Functional Dependencies and Moore-Set Completions of Abstract Interpretations and Semantics

Roberto Giacobazzi
Dipartimento di Informatica
Università di Pisa
Corso Italia 40, 56125 Pisa, Italy
giaco@di.unipi.it

Francesco Ranzato
Dipartimento di Matematica Pura ed Applicata
Università di Padova
Via Belzoni 7, 35131 Padova, Italy
franz@hilbert.math.unipd.it

Abstract

We introduce the notion of functional dependencies of abstract interpretations relatively to a binary operator of composition. Functional dependencies are obtained by a functional composition of abstract domains, and provide a systematic approach to construct new abstract domains. In particular, we study the case of autodependencies, namely monotone operators on a given abstract domain. Under suitable hypotheses, this corresponds to a Moore-set completion of the abstract domain, providing a compact lattice-theoretic representation for dependencies. We prove that the abstract domain *Def* for ground-dependency analysis of logic programs can be systematically derived by autodependencies of a more abstract (and simple) domain for pure groundness analysis. Furthermore, we show that functional dependencies can be applied in collecting semantics design by abstract interpretation to systematically derive compositional semantics for logic programs.

1 Introduction

Functional dependencies belong to the standard Cousot and Cousot's theory of abstract interpretation, and have been firstly introduced in [11] as domain combination for attribute dependent (or relational) analyses. In [11] the domain of functional dependencies corresponds precisely to the domain of monotone mappings between two abstract domains. Cousot and Cousot proved that this domain enjoys a Galois insertion under suitable hypotheses, like reduction with respect to concretization.

In this paper we exploit the definition of functional dependency in [11] by extending this notion to the functional composition of abstract domains relatively to a given concrete binary operator of composition, and apply it to data-flow analysis and semantics of logic programs. Our construction is general and can be applied to any abstract interpretation in the standard framework of Cousot and Cousot. The idea is to study the domain of monotone mappings between abstract domains which encode the data-dependencies between the corresponding abstractions *via* a concrete operator of composition. In particular, we prove that simple algebraic properties of the underlying operator of composition naturally induce relevant lattice-theoretic properties of the corresponding domain of functional dependencies. This is particularly significative in the case of *autodependencies*, i.e., the domain of monotone operators on a given abstract domain. In this case, the space of autodependencies can be associated with some well known space of functions, like closure operators, enjoying remarkable lattice-theoretic properties. We introduce the *Moore-set completion* for an abstract domain, and prove that, under suitable hypotheses, the space of autodependencies of an abstract domain corresponds precisely to its (dual-) Moore-set completion. The Moore-set completion provides a different and alternative information with respect to well known techniques for disjunctive completion of abstract domains, like down-set (order ideal) and anti-chain completion ([12, 14]). The Moore-set completion corresponds indeed to upgrade a given abstract domain with its autodependencies relatively to a *lower bound* concrete operator of composition.

Domain composition by functional dependencies provides a uniform approach to study the relational information in data-flow analysis and semantics design by abstract interpretation. In data-flow analysis, new and possibly more precise abstract domains can be derived by functional composition of simpler (abstract) domains. The relation between the Moore-set completion of an abstract domain and the space of its autodependencies is here a key point in order to give an alternative representation for dependencies in abstract domains. In particular, we prove in Subsection 5.1 that the well known domain *Def* of definite propositional formulae ([15]) for ground-dependency analysis of logic programs ([24]), is actually isomorphic to the functional autodependencies of the more simple domain *Ground* (denoted $\mathcal{G}$) for pure groundness analysis ([23, 24]), relatively to the operator of intersection of order ideals of substitutions. Also, *Def* is isomorphic to the dual-Moore-set completion of $\mathcal{G}$. In semantics, abstract interpretation provides a systematic way to derive semantics at different levels of abstraction ([13, 21]). Functional composition can therefore be interpreted as a basic operator for semantic composition, providing a systematic method to design new more descriptive semantics for programming languages. In Subsection 5.2, we show that a compositional semantics for logic programs, with respect to the union of sets of clauses (i.e. programs), can be systematically derived by abstract interpretation from a more concrete semantics of *SLD*-traces. The compositional semantics corresponds to the functional autodependencies of a very abstract (non-compositional) semantics for *computed answer substitutions* ([17]), relatively to an operator of *trace-unfolding*. This provides an alternative characterization of the semantics in [6], and introduces a framework for the definition of new compositional semantics for logic programs. This approach has several advantages with respect to previous techniques (e.g., the approach in [6]). In particular, it can be applied to arbitrary semantics, provided that they can be derived by abstract interpretation from a more concrete (operational) one. This is the case (see [21]) for the semantics of *call patterns* ([19]), *computed answer substitutions* ([17]), *Clark* ([17]) and *Herbrand* ([27]), and other semantics for logic programs. This approach is general and independent from specific semantic representations (e.g., by means of atoms, clauses, *etc.*), and therefore it can be universally applied to design compositional semantics with respect to arbitrary observable properties.

Throughout the paper, we will assume familiarity with the standard notions of lattice theory ([5]) and abstract interpretation ([10, 11]).

2 Preliminaries

If A is a poset 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. If A is a poset with bottom element $\perp$ then an element $a \in A$ is an *atom* if $a \neq \perp$ and for all $x \in A$, $\perp < x \leq a$ implies $x = a$. A lattice is *atomistic* if every element different from $\perp$ is the join of the atoms which precede it. The complete lattice of total (monotone, continuous) functions from the complete lattice A into the complete lattice B equipped with the functional pointwise order (i.e., $f \sqsubseteq g$ iff $\forall x \in A.\ f(x) \leq_B g(x)$) is denoted $A \rightarrow B$ ($A \rightarrow^m B$, $A \rightarrow^c B$). Let $\langle A, \leq, \wedge, \vee, \top, \perp \rangle$ be a complete lattice (sometimes simply denoted $A_{\leq}$). An *upper* (*lower*) *closure operator* on A is a map $\rho : A \rightarrow A$ monotone, idempotent and extensive (reductive), i.e., for every $x \in A$, $x \leq \rho(x)$ ($\rho(x) \leq x$). Any (upper or lower) closure operator ρ is uniquely determined by the set of its fixpoints, which is its image $\rho(A) = fp(\rho)$. We denote $uco(A)$ ($lco(A)$) the set of all upper (lower) closure operators on A. $uco(A)$ and $lco(A)$ are complete lattices with respect to the pointwise ordering. If $\rho \in uco(A)$ ($\rho \in lco(A)$) then $\rho(A)$ is a (*dual-*)*Moore-set*, i.e. a subset of A which contains $\top$ ($\perp$) and it is closed with respect to the *glb* (*lub*). If A and D are posets and $\alpha : D \rightarrow A$, $\gamma : A \rightarrow D$ are monotone maps such that $\forall x \in D.\ x \leq_D \gamma(\alpha(x))$ and $\forall y \in A.\ \alpha(\gamma(y)) \leq_A y$, then we call the quadruple (D, α, A, γ) a *Galois connection* (G.c.) between D and A. If in addition $\forall y \in A.\ \alpha(\gamma(y)) = y$, then we call (D, α, A, γ) a *Galois insertion* (G.i.) of A in D. In the setting of abstract interpretation, D and A are called, respectively, 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. Also, A is called an *abstract interpretation* (or *abstraction*) of D. If (D, α, A, γ) is a G.i., then the concretization and abstraction mappings, γ and α, are 1-1 and onto, respectively. Moreover, any G.c. may be lifted to a G.i. identifying in an equivalence class those values of the abstract domain with the same concrete meaning. This process is known as *reduction* of the abstract domain.

In the following, Σ, Π and *Var* will respectively denote a set of function symbols, a set of predicate symbols and a denumerable set of variables defining a first-order language $\mathcal{L}$. The set of terms, atoms and Horn clauses over $\mathcal{L}$ are denoted *Term*, *Atom* and *Clause*, respectively. $Term_{\emptyset}$ and $Atom_{\emptyset}$ denote the corresponding sets of ground objects. The set of (definite) logic programs is denoted *Program*. Sequences of objects with the same type (variables, atoms, *etc.*) are sometimes denoted $\bar{s}$. Concatenation of sequences is denoted ::. The empty sequence is Λ, while $a \mid \bar{s}$ denotes a sequence with first element a. If A is a set of syntactic objects then $inst(A)$ and $ground(A)$ denote respectively the set of instances and ground instances of elements in A. We restrict our interest to idempotent substitutions ranging in *Sub*, unless explicitly stated otherwise. A syntactic object t is *more general* than t' (denoted $t' \leq t$) iff there exists a substitution θ such that $t' = t\theta$. 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 $t_2 \leq t_1$. $Term_{/\sim}$, $Atom_{/\sim}$, $Clause_{/\sim}$ and $Sub_{/\sim}$ are complete lattices with respect to $\leq$. With abuse of notation, they will be often denoted *Term*, *Atom*, *Clause* and *Sub*. 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.

3 Functional dependencies of abstract interpretations

In this section we study the functional dependencies between abstract domains relatively to a fixed basic operator of composition for concrete denotations. The basic underlying structure is called *semantic structure*, and it is defined as a complete lattice together with the binary operator of composition. This name is justified by its use in the semantic definition of logic programming, as shown later in Section 5.

Definition 3.1 A *semantic structure* is a pair $\langle D_{\leq}, \odot \rangle$, where $D_{\leq}$ is a complete lattice and $\odot : D \times D \rightarrow D$ is a monotone binary operator on D. □

We consider functional dependencies relatively to a fixed semantic structure $\langle D_{\leq}, \odot \rangle$, namely we consider the domain of functional dependencies of denotations in D relatively to the basic composition operator $\odot$.

Definition 3.2 Let $\langle D_{\leq}, \odot \rangle$ be a semantic structure, $\gamma_1 : D_1 \rightarrow D$ be a monotonic function between complete lattices and $(D, \alpha_2, D_2, \gamma_2)$ be a Galois connection. The domain of *functional dependencies from* D_1 *into* D_2 *relatively to* $\odot$ is the set of functions $\{\lambda x.\alpha_2(d \odot \gamma_1(x)) \in D_1 \rightarrow D_2 \mid d \in D\}$, partially ordered by the pointwise functional ordering relation, and denoted by $D_1 \rightarrow^{\odot} D_2$. □

It is worth noting that $|D_1 \rightarrow^{\odot} D_2| \leq |D|$. Moreover, in the following we will often assume that γ_1 belongs to a Galois connection, i.e. $(D, \alpha_1, D_1, \gamma_1)$ is a G.c.. In this case we call D_1 and D_2 the *domain* and *range* abstract interpretations.

The above definition of functional dependencies generalizes the original proposal of Cousot and Cousot in [11]. A functional dependency is here associated with any concrete denotation $d \in D$, and it is a mapping which associates with any abstract value $x \in D_1$ an element of D_2 which corresponds to the composition of d and x *via* $\odot$. Therefore, a functional dependency in $D_1 \rightarrow^{\odot} D_2$, for a concrete object $d \in D$, encodes how d reacts when composed with any object described by the domain abstract interpretation D_1, with respect to the range abstract interpretation D_2.

Clearly, each functional dependency $\lambda x.\alpha_2(d \odot \gamma_1(x))$ is a monotone mapping, i.e., $D_1 \rightarrow^{\odot} D_2 \subseteq D_1 \rightarrow^{m} D_2$. The poset of functional dependencies $D_1 \rightarrow^{\odot} D_2$ is indeed a complete lattice, where the *lub* coincides with that of $D_1 \rightarrow^{m} D_2$, whenever $\odot$ is *left-additive*.

Proposition 3.3 *If $\odot$ is left-additive then $D_1 \rightarrow^{\odot} D_2$ is a complete join-sublattice of $D_1 \rightarrow^{m} D_2$.*

In general, the lattice of monotone mappings $D_1 \rightarrow^{m} D_2$ cannot be viewed as an abstract interpretation of D, unless $\odot$ is left-additive, as specified below.

Theorem 3.4 *If $\odot$ is left-additive then $(D, \alpha, D_1 \rightarrow^{m} D_2, \gamma)$ is a Galois connection, where for any $d \in D$, $\alpha(d) = \lambda x \in D_1.\alpha_2(d \odot \gamma_1(x))$.*

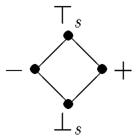
It is now simple to associate with $(D, \alpha, D_1 \rightarrow^{m} D_2, \gamma)$ a corresponding Galois insertion by reducing the abstract domain $D_1 \rightarrow^{m} D_2$.

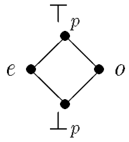
Corollary 3.5 *Under the hypotheses of Theorem 3.4, $D_1 \rightarrow^{\odot} D_2$ is isomorphic to $(D_1 \rightarrow^{m} D_2)_{/\equiv_\gamma}$, where $f \equiv_\gamma g$ iff $\gamma(f) = \gamma(g)$, for any $f, g \in D_1 \rightarrow^{m} D_2$.*

In view of the above basic results, in the rest of the paper we will take for granted the following assumption.

Assumption 3.6 *In any semantic structure $\langle D_{\leq}, \odot \rangle$, $\odot$ is left-additive.* □

Example 3.7 Consider the following two abstract domains *Sign* and *Parity* for *sign* and *parity analysis* of an integer variable ([10, 11]).





The corresponding G.i.'s are $(\wp(\mathbb{Z})_{\subseteq}, \alpha_s, Sign, \gamma_s)$, $(\wp(\mathbb{Z})_{\subseteq}, \alpha_p, Parity, \gamma_p)$, where abstraction and concretization mappings are the most natural.
The domain of functional dependencies from *Sign* into *Parity* relatively to the *glb* $\cap$ of $\wp(\mathbb{Z})$ is $Sign \rightarrow^{\cap} Parity = \{\lambda x.\alpha_p(S \cap \gamma_s(x)) \in Sign \rightarrow Parity \mid S \in \wp(\mathbb{Z})\}$, and, since $\cap$ is obviously left-additive, by Corollary 3.5, it enjoys a Galois insertion into $\wp(\mathbb{Z})$. This lattice consists of twenty elements. It is a very expressive abstract interpretation since it allows to represent several sets of integers which are not representable neither in *Sign* nor in *Parity*. For example, this abstract interpretation is able to represent the sets of integers below.

- *The null value*

$$\lambda x.\alpha_p(\{0\} \cap \gamma_s(x)) = \begin{cases} \top_s & \mapsto e \\ - & \mapsto \bot_p \\ + & \mapsto \bot_p \\ \bot_s & \mapsto \bot_p \end{cases} \stackrel{\gamma}{\longmapsto} \{0\}$$

- *The positive and even integers*

$$\lambda x.\alpha_p(\{x \in \mathbb{Z} \mid x > 0,\ x \text{ even}\} \cap \gamma_s(x)) = \begin{cases} \top_s & \mapsto e \\ - & \mapsto \bot_p \\ + & \mapsto e \\ \bot_s & \mapsto \bot_p \end{cases} \stackrel{\gamma}{\longmapsto} \{x \in \mathbb{Z} \mid x > 0,\ x \text{ even}\}$$

□

4 Autodependencies and Moore-set completions

A relevant case of functional dependencies between abstract interpretations is given by the autodependencies of an abstract domain.

Definition 4.1 Let $\langle D_{\leq}, \odot \rangle$ be a semantic structure, and (D, α, A, γ) be a Galois connection. The domain of *autodependencies of A relatively to $\odot$* is $A \rightarrow^{\odot} A$. □

Therefore, $A \rightarrow^{\odot} A = \{\lambda x.\alpha(d \odot \gamma(x)) \in A \rightarrow^{m} A \mid d \in D\}$. Corollary 3.5 implies that autodependencies enjoy a Galois insertion into D (whenever $\odot$ is left-additive). In the following, we adopt a more concise notation and often denote by $Dep_{\odot}(A)$ the domain of autodependencies $A \rightarrow^{\odot} A$. Note that $Dep_{\odot}(A)$ is precisely the set of best correct approximations ([11]) of the family of concrete operators $\lambda x.d \odot x \in D \rightarrow^{m} D$, for $d \in D$. Moreover, note that when $\odot$ is an idempotent operator (i.e., $\forall x.\ x \odot x = x$) which implements a lower (upper) bound operator on D (i.e., $\forall x, y.\ x \odot y \leq x, y$ ($x, y \leq x \odot y$)), then $\lambda x.d \odot x \in lco(D)$ ($\lambda x.d \odot x \in uco(D)$), for any $d \in D$.

The case of autodependencies for an abstract interpretation is important because it is always possible to identify some special cases where $Dep_{\odot}(A)$ corresponds to well known function spaces. In particular, when $\odot$ implements an idempotent lower (upper) bound operator on D, we obtain lower (upper) closure operators as denotations for autodependencies, as specified by the following result.

Theorem 4.2 *Let (D, α, A, γ) be a G.c. and $\odot$ be an idempotent binary operator.*

(i) If $\odot$ is a lower bound such that for any $d, d' \in D$, $\gamma(\alpha(d \odot d')) \leq d \odot \gamma(\alpha(d'))$, then $Dep_{\odot}(A) \subseteq lco(A)$.

(ii) If (D, α, A, γ) is a G.i. and $\odot$ is an upper bound such that for any $d, d' \in D$, $d \odot \gamma(\alpha(d')) \leq \gamma(\alpha(d \odot d'))$, then $Dep_{\odot}(A) \subseteq uco(A)$.

Whenever $\odot$ coincides with the *glb* of D, i.e. $\odot = \wedge$, Assumption 3.6 of left-additivity of $\odot$ corresponds to the requirement that D is *infinitely inf-distributive*. Further, whenever $\odot = \vee$, Assumption 3.6 is obviously always satisfied. We call the dependencies in $Dep_{\wedge}(A)$ and $Dep_{\vee}(A)$, *meet-autodependencies* and *join-autodependencies*, respectively. It is possible to show that in such cases the hypotheses for $\odot$ in *(i)* and *(ii)* of Theorem 4.2 are satisfied.

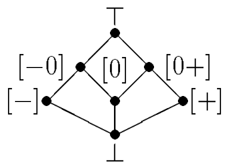
Corollary 4.3

(i) If D is infinitely inf-distributive then $Dep_{\wedge}(A)$ is a complete join-sublattice of $lco(A)$.

(ii) If (D, α, A, γ) is a G.i. then $Dep_{\vee}(A) \subseteq uco(A)$.

The following is a simple example of meet-autodependencies.

Example 4.4 Consider the domain *Sign* of Example 3.7. Taking as concrete composition operator the *glb* $\cap$ of $\wp(\mathbb{Z})$, by Corollary 4.3 *(i)*, we have that $Dep_{\cap}(Sign) \subseteq lco(Sign)$. It is possible to verify that $Dep_{\cap}(Sign) = lco(Sign)$, and they correspond precisely to the following abstract domain, which includes denotations for possibly null values.



The key point in this example is that $Dep_{\cap}(Sign)$ captures the null value [0], which is encoded by the lower closure $\lambda x.\alpha(\{0\} \cap \gamma(x))$, which corresponds to its dual-Moore-set of fixpoints $\{\top, \bot\}$. This provides the basis to define denotations for sets of non-negative and non-positive values, denoted respectively by [0+] and [−0], and corresponding to the lower closure operators $\lambda x.\alpha(I^+ \cap \gamma(x))$ and $\lambda x.\alpha(I^- \cap \gamma(x))$, where I^+ and I^- are any element in $\{I \cup \{0\} \in \wp(\mathbb{Z}) \mid I \neq \emptyset, \forall x \in I.\ x > 0\}$ and $\{I \cup \{0\} \in \wp(\mathbb{Z}) \mid I \neq \emptyset, \forall x \in I.\ x < 0\}$, respectively. The set of fixpoints of these closures are, respectively, $\{\top, +, \bot\}$ and $\{\top, -, \bot\}$, and they represent precisely the sets of non-negative and non-positive integers, respectively. $\square$

We now introduce a powerset completion for abstract interpretations which provides a representation for spaces of functional autodependencies of abstract domains.

Definition 4.5 Let (D, α, A, γ) be a Galois connection. The *(dual-)Moore-set completion* of A is $\mathcal{M}(A)$ $(\mathcal{DM}(A))$, where $\mathcal{M}(A) = \{M \subseteq A \mid M \text{ is a Moore-set}\}$ $(\mathcal{DM}(A) = \{M \subseteq A \mid M \text{ is a dual-Moore-set}\})$, ordered by the supset (subset) relation. □

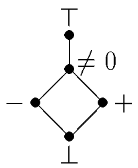
It is well known that $\mathcal{M}(A)$ is (isomorphic to) the complete lattice $uco(A)$ of upper closure operators on A, while $\mathcal{DM}(A)$ is (isomorphic to) the complete lattice $lco(A)$ of lower closure operators on A ([28]). The following two results relate the dual-Moore-set completion $\mathcal{DM}(A)$ to the abstract and concrete domains A and D.

Theorem 4.6 *$(\mathcal{DM}(A), \widehat{\alpha}, A, \widehat{\gamma})$ is a Galois insertion, where $\widehat{\alpha}(M) = \vee_A M$ and $\widehat{\gamma}(a) = \downarrow a$, for any $M \in \mathcal{DM}(A)$ and $a \in A$.*

Theorem 4.7 *Under the hypotheses of Theorem 4.2 (i), $(D, \tilde{\alpha}, \mathcal{DM}(A), \tilde{\gamma})$ is a Galois connection, where for any $d \in D$, $\tilde{\alpha}(d) = \{x \in A \mid \alpha(d \odot \gamma(x)) = x\}$.*

Therefore, the dual-Moore-set completion of an abstract domain is an abstract interpretation (i.e. it enjoys a Galois insertion into the concrete domain), when reduced relatively to the corresponding autodependencies. Autodependencies are then the right way to interpret dual-Moore-set completions. Moreover, by Theorem 4.6, the dual-Moore-set completion is a *domain refinement*, namely it contains the original domain as an abstract interpretation. Note that the dual-Moore-set completion cannot be compared with the disjunctive completions introduced in [12, 14]. We consider the case of the *down-set completion* in the following example.

Example 4.8 Consider the domain *Sign* of Example 3.7. As pointed out in Example 4.4, the dual-Moore-set completion of *Sign* (viz. $lco(Sign)$) is exactly $Dep_{\cap}(Sign)$. By contrast, the down-set (or order ideal) completion of *Sign* is the domain $\wp^{\downarrow}(Sign)$ depicted below.



The dual-Moore-set completion of *Sign*, $Dep_{\cap}(Sign)$, does not provide a denotation for non-null values, which instead can be obtained by the down-set completion of *Sign*, $\wp^{\downarrow}(Sign)$, notably by the order ideal $\{+, -, \bot\}$. This example shows that the dual-Moore-set completion does not correspond to the disjunctive down-set completion. □

The following result shows that the domain A is always an abstract interpretation of its meet-autodependencies $Dep_{\wedge}(A)$, therefore extending Theorem 4.6 when[1] $Dep_{\wedge}(A) \subset lco(A)$ (cf. Corollary 4.3 *(i)*).

Proposition 4.9 *If D is infinitely inf-distributive then $(Dep_{\wedge}(A), \bar{\alpha}, A, \bar{\gamma})$ is a Galois insertion, where $\bar{\alpha}(\lambda x.\alpha(d \wedge \gamma(x))) = \vee_A\{y \in A \mid \alpha(d \wedge \gamma(y)) = y\}$ and $\bar{\gamma}(y) = \lambda x.y \wedge x$, for any $d \in D$ and $y \in A$.*

We now give a sufficient condition on abstract interpretations (D, α, A, γ) in order that meet-autodependencies of A and the dual-Moore-set completion of A coincide, i.e. $Dep_{\wedge}(A) = lco(A)$.

[1] We write $S \subset T$ if S is a proper subset of T.

Definition 4.10 A Galois insertion (D, α, A, γ) is *separated* if for any $x \in A$, there exists $d_x \in D$ such that $\alpha(d_x) = x$, and for each $y \in A$ such that $x \not\leq_A y$, $d_x \wedge \gamma(y) = \bot_D$ holds. □

It is well known ([11]) that in any G.i. (D, α, A, γ), the abstract domain A is isomorphic (by γ) with a (Moore-)set $A_D \subseteq D$. In separated abstract interpretations the elements in A_D are all *join-irreducible*, i.e., for any $x \in A_D$, $x \neq \bot_D$, and $Y \subseteq A_D$, $x = \vee_D Y \Rightarrow x \in Y$. Intuitively, in separated abstract interpretations each abstract denotation represents some concrete objects which cannot be described through the other denotations. Examples of separated abstract interpretations are the domain *Sign* of Example 3.7 and the abstract domain $\mathcal{G}$ of Subsection 5.1 for pure groundness analysis of logic programs. Note that both $Dep_{\cap}(Sign)$ and $\wp^{\downarrow}(sign)$ (cf. Example 4.4 and 4.8) are not separated (take x as $[-0]$ and $\neq 0$ respectively).

Theorem 4.11 *If (D, α, A, γ) is a separated Galois insertion and D is infinitely inf-distributive, then $Dep_{\wedge}(A) = lco(A)$.*

When $Dep_{\wedge}(A) = lco(A)$, the domain of meet-autodependencies enjoys the following lattice-theoretic properties inherited from $lco(A)$: (1) $Dep_{\wedge}(A)$ is atomistic ([28]); (2) $Dep_{\wedge}(A)$ is distributive or complemented or a Boolean algebra iff A is a complete well-ordered chain ([25]); and (3) if A satisfies the descending chain condition then $Dep_{\wedge}(A)$ is dual-pseudo-complemented ([22]), namely for any $x \in Dep_{\wedge}(A)$, there exists a unique $x^* \in Dep_{\wedge}(A)$ such that $x \sqcup x^* = \top$, and if $x \sqcup y = \top$ then $x^* \sqsubseteq y$, for any $y \in Dep_{\wedge}(A)$. Atomicity is an interesting property in abstract interpretation: atoms represent *primitive* properties, and each (non primitive) property can be generated by collecting primitive ones.

It is possible to see that in the case of join-autodependencies of A, $Dep_{\vee}(A)$ is always isomorphic to A.

Proposition 4.12 *If (D, α, A, γ) is a Galois insertion then $Dep_{\vee}(A) \cong A$.*

Therefore, we can conclude that while the meet-autodependencies may capture the dual-Moore-set completions, the join-autodependencies does not add any further information to the original domain of abstraction.

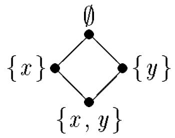
5 Applications: data-flow analysis and semantics

In this section, we consider two examples of applications of functional dependencies in the context of data-flow analysis of logic programs and systematic design of semantics for logic programs by abstract interpretation.

5.1 Systematic derivation of ground-dependency analysis

We show that the abstract domain *Def*, introduced by Dart in [15], and used by Marriott and Søndergaard for ground-dependency analysis in [24], can be systematically derived by meet-autodependencies of the more abstract (and simple) domain for pure groundness analysis $\mathcal{G}$ ([23, 24]), relatively to the set-theoretic intersection of order ideals of concrete substitutions.

Consider the following semantic structure for sets of concrete substitutions $\langle \wp^{\downarrow}(Sub)_{\subseteq}, \cap \rangle$, containing sets of substitutions closed by instantiation, viz. order ideals. The analysis of pure groundness for sets of substitutions in $\wp^{\downarrow}(Sub)$, relatively to a (non-empty) finite set of variables of interest $V \subseteq Var$, can be obtained by the Galois insertion $(\wp^{\downarrow}(Sub), \alpha_g, \wp(V)_{\supseteq}, \gamma_g)$, where for any $\Theta \in \wp^{\downarrow}(Sub)$ and $W \in \wp(V)$, $\alpha_g(\Theta) = \{x \in V \mid \forall \sigma \in \Theta.\ var(\sigma(x)) = \emptyset\}$, while $\gamma_g(W) = \{\sigma \in Sub \mid \forall x \in W.\ var(\sigma(x)) = \emptyset\}$. It is worth noting that γ_g is closed by instantiation (i.e., if $\sigma \in \gamma_g(W)$ and $\sigma' \leq \sigma$ then $\sigma' \in \gamma_g(W)$), and therefore, correctly, $\gamma_g(W)$ is an order ideal. We denote by $\mathcal{G}_V$ the abstract domain $\wp(V)_{\supseteq}$, which is, clearly, a finite distributive lattice. In the case of two variables, $\mathcal{G}_{\{x,y\}}$ is depicted below.



It is clear that Boolean functions can be represented by means of propositional formulae ([2, 24]). Recall that a Boolean function f is *positive* if $f(true, \ldots, true) = true$. Def_V is the finite lattice (with respect to the usual implication partial order) of *definite* formulae on V, i.e. positive Boolean functions on V whose models are closed under intersection (for more details see [2]). The abstraction and concretization maps between Def and $\wp^{\downarrow}(Sub)$ are well known, and can be found, e.g., in [24]. Also in this case, we remark that the concretization of any definite formula is an order ideal of substitutions. As an example, assuming $V = \{x, y, z, u\}$, the formula $x \wedge (y \leftrightarrow z)$ is an element of Def_V that represents all the substitutions σ such that for any instance σ' of σ the following conditions hold: the term $\sigma'(x)$ is ground, and $\sigma'(y)$ is ground iff $\sigma'(z)$ is ground. In particular, $\sigma_1 = \{x/a, y/b, z/c\}$ and $\sigma_2 = \{x/a, y/w, z/w, v/u\}$ satisfy this property. Thus, $\{\sigma_1, \sigma_2\} \subseteq \gamma(x \wedge (y \leftrightarrow z))$.

Clearly, the domain for groundness analysis $\mathcal{G}_V$ may be equivalently represented by means of conjunctive propositional formulae on V, by associating any $W \in \mathcal{G}_V$ with the formula $\wedge W$ (viz. the conjunction of all the variables in W).

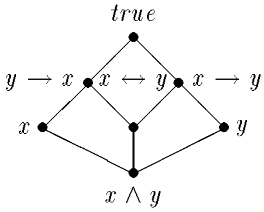
Def_V can be systematically derived by lifting $\mathcal{G}_V$ to its autodependencies, or, equivalently, by dual-Moore-set completion of $\mathcal{G}_V$.

Proposition 5.1 *$(\wp^{\downarrow}(Sub), \alpha_g, \mathcal{G}_V, \gamma_g)$ is a separated abstract interpretation.*

Therefore, by Theorem 4.11, $Dep_{\cap}(\mathcal{G}_V) = lco(\mathcal{G}_V) = \mathcal{DM}(\mathcal{G}_V)$. The following result provides an alternative lattice-theoretic characterization of Def in terms of meet-autodependencies of groundness (or, alternatively, as dual-Moore-set completion of groundness).

Theorem 5.2 *$Def_V \cong Dep_{\cap}(\mathcal{G}_V) = \mathcal{DM}(\mathcal{G}_V)$.*

For the case of $Def_{\{x,y\}}$, we obtain the same lattice structure in Example 4.4.



The correspondence between Def-formulae and dual-Moore-sets of $\mathcal{G}$ is given by the model-theoretic interpretation of Def-formulae. For $V = \{x, y\}$ we have: $x \wedge y = \{\{x, y\}\}$, $x = \{\{x\}, \{x, y\}\}$, $y = \{\{y\}, \{x, y\}\}$, $x \leftrightarrow y = \{\emptyset, \{x, y\}\}$, $x \rightarrow y = \{\emptyset, \{y\}, \{x, y\}\}$, $y \rightarrow x = \{\emptyset, \{x\}, \{x, y\}\}$, and $true = \{\emptyset, \{x\}, \{y\}, \{x, y\}\}$.

It is worth noticing that the operation of conjunction, which usually implements abstract unification in ground-dependency analysis by Def, can be easily implemented by set-intersection on the corresponding dual-Moore-set completion. Furthermore, Def shares the lattice-theoretic properties of lower closures, being atomistic and dual-pseudo-complemented. In particular, by the first property, any element in Def_V can be represented as the join of the atoms that imply it. An atom is here a dual-Moore-set $\{W, V\}$, where $W \in \wp(V)$. For instance, the dependency $x \rightarrow y$ can be represented by joining the atomic dual-Moore sets $\{\{y\},\{x,y\}\}$ and $\{\emptyset,\{x,y\}\}$. This may provide a concise representation for ground-dependencies. It is worth noting that $Dep_{\cap}(Def) \neq Def$, namely Def is not *dependency closed.* In particular, the dependency relation $(x \rightarrow y) \rightarrow y$ is precisely $x \vee y \notin Def$.[2]

5.2 Systematic derivation of compositional semantics

In this subsection we systematically derive compositional semantics for logic programs by composing semantics according to their functional dependencies. In the following, a semantics for logic programs is a pair $\langle C, T\rangle$, where $C_{\sqsubseteq}$ is a complete lattice (the semantic domain) and $T : Program \rightarrow (C \xrightarrow{c} C)$. $T(P)$ is often denoted T_P, for $P \in Program$.

As Cousot and Cousot proved in [13], semantics of a programming language can all be derived by abstract interpretation of a more concrete *reference semantics.* A natural choice for a reference concrete semantics is the operational description of the computational process, which is in logic programming *SLD resolution.* In the following, we recall some basic results from [21], where a hierarchy of semantics for logic programs has been introduced, based on standard *SLD* resolution. We fix the Prolog left-to-right selection rule. The operational semantics of a logic program P is defined as execution traces in a labeled transition system defining *SLD* resolution as follows: $SLD = \langle State, \{\xrightarrow{c} \mid c \in P\}\rangle$, where states are goals $State = Atom^*$, and transitions are labeled with program clauses, $\xrightarrow{c} \subseteq State \times State$, such that $a \mid \bar{b} \xrightarrow{c} (body :: \bar{b})\vartheta$ iff $c = h \leftarrow body$ is a renamed apart clause in P, and $\vartheta = mgu(a, h)$. The transitive closure of $\longrightarrow$ is denoted $\xrightarrow{\bar{c}}{}^*$, where $\bar{c} \in P^*$. We denote $\mathcal{T}_P(SLD)$ the set of (finite) execution traces of SLD, with arbitrary elements π. π_0 denotes the first element of the trace π. $\mathcal{T}_P(SLD)$ is inductively defined by the rules:

$$\frac{s \in State}{s \in \mathcal{T}_P(SLD)} \qquad \frac{s \xrightarrow{c} \pi_0 \ \wedge \ \pi \in \mathcal{T}_P(SLD)}{s \xrightarrow{c} \pi \in \mathcal{T}_P(SLD)}.$$

A key point in this construction is that execution traces can be equivalently represented by restricting the interest to *AND-compositional execution traces* only. Intuitively a set of traces is AND-compositional if the execution trace of any (possibly non-atomic) goal can be reconstructed by composing traces for its atomic subgoals. The inductive definition of the set $\mathcal{E}$ of AND-compositional execution traces for atomic goals is given in [21] as in Table 1. Rules 1 and 2 specify a big-step semantics as (positive) inductive definition with universe the set of traces from atomic goals only, denoted $\mathcal{T}_P^a(SLD) \subseteq \mathcal{T}_P(SLD)$. The first rule specifies an atomic transition from an atomic goal h with clause c. The second rule specifies the AND-compositionality of derivations for a clause c. This is obtained by composing the successful transitions for the first $k-1$ atoms of the body, with the state (goal) produced from a derivation of the k^{th} atom of the body. The following theorem justifies the interest in AND-compositional traces, relating SLD-traces with AND-compositional ones.

[2]This observation is due to Roberto Bagnara.

$$
1\ \frac{c = h \leftarrow b_1, ..., b_n \in P}{h \xrightarrow{c} \langle b_1, ..., b_n\rangle \in \mathcal{E}}
\qquad
2\ \frac{\begin{array}{c} c = h \leftarrow b_1, ..., b_n \in P \\ \langle\langle a_i \xrightarrow{\bar{c}_i} \Lambda\rangle_{i=1}^{k-1}, a_k \xrightarrow{\bar{c}_k} \bar{b}\rangle \ll_c \mathcal{E} \ \ (1 \le k \le n) \\ \theta = mgu(\langle b_1, ..., b_k\rangle, \langle a_1, ..., a_k\rangle) \end{array}}{(h \xrightarrow{c} \langle b_1, ..., b_n\rangle \xrightarrow{\bar{c}_1} ... \xrightarrow{\bar{c}_k} \bar{b} :: \langle b_{k+1}, ..., b_n\rangle)\theta \in \mathcal{E}}
$$

Table 1: AND-compositional traces

Theorem 5.3 ([21]) *Let $G = g_1, ..., g_n$ be a goal. $G \xrightarrow{\bar{c}}{}^{+} \bar{b} \in \mathcal{T}_P(SLD)$ iff there exist $\langle\langle h_i \xrightarrow{\bar{c}_i} \Lambda\rangle_{i=1}^{k-1}, h_k \xrightarrow{\bar{c}_k} \bar{b}_k\rangle \ll_G \mathcal{E}$ $(1 \le k \le n)$ s.t. $\delta = mgu(\langle g_1, ..., g_k\rangle, \langle h_1, ..., h_k\rangle)$, $\bar{c} = \bar{c}_1 :: ... :: \bar{c}_k$, and $\bar{b} \sim (\bar{b}_k :: \langle g_{k+1}, ..., g_n\rangle)\delta$.*

As usual (cf. [1]), a continuous operator φ_P on $\wp(\mathcal{T}_P^a(SLD))$ can be systematically derived by the inductive definition of $\mathcal{E}$ in Table 1, such that $\mathcal{E} = lfp(\varphi_P)$. In the following, we consider $\langle \wp(\mathcal{T}_P^a(SLD))_{\subseteq}, \varphi_P\rangle$ as the *reference semantics*. Elements in $\wp(\mathcal{T}_P^a(SLD))$ are called *trace interpretations*. This semantics models AND-compositional execution traces for atomic goals, yet providing a fixpoint operational semantics for logic programs which is equivalent to SLD resolution by Theorem 5.3.

The semantic domain $\wp(\mathcal{T}_P^a(SLD))$ can be extended to a semantic structure by considering the *trace-unfolding* as basic binary operator for trace composition.

Definition 5.4 Define $\nabla : \wp(\mathcal{T}_P^a(SLD)) \times \wp(\mathcal{T}_P^a(SLD)) \to \wp(\mathcal{T}_P^a(SLD))$ such that for any $X, Y \in \wp(\mathcal{T}_P^a(SLD))$,

$$
X \nabla Y = \left\{ \pi\vartheta \;\middle|\; \begin{array}{l} c = h \xrightarrow{\bar{c}} b_1, ..., b_n \in X, \\ \langle\langle a_i \xrightarrow{\bar{c}_i} \Lambda\rangle_{i=1}^{k-1}, a_k \xrightarrow{\bar{c}_k} \bar{b}\rangle \ll_c Y \ \ (1 \le k \le n) \\ \vartheta = mgu(\langle b_1, ..., b_k\rangle, \langle a_1, ..., a_k\rangle), \\ \pi = h \xrightarrow{\bar{c}} \langle b_1, ..., b_n\rangle \xrightarrow{\bar{c}_1} ... \xrightarrow{\bar{c}_k} \bar{b} :: \langle b_{k+1}, ..., b_n\rangle \end{array} \right\}.
$$

□

The trace-unfolding operator above satisfies the following basic algebraic properties.

Proposition 5.5 *The operator ∇ is left-additive, i.e. $(\cup_{i\in I} X_i)\nabla X = \cup_{i \in I}(X_i \nabla X)$, associative ([16]), and $\emptyset$ is a left-annihilator for ∇, i.e. $\emptyset \nabla X = \emptyset$.*

Hence, $\langle \wp(\mathcal{T}_P^a(SLD))_{\subseteq}, \nabla\rangle$ is a semantic structure where the basic Assumption 3.6 is satisfied. In particular, Corollary 3.5 implies that if $(\wp(\mathcal{T}_P^a(SLD)), \alpha_1, D_1, \gamma_1)$ and $(\wp(\mathcal{T}_P^a(SLD)), \alpha_2, D_2, \gamma_2)$ are G.c.'s, then $(\wp(\mathcal{T}_P^a(SLD)), \alpha, D_1 \rightarrow^{\nabla} D_2, \gamma)$ is a Galois insertion, where for any trace interpretation $I \in \wp(\mathcal{T}_P^a(SLD))$, $\alpha(I) = \lambda x.\alpha_2(I \nabla \gamma_1(x))$. A trace interpretation is therefore abstracted in $D_1 \rightarrow^{\nabla} D_2$ by a function that encodes the transformation by unfolding of traces approximated in D_1 into traces approximated in D_2.

The key point here is that each Galois connection $(\wp(\mathcal{T}_P^a(SLD)), \alpha, D, \gamma)$ actually induces an (abstract) semantics for logic programs $\langle D, \alpha \circ \varphi_P \circ \gamma\rangle$, which is clearly correct[3] with respect to the reference semantics $\langle \wp(\mathcal{T}_P^a(SLD)), \varphi_P\rangle$, yet providing the best correct approximation for φ_P in D. Therefore, any pair of semantics which can be derived from $\langle \wp(\mathcal{T}_P^a(SLD)), \varphi_P\rangle$ by abstract interpretation, can be functionally composed, yet providing a semantics for logic programs.

[3]In the sense of abstract interpretation ([10]), i.e. $\alpha(lfp(\varphi_P)) \le_D lfp(\alpha \circ \varphi_P \circ \gamma)$.

The following is a list of some well known semantics for logic programs that can all be derived by abstract interpretation from $\langle \wp(\mathcal{T}_P^a(SLD)), \varphi_P \rangle$. Some of them have been used in [4, 8, 18] for data-flow analysis of logic programs. Each semantics is provided with the corresponding abstraction, hence with a G.c. (indeed insertion). A more complete treatment for semantics derivable by abstract interpretation from a reference semantics is in [9, 21]. In what follows, we denote γ_x the unique upper adjoint to α_x for $x \in \{s, Call, \mathcal{S}, \mathcal{C}, \mathcal{H}\}$, and assume $X \in \wp(\mathcal{T}_P^a(SLD))$.

• The *sequence abstraction* α_s is obtained by approximating finite traces by the pair of their initial and final states in a clause-like form. $\alpha_s : \wp(\mathcal{T}_P^a(SLD)) \rightarrow \wp(Clause)$ is such that $\alpha_s(X) = \{h \leftarrow \bar{b} \mid h \xrightarrow{\bar{c}}{}^* \bar{b} \in X\}$. This abstraction induces a semantics which is equivalent to the semantics for *partial answers* in [19]. The semantics for *call patterns* in [19] can also be derived by further approximating partial answers. Recall that an atom $a \in Atom$ is a *call pattern* for a goal G in a program P if $G \xrightarrow{\bar{c}}{}^* a \mid \bar{b} \in \mathcal{T}_P(SLD)$. The corresponding abstraction $\alpha_{Call} : \wp(\mathcal{T}_P^a(SLD)) \rightarrow \wp(BClause)$ is $\alpha_{Call}(X) = \{h \leftarrow a \mid h \xrightarrow{\bar{c}}{}^* a \mid \bar{b} \in X\}$, and induces a semantics which is equivalent to $Call = \langle \wp(BClause)_{\subseteq}, T_P^{Call} \rangle$ (i.e., $T_P^{Call} = \alpha_{Call} \circ \varphi_P \circ \gamma_{Call}$), where denotations are sets of *binary clauses* ranging in $BClause$, representing call patterns for atomic goals, and where by defining $\Phi = \{h \leftarrow h \mid h \in Atom\}$, for each $I \in \wp(BClause)$:

$$T_P^{Call}(I) = \left\{ (h \leftarrow b)\vartheta \;\middle|\; \begin{array}{l} c = h \leftarrow b_1, ..., b_n \in P \ (1 \leq k \leq n) \\ \langle\langle a_i \leftarrow \Lambda\rangle_{i=1}^{k-1}, \langle a_k \leftarrow b\rangle\rangle \ll_c X \cup \Phi \\ \vartheta = mgu(\langle b_1, ..., b_k\rangle, \langle a_1, ..., a_k\rangle) \end{array} \right\}.$$

• The *success abstraction* $\alpha_{\mathcal{S}} : \wp(\mathcal{T}_P^a(SLD)) \rightarrow \wp(Atom)$ approximates any finite successful trace with its initial state, while non successful traces are simply ignored: $\alpha_{\mathcal{S}}(X) = \{h \mid h \xrightarrow{\bar{c}}{}^* \Lambda \in X\}$. Notice that it induces a best correct approximation which is equivalent to the s-semantics ([17]) for computed answers, $\mathcal{S} = \langle \wp(Atom)_{\subseteq}, T_P^{\mathcal{S}} \rangle$ (i.e., $T_P^{\mathcal{S}} = \alpha_{\mathcal{S}} \circ \varphi_P \circ \gamma_{\mathcal{S}}$), where for each $I \in \wp(Atom)$:

$$T_P^{\mathcal{S}}(I) = \left\{ h\vartheta \;\middle|\; \begin{array}{l} c = h \leftarrow b_1, ..., b_n \in P \\ \langle b'_1, ..., b'_n\rangle \ll_c I, \ n \geq 0 \\ \vartheta = mgu(\langle b_1, ..., b_n\rangle, \langle b'_1, ..., b'_n\rangle) \end{array} \right\}.$$

• The *Clark's abstraction* $\alpha_{\mathcal{C}} = inst \circ \alpha_{\mathcal{S}}$ maps successful traces to the set of instances of their initial states, while non successful traces are ignored. Notice that it induces a best correct approximation which is equivalent to Clark's semantics for correct answers, $\mathcal{C} = \langle inst(\wp(Atom))_{\subseteq}, T_P^{\mathcal{C}} \rangle$ (i.e., $T_P^{\mathcal{C}} = \alpha_{\mathcal{C}} \circ \varphi_P \circ \gamma_{\mathcal{C}}$), where for each $I \in inst(\wp(Atom))$: $T_P^{\mathcal{C}}(I) = \{h\vartheta \mid c = h \leftarrow b_1, ..., b_n \in P, \vartheta \in Sub, \ b_1\vartheta, ..., b_n\vartheta \in I\}$.

• Finally, the *Herbrand's abstraction* $\alpha_{\mathcal{H}} = ground \circ \alpha_{\mathcal{S}}$ maps successful traces to the set of ground instances for their initial states, while non-successful traces are ignored. The corresponding semantics is the well known Herbrand's semantics ([27]), $\mathcal{H} = \langle \wp(Atom_\emptyset)_{\subseteq}, T_P^H \rangle$ (i.e., $T_P^H = \alpha_{\mathcal{H}} \circ \varphi_P \circ \gamma_{\mathcal{H}}$), where for each $I \in \wp(Atom_\emptyset)$: $T_P^H(I) = \{h \in Atom_\emptyset \mid h \leftarrow \bar{b} \in ground(P), \ \bar{b} \subseteq I\}$.

Let $\mathcal{A}$ and $\mathcal{B}$ be semantics. In the following we abuse by letting $\mathcal{A} \rightarrow^{\nabla} \mathcal{B}$ denote both the semantics which corresponds to the functional composition of $\mathcal{A}$ and $\mathcal{B}$, and its semantic domain. The following result provides a first semantic interpretation of functional autodependencies for the s-semantics $\mathcal{S}$ defined in [17].

Theorem 5.6 *$\mathcal{S} \rightarrow^{\nabla} \mathcal{S}$ is isomorphic to $\langle \wp(Clause), \alpha_s \circ \varphi_P \circ \gamma_s \rangle$.*

The abstraction of a trace interpretation I is an element in $\mathcal{S}\rightarrow^{\nabla}\mathcal{S}$ which encodes the dependency between input successful traces and output successful computations that can be obtained by unfolding traces in I. In particular, the elements in $\mathcal{S}\rightarrow^{\nabla}\mathcal{S}$ (i.e., $\lambda x.\alpha_{\mathcal{S}}(I\nabla\gamma_{\mathcal{S}}(x))$) are actually the best correct approximations for the concrete operators $\lambda x.I\nabla x : \wp(\mathcal{T}_P^a(SLD)) \rightarrow \wp(\mathcal{T}_P^a(SLD))$, for $I \in \wp(\mathcal{T}_P^a(SLD))$. It is well known that $\lambda x.I\nabla x$ is a T_P-like function ([16]). Hence, the isomorphism above is based on the correspondence between T_P-like functions and clauses, the latter encoding the functional dependency between their bodies and heads. It is worth noticing that $\mathcal{S}\rightarrow^{\nabla}\mathcal{S}$ corresponds to the well known compositional semantics of Bossi *et al.* introduced in [6], specialized for left-to-right selection rule and where each predicate in P is considered *open*. Denotations in $\mathcal{S}\rightarrow^{\nabla}\mathcal{S}$ are T_P-like functions. The main difference with the approach of Brogi and Turini [7] is that in [7] the semantics of a module is a T_P-like function, and compositionality is achieved by function composition, while in $\mathcal{S}\rightarrow^{\nabla}\mathcal{S}$ the semantics of a module is the fixpoint iteration of a transformation of T_P-like functions. It is straightforward to prove that $\mathcal{S}\rightarrow^{\nabla}\mathcal{S}$ is compositional in the sense of [6], i.e., if $\mathcal{S}\rightarrow^{\nabla}\mathcal{S} = \langle D, T_P\rangle$ and $\mathcal{F}(P) = \mathit{lfp}(T_P)$ then for any $P_1, P_2 \in \mathit{Program}$, $\mathcal{F}(P_1 \cup P_2) = \mathcal{F}(\mathcal{F}(P_1) \cup \mathcal{F}(P_2))$. Likewise, it is easy to define the semantics $\mathcal{C}\rightarrow^{\nabla}\mathcal{C}$ and $\mathcal{H}\rightarrow^{\nabla}\mathcal{H}$, being all compositional in the sense of [6]. Note that $\mathcal{H}\rightarrow^{\nabla}\mathcal{H}$ is the compositional semantics in [20].

An interesting feature of this approach is that, being based on arbitrary Galois connections, it is independent on specific choices for semantic denotations. In particular, we can compose arbitrary semantics, without being constrained to look for corresponding syntactic objects as denotation, like clauses, binary clauses, atoms, *etc.*, which is the case in the standard s-semantics approach. An interesting application of this idea is $\mathcal{S}\rightarrow^{\nabla}\mathit{Call}$. This semantics models the functional dependencies between successful computations for predicates and goal-independent call patterns. It is easy to see that denotations in $\mathcal{S}\rightarrow^{\nabla}\mathit{Call}$ are functions from $\wp(\mathit{Atom})$ to $\wp(\mathit{BClause})$. In particular, a trace interpretation $I \in \wp(\mathcal{T}_P^a(SLD))$ is abstracted by α_{s-c} into an object in $\mathcal{S}\rightarrow^{\nabla}\mathit{Call}$ such that $\alpha_{s-c}(I) = \lambda Y \in \wp(\mathit{Atom}).T^{\mathit{Call}}_{\alpha_s(I)}(Y)$. A semantic interpretation for $\mathcal{S}\rightarrow^{\nabla}\mathit{Call}$ can be obtained in *modular logic programming* by the following result. We adopt the notation in [8], and say that for a logic program (or module) P, $\mathit{open}(P)$ denotes the set of predicates that occur in the body of a clause in P but are not defined in P. A program Q extends P if Q defines $\mathit{open}(P)$. Let γ_{s-c} be the upper adjoint of α_{s-c}, and $T_P^{s-c} = \alpha_{s-c}\circ\varphi_P\circ\gamma_{s-c}$.

Theorem 5.7 *Let P and Q be program modules such that Q extends P, and let $G = g_1, ..., g_n$ be a goal. If $f_1, ..., f_k \in \mathit{lfp}(T_P^{s-c})$, $1 \le k \le n$ and for $1 \le i \le k-1$, $h_i \ll_G f_i(\mathit{lfp}(T_Q^{\mathcal{S}}))$ and $h_k \leftarrow b_k \ll_G f_k(\mathit{lfp}(T_Q^{\mathcal{S}}))$ are renamed apart, and $\vartheta = mgu(\langle h_1, ..., h_k\rangle, \langle g_1, ..., g_k\rangle)$, then $b_k\vartheta$ is a call pattern for G in $P \cup Q$.*

6 Related works

The most related works are [3, 11, 12, 26]. Cousot and Cousot firstly defined the *reduced cardinal power* of abstract interpretations in [11]. This construction, relied upon the particular semantics of assertions, was based on functional composition with respect to the meet-operation on assertions, which restricted to additive functions is isomorphic to Nielson's *tensor product* of [26]. We generalize those results for arbitrary operators of composition for concrete denotations, allowing the approach to be applicable to semantics of logic programming, where trace-unfolding does not behave like a meet-operation (e.g., it is not commutative). Note, however, that functional dependencies relatively to an operation of composition are included into the relational analysis defined by combination of reduced product and down-set completion ([12]). Recently, Bagnara in [3] has developed an elegant meta-language for domain combination which is based on a subset of a *cc* language, providing asynchronous interaction of domains. Because of this feature, the composition of abstract domains relies upon (lower) closure operators, which provide the standard semantic interpretation of *cc*. The use of *cc*-like expressions to combine domains is therefore equivalent to upgrade a domain with some of its lower closures, and it is in this sense an instance of the general notion of functional dependency, which is however sufficient to give an alternative characterization for *Def*. The key point here is that a *cc* language can be too restrictive to express arbitrary dependency relations between different abstract domains relatively to non-meet like operations of composition (e.g., in semantics design).

Acknowledgments

The stimulating discussions with Roberto Bagnara, Patrick Cousot and Enea Zaffanella are gratefully acknowledged. This work was partly done while the authors were visiting the *LIX, Laboratoire d'Informatique, École Polytechnique*, Paris. They thank *LIX*, and in particular Radhia Cousot, for the kind hospitality.

References

[1] P. Aczel. An introduction to inductive definitions. In *Handbook of Mathematical Logic*, pp. 739–782. North-Holland, 1977.

[2] T. Armstrong, K. Marriott, P. Schachte, and H. Søndergaard. Boolean functions for dependency analysis: algebraic properties and efficient representation. In *Proc. SAS '94*, LNCS 864, pp. 266–280, 1994.

[3] R. Bagnara. Constraint systems for pattern analysis of constraint logic-based languages. In P. Codognet, editor, *Proc. CCP '95 Workshop*, 1995. URL `http://www.di.unipi.it/~bagnara/bagnara.html`.

[4] R. Barbuti, R. Giacobazzi, and G. Levi. A general framework for semantics-based bottom-up abstract interpretation of logic programs. *ACM TOPLAS*, 15(1):133–181, 1993.

[5] G. Birkhoff. *Lattice Theory.* AMS Colloq. Public. vol. XXV, 3rd ed., 1967.

[6] A. Bossi, M. Gabbrielli, G. Levi, and M.C. Meo. A compositional semantics for logic programs. *TCS*, 122(1–2):3–47, 1994.

[7] A. Brogi and F. Turini. Fully abstract compositional semantics for an algebra of logic programs. To appear in *TCS*, 1995.

[8] M. Codish, S. Debray, and R. Giacobazzi. Compositional analysis of modular logic programs. In *Proc. ACM POPL '93*, pp. 451–464, 1993.

[9] M. Comini and G. Levi. An algebraic theory of observables. In *Proc. ILPS '94*, pp. 172–186, 1994.

[10] 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.

[11] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. ACM POPL '79*, pp. 269–282, 1979.

[12] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *J. of Logic Programming*, 13(2,3):103–179, 1992.

[13] P. Cousot and R. Cousot. Inductive definitions, semantics and abstract interpretation. In *Proc. ACM POPL '92*, pp. 83–94, 1992.

[14] P. Cousot and R. Cousot. Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and PER analysis of functional languages). In *Proc. IEEE ICCL '94*, pp. 95–112, 1994.

[15] P. Dart. On derived dependencies and connected databases. *J. of Logic Programming*, 11(2):163–188, 1991.

[16] F. Denis and J.-P. Delahaye. Unfolding, procedural and fixpoint semantics of logic programs. In *Proc. STACS '91*, LNCS 480, pp. 511–522, 1991.

[17] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. Declarative modeling of the operational behavior of logic languages. *TCS*, 69(3):289–318, 1989.

[18] M. Gabbrielli and R. Giacobazzi. Goal independency and call patterns in the analysis of logic programs. In *Proc. ACM SAC '94*, pp. 394–399, 1994.

[19] 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.

[20] H. Gaifman and E. Shapiro. Fully abstract compositional semantics for logic programs. In *Proc. ACM POPL '89*, pp. 134–142, 1989.

[21] R. Giacobazzi. On the collecting semantics of logic programs. In *Proc. Post-ICLP '94 Workshop on Verif. and Analysis of Logic Lang.*, pp. 159–174, 1994. URL `http://www.di.unipi.it/~giaco/giaco.html`.

[22] R. Giacobazzi, C. Palamidessi, and F. Ranzato. Weak relative pseudo-complements of closure operators. Technical Report LIX/RR/95/04, Laboratoire d'Informatique, École Polytechnique, Paris, 1995. URL `http://www.di.unipi.it/~giaco/giaco.html`.

[23] N.D. Jones and H. Søndergaard. A semantics-based framework for the abstract interpretation of PROLOG. In *Abstract Interpretation of Declarative Languages*, pp. 123–142. Ellis Horwood Ltd., 1987.

[24] K. Marriott and H. Søndergaard. Precise and efficient groundness analysis for logic programs. *ACM LOPLAS*, 2(1–4):181–196, 1993.

[25] J. Morgado. Note on complemented closure operators of complete lattices. *Portugal. Math.*, 21(3):135–142, 1962.

[26] F. Nielson. Tensor products generalize the relational data flow analysis method. In *Proc. 4th Hung. Comp. Science Conf.*, pp. 211–225, 1985.

[27] 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.

[28] M. Ward. The closure operators of a lattice. *Ann. of Math.*, 43(2):191–196, 1942.