\underline{n}} = \square_{\underline{n}} \cdot_{\underline{n}} x = x$$

and of axioms of the form

$$\sigma(x \cdot_{\underline{n}} y) = \sigma(x) \cdot_{\underline{m}} \sigma(y) \quad \sigma(\square_{\underline{n}}) = \square_{\underline{m}}$$

for every substitution operation $\sigma: \underline{n} \rightarrow \underline{m}$ in $\Sigma_{\text{SM}} \cap \Sigma_{\text{Sub}}$.

We call the Γ_{SM} -algebras substitutive monoids or also **SM**-algebras. An algebra for Γ_{SM} is basically a countable family of monoids (parametrized by the sorts) with a family of monoid homomorphisms between them (parametrized by the substitutions), whence the name substitutive monoids. Another way to see these algebras is as functors from the above mentioned category of substitutions into the category of monoids. Goals provide algebras for this algebraic theory, the sorts are interpreted by sets of goals with fixed-finite sets of variables, the monoidal operations are given by conjunction operations and finally the substitution operations are given by the variable substitutions.

Definition 12 (The theory of Π -substitutive monoids). *The theory of Π -substitutive monoids $\Gamma_{\Pi\text{-SM}} = (S_{\Pi\text{-SM}}, \Sigma_{\Pi\text{-SM}}, E_{\Pi\text{-SM}})$ has sorts and equations as those of Γ_{SM} but the operation symbols are given by $\Sigma_{\Pi\text{-SM}} = \Sigma_{\text{SM}} \cup \Pi$, that is they are those of Σ_{SM} plus the predicates of the logic signature. Each predicate $P \in \Pi$ with arity n is interpreted as a constant symbol of type $P: \underline{n}$. The other operators, inherited from Σ_{SM} , keep the signature they have in Γ_{SM} .*

The algebras of $\Gamma_{\Pi\text{-SM}}$ are substitutive monoids with selected constants parametrized by predicates. In the next section we focus on the *most important Π -SM-algebra*, the algebra of goals: this will show how $\Gamma_{\Pi\text{-SM}}$ is actually the *theory of goals*, meaning that it characterizes the algebraic structure of goals.

3.2 The goal algebra as the initial Π -SM-algebra

Definition 13 (The goal algebra). *Goals form a Π -SM-algebra \mathbb{G} such that*

- \mathbb{G} interprets sort $\underline{n} \in S_{\Pi\text{-SM}}$ into the set $\mathbb{G}_{\underline{n}}$ of the goals with free variables in the canonical set $\{x_1, \dots, x_n\}$;
- each predicate symbol $P: \underline{n}$ is interpreted in the atomic formula

$$P^{\mathbb{G}} = P(x_1, \dots, x_n);$$

- each substitution symbol $\underline{\sigma}: \underline{n} \rightarrow \underline{m}$ is interpreted in the corresponding substitution operation $\underline{\sigma}^{\mathbb{G}} = \sigma: \mathbb{G}_{\underline{n}} \rightarrow \mathbb{G}_{\underline{m}}$;
- each monoid operation $\cdot_{\underline{n}}$ is interpreted into goal conjunction, i.e. $\cdot_{\underline{n}}^{\mathbb{G}} = \wedge: \mathbb{G}_{\underline{n}} \times \mathbb{G}_{\underline{n}} \rightarrow \mathbb{G}_{\underline{n}}$;
- each $\square_{\underline{n}}$ is interpreted into the empty goal in $\mathbb{G}_{\underline{n}}$.

It is easy to prove that these data satisfy the axiom of $\Gamma_{\Pi\text{-SM}}$ hence that \mathbb{G} is indeed as Π -SM-algebra.

Remark 5. It is not hard to see that every atomic formula $P(t_1, \dots, t_n)$ can be uniquely represented as the element $\sigma_t^{\mathbb{G}}(P^{\mathbb{G}})$, where σ_t is the unique substitution sending the variable x_i into the term t_i (see Remark 3).

In the same way, if we have a goal G in the form $G = A_1 \wedge \dots \wedge A_k$ with $A_i = P_i(t_1^i, \dots, t_{n_i}^i)$ we have the representation

$$G = A_1^{\mathbb{G}} \cdot_{\underline{n}}^{\mathbb{G}} (\dots (A_{k-1}^{\mathbb{G}} \cdot_{\underline{n}}^{\mathbb{G}} A_k^{\mathbb{G}}) \dots),$$

where we put $A_i^{\mathbb{G}} = \sigma_{t^i}^{\mathbb{G}}(P_i^{\mathbb{G}})$.

This representation allows us to express any goal as a canonical term in the language of $\Gamma_{\Pi\text{-SM}}$, more specifically as a term in the form

$$\underline{\sigma}_1(P_1) \cdot_{\underline{n}} (\dots \cdot_{\underline{n}} (\underline{\sigma}_{k-1}(P_{k-1}) \cdot_{\underline{n}} \underline{\sigma}_k(P_k)) \dots).$$

In particular, the atomic formulas of the form $P(x_1, \dots, x_n)$ can be expressed via the term $\text{id}_{\underline{n}}(P)$. We will exploit this representation in what follows.

The algebra of goals has an important characterization.

Theorem 1 (Initiality of \mathbb{G}). *The algebra \mathbb{G} is the initial Π -SM-algebra.*

Proof. We prove that for any other Π -SM-algebra A there is a unique homomorphism $f_A: \mathbb{G} \rightarrow A$.

We start observing that for every goal $G \in \mathbb{G}_{\underline{n}}$ in the form

$$G = \underline{\sigma}_1(P_1) \cdot_{\underline{n}} \dots \cdot_{\underline{n}} \underline{\sigma}_k(P_k)$$

every homomorphism $f: \mathbb{G} \rightarrow A$ should satisfy the following equation

$$f(G) = \underline{\sigma}_1^A(P_1^A) \cdot_{\underline{n}}^A \dots \cdot_{\underline{n}}^A \underline{\sigma}_k^A(P_k^A).$$

So the only possible homomorphism is given by the family of maps $f_{\underline{n}}^A: \mathbb{G}_{\underline{n}} \rightarrow A_{\underline{n}}$ defined by the equation

$$f_{\underline{n}}^A(\underline{\sigma}_1(P_1) \cdot_{\underline{n}} \dots \cdot_{\underline{n}} \underline{\sigma}_k(P_k)) = \underline{\sigma}_1^A(P_1^A) \cdot_{\underline{n}}^A \dots \cdot_{\underline{n}}^A \underline{\sigma}_k^A(P_k^A).$$

By calculations it is easy to prove that this indeed is an homomorphism. \square

Table 2: SOS-rules for $\Sigma_{\Pi}\text{-}\mathbf{SM}$ algebras.

$$\frac{\sigma(P) :- B \in \mathbb{P} \quad \gamma \in \mathbf{Th}(\Sigma)[m, m] \text{ is an isomorphism}}{P \xrightarrow{\gamma \circ \sigma} \underline{\gamma}(B)} \quad (\text{constant-rule})$$

where $\sigma \in \mathbf{Th}(\Sigma)[n, m]$ is a substitution, $P \in \Pi$ has arity n and $B \in (T_{\Sigma_{\Pi}\text{-}\mathbf{SM}})_m$

$$\frac{G \xrightarrow{\sigma} B \quad (\tau, \sigma, \sigma', \tau') \text{ is a pushout}}{\tau(G) \xrightarrow{\sigma'} \tau'(B)} \quad (\text{substitution-rule})$$

$$\frac{G \xrightarrow{\sigma} B}{G \cdot G' \xrightarrow{\sigma} B \cdot \sigma(G')} \quad \frac{G \xrightarrow{\sigma} B}{G' \cdot G \xrightarrow{\sigma} \sigma(G') \cdot B} \quad (\text{monoid-rule})$$

where G and G' are terms of the same type

4 Coalgebraic Semantics

In this section we introduce a structured coalgebra that provides the operational semantics for logic programs. To this aim we proceed as follows: First we provide a set of SOS rules which describe how to generate the transitions of the semantics and then using the abstract machinery of structured coalgebras, developed by Plotkin and Turi [25] (see also [9]), we show how the transitions form a structured coalgebra over the algebra of goals \mathbb{G} . In order to do that we need to prove that the axioms of $\Gamma_{\Sigma_{\Pi}\text{-}\mathbf{SM}}$ bisimulate; this is an important property that we have to prove, because our state-space is an algebra satisfying some axioms and not just a syntactic (term) algebra. Next we prove that our SOS-rules generate the transitions of the classical operational semantics for logic programs. Finally we prove the existence of a terminal coalgebra: this result allows us to use the coalgebraic-bisimulation semantics described in Section 2.

4.1 SOS rules and coalgebras

In what follows we work in the category $\mathbf{Alg}(\Sigma_{\Pi}\text{-}\mathbf{SM})$ of $\Sigma_{\Pi}\text{-}\mathbf{SM}$ -algebra, that is those algebras for the algebraic theory obtained by $\Gamma_{\Pi}\text{-}\mathbf{SM}$ dropping the axioms. Working in this larger category allows us to reuse the results of Plotkin-Turi for automatically generate endofunctors and coalgebras from an SOS specification. This machinery does not generalize well to categories of algebras satisfying axioms, hence the choice to drop the axioms.

Table 2 provides a set of SOS rules. These rules are in *De Simone format* [11], whence the following result.

Proposition 1. *The SOS-rules in Table 2 induce a functor $\mathbf{B}^{\mathbb{P}} : \mathbf{Alg}(\Sigma_{\Pi}\text{-}\mathbf{SM}) \rightarrow \mathbf{Alg}(\Sigma_{\Pi}\text{-}\mathbf{SM})$ and a coalgebra $p : T_{\Sigma_{\Pi}\text{-}\mathbf{SM}} \rightarrow \mathbf{B}^{\mathbb{P}}(T_{\Sigma_{\Pi}\text{-}\mathbf{SM}})$.*

The functor $\mathbf{B}^{\mathbb{P}}$ associates with each $\Sigma_{\Pi}\text{-SM}$ -algebra A the algebra $\mathbf{B}^{\mathbb{P}}(A)$ that interprets each sort \underline{n} with the set

$$\mathbf{B}^{\mathbb{P}}(A)_{\underline{n}} = \mathcal{P}_f \left(\left(\coprod_m \mathbf{Th}(\Sigma)[n, m] \times A_{\underline{m}} \right) \amalg A_{\underline{n}} \right)$$

whose elements are sets of substitutions' labeled transitions (i.e. pairs of the form $(\sigma, a) \in \coprod_m \mathbf{Th}(\Sigma)[n, m] \times A_{\underline{m}}$) and unlabeled (idle) transitions.

The coalgebra p associates with each term t in $T_{\Sigma_{\Pi}\text{-SM}}$, the initial $\Sigma_{\Pi}\text{-SM}$ -algebra, the set

$$p(t) = \left\{ (\sigma, t') : t \xrightarrow{\sigma} t' \text{ is a derivable sequent} \right\} \cup \{t\}.$$

The goal algebra \mathbb{G} is a $\Sigma_{\Pi}\text{-SM}$ -algebra and so it has a natural (unique) homomorphism $\pi : T_{\Sigma_{\Pi}\text{-SM}} \rightarrow \mathbb{G}$ from the initial algebra $T_{\Sigma_{\Pi}\text{-SM}}$. Since \mathbb{G} is the initial $\Gamma_{\Sigma_{\Pi}\text{-SM}}$ -algebra (as shown in Theorem 1), \mathbb{G} is obtained from $T_{\Sigma_{\Pi}\text{-SM}}$ quotienting for the axioms in $\Gamma_{\Sigma_{\Pi}\text{-SM}}$ and π is a surjective homomorphism.

Since $\pi : T_{\Sigma_{\Pi}\text{-SM}} \rightarrow \mathbb{G}$ is a surjective homomorphism, there can be at most only one $p' : \mathbb{G} \rightarrow \mathbf{B}^{\mathbb{P}}(\mathbb{G})$ such that the diagram below commutes

$$\begin{array}{ccc} T_{\Sigma_{\Pi}\text{-SM}} & \xrightarrow{\pi} & \mathbb{G} \\ p \downarrow & & \downarrow p' \\ \mathbf{B}^{\mathbb{P}}(T_{\Sigma_{\Pi}\text{-SM}}) & \xrightarrow{\mathbf{B}^{\mathbb{P}}(\pi)} & \mathbf{B}^{\mathbb{P}}(\mathbb{G}) \end{array}$$

and such p' exists if and only if the morphism $\mathbf{B}^{\mathbb{P}}(\pi) \circ p$ respects the axioms of $\Gamma_{\Sigma_{\Pi}\text{-SM}}$, that is if and only if for every equation $t = t'$ derivable from the axioms of $\Gamma_{\Sigma_{\Pi}\text{-SM}}$ the sets $\mathbf{B}^{\mathbb{P}}(p(t))$ and $\mathbf{B}^{\mathbb{P}}(p'(t'))$ are equal.

By the definitions of the functor $\mathbf{B}^{\mathbb{P}}$ and the homomorphism p , this amounts to prove that for every $\Gamma_{\Sigma_{\Pi}\text{-SM}}$ -derivable equation $t = t'$ and every $s \in T_{\Sigma_{\Pi}\text{-SM}}$ such that $t \xrightarrow{\sigma} s$ is derivable from the rules in Table 2, there is a s' such that $t' \xrightarrow{\sigma} s'$ is also derivable and $s = s'$ is an equation provable from the axioms in $\Gamma_{\Sigma_{\Pi}\text{-SM}}$, and the symmetric property holds for every transition $t' \xrightarrow{\gamma} s'$. We say that an equation $t = t'$ *bisimulates* if it satisfies this property.

Theorem 2. *Let Γ be an algebraic theory and let R be a set of SOS rules defined over the signature of Γ . If every closed instance of the axioms in Γ bisimulates, then every Γ -derivable equation bisimulates as well.*

Proof. Since every Γ -derivable closed equation can be derived by reflexivity, symmetry, transitivity, and congruence by the axioms, we can prove the thesis by induction on these inference rules.

The proof is straightforward for all the rules with the exception of *congruence*, which is the rule requiring that SOS-rules are in *De Simone Format* [11], that is why we will focus only on this case.

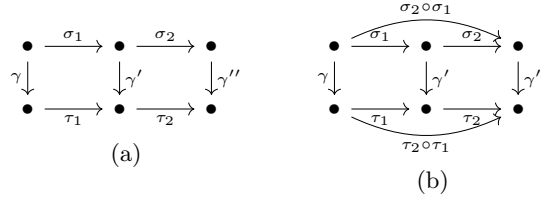


Fig. 2: Pushout (de)composition

We recall that a rule is in *De Simone Format* if it is in the format

$$\frac{\{x_i \xrightarrow{\gamma_i} y_i\}_{i \in I}}{o(x_1, \dots, x_n) \xrightarrow{\gamma} g(y_1, \dots, y_n)} \quad (1)$$

where o is an operation symbol of the signature, $g(y_1, \dots, y_n)$ is a term containing only the variables y_i 's, and the variables x_i 's and y_i 's are all distinct with the exception of the pairs (x_i, y_i) for the $i \notin I$, in this case we have $x_i \equiv y_i$.

Let assume we have a family of derivable equations $t_i = s_i$ that bisimulate. We want to prove that the equation $o(t_1, \dots, t_n) = o(s_1, \dots, s_n)$, derived by congruence-rule, bisimulates too.

Let $o(t_1, \dots, t_n) \xrightarrow{\gamma} u$ be a transition derived by the SOS-rules. We may assume that such transition is derived by a rule as the one in formula (1), so there must exist a family of terms u_i such that we have derivable transitions of the form $t_i \xrightarrow{\gamma_i} u_i$, for the $i \in I$, $t_i \equiv u_i$ for the $i \notin I$, and with $u \equiv g(u_1, \dots, u_n)$.

By the hypothesis that the $t_i = s_i$ bisimulate we can conclude that for every index $i \in I$ there must be a \bar{u}_i such that $s_i \xrightarrow{\gamma_i} \bar{u}_i$, and by letting $\bar{u}_i \equiv s_i$ for $i \notin I$, applying the De Simone rule, we get a transition $o(s_1, \dots, s_n) \xrightarrow{\gamma} g(\bar{u}_1, \dots, \bar{u}_n)$.

By theorems of equational logic, since the equalities $u_i = \bar{u}_i$ are derivable, it follows that the equality

$$u \equiv g(u_1, \dots, u_n) = g(\bar{u}_1, \dots, \bar{u}_n)$$

is derivable. So by letting $\bar{u} \equiv g(\bar{u}_1, \dots, \bar{u}_n)$ we have found that for the transition $o(t_1, \dots, t_n) \xrightarrow{\gamma} u$ there is a transition $o(s_1, \dots, s_n) \xrightarrow{\gamma} \bar{u}$ with $u = \bar{u}$ derivable.

By a symmetric argument we can prove the bisimulation property for transitions of the form $o(s_1, \dots, s_n) \xrightarrow{\gamma} \bar{u}$.

As stated in the beginning of the proof, by induction on the inference rules of equational logic it follows that every derivable equation bisimulates. \square

Proposition 2 (Axioms in $\Gamma_{\Sigma_{\Pi}\text{-SM}}$ bisimulate). *The axioms of $\Gamma_{\Sigma_{\Pi}\text{-SM}}$ bisimulate with respect to the SOS rules of Table 2.*

Proof. For each instance of the axioms in $\Gamma_{\Sigma_{II} - \text{SM}}$ one can prove by induction on the inference rules (of the SOS specification in Table 2) the thesis. As an interesting case we consider the substitution axiom $(\sigma_2(\sigma_1(t)) = (\sigma_2 \circ \sigma_1)(t))$.

First, we want to prove that for every closed term t and for every transition $\sigma_2(\sigma_1(t)) \xrightarrow{\gamma''} s''$ there is a transition $(\sigma_2 \circ \sigma_1)(t) \xrightarrow{\gamma''} \bar{s}''$ such that $s'' = \bar{s}''$ is a $\Gamma_{\Sigma_{II} - \text{SM}}$ -derivable equation. The only rule in Table 2 that produces a transition for terms of the form $\sigma_2(\sigma_1(t))$ is the substitution-rule, hence we may assume that there is a transition $\sigma_1(t) \xrightarrow{\gamma'} s'$ and a pushout square $(\gamma', \sigma_2, \tau_2, \gamma'')$ such that $s'' \equiv \tau_2(s')$. For the same reason we may assume that there is a transition $t \xrightarrow{\gamma} s$ and a pushout square $(\gamma, \sigma_1, \tau_1, \gamma')$ such that $s' \equiv \tau_1(s)$. We have that, for the diagram in Fig. 2a the two internal squares are pushouts, hence the external rectangle $(\gamma, \sigma_2 \circ \sigma_1, \tau_2 \circ \tau_1, \gamma'')$ is a pushout as well. By the substitution rule we have the transition $(\sigma_2 \circ \sigma_1)(t) \xrightarrow{\gamma''} (\tau_2 \circ \tau_1)(s)$, and by the substitution axiom $(\tau_2 \circ \tau_1)(s) = \tau_2(\tau_1(s))$, so letting $\bar{s}'' \equiv (\tau_2 \circ \tau_1)(s)$ we get our claim.

Second, for any transition $(\sigma_2 \circ \sigma_1)(t) \xrightarrow{\gamma''} s''$ we want to prove that there is a transition $\sigma_2(\sigma_1(t)) \xrightarrow{\gamma''} \bar{s}''$ such that $s'' = \bar{s}''$ is a derivable equation. Indeed every transition for a term of the form $(\sigma_2 \circ \sigma_1)(t)$ can be only derived by the substitution rule, hence we may assume that there is a transition $t \xrightarrow{\gamma} s$ and a pushout square $(\gamma, \sigma_2 \circ \sigma_1, \gamma'', \tau)$ such that $s'' \equiv \tau(s)$. It is well known that the pushout square can be decomposed in two pushouts as shown in Fig. 2b, where $\tau \equiv \tau_2 \circ \tau_1$. It follows that $\sigma_1(t) \xrightarrow{\gamma'} \tau_1(s)$ and $\sigma_2(\sigma_1(t)) \xrightarrow{\gamma''} \tau_2(\tau_1(s))$. By the substitution axiom we also have $\tau_2(\tau_1(s)) = (\tau_2 \circ \tau_1)(s)$. \square

Combining Theorem 2 with Proposition 2 we get:

Corollary 1 (Equalities in $\Gamma_{\Sigma_{II} - \text{SM}}$ bisimulate). *All $\Gamma_{\Sigma_{II} - \text{SM}}$ -derivable equations bisimulate.*

From the above discussion it follows that:

Theorem 3 (Coalgebraic semantics of \mathbb{G}). *There is a unique $\mathbf{B}^{\mathbb{P}}$ -coalgebra $p': \mathbb{G} \rightarrow \mathbf{B}^{\mathbb{P}}(\mathbb{G})$ that makes the diagram below commute:*

$$\begin{array}{ccc} T_{\Sigma_{II} - \text{SM}} & \xrightarrow{\pi} & \mathbb{G} \\ p \downarrow & & \downarrow p' \\ \mathbf{B}^{\mathbb{P}}(T_{\Sigma_{II} - \text{SM}}) & \xrightarrow{\mathbf{B}^{\mathbb{P}}(\pi)} & \mathbf{B}^{\mathbb{P}}(\mathbb{G}) \end{array}$$

Remark 6. The coalgebra $p': \mathbb{G} \rightarrow \mathbf{B}^{\mathbb{P}}(\mathbb{G})$ associates with every goal g those transitions (σ, g') such that $g \xrightarrow{\sigma} g'$ is an instance of a derivable sequent of our SOS rules in the algebra \mathbb{G} .

The following theorem establishes the relation between our coalgebra and the operational semantics of logic programs, the SLD-reduction.

Theorem 4 (SLD-reduction as a coalgebra). *The LTS underlying the coalgebra $p': \mathbb{G} \rightarrow \mathbf{B}^{\mathbb{P}}(\mathbb{G})$ is the same generated by the rules for the SLD derivation.*

Proof. The LTS generated by the SLD-reduction is the one generated by the SOS-rules obtained via the application of the syntactic transformation that turns every sequent of the form $\mathbb{P} \models G \Rightarrow_{\sigma} G'$ into a sequent of the form $G \xrightarrow{\sigma} G'$.

It can be shown that these SOS-rules can be derived by the SOS-rules of Table 2 and vice versa. \square

4.2 Final $\mathbf{B}^{\mathbb{P}}$ -coalgebra

We conclude this section with a proof of the existence of the terminal $\mathbf{B}^{\mathbb{P}}$ -coalgebra. This allows to use the bisimulation semantics via terminal-coalgebra as described in Section 2. We need the following result.

Theorem 5. *For every locally presentable category \mathbf{C} and every accessible endofunctor $B: \mathbf{C} \rightarrow \mathbf{C}$ the forgetful functor $V_B: \mathbf{Coalg}(B) \rightarrow \mathbf{C}$ is a left adjoint.*

Proof. The proof is basically the same as the one presented in [3, Theorem 1.2]. It follows from the *Special Adjoint Functor Theorem* using the fact that \mathbf{C} and $\mathbf{Coalg}(B)$ are locally small and locally presentable, hence cocomplete and with a generating set, that $\mathbf{Coalg}(B)$ is cocomplete, for [1, Theorem 1.58], and that the forgetful functor V_B preserves colimits. \square

Corollary 2. *With the hypothesis of Theorem 5, if \mathbf{C} has a terminal object, $\mathbf{Coalg}(B)$ has a terminal object as well.*

Proof. If $F_B: \mathbf{C} \rightarrow \mathbf{Coalg}(B)$ is the right adjoint of V_B , which exists for Theorem 5, it preserves limits and so, letting $T \in \mathbf{C}$ be a terminal object, $F_B(T)$ is a terminal object too. \square

Theorem 6 ($\mathbf{B}^{\mathbb{P}}$ is accessible). *The functor $\mathbf{B}^{\mathbb{P}}$ is accessible.*

Proof. Since $\mathbf{Alg}(\Sigma_{\Pi}\text{-}\mathbf{SM})$ is an algebraic category $\mathbf{B}^{\mathbb{P}}$ is accessible if and only if for every sort $\underline{n} \in S_{\Sigma_{\Pi}\text{-}\mathbf{SM}}$ the functor $\mathbf{B}_{\underline{n}}^{\mathbb{P}}: \mathbf{Alg}(\Sigma_{\Pi}\text{-}\mathbf{SM}) \rightarrow \mathbf{Set}$ such that $\mathbf{B}_{\underline{n}}^{\mathbb{P}}(A) = \mathbf{B}^{\mathbb{P}}(A)_{\underline{n}}$ is accessible. Remember that

$$\mathbf{B}_{\underline{n}}^{\mathbb{P}}(A) = \mathcal{P}_f \left(\left(\coprod_m \mathbf{Th}(\Sigma)[n, m] \times A_m \right) \amalg A_{\underline{n}} \right).$$

This equation shows that $\mathbf{B}_{\underline{n}}^{\mathbb{P}}$ is obtained combining the valuation functors (those that send algebras in their carriers), that are accessible, with accessible functors, namely the finite powerset \mathcal{P}_f and the multiplication functors $\mathbf{Th}(\Sigma)[n, m] \times -$. Since these functors are composed via coproduct and functor composition, that preserve accessibility, it follows that $\mathbf{B}_{\underline{n}}^{\mathbb{P}}$ is accessible as well.

Since this holds for every $\underline{n} \in S_{\Sigma_{\Pi}\text{-}\mathbf{SM}}$, by the above mentioned property of algebraic categories, it follows that $\mathbf{B}^{\mathbb{P}}$ is accessible. \square

By Theorem 6 with Corollary 2 it follows that

Proposition 3 (Finality). *The category $\mathbf{Coalg}(\mathbf{B}^{\mathbb{P}})$ has a terminal object.*

4.3 Examples

We conclude this section by showing some examples of (in)equivalence.

Example 1. Let us consider the logic program:

$$\begin{array}{lll} P(x, y, z) :- Q(x, y), R(y, z) & S(x, y, z) :- T(x, y, z) & \\ Q(a, c) :- \square & T(a, c, z) :- V(z) & V(b) :- \square \\ R(c, b) :- \square & T(x, c, b) :- U(x) & U(a) :- \square \end{array}$$

The goals $P(x, y, z)$ and $S(x, y, z)$ are bisimilar: they yield isomorphic LTSs.

Example 2. Let us consider the logic program:

$$P(f(x)) :- P(x) \quad Q(f(x)) :- R(x) \quad R(f(x)) :- Q(x)$$

The goals $P(x)$, $Q(x)$ and $R(x)$ are all bisimilar. They are logic *perpetual* processes [21]: even with the impossibility to terminate, at each transition new substitutions of the form $f(y)/z$ are computed to approximate the result.

Example 3. Let us consider the logic program:

$$\begin{array}{llll} P(a, y) :- Q(y) & Q(b) :- \square & S(a, y) :- T(y) & T(b) :- \square \\ P(a, y) :- R(y) & R(c) :- \square & & T(c) :- \square \end{array}$$

The goals $P(x, y)$ and $S(x, y)$ have the same answer substitutions but are not bisimilar, because the choice to substitute b or c for y is done by P implicitly at the first step while it is postponed to the second transition by S .

5 Conclusion

When exploiting LP as a process description language defined by an LTS, it is natural to look for a structured coalgebraic semantics.

In the paper, states, i.e. goals, are represented as the substitutive monoids freely generated by predicate symbols, and transitions are goal reductions via unification. The construction guarantees the existence of a final coalgebra yielding the abstract semantics. More precisely, coalgebras live in the category of algebras equipped with the operations of substitutive monoids, but without their axioms. Thus goal bisimulations respect monoidal and substitution operations.

In [4] the authors introduced a structured coalgebra that models the operational semantics of logic programming in order to apply the theory of reactive systems [20]. Their coalgebra uses a presheaf to model the state-space, which can be viewed as a multisorted-algebra having only unary-typed operations indexed by a family of substitutions. We have chosen a different algebraic structure that seems closer to the natural structure of goals of logic programming. For instance, to define the coalgebraic endofunctor we refer to an original, suggestive SOS specification quite close to the rules for SLD derivation (see Table 1). Nevertheless the two approaches are related. It is possible to obtain two forgetful

functors from the two categories of algebras in the category $\mathbb{N}\text{-}\mathbf{Set}$, of families of sets indexed by natural numbers, and an endofunctor B over $\mathbb{N}\text{-}\mathbf{Set}$ such that the forgetful functors send the coalgebras into the same B -coalgebra in $\mathbb{N}\text{-}\mathbf{Set}$, up to some technicalities due to differences into the behavioural endofunctors used for the structured coalgebras. Thus the two constructions can be considered as consisting of two different enrichments applied to the same coalgebra in \mathbf{Set} .

Our construction could be adapted to simulate other models of computation which are structurally similar to LP, importing the well known properties of coalgebras regarding congruence, logic semantics and higher order. For instance, in [19] two process calculi, Fusion Calculus (a variant of pi-calculus) and Synchronized Hyperedge Replacement with Hoare Synchronization (a graph rewriting calculus with synchronization and mobility) are mapped into LP. Implementation efficiency can also get involved: constraint problems may be described by networks of constraints with an algebraic specification similar to our substitutive monoids; interestingly, an additional operation of restriction [24] allows to equip the networks with a hierarchical structure, which allows often to decompose the constraint problem and to solve it by means of an efficient dynamic programming algorithm. Moreover, in a Datalog-style setting [23], the decomposition process is automatically suggested by the goal reductions themselves.

We plan to study the relations between these models of computation taking advantage of the well known expressive power of the categorical approach.

In the present work we modeled the coalgebra in the category of algebras for a given signature—without axioms—to use the machinery of [25]. In [6] the authors provide some tools to generate behavioural endofunctors over categories of algebras *with* axioms. It could be interesting to investigate how to use these tools to turn our bi-algebraic semantics in a coalgebra over the category of substitutive monoids, instead of the wider category of algebras that are not required to respect the axioms of $\Gamma_{\Sigma_{\Pi}\text{-}\mathbf{SM}}$. However, one of the advantages of the current presentation is to be self-contained and more concrete than the framework in [6], so to be possibly accessible to a wider audience.

Acknowledgement. We thank Andrea Corradini who read a preliminary version of this paper and helped us to improve the presentation. We also thank the anonymous referees for their useful remarks and pointers to the literature.

References

1. J. Adamek and J. Rosicky. *Locally Presentable and Accessible Categories*. London Mathematical Society Lecture Note Series. Cambridge University Press, 1994.
2. G. Amato, J. Lipton, and R. McGrail. On the algebraic structure of declarative programming languages. *Theor. Comp. Sci.*, 410(46):4626–4671, 2009.
3. M. Barr. Terminal coalgebras in well-founded set theory. *Theor. Comp. Sci.*, 114(2):299–315, June 1993.
4. F. Bonchi and U. Montanari. Reactive systems, (semi-)saturated semantics and coalgebras on presheaves. *Theoretical Computer Science*, 410(41):4044 – 4066, 2009. Festschrift for Mogens Nielsen’s 60th birthday.

5. F. Bonchi and F. Zanasi. Bialgebraic semantics for logic programming. *Logical Methods in Computer Science*, 11(1), 2015.
6. M. M. Bonsangue, H. H. Hansen, A. Kurz, and J. Rot. Presenting distributive laws. In R. Heckel and S. Milius, editors, *Algebra and Coalgebra in Computer Science*, pages 95–109, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
7. R. Bruni, U. Montanari, and F. Rossi. An interactive semantics of logic programming. *Th. and Pract. of Logic Prog.*, 1(6):647–690, 2001.
8. A. Corradini and A. Asperti. A categorical model for logic programs: Indexed monoidal categories. In *Semantics: Foundations and Applications, REX Workshop*, volume 666 of *LNCS*, pages 110–137. Springer, 1992.
9. A. Corradini, R. Heckel, and U. Montanari. From SOS specifications to structured coalgebras: How to make bisimulation a congruence. *ENTCS*, 19:118–141, 1999.
10. A. Corradini and U. Montanari. An algebraic semantics for structured transition systems and its applications to logic programs. *Theor. Comp. Sci.*, 103(1):51–106, 1992.
11. R. de Simone. Higher-level synchronising devices in meije-sccs. *Theor. Comput. Sci.*, 37:245–267, 1985.
12. M. Falaschi, G. Levi, C. Palamidessi, and M. Martelli. Declarative modeling of the operational behavior of logic languages. *Theor. Comp. Sci.*, 69(3):289–318, 1989.
13. S. E. Finkelstein, P. J. Freyd, and J. Lipton. Logic programming in tau categories. In *CSL’94*, volume 933 of *LNCS*, pages 249–263. Springer, 1994.
14. J. Gray. The category of sketches as a model for algebraic semantics. In *Categories in Computer Science and Logic*, volume 92 of *Contemp. Math.* AMS, 1989.
15. E. Komendantskaya and J. Power. Coalgebraic semantics for derivations in logic programming. In *CALCO’11*, volume 6859 of *LNCS*, pages 268–282. Springer, 2011.
16. E. Komendantskaya and J. Power. Logic programming: Laxness and saturation. *J. Log. Algebr. Meth. Program.*, 101:1–21, 2018.
17. E. Komendantskaya, J. Power, and M. Schmidt. Coalgebraic logic programming: from semantics to implementation. *J. Log. Comput.*, 26(2):745–783, 2016.
18. R. A. Kowalski. Algorithm = logic + control. *Comm. ACM*, 22(7):424–436, 1979.
19. I. Lanese and U. Montanari. Mapping fusion and synchronized hyperedge replacement into logic programming. *Th. and Pract. of Logic Prog.*, 7(1-2):123–151, 2007.
20. J. J. Leifer and R. Milner. Deriving bisimulation congruences for reactive systems. In C. Palamidessi, editor, *CONCUR’00*, volume 1877 of *LNCS*, pages 243–258. Springer, 2000.
21. G. Levi and C. Palamidessi. Contributions to the semantics of logic perpetual processes. *Acta Inf.*, 25(6):691–711, 1988.
22. J. W. Lloyd. *Foundations of Logic Programming, 2nd Edition*. Springer, 1987.
23. U. Montanari and F. Rossi. Perfect relaxation in constraint logic programming. In *ICLP’91*, pages 223–237. MIT Press, 1991.
24. U. Montanari, M. Sammartino, and A. T. Siwe. Decomposition structures for soft constraint evaluation problems: An algebraic approach. In *Graph Transformation, Specifications, and Nets*, volume 10800 of *LNCS*, pages 179–200. Springer, 2018.
25. D. Turi and G. D. Plotkin. Towards a mathematical operational semantics. In *LICS’97*, pages 280–291. IEEE Computer Society, 1997.