An Algebraic Theory of Observables

Marco Comini and **Giorgio Levi**
Dipartimento di Informatica
Università di Pisa
Corso Italia 40, 56125 Pisa, Italy
{comini, levi}@di.unipi.it

Abstract

We give an algebraic formalization of *SLD*-trees and their abstractions (ob servables). We can state and prove in the framework several useful theorems (*AND*-compositionality, correctness and full abstraction of the denotation, equivalent top-down and bottom-up constructions) about semantic properties of various observables. Observables are represented by Galois co-insertions and can be used to model abstract interpretation. The constructions and the theorems are inherited by all the observables which can be formalized in the framework. The power of the framework is shown by reconstructing some known examples (answer constraints, call patterns, correct call patterns and ground dependencies call patterns).

1 Introduction

SLD-trees are structures used to describe the operational semantics of logic programs. From an *SLD*-tree we can derive several operational properties which are useful for reasoning about programs. Examples are *SLD*-derivations, resultants, call patterns, partial answers, computed answers. All these properties, that we call *observables*, can be obtained as abstractions of the *SLD*-tree. Let for example T be an *SLD*-tree with initial goal g_0. Then, an *SLD*-derivation is any path of T; a call pattern is any atom selected in T; a partial answer is the substitution associated to any node n in T (restricted to g_0), while a computed answer is the substitution associated to any node n, such that g_n is the empty clause.

The behavior of a program p (via selection rule r) with respect to a given observable can be understood by observing the corresponding properties for all possible goals. We know from recent results on the semantics [11, 15, 2] that we can characterize this behavior just by observing the property for some specific atomic goals, namely the "most general" atomic goals. This result was first obtained for computed answers in the *s*-semantics framework [9, 10] and then proved for other less abstract observables, such as resultants, call patterns and partial answers in [11, 15]. The behaviors for most general atomic goals can then be considered a program denotation. In [15] this approach is used to define a semantic framework, where one can define denotations modeling various observables, by inheriting from the framework

the basic constructions and theorems. Some of the denotations enjoy additional properties, such as full abstraction. The technical tool used to define the abstractions is the definition of suitable equivalence relations.

We approach here the same problem with a different emphasis and a different technical tool. Our main objective is in fact the formalization of the observable, while our abstractions will be based on abstract interpretation techniques [8]. This will allow us to model, within the same framework, the approximation which is involved in the abstractions used for program analysis. A similar approach can be found in [18]. Our main results are the definition of an algebraic framework for reasoning about *SLD*-trees and their abstractions (observables) in the case of Constraint Logic Programs. The framework is provided with several general theorems (*AND*-composition ality, correctness and full abstraction, equivalent top-down and bottom-up constructions), which are valid for any abstraction, possibly in a weaker form in the case of abstract interpretation. We give the reconstruction of several existing constructions to show the expressive power of the framework.

The paper is organized as follows. Section 2 characterizes the general theory in the case of *SLD*-trees. Section 3 formalizes the main class of ob servables, i.e. the $\mathcal{S}$-observables for which all the general results are valid. Finally section 4 considers the $\mathcal{I}$-observables, where the results are weaker yet meaningful from the abstract interpretation theory viewpoint. All the proofs can be found in [6].

For a comprehensive description of the semantics of (positive) logic programs see [21, 1]. Σ, Π and V denote a set of function symbols, a set of predicate symbols and a denumerable set of variables respectively. Tuples of variables and terms are sometimes denoted by $\tilde{x}, \tilde{y}, \ldots$ and $\tilde{t}, \tilde{s}, \ldots$. We denote by $\tilde{t}$ both the tuple and the set of corresponding syntactic objects. $\tilde{x}_i$ denotes $\tilde{x}_{i1}, \ldots, \tilde{x}_{in_i}$. $\tilde{b}$ denotes a (possibly empty) conjunction of atoms $b_1, \ldots, b_n$ and $\tilde{b}, \tilde{b'}$ denotes the conjunction $b_1, \ldots, b_m, b'_1, \ldots, b'_n$. For a comprehensive description of abstract interpretation see [8]. For a comprehensive description of term systems, closed semirings and constraints systems see [17]. In this paper we consider Constraint Logic Programs over a generic constraint system $\mathcal{A}$ ($CLP(\mathcal{A})$).

2 The basic framework

When we want to formalize program execution we must take into account, in addition to the inference rules which specify how derivations are made, the properties we observe in a computation (*observables*). An observable is any property which can be extracted from a goal computation, i.e. *observables are abstractions of SLD-trees*. We assume the reader to be familiar with the notions of *SLD*-resolution and *SLD*-tree (see [21, 1]). We represent here, for notational convenience, *SLD*-trees as sets of nodes.

Definition 2.1 (well-formed sets of nodes) *Let* T *be an SLD-tree rooted at the goal* g. *A* node *in* T *is a triple* $\langle \mathsf{g}; \mathsf{c} \square \tilde{\mathsf{b}}; \mathsf{ks} \rangle$, *where* $\mathsf{c} \square \tilde{\mathsf{b}}$ *is the goal associated to the node (*c *is the accumulated constraint and* $\tilde{\mathsf{b}}$ *is a conjunction of atoms), and* ks *is the sequence of (renamed apart) clauses used in the derivation of* $\mathsf{c} \square \tilde{\mathsf{b}}$ *from* g. $Nod_{\mathsf{p}}^{\mathsf{r},\mathsf{g}}$ *is the set of all the nodes of* g *in* p *via* r. *We define a partial order* $\leq$ *on nodes:* $\langle \mathsf{g}; \mathsf{c} \square \tilde{\mathsf{b}}; \mathsf{ks} \rangle \leq \langle \mathsf{g}; \mathsf{c}' \square \tilde{\mathsf{b}}'; \mathsf{ks}' \rangle$ *iff* $\mathsf{ks}' = \mathsf{ks} :: \mathsf{ks}''$ *for some* ks''.

A set A *of nodes is* well-formed *if* $\rho \in \mathsf{A}$ *implies* $\forall \rho' \leq \rho,\ \rho' \in \mathsf{A}$.

We define the domain R of all the well-formed sets of nodes partially ordered by set inclusion. This partial order formalizes the evolution of the computation process. It is easy to prove that $(\mathsf{R}, \subseteq)$ is a complete lattice.

2.1 The observables

An observable property domain is a set of properties of the derivation with an ordering relation which can be viewed as an approximation structure. An observation consists in looking at an *SLD*-tree, and then extracting some property (abstraction). An *SLD*-tree is represented as a well formed set of nodes. R is the domain of all the well formed sets of nodes. Therefore the observable is a function from R to a suitable property domain D, which preserves the approximation structure. Such a function must be a Galois connection.

Definition 2.2 (observable) *Let* R *be the domain of SLD-trees and* D *be an observable domain.* $\alpha : \mathsf{R} \to \mathsf{D}$ *is an* observable *when there exists* γ *s.t.* $(\alpha, \gamma) : \mathsf{R} \to \mathsf{D}$ *is a Galois co-insertion.*

We denote by the same symbol an observable and the Galois connection it can be extended to.

Since we are interested in all the *SLD*-trees *of a program* p, we define the *behavior* of p as $\mathsf{B}^{\mathsf{r}}(\mathsf{p}) = \cup_{\mathsf{g} \in \mathsf{G}} Nod_{\mathsf{p}}^{\mathsf{r},\mathsf{g}} \in \mathsf{R}$, i.e. the set of all the nodes of *SLD*-trees of g in p, for any goal g. The *abstract behavior* $\mathsf{B}_{\alpha}^{\mathsf{r}}(\mathsf{p})$ w.r.t. the observable α is simply defined as $\alpha(\mathsf{B}^{\mathsf{r}}(\mathsf{p}))$. α induces an *observational equivalence* $=_{\alpha}$ on programs. Namely $\mathsf{p}_1 =_{\alpha} \mathsf{p}_2$ iff $\alpha(\mathsf{B}^{\mathsf{r}}(\mathsf{p}_1)) = \alpha(\mathsf{B}^{\mathsf{r}}(\mathsf{p}_2))$, i.e. if p_1 and p_2 cannot be distinguished by looking at their abstract behaviors. Observational equivalences can be used to define a partial order $\leq$ on observables. Namely $\alpha' \leq \alpha$ (α is stronger than α') if $\mathsf{p}_1 =_{\alpha} \mathsf{p}_2$ implies $\mathsf{p}_1 =_{\alpha'} \mathsf{p}_2$. This in turn means that there exists a Galois connection between the domains of the observables. Hence an observable $\alpha : \mathsf{R} \to \mathsf{D}$ *approximates* $\alpha' : \mathsf{R} \to \mathsf{D}'$ if there exists a co-insertion $\beta : \mathsf{D} \to \mathsf{D}'$ s.t. $\alpha' = \beta \circ \alpha$. Let $(\mathbb{O}, \leq)$ be the set of observables ordered by approximation. It follows from [8] that $(\mathbb{O}, \leq)$ is a complete lattice.

Example 2.3 If ξ denotes *computed answer constraints* we can take $\mathsf{D} = (\mathcal{P}(\mathsf{G} \times \mathcal{A}), \subseteq)$ as properties domain and extend to a connection the function

$\xi : R \to D$, $\xi(A) = \{ (g; c) \mid \langle g; c \,\square; ks\rangle \in A \}$. It is easy to see that $p_1 =_\xi p_2$ iff for any goal g, g has the same answer constraints in p_1 and in p_2. ■

2.2 Semantic properties of *SLD*-trees

The goal we want to achieve is to develop a denotation modeling *SLD*-trees. We follow the approach in [14, 2], by first defining a "syntactic" semantic domain (π-interpretation). Our modeling of *SLD*-trees is essentially the basic denotation defined in terms of clauses in [11, 15], extended to handle constraint systems in the style of [17, 19, 20]. In the following for the sake of simplicity we consider the PROLOG leftmost selection rule (denoted by *lm*). All our results can be generalized to local selection rules [22].

Let us consider the equivalence relation of variance extended to nodes $\equiv$.

Definition 2.4 *A π-interpretation $\mathcal{I}$ is an element of $R_{/\equiv}$. We denote by $\mathcal{R}$ the set of π-interpretations. $(\mathcal{R}, \subseteq)$ is a complete lattice.*

A denotation of the program characterizing its *SLD*-trees computed by using the rule lm might be the set of the *SLD*-trees for all the possible goals modulo variance, i.e. $B^{lm}(p)_{/\equiv}$. Because of the *AND*-compositionality theorem 2.6 below, this set can be obtained from the top-down *SLD*-trees denotation, which is the set of *SLD*-trees for the most general atomic goals.

Definition 2.5 (top-down *SLD*-trees denotation) *Let p be a program. The* top-down *SLD*-trees denotation *of p according to lm is the π-interpre tation*

$$\mathcal{O}(p) = \{ \langle q(\tilde{x}); c \,\square\, \tilde{b}; ks\rangle \in Nod_p^{lm} \mid q \in \Pi_n,\ \tilde{x} \in V^n \}_{/\equiv}.$$

It is easy to see that $\mathcal{O}(p)$ is well-formed. Now we prove that this denotation fully characterizes all the *SLD*-trees of p. This is obtained by first proving a lemma which relates the *SLD*-trees of an atomic goal to the *SLD*-trees of the corresponding most general atomic goal. The second step, i.e. the *AND*-compositionality theorem, relates the *SLD*-trees of a conjunctive goal to the *SLD*-trees of the atomic goals.

Theorem 2.6 (*AND*-compositionality) *Let $g = c_g \,\square\, q_1(\tilde{t}_1), \ldots, q_n(\tilde{t}_n)$ be a goal and p be a program. Then $g \overset{ks}{\rightsquigarrow} c \,\square\, \tilde{b}$ if and only if $\exists\, r_j = \langle q_j(\tilde{x}_j); c_j \,\square; ks_j\rangle \in \mathcal{O}(p)$, $1 \le j < m$, $\exists\, r_m = \langle q_m(\tilde{x}_m); c_m \,\square\, \tilde{b}_m; ks_m\rangle \in \mathcal{O}(p)$ s.t. $c = \exists \left(c_g \otimes \tilde{x}_1{=}\tilde{t}_1 \otimes \ldots \otimes \tilde{x}_m{=}\tilde{t}_m \otimes c_1 \otimes \ldots \otimes c_m\right)_{g,\tilde{b}}$, $\tilde{b} = \tilde{b}_m, q_{m+1}(\tilde{t}_{m+1}), \ldots, q_n(\tilde{t}_n)$ and $ks = ks_1 :: \ldots :: ks_m$.*

The above closure property allows us to show that the semantics $\mathcal{O}(p)$ is correct and fully abstract for the identity observable.

Corollary 2.7 (correctness and full abstraction) *Let p_1, p_2 be two programs. Then $p_1 =_{id} p_2 \iff \mathcal{O}(p_1) = \mathcal{O}(p_2)$.*

The restriction to local rules plays a fundamental role in the definition of the bottom-up denotation. By using local rules we are able to reconstruct a derivation "from the bottom", because the local rule chooses only among the atoms introduced in the last derivation step and then "forgets" about the previous steps, which, in a bottom-up construction, are not available yet. This is the definition of the immediate consequences operator for the *leftmost* case.

Definition 2.8 (immediate consequences operator T_p) *Let $\mathcal{I}$ be a π-interpretation and p be a program. The* immediate consequences operator $\mathsf{T}_\mathsf{p} : \mathcal{R} \to \mathcal{R}$ *of p via lm is:*

$$\begin{aligned}\mathsf{T}_\mathsf{p}(\mathcal{I}) = \{\, \langle \mathsf{q}(\tilde{\mathsf{x}}); \mathsf{c} \,\square\, \tilde{\mathsf{b}}; \mathsf{ks}\rangle \mid\ & \\ & \mathsf{ks} = [\mathsf{k}] :: \mathsf{ks}_1 :: \ldots :: \mathsf{ks}_m,\ \tilde{\mathsf{b}} = \tilde{\mathsf{b}}_m, \mathsf{a}_{m+1}, \ldots, \mathsf{a}_n, \\ & \mathsf{k} = \mathsf{q}(\tilde{\mathsf{t}}) :\!-\ \mathsf{c}_\mathsf{k} \,\square\, \mathsf{q}_1(\tilde{\mathsf{t}}_1), \ldots, \mathsf{q}_m(\tilde{\mathsf{t}}_m), \mathsf{a}_{m+1}, \ldots, \mathsf{a}_n \in \mathsf{p}, \\ & \langle \mathsf{q}_m(\tilde{\mathsf{x}}_m); \mathsf{c}_m \,\square\, \tilde{\mathsf{b}}_m; \mathsf{ks}_m\rangle \in \mathcal{I},\ \langle \mathsf{q}_j(\tilde{\mathsf{x}}_j); \mathsf{c}_j \,\square\, ; \mathsf{ks}_j\rangle \in \mathcal{I},\ 1 \le j < m, \\ & \mathsf{c} = \exists \left(\tilde{\mathsf{x}}{=}\tilde{\mathsf{t}} \otimes \mathsf{c}_\mathsf{k} \otimes \tilde{\mathsf{x}}_1{=}\tilde{\mathsf{t}}_1 \otimes \mathsf{c}_1 \otimes \ldots \otimes \tilde{\mathsf{x}}_m{=}\tilde{\mathsf{t}}_m \otimes \mathsf{c}_m\right)_{\tilde{\mathsf{x}},\tilde{\mathsf{b}}} \,\}.\end{aligned}$$

Since T_p is continuous we can define the *fixpoint denotation* of the program p according to lm as the π-interpretation $\mathcal{F}(\mathsf{p}) = \mathsf{T}_\mathsf{p} \uparrow \omega$. The following theorem states the equivalence between the top-down and the bottom-up constructions, and shows that $\mathcal{F}(\mathsf{p})$ is also correct and fully-abstract w.r.t. the identity observable.

Theorem 2.9 *Let p be a program. Then $\mathcal{F}(\mathsf{p}) = \mathcal{O}(\mathsf{p})$.*

2.3 An algebraic formalization of *SLD*-trees semantic properties

The properties we found for $\mathcal{O}$ and $\mathcal{F}$ allow us to claim that we have a good denotation modeling *SLD*-trees. Our goal however is to find the same results for the denotations modeling more abstract observables. We want then to develop a theory according to which the semantic properties of *SLD*-trees shown in subsection 2.2 are inherited by the denotations which model abstractions of the *SLD*-trees.

In order to define the denotation as a function of the observable, we need a mathematical formalization where one can model the abstraction process and specify properties which have to be shared by the constructions associated to the various abstractions. The first interesting property is the lifting one, which can be modeled only if we can instantiate variables in the derivation by means of constraints. Thus we can define an operation $\cdot$ which adds a constraint to a denotation:

$$\mathsf{c} \cdot \mathsf{A} = \{\, \langle \mathsf{c} \otimes \mathsf{c}' \,\square\, \tilde{\mathsf{b}}'; \mathsf{c} \otimes \mathsf{c}'' \,\square\, \tilde{\mathsf{b}}''; \mathsf{ks}\rangle \mid \langle \mathsf{c}' \,\square\, \tilde{\mathsf{b}}'; \mathsf{c}'' \,\square\, \tilde{\mathsf{b}}''; \mathsf{ks}\rangle \in \mathsf{A} \,\}$$

This operation is related to $\otimes$ by the following property: $\forall \nu \in \mathcal{R}$ and $\mathsf{c}_1, \mathsf{c}_2 \in \mathcal{A}$, $(\mathsf{c}_1 \otimes \mathsf{c}_2) \cdot \nu = \mathsf{c}_1 \cdot (\mathsf{c}_2 \cdot \nu)$.

The next relevant property is *AND*-compositionality. We assume that there exists an operation $\times$, defined over the set of denotations, which computes the *AND*-composition of two denotations. In the case of *SLD*-trees denotations, the definition of $\times$ is shown by the following equation.

$$A \times B = \{ \langle c_1 \otimes c_2 \,\square\, \tilde{b}_1, \tilde{b}_2; c_1' \otimes c_2' \,\square\, \tilde{b}_1', \tilde{b}_2'; ks_1 :: ks_2 \rangle \mid \langle c_1 \,\square\, \tilde{b}_1; c_1' \,\square\, \tilde{b}_1'; ks_1 \rangle \in A, \langle c_2 \,\square\, \tilde{b}_2; c_2' \,\square\, \tilde{b}_2'; ks_2 \rangle \in B \}.$$

$\times$ is related to the operation $\otimes$ defined over $\mathcal{A}$, to the conjunction of atom sequences and to the *AND*-compositionality property of theorem 2.6.

Another essential feature that we want to preserve is the *SLD*-trees branching structure. The operation which puts together two denotations nondeterministically is the union of well-formed sets of nodes. By using an algebraic notation, for each pair A, B of well-formed sets of nodes we write $A + B = A \cup B$. The operation $+$ is related to the operation $\oplus$ defined over the constraint system $\mathcal{A}$ by the properties: $(c_1 \oplus c_2)\, d = c_1 d + c_2 d$ and $c\,(d_1 + d_2) = c d_1 + c d_2$. In analogy to what happens in $\mathcal{A}$ for $\oplus$ and $\otimes$, the product $\times$ is (left and right) distributive w.r.t. $+$, i.e. $d_1 \times (d_2 + d_3) = d_1 \times d_2 + d_1 \times d_3$. This property shows that the answers of conjunctive goals are all the compositions of the answers of the conjuncts.

The last issue we must be concerned with is that all the properties must hold "modulo variance". This property is usually modeled by renamings. Therefore we define a "renaming operation" ∇ on the objects of the do main. ∇ commutes with $+$ and $\times$ and is defined as a family of renamings ∇_ϑ depending on the renaming ϑ. In order to satisfy the usual properties of renamings, $\nabla_\vartheta \circ \nabla_{\vartheta'} = \nabla_{\vartheta \circ \vartheta'}$ must hold. Furthermore ∇_ϑ must be an extension of the renaming operation ∂_ϑ of the constraint system $\mathcal{A}$, on which the domain is defined. For each well-formed set A we define

$$\nabla_\vartheta(A) = \{ \langle \partial_\vartheta(c) \,\square\, \tilde{b}\vartheta; \partial_\vartheta(c') \,\square\, \tilde{b}'\vartheta; ks\vartheta \rangle \mid \langle c \,\square\, \tilde{b}; c' \,\square\, \tilde{b}'; ks \rangle \in A \}.$$

∇ is a renaming operator. Let $(\Theta, \circ, id)$ be the group of renamings. A *renaming operator* ∇ on D is an injective group homomorphism $(\Theta, \circ, id) \rightarrow (D \rightarrow D, \circ, id)$ s.t. for each $c \in \mathcal{A}$, $d \in D$ and $\vartheta \in \Theta$ $\nabla_\vartheta\,(cd) = \partial_\vartheta(c)\nabla_\vartheta(d)$.

A renaming operator induces a "variance" relation $=_\nabla$. Namely $x =_\nabla y \iff \exists\, \vartheta : \nabla_\vartheta x = y$. Note that the above renaming operator on R induces exactly the variance relation $\equiv$, i.e. $x =_\nabla y \Leftrightarrow x \equiv y$.

We will appreciate the power of the above algebraic construction in the following section where the operations $+$, $\times$ and ∇ will play a relevant role in the definition of abstract denotations.

3 The abstraction framework

We consider two classes of observables, namely $\mathcal{S}$-observables and $\mathcal{I}$-ob servables. The $\mathcal{S}$-observables ($\mathcal{S}$emantic observables) are observables for which we can define a *denotation*, which generalizes the properties of the

s-semantics [2]. This denotation provides a correct and complete characterization of the (abstract) program behavior. The program denotation is defined by collecting the behaviors for *most general atomic goals*, i.e. goals consisting of the application of a predicate symbol to a tuple of distinct variables. We show how we can reconstruct within the framework some existing semantics, such as the answer constraint semantics [13] and the call patterns semantics [15, 16], thus obtaining all the relevant theorems simply by specializing the theorems which are valid in the framework.

The $\mathcal{I}$-observables (abstract $\mathcal{I}$nterpretation observables) are meant to capture the abstractions involved in abstract interpretation, where approximation is the rule of the game. Theorems valid for $\mathcal{I}$-observables are therefore weaker and denotations provide characterizations of semantic properties which are correct in the sense of abstract interpretation theory. We show how we can reconstruct the abstract semantics defined in [12], which allows us to derive groundness relations among the arguments of procedure calls.

3.1 An algebraic formalization of observables: the $\mathcal{S}$-observables

Given an observable we want to be able to observe computations of conjunctive goals from the single conjuncts computations. Moreover we do not want to loose the non-deterministic structure and the independence of the results upon renaming. We enforce all these properties by using an $\mathcal{S}$-domain.

Definition 3.1 ($\mathcal{S}$-domain) *A nonempty set* D *is an* $\mathcal{S}$-domain *on a constraint system* $\mathcal{A}(\otimes, \oplus, \mathsf{1}, \mathsf{0})$*, and is denoted by* $\mathsf{D}(+, \times, \nabla)$*, if there exist two operations* $+$*,* $\times$ *on* D*, a renaming* ∇ *(on* D*), and two elements 0, 1 in* D *s.t.* $(\mathsf{D}, \times, +, \mathsf{1}, \mathsf{0})$ *is a closed semiring. Moreover for each* $\mathsf{c} \in \mathcal{A}$ *and* $\nu \in \mathsf{D}$ *there exists an element* $\mathsf{c} \cdot \nu$ *in* D *s.t. for each* $\nu_1, \nu_2 \in \mathsf{D}$

1) $\mathsf{c} \cdot (\nu_1 + \nu_2) = \mathsf{c} \cdot \nu_1 + \mathsf{c} \cdot \nu_2$, *4)* $\mathsf{c}_1 \cdot (\mathsf{c}_2 \cdot \nu) = (\mathsf{c}_1 \otimes \mathsf{c}_2) \cdot \nu$,
2) $(\mathsf{c}_1 \oplus \mathsf{c}_2) \cdot \nu = \mathsf{c}_1 \cdot \nu + \mathsf{c}_2 \cdot \nu$, *5)* $\mathsf{1} \cdot \nu = \nu$.
3) $\mathsf{0} \cdot \nu = \mathsf{0}$,

Furthermore for each $\vartheta_1, \vartheta_2 \in \Theta$*, there exists* $\vartheta \in \Theta$ *s.t.*

a) $\nabla_{\vartheta_1} \nu_1 \times \nabla_{\vartheta_2} \nu_2 = \nabla_{\vartheta}(\nu_1 \times \nu_2)$, *b)* $\nabla_{\vartheta_1} \nu_1 + \nabla_{\vartheta_2} \nu_2 = \nabla_{\vartheta}(\nu_1 + \nu_2)$.

We can define, for each $\mathcal{S}$-domain D, a canonical ordering as follows: $\nu_1 \leq \nu_2$ iff $\nu_1 + \nu_2 = \nu_2$. It is easy to see that $(\mathsf{D}, \leq)$ is a complete lattice.

Example 3.2 The set R of *SLD*-trees of section 2 is an $\mathcal{S}$-domain. Moreover the set $\mathsf{D} = \mathcal{P}(\mathsf{G} \times \mathcal{A})$ (the domain of $\xi : \mathsf{R} \to \mathsf{D}$ of example 2.3) is an $\mathcal{S}$-domain. In fact $\mathsf{c} \cdot \mathsf{A} = \{ (\mathsf{c} \otimes \mathsf{c}' \square \tilde{\mathsf{b}}; \mathsf{c} \otimes \mathsf{s}) \mid (\mathsf{c}' \square \tilde{\mathsf{b}}; \mathsf{s}) \in \mathsf{A}, \mathsf{c} \otimes \mathsf{s} > \mathsf{0} \}$ and $\mathsf{A} \times \mathsf{B} = \{ (\mathsf{c} \otimes \mathsf{c}' \square \tilde{\mathsf{b}}, \tilde{\mathsf{b}}'; \mathsf{s} \otimes \mathsf{s}') \mid (\mathsf{c} \square \tilde{\mathsf{b}}; \mathsf{s}) \in \mathsf{A}, (\mathsf{c}' \square \tilde{\mathsf{b}}'; \mathsf{s}') \in \mathsf{B}, \mathsf{s} \otimes \mathsf{s}' > \mathsf{0} \}$, where in case of conflict variables are renamed. The sum operation is set union, while the renaming operation is the usual renaming of *CLP*, i.e. $\nabla_{\vartheta}(\mathsf{A}) = \{ (\partial_{\vartheta}(\mathsf{c}) \square \tilde{\mathsf{b}}\vartheta; \partial_{\vartheta}(\mathsf{s})) \mid (\mathsf{c} \square \tilde{\mathsf{b}}; \mathsf{s}) \in \mathsf{A} \}$. ∎

We want now to define a notion of (forgetful) morphism between $\mathcal{S}$-domains.

Definition 3.3 ($\mathcal{S}$-observable) *A* morphism between $\mathcal{S}$-domains *($\mathcal{S}$-morphism),* $\alpha : \mathsf{D}(+, \times, \nabla) \to \bar{\mathsf{D}}(\bar{+}, \bar{\times}, \bar{\nabla})$ *is a surjective mapping* $\alpha : \mathsf{D} \to \bar{\mathsf{D}}$ *s.t.* $\forall x, y \in \mathsf{D}$, $c \in \mathcal{A}$, $\vartheta \in \Theta$

$$\begin{array}{llll} 1)\ \alpha(x + y) = \alpha(x) \,\bar{+}\, \alpha(y), & \alpha(0) = \bar{0}, & 3)\ \alpha(c \cdot x) = c \cdot \alpha(x), \\ 2)\ \alpha(x \times y) = \alpha(x) \,\bar{\times}\, \alpha(y), & \alpha(1) = \bar{1}, & 4)\ \alpha(\nabla_\vartheta(x)) = \bar{\nabla}_\vartheta(\alpha(x)). \end{array}$$

A morphism $\alpha : \mathcal{R}(+, \times, \nabla) \to \bar{\mathsf{D}}(\bar{+}, \bar{\times}, \bar{\nabla})$ *is an observable and we call it* $\mathcal{S}$-observable. $\mathbb{O}_s$ *denotes the set of* $\mathcal{S}$*-observables.*

Additivity and surjectivity allow the morphism to associate the right observation in $\bar{\mathsf{D}}$ to any concrete object in D. This is because $\mathcal{S}$-morphisms are Galois co-insertions with respect to the canonical orderings.

Example 3.4 The observable ξ of example 2.3 is an $\mathcal{S}$-morphism. ∎

3.2 Semantic properties of $\mathcal{S}$-observables

$\mathcal{S}$-domains are strongly related to semantic domains. The operations $+, \times$ of $\mathsf{R}(+, \times, \nabla)$ are similar to those of $\mathcal{R}$ defined in subsection 2.2 (that we will still denote by $+, \times$). We can map an $\mathcal{S}$-domain D in a semantic domain $\mathcal{D} = \mathsf{D}_{/=_\nabla}$ by using the canonical equivalence induced by ∇: $[\cdot]_\nabla : \mathsf{D} \to \mathcal{D}$, $[x]_\nabla = [x]_{/=_\nabla}$. We have that $\forall [\nu_1]_\nabla, [\nu_2]_\nabla \in \mathcal{R}$ $[\nu_1]_\nabla + [\nu_2]_\nabla = [\nu_1 + \nu_2]_\nabla$ and $[\nu_1]_\nabla \times [\nu_2]_\nabla = [\nu_1 \times \nu_2]_\nabla$.

These properties can be generalized to each $\mathcal{S}$-domain $\mathsf{D}(+, \times, \nabla)$ simply by defining for each $[x]_\nabla, [y]_\nabla \in \mathcal{D}$ $[x]_\nabla \,\bar{+}\, [y]_\nabla = [x + y]_\nabla$ and $[x]_\nabla \,\bar{\times}\, [y]_\nabla = [x \times y]_\nabla$. The set $\mathcal{D}(\bar{+}, \bar{\times})$ inherits from D all the properties we have discussed in section 2.2 for $\mathcal{R}$. From now on, we will call the structure $\mathcal{D}(\bar{+}, \bar{\times})$ *semantic domain*, and denote by $\mathfrak{D}$ the set of all the semantic domains.

Each morphism $\alpha : \mathsf{R} \to \mathsf{D}$ can be transformed into a "semantic domains morphism" (that we still denote by α) from $\mathcal{R}$ to the semantic domain $\mathcal{D} = \mathsf{D}_{/=_\nabla}$. We only need to set $\alpha([x]_\nabla) = [\alpha(x)]_\nabla$. This morphism is well defined thanks to axiom 4 in definition 3.3 which states compatibility between α and $=_\nabla$. Thus we have a syntactic abstraction which *preserves the interesting semantic properties of* $\mathcal{R}$. Now we show how the algebraic construction can be used to easily derive these properties. First define the α-top-down denotation $\mathcal{O}_\alpha(p)$ of a program p as $\alpha(\mathcal{O}(p))$.

Theorem 3.5 (abstract *AND*-compositionality) *Let* p *be a program,* $\alpha : \mathcal{R}(+, \times) \to \mathcal{D}(\bar{+}, \bar{\times})$ *be an* $\mathcal{S}$*-observable. Then* $\bar{\nu} \in \mathcal{B}^{\mathrm{lm}}_\alpha(p) \iff \exists c \in \mathcal{A}$, $\bar{e}_1, \ldots, \bar{e}_n \in \mathcal{O}_\alpha(p)$ *s.t.* $\bar{\nu} = c \cdot (\{\, \bar{e}_1 \,\} \bar{\times} \ldots \bar{\times} \{\, \bar{e}_n \,\})$.

Corollary 3.6 (correctness and full abstraction) *Let* $\alpha : \mathcal{R} \to \mathcal{D}$ *be an* $\mathcal{S}$*-observable and* p_1, p_2 *be programs. Then* $p_1 =_\alpha p_2 \iff \mathcal{O}_\alpha(p_1) = \mathcal{O}_\alpha(p_2)$.

In the bottom-up case the best approximation for the immediate consequences operator is $\mathsf{T}_{\mathfrak{p},\alpha} = \alpha \circ \mathsf{T}_{\mathfrak{p}} \circ \gamma$. If $\leq$ is the canonical ordering on $\mathcal{D}$, $\mathsf{T}_{\mathfrak{p},\alpha}$ is continuous on the lattice $(\mathsf{D}, \leq)$. Then we define the fixpoint semantics as $\mathcal{F}_\alpha(\mathfrak{p}) = \mathsf{T}_{\mathfrak{p},\alpha} \uparrow \omega$.

Definition 3.7 (compatibility) α *is* compatible *with* $\mathsf{T}_{\mathfrak{p}}$ *if* $\mathsf{T}_{\mathfrak{p},\alpha} \circ \alpha = \alpha \circ \mathsf{T}_{\mathfrak{p}}$.

Theorem 3.8 (bottom-up vs top-down) *Let* $\mathfrak{p}$ *be a program and* α *be an* $\mathcal{S}$*-observable. Then* $\mathcal{O}_\alpha(\mathfrak{p}) \leq \mathcal{F}_\alpha(\mathfrak{p})$. *Moreover if* α *is compatible with* $\mathsf{T}_{\mathfrak{p}}$ *then* $\mathcal{O}_\alpha(\mathfrak{p}) = \mathcal{F}_\alpha(\mathfrak{p})$.

Some remarks about the above results are necessary. The top-down abstract denotation, which is defined simply as the abstraction of the top-down denotation, has exactly the same properties of the top-down *SLD*-trees denotation, namely *AND*-compositionality, correctness and full abstraction. The bottom-up abstract denotation is in general less precise. The loss of precision is due to the fact that the abstract immediate consequences operator $\mathsf{T}_{\mathfrak{p},\alpha}$ is obtained by specializing the general immediate consequences operator $\mathsf{T}_{\mathfrak{p}}$. It is exactly this specialization which may sometimes result in a loss of precision. However, if the observable α is compatible with $\mathsf{T}_{\mathfrak{p}}$, the two constructions are equivalent. It is worth noting that most reasonable observables are indeed compatible with $\mathsf{T}_{\mathfrak{p}}$ (see the examples below). A similar relation between the top-down and bottom-up constructions was already noted for abstractions of the answer constraint in [17, 19, 20]. In that framework the equivalence was guaranteed in the case of distributive constraint systems. In such a case, our compatibility condition is always satisfied.

Let us finally note that, when the two constructions are equivalent, the bottom-up one is indeed more efficient, since abstraction is used at every step of the fixpoint construction, thus deriving only the minimal amount of information about the *SLD*-trees which is needed to characterize the observ able property. The top-down construction, on the contrary, is always forced to build complete *SLD*-trees which have later to be abstracted to get the observation.

Example 3.9 We show now how to reconstruct the *CLP* version of the $\mathcal{S}$-semantics [9, 10]. We have already shown in example 3.4 that ξ is an $\mathcal{S}$-observable. We can than apply theorem 3.5 and the definition of node, to obtain the following denotations:

$$\mathcal{B}^{\mathrm{lm}}_{\xi}(\mathfrak{p}) = \{\, (\mathsf{g};\mathsf{c})_{/\equiv} \mid \mathsf{g} \overset{\mathrm{ks}}{\rightsquigarrow} \mathsf{c}\,\Box \,\},\ \mathcal{O}_{\xi}(\mathfrak{p}) = \{\, (\mathsf{q}(\tilde{\mathsf{x}});\mathsf{c})_{/\equiv} \mid \mathsf{q}(\tilde{\mathsf{x}}) \overset{\mathrm{ks}}{\rightsquigarrow} \mathsf{c}\,\Box \,\}.$$

Note that $\mathcal{B}^{\mathrm{lm}}_{\xi}(\mathfrak{p})$ contains all the answer constraints of $\mathfrak{p}$, while $\mathcal{O}_{\xi}(\mathfrak{p})$ is *exactly* the CLP version of the top-down definition of the $\mathcal{S}$-semantics [13]. Corollary 3.6 tells us that $\mathcal{O}_{\xi}$ is correct and fully abstract w.r.t. answer constraints. Moreover theorem 3.5 tells us that answer constraints for any

goal can be derived from the answer constraints of the most general atomic goals. For the bottom-up case, by applying the construction, we obtain

$$\mathsf{T}_{p,\xi}(X) = \{\, (q(\tilde{x}); c) \mid \exists\, q(\tilde{t}) :\!-\ c_p \,\square\, q_1(\tilde{t}_1), \ldots, q_n(\tilde{t}_n) \in p,\ (q_i(\tilde{x}_i); c_i) \in X,\\ c = \tilde{x}{=}\tilde{t} \otimes c_p \otimes \tilde{x}_1{=}\tilde{t}_1 \otimes c_1 \otimes \ldots \otimes \tilde{x}_n{=}\tilde{t}_n \otimes c_n \,\}.$$

Moreover T_p is compatible with ξ, and, because of theorem 3.8, $\mathcal{O}_\xi = \mathcal{F}_\xi$. This result shows the power of our theory. In fact the proof of the same result, using classical methods [13], needs much more effort. ∎

Example 3.10 The call patterns of a program p for a goal g and a selection rule r are the atoms selected in any *SLD*-derivation of g in p via r. We define the $\mathcal{S}$-domain $\mathsf{CP}_\mathcal{A}$ made of objects of the form $(g, c \,\square\, a)$, where g is a goal and $c \,\square\, a$ is an atom. The interpretation is "the execution of the goal g generates a procedure call a with state (constraint) c". The operations can be defined as follows: $A + B = A \cup B$,

$$c \cdot A = \{\, (c \otimes c' \,\square\, \tilde{b}; c \otimes c'' \,\square\, a) \mid (c' \,\square\, \tilde{b}; c'' \,\square\, a) \in A, c \otimes c'' > 0 \,\},$$

$$A \times B = \{\, (g, g'; c \,\square\, a) \mid (g; c \,\square\, a) \in A \,\} \cup \\ \{\, (g, g'; c \otimes c' \,\square\, a') \mid (g; c \,\square\,) \in A, (g'; c' \,\square\, a') \in B \,\},$$

$$\nabla_\vartheta(A) = \{\, (\partial_\vartheta(c) \,\square\, \tilde{b}\vartheta; \partial_\vartheta(c') \,\square\, a\vartheta) \mid (c \,\square\, \tilde{b}; c' \,\square\, a) \in A \,\}.$$

The abstraction which allows us to obtain the call patterns is $\eta(A) = \{\, (g; c \,\square\, a) \mid \langle g; c \,\square\, a, \tilde{b}; ks\rangle \in A \,\}$. By using the definition of node, we have that $\mathcal{B}^{lm}_\eta(p) = \{\, (g; c \,\square\, a) \mid g \overset{ks}{\rightsquigarrow} c \,\square\, a, \tilde{b} \,\}$, which is exactly the set of all the call patterns of p. The top-down denotation is $\mathcal{O}_\eta(p) = \{\, (q(\tilde{x}); c \,\square\, a) \mid q(\tilde{x}) \overset{ks}{\rightsquigarrow} c \,\square\, a, \tilde{b} \,\}$, while the immediate consequences operator turns out to be

$$\mathsf{T}_{p,\eta}(X) = \{\, (q(\tilde{x}); c \,\square\, a) \mid \exists\, q(\tilde{t}) :\!-\ c_p \,\square\, q_1(\tilde{t}_1), \ldots, q_n(\tilde{t}_n) \in p, \text{ and} \\ c = \tilde{x}{=}\tilde{t} \otimes c_p,\ a = q_1(\tilde{t}_1) \text{ or} \\ c = \tilde{x}{=}\tilde{t} \otimes c_p \otimes \tilde{t}_1{=}\tilde{y} \otimes c',\ (q_1(\tilde{y}); c' \,\square\, a) \in X \,\}.$$

Since T_p is compatible with η, $\mathcal{F}_\eta$ is correct and fully abstract w.r.t. $=_\eta$. We obtained the call pattern semantics defined in [16, 11, 15]. ∎

3.3 Other observables

For a generic observable $\alpha : R \to D$, the previous theorems state that if α is not an $\mathcal{S}$-observable then we cannot have a correct and fully abstract denotation which is *AND*-compositional. Instead we can have a correct *AND*-compositional denotation which is minimal w.r.t. the content of information. In analogy to what we did in the case of observables, we can consider the set $\mathcal{S}$ of *AND*-compositional denotations $\mathcal{S} : \mathcal{P} \to \mathcal{D}$ with $\mathcal{D} \in \mathbb{D}$. We can partially order $\mathcal{S}$ by approximation. A denotation $\mathcal{S} : \mathcal{P} \to \mathcal{D} \in \mathcal{S}$ approximates $\mathcal{S}' : \mathcal{P} \to \mathcal{D}'$ if there exists a Galois co-insertion $\alpha : \mathcal{D} \to \mathcal{D}'$ s.t. $\mathcal{S}' = \alpha \circ \mathcal{S}$. Note that every denotation which approximates another denotation which is correct w.r.t. an observable is still correct w.r.t. the same observable.

Theorem 3.11 (the denotation) *For each observable α there exists a minimal AND-compositional denotation correct w.r.t. it, i.e. which is approximated by all the other AND-compositional correct denotations.*

4 Abstract interpretation

One more motivation for our algebraic construction can be found in abstract interpretation. The essence of abstract interpretation is to give a non–standard interpretation to the language. In the *CLP* case, as shown in [5, 17], we only need to give a non–standard interpretation to constraints.

In general a constraint system is an interpretation (in a semi-closed semiring) for constraint formulas. According to the approach of [19, 17, 20] constraint systems are related by means of constraint system semi-morphisms. The program interpretation process is expressed in terms of a set of algebraic operators which model how data objects are collected during the computation. An abstract interpretation for a given $CLP(\mathcal{A})$ program is then the semantics of an abstract program in $CLP(\mathcal{A}')$ where $\mathcal{A}'$ is a suitable constraint system correct w.r.t. $\mathcal{A}$.

In our framework, abstract interpretation can be viewed as the composition of a constraint system semi-morphism (the abstraction of the domain interpretation) and of an $\mathcal{S}$-observable, which chooses an adequate abstraction of *SLD*-trees. The construction is based on semi-morphisms on constraint systems. A constraint system semi-morphism $\mu : \mathcal{A} \to \mathcal{A}'$ can be extended to an $\mathcal{S}$-domain semi-morphism $\mu^{\mathcal{D}} : \mathsf{D}_{\mathcal{A}} \to \mathsf{D}_{\mathcal{A}'}$ in a natural way.

Definition 4.1 ($\mathcal{S}$-semi-morphism) *Let $\mathsf{D}_{\mathcal{A}}$, $\mathsf{D}'_{\mathcal{A}'}$ be $\mathcal{S}$-domains, with $\mathcal{A}'$ correct w.r.t. $\mathcal{A}$. An $\mathcal{S}$-*semi-morphism $\alpha_\mu : \mathsf{D}_{\mathcal{A}} \to \mathsf{D}'_{\mathcal{A}'}$*, based on the constraint system semi-morphism $\mu : \mathcal{A} \to \mathcal{A}'$, is the composition of μ^{D} and an $\mathcal{S}$-morphism $\alpha : \mathsf{D}_{\mathcal{A}'} \to \mathsf{D}'_{\mathcal{A}'}$, i.e. $\alpha_\mu = \alpha \circ \mu^{\mathsf{D}}$.*

Note that $\mathcal{S}$-semi-morphisms are Galois co-insertion. Since they are strongly related to abstract interpretation, we will call them *$\mathcal{I}$-observables*.

The following discussion on the abstract denotations relies on the notion of "abstracting the program". The idea is to replace the constraints (defined on the constraint system $\mathcal{A}$) in the original program with their abstract version (defined on a constraint system $\mathcal{A}'$ correct w.r.t. $\mathcal{A}$), thus obtaining a *CLP* program on a different constraint system. If $\mu : \mathcal{A} \to \mathcal{A}'$ is the semi-morphism which formalizes the (correct) constraint system abstraction we denote by $\mu^{\mathcal{P}} : \mathcal{P}_{\mathcal{A}} \to \mathcal{P}_{\mathcal{A}'}$ the homomorphism obtained by extending μ to programs in the natural way, i.e. by applying μ to the constraints and terms occurring in the program. The abstract denotation of $\mathtt{p}$ is simply the denotation $\mathcal{O}(\mu^{\mathcal{P}}(\mathtt{p}))$ of the abstract program $\mu^{\mathcal{P}}(\mathtt{p})$. When clear from the context we denote $\mu^{\mathcal{P}}$ by μ. The following theorem generalizes a result in [17] and states that the semantics of the "abstract program" is a safe approximation of the abstraction of the semantics of the program.

Theorem 4.2 (abstract program) *Let* $\mu : \mathcal{A} \to \mathcal{A}'$ *be a constraint system semi-morphism. Then* $\mu^{\mathcal{R}}(\mathcal{O}(p)) \leq \mathcal{O}(\mu(p))$.

Since an $\mathcal{I}$-observable on the constraint system $\mathcal{A}$ is obtained by an $\mathcal{S}$-observable on the constraint system $\mathcal{A}'$ we expect the semantics construction to be almost the same for the abstract interpretation case. Obviously the theorems are weaker, because of the lack of precision of the denotations.

Definition 4.3 (abstract denotations) *The ideal top-down denotation of an* $\mathcal{I}$*-observable* $\alpha_\mu : \mathcal{R}_\mathcal{A} \to \mathcal{D}_{\mathcal{A}'}$ *is* $\mathcal{O}_{\alpha_\mu}(p) = \alpha_\mu(\mathcal{O}(p))$*, while the abstract top-down denotation is* $\mathcal{O}_\alpha(\mu(p)) = \alpha(\mathcal{O}(\mu(p)))$*. The ideal immediate consequences operator is* $T_{p,\alpha_\mu} = \alpha_\mu \circ T_p \circ \gamma_\mu$*, while the abstract immediate consequences operator is* $T_{\mu(p),\alpha} = \alpha \circ T_{\mu(p)} \circ \gamma$*. The ideal bottom-up denotation is* $\mathcal{F}_{\alpha_\mu}(p) = T_{p,\alpha_\mu} \uparrow \omega$*, while the abstract bottom-up denotation is* $\mathcal{F}_\alpha(\mu(p)) = T_{\mu(p),\alpha} \uparrow \omega$.

The following theorem relates the different abstract interpretation mechanisms, i.e. the bottom-up execution of the abstract program $\mathcal{F}_\alpha(\mu(p))$, the top-down abstract execution of the abstract program $\mathcal{O}_\alpha(\mu(p))$, the abstraction of the top-down concrete execution $\mathcal{O}_{\alpha_\mu}(p)$ and the (specialized) bottom-up execution of the concrete program $\mathcal{F}_{\alpha_\mu}(p)$. As one could expect, the ideal top-down denotation is (safely) approximated by all the other denotations. $\mathcal{F}_\alpha(\mu(p))$ is the least precise and, in the case of compatibility, we have the usual equivalence between top-down and bottom-up executions. This is shown by the following theorem.

Theorem 4.4 *Let* p *be a program,* $\alpha_\mu : \mathcal{R}_\mathcal{A} \to \mathcal{D}_{\mathcal{A}'}$ *be an* $\mathcal{I}$*-observable. Then* $\mathcal{F}_\alpha(\mu(p)) \geq \mathcal{O}_\alpha(\mu(p))$*,* $\mathcal{F}_\alpha(\mu(p)) \geq \mathcal{F}_{\alpha_\mu}(p)$*,* $\mathcal{O}_\alpha(\mu(p)) \geq \mathcal{O}_{\alpha_\mu}(p)$ *and* $\mathcal{F}_{\alpha_\mu}(p) \geq \mathcal{O}_{\alpha_\mu}(p)$*. Moreover, if* α *is compatible with* $T_{\mu(p)}$*, we have* $\mathcal{F}_\alpha(\mu(p)) = \mathcal{O}_\alpha(\mu(p)) \geq \mathcal{F}_{\alpha_\mu}(p) = \mathcal{O}_{\alpha_\mu}(p)$.

The next theorem shows that *AND*-compositionality holds in the denotation based on the abstract program.

Theorem 4.5 (*AND*-compositionality) *Let* p *be a program,* $\alpha_\mu : \mathcal{R}_\mathcal{A}(+, \times) \to \mathcal{D}_{\mathcal{A}'}(\bar{+}, \bar{\times})$ *an* $\mathcal{I}$*-observable. Then* $\nu' \in \mathcal{B}^{lm}_\alpha(\mu(p)) \iff \exists\, c' \in \mathcal{A}', e'_1, \ldots, e'_n \in \mathcal{O}_\alpha(\mu(p))$ *s.t.* $\nu' = c' \cdot (\{\, e'_1 \,\}\bar{\times} \ldots \bar{\times}\{\, e'_n \,\})$, $\nu' \in \mathcal{B}^{lm}_{\alpha_\mu}(p) \iff \exists\, c' \in \mathcal{A}', e'_1, \ldots, e'_n \in \mathcal{O}_{\alpha_\mu}(p)$ *s.t.* $\nu' \leq c' \cdot (\{\, e'_1 \,\}\bar{\times} \ldots \bar{\times}\{\, e'_n \,\})$.

Example 4.6 Let $\mathcal{A}_\mathcal{H}$ be the Herbrand constraint system. We show how to reconstruct the ground dependency analysis for call patterns described in [12]. The abstract domain of computation is *Prop* [7], consisting of propositional formulas which provide a concise representation of abstract substitutions which describe ground dependency relations among arguments of a procedure call. $\mathcal{A}_{Prop}$ is the algebra of possibly existentially quantified disjunctions of formulas $\big(Prop, \wedge, \vee, \mathit{true}, \mathit{false}, \exists_x, \partial^t_x, \Lambda(t) \leftrightarrow \Lambda(t')\big)$, where each t belongs to V^n (for some n) and $\Lambda(t) = x_1 \wedge \ldots \wedge x_n$ for $t = \{x_1, \ldots, x_n\}$ ($\Lambda(\emptyset) = \mathtt{true}$).

We define a constraint system semi-morphism $\mu : \mathcal{A}_{\mathcal{H}} \to \mathcal{A}_{Prop}$ as

$$\mu(c) = \begin{cases} \mathtt{false} & \text{if } c = 0 \\ \Lambda(\mathit{var}(c)) & \text{if } c \text{ is a simple constraint} \\ \Lambda(\mu(t)) \leftrightarrow \Lambda(\mu(t')) & \text{if } c = t{=}t' \\ \mu(c_1) \wedge \mu(c_2) & \text{if } c = c_1 \otimes c_2 \\ \mu(c_1) \vee \mu(c_2) & \text{if } c = c_1 \oplus c_2 \end{cases}$$

and extend it to well-formed sets of nodes as $\mu^{\mathcal{R}}(A) = \{\langle \mu(c) \,\square\, \mu(b); \mu(c') \,\square\, \mu(b'); ks\rangle \mid \langle c \,\square\, b; c' \,\square\, b'; ks\rangle \in A\}$. Since the call pattern observable η of example 3.10 was defined for any constraint system $\mathcal{A}$, we can compose it with $\mu^{\mathcal{R}}$: $\eta_\mu(A) = (\eta \circ \mu^{\mathcal{R}})(A) = \{(g; c \,\square\, a) \mid \langle g; c \,\square\, a, \tilde{b}; ks\rangle \in \mu^{\mathcal{R}}(A)\}$ and obtain an abstract interpretation in *Prop*. The ground dependencies call patterns denotation are

$$\mathcal{O}_{\eta_\mu}(p) = \{(q(\tilde{x}); c_\mu \,\square\, q_1(\tilde{t}_\mu)) \mid q(\tilde{x}) \overset{ks}{\rightsquigarrow} c \,\square\, q_1(\tilde{t}), \tilde{b},\ c_\mu = \mu(c),\ \tilde{t}_\mu = \mu(\tilde{t})\},$$

$$\mathcal{O}_{\eta}(\mu(p)) = \{(q(\tilde{x}); c \,\square\, a) \mid q(\tilde{x} \overset{ks}{\underset{\mu(p)}{\rightsquigarrow}} c \,\square\, a, \tilde{b}\},$$

$$\begin{aligned} T_{p,\eta_\mu}(X) = \{ & (q(\tilde{x}); c_\mu \,\square\, q_1(\tilde{t}_\mu)) \mid \exists\, q(\tilde{t}) :\!-\ c_p \,\square\, q_1(\tilde{t}_1), \ldots, q_n(\tilde{t}_n) \in p, \text{ and} \\ & c_\mu = \Lambda(\tilde{x}) \leftrightarrow \Lambda(\tilde{t}) \wedge \mu(c_p),\ \tilde{t}_\mu = \mu(\tilde{t}_1) \text{ or} \\ & c_\mu = \Lambda(\tilde{x}) \leftrightarrow \Lambda(\tilde{t}) \wedge \mu(c_p) \wedge \Lambda(\tilde{t}_1) \leftrightarrow \Lambda(\tilde{y}) \wedge c'_\mu, \\ & (q_1(\tilde{y}); c'_\mu \,\square\, q_1(\tilde{t}_\mu)) \in X\}, \end{aligned}$$

$$\begin{aligned} T_{\mu(p),\eta}(X) = \{ & (q(\tilde{x}); c \,\square\, a) \mid \exists\, q(\tilde{t}) :\!-\ c_p \,\square\, q_1(\tilde{t}_1), \ldots, q_n(\tilde{t}_n) \in \mu(p), \text{ and} \\ & c = \Lambda(\tilde{x}) \leftrightarrow \Lambda(\tilde{t}) \wedge c_p,\ a = q_1(\tilde{t}_1) \text{ or} \\ & c = \Lambda(\tilde{x}) \leftrightarrow \Lambda(\tilde{t}) \wedge c_p \wedge \Lambda(\tilde{t}_1) \leftrightarrow \Lambda(\tilde{y}) \wedge c', \\ & (q_1(\tilde{y}); c' \,\square\, a) \in X\}. \end{aligned}$$

Since $T_{\mu(p)}$ is compatible with η, $\mathcal{F}_\alpha(\mu(p)) = \mathcal{O}_\alpha(\mu(p))$. The semantics obtained in [12] and, through a magic set like transformation, in [4] is $\mathcal{O}_\alpha(\mu(p))$. It is a goal independent denotation computable either top-down or bottom-up. ■

5 Conclusions

We have defined an algebraic framework which allows us to prove several properties of concrete and abstract *SLD*-trees. The framework provides: a) a denotation consisting of all the *SLD*-trees obtained from most general atomic goals, with an equivalent alternative fixpoint construction; b) a set of important theorems which show that the denotation characterizes the *SLD*-trees for any goal; c) a mechanism ($\mathcal{S}$-observable) for abstracting the semantics, which guarantees that the general theorems do hold for any abstraction and always leads to the best (correct and fully abstract) denotation; d) a mechanism ($\mathcal{I}$-observable) to model abstraction by approximation,

which guarantees that a weaker form of the general theorems is still valid and provides the semantic basis for abstract interpretation.

We have not considered yet two issues that could be discussed within the framework. The first one is related to *OR-compositionality* [3], which is a relevant property if we want to be able to reason about programs in a modular way. *SLD*-trees are indeed *OR*-compositional. However, the abstraction process can destroy this property. For example, two of the abstractions that we have considered, namely computed answer constraints and call patterns, are not *OR*-compositional. The theory should be extended with a characterization of *OR*-compositional observables. For non-*OR*-compositional observables there should be a result similar to the one of theorem 3.11. The second issue is related to the computation rule. The current results apply to the case of local selection rules, even if we have considered the leftmost selection rule only. The property that could be analyzed within the framework is the *independence from the selection rule*. *SLD*-trees do depend on the selection rule. However, more abstract observables, such as answer constraints, are independent from the selection rule. A second relevant extension of the framework might be the definition of conditions which guarantee that the abstractions are independent from the selection rule.

References

[1] K. R. Apt. Introduction to Logic Programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 495–574. Elsevier, Amsterdam and The MIT Press, Cambridge, 1990.

[2] A. Bossi, M. Gabbrielli, G. Levi, and M. Martelli. The s-semantics approach: Theory and applications. *Journal of Logic Programming*, 1994. to appear.

[3] A. Bossi, M. Gabbrielli, G. Levi, and M. C. Meo. A Compositional Semantics for Logic Programs. *Theoretical Computer Science*, 122(1-2):3–47, 1994.

[4] M. Codish and B. Demoen. Analysing Logic Programs using "`prop`"-ositional Logic Programs and a Magic Wand. In D. Miller, editor, *Proc. 1993 Int'l Symposium on Logic Programming*, pages 114–129. The MIT Press, Cambridge, Mass., 1993.

[5] P. Codognet and G. Filè. Computations, Abstractions and Constraints. In *Proc. Fourth IEEE Int'l Conference on Computer Languages*. IEEE Press, 1992.

[6] M. Comini and G. Levi. An algebraic theory of observables. Technical report, Dipartimento di Informatica, Università di Pisa, 1994.

[7] A. Cortesi, G. Filè, and W. Winsborough. *Prop* revisited: Propositional Formula as Abstract Domain for Groundness Analysis. In *Proc. Sixth IEEE Symp. on Logic In Computer Science*, pages 322–327. IEEE Computer Society Press, 1991.

[8] P. Cousot and R. Cousot. Systematic Design of Program Analysis Frameworks. In *Proc. Sixth ACM Symp. Principles of Programming Languages*, pages 269–282, 1979.

[9] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. A new Declarative Semantics for Logic Languages. In R. A. Kowalski and K. A. Bowen, editors, *Proc. Fifth Int'l Conf. on Logic Programming*, pages 993–1005. The MIT Press, Cambridge, Mass., 1988.

[10] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. Declarative Modeling of the Operational Behavior of Logic Languages. *Theoretical Computer Science*, 69(3):289–318, 1989.

[11] M. Gabbrielli. *The Semantics of Logic Programming as a Programming Language*. PhD thesis, Dipartimento di Informatica, Università di Pisa, 1992.

[12] M. Gabbrielli and R. Giacobazzi. Goal independency and call patterns in the analysis of logic programs. In *Proc. SAC'94*, 1994.

[13] M. Gabbrielli and G. Levi. Modeling Answer Constraints in Constraint Logic Programs. In K. Furukawa, editor, *Proc. Eighth Int'l Conf. on Logic Programming*, pages 238– 252. The MIT Press, Cambridge, Mass., 1991.

[14] M. Gabbrielli and G. Levi. On the Semantics of Logic Programs. In J. Leach Albert, B. Monien, and M. Rodriguez-Artalejo, editors, *Automata, Languages and Programming, 18th International Colloquium*, volume 510 of *Lecture Notes in Computer Science*, pages 1–19. Springer-Verlag, Berlin, 1991.

[15] M. Gabbrielli, G. Levi, and M. C. Meo. Observational Equivalences for Logic Programs. In K. Apt, editor, *Proc. Joint Int'l Conf. and Symposium on Logic Programming*, pages 131–145. The MIT Press, Cambridge, Mass., 1992.

[16] M. Gabbrielli and M. C. Meo. Fixpoint Semantics for Partial Computed Answer Substitutions and Call Patterns. In H. Kirchner and G. Levi, editors, *Algebraic and Logic Programming, Proceedings of the Third International Conference*, volume 632 of *Lecture Notes in Computer Science*, pages 84–99. Springer-Verlag, Berlin, 1992.

[17] R. Giacobazzi. *Semantic Aspects of Logic Program Analysis*. PhD thesis, Dipartimento di Informatica, Università di Pisa, 1992.

[18] R. Giacobazzi. On the Collecting Semantics of Logic Programs. In F. S. de Boer and M. Gabbrielli, editors, *Verification and Analysis of Logic Languages, Proc. of the Post-Conference ICLP Workshop*, pages 159–174, 1994.

[19] R. Giacobazzi, S. K. Debray, and G. Levi. A Generalized Semantics for Constraint Logic Programs. In *Proceedings of the International Conference on Fifth Generation Computer Systems 1992*, pages 581–591, 1992.

[20] R. Giacobazzi, G. Levi, and S. K. Debray. Joining Abstract and Concrete Computations in Constraint Logic Programming. In M. Nivat, C. Rattray, T. Rus, and G. Scollo, editors, *Algebraic Methodology and Software Technology (AMAST'93), Proceedings of the Third International Conference on Algebraic Methodology and Software Technology*, Workshops in Computing, pages 111–127. Springer-Verlag, Berlin, 1993.

[21] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, 1987. Second edition.

[22] L. Vieille. Recursive query processing: the power of logic. *Theoretical Computer Science*, 69:1–53, 1989.