Complementing Logic Program Semantics

Roberto Giacobazzi[⋆] Francesco Ranzato[⋆⋆]

[⋆] *Dipartimento di Informatica, Università di Pisa*
Corso Italia 40, 56125 Pisa, Italy
giaco@di.unipi.it

[⋆⋆] *Dipartimento di Matematica Pura ed Applicata, Università di Padova*
Via Belzoni 7, 35131 Padova, Italy
franz@math.unipd.it

Abstract. We consider abstract interpretation, and in particular the basic operators of reduced product and complementation of abstract domains, as a tool to systematically derive denotational semantics by composition and decomposition. Reduced product allows to perform the logical conjunction of semantics, while complementation characterizes what is left from a semantics when the information concerning a given observable property is "subtracted" from it. We apply this idea to the case of logic programming, characterizing in a uniform algebraic setting, the interaction between a number of well known declarative semantics for logic programs.

1 Introduction

Abstract interpretation ([6, 7]) is a well established theory for semantics approximation, which is traditionally applied in the field of program analysis, e.g. for the design of program analysis frameworks. In POPL'92 ([8]), Cousot and Cousot showed that abstract interpretation can also be used to systematically derive more abstract semantics, as well as approximate semantics, from a given ground (or standard) semantics for the considered programming language. This is the case for denotational and relational semantics, which can be derived by abstraction from a rule-based operational semantics (e.g. SOS) for the language. Abstract interpretation provides therefore both a mathematical framework where different semantics at different levels of abstraction can be compared to each other, and a number of systematic methods to specify new semantic definitions from more concrete ones by abstraction. As observed in [14], among these methods, the operations for abstract domain refinement and their inverses, like *reduced product* ([7]), *complementation* ([5]), *disjunctive completion* ([7]), *least disjunctive basis* ([20]), and *reduced power* for functional dependencies ([7, 19]), may play a central role in semantics design, as well as in analysis, in order to provide formal algebraic methods to manipulate and compare semantics. In this work, we show how *reduced product*, and its inverse operation which is *complementation*, can be used to study the semantics of programming languages. The idea is that new, as well as known semantics for programming languages, can be studied and derived in a systematic way by applying the algebraic operations of domain refinements and their inverses, like reduced product and complementation. Abstract interpretation provides here the right framework in order to include these algebraic operations into a lattice structure, where each semantics corresponds to a suitable abstraction, and semantics can be compared and composed by comparing and composing the corresponding abstractions. Here, a semantics $\mathcal{S}$ is finer (more precise) than a semantics $\mathcal{A}$, if $\mathcal{A}$ can be derived by abstraction from $\mathcal{S}$.

The reduced product is probably the most common and widely known operation for domain composition in program analysis, and it is defined as the reduced cardinal product of abstract domains. We show that the reduced product of two semantics $\mathcal{S}$ and $\mathcal{T}$ is a new semantics inducing a finer equivalence relation on programs, which is precisely the intersection of the equivalences for $\mathcal{S}$ and $\mathcal{T}$. Complementation corresponds to the natural inverse operation for reduced product, namely an operation which, starting from any two semantics $\mathcal{S}$ and $\mathcal{T}$, with $\mathcal{T}$ more abstract than $\mathcal{S}$, gives as result the most abstract semantics whose reduced product with $\mathcal{T}$ is exactly $\mathcal{S}$. Semantics, analogously to domains for analysis, can be decomposed by complementation, providing the least amount of information which is necessary to upgrade a given semantics by reduced product. Thus, complementation is a relevant tool for semantics decomposition into simpler factors, which turns out to be useful to understand the internal structure of a semantics, and the interaction between its factors. Also, semantics decomposition may be important in order to exploit modularity in program verification problems: Instead of proving properties for complex semantics definitions, we can more easily prove that they hold for the corresponding factors, provided that these properties are preserved under reduced product.

In the following of the paper, we will be mainly concerned with *logic programming*. Logic programming is an example of high-level programming language enjoying a simple semantic definition. Due to its semantic simplicity, logic programming is probably the programming language where the abstract interpretation-based approach to semantics design was mostly successful, as shown by the increasing literature in this field (cf. [3, 4, 17, 19]). Our approach gives rise to a hierarchy of declarative semantics for logic programs where semantics, and the corresponding observable properties, can be manipulated algebraically by complementation and reduced product. After some results in Section 4 on complementation and reduced product in denotational semantics, we show in Sections 5–7 the interaction between well known semantics, like the s-semantics for computed answers of [12], Clark's semantics for correct answers ([2, 12]), least Herbrand model semantics for successful computations ([10]), and the semantics for call patterns in [16]. Logic programming provides here a clean and simple interpretation for the operators of our framework.

2 Preliminaries

Basic Notation. Let A and B be sets. The powerset of A is denoted by $\wp(A)$, the set-difference between A and B is denoted by $A \setminus B$, while $A \subset B$ denotes strict inclusion. A^* denotes the set of finite sequences of objects of A, and sequences are typically denoted by $\langle a_1, ..., a_n \rangle$, or simply $a_1, ..., a_n$, for a_i's symbols in A. The empty sequence is denoted by Λ. Concatenation of sequences $s_1, s_2 \in A^*$ is denoted by $s_1 :: s_2$. The set A equipped with a partial order $\leq$ is denoted by $A_{\leq}$. If A is a poset, we usually denote $\leq_A$ the corresponding partial order. By $x < y$ we mean $x \leq y$ and $x \neq y$. If A is a pre-ordered set and $I \subseteq A$ then $\downarrow I = \{x \in A \mid \exists y \in I.\ x \leq_A y\}$. For $x \in A$, $\downarrow x$ is a shorthand for $\downarrow \{x\}$. $\wp^{\downarrow}(A)$ denotes the set of *order ideals* of A, where

$I \subseteq A$ is an order ideal if $I = \downarrow I$. $\wp^{\downarrow}(A)$ is a complete lattice with respect to set-theoretic inclusion, where the *lub* is set union and the *glb* is set intersection. If A is a poset and $I \subseteq A$, then $Max(I) = \{x \in I \mid \forall y \in I.\ x \leq_A y \Rightarrow x = y\}$. A complete lattice A with partial ordering $\leq$, *lub* $\vee$, *glb* $\wedge$, bottom element $\bot = \vee\emptyset = \wedge A$, and top element $\top = \wedge\emptyset = \vee A$, is denoted by $\langle A, \leq, \vee, \wedge, \bot, \top\rangle$. When A is a lattice, $\vee_A$, $\wedge_A$, $\bot_A$ and $\top_A$ denote the corresponding basic operators and elements. In the following, we will often abuse notation by denoting lattices with their poset notation. We use $\cong$ to denote isomorphism of ordered structures. We write $f : A \to B$ to mean that f is a total function from the set A to the set B. The identity function $\lambda x.x$ is sometimes denoted id. If $C \subseteq A$ then $f(C) = \{f(x) \mid x \in C\}$. By $g \circ f$ we denote the function $\lambda x.g(f(x))$. The set of fixpoints of a function $f : A \to A$ is denoted by $fp(f)$, and, if A is a poset, the least fixpoint is denoted by $lfp(f)$, if it exists.

Abstract Interpretation and Closure Operators. Abstract interpretation is traditionally defined in terms of a pair of adjoint functions which form a Galois connection. If C and D are posets and $\alpha : C \to D$, $\gamma : D \to C$ are monotonic functions such that $\forall c \in C.\ c \leq_C \gamma(\alpha(c))$ and $\forall d \in D.\ \alpha(\gamma(d)) \leq_D d$, then we call the quadruple (C, α, D, γ) a *Galois connection* (G.c.) between C and D. If in addition $\forall d \in D.\ \alpha(\gamma(d)) = d$, then (C, α, D, γ) is a *Galois insertion* (G.i.) of D in C. In the setting of abstract interpretation, C and D are called the *concrete* and the *abstract domain*, and they are assumed to be complete lattices, whereas α and γ are called the *abstraction* and the *concretization* maps, respectively. D is called an *abstraction* (or *abstract interpretation*) of C, and C a *concretization* of D, while D is a *proper* abstraction if $\gamma \circ \alpha \neq \lambda x.x$.

An alternative, but equivalent, approach to abstract interpretation is by closure operators (cf. [7]). Let $\langle L, \leq, \vee, \wedge, \bot, \top\rangle$ be a complete lattice. An (*upper*) *closure operator* on L is an operator $\rho : L \to L$ monotonic, idempotent and extensive (viz. $\forall x \in L.\ x \leq \rho(x)$). Each closure operator ρ is uniquely determined by the set of its fixpoints, which is its image $\rho(L)$. A set $X \subseteq L$ is the set of fixpoints of a closure operator iff X is a *Moore-family* of L, i.e. $\top \in X$ and X is meet-closed (viz. for any non-empty $Y \subseteq X$, $\wedge Y \in X$). In the following, we will often denote a closure operator by the set of its fixpoints. We denote by $\langle uco(L), \sqsubseteq, \sqcup, \sqcap, id, \lambda x.\top\rangle$ the complete lattice of all upper closure operators on the complete lattice L, with bottom element id, top element $\lambda x.\top$, and where for every $\rho, \eta \in uco(L)$, $\{\rho_i\}_{i \in I} \subseteq uco(L)$ and $x \in L$: $\rho \sqsubseteq \eta$ iff $\forall x \in L.\ \rho(x) \leq \eta(x)$, or equivalently $\rho \sqsubseteq \eta$ iff $\eta(L) \subseteq \rho(L)$; $(\sqcup_{i \in I} \rho_i)(x) = x \Leftrightarrow \forall i \in I.\ \rho_i(x) = x$; $(\sqcap_{i \in I} \rho_i)(x) = \wedge_{i \in I} \rho_i(x)$. The logical meaning of an abstract domain is exactly captured by the associated closure operator on the concrete domain. More formally, on the one hand, if (C, α, D, γ) is a G.i. then the closure associated with D is the operator $\rho_D = \gamma \circ \alpha$ on C. On the other hand, if ρ is a closure on C and $\iota : \rho(C) \to D$ is an isomorphism of complete lattices (with inverse ι^{-1}) then $(C, \iota \circ \rho, D, \iota^{-1})$ is a G.i.. The complete lattice of all abstract interpretations (identified up to isomorphism) of a domain C is therefore isomorphic to $uco(C)$. By the above equivalence, it is not restrictive to use the closure operator approach to reason about abstract properties up to isomorphic representations of abstract domains. Thus, in the rest of the paper, we will feel free to use most of the times this approach, and whenever we will say that D is an abstraction of C, we will mean that D is isomorphic to $\rho_D(C)$, for some closure $\rho_D \in uco(C)$. In this approach, the order relation on $uco(C)$ corresponds to the order by means of which abstract domains are compared with regard to their precision. More formally, if $\rho_i \in uco(C)$ and $D_i \cong \rho_i(C)$ $(i = 1, 2)$, D_1 is *more precise* than D_2 iff $\rho_1 \sqsubseteq \rho_2$ (i.e. $\rho_2(C) \subseteq \rho_1(C)$). Therefore, to compare domains with regard to their precision, we will only speak about abstractions between them, and use $\sqsubseteq$ to relate both closure operators and domains, and the equality symbol $=$ instead of $\cong$. In view of this equivalence, the *lub* and *glb* on $uco(C)$ get a clear meaning. Suppose $\{\rho_i\}_{i \in I} \subseteq uco(C)$ and $D_i \cong \rho_i(C)$ for each $i \in I$. Then, $(\sqcup_{i \in I} \rho_i)(C)$ is the most concrete among the domains which are abstractions of all the D_i's, while $(\sqcap_{i \in I} \rho_i)(C)$ is (isomorphic to) the *reduced product* ([7]) of all the D_i's, which is the most abstract among the domains (abstracting C) which are more concrete than every D_i.

Logic Programming. In the following, *Atom* and *Program* denote, respectively, the set of atoms and (definite), possible infinite, logic programs built on a first-order language $\mathcal{L}$ (where Π denotes the set of predicate symbols). A *unit-clause* is a clause with an empty body. Atoms and unit-clauses will be considered equivalent notions. *BClause* is the set of atoms (unit clauses) and *binary clauses* of the form $h \leftarrow b$, with $h, b \in Atom$. Tuples of syntactic objects of the same type (like variables, terms, *etc.*) are sometimes denoted $\bar{s}$. The set of variables that occur in a syntactic object s is denoted by $var(s)$. The application of a substitution σ to a syntactic object s is denoted by $s\sigma$. The composition of substitutions θ and σ is defined as function composition and denoted $\theta\sigma$. If Θ is a set of substitutions, then $s\Theta = \{s\theta \mid \theta \in \Theta\}$. We restrict our interest to idempotent substitutions ranging in *Sub*, unless explicitly stated otherwise. A *variable renaming* is a (not necessarily idempotent) substitution which is a bijection on the variables. A syntactic object t is *more general* than t' (denoted $t' \leq t$) iff there exists a substitution σ such that $t' = t\sigma$. Syntactic objects t_1 and t_2 are *equivalent up to renaming*, denoted $t_1 \sim t_2$, iff $t_1 \leq t_2$ and $t_2 \leq t_1$. $Atom_{/\sim}$ and $BClause_{/\sim}$ are partially ordered with respect to $\leq$. With abuse of notation, they are still denoted by *Atom* and *BClause*. Since all the definitions in the paper are independent on syntactic variable names, we will let a syntactic object to denote its equivalence class by renaming. For a syntactic object s and a set of (equivalence classes by renaming of) objects I, we denote by $\langle c_1, \ldots, c_n \rangle \ll_s I$ $(n \geq 0)$, that $c_1, \ldots, c_n$ are representatives of elements of I renamed apart from s and from each other. If A is a set of syntactic objects then $\lfloor A \rfloor$ denotes the set of ground instances of elements in A. We fix a partial function *mgu* which maps a pair of syntactic objects (or a set of equations) into an idempotent most general unifier of the objects, if such exists. The operational semantics of logic programs is defined by SLD-resolution. We assume the Prolog selection rule. If $P \in Program$ and $G = b_1, ..., b_n$ $(n \geq 0)$ is a goal, then we write $G \xrightarrow{\vartheta}_P G'\vartheta$ iff there exists $h \leftarrow \bar{b} \ll_G P$, such that $\vartheta = mgu(b_1, h)$ and $G' = \bar{b} :: \langle b_2, ..., b_n \rangle$. $G \longrightarrow^*_P G'$ denotes that there exists ϑ such that $G \xrightarrow{\vartheta}{}^*_P G'$, and $G = G\vartheta$.

3 Complementation in Abstract Interpretation

Complementation ([5]) corresponds to the *inverse of the reduced product* ([14]), namely an operation which, starting from any two domains $C \sqsubseteq D$, gives as result the most abstract domain $C \sim D$, whose reduced product with D is exactly

C (i.e., $(C \sim D) \sqcap D = C$). By the equivalence between closure operators and abstract interpretation, the above notion of complementation corresponds precisely to *pseudo-complementation* for ρ_D in $uco(C)$.[1] Recall that a complete lattice C is *meet-continuous* if for any chain $Y \subseteq C$ and $x \in C$, $x \wedge (\vee Y) = \vee_{y \in Y}(x \wedge y)$. The following result is recalled from [18].

Theorem 3.1 ([18]) *If C is meet-continuous then $uco(C)$ is pseudo-complemented.*

This result has been firstly applied in abstract interpretation for analysis in [5], using the above notion of complementation, which is more precisely formulated as follows: whenever C is meet-continuous, the complement $C \sim D$ exists, and it is defined as:

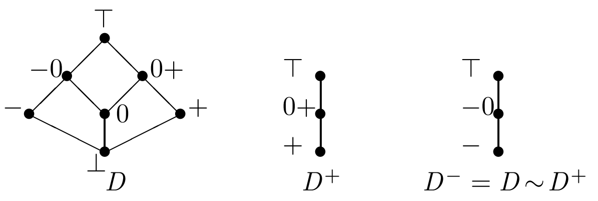
$$C \sim D = \sqcup\{\rho \in uco(C) \mid (\rho_D \sqcap \rho)(C) = C\}.$$

Assume that $C \sqsubseteq D, E$ and let $\top$ be the most abstract interpretation of C (i.e. the top closure $\lambda x.\top$). The following are some basic algebraic properties of complementation ([5]):

(a) $D \sqsubseteq C \sim (C \sim D)$;
(b) $(D \sqsubseteq E) \Rightarrow (C \sim E) \sqsubseteq (C \sim D)$;
(c) $(C \sim D) = C \sim (C \sim (C \sim D))$;
(d) $C \sim \top = C$ and $C \sim C = \top$.

Complementation is essential for *abstract domain decompositions*. If $C \sqsubseteq D$ then $\langle C \sim D, D\rangle$ is a (conjunctive binary) decomposition for C, namely C can be reconstructed by reduced product of its factors.

Example 3.2 Consider the usual lattice D for sign analysis of an integer variable ([7]), depicted below. The concrete domain is $\wp(\mathbb{Z})_{\subseteq}$, and concretization and abstraction maps are the most natural. The (more abstract) lattice of positive values D^+ can be "subtracted" from D by complementation (viz., $D \sim D^+$), yielding the lattice of negative values D^-.



$\langle D^+, D^-\rangle$ is therefore a decomposition for D (i.e., $D = D^+ \sqcap D^-$). In particular, 0 and $\bot$, which are not in $D^+ \cup D^-$, can be both reconstructed by reduced product from D^+ and D^-. □

[1] If L is a meet-semilattice with bottom then the *pseudo-complement* of $x \in L$, if it exists, is the (unique) element $x^* \in L$ such that $x \wedge x^* = \bot$ and $\forall y \in L.\ (x \wedge y = \bot) \Rightarrow (y \leq x^*)$. In a complete lattice L, if the pseudo-complement x^* exists then $x^* = \vee\{y \in L \mid x \wedge y = \bot\}$. If every $x \in L$ has the pseudo-complement, L is *pseudo-complemented*.

4 Semantics Design via Abstract Interpretation

We assume that *Program* denotes the set of syntactically well-defined programs of the considered programming language. We formalize the denotational semantics by means of a *semantic valuation mapping* $[\![\cdot]\!] : \mathit{Program} \to D$ assigning to each program $P \in \mathit{Program}$ its denotational meaning $[\![P]\!]$, which is a value of a complete lattice D, called the *semantic domain*. Elements of the semantic domain D can be execution traces, functions, sets of states, logical interpretations, *etc*. For instance, for the standard denotational semantics of functional programs the semantic domain will be a function space, while for logic programs it will be the powerset (or powerdomain) of a (ground or non-ground) Herbrand universe. Evidently, any semantic definition induces an equivalence relation $\equiv$ on the set of programs, identifying programs with the same behavior (namely denotation) for the semantics: $P \equiv Q \;\Leftrightarrow\; [\![P]\!] = [\![Q]\!]$.

Denotational semantics models possible behaviours of a program: the degree of accuracy of this process depends both on the semantic evaluation mapping and on the semantic domain. Hence, varying these two parameters, one may get semantics at different levels of abstraction. Following [8], it is possible to use the framework of abstract interpretation to establish the relationships between various semantics at different levels of abstraction. We adopt here the closure operator approach to abstract interpretation, which is completely equivalent to the more frequently used Galois connection approach (see Section 2), followed, e.g., in [8], while having the advantage of a higher mathematical simplicity.

Assume that for the considered programming language a standard denotational definition $[\![\cdot]\!] : \mathit{Program} \to D$ is fixed. An abstraction of the standard semantics is obtained by considering any closure operator on the semantic domain. If $\rho \in \mathit{uco}(D)$ then the *abstract semantics induced by* ρ is defined as $[\![\cdot]\!]_\rho : \mathit{Program} \to \rho(D)$, where $\rho(D)$ is the image of D through ρ (or, equivalently, the set of fixpoints of ρ), which is a complete lattice with respect to the induced order, while for any $P \in \mathit{Program}$, $[\![P]\!]_\rho = \rho([\![P]\!])$. In the following, we simply denote by $[\![\cdot]\!]_\rho$ the abstract semantics induced by ρ. Clearly, this definition leads to a hierarchy of abstract denotational semantics, where semantics are related with respect to their precision: for two abstract semantics $[\![\cdot]\!]_\rho$ and $[\![\cdot]\!]_\eta$, the first one is *more precise* than the second one iff $\rho \sqsubseteq \eta$ (in $\mathit{uco}(D)$). Each abstract semantics induces an equivalence relation between programs: we denote by $\equiv_\rho$ the equivalence induced by the semantics $[\![\cdot]\!]_\rho$. The following lemma shows the relationship existing between the ordering for closures and the corresponding induced equivalences on programs.

Lemma 4.1 *For any* $\rho, \eta \in \mathit{uco}(D)$*,* $\rho \sqsubseteq \eta \;\;\Rightarrow\;\; \equiv_\rho \subseteq \equiv_\eta$.

Thus, the above result shows that if $[\![\cdot]\!]_\rho$ is a more precise abstract semantics than $[\![\cdot]\!]_\eta$ (viz. $\rho \sqsubseteq \eta$ in $\mathit{uco}(D)$), then the equivalence induced by $[\![\cdot]\!]_\rho$ is *finer than* the equivalence induced by $[\![\cdot]\!]_\eta$, i.e. if $[\![\cdot]\!]_\eta$ is able to distinguish two programs (viz. they have different denotations) then $[\![\cdot]\!]_\rho$ is able too. It is worth remarking that Lemma 4.1, which is a basic requirement in our approach for semantics comparison, strongly depends on the fact that we deal with closure operators, namely that we use abstract interpretation to relate different semantics. This enforces the importance of using abstract interpretation in this context.

4.1 Product and Complement of Semantics

One of the main advantages offered by abstract interpretation to the design of domains for program analysis is the ability to build abstract domains in an incremental way by combining and refining smaller and simpler domains by means of systematic operations. Below, we consider reduced product and complementation in the field of program semantics.

Reduced Product and Least Common Abstraction of Semantics. As recalled in Section 2, the reduced product of abstract domains corresponds to the meet operation on the respective closure operators, while the *lub* of a family of abstract domains provides the most concrete among the domains which are abstractions of all the domains in the family.

Clearly, using the approach of this paper, all the facilities offered by abstract interpretation for the design of domains for program analysis transfer "verbatim" in denotational semantics design. In particular, we can combine different semantics (which are abstractions of a common standard semantics) via reduced product, to obtain a richer and more precise semantics. Following the notation of Section 2, if $\{[\![\cdot]\!]_{\rho_i}\}_{i\in I}$ is a family of abstract semantics then we denote by $[\![\cdot]\!]_{\sqcap_{i\in I}\rho_i}$ the reduced product semantics. This abstract semantics $[\![\cdot]\!]_{\sqcap_{i\in I}\rho_i} : \mathit{Program} \to \sqcap_{i\in I}\rho_i(D)$ is defined as $[\![P]\!]_{\sqcap_{i\in I}\rho_i} = \wedge_{i\in I}\rho_i([\![P]\!])$, where $\wedge$ is the *glb* of the semantic domain D. The following result formally expresses the intuition that the reduced product semantics corresponds precisely to the logical conjunction of the components semantics.

Theorem 4.2 *For any $P, Q \in \mathit{Program}$, $P \equiv_{\sqcap_{i\in I}\rho_i} Q \;\;\Leftrightarrow\;\; \forall i \in I.\ P \equiv_{\rho_i} Q$.*

The above result says that the reduced product semantics logically coincides with the conjunction of all the components. Stated in other terms, the reduced product semantics does distinguish programs iff all its components do.

Similarly to what done for reduced product, it is possible to interpret the *lub* operation $\sqcup$ between closure operators as a systematic operator on semantics. The *lub* of a family $\{[\![\cdot]\!]_{\rho_i}\}_{i\in I}$ of abstract semantics defines the *least common abstraction* of all $[\![\cdot]\!]_{\rho_i}$'s, i.e. the most precise among the abstract semantics less precise of any $[\![\cdot]\!]_{\rho_i}$ in the family.

Complements of Semantics. Following the lines of the reduced product of semantics above, we can introduce an operation of *complement of denotational semantics* as follows. If $[\![\cdot]\!]_\rho$ and $[\![\cdot]\!]_\eta$ are two abstract semantics such that $[\![\cdot]\!]_\rho$ is more precise than $[\![\cdot]\!]_\eta$, then $[\![\cdot]\!]_{\rho\sim\eta}$ denotes the complement semantics of $[\![\cdot]\!]_\eta$ in $[\![\cdot]\!]_\rho$. This semantics $[\![\cdot]\!]_{\rho\sim\eta} : \mathit{Program} \to (\rho\sim\eta)(D)$ is defined as $[\![P]\!]_{\rho\sim\eta} = (\rho\sim\eta)([\![P]\!])$, where $\rho\sim\eta$ is the closure on D corresponding to the complement of η in ρ. Recalling the meaning of the operation of complementation, we can give a precise interpretation of the complement semantics: $[\![\cdot]\!]_{\rho\sim\eta}$ is the most abstract semantics (i.e. the least precise semantics) whose reduced product with $[\![\cdot]\!]_\eta$ gives exactly $[\![\cdot]\!]_\rho$. It is practically always possible to define complements of semantics, since the conditions assuring their existence are extremely weak. In fact, it is well known that each continuous (and therefore algebraic) complete lattice is meet-continuous, and hence for it, the hypothesis of Theorem 3.1 is satisfied. Thus, in order that complements exist, it is enough that the standard semantic domain D is (meet-)continuous, or even better it is sufficient to check locally for each complement $\rho \sim \eta$ that ρ induces a meet-continuous complete lattice $\rho(D)$. Clearly, these conditions are of wide applicability when dealing with domains for denotational semantics, as in our case.

4.2 Least Fixpoint Semantics

One of the most popular and frequent presentations for denotational semantics is as least fixpoint of a suitable operator on a complete lattice. We suppose that for the considered programming language, a standard denotational semantics is given in least fixpoint form as follows. To any $P \in \mathit{Program}$ is assigned a *semantic operator* or *transformer* T_P, which is a function $T_P : D \to D$ on the semantic domain D (which is a complete lattice) having a least fixpoint (when clear from the context, we simply denote T_P by T). Least fixpoint semantics are therefore presented as pairs $\langle D, T\rangle$. To assure the existence of the least fixpoint, as usual, we assume that T is a monotonic operator; in this case, by Tarski's Theorem, T has a least fixpoint denoted by $\mathit{lfp}(T)$. The standard least fixpoint semantics is therefore given by the semantic valuation mapping $[\![\cdot]\!] : \mathit{Program} \to D$ defined as $[\![P]\!] = \mathit{lfp}(T_P)$.

It is well known in abstract interpretation (cf. [7]) that each domain abstraction uniquely determines the best correct approximation for any semantic operation. This leads in our case to the notion of *abstract least fixpoint semantics*. In fact, each closure operator $\rho \in \mathit{uco}(D)$ actually induces a (more abstract) least fixpoint semantics $[\![\cdot]\!]_\rho$ on the abstract semantic domain $\rho(D)$ defined as least fixpoint of the abstract semantic operator $\rho\circ T : \rho(D) \to \rho(D)$, namely $[\![P]\!]_\rho = \mathit{lfp}(\rho\circ T)$. It is worth noting that $[\![\cdot]\!]_\rho$ can be also equivalently defined as least fixpoint of the semantic operator $\rho\circ T\circ\rho : D \to D$ defined on D. If one prefers to adopt the Galois insertion approach, it is possible to give similar definitions. If (D, α, A, γ) is a G.i. then the abstract semantics $[\![\cdot]\!]_\alpha$ of a program is the least fixpoint of the operator $\alpha\circ T\circ\gamma : A \to A$, that is $[\![P]\!]_\alpha = \mathit{lfp}(\alpha\circ T\circ\gamma)$. In both definitions, the abstract semantic operator is the *best correct approximation* of T.

The approach here slightly differs from the general one presented above. In fact, the abstract semantics of the program is not defined as the given abstraction (ρ) of the standard semantics of the program, viz. $\rho(\mathit{lfp}(T))$, and, in general, $\mathit{lfp}(\rho\circ T) \neq \rho(\mathit{lfp}(T))$. However, it is known (see e.g. [9]) that for any monotonic operator T, $\rho(\mathit{lfp}(T)) \leq \mathit{lfp}(\rho\circ T)$ holds. *Completeness* of the abstraction is a well known sufficient condition assuring that the equality holds (see [9, 21]). The abstraction $\rho \in \mathit{uco}(D)$ is *complete* if $\rho\circ T = \rho\circ T\circ\rho$; analogously, the abstraction given by the Galois insertion (D, α, A, γ) is *complete* if $\alpha\circ T = \alpha\circ T\circ\gamma\circ\alpha$. Thus, we will say that an abstract least fixpoint semantics $[\![\cdot]\!]_\rho$ or $[\![\cdot]\!]_\alpha$ is complete if the corresponding abstraction is complete.

Proposition 4.3 ([9]) *If $[\![\cdot]\!]_\rho$ is complete and $P \in \mathit{Program}$ then $[\![P]\!]_\rho = \rho(\mathit{lfp}(T_P))$.*

Clearly, $[\![\cdot]\!]_{id} = [\![\cdot]\!]$ and $[\![\cdot]\!]_{\lambda x.\top}$ are always complete abstractions. In the following, if $\mathcal{X} = \langle D, T^{\mathcal{X}}\rangle$ specifies a least fixpoint semantics, we will denote by $[\![\cdot]\!]^{\mathcal{X}}$ the corresponding least fixpoint semantics. Also, if $\rho \in \mathit{uco}(D)$ and $\mathcal{Y} = \langle \rho(D), \rho\circ T^{\mathcal{X}}\rangle$

$$T_P^{\mathcal{S}} = \lambda I \in \wp(Atom).\ \{h\vartheta \mid C = h \leftarrow \bar{b} \in P,\ \bar{b}' \ll_C I,\ \vartheta = mgu(\bar{b}, \bar{b}')\}$$

$$T_P^{\mathcal{C}} = \lambda I \in \wp^{\downarrow}(Atom).\ \{h\vartheta \mid C = h \leftarrow \bar{b} \in P,\ \vartheta \in Sub,\ \bar{b}\vartheta \in I\}$$

$$T_P^{\mathcal{H}} = \lambda I \in \wp(\lfloor Atom \rfloor).\ \{h \in \lfloor Atom \rfloor \mid h \leftarrow \bar{b} \in \lfloor P \rfloor,\ \bar{b} \subseteq I\}$$

Table 1: The semantic operators.

is the corresponding abstract semantics, then $[\![\cdot]\!]^{\mathcal{Y}} = [\![\cdot]\!]^{\mathcal{X}}_{\rho}$. Further, we will denote by $Sem(\mathcal{X})$ the class of semantic definitions $\mathcal{Y}$ such that $[\![\cdot]\!]^{\mathcal{Y}} = [\![\cdot]\!]^{\mathcal{X}}_{\rho}$, for some $\rho \in uco(D)$. Therefore, $uco(D) \cong Sem(\mathcal{X})$. We keep $uco(D)$ and $Sem(\mathcal{X})$ distinct notations in order to identify the use of closure operators as semantics approximators. However, in order to simplify the notation, we often let the elements in $Sem(\mathcal{X})$ denote both abstract semantics for $\mathcal{X}$, and the corresponding closures, as well as we use $\sqcap$, $\sqcup$, and $\sim$ to combine elements in $Sem(\mathcal{X})$. Confusing closure names with semantics names may be source of ambiguity. Therefore, in the following, if $\mathcal{Y} \in uco(D)$, $P \in Program$ and $T_P^{\mathcal{Y}} : \mathcal{Y}(D) \to \mathcal{Y}(D)$ is a semantic operator, then the notation $[\![P]\!]^{\mathcal{Y}} = [\![P]\!]^{\mathcal{X}}_{\mathcal{Y}}$ means that $lfp(T_P^{\mathcal{Y}}) = lfp(\mathcal{Y} \circ T_P^{\mathcal{X}})$. If this holds then $\langle \mathcal{Y}(D), T^{\mathcal{Y}} \rangle$ is equivalent to $\langle \mathcal{Y}(D), \mathcal{Y} \circ T^{\mathcal{X}} \rangle$. In this case, we abuse notation by letting also $\langle \mathcal{Y}(D), T^{\mathcal{Y}} \rangle \in Sem(\mathcal{X})$.

4.3 A Hierarchy of Logic Program Semantics

s-semantics, denoted by $\mathcal{S}$, has been introduced in [12] in order to provide a fully abstract description of *computed answer substitutions* of logic programs. Recall that ϑ is a computed answer substitution for a goal G and a program P iff $G \overset{\vartheta}{\longrightarrow}{}^{*}_{P} \Lambda$. Clark's semantics, denoted by $\mathcal{C}$, has been introduced in [2]. In [12], it was proved that $\mathcal{C}$ is fully abstract with respect to the "more abstract" notion of *correct answer substitution*. Recall that ϑ is a correct answer substitution for a goal G and a program P iff $G\vartheta \longrightarrow^{*}_{P} \Lambda$. Finally, Herbrand's semantics, denoted by $\mathcal{H}$, coincides with the least Herbrand model of the program, and has been fully characterized by van Emden and Kowalski in [10]. In what follows, we assume that a first-order language $\mathcal{L}$ is given. The semantic domains for $\mathcal{S}$, $\mathcal{C}$ and $\mathcal{H}$ are respectively $\wp(Atom)_{\subseteq}$, $\wp^{\downarrow}(Atom)_{\subseteq}$, and $\wp(\lfloor Atom \rfloor)_{\subseteq}$, while for any program P, the corresponding operators are $T_P^{\mathcal{S}} : \wp(Atom) \to \wp(Atom)$, $T_P^{\mathcal{C}} : \wp^{\downarrow}(Atom) \to \wp^{\downarrow}(Atom)$, and $T_P^{\mathcal{H}} : \wp(\lfloor Atom \rfloor) \to \wp(\lfloor Atom \rfloor)$, as defined in Table 1, and are all continuous functions. Thus, $[\![P]\!]^{\mathcal{S}} = lfp(T_P^{\mathcal{S}})$, $[\![P]\!]^{\mathcal{C}} = lfp(T_P^{\mathcal{C}})$, and $[\![P]\!]^{\mathcal{H}} = lfp(T_P^{\mathcal{H}})$.

As proved in [17], s-semantics, Clark's semantics and Herbrand's semantics constitute a hierarchy in our framework (i.e. for abstract interpretation), where $\mathcal{S}$ is more precise than $\mathcal{C}$, which in turn is more precise than $\mathcal{H}$. We now recall from [17] the details of the abstraction mappings of this hierarchy.

As expected, the operator of instantiation $\downarrow$ is the closure on the domain of s-semantics defining Clark's semantics. In particular, we have that $T^{\mathcal{C}} = \downarrow \circ T^{\mathcal{S}}$. Also, this abstraction is complete, i.e. for any $I \in \wp(Atom)$, $\downarrow(T^{\mathcal{S}}(I)) = \downarrow(T^{\mathcal{S}}(\downarrow(I)))$, viz. $T^{\mathcal{C}} \circ \downarrow = \downarrow \circ T^{\mathcal{S}}$. Hence, by Proposition 4.3, this implies that for any $P \in Program$, $[\![P]\!]^{\mathcal{C}} = \downarrow([\![P]\!]^{\mathcal{S}}) = [\![P]\!]^{\mathcal{S}}_{\downarrow}$. This last observation has been firstly reported in [13]. It is worth noting that we have derived it by exploiting the techniques offered by abstract interpretation, which are intrinsically language-independent.

Herbrand's semantics is an abstraction of Clark's semantics through the following Galois insertion: $(\wp^{\downarrow}(\mathit{Atom}), \lfloor\cdot\rfloor, \wp(\lfloor \mathit{Atom}\rfloor), \lceil\cdot\rceil)$, where $\lceil\cdot\rceil$ is the unique right adjoint to $\lfloor\cdot\rfloor$, i.e. for any $I \subseteq \lfloor \mathit{Atom}\rfloor$, $\lceil I\rceil = \cup\{J \in \wp^{\downarrow}(\mathit{Atom}) \mid \lfloor J\rfloor \subseteq I\}$. Thus, $T^{\mathcal{H}} = \lfloor\cdot\rfloor \circ T^{\mathcal{C}} \circ \lceil\cdot\rceil$, i.e. the immediate consequence operator is the best correct approximation of the semantic operator of Clark's semantics. This abstraction is also complete, namely $\lfloor\cdot\rfloor \circ T^{\mathcal{C}} = T^{\mathcal{H}} \circ \lfloor\cdot\rfloor$. Again by Proposition 4.3, we get that $[\![P]\!]^{\mathcal{H}} = \lfloor [\![P]\!]^{\mathcal{C}} \rfloor$, another observation already reported in [13].

Finally, composing the previous two abstractions, we get the abstraction between s-semantics and Herbrand's semantics. This is defined in terms of the G.i. $(\wp(\mathit{Atom}), \lfloor\cdot\rfloor, \wp(\lfloor \mathit{Atom}\rfloor), \lceil\cdot\rceil)$, where, by a slight abuse of notation, we again denote by $\lceil\cdot\rceil$ the right adjoint to $\lfloor\cdot\rfloor$ for the concrete domain $\wp(\mathit{Atom})$. Also in this case, we have a complete abstraction. Hence, $\lfloor\cdot\rfloor \circ T^{\mathcal{S}} = T^{\mathcal{H}} \circ \lfloor\cdot\rfloor$, and, by Proposition 4.3, $[\![P]\!]^{\mathcal{H}} = \lfloor [\![P]\!]^{\mathcal{S}} \rfloor$. As above, this last observation is in [13]. In this case, $\lambda I.\lceil \lfloor I\rfloor \rceil \in \mathit{uco}(\wp(\mathit{Atom}))$ is the closure associated with Herbrand's semantics.

5 Decomposition of Semantics

Let $\mathcal{O}$ be a given standard semantics. A (possible infinite) family of semantics $\langle \mathcal{X}_i\rangle_{i\in I} \subseteq \mathit{Sem}(\mathcal{O})$ is a (*conjunctive*) *decomposition* for $\mathcal{Y} \in \mathit{Sem}(\mathcal{O})$ if $\mathcal{Y} = \sqcap_{i\in I}\mathcal{X}_i$. Filé and Ranzato observed in [15] that $\langle \rho_i\rangle_{i\in I}$ is a *minimal decomposition* for a closure δ (i.e., for all $k \in I$ and η closure, $(\rho_k \sqsubset \eta) \Rightarrow (\delta \sqsubset (\sqcap_{i\in I\setminus\{k\}}\rho_i) \sqcap \eta))$ iff for any $k \in I$, $\delta \sim (\sqcap_{i\in I\setminus\{k\}}\rho_i) = \rho_k$. Among the possible minimal decompositions, we consider those given by the most abstract factors. It is well known (cf. [22]) that if L is a complete lattice, then $\mathit{uco}(L)$ is a dual-atomistic complete lattice, viz. for any $\rho \in \mathit{uco}(L)$, $\rho = \sqcap_{x\in\rho(L)\setminus\{\top\}}\varphi_x$, where $\varphi_x = \{\top, x\}$. We call the φ_x's *atomic closures*. We exploit these results, by giving a sufficient condition in order that any closure operator, and therefore any abstract semantics, admits a *unique* minimal factorization, involving the least number of atomic closures.[2]

Theorem 5.1 *Assume that L is an algebraic complete lattice and $\rho \in \mathit{uco}(L)$. Then, $\langle\varphi_x\rangle_{x\in MI(\rho(L))}$ is a minimal decomposition for ρ, involving the least number of atomic closures only.*

Morgado proved in [23] that atomic closures φ_x defined on meet-irreducible elements are always complemented, i.e. for any $x \in MI(L)$, there exists $\bar{\varphi}_x \in \mathit{uco}(L)$ such that $\varphi_x \sqcap \bar{\varphi}_x = id$, and $\varphi_x \sqcup \bar{\varphi}_x = \lambda x.\top$. In particular, since algebraic lattices are also meet-continuous, then $\bar{\varphi}_x = \varphi_x^{*} = id \sim \varphi_x$, for any $x \in MI(L)$. A natural question is therefore: is it possible to give a characterization of the pseudo-complements of closure operators in terms of the complements of their atomic factors? Actually, we show that a weak form of De Morgan's identity, involving pseudo-complements, always holds for atomic closures, and that if L is algebraic then the pseudo-complement of any closure operator is uniquely determined by the real complements of its atomic factors.

[2] If L is a lattice then $x \in L$ is *meet-irreducible* if $\forall y, z \in L.\ (x = y \wedge z) \Rightarrow (x = y$ or $x = z)$. We denote by $MI(L)$ the set of meet-irreducible elements of L.

Lemma 5.2 *If L is an algebraic complete lattice and $I \subseteq L\backslash\{\top\}$ then,* $(\sqcap_{x\in I}\, \varphi_x)^* = \sqcup_{x\in I}\, \varphi_x^* = \sqcup_{x\in I\cap MI(L)}\, \bar{\varphi}_x$.

Hence, we obtain the following characterization for pseudo-complements of closure operators in terms of complements of atomic closures.

Theorem 5.3 *If L is an algebraic complete lattice and $\rho \in uco(L)$ then,* $id \sim \rho = \rho^* = \sqcup_{x\in\rho(L)\cap MI(L)}\, \bar{\varphi}_x$.

The interest in this result is that the semantic interpretation for atomic closures and their complements is easier. Again, logic programming provides a clean and immediate meaning for these atomic semantics. Consider the s-semantics $\mathcal{S}$ recalled in Section 4.3. Since $\wp(Atom)_{\subseteq}$ is evidently algebraic, then the least (unique) minimal decomposition for $\mathcal{S}$ $(= id \in uco(\wp(Atom)))$ of Theorem 5.1, is given by the atomic closures $\langle \mathcal{A}_h \rangle_{h\in Atom}$, where $\mathcal{A}_h = \{Atom, Atom \setminus \{h\}\}$, for any $h \in Atom$. In this case, $\bar{\mathcal{A}}_h = \{I \subseteq Atom \mid h \in I\}$ are the corresponding complements. If P is a program, then $T_P^{\mathcal{A}_h} : \mathcal{A}_h \to \mathcal{A}_h$ is such that:

$$T_P^{\mathcal{A}_h}(I) = \begin{cases} Atom & \text{if } h \in T_P^{\mathcal{S}}(I); \\ Atom \setminus \{h\} & \text{otherwise.} \end{cases}$$

It is possible to show that this is a good definition, since for any $h \in Atom$, $[\![P]\!]^{\mathcal{A}_h} = [\![P]\!]^{\mathcal{S}}_{\mathcal{A}_h}$. The semantics associated with each atomic closure $\mathcal{A}_h$, and with its complement $\bar{\mathcal{A}}_h$, for $h \in Atom$, have then a clear meaning given by the following result.

Theorem 5.4 *Let P and Q be programs, and $h \in Atom$.*

1. $P \equiv_{\mathcal{A}_h} Q$ *iff* $(\exists R.\ h \notin [\![R]\!]^{\mathcal{S}} \,\&\, h \in [\![P \cup R]\!]^{\mathcal{S}}) \Leftrightarrow (\exists T.\ h \notin [\![T]\!]^{\mathcal{S}} \,\&\, h \in [\![Q \cup T]\!]^{\mathcal{S}})$.
2. $P \equiv_{\bar{\mathcal{A}}_h} Q$ *iff* $[\![P \cup \{h\}]\!]^{\mathcal{S}} = [\![Q \cup \{h\}]\!]^{\mathcal{S}}$.

Atomic semantics are therefore able only to distinguish whether a given atom h can be derived from a program P when this program has been composed with another program module R, whereas their complements, which are almost like the s-semantics, cannot provide any information about h. By Theorem 5.3, for any $\mathcal{X} \in Sem(\mathcal{S})$, we have:

$$\mathcal{S} \sim \mathcal{X} = \bigsqcup_{\substack{h \,\in\, Atom \\ Atom\setminus\{h\} \,\in\, \mathcal{X}}} \bar{\mathcal{A}}_h = \bigcap_{\substack{h \,\in\, Atom \\ Atom\setminus\{h\} \,\in\, \mathcal{X}}} \bar{\mathcal{A}}_h .$$

Consider, as a limit case, Herbrand's semantics $\mathcal{H}$. This complement is somewhat surprising. Indeed, since for any $h \in Atom$, $Atom\backslash\{h\} \notin \lceil\lfloor\wp(Atom)\rfloor\rceil$, then $\mathcal{S} \sim \mathcal{H} = \sqcup\emptyset = \mathcal{S}$. Thus, for any program P, $[\![P]\!]^{\mathcal{S}\sim\mathcal{H}} = [\![P]\!]^{\mathcal{S}}$. If we recall the meaning of the "best" semantics for analysis introduced in [17], we can draw a striking consequence of this fact. Due to lack of space, we omit the details of the constructions of [17], to which the reader is referred. Roughly, [17] calls a semantics $\mathcal{X}$ *too concrete* for a given property ρ on $\mathcal{X}$ (indeed a closure on the semantic domain of $\mathcal{X}$) if there exists a proper abstraction $\mathcal{Y}$ of $\mathcal{X}$ that allows to perform the desired analysis (viz. ρ) without loosing any information with respect to $\mathcal{X}$. In this sense, [17] shows that there exists always a best semantics for analysis, and provides an equational characterization for it involving the reduced product. Thus, exploiting these results, we may conclude that s-semantics $\mathcal{S}$ *cannot be* the best semantics for the analysis of any property which includes, at least, the least Herbrand model semantics $\mathcal{H}$, i.e. no proper abstraction of $\mathcal{S}$ can be combined with $\mathcal{H}$ to get back $\mathcal{S}$.

6 Complements in the Hierarchy

We now consider the remaining complements for the semantics of the hierarchy in Section 4.3.

6.1 Logic vs. Operational Semantics

Clark's semantics is considered a *logical* semantics for programs (cf. [1]). Instead, the s-semantics captures the *operational* notion of computed answer substitution. Clark's semantics is here characterized by the closure $\lambda x.\downarrow x \in uco(\wp(Atom)_{\subseteq})$, where $Atom_{\leq}$ is a poset.

Lemma 6.1 *If P is a poset then $(\lambda x.\downarrow x)^* = \lambda x.x \cup Max(P)$.*

We can constructively characterize the semantics of the complement $\mathcal{S} \sim \mathcal{C} \in Sem(\mathcal{S})$ as the semantics induced by the abstraction associated with the pseudo-complement $\lambda I.I \cup Max(Atom)$. In this case, for a given first-order language $\mathcal{L}$, $Max(Atom) = \{p(\bar{x}) \mid p \in \Pi\}$, and therefore $\mathcal{S} \sim \mathcal{C} = \{I \cup Max(Atom) \mid I \subseteq Atom\}$. Given a program P, the semantic operator $T_P^{\mathcal{S}\sim\mathcal{C}}$ associated with $\mathcal{S}\sim\mathcal{C}$ is $T_P^{\mathcal{S}\sim\mathcal{C}} = \lambda I.T_P^{\mathcal{S}}(I \cup Max(Atom)) \cup Max(Atom)$. The following theorem characterizes the semantics of the complement $\mathcal{S}\sim\mathcal{C}$ in terms of the semantics of computed answer substitutions of a simple transformed program.

Theorem 6.2 *Let $P \in Program$. Then,*

(i) $[\![P]\!]^{\mathcal{S}\sim\mathcal{C}} = [\![P]\!]^{\mathcal{S}}_{\mathcal{S}\sim\mathcal{C}} = [\![P \cup Max(Atom)]\!]^{\mathcal{S}}$*;*
(ii) $[\![\cdot]\!]^{\mathcal{S}}_{\mathcal{S}\sim\mathcal{C}}$ *is not complete (i.e., $\exists Q.\ \mathcal{S}\sim\mathcal{C}([\![Q]\!]^{\mathcal{S}}) \subset [\![Q]\!]^{\mathcal{S}}_{\mathcal{S}\sim\mathcal{C}}$).*

A consequence of Theorem 6.2-(i) is that the semantics $\mathcal{S}\sim\mathcal{C}$ corresponds precisely to the fully abstract semantics for partial computed answer substitutions introduced in [11]. Recall that a substitution ϑ is a partial computed answer for the goal G in the program P iff there exists G' and σ such that $G \xrightarrow{\sigma}{}^{*}_{P} G'$ and $G\vartheta = G\sigma$ (cf. [11]). Informally, the atoms in $Max(Atom)$ allow us to transform any partial derivation for G in P into a successful derivation for G in $P \cup Max(Atom)$. The following result is therefore immediate by Theorem 6.2 and Theorem 5.4 in [11].

Theorem 6.3 *Let $P \in Program$ and G be a goal. ϑ is a partial computed answer substitution for G in P iff there exist $\bar{b} \ll_G [\![P]\!]^{\mathcal{S}\sim\mathcal{C}}$ such that $G\vartheta = G\, mgu(\bar{g}, \bar{b})$.*

Thus, if $P, Q \in Program$, then $P \equiv_{\mathcal{S}\sim\mathcal{C}} Q$ iff P and Q allow the same partial computed answer substitutions for any goal. The fact that the composition by reduced product of the partial computed answer and correct answer semantics is the semantics of computed answers, should be clear because: *"a substitution which is a partial computed and correct answer for a goal in a program is always a computed answer"*. Also, the semantics of partial computed answers is the (unique) most abstract semantics which, whenever composed with the semantics of correct answers, gives as result the s-semantics of computed answers. The intuition behind this result is clear. Clark's semantics is unable to distinguish programs augmented with $\mathit{Max}(\mathit{Atom})$. This because, if $P \in \mathit{Program}$ then any atom is a possible logical consequence of $P \cup \mathit{Max}(\mathit{Atom})$, i.e. Atom is the least $\mathcal{C}$-model for $P \cup \mathit{Max}(\mathit{Atom})$. In particular, the program transform $\lambda P \in \mathit{Program}.P \cup \mathit{Max}(\mathit{Atom})$ maps possibly logically different programs into logically equivalent ones. In this sense, $P \cup \mathit{Max}(\mathit{Atom})$ represents what is left from a logic program, once the information about its logical consequences has been removed, which is the semantics $\mathcal{S} \sim \mathcal{C}$ of partial computed answers. Further, the semantics of partial computed answers $\mathcal{S} \sim \mathcal{C}$ is also disjoint with respect to Clark's semantics, namely it is the lattice-theoretic complement of $\mathcal{C}$ into $\mathit{Sem}(\mathcal{S})$, i.e. $(\mathcal{S} \sim \mathcal{C}) \sqcup \mathcal{C} = \lambda x.\mathit{Atom}$. Hence, there is no semantics which can be at the same time correct with respect to both correct and partial answer semantics. For instance, Herbrand's semantics is not even correct with respect to $\mathcal{S} \sim \mathcal{C}$, i.e. $\mathcal{H} \notin \mathit{Sem}(\mathcal{S} \sim \mathcal{C})$.

6.2 Clark's vs. Herbrand's Semantics

We conclude by characterizing the complement $\mathcal{C} \sim \mathcal{H}$ of the least Herbrand model semantics into Clark's semantics.

Theorem 6.4 $\mathcal{C} \sim \mathcal{H} = \lambda I.I \cup \lfloor \mathit{Atom} \rfloor \in \mathit{uco}(\wp^{\downarrow}(\mathit{Atom})_{\subseteq})$.

In other terms, the semantic domain for $\mathcal{C} \sim \mathcal{H}$ is $\{I \in \wp^{\downarrow}(\mathit{Atom}) \mid \lfloor \mathit{Atom} \rfloor \subseteq I\}$. For a program P the semantic operator for $\mathcal{C} \sim \mathcal{H}$ is defined as follows:

$$T_P^{\mathcal{C} \sim \mathcal{H}} = \lambda I.\, T_P^{\mathcal{C}}(I \cup \lfloor \mathit{Atom} \rfloor) \cup \lfloor \mathit{Atom} \rfloor.$$

It is possible to show that $\mathcal{C} \sim \mathcal{H}$ is not complete. Analogously to the case of $\mathcal{S} \sim \mathcal{C}$, we can give a characterization of the least fixpoint semantics $\mathcal{C} \sim \mathcal{H}$ in terms of Clark's semantics of a transformed program.

Theorem 6.5 *For any* $P \in \mathit{Program}$, $[\![P]\!]^{\mathcal{C}}_{\mathcal{C} \sim \mathcal{H}} = [\![P]\!]^{\mathcal{C} \sim \mathcal{H}} = [\![P \cup \lfloor \mathit{Atom} \rfloor]\!]^{\mathcal{C}}$.[3]

The equivalence induced by this complement is as follows: if $P, Q \in \mathit{Program}$ then $P \equiv_{\mathcal{C} \sim \mathcal{H}} Q \Leftrightarrow P \cup \lfloor \mathit{Atom} \rfloor \equiv_{\mathcal{C}} Q \cup \lfloor \mathit{Atom} \rfloor$. This makes clear the intuitive meaning of this semantics. For instance, $P = \{p(a).\}$ and $Q = \{q(a).\}$ have the same $\mathcal{C} \sim \mathcal{H}$ semantics (which is $\lfloor \mathit{Atom} \rfloor = \{p(a), q(a)\}$), while they are obviously distinguished by Clark's semantics $\mathcal{C}$. In view of the general meaning of the complement operator, this is actually what one can expect from the semantics $\mathcal{C} \sim \mathcal{H}$.

[3] It should be noted that the transform $\lambda P \in \mathit{Program}.P \cup \lfloor \mathit{Atom} \rfloor$, in general, maps finite programs in infinite programs.

7 Semantics Interaction: Call vs. Success Patterns

Semantics at different levels of abstraction can interact with each other when composed via reduced product. Let $\mathcal{O}$ be a given semantics. The interaction among the factors of a conjunctive decomposition $\langle \mathcal{X}_i \rangle_{i \in I}$ of $\mathcal{O}$ (viewed as a closure) is captured precisely by the fact that for some program P, it may happen that $[\![P]\!]^{\mathcal{O}} < \wedge_{i \in I} [\![P]\!]^{\mathcal{O}}_{\mathcal{X}_i}$, even if $\mathcal{O} = \sqcap_{i \in I} \mathcal{X}_i$ as closure operators. In this case, we say that the factors $\{\mathcal{X}_i\}_{i \in I}$ *cooperate* in $\mathcal{O}$. Clearly, if $\mathcal{O}$ is a least fixpoint semantics and the $\mathcal{X}_i$'s are all complete abstractions, then the factors $\{\mathcal{X}_i\}_{i \in I}$ do not cooperate in the definition of $\mathcal{O}$. This is however only a sufficient condition to check in order to verify whether semantics do not cooperate, as shown below for the case of Clark's and partial answer semantics.

The following is a simple sufficient condition to check semantic cooperation among factors in least fixpoint semantics.

Theorem 7.1 *Let $\langle \mathcal{X}_i \rangle_{i \in I} \subseteq Sem(\mathcal{O})$ be a conjunctive decomposition for the least fixpoint semantics $\mathcal{O}$. If for some program P, $[\![P]\!]^{\mathcal{O}} < \wedge_{i \in I} T_P(\mathcal{X}_i([\![P]\!]^{\mathcal{O}}))$, then $\{\mathcal{X}_i\}_{i \in I}$ cooperate in $\mathcal{O}$.*

Hence, we may characterize when a semantics (viz. a property) $\mathcal{X} \in Sem(\mathcal{O})$ is *used* in the definition of $\mathcal{O}$. Intuitively, this is the case when $\langle \mathcal{X}, \mathcal{O} \sim \mathcal{X} \rangle$ is a binary decomposition with cooperating factors for $\mathcal{O}$. In this case, the semantics $[\![\cdot]\!]^{\mathcal{O}}$ cannot be constructed by composing the semantics $[\![\cdot]\!]^{\mathcal{X}}$ and $[\![\cdot]\!]^{\mathcal{O} \sim \mathcal{X}}$.

Note that there is no cooperation in $\mathcal{S}$ between Clark's semantics and the semantics of partial answers, even if, by Theorem 6.2-(ii), $\mathcal{S} \sim \mathcal{C}$ is not a complete abstraction. In fact, it is easy to prove that for any program P, $[\![P]\!]^{\mathcal{S}} = [\![P]\!]^{\mathcal{S} \sim \mathcal{C}} \cap [\![P]\!]^{\mathcal{C}}$. This confirms that completeness is only a sufficient condition to ensure independency among the factors of a semantic decomposition. Likewise, being $\mathcal{S} \sim \mathcal{H} = \mathcal{S}$, $\mathcal{H}$ is "useless" for $\mathcal{S}$.

A typical case of semantic interaction in logic programming is given by the semantics for computed answer substitutions and call patterns. Recall that an atom $a \in Atom$ is a *call pattern* for a goal G in a program P if $G \xrightarrow{\vartheta}{}^{*}_{P} a \mid \bar{b}$. From this definition it is clear that, in order to have a call pattern for $p \in \Pi$ for a goal $G = \bar{b} :: p(\bar{t}) :: \bar{b}'$, then $\bar{b}$ needs to have a computed answer substitution ϑ in the program, and $p(\bar{t})\vartheta$ will be the corresponding call pattern. Hence, call patterns need computed answers to be computed. We formally justify this by complementing the semantics of computed answer substitutions in that of call patterns.

A least fixpoint semantics for call patterns has been introduced by Gabbrielli and Meo in [16] as $\Pi = \langle \wp(BClause), T_P^{\Pi} \rangle$, where, if $\Phi = \{p(\bar{x}) \leftarrow p(\bar{x}) \mid p \in \Pi\}$ is the set of tautological clauses, then, for any $I \subseteq BClause$,

$$T_P^{\Pi}(I) = \left\{ (h \leftarrow b'')\vartheta \;\middle|\; \begin{array}{l} c = h \leftarrow b_1, ..., b_{k-1}, b_k, ..., b_n,\ k \leq n \\ \langle b'_1, ..., b'_{k-1}, b'_k \leftarrow b'' \rangle \ll_c I \cup \Phi \\ \vartheta = mgu(\langle b_1, ..., b_k \rangle, \langle b'_1, ..., b'_k \rangle) \end{array} \right\}.$$

Gabbrielli and Meo proved that a is a call pattern for $G = b_1, ..., b_{k-1}, b_k, ..., b_n$ in P iff there exists $\langle b'_1, ..., b'_{k-1}, b'_k \leftarrow b'' \rangle \ll_c lfp(T_P^{\Pi}) \cup \Phi$, such that $a = b''\vartheta$ and $\vartheta = mgu(\langle b_1, ..., b_k \rangle, \langle b'_1, ..., b'_k \rangle)$.

The interest in call patterns is that $\mathcal{S}$ can be derived by abstract interpretation from Π. This provides a formal account on the interaction between call patterns and computed answer substitutions in the semantics of logic programs. It is immediate to see that $\mathcal{S}$ corresponds to the closure operator $\lambda I. I \cup (\mathit{BClause} \setminus \mathit{Atom}) \in \mathit{uco}(\wp(\mathit{BClause}))$, while Π is the identical closure on $\wp(\mathit{BClause})$. It is possible to show that $\llbracket\cdot\rrbracket^{\Pi}_{\mathcal{S}}$ is complete, and for any program P, $\llbracket P\rrbracket^{\mathcal{S}} = \llbracket P\rrbracket^{\Pi}_{\mathcal{S}} \cap \mathit{Atom}$.

The semantics of computed answer substitutions can be "subtracted" from that of call patterns, providing a semantics for logic programs which characterizes non-successful call patterns, and is associated with a simple program transform. We call this information *naïve call patterns*.

Proposition 7.2 $\Pi \sim \mathcal{S} = \lambda I. I \cup \mathit{Atom}$.

As expected, the semantics of naïve call patterns $\Pi \sim \mathcal{S}$ is not able to distinguish computed answer substitutions, while it is able to provide a weak form of call patterns. Intuitively, a naïve call pattern in $\Pi \sim \mathcal{S}$ does not consider the computed answer substitutions for the precedings atoms in the goal. Therefore, a is a naïve call pattern in $\Pi \sim \mathcal{S}$ for a goal $b_1, ..., b_{k-1}, b_k, ..., b_n$ with $k \leq n$ in a program P, if $a = b_k\vartheta$ for some $\vartheta \in \mathit{Sub}$ defined on $\mathit{var}(b_1, ..., b_{k-1})$. This is obtained independently from the fact that ϑ is actually a computed answer substitution for $b_1, ..., b_{k-1}$. This intuition is captured precisely by the following operator. Let P be a program, and define $T_P^{\Pi\sim\mathcal{S}} : \wp(\mathit{BClause}) \to \wp(\mathit{BClause})$ such that for any $I \subseteq \mathit{BClause}$,

$$T_P^{\Pi\sim\mathcal{S}}(I) = \left\{ (h \leftarrow b'')\vartheta \;\middle|\; \begin{array}{l} c = h \leftarrow b_1, ..., b_{k-1}, b_k, ..., b_n,\ k \leq n \\ \langle b'_1, ..., b'_{k-1}\rangle \ll_c \mathit{Atom},\ b'_k \leftarrow b'' \ll_{c, b'_1, ..., b'_{k-1}} I \cup \Phi \\ \vartheta = \mathit{mgu}(\langle b_1, ..., b_k\rangle, \langle b'_1, ..., b'_k\rangle) \end{array} \right\}.$$

The following result proves that $T_P^{\Pi\sim\mathcal{S}}$ is exactly the immediate consequence operator associated with the naïve call patterns semantics $\Pi \sim \mathcal{S}$, and specifies the semantics of $\Pi \sim \mathcal{S} \in \mathit{Sem}(\Pi)$ in terms of a simple program transform. This transform corresponds to enhance a given program by including any possible substitution as computed answer. $\llbracket P\rrbracket^{\Pi\sim\mathcal{S}}$ captures precisely the non-successful call patterns of the enhanced program. It is also possible to verify that $\Pi \sim \mathcal{S}$ is not complete.

Theorem 7.3 *Let P be a program.*

1. *For any $I \in \mathit{BClause}$, $T_P^{\Pi\sim\mathcal{S}}(I) = T_P^{\Pi}(\Pi \sim \mathcal{S}(I)) \setminus \mathit{Atom}$;*
2. *$\llbracket P\rrbracket^{\Pi\sim\mathcal{S}} = \llbracket P \cup \mathit{Atom}\rrbracket^{\Pi} \setminus \mathit{Atom}$;*
3. *$\exists Q \in \mathit{Program}.\ \Pi \sim \mathcal{S}(\llbracket Q\rrbracket^{\Pi}) \subset \llbracket Q\rrbracket^{\Pi}_{\Pi\sim\mathcal{S}} = \Pi \sim \mathcal{S}(\llbracket Q\rrbracket^{\Pi\sim\mathcal{S}}) = \llbracket Q \cup \mathit{Atom}\rrbracket^{\Pi}$.*

Moreover, by Theorem 7.1, it is easy to observe that the semantics of computed answer substitutions cooperates with that of naïve call patterns in order to compute call patterns for logic programs. In particular, $\llbracket\cdot\rrbracket^{\Pi} \neq \llbracket\cdot\rrbracket^{\Pi}_{\mathcal{S}} \cap \llbracket\cdot\rrbracket^{\Pi}_{\Pi\sim\mathcal{S}}$, as in the case of the following simple program: $P = \{\mathit{tr}(x, y) \leftarrow R(x, z), \mathit{tr}(z, y).;\ R(x, x).\}$. In this case, $\mathit{tr}(x, y) \leftarrow \mathit{tr}(z, y) \notin \llbracket P\rrbracket^{\Pi}$, while $\mathit{tr}(x, y) \leftarrow \mathit{tr}(z, y) \in \llbracket P\rrbracket^{\Pi}_{\mathcal{S}} \cap \llbracket P\rrbracket^{\Pi}_{\Pi\sim\mathcal{S}}$.

Acknowledgments. We would like to thank Gilberto Filé for his valuable contribution.

References

1. K.R. Apt and M. Gabbrielli. Declarative interpretations reconsidered. In *Proc. ICLP '94*, pp. 74–89, The MIT Press, 1994.
2. K.L. Clark. Predicate logic as a computational formalism. Res. Report DOC 79/59, Dept. of Computing, Imperial College, London, 1979.
3. M. Comini and G. Levi. An algebraic theory of observables. In *Proc. ILPS '94*, pp. 172–186, The MIT Press, 1994.
4. M. Comini, G. Levi, and M.C. Meo. Compositionality of *SLD*-derivations and their abstractions. In *Proc. ILPS '95*, pp. 561–575, The MIT Press, 1995.
5. A. Cortesi, G. Filé, R. Giacobazzi, C. Palamidessi, and F. Ranzato. Complementation in abstract interpretation. In *Proc. SAS '95*, LNCS 983, pp. 100–117, 1995.
6. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. ACM POPL '77*, pp. 238–252, 1977.
7. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. ACM POPL '79*, pp. 269–282, 1979.
8. P. Cousot and R. Cousot. Inductive definitions, semantics and abstract interpretation. In *Proc. ACM POPL '92*, pp. 83–94, 1992.
9. P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *J. of Logic Programming*, 13(2,3):103–179, 1992.
10. M.H. van Emden and R.A. Kowalski. The semantics of predicate logic as a programming language. *J. of the ACM*, 23(4):733–742, 1976.
11. M. Falaschi and G. Levi. Finite failure and partial computations in concurrent logic languages. *TCS*, 75(7):45–66, 1990.
12. M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. Declarative modeling of the operational behavior of logic languages. *TCS*, 69(3):289–318, 1989.
13. M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. A model theoretic reconstruction of the operational semantics of logic programs. *Inf. & Comp.*, 103(1):86–113, 1993.
14. G. Filé, R. Giacobazzi, and F. Ranzato. A unifying view of abstract domain design. *ACM Comp. Surveys*, 28(2), Symp. on Models of Progr. Lang. and Computation, 1996.
15. G. Filé and F. Ranzato. Complementation of abstract domains made easy. In *Proc. JICSLP '96*, The MIT Press, 1996.
16. M. Gabbrielli and M.C. Meo. Fixpoint semantics for partial computed answer substitutions and call patterns. In *Proc. ALP '92*, LNCS 632, pp. 84-99, 1992.
17. R. Giacobazzi. "Optimal" collecting semantics for analysis in a hierarchy of logic program semantics. In *Proc. STACS '96*, LNCS 1046, pp. 503-514, 1996.
18. R. Giacobazzi, C. Palamidessi, and F. Ranzato. Weak relative pseudo-complements of closure operators. *Algebra Universalis*, 1996. To appear.
19. R. Giacobazzi and F. Ranzato. Functional dependencies and Moore-set completions of abstract interpretations and semantics. In *Proc. ILPS '95*, pp. 321-335, 1995.
20. R. Giacobazzi and F. Ranzato. Compositional optimization of disjunctive abstract interpretations. In *Proc. ESOP '96*, LNCS 1058, pp. 141–155, 1996.
21. A. Mycroft. Completeness and predicate-based abstract interpretation. In *Proc. ACM PEPM '93*, 1993.
22. J. Morgado. Some results on the closure operators of partially ordered sets. *Portug. Math.*, 19(2):101–139, 1960.
23. J. Morgado. Note on complemented closure operators of complete lattices. *Portug. Math.*, 21(3):135–142, 1962.