Complementation in Abstract Interpretation

Agostino Cortesi* Gilberto Filé** Roberto Giacobazzi***

Catuscia Palamidessi† Francesco Ranzato‡

Abstract. The reduced product of abstract domains is a rather well known operation in abstract interpretation. In this paper we study the inverse operation, which we call complementation. Such an operation allows to systematically decompose domains; it provides a systematic way to design new abstract domains; it allows to simplify domain verification problems, like correctness proofs; and it yields space saving representations for domains. We show that the complement exists in most cases, and we apply complementation to two well known abstract domains, notably to the Cousot and Cousot's comportment domain for analysis of functional languages and to the complex domain *Sharing* for aliasing analysis of logic languages.

1 Introduction

Compositionality is a fundamental feature of the standard Cousot and Cousot theory of abstract interpretation ([8, 9]). Compositionality can be both related with the underlying semantics, yielding the compositional design of program analyses inductively on the program's structure, and with data-approximation, providing some basic operators to compose abstractions. The latter case, being independent from any specific programming language and semantics, is a key feature of abstract interpretation. In this case, abstract domains for analysis can be incrementally designed by successive abstractions, and more precise approximations can be obtained by composing domains or by lifting them by suitable property-completions (e.g. the disjunctive completions in [12, 13, 16]).

The *reduced product* ([9]) is probably the most common and well known operation that composes abstract domains. It provides a systematic way to achieve more descriptive abstract domains from simpler ones, and it corresponds to the cardinal, *attribute independent* product of domains. The reduction is essential in this process to achieve a minimal representation for abstract denotations. In this case, tuples of abstract objects are considered equivalent (and therefore reduced) provided

* Dipartimento di Matematica Applicata e Informatica, Università di Venezia, Via Torino 155, 30173 Mestre-Venezia, Italy, cortesi@moo.dsi.unive.it

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

*** LIX, Laboratoire d'Informatique, École Polytechnique, 91128 Palaiseau cedex, France, giaco@lix.polytechnique.fr

† Dipartimento di Informatica e Scienze dell'Informazione, Università di Genova, Viale Benedetto XV 3, 16132 Genova, Italy, catuscia@disi.unige.it

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

that their conjunction represents the same concrete denotation. The cost/precision tradeoff between separate abstract interpretations and their reduced product has been experimentally evaluated in [3] for analysis of logic programs. This operation is also essential in *attribute-dependent* or *relational* analysis, where it can be combined with disjunctive completions (e.g. those described in [12, 13]) to generalize the relational *tensor product* ([27]).

A natural question that arises in this setting is whether it is possible to define the *inverse operation for reduced product*, namely, an operation which, starting from any two domains C and D, with D more abstract than C, gives as result the most abstract domain $C \sim D$, whose reduced product with D is exactly C. The closure operator approach to abstract interpretation provides a very useful mathematical framework for studying this question. Indeed, every abstract domain that enjoys a Galois insertion with the concrete domain C, can be associated with an upper closure operator on C, and the operation of reduced product between abstract domains can be interpreted as the corresponding lattice-theoretic operation of *greatest lower bound* on the complete lattice $uco(C)$ of upper closure operators on C ([9]). Thus, technically, the question above is equivalent to ask whether $uco(C)$ is *pseudo-complemented.* A recent result in [19] shows that the lattice of closure operators of a complete lattice C is pseudo-complemented whenever C is *chain inf-distributive*, namely when the *glb* of C is a continuous binary operation.

It is also important to remark that the above definition of complementation cannot be strengthened by requiring also that $C \sim D$ be disjoint from D. In fact, this stronger definition would be equivalent to the requirement that $uco(C)$ is *complemented*, and Dwinger ([15]) and, successively, Morgado ([23]) proved that $uco(C)$ is complemented if and only if C is a complete well-ordered chain. This condition is clearly too restrictive to be applied in static analysis and abstract interpretation of programming languages, because concrete and abstract domains for semantics and analysis are not in general complete chains.

In this paper, we observe that the condition of chain inf-distributivity is satisfied by most of the known concrete and abstract domains for semantics and analysis. Thus, exploiting the above mentioned result of [19], we introduce the notion of *complementation in abstract interpretation*, calling abusively complement of an abstract domain the pseudo-complement of its associated closure operator.

As the theory developed in Section 4 shows, complementation allows to systematically decompose abstract domains into simpler factors. This is important for four main reasons: (i) it provides a systematic way to design new abstract domains; (ii) it helps understanding the internal workings of complex domains; (iii) it supports verification problems for complex domains, like correctness proofs, by allowing decomposition into simpler problems for their factors; (iv) it provides compact representations of complex domains that enhances space saving techniques.

As examples of use of complementation for domain decomposition, we consider a domain for aliasing analysis in logic programming, notably *Sharing* introduced by Jacobs and Langen in [21], and the Cousot and Cousot's *comportment* analysis for higher-order functional languages ([13]), which generalizes Mycroft's *strictness* and *termination* analysis ([25, 26]), Hughes and Wadler's *projection* analysis ([30]) and Hunt's *PER* analysis ([20]). In the latter case we apply complementation to factorize the domain of comportment analysis which is originally obtained by disjunctive completion of a simpler domain, providing a sensible reduction of the lattice-structure of comportments. In the case of *Sharing*, we use complementation to characterize what is left when we eliminate from it the information useful for ground-dependency analysis. In [5] it is proved that the information for ground-dependency analysis of *Sharing* is expressed by a more abstract domain, which we show to coincide with the domain *Def*. *Def* was introduced by Dart in [14] for groundness analysis in deductive databases, and used by Marriott and Søndergaard for ground-dependency analysis in [22]. As expected, the complement of *Def* with respect to *Sharing*, called *Sharing*$^+$, captures precisely variable aliasing and no ground-dependency information, and shares with *Sharing* an elegant representation.

2 Preliminaries

Throughout the paper we will assume familiarity with the basic notions of lattice theory (e.g. see [2]) and abstract interpretation ([8, 9]). Now, we briefly introduce some notation and recall some well known notions.

If C and D are posets and $\alpha : C \to D$, $\gamma : D \to C$ are monotonic functions such that $\forall c \in C.\ c \preceq_C \gamma(\alpha(c))$ and $\forall d \in D.\ \alpha(\gamma(d)) \preceq_D d$, then we call the quadruple (γ, D, C, α) a *Galois connection* (G.c.) between C and D. If in addition $\forall d \in D.\ \alpha(\gamma(d)) = d$, then we call (γ, D, C, α) a *Galois insertion* (G.i.) of D in C. In the setting of abstract interpretation, C and D 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, D is called an *abstraction* (or *abstract interpretation*) of C, and C a *concretization* of D. Further, D is a *strict* abstraction of C if $\gamma \circ \alpha \neq \lambda x.x$ (or, equivalently, if D is an abstraction of C, while C is not an abstraction of D). In the following, $C \trianglelefteq D$ denotes that D is an abstraction of C. If (γ, D, C, α) 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.

Let $\langle L, \preceq, \wedge, \vee, \top, \bot \rangle$ be a complete lattice. In the rest of the paper, we will weaken or strengthen the hypotheses on the structure of L according to the case. An *(upper) closure operator* on the poset L is an operator $\rho : L \to L$ monotonic, idempotent and extensive (viz. $\forall x \in L.\ x \preceq \rho(x)$). If L is a complete lattice then each closure operator ρ is uniquely determined by the set of its fixpoints, which is its image $\rho(L)$ ([31]). 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 $Y \subseteq X$, $\wedge Y \in X$). Furthermore, the set of fixpoints $\rho(L)$ is a complete lattice with respect to the order of L, but, in general, it is not a complete sub-lattice of L, since the *lub* in $\rho(L)$ might be different from that in L. In the following, we will often denote a closure operator by the set of its fixpoints. We denote by $\langle uco(L), \sqsubseteq, \sqcap, \sqcup, \lambda x.\top, \lambda x.x \rangle$ the complete lattice of all closure operators on the complete lattice L, where for every $\rho, \eta \in uco(L)$, $\{\rho_i\}_{i \in I} \subseteq uco(L)$ and $x \in L$: (i) $\rho \sqsubseteq \eta$ iff $\forall x \in L.\ \rho(x) \preceq \eta(x)$, or equivalently ([31]), $\rho \sqsubseteq \eta$ iff $\eta(L) \subseteq \rho(L)$; (ii) $(\sqcap_{i \in I} \rho_i)(x) = \wedge_{i \in I} \rho_i(x)$; (iii) $(\sqcup_{i \in I} \rho_i)(x) = x \Leftrightarrow \forall i \in I.\ \rho_i(x) = x$; (iv) $\lambda x.\top$ is the top element, whereas $\lambda x.x$ is the bottom element.

2.1 Abstract Interpretation and Closure Operators

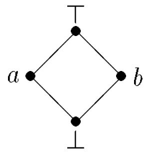
A key point in Cousot and Cousot abstract interpretation theory ([9]) is the equivalence between the Galois insertion and closure operator approach to the design of abstract domains. Usually, the Galois insertion approach is the most used. In this case, D is an abstraction of C if there exist α and γ such that (γ, D, C, α) is a Galois insertion. It is well known since [9] that the real essence of an abstract domain lies with the closure operator associated with the corresponding G.i.. Actually, an abstract domain is just a "computer representation" of its logical meaning, namely its image in the concrete domain. In fact, using a different but lattice-theoretic isomorphic domain changes nothing in the abstract reasoning. This logical meaning of an abstract domain is exactly captured by the associated closure operator on the concrete domain. More formally, on one hand, if (γ, D, C, α) 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 $\iota : \rho(C) \rightarrow D$ is an isomorphism of complete lattices (with inverse ι^{-1}) then $(\iota^{-1}, D, C, \iota \circ \rho)$ is a G.i.. 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 (or C a concretization of D), viz. $C \trianglelefteq D$, we will mean that $D \cong \rho_D(C)$ for some closure $\rho_D \in uco(C)$. It is well known ([9]) that the order relation on $uco(C)$ corresponds to the order by means of which abstract domains are compared with regard to their precision of representation. 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. $\rho_2(C) \subseteq \rho_1(C)$). Since, clearly, D_1 is more precise than D_2 iff there exists $\rho \in uco(D_1)$ such that $D_2 \cong \rho(D_1)$, then we can equivalently write $D_1 \trianglelefteq D_2$. Therefore, to compare domains with regard to their precision, we will only speak about abstractions between them, and use $\trianglelefteq$ to relate both non-homogeneous domains, i.e. domains which are not Moore-families of a more concrete domain, and homogeneous domains, i.e. closure operators on a concrete domain. Further, we will often use the equality symbol $=$ between domains instead of the more rigorous symbol of isomorphism $\cong$.

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 $i \in I$. Any domain D isomorphic to the *lub* $(\sqcup_{i\in I}\rho_i)(C)$ is the most concrete among the domains which are abstractions of all the D_i's. The interpretation of the *glb* operation on $uco(C)$ is twofold. Firstly, any domain D isomorphic to the *glb* $(\sqcap_{i\in I}\rho_i)(C)$ is (isomorphic to) the well known *reduced product* ([9]) of all the domains D_i. Further, the *glb* D, and hence the reduced product, is the most abstract among the domains (abstracting C) which are more concrete than every D_i. Thus, we will denote the reduced product of abstract domains by the *glb* symbol $\sqcap$.

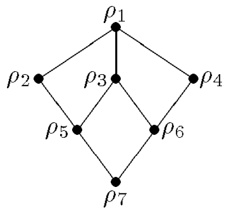
3 (Pseudo-)Complements of Abstract Interpretations

A consequence of the isomorphism between the lattice of abstract interpretations of a concrete domain C and the corresponding lattice of closure operators $uco(C)$, is that it is not in general possible to define the lattice-theoretic complement of abstract interpretations. This follows from results in [15, 23] saying that $uco(C)$ is complemented (or, equivalently, distributive or a Boolean algebra) iff C is a complete well-ordered chain. This condition is clearly too restrictive for abstract interpretation of programming languages. The following example shows this problem in a simple finite lattice.

Example 1. Consider the following finite lattice C.



The closure operators (or equivalently abstract interpretations) on C are the following: $\rho_1 = \{\top\}$, $\rho_2 = \{\top, a\}$, $\rho_3 = \{\top, \bot\}$, $\rho_4 = \{\top, b\}$, $\rho_5 = \{\top, a, \bot\}$, $\rho_6 = \{\top, b, \bot\}$, $\rho_7 = \{\top, a, b, \bot\}$.
They form the lattice $uco(C)$ depicted below.



It is immediate to observe that $uco(C)$ is not complemented. For instance, ρ_3 does not have the complement in $uco(C)$. $\square$

The idea of this paper is to use a different and weaker notion of complementation in $uco(C)$ as a systematic approach for complementation of abstract interpretations. Indeed, while $uco(C)$ is not in general complemented, it is in practice always *pseudo-complemented*, as proved in [19].

Let L be a meet semi-lattice with bottom element $\bot$. The *pseudo-complement* of $x \in L$, if it exists, is the (unique) element $x^* \in L$ such that $x \wedge x^* = \bot$ and $\forall y \in L.\, (x \wedge y = \bot) \Rightarrow (y \preceq x^*)$. If every $x \in L$ has the pseudo-complement, we say that L is *pseudo-complemented*. In a complete lattice L, if the pseudo-complement of $x \in L$ exists then it is always defined as $x^* = \vee\{y \in L \mid x \wedge y = \bot\}$.

Example 2. Consider the lattice of closure operators of Example 1. It is easy to verify the following pseudo-complements for the elements in $uco(C)$: $\rho_1^* = \rho_7$, $\rho_2^* = \rho_4$, $\rho_3^* = \rho_7$, $\rho_4^* = \rho_2$, $\rho_5^* = \rho_4$, $\rho_6^* = \rho_2$ and $\rho_7^* = \rho_1$. $\square$

The following definition and results are recalled from the recent paper [19]. In what follows, we assume that L is a complete lattice.

Definition 3. L is *chain inf-distributive* if for any chain $C \subseteq L$ and for each $x \in L$, $x \wedge (\vee C) = \vee_{y \in C}(x \wedge y)$. $\square$

Remark. It is worth noting (as pointed out in [19]) that the chain inf-distributivity property is strictly weaker than the well known complete inf-distributivity property (viz. $\forall x \in L. \forall Y \subseteq L.\ x \wedge (\vee Y) = \vee_{y \in Y}(x \wedge y)$). For instance, any lattice satisfying the ascending chain condition is obviously chain inf-distributive, but not necessarily complete inf-distributive (which for a finite lattice amounts to be distributive). $\square$

This notion of chain inf-distributivity is central in the following result.

Theorem 4. *If L is chain inf-distributive then $uco(L)$ is pseudo-complemented.*

By the above remark, the following corollary is immediate.

Corollary 5. *If L satisfies the ascending chain condition then $uco(L)$ is pseudo-complemented.*

Therefore, it is possible to define a weaker notion of complementation for abstract interpretation, which is precisely pseudo-complementation. In this case, the abstract domain to factorize plays the rôle of L.

Suppose D is an abstraction of the complete lattice C, and assume that C is chain inf-distributive. Exploiting Theorem 4 we can give the following definition of *complement* of abstract interpretations[1].

Definition 6. The *complement* of D with respect to C is the complete lattice $C \sim D$ given by the set of fixpoints of the pseudo-complement ρ_D^* (in $uco(C)$) of ρ_D. $\square$

It is immediate to observe that if $C \trianglelefteq D$ then $C \trianglelefteq C \sim D$. Therefore, the complement $C \sim D$ is an abstraction of C. As observed previously, the pseudo-complement of ρ_D is expressible as $\rho_D^* = \sqcup\{\rho \in uco(C) \mid (\rho_D \sqcap \rho)(C) = C\}$. This equality makes clear the meaning of the complement: $C \sim D$ is the most abstract among the domains (abstracting C and) such that their reduced product with D yields C. Moreover, C may be thought of either as a concrete domain or as an abstract domain. In both cases, the complement $C \sim D$ intuitively captures what program properties representable by the domain C are ignored and left out by its abstraction D, allowing to understand more in depth how the abstraction process led from C to D. By Definition 6, we fix in C the representation of the complement $C \sim D$, being $C \sim D = \rho_D^*(C)$, and hence a subset of C. Obviously, any other lattice isomorphic to $C \sim D$ can be considered in all respects as the complement. From now on, whenever we will speak about complements we will suppose that the conditions for their existence hold.

The following result, which is recalled from [7, Theorem 4.2.0.4.7], provides a simple way to generalize the pseudo-complement to arbitrary abstractions in the lattice of abstract interpretations.

Proposition 7. *Let L be a complete lattice, and $\eta \in uco(L)$. Then, $uco(\eta(L)) \cong {\uparrow}\eta = \{\rho \in uco(L) \mid \eta \sqsubseteq \rho\}$.*

[1] We abuse terminology by (re)naming complement the notion of pseudo-complement.

Therefore, we can apply Theorem 4 to arbitrary pairs of elements in the lattice of abstract interpretations, namely we can always complement a domain with respect to any more concrete chain inf-distributive domain in the lattice of abstract interpretations. In particular, assume C be a concrete domain, D an abstraction of C and E an abstraction of D, with corresponding closures ρ_D and ρ_E, respectively. In this case, while the computation of $D \sim E$ requires that D is chain inf-distributive, the domain C can be merely a complete lattice. The complement $D \sim E$ is given by the set of fixpoints of the pseudo-complement $\rho_E^* = \sqcup\{\rho \in uco(D) \mid (\rho_E \sqcap \rho)(D) = D\}$, a closure operator on D. Proposition 7 says that $uco(D)$ and $\{\rho \in uco(C) \mid \rho_D \sqsubseteq \rho\}$ are isomorphic complete lattices. Hence, the complement $D \sim E$ can be obtained as a pseudo-complement on the domain C, namely $\rho_E^* \circ \rho_D = \sqcup\{\rho \in uco(C) \mid (\rho_E \circ \rho_D) \sqcap \rho = \rho_D\}$, which corresponds precisely to the expected intuitive meaning of $D \sim E$.

The following algebraic properties of the complement operation $\sim$ on abstract interpretations can be easily derived from similar properties of pseudo-complemented lattices (see [2, 17, 29]).

Proposition 8. *Let C be a chain inf-distributive lattice, $C \trianglelefteq D, E$, and $\top$ the most abstract interpretation of C. Then,*

(a) $D \trianglelefteq C \sim (C \sim D)$;
(b) $(D \trianglelefteq E) \Rightarrow (C \sim E) \trianglelefteq (C \sim D)$;
(c) $(C \sim D) = C \sim (C \sim (C \sim D))$;
(d) $(C \sim D = \top) \Leftrightarrow (D = C)$;
(e) $C \sim \top = C$ *and* $C \sim C = \top$.

There exists a wide class of abstract domains for which we can always compute the complement. Indeed, the overwhelming majority of the abstract domains used as basis of a static analysis satisfies the ascending chain condition (even, most of them are finite domains). Furthermore, even if the abstract domain does not satisfy the ascending chain condition (this happens whenever some widening/narrowing operators used to accelerate the convergence above least fixpoints are provided), the chain inf-distributivity property can be checked for it. To this aim, it is worth noting that any powerset ordered with the subset (or supset) relation is complete inf-distributive, and hence chain inf-distributive. Moreover, as a remarkable example, the abstract lattice of intervals of integer numbers introduced in [8] to analyze the values of an integer variable does not satisfy the ascending chain condition and is not distributive, but it is chain inf-distributive.

4 Complements to Decompose Abstract Domains

Complements of abstract interpretations can be used to design new abstract domains for analysis. The following two sections apply this idea, in particular exploiting factorizations of abstract domains. Very often, abstract domains for analysis are incrementally designed using the reduced product operation of simpler domains (e.g., in logic programs analysis see [3, 24, 28]). This introduces modularity in domain design, which is helpful both to design domain dependent abstract operations and to simplify proofs of correctness for complex domains of analysis.

Definition 9. A *decomposition* for a domain D is a tuple $\langle D_i \rangle_{i \in I}$ such that $D = \sqcap_{i \in I} D_i$. $\square$

Obviously, if $\langle D_i \rangle_{i \in I}$ is a decomposition of D then each D_i is an abstraction of D. Moreover, notice that a trivial decomposition always exists. In fact, for any domain D the pair $\langle D, \top \rangle$ (and $\langle \top, D \rangle$) is evidently a decomposition of D.

Clearly, the complement operation provides a systematic way to *factorize* a given domain into binary decompositions. This may be helpful to decompose domains that are not originally designed by incremental products of more abstract domains. The case of the domain *Sharing* for aliasing analysis of logic programs ([21]) is a typical case of a complex abstract domain for which no decomposition is known in the literature. This case will be discussed in Section 6. The advantage of abstract domain decompositions is therefore evident: instead of proving properties for general domains, e.g. correctness for abstract operations, we can prove properties for more abstract (and simple) factors, provided that these properties are preserved under composition, which is in our case the reduced product of abstract interpretations.

As mentioned above, an example of such properties is correctness for abstract operations. Let (γ, D, C, α) be a Galois insertion. An abstract operation $op^a : D \longrightarrow D$ is a *correct approximation* of $op : C \longrightarrow C$ iff $\alpha \circ op \circ \gamma \preceq_D op^a$ ([8]).

Proposition 10. *Let C be a domain, $op : C \longrightarrow C$ be a monotonic operation, and $C \trianglelefteq D$. If $\langle D_I \rangle_{i \in I}$ is a decomposition of D with Galois insertions $(\gamma_i, D_i, D, \alpha_i)$ $(i \in I)$, and $op_i : D_i \longrightarrow D_i$ $(i \in I)$ are corresponding correct approximations of op, then $\lambda x. \wedge_{i \in I} \gamma_i(op_i(\alpha_i(x)))$ is a correct approximation of op.*

More in general, we can characterize a class of properties of abstract domains which is compositional with respect to the reduced product.

Definition 11. Let C be a domain and $uco(C)$ the corresponding lattice of abstract interpretations of C. A *conjunctive property of abstractions* is a set of abstractions of C, $\pi \subseteq uco(C)$, such that if $\{D_i\}_{i \in I} \subseteq \pi$ then $\sqcap_{i \in I} D_i \in \pi$. $\square$

Clearly, any closure on $uco(C)$, i.e. an element of $uco(uco(C))$, identifies a conjunctive property of abstractions of C, being a Moore-family of abstract domains. It is therefore immediate to prove the following result.

Proposition 12. *Let $C \trianglelefteq D$, and π be a conjunctive property on C. If $D \in \pi$ and $C \sim D \in \pi$ then $C \in \pi$.*

Example 13. Cousot and Cousot introduced in [9] the notion of *disjunctive abstract domain*. An abstract domain $D \in uco(C)$ is *disjunctive* iff D is an additive closure. Let $\pi^\vee \subseteq uco(C)$ be the set of disjunctive abstractions of C. It is easy to verify that $\pi^\vee$ is actually a Moore-family when C is a completely distributive lattice. Thus, by Proposition 12, $\pi^\vee$ can be verified compositionally on abstract domain decompositions. A simple example is in Example 16. $\square$

Analogously, domain decomposition may help to minimize space-complexity in presence of complex domains. This may provide more compact representations for abstract domains as tuples of factors.

Definition 14. Let $\langle D_i \rangle_{i \in I}$ and $\langle D_j \rangle_{j \in J}$ be decompositions of D. $\langle D_i \rangle_{i \in I}$ is *better* than $\langle D_j \rangle_{j \in J}$ if $\sum_{i \in I} |D_i| \leq \sum_{j \in J} |D_j|$. A decomposition for a domain is *minimal* if it has no better decompositions. $\square$

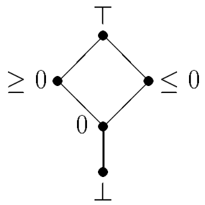
Complementation can also be used to optimize reduced product of abstract domains. Let us consider a decomposition $\langle D_i \rangle_{i \in I}$ for a domain D. The complement $(\sqcap_{i \in I} D_i) \sim D_k$ $(k \in I)$ of D_k with respect to the product $\sqcap_{i \in I} D_i$ is the most abstract domain such that the product with D_k is (isomorphic to) $\sqcap_{i \in I} D_i$. Thus, fixing $I = \{1, 2\}$ and $k = 1$, $(D_1 \sqcap D_2) \sim D_1$ is an abstraction of D_2, and $(D_1 \sqcap D_2) \sim D_1$ may be used to substitute D_2 in the decomposition of D. Indeed, in this case, $\langle D_1, (D_1 \sqcap D_2) \sim D_1 \rangle$ (and $\langle (D_1 \sqcap D_2) \sim D_2, D_2 \rangle$) is better than $\langle D_1, D_2 \rangle$. This decomposition is also the least decomposition among the decompositions having D_1 (D_2) as factor. A further improvement can be obtained by introducing the following equivalence relation on abstract domains.

Definition 15. Let D be a domain, and $D \trianglelefteq D_1, D_2$. Then, $D_1 \approx_D D_2$ iff $D \sim D_1 = D \sim D_2$. $\square$

Clearly, $\approx_D$ is an equivalence relation on $uco(D)$. Also, it is immediate to prove that $D_1 \approx_D D_2$ iff $D \sim (D \sim D_1) = D \sim (D \sim D_2)$. Since $\lambda x.D \sim (D \sim x)$ is a closure operator on $uco(D)$ (by *(a)*, *(b)* and *(c)* in Proposition 8), then $\approx_D$ is a join-complete congruence relation on $uco(D)$ ([9]). Thus, $\sqcup [D_1]_{\approx_D}$ is the most abstract domain which factorizes D as D_1 does. In particular, $(\sqcup [D_1]_{\approx_D}) \sqcap (D \sim D_1) = D$. Hence, a further improvement of the decomposition $\langle D_1, D \sim D_1 \rangle$ can be obtained by considering $\langle \sqcup [D_1]_{\approx_D}, D \sim D_1 \rangle$.

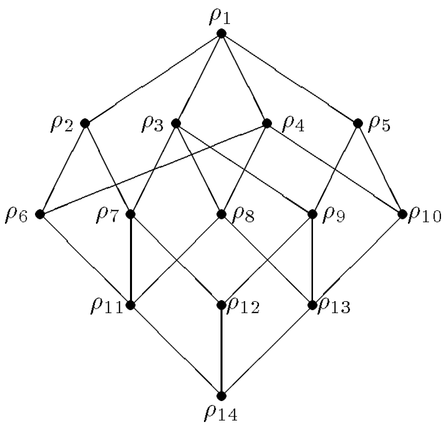
This approach provides a way to avoid possible redundancies due to elements which can be recostructed by composing more abstract factors, and may help to improve the space-complexity of the product. Furthermore, this is clearly the best we can do in this direction. Of course, a decomposition can be decomposed further by complementing one of the factors. Examples of this use of the complement will be given later on. We now give a simple example to illustrate how the notion of complement actually works.

Example 16. Let us consider the typical example of the rule of signs ([8, 9]). The concrete domain is $\wp(\mathbb{Z})$ (ordered with inclusion $\subseteq$), whereas the abstract domain D is depicted below. The concretization and abstraction maps are the most natural.

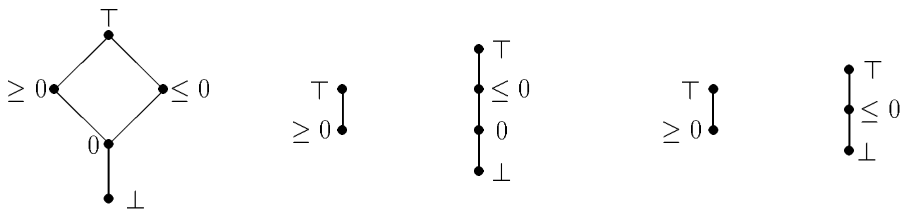


It is easy to verify that all the possible abstractions of this domain, i.e. all the closures on D, are the following: $\rho_1 = \{\top\}$, $\rho_2 = \{\top, \geq 0\}$, $\rho_3 = \{\top, 0\}$, $\rho_4 = \{\top, \bot\}$, $\rho_5 = \{\top, \leq 0\}$, $\rho_6 = \{\top, \geq 0, \bot\}$, $\rho_7 = \{\top, \geq 0, 0\}$, $\rho_8 = \{\top, 0, \bot\}$, $\rho_9 = \{\top, \leq 0, 0\}$, $\rho_{10} = \{\top, \leq 0, \bot\}$, $\rho_{11} = \{\top, \geq 0, 0, \bot\}$, $\rho_{12} = \{\top, \leq 0, \geq 0, 0\}$, $\rho_{13} = \{\top, \leq 0, 0, \bot\}$, $\rho_{14} = D$.

Since D is a finite lattice, by Corollary 5, $uco(D)$ is a pseudo-complemented lattice. In fact, $uco(D)$ is the lattice depicted below, and it is simple to verify the pseudo-complementation of $uco(D)$ straight from its Hasse diagram.



Indeed, the pseudo-complements, i.e. the complements with respect to D of the abstractions, are as follows: $\rho_1^* = \rho_{14}$, $\rho_2^* = \rho_{10}$, $\rho_3^* = \rho_{14}$, $\rho_4^* = \rho_{12}$, $\rho_5^* = \rho_6$, $\rho_6^* = \rho_5$, $\rho_7^* = \rho_{10}$, $\rho_8^* = \rho_{12}$, $\rho_9^* = \rho_6$, $\rho_{10}^* = \rho_2$, $\rho_{11}^* = \rho_5$, $\rho_{12}^* = \rho_4$, $\rho_{13}^* = \rho_2$, $\rho_{14}^* = \rho_1$. Suppose D has been incrementally designed by reduced product of the domains D_1 and D_2 corresponding to the closures ρ_2 and ρ_{13}, respectively. The complement $(D_1 \sqcap D_2) \sim D_1$ is just the domain corresponding to the closure $\rho_2^* = \rho_{10}$. In this case, $(D_1 \sqcap D_2) \sim D_1$ is a strict abstraction of D_2. Therefore, we can safely substitute D_2 by $(D_1 \sqcap D_2) \sim D_1$, getting a more concise and less expensive representation of the product (see the figure below).



$D \quad = \quad D_1 \quad \sqcap \quad D_2 \quad = \quad D_1 \quad \sqcap \quad (D_1 \sqcap D_2) \sim D_1$

Moreover, both D_1 and $(D_1 \sqcap D_2) \sim D_1$ (or D_2) are disjunctive, and $\wp(\mathbb{Z})$ is clearly completely distributive. Hence, by Example 13, D is disjunctive as well. $\square$

5 Functional Programming: Complementing Comportment Analysis

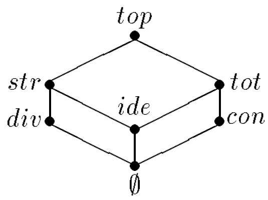
In this application we consider complements with respect to the lattice of *comportment* analysis designed by Cousot and Cousot in [13] to generalize Mycroft's *strictness* and *termination* analysis ([25, 26]), Hughes and Wadler's *projection* analysis ([30]), and Hunt's *PER* analysis ([20]). This provides a decomposition of the

truth	$\gamma^{\beta\to\beta}(top)$	$= D^{\beta\to\beta}$
strictness	$\gamma^{\beta\to\beta}(str)$	$= \{f \mid f(\bot) = \bot\}$
totality	$\gamma^{\beta\to\beta}(tot)$	$= \{f \mid \forall x \in D^{\beta} \setminus \{\bot\}.\ f(x) \neq \bot\}$
identity	$\gamma^{\beta\to\beta}(ide)$	$= \{f \mid \forall x \in D^{\beta}.\ f(x) = \bot \Leftrightarrow x = \bot\}$
divergence	$\gamma^{\beta\to\beta}(div)$	$= \{f \mid \forall x \in D^{\beta}.\ f(x) = \bot\}$
convergence	$\gamma^{\beta\to\beta}(con)$	$= \{f \mid \forall x \in D^{\beta}.\ f(x) \neq \bot\}$
falsity	$\gamma^{\beta\to\beta}(\emptyset)$	$= \emptyset$

Table 1: Basic comportment analysis $\mathcal{B}_{\mathcal{C}}$.

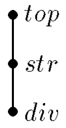
comportment lattice into sensible factors, whose conjunction corresponds precisely to the original domain.

The comportment analysis applies to higher order monomorphically typed lazy functional programming languages. To illustrate the Cousot and Cousot's comportment analysis, we consider abstract interpretation of a simply typed lambda calculus with basic types β. Denote D^{τ} the domain of values of a type τ, and by $\bot$ its bottom element. For simplicity we consider abstraction of function basic types $\beta \to \beta$ (i.e., elements in $D^{\beta\to\beta} = D^{\beta} \to D^{\beta}$, the lattice of total monotonic functions from D^{β} to D^{β} ordered pointwise). The following abstract domain $\mathcal{B}_{\mathcal{C}}$ represents the lattice of *basic comportment analysis*, ordered with respect to the approximation order, for function types $\beta \to \beta$.

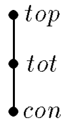


Basic comportment $\mathcal{B}_{\mathcal{C}}$

The meaning of basic comportments in $\mathcal{B}_{\mathcal{C}}$ is given in Table 1, in terms of a concretization function $\gamma^{\beta\to\beta}$ mapping basic comportments into $\wp(D^{\beta\to\beta})$, the concrete domain of the standard collecting semantics.
It is easy to verify that both the standard Mycroft's strictness $\mathcal{S}$ and termination $\mathcal{T}$ analyses are actually abstract interpretations of $\mathcal{B}_{\mathcal{C}}$, yielding the following simpler domains, respectively.



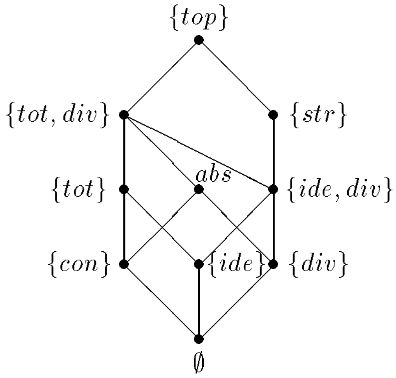
Strictness $\mathcal{S}$



Termination $\mathcal{T}$

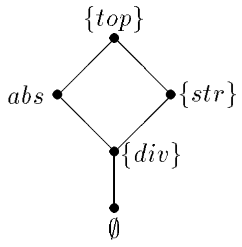
It is easy to see that $\mathcal{B}_{\mathcal{C}} \sim \mathcal{S} = \mathcal{T}$ and $\mathcal{B}_{\mathcal{C}} \sim \mathcal{T} = \mathcal{S}$, namely the complement of the strictness (termination) analysis with respect to the basic comportment is the termination (strictness) analysis. In particular, the identity information *ide* as well as $\emptyset$ can be constructed by conjunction of strictness and termination values (*str* and *tot* for *ide*, *div* and *con* for $\emptyset$). Therefore, the basic comportment lattice $\mathcal{B_C}$ is precisely the reduced product of strictness and termination, being strictness and termination the most abstract factors of comportment analysis. As we will show later on, the identity information will be always definable as conjunction of factors involving strictness information (i.e., in factorizations by complements of analysis involving strictness, like strictness or projection analysis), even though the comportment lattice will be lifted at a powerset level.

As proved by Cousot and Cousot in [13], more precise comportment properties for higher-order functional languages can be characterized by abstraction of a collecting semantics. The abstraction of sets of functions in $D^{\beta \to \beta}$ yields a corresponding abstract domain for comportments which can be systematically derived by a powerset completion on the basic comportment lattice $\mathcal{B_C}$. In this case, the meaning of sets Ψ of basic comportments is given by a concretization function $\gamma^{\wp}$ such that $\gamma^{\wp}(\Psi) = \cup\{\gamma^{\beta \to \beta}(\psi) \mid \psi \in \Psi\}$. The following lattice $\mathcal{C}$, ordered by the approximation order, corresponds precisely to this (extended) comportment analysis. It is obtained by (e.g. anti-chain) powerset completion and reduction (viz. sets of basic comportments denoting the same object in $\wp(D^{\beta \to \beta})$ are identified ([13])). The new element *abs* corresponds here to the set of basic comportments $\{con, div\}$ and represents *absence*.

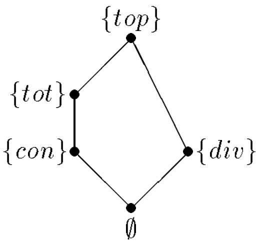


Comportment $\mathcal{C}$

As shown in [13], this lattice generalizes: projection $\mathcal{P}$ and dual-projection $\mathcal{D_P}$ depicted respectively below, as well as the above strictness $\mathcal{S}$ and termination $\mathcal{T}$ analyses (in the latter case, the concretization of an element x is the singleton $\{x\}$).



Projection $\mathcal{P}$



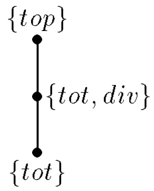
Dual-projection $\mathcal{D_P}$

Some interesting properties of comportment analysis can be studied by looking at the complements of projection, dual-projection, strictness and termination with respect to $\mathcal{C}$. In particular, we observe that new abstract domains can be systematically derived as factors of the comportment lattice.

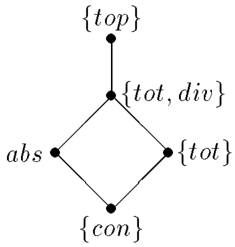
Note that $\mathcal{C} \sim \mathcal{P}$ characterizes possible divergence in total functions. This domain factorizes $\mathcal{C}$, where in particular the (disjunctive) identity information, i.e. $\{ide\}$ and $\{ide, div\}$, as well as the convergence $\{con\}$ can be reconstructed by composing $\mathcal{C} \sim \mathcal{P}$ with projection $\mathcal{P}$. The identity and convergence informations are therefore redundant for the factors $\langle \mathcal{P}, (\mathcal{C} \sim \mathcal{P}) \rangle$.

A domain for totality analysis can be obtained by factorizing comportment with respect to strictness, i.e. considering the domain $\mathcal{C} \sim \mathcal{S}$. This domain characterizes precisely the non-strictness comportments. It is worth noticing that also in this case, the identity information as well as $\emptyset$ can be reconstructed by composing $\mathcal{C} \sim \mathcal{S}$ with strictness, and it is therefore redundant for the factors $\langle \mathcal{S}, (\mathcal{C} \sim \mathcal{S}) \rangle$.

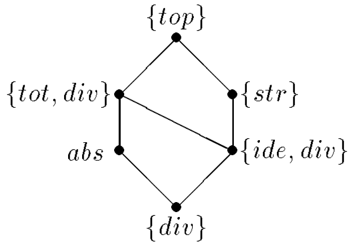
We finally observe that $\mathcal{C} \sim \mathcal{T} = \mathcal{C} \sim \mathcal{D}_{\mathcal{P}}$. This domain characterizes exactly the non-terminating (or divergent) comportments.



$\mathcal{C} \sim \mathcal{P}$



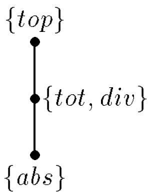
$\mathcal{C} \sim \mathcal{S}$



$\mathcal{C} \sim \mathcal{T} = \mathcal{C} \sim \mathcal{D}_{\mathcal{P}}$

Note that both $\langle \mathcal{P}, (\mathcal{C} \sim \mathcal{P}) \rangle$ and $\langle \mathcal{S}, (\mathcal{C} \sim \mathcal{S}) \rangle$ provide minimal binary factorizations of the comportment lattice, which are actually better than $\langle \mathcal{T}, (\mathcal{C} \sim \mathcal{T}) \rangle$.

A domain for non-terminating & non-strictness comportments can be further obtained as the complement of strictness with respect to $\mathcal{C} \sim \mathcal{T}$, or equivalently as the complement of termination with respect to $\mathcal{C} \sim \mathcal{S}$. Notice that $(\mathcal{C} \sim \mathcal{S}) \sim \mathcal{T} = (\mathcal{C} \sim \mathcal{T}) \sim \mathcal{S} = (\mathcal{C} \sim \mathcal{D}_{\mathcal{P}}) \sim \mathcal{S}$, and this domain characterizes non-termination & non-strictness analysis, as depicted below.



$(\mathcal{C} \sim \mathcal{S}) \sim \mathcal{T} = (\mathcal{C} \sim \mathcal{T}) \sim \mathcal{S} = (\mathcal{C} \sim \mathcal{D}_{\mathcal{P}}) \sim \mathcal{S}$

This application shows the logical relation between comportment analysis and strictness, projection, dual-projection and termination analysis in higher-order abstract interpretation of functional languages. In particular, it shows that comportment analysis cannot be obtained as reduced product of projection and dual-projection, as the elements $\{tot, div\}$ and $\{ide, div\}$ cannot be reconstructed. Indeed,

$\mathcal{C} \sim \mathcal{P}$ is not comparable (as abstract interpretation) with $\mathcal{D}_\mathcal{P}$. However, note that $\mathcal{C} = \mathcal{P} \sqcap \mathcal{D}_\mathcal{P} \sqcap (\mathcal{C} \sim \mathcal{P})$ may provide an alternative factorization of the comportment domain. The Cousot and Cousot's comportment lattice is essential in order to apply our complement factorization to strictness, termination, projection and dual-projection analysis. This because it provides a common base to reason about these analyses in the standard framework of abstract interpretation based on Galois insertions.

6 Logic Programming: Complementing *Def* w.r.t. *Sharing*

In this section, we apply the theory developed previously to the case of *Sharing*, a well known domain for variable aliasing and groundness analysis of logic programs, introduced by Jacobs and Langen in [21]. The ability of this domain to represent, in addition to variable sharing, ground-dependency (also called covering) has already been studied in [5]. In particular, in [5] it has been shown that *Sharing* enjoys a Galois insertion with a more abstract domain that completely captures its ability to express ground-dependency. We show that indeed this domain coincides with the domain *Def*, introduced by Dart in [14]. It is natural then to try to characterize what is left of *Sharing* once we take *Def* out of it, i.e. the *complement* of *Def* with respect to *Sharing*. Of course, this domain must capture *exactly* the information that is represented by *Sharing* but ignored by *Def*. Thus, on the one hand, it must represent variable independence and sharing, and, on the other hand, it must disregard ground-dependency. We will show that such a domain is characterized by a surprisingly simple closure operator on *Sharing*.

6.1 The Domain *Sharing*

Let Var be a countable set of variables, and let VI be any (non-empty) finite subset of Var containing the variables of interest. As usual, variables are denoted by $x, y, z, u, \ldots$. We assume that the concrete domain of computation of a given logic program is the powerset $\wp(Subst)$ of idempotent substitutions, ordered with set-theoretic inclusion. Every substitution $\sigma \in Subst$ is an idempotent function mapping each $x \in Var$ to a term $\sigma(x)$ built on the variables in Var, such that $\sigma(x) \neq x$ for a finite number of variables x. A substitution σ is typically specified by listing its non-trivial bindings, viz. $\sigma = \{x/\sigma(x) \mid \sigma(x) \neq x\}$.

In what follows, several abstract domains will de described. For the sake of simplicity, the abstraction and concretization functions for a domain will be denoted simply by α and γ, since the context will allow to disambiguate them correctly.

The abstract domain *Sharing* is defined by $\{S \subseteq \wp(VI) \mid S \neq \emptyset \Rightarrow \emptyset \in S\}$. *Sharing* is a finite distributive lattice with respect to the partial order given by set-theoretic inclusion. It enjoys a Galois insertion into the concrete domain $\wp(Subst)$. For $y \in Var$, let $occ(\sigma, y) = \{z \in VI \mid y \in var(\sigma(z))\}$. The abstraction of a singleton $\{\sigma\}$ is defined as $\alpha(\{\sigma\}) = \{occ(\sigma, y) \mid y \in Var\}$. Thus, the abstraction and concretization functions between $\wp(Subst)$ and *Sharing* are defined as follows: $\alpha(\Sigma) = \cup\{\alpha(\{\sigma\}) \mid \sigma \in \Sigma\}$ and $\gamma(S) = \cup\{\Sigma \mid \alpha(\Sigma) \subseteq S\}$. Hence, the following points hold about $\alpha(\{\sigma\})$. Let x and y be in VI: (i) x and y share in

σ iff $\exists A \in \alpha(\{\sigma\})$ such that $\{x, y\} \subseteq A$; (ii) x is ground in σ iff for no element $A \in \alpha(\{\sigma\})$, $x \in A$. For instance, assuming $VI = \{x, y, z, u\}$, the element $\{\emptyset, \{y, z\}, \{y, z, u\}\}$ is an element of *Sharing* representing substitutions with respect to which x is ground, and z and y may share, and so may y and u, and z and u. In particular[2], $\sigma_1 = \{x/a, y/b, z/c\}$ and $\sigma_2 = \{x/b, y/v, z/v, u/v\}$ satisfy these properties. Therefore, $\{\sigma_1, \sigma_2\} \subseteq \gamma(\{\emptyset, \{y, z\}, \{y, z, u\}\})$.

6.2 *Def*: an Abstraction of *Sharing* Expressing its Ground-Dependency Information

The ground-dependency information on VI represented by *Sharing* has been characterized in [5] by means of another domain whose elements are Boolean functions on VI. As shown below, this domain coincides exactly with the well known domain *Def* ([14]). It is well known that Boolean functions can be represented by means of propositional formulae (e.g. see [1, 22]). Recall that a Boolean function f is *positive* if $f(true, \ldots, true) = true$. *Def* is the finite lattice (with respect to the usual implication partial order $\models$) of positive Boolean functions whose models are closed under intersection, plus the bottom element $false$ (for more details see [1]). The abstraction and concretization maps between *Def* and $\wp(Subst)$ are well known, and can be found, e.g., in [22]. For instance, assuming $VI = \{x, y, z, u\}$, the formula $x \wedge (y \leftrightarrow z)$ is an element of *Def* that represents 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 also $\sigma'(z)$ 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, $\{\sigma_1, \sigma_2\} \subseteq \gamma(x \wedge (y \leftrightarrow z))$.

The abstraction function which maps an element S in *Sharing* into a formula capturing its ground-dependency information is defined as follows[3] ([5]):

$$\mathcal{C}(S) = \wedge\{\wedge W_1 \rightarrow \wedge W_2 \mid W_1, W_2 \subseteq VI\ ,\ \forall A \in S. (W_2 \cap A \neq \emptyset) \Rightarrow (W_1 \cap A \neq \emptyset)\}.$$

For instance, if $VI = \{x, y, z, u\}$ and $S = \{\emptyset, \{x, y\}, \{x, z\}\}$, then the formula $\mathcal{C}(S) = u \wedge (x \rightarrow (y \wedge z))$ outlines the fact that for every $\sigma \in \gamma(S)$ the variable u is ground in σ, and $\{x\}$ covers $\{y, z\}$ in σ.

Actually, the image of *Sharing* through the abstraction map $\mathcal{C}$ is the domain *Def*, as stated below.

Proposition 17. *The abstraction map $\mathcal{C}$ defines a G.i. of Def into Sharing.*

The interest in this abstraction of *Sharing* relies also on the fact that it completely captures the computational behaviour of *Sharing* with respect to groundness, i.e. if we consider any abstract computation in *Sharing* starting from an element S and resulting in the element S', the corresponding computation in *Def* starting from $\mathcal{C}(S)$ produces $\mathcal{C}(S')$ as result ([6]).

---

[2] We adopt the usual notation denoting ground terms by $a, b, c, \ldots$.

[3] If W is a set of formulae then $\wedge W$ denotes the conjunction of these formulae.

6.3 *Sharing*$^+$: the Complement of *Def* w.r.t. *Sharing*

We have already observed that the elements of *Sharing* contain several sorts of information on variable instantiation and dependency. An important rôle is played by the singletons. Consider, for instance, $S \in Sharing$ that contains the singleton $\{x\}$, for some $x \in VI$; the following two points hold for such an S.

(i) From the definition of $\mathcal{C}$, it follows that there is no $W \subseteq VI$ such that $x \notin W$ and $\mathcal{C}(S) \models \wedge W \rightarrow x$. This means that S carries neither groundness nor covering information about x.

(ii) From the observations at the end of Subsection 6.1, it easily follows that if we consider S and $S' = S \setminus \{x\}$, then for all $y, z \in VI$, there exists $\sigma \in \gamma(S)$ such that y and z share in σ iff there exists $\sigma' \in \gamma(S')$ such that y and z share in it. Thus, singletons have no influence on the sharing information contained in S.

Using the above two points together, we can conclude that adding singletons to an element $S \in Sharing$ eliminates its groundness and covering information and preserves its sharing information. This suggests to define a new abstract domain, that we call *Sharing*$^+$, by considering only those elements of *Sharing* that contain all the singletons of VI. Let $T = \{\{x\} \mid x \in VI\} \cup \{\emptyset\} \in Sharing$, and consider the operator $(\cdot)^+$ on *Sharing* defined as $S^+ = S \cup T$, for all $S \in Sharing$.

Proposition 18. *$(\cdot)^+ : Sharing \rightarrow Sharing$ is a closure operator.*

Thus, the domain *Sharing*$^+$ is exactly the image, i.e. the set of fixpoints, of this closure operator, namely $Sharing^+ = \{S \cup T \mid S \in Sharing\}$.

By point (i) above, next result follows easily. It makes formal the intuition that each element of *Sharing*$^+$ expresses no covering information.

Proposition 19. *$\forall S \in Sharing^+ .\ \mathcal{C}(S) = true$.*

Next theorem states the main result of this section: *Sharing*$^+$ is the complement of *Def* with respect to *Sharing*, and *Def* is the most abstract domain that factorizes *Sharing* into *Sharing*$^+$.

Theorem 20. *$Sharing \sim Def = Sharing^+$ and $Sharing \sim Sharing^+ = Def$.*

The interest of the above theorem is twofold. First, it is an example of the practical impact of the notion of complement presented in the paper. Secondly, it is an example of modularization of abstract domains, that can be easily integrated with the results in [3] to improve efficiency and precision of the analysis.

7 Conclusion

In this paper we have introduced the notion of complementation in abstract interpretation. Complementation can also be used for semantics related by abstract interpretation. Cousot and Cousot proved in [10, 11] that abstract interpretation can be used to systematically design hierarchies of semantics. In particular, both the standard denotational and axiomatic semantics can be derived by abstract interpretation of a generalized SOS operational semantics of the language. This technique has been recently applied in logic programming in [4, 18], where hierarchies of collecting semantics are designed by abstracting SLD resolution. In particular, [18] proved that it is always possible to design "optimal" collecting semantics for analysis of logic programs, by composing (with reduced product) the declarative semantics of Herbrand models with the specific property to model. The interest in complementation is therefore apparent in this field: semantics, as well as analyses, can be composed and complemented, providing a *real algebra* of observable properties and semantics of programming languages.

Acknowledgements. The work of Roberto Giacobazzi has been partly supported by the EEC *Human Capital and Mobility* individual grant: "Semantic Definitions, Abstract Interpretation and Constraint Reasoning", n. ERBCHBICT930822. The work of Francesco Ranzato has been partly done during a stay at *LIX, Laboratoire d'Informatique, École Polytechnique*, Paris. He thanks *LIX*, and in particular Radhia Cousot, for the kind hospitality.

References

1. T. Armstrong, K. Marriott, P. Schachte, and H. Søndergaard. Boolean functions for dependency analysis: algebraic properties and efficient representation. In *Proc. of SAS '94*, LNCS 864, pp. 266–280, 1994.
2. G. Birkhoff. *Lattice Theory.* AMS Colloq. Publ., vol. XXV, 3rd edition, 1967.
3. M. Codish, A. Mulkers, M. Bruynooghe, M. García de la Banda, and M. Hermenegildo. Improving abstract interpretations by combining domains. In *Proc. of ACM PEPM '93*, pp. 194–206, 1993.
4. M. Comini and G. Levi. An algebraic theory of observables. In *Proc. of ILPS '94*, pp. 172–186, 1994.
5. A. Cortesi, G. Filé, and W. Winsborough. Comparison of abstract interpretations. In *Proc. of ICALP '92*, LNCS 623, pp. 521–532, 1992.
6. A. Cortesi, G. Filé, and W. Winsborough. The quotient of an abstract interpretation. Tech. Rep. 12/94, Dip. di Mat. Pura ed Appl., Univ. di Padova, 1994.
7. P. Cousot. *Méthodes Itératives de Construction et d'Approximation de Points Fixes d'Opérateurs Monotones sur un Treillis, Analyse Sémantique des Programmes.* Ph.D. Thesis, Univ. de Grenoble, 1978.
8. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. of ACM POPL '77*, pp. 238–252, 1977.
9. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. of ACM POPL '79*, pp. 269–282, 1979.
10. P. Cousot and R. Cousot. Constructing hierarchies of semantics by abstract interpretation. Invited Talk at *WSA '92*, Bordeaux, 1992.
11. P. Cousot and R. Cousot. Inductive definitions, semantics and abstract interpretation. In *Proc. of ACM POPL '92*, pp. 83–94, 1992.
12. P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2,3):103–179, 1992.
13. 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. of ICCL '94*, pp. 95–112, 1994.

14. P. Dart. On derived dependencies and connected databases. *Journal of Logic Programming*, 11(2):163–188, 1991.
15. P. Dwinger. On the closure operators of a complete lattice. *Indagationes Math.*, 16:560–563, 1954.
16. G. Filé and F. Ranzato. Improving abstract interpretations by systematic lifting to the powerset. In *Proc. of ILPS '94*, pp. 655–669, 1994.
17. O. Frink. Pseudo-complements in semi-lattices. *Duke Math. Journal*, 29:505–514, 1962.
18. R. Giacobazzi. On the collecting semantics of logic programs. In *Proc. of the Post-ICLP '94 Workshop on Verif. and Analysis of Logic Lang.*, pp. 159–174, 1994.
19. R. Giacobazzi, C. Palamidessi, and F. Ranzato. Weak relative pseudo-complements of closure operators. Tech. Rep. LIX/RR/95/04, Laboratoire d'Informatique, École Polytechnique, Paris, 1995.
20. S. Hunt. PERs generalize projections for strictness analysis. In *Proc. of the 1990 Glasgow Funct. Progr. Workshop*, pp. 156–168, 1990.
21. D. Jacobs and A. Langen. Static analysis of logic programs for independent AND-parallelism. *Journal of Logic Programming*, 13(2,3):154–165, 1992.
22. K. Marriott and H. Søndergaard. Precise and efficient groundness analysis for logic programs. *ACM LOPLAS*, 2(1–4):181–196, 1993.
23. J. Morgado. Note on complemented closure operators of complete lattices. *Portugal. Math.*, 21(3):135–142, 1962.
24. K. Muthukumar and M. Hermenegildo. Combined determination of sharing and freeness of program variables through abstract interpretation. In *Proc. of ICLP '91*, pp. 49–63, 1991.
25. A. Mycroft. The theory and practice of transforming call-by-need into call-by-value. In *Proc. of 4th Intern. Symp. on Programming*, LNCS 83, pp. 270–281, 1980.
26. A. Mycroft. *Abstract interpretation and optimizing transformations for applicative programs*. Ph.D. Thesis, Dept. of Comp. Sc., Univ. of Edinburgh, CST-15-81, 1981.
27. F. Nielson. Tensor products generalize the relational data flow analysis method. In *Proc. of 4th Hung. Comp. Sc. Conf.*, pp. 211–225, 1985.
28. R. Sundararajan and J. Conery. An abstract interpretation scheme for groundness, freeness, and sharing analysis of logic programs. In *Proc. of FST&TCS '92*, LNCS 652, pp. 203–216, 1992.
29. J. Varlet. Contribution à l'étude des treillis pseudo-complémentés et des treillis de Stone. *Mém. de la Soc. Roy. des Sc. de Liège*, 5eme série, 8(4):1–71, 1963.
30. P.L. Wadler and R.J.M. Hughes. Projections for strictness analysis. In *Proc. of ACM FPCA '87*, LNCS 274, pp. 385–407, 1987.
31. M. Ward. The closure operators of a lattice. *Ann. of Math.*, 43(2):191–196, 1942.