Properties of the lattice of observables in logic programming

Gianluca Amato **Giorgio Levi**
Dipartimento di Informatica, Università di Pisa
Corso Italia 40, 56125 Pisa, Italy
E-mail: {amato, levi}@di.unipi.it
Ph.: +39-50-887284, 887246 Fax: +39-50-887226

Abstract
We show several properties of the abstract interpretation settings regarding relationships between precision of semantic operators and abstract domains composition. Then, we apply these results to the framework for logic programs introduced in [3], extended with the new class of operational observables. We prove that the classes of perfect, denotational and operational observables are complete lattices and we discuss some problems that arise studying them. Finally, we show how to use functional dependencies to systematically derive new domains in which our semantic operators enjoy desired precision properties.

keywords: logic programming, semantics, compositionality, abstract interpretation, abstract semantics.

1 Introduction

Our goal is showing several useful properties enjoyed by the lattice of abstractions introduced in [3] and subsequently developed in [17], [4] and [5]. In these papers, an operational top-down and a denotational bottom-up semantics for positive logic programs are defined, both of them expressed in terms of SLD-derivations, and several properties of compositionality, correctness, minimality, and equivalence are stated. Then, abstract interpretation techniques are introduced to model abstraction, giving as a result a set of simple conditions which guarantee the validity of several general theorems.

This brings to the introduction of some classes of abstractions characterized by different properties. Some of these classes (i.e. perfect and denotational observables) contain abstractions such as resultants [12], computed answers [10, 11] and call patterns [13], already used to define various notions of semantics for logic programs. Other classes (i.e. semi-denotational), on the contrary, are characterized by a loss of precision which makes them useful for program static analysis.

In this paper we focus on the first kind of abstraction classes. The properties which are used to characterize them are based on the precision of semantic operators. Since precision is a general property of abstract interpretation frameworks, some of the results we obtain are useful in this general setting too.

After some preliminary definitions, in section 2 we prove that meet and join on the abstraction lattices preserve precision for every additive operator over the concrete domain. Moreover, we state some useful conditions that imply precision in functional dependencies domains.

The framework for logic programs is briefly discussed in section 3, where it is extended with the new class of operational observables, characterized by a precise and goal-compositional operational semantics.

In section 3 we apply the results of section 2 to the framework. We prove that the classes of perfect, denotational and operational observables are complete lattices and we discuss some problems that arise studying them. One is the existence of observables which, although their operational and/or denotational semantic is precise and compositional, do not fall in any of our categories. Another problem is the existence of perfect (or denotational or operational) observables that are not comparable with well known abstract domains, such as resultants, that we thought strictly related to all observables in that class. We can partially overcome this problem restricting ourselves to the subset of all observables that are concretizations of Herbrand's success set. Finally, we show how to use functional dependencies to systematically derive new domains in which our semantic operators enjoy desired precision properties.

Conclusions and notes about possible future works complete the paper in section 5.

1.1 Preliminaries

Throughout the paper we will assume familiarity with the basic notions of lattice theory [2], logic programming [1, 16] and abstract interpretation [7], in the form presented in [9]. We will model abstractions by upper closure operators rather then by Galois insertions, as done, for instance, in [6].

Remember that, given a complete lattice C, an *(upper) closure operator* on C (uco in short) is a function $\rho : C \to C$ monotonic, idempotent and extensive (viz. $\forall x \in C.\ x \sqsubseteq \rho(x)$). Each closure operator is uniquely determined by the set of its fix points, equals to $\rho(C)$. To be more precise, it is $\rho(x) = \bigsqcap\{y \sqsupseteq x \mid y \in \rho(C)\}$. A set $X \subseteq C$ is the set of fixpoints of a closure operator if and only if X is a *Moore family* of C, i.e. $\top_C \in X$ and X is meet closed. Moreover, the set of fixpoints of an uco ρ is a complete lattice but, unless ρ is additive, it is not a sublattice of C because the join operators are different.

We denote with $\mathrm{uco}(C)$ the set of all the upper closure operators on C. $\mathrm{uco}(C)$ is a complete lattice such that, for each $\rho, \eta \in \mathit{uco}(C)$, $\{\rho_i\}_{i\in I} \subseteq \mathrm{uco}(C)$ and $x \in C$:

1. $\rho \sqsubseteq \eta$ iff $\forall x \in C.\ \rho(x) \sqsubseteq \eta(x)$ iff $\rho(C) \supseteq \eta(C)$,
2. $\left(\bigsqcap_{i\in I} \rho_i\right)(x) = \bigsqcap_{i\in I} \rho_i(x)$,
3. $\left(\bigsqcup_{i\in I} \rho_i\right)(C) = \bigsqcap_{i\in I} \rho_i(C)$.

It is possible [8] to assign to each Galois insertion $\langle \gamma, C, D, \alpha\rangle$ the upper closure operator $\gamma \circ \alpha$ and, in the opposite direction, to each uco ρ on C, the Galois insertion $\langle \iota^{-1}, C, D, \iota \circ \rho\rangle$ where $\iota : \rho(C) \to D$ is an isomorphism of complete lattices. Therefore, using uco's we can reason about properties of abstractions up to isomorphism of abstract domains, with a simplified notation compared with the approach that use Galois insertions.

Definitions of correct, optimal and precise operators have a formal counterpart in this setting. Given $\tilde{\mathbf{op}} : \rho(C)^n \to \rho(C)$ and $\mathbf{op} : C^n \to C$, we say that $\tilde{\mathbf{op}}$, w.r.t. $\mathbf{op}$, is:

- correct if $\forall \boldsymbol{x} \in C^n.\, \mathbf{op}(\boldsymbol{x}) \sqsubseteq \tilde{\mathbf{op}}(\rho(\boldsymbol{x}))$
- optimal if $\forall \boldsymbol{x} \in \rho(C)^n.\, \tilde{\mathbf{op}}(\boldsymbol{x}) = \rho(\mathbf{op}(\boldsymbol{x}))$
- precise if $\forall \boldsymbol{x} \in C^n.\, \tilde{\mathbf{op}}(\rho(\boldsymbol{x})) = \rho(\mathbf{op}(\boldsymbol{x}))$.

The optimal abstract operator corresponding to $\mathbf{op}$ is precise if and only if

$$\forall \boldsymbol{x} \in C^n.\ \rho(\mathbf{op}(\rho(\boldsymbol{x}))) = \rho(\mathbf{op}(\boldsymbol{x}))$$

If this happens, we say that ρ is precise w.r.t. to $\mathbf{op}$ or $\mathbf{op}$ is precise over ρ. Clearly, an operator $\mathbf{op}$ can be precise only on a subset of its arguments.

2 Some properties of the lattice of upper closure operators

As already noted, our abstraction classes are characterized by various precision properties w.r.t. semantic operators. Since precision is a general property in abstract interpretation, several interesting problems can better be faced in the general setting, rather than in our instance for logic programs.

In particular, we are interested in studying how precision is affected by operators over abstract domains. There is a vast literature on this subject and there exist several operators, defined to perform different kinds of composition or refinement. Since the set of all the abstractions is a complete lattice, the very first operators which come to mind are the meet and join operator over this lattice. Meet has been widely used for attribute independent analysis in [8] while join, partly because of its intrinsic non-constructive flavour, has not found a similar interest in the research community. However, a better knowledge of the properties of both operators can improve our understanding of the structure of abstraction lattices.

2.1 Meet and join

What we want to know is whether meet and join do preserve precision or, to be more precise, whether meet and join of abstract domains, which are precise w.r.t. some operator **op**, are still precise w.r.t. **op**. The case of meet is quite simple, since it has a constructive and local definition. If ρ_1 and ρ_2 are two different abstractions, let ρ be $\rho_1 \sqcap \rho_2$. Then $\rho(x)$, for every element x, can be derived from the values of $\rho_1(x)$ and $\rho_2(x)$ by means of a simple meet operation over the concrete domain. The following result is almost trivial.

Theorem 2.1 *Let* $\mathbf{op} : C^n \to C$ *be a monotonic function and* $\{\rho_i\}_{i\in I} \subseteq \mathrm{uco}(C)$ *be precise w.r.t.* **op**. *Then* $\rho = \bigsqcap_{i\in I} \rho_i$ *is precise w.r.t.* **op**.

Moreover, if the ρ_i are precise only on a subset X of C, then ρ is precise on that subset too. One might guess that the same theorem still holds if we replace $\sqcap$ by $\sqcup$. Unfortunately, this is not the case, as shown by the following example.

Example 2.2 Take the following concrete domain

$$C = \omega + 2 = \{0, 1, \ldots, n, \ldots, \omega, \omega + 1\}$$

and let ρ_1 and ρ_2 be two abstractions such that $\rho_1(C) = \{2n \mid n \in \mathbb{N}\} \cup \{\omega, \omega + 1\}$ and $\rho_2(C) = \{2n + 1 \mid n \in \mathbb{N}\} \cup \{\omega, \omega + 1\}$. Then consider the unary operator $\mathbf{op} : C \to C$ defined as

$$\mathbf{op}(x) = \begin{cases} \omega & \text{if } x < \omega \\ \omega + 1 & \text{otherwise} \end{cases}$$

It's easy to verify that **op** is precise over both ρ_1 and ρ_2. In the ρ_1 case, we have

$$(\rho_1 \circ \mathbf{op} \circ \rho_1)(x) = \begin{cases} (\rho_1 \circ \mathbf{op})(x) & \text{if } x \in \rho_1(C) \\ \rho_1(\mathbf{op}(x + 1)) = \rho_1(\omega) = (\rho_1 \circ \mathbf{op})(x) & \text{otherwise} \end{cases}$$

and the ρ_2 case is similar. Nevertheless, **op** is not precise over $\rho = \rho_1 \sqcup \rho_2$. In fact, for every $n \in \mathbb{N}$

$$(\rho \circ \mathbf{op} \circ \rho)(n) = \rho(\mathbf{op}(\omega)) = \omega + 1$$

but

$$(\rho \circ \mathbf{op})(n) = \rho(\omega) = \omega$$

and so $\rho \circ \mathbf{op} \circ \rho \neq \rho \circ \mathbf{op}$. ■

Moreover, if $\rho = \rho_1 \sqcup \rho_2$, $\rho(x)$ needs to be computed from the images of ρ_1 and ρ_2, possibly infinite sets, in an essentially non-constructive way. For this reason it is much more difficult to study the join rather than the meet operator.

However, we can strengthen our hypothesis by requiring **op** to be additive rather than monotonic. In this case, we will be able to prove the precision of $\bigsqcup_{i\in I} \rho_i$. The proof can better be based on the following constructive characterization of the join operator.

Theorem 2.3 *Let $\{\rho_i\}_{i\in I} \subseteq \mathrm{uco}(C)$ and $T : C \to C$ be the operator*

$$T(x) = \bigsqcup_{i\in I} \rho_i(x).$$

Then

$$\Big(\bigsqcup_{i\in I} \rho_i\Big)(x) = \bigsqcap\{y \in C \mid T(y) = y \wedge y \sqsupseteq x\} = T^{\alpha}(x)$$

for some ordinal α.

In short, $\rho(x)$ is the least fix point of T greater than x. Theorem 2.3 gives a constructive way of computing values of the joint abstraction. However, α is in general greater than ω. Therefore, apart from the case of finite abstract domains, the computation is nonterminating. However, Theorem 2.3 is still useful to prove, by transfinite induction, the following

Theorem 2.4 *Let* $\mathbf{op} : C^n \to C$ *be an additive function and $\{\rho_i\}_{i\in I} \subseteq \mathrm{uco}(C)$ be precise w.r.t.* **op**. *Then $\rho = \bigsqcup_{i\in I} \rho_i$ is precise w.r.t.* **op**.

Furthermore, although still not proved, we think that if additiveness condition is relaxed to continuity the above theorem still holds.

2.2 Functional dependencies

If the meet operator is used for attribute independent analysis, the operator of functional dependencies provides a systematic approach to build new abstract domains, which has been first exploited in [8] as domain for attribute dependent analyses. In [8] it was essentially the domain of monotonic functions between two abstract domains. In [15] this definition was extended by introducing a concrete binary operator which encodes the data-dependencies between two different abstract interpretations and some applications to logic programs as been shown.

Let us first recall the definition of functional dependencies operator [15], adapted to our formalization by means of upper closure operators.

Definition 2.5 (Functional dependencies operator) *Let ρ_1 and ρ_2 be two* uco*'s over the complete lattice C and $\odot$ be a left-additive binary operator over C. Then, we define*

$$(\rho_1 \to^{\odot} \rho_2)(x) = \bigsqcup\{x' \in C \mid \forall y \in \rho_1(C).\rho_2(x' \odot y) \sqsubseteq \rho_2(x \odot y)\}.$$

It is possible to show that $\rho_1 \to^{\odot} \rho_2$ is a closure operator over C. If $\rho_1 = \rho_2 = \rho$, then we denote $\rho \to^{\odot} \rho$ by $Dep^{\odot}\,\rho$, and call it autodependencies operator.

We might expect the functional dependencies operator not only to preserve but also to improve the precision. Neither of these expectations can fully be satisfied. However, there are some results in both directions.

First of all, it is interesting to know which is the accuracy of $\rho_1 \to^{\odot} \rho_2$ w.r.t. the accuracy of ρ_1 and ρ_2.

Theorem 2.6 *If $\odot$ is left-precise over ρ_2, then $\rho_1 \to^{\odot} \rho_2 \sqsupseteq \rho_2$.*

Theorem 2.7 *If there exists $y \in \rho_1(C)$ such that $x \odot y = x$ for all $x \in C$, then $\rho_1 \to^{\odot} \rho_2 \sqsubseteq \rho_2$.*

Hence $\rho_1 \to^{\odot} \rho_2$ can be both more or less accurate of ρ_2. The following theorem gives a strict lower bound on the accuracy of functional dependencies.

Theorem 2.8

$$\rho_1 \to^{\odot} \rho_2 \ \sqsupseteq\ \bigsqcup\{\rho' \sqsubseteq \rho_2 \mid \rho' \ \odot \textit{ is left-precise in } \rho' \ \}$$

Finally, we can give some conditions under which $\odot$ is precise over $\rho_1 \to^{\odot} \rho_2$.

Theorem 2.9 *Let $\rho' = Dep^{\odot}\rho$ with $\rho \in \mathrm{uco}(C)$. Let $\odot$ be a left-additive and right-precise operator on ρ and $\rho' \sqsupseteq \rho$. Then $\odot$ is left-precise on ρ'.*

Theorem 2.10 *Let $\rho' = Dep^{\odot}\rho$ with $\rho \in \mathrm{uco}(C)$. Let **op** be an n-ary additive operator over C. If for each $y_1, z_1, \ldots, y_n, z_n \in C$,*

$$\big(\forall x \in \rho(C).\ \rho(y_1 \odot x) \sqsubseteq \rho(z_1 \odot x) \wedge \cdots \wedge \rho(y_n \odot x) \sqsubseteq \rho(z_n \odot x)\big) \Longrightarrow$$
$$\big(\forall x \in \rho(C).\ \rho(\mathbf{op}(y_1, \ldots, y_n) \odot x) \sqsubseteq \rho(\mathbf{op}(z_1, \ldots, z_n) \odot x)\big)$$

then ρ' is precise w.r.t. **op**.

Corollary 2.11 *Let $\rho' = Dep^{\odot}\rho$, with $\odot$ associative and right-precise on ρ. Then $\odot$ is precise on ρ'.*

As one can easily note, the properties which hold in the case of functional dependencies are much less general than in case of the meet and join operators, and require a lot of additional hypotheses. Nevertheless, this is often enough to derive in a systematic way abstract domains which are precise w.r.t. a given operator **op**, as we will see in the logic programming setting.

3 A semantic framework for logic programs

In order to discuss abstraction in logic programming we need to choose a concrete domain and the related operational and denotational semantics. In this paper we will use the semantic framework introduced in [3] and developed in [4] and [5]. Here we recall only the main definitions and results.

3.1 Basic framework

In this framework we are able to reason about compositional properties of SLD derivations and their abstractions (observables) in the case of definite logic programs. An operational and a denotational semantics are defined, both of them expressed in terms of basic semantic operators on the concrete domain, which represents SLD trees up to renaming of mgu's and clauses.

The denotational semantics is characterized by a different semantic function for every syntactical category in the language:

$$\begin{array}{lll}
\mathcal{Q} : \mathit{QUERY} & \longrightarrow & \mathbb{D}, \\
\mathcal{G} : \mathit{GOAL} & \longrightarrow & (\mathbb{I} \to \mathbb{D}), \\
\mathcal{A} : \mathit{ATOM} & \longrightarrow & (\mathbb{I} \to \mathbb{D}), \\
\mathcal{P} : \mathit{PROG} & \longrightarrow & (\mathbb{I} \to \mathbb{I}), \\
\mathcal{C} : \mathit{CLAUSE} & \longrightarrow & (\mathbb{I} \to \mathbb{I}),
\end{array}$$

where $\mathbb{D}$ is the set of *collections* and $\mathbb{I}$ is the set of *interpretations*. A collection is the flat representation of a family of SLD trees. Every SLD tree is represented by the set of all the SLD derivations, modulo renaming of mgu's and clauses, obtained following a path from the root to another node. An interpretation is a collection for only pure atomic goals modulo variance. *QUERY* is the syntactical category corresponding to statements of the form "$\boldsymbol{G}$ `in` P" where $\boldsymbol{G}$ is a goal and P is a program.

These are the definitions of the semantic functions:

$$\begin{array}{rcll}
\mathcal{Q}[\![\boldsymbol{G} \texttt{ in } P]\!] & = & \mathcal{G}[\![\boldsymbol{G}]\!]_{\mathrm{lfp}\,\mathcal{P}[\![P]\!]} & (3.1) \\
\mathcal{G}[\![\Box]\!]_I & = & \mathit{Id}|_{\Box} & (3.2) \\
\mathcal{G}[\![A, \boldsymbol{G}]\!]_I & = & \mathcal{A}[\![A]\!]_I \times \mathcal{G}[\![\boldsymbol{G}]\!]_I & (3.3) \\
\mathcal{A}[\![A]\!]_I & = & A \cdot I & (3.4) \\
\mathcal{P}[\![\emptyset]\!]_I & = & \mathit{Id}|_{\mathbb{I}} & (3.5) \\
\mathcal{P}[\![\{c\} \cup P]\!]_I & = & \left[\mathcal{C}[\![c]\!]_I + \mathcal{P}[\![P]\!]_I\right]_{\equiv} & (3.6) \\
\mathcal{C}[\![p(\boldsymbol{t}) \leftarrow \boldsymbol{B}]\!]_I & = & \left[\mathrm{tree}(p(\boldsymbol{t}) \leftarrow \boldsymbol{B}) \bowtie \mathcal{G}[\![\boldsymbol{B}]\!]_I\right]_{\equiv} & (3.7)
\end{array}$$

The informal meaning of the semantic operators is the following. The $\cdot$ operator "solves" an atomic goal A in an interpretation I, $\times$ computes the and-conjunction of two interpretations and $\bowtie$ computes the interpretations obtained by replacement. Finally, $\sum$ computes the non-deterministic union of a class of interpretations. Note that when the class is finite, we use the infix notation $+$. Moreover, $\mathrm{tree}(c)$ is a tree representation of the clause c and *Id* is the family of all the SLD trees of depth equals to zero. We can also define the fix-point denotation of a program P as

$$\mathcal{F}[\![P]\!] = \mathrm{lfp}\,\mathcal{P}[\![P]\!] = \mathcal{P}[\![P]\!] \uparrow \omega \tag{3.8}$$

Operational semantic is build, using the same semantic operators, from the transition system $\mathcal{T} = (\mathbb{D}, \overset{P}{\longmapsto})$, with the following transition rule

$$\frac{D \in \mathbb{D},\ D \neq D \bowtie \mu(\mathrm{tree}(P))}{D \overset{P}{\longmapsto} D \bowtie \mu(\mathrm{tree}(P))} \tag{3.9}$$

where

$$\mu(D) = \sum \{(A \cdot D|_{\mathbb{I}}) \times \mathit{Id}\}_{A \in \mathit{Atoms}} \tag{3.10}$$

is a kind of *sequential unfolding* operator. Using $\mathcal{T}$ we can define the *behavior* (operational semantic) of goals as

$$\mathcal{B}[\![\boldsymbol{G} \texttt{ in } P]\!] = \sum \{ D \mid Id|_{\boldsymbol{G}} \overset{P}{\longmapsto}^{*} D \} \tag{3.11}$$

and the top-down denotation of a program P as

$$\mathcal{O}[\![P]\!] = \left[\sum \{ \mathcal{B}[\![p(\boldsymbol{x}) \texttt{ in } P]\!] \}_{p \in \Pi} \right]_{\equiv} \tag{3.12}$$

As the intuition suggests the transition system $\mathcal{T}$ defines the usual notion of SLD derivation, so that

$$\mathcal{B}[\![\boldsymbol{G} \texttt{ in } P]\!] = \{ [d]_{\overset{\text{der}}{\approx}} \mid d = \boldsymbol{G} \xrightarrow[P]{\theta}^{*} \boldsymbol{B} \}$$

where $\overset{\text{der}}{\approx}$ is equality up to renaming of mgu's and clauses. Finally, we define the equivalence $\approx$ between two programs P_1 and P_2 as the equivalence of the behaviors of the two programs, i.e.

$$P_1 \approx P_2 \Leftrightarrow \forall \boldsymbol{G} \in \text{Goals}, \mathcal{B}[\![\boldsymbol{G} \texttt{ in } P_1]\!] = \mathcal{B}[\![\boldsymbol{G} \texttt{ in } P_2]\!] \tag{3.13}$$

In this framework, operational and denotational semantics enjoy several interesting properties, which are stated below

- Operational behavior is compositional

  $$\mathcal{B}[\![A \texttt{ in } P]\!] = A \cdot \mathcal{O}[\![P]\!],$$
  $$\mathcal{B}[\![(\boldsymbol{G}_1, \boldsymbol{G}_2) \texttt{ in } P]\!] = \mathcal{B}[\![\boldsymbol{G}_1 \texttt{ in } P]\!] \times \mathcal{B}[\![\boldsymbol{G}_2 \texttt{ in } P]\!]$$

- Operational semantic is correct and minimal

  $$P_1 \approx P_2 \Longleftrightarrow \mathcal{O}[\![P_1]\!] = \mathcal{O}[\![P_2]\!]$$

- Operational semantic is OR-compositional

  $$\mathcal{O}[\![P_1 \cup P_2]\!] = \mathcal{O}[\![P_1]\!] \uplus \mathcal{O}[\![P_2]\!]$$

  where $\uplus$ is an appropriate semantic operator

- Operational and denotational semantics are equal

  $$\mathcal{O}[\![P]\!] = \mathcal{F}[\![P]\!]$$
  $$\mathcal{Q}[\![\boldsymbol{G} \texttt{ in } P]\!] = \mathcal{B}[\![\boldsymbol{G} \texttt{ in } P]\!]$$

3.2 Abstraction framework

We will use uco's over collections to model observables, taking care of the fact that we want to abstract only one SLD tree at the time and not an entire family of SLD trees. For this reason, we call *observable* an uco over $\mathbb{D}$ such that:

- $\rho(\emptyset) = \emptyset$,
- $\rho|_{\text{WFS}_{\boldsymbol{G}}}$ is an uco over $\text{WFS}_{\boldsymbol{G}}$ for all $\boldsymbol{G} \in \text{Goals}$,

- $D \equiv D' \Rightarrow \rho(D) \equiv \rho(D')$

where WFS_G is the set of all SLD trees for the goal $\boldsymbol{G}$ and $\equiv$ is equality of SLD trees up to renaming of initial goals. Often we will define an observable ρ by an auxiliary operator

$$\rho^* : \bigcup_{\boldsymbol{G}} \mathrm{WFS}_{\boldsymbol{G}} \to \bigcup_{\boldsymbol{G}} \mathrm{WFS}_{\boldsymbol{G}}$$

which abstracts SLD trees. From ρ^* we can define ρ as follows:

$$\rho(D) = \bigcup_{\boldsymbol{G}} \rho^*(D \cap \top_{\mathrm{WFS}_{\boldsymbol{G}}})$$

By means of standard abstract interpretation techniques we can derive the abstract semantics, replacing semantic operators seen above with their optimal abstract counterpart. We can define two major classes of observables, according to precision w.r.t. the semantic operators. The first class is that of *perfect observables*, for which

$$\begin{aligned} \rho(A \cdot D) &= \rho(A \cdot \rho(D)) && (3.14)\\ \rho(D_1 \times D_2) &= \rho(\rho(D_1) \times \rho(D_2)) && (3.15)\\ \rho(D_1 \bowtie D_2) &= \rho(\rho(D_1) \bowtie \rho(D_2)) && (3.16) \end{aligned}$$

Abstract semantics for perfect observables enjoy all the properties we have already seen in the concrete case. Moreover, abstract semantics are precise.

$$\rho(\mathcal{B}[\![\boldsymbol{G} \texttt{ in } P]\!]) = \mathcal{B}_\rho[\![\boldsymbol{G} \texttt{ in } P]\!] = \rho(\mathcal{Q}[\![\boldsymbol{G} \texttt{ in } P]\!]) = \mathcal{Q}_\rho[\![\boldsymbol{G} \texttt{ in } P]\!]$$
$$\rho(\mathcal{O}[\![P]\!]) = \mathcal{O}_\rho[\![P]\!] = \rho(\mathcal{F}[\![P]\!]) = \mathcal{F}_\rho[\![P]\!]$$

We can relax axiom (3.16) by requiring the precision of $\bowtie$ only on the second argument:

$$\rho(D_1 \bowtie D_2) = \rho(D_1 \bowtie \rho(D_2)) \qquad (3.17)$$

In this case we speak of *denotational observables.* We can still obtain a precise denotational semantics using the abstract optimal counterpart of $\mathcal{C}$ as abstract semantic function for clauses. We have as a result:

$$\rho(\mathcal{Q}[\![\boldsymbol{G} \texttt{ in } P]\!]) = \mathcal{Q}_\rho[\![\boldsymbol{G} \texttt{ in } P]\!]$$
$$\rho(\mathcal{F}[\![P]\!]) = \mathcal{F}_\rho[\![P]\!]$$

and the following relations between operational and denotational semantics:

$$\mathcal{F}_\rho[\![P]\!] \sqsubseteq \mathcal{O}_\rho[\![P]\!]$$
$$\mathcal{Q}_\rho[\![\boldsymbol{G} \texttt{ in } P]\!] \sqsubseteq \mathcal{B}_\rho[\![\boldsymbol{G} \texttt{ in } P]\!]$$

3.3 Operational observables

We can improve the semantic framework by adding a new abstraction class, that of operational observables. They can be obtained from perfect observables by relaxing the precision condition of axiom (3.16) as in denotational observables, but requiring the precision on the left rather than on the right argument.

Definition 3.1 *Let $\rho \in \operatorname{uco}(\mathbb{D})$ be an observable. Then ρ is an operational observable if*

$$\begin{aligned}
\rho(A \cdot D) &= \rho(A \cdot \rho(D)) \\
\rho(D_1 \times D_2) &= \rho(\rho(D_1) \times \rho(D_2)) \\
\rho(D_1 \bowtie D_2) &= \rho(\rho(D_1) \bowtie D_2)
\end{aligned} \tag{3.18}$$

The definition of operational observables is symmetric w.r.t. that of the denotational ones, and the same is true for the properties they enjoy. Hence, it is not surprising the following

Theorem 3.2 *If ρ is a denotational and operational observable, then it is a perfect observable.*

The relaxed properties of operational observables allow us to define an abstract operational semantic, characterized by the following slight variation of the original transition rule:

$$\frac{X \in \rho(\mathbb{D}), \quad X \neq X \mathbin{\tilde{\bowtie}} \mu(\text{tree}(P))}{X \xmapsto{P}_{\rho} X \mathbin{\tilde{\bowtie}} \mu(\text{tree}(P))},$$

where $\tilde{\bowtie}$: $\rho(\mathbb{D}) \times \mathbb{D} \to \rho(\mathbb{D})$ is defined as $X \mathbin{\tilde{\bowtie}} D = \rho(X \bowtie D)$. The trick is to use the $\bowtie$ operator in such a way that its second argument is always taken before abstracting.

The operational semantic of operational observables enjoys all the properties that the denotational semantic has in the case of denotational observables, namely the compositionality properties shown by Theorem 3.3 and the precision properties of Theorem 3.4.

Theorem 3.3 *Let ρ be an operational observable, A be an atom, $\boldsymbol{G}, \boldsymbol{G}_1\boldsymbol{G}_2$ be goals and P be a program. Then*

1. $\mathcal{B}_\rho[\![A \ \textbf{\textit{in}} \ P]\!] = A \mathbin{\tilde{\cdot}} \mathcal{O}_\rho[\![P]\!]$
2. $\mathcal{B}_\rho[\![\boldsymbol{G}_1, \boldsymbol{G}_2 \ \textbf{\textit{in}} \ P]\!] = \mathcal{B}_\rho[\![\boldsymbol{G}_1 \ \textbf{\textit{in}} \ P]\!] \mathbin{\tilde{\times}} \mathcal{B}_\rho[\![\boldsymbol{G}_2 \ \textbf{\textit{in}} \ P]\!]$

Theorem 3.4 *Let ρ be an operational observable, $\boldsymbol{G} \in$ Goals and $P \in$ Progs. Then*

1. $\rho(\mathcal{B}[\![\boldsymbol{G} \ \textbf{\textit{in}} \ P]\!]) = \mathcal{B}_\rho[\![\boldsymbol{G} \ \textbf{\textit{in}} \ P]\!]$
2. $\rho(\mathcal{O}[\![P]\!]) = \mathcal{O}_\rho[\![P]\!]$

The following corollary states the correctness and full abstraction of the operational semantic.

Corollary 3.5 *Let ρ be an operational observable and P_1, P_2 be programs. Then*

$$P_1 \approx_\rho P_2 \Leftrightarrow \mathcal{O}_\rho[\![P_1]\!] = \mathcal{O}_\rho[\![P_2]\!]$$

The denotational semantic is still correct, being derived using abstract interpretation. However it is in general less accurate than the operational one, as stated below.

Corollary 3.6 *Let ρ be an operational observable, P be a program and $\boldsymbol{G}$ be a goal. Then*

1. $\mathcal{O}_\rho[\![P]\!] \sqsubseteq \mathcal{F}_\rho[\![P]\!]$,

2. $\mathcal{B}_\rho[\![\boldsymbol{G}\ \textit{in}\ P]\!] \sqsubseteq \mathcal{Q}_\rho[\![\boldsymbol{G}\ \textit{in}\ P]\!]$.

We conclude by showing one example of operational observable. Roughly speaking, an observable is operational when it keeps some information which cannot be computed in a bottom-up way. For example,

$$\rho(S) = \mathrm{gwf}\{d \in \mathrm{Derivs}_{/\overset{\mathrm{der}}{\approx}} \mid \exists d' \in S \text{ such that } d \propto d'\}$$

with

$$\begin{aligned} d \propto d' \iff & d = \boldsymbol{G}_0 \to \ldots \to \boldsymbol{G}_m,\ d' = \boldsymbol{G}'_0 \to \ldots \to \boldsymbol{G}'_k, \\ & \text{there exist } p \text{ and } \boldsymbol{G} \text{ s.t. } \boldsymbol{G}'_0 = \boldsymbol{G}_0 \leq (p(\boldsymbol{x}), \boldsymbol{G}), \\ & \text{with } \boldsymbol{x} \text{ renamed apart from } d \text{ and } d', \\ & \mathrm{result}(d) \equiv \mathrm{result}(d'), \\ & \big(\boldsymbol{G} \neq \Box \text{ or } \#\{i \mid \mathrm{first}(\boldsymbol{G}_i) \leq p(\boldsymbol{x})\} = \#\{i \mid \mathrm{first}(\boldsymbol{G}'_i) \leq p(\boldsymbol{x})\}\big), \end{aligned}$$

where $\mathrm{gwf}(X)$ is the least SLD tree containing the SLD derivations in X and $\#X$ is the cardinality of the set X. The observable ρ is a concretization of computed resultants. When the initial goal is atomic, it counts how many times the same predicate of this goal is called in the derivation.

4 The lattice of observables

We know that the set of abstractions is a complete lattice. However, in the logic programming case, observables are a subset of all the abstractions. Hence we have to prove they are still a lattice.

Theorem 4.1 *For each $i \in I$, let ρ_i be an observable. Then $\bigsqcup_{i\in I} \rho_i$ and $\bigsqcap_{i\in I} \rho_i$ are observables.*

From theorems 2.4 and 2.1 we immediately derive our first result concerning the lattice of observables.

Theorem 4.2 *The sets of denotational, operational and perfect observables are complete lattices.*

Now, if we compose different observables by means of the meet and join operators finding their most abstract common concretizations or most concrete common abstractions, we know that the result will be in the same class in which we took the operands.

However our understanding of the lattice of observables is still far from being satisfactory. In particular, two questions arise.

1. are the denotational, operational and perfect classes able to characterize all the observables which enjoy the related properties?

2. which are the infimum and supremum of our classes of observables?

The first question has a negative answer. There are examples of observables which are not perfect (and neither denotational nor operational) and still enjoy all the properties of perfect observables. Consider, for example

$$\rho(D) = D + \left\{t(\boldsymbol{x}) \xrightarrow{\epsilon}{}^{n} t(\boldsymbol{x}) \mid t(\boldsymbol{x}) \xrightarrow{\epsilon} t(\boldsymbol{x}) \in D\right\},$$

which looks like the trivial observable $\rho(D) = D$ with some added useless derivations, which are obviously generated from both the operational and the denotational semantics. It can be shown that none of our semantic operators is precise on it. Nevertheless its compositional and accuracy properties are those of perfect observables.

As far as the second question is concerned, we note that $\rho_\top = \top_{\mathbb{D}}$ and $\rho_\perp = Id$ are perfect observables (and therefore also denotational and operational). Hence they are the infimum and supremum we looked for. However we are not really interested in these trivial observables. Hence we look for infimum and supremum in the set of all the observables but $\rho_\top$ and $\rho_\perp$. If we consider perfect observables, the intuition says that the infimum should be an observable similar to ground resultants. Unfortunately, this is not the case. The problem seems to be related to the existence of perfect observables which have nothing to do with known semantics for logic programs. Consider, for instance, the observable of Example 4.3.

Example 4.3 Given the predicate symbol t, define

$$\rho(D) = D \bowtie \left\{d \in \text{Derivs}_{/\overset{\text{der}}{\approx}} \mid \exists \theta, \boldsymbol{G} \text{ such that } \text{first}(d) = (t(\boldsymbol{x})\theta, \boldsymbol{G})\right\}$$

The idea of this observable is that there exists one predicate, t, which causes the loss of every subsequent information on the computation. We can prove that ρ is perfect. However it is not comparable with the ground resultant observable, that we thought strictly related to all the observables in that class. ■

We can partially solve the above problem by considering only those observables which are concretizations of the observable ρ_h, defined as follows.

$$\rho_h^*(S) = \begin{cases} \emptyset & \text{if } S = \emptyset \\ \{\boldsymbol{G} \longrightarrow^* \boldsymbol{B} \mid \boldsymbol{B} \neq \Box\} & \text{if } S \neq \emptyset \text{ doesn't contain refutaions} \\ \top_{\text{WFS}_{\boldsymbol{G}}} & \text{otherwise} \end{cases}$$

ρ_h is the observable corresponding to the least Herbrand model semantic. Therefore, the above constraint is the same as considering only the collecting semantics, according to [14]. Now that all the "strange" abstractions are cut off, we can prove the following theorem.

Theorem 4.4 *The most abstract of all the denotational observables which are concretization of ρ_h is the observable of ground computed answers ρ_g, defined as follow*

$$\rho_g^*(S) = \Big\{d = \boldsymbol{G} \xrightarrow{\theta}{}^* \boldsymbol{B} \mid \boldsymbol{B} = \Box \Rightarrow \big(\forall \boldsymbol{G}' \leq \boldsymbol{G}\theta,\ \boldsymbol{G}' \textit{ ground} \Rightarrow \\ \exists d' = \boldsymbol{G} \xrightarrow{\theta'}{}^* \Box \in S \textit{ such that } \boldsymbol{G}' \leq \boldsymbol{G}\theta'\big)\Big\} \tag{4.1}$$

4.1 Functional dependencies

While the meet and join operators are strictly related to the lattice of observables, they are not really useful in deriving new abstract domains, if we exclude the trivial case of attribute independent analysis. This is better accomplished by other operators, such as functional dependencies, disjunctive completion and so on.

In the rest of the paper we will study the functional dependencies operator in the setting of logic programming. The first thing to prove, as in the meet and join case, is that the application of the operator really brings to abstractions which are indeed observables.

Theorem 4.5 *Let ρ_1 and ρ_2 be observables, $\odot$ be a left-additive binary operator over $\mathbb{D}$, such that*

1. *if $S \in WFS_{\boldsymbol{G}}$, then $S \odot X \in WFS_{\boldsymbol{G}}$, for all $X \in \rho_1(\mathbb{D})$,*
2. *$D_1 \odot D_2 = \emptyset$ if and only if $D_1 = \emptyset$.*
3. *$S \equiv S'$ implies $S \odot X \equiv S' \odot X$, for each $X \in \rho_1(\mathbb{D})$.*

Then $\rho_1 \to^{\odot} \rho_2$ is an observable.

A really interesting case is that of autodependencies w.r.t. the operator $\bowtie$, which fully satisfies all the properties of the above theorem. We can prove the following theorem.

Theorem 4.6 *If ρ is an observable, then $Dep^{\bowtie} \rho$ is an observable too, and enjoys the following properties:*

1. *$Dep^{\bowtie} \rho$ is a refinement of ρ,*
2. *if ρ is operational, then $Dep^{\bowtie} \rho = \rho$,*
3. *if ρ is denotational, then $Dep^{\bowtie} \rho$ is the most abstract among all the observables which are concretization of ρ and for which $\bowtie$ is left-precise. Moreover, $\bowtie$ is also right-precise on $Dep^{\bowtie} \rho$.*

For example, in [15] has been shown, although in a different setting, that functional autodependencies with respect to $\bowtie$ of the computed answers observable is the observable of computed resultants. Moreover, follows from Theorem 4.6 and from the properties of the observable ρ_g that $\mathrm{Dep}^{\bowtie} \rho_g$ is the most abstract perfect observable among all the observables that are concretizations of the least Herbrand's model.

5 Conclusions and Future Works

This paper was originated by an attempt to better understand the structure of the lattice of observables in logic programs. Because this is strongly related to the precision of the semantic operators, we have first analyzed the problem in a general abstract interpretation setting, when possible, and then applied the results to the case of logic programs.

This line of research can be further pursued by studying other operators on observables or by studying how the precision of each semantic operator affects the properties of observables. What we expect is that $\cdot$, $\times$ and $\bowtie$ are related to compositionality w.r.t procedure call, goal-composition and program union.

However, it is probably more interesting to take into account the classes of observables which are used for static analyses. In this case we want to find a set of operators on observables by which systematically derive new abstract domains, enjoying desired properties of precision and/or compositionality. For example, starting with the basic abstract domain for groundness, which simply states whether a variable is ground or not, we could automatically build more complex observables, like *Def* or *Pos*. Moreover, if these operators were constructive, we could be able to develop a static analysis software in which abstract domains are specified giving the basic domain of interest and the desired properties. All the remaining tasks, such as deciding how to compose the basic abstract domains to satisfy the request of the user or performing the effective analysis, would be a matter of the software.

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 and The MIT Press, 1990.

[2] G. Birkhoff. Lattice Theory. In *AMS Colloquium Publication, third ed.*, 1967.

[3] M. Comini, G. Levi, and M. C. Meo. Compositionality of *SLD*-derivations and their abstractions. In J. Lloyd, editor, *Proceedings of the 1995 Int'l Symposium on Logic Programming*, pages 561–575. The MIT Press, 1995.

[4] M. Comini, G. Levi, and M. C. Meo. A theory of observables for logic programs. Submitted for publication, 1996.

[5] M. Comini and M. C. Meo. Compositionality properties of *SLD*-derivations. Submitted for publication, 1996.

[6] A. Cortesi, G. Filè, R. Giacobazzi, C. Palamidessi, and F. Ranzato. Complementation in abstract interpretation. In A. Mycroft, editor, *Proc. of Int. Static Analysis Symposium, (SAS'95)*, volume 983 of *Lecture Notes in Computer Science*, pages 100–117, Glashow, UK, 1995. Springer-Verlag, Berlin.

[7] P. Cousot and R. Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proc. Fourth ACM Symp. Principles of Programming Languages*, pages 238–252, 1977.

[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] P. Cousot and R. Cousot. Abstract Interpretation and Applications to Logic Programs. *Journal of Logic Programming*, 13(2 & 3):103–179, 1992.

[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. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. A Model-Theoretic Reconstruction of the Operational Semantics of Logic Programs. *Information and Computation*, 102(1):86–113, 1993.

[12] M. Gabbrielli, G. Levi, and M. C. Meo. Resultants semantics for PROLOG. *Journal of Logic and Computation*, 6(4):491–521, 1996.

[13] 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, 1992.

[14] R. Giacobazzi. "optimal" collecting semantics for analysis in a hierarchy of logic program semantics. In C. Puech and R. Reischuk, editors, *Proc. 13th International Symposium on Theoretical Aspects of Computer Science (STACS'96)*, volume 1046 of *Lecture Notes in Computer Science*, pages 503–514. Springer-Verlag, 1996.

[15] R. Giacobazzi and F. Ranzato. Functional dependencies and moore-set completions of abstract interpretations and semantics. In J. Lloyd, editor, *Proc. 1995 Int'l Symposium on Logic Programming (ILPS'95)*, pages 321–335, Portland, OR, 1995. The MIT Press.

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

[17] M.C. Meo. *A framework for reasoning about the Semantics of Logic Program.* PhD thesis, Dipartimento di Informatica, Università di Pisa, 1996.