Complementation in Abstract Interpretation

AGOSTINO CORTESI
Università di Venezia
GILBERTO FILÉ
Università di Padova
ROBERTO GIACOBAZZI
Università di Pisa
CATUSCIA PALAMIDESSI
Università di Genova
and
FRANCESCO RANZATO
Università di Padova

Reduced product of abstract domains is a rather well-known operation for domain composition in abstract interpretation. In this article, we study its inverse operation, introducing a notion of domain complementation in abstract interpretation. Complementation provides a systematic way to design new abstract domains, and it allows to systematically decompose domains. Also, such an operation allows to simplify domain verification problems, and it yields space-saving representations for complex domains. We show that the complement exists in most cases, and we apply complementation to three well-known abstract domains, notably to Cousot and Cousot's interval domain for integer variable analysis, to Cousot and Cousot's domain for comportment analysis of functional languages, and to the domain *Sharing* for aliasing analysis of logic languages.

Categories and Subject Descriptors: D.3.1 [**Programming Languages**]: Formal Definitions and Theory—*semantics*; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*logics of programs*

General Terms: Languages, Theory

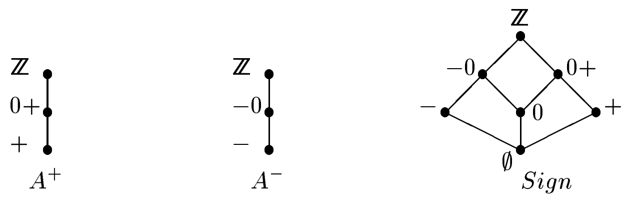
Additional Key Words and Phrases: Abstract domain, abstract interpretation, closure operator, complementation, functional and logic programming, program analysis

---

The work of R. Giacobazzi has been partly supported by the EEC-HCM individual grant: "Semantic Definitions, Abstract Interpretation and Constraint Reasoning," n. ERBCHBICT930822. The work of A. Cortesi and G. Filé has been partly supported by "MURST-NRP: Modelli della Computazione e dei Linguaggi di Programmazione."
A preliminary version of this article appeared in *Proceedings of the 2nd International Static Analysis Symposium* (*SAS '95*), Glasgow, September 1995.
Authors' addresses: A. Cortesi, Università di Venezia, Dipartimento di Matematica Applicata e Informatica, Via Torino 155, 30173 Mestre-Venezia, Italy; email: cortesi@moo.dsi.unive.it; G. Filé and F. Ranzato, Università di Padova, Dipartimento di Matematica Pura ed Applicata, Via Belzoni 7, 35131 Padova, Italy; email: {gilberto; franz}@math.unipd.it; R. Giacobazzi, Università di Pisa, Dipartimento di Informatica, Corso Italia 40, 56125 Pisa, Italy; email: giaco@di.unipi.it; C. Palamidessi, Università di Genova, Dipartimento di Informatica e Scienze dell'Informazione, Via Dodecaneso 35, 16146 Genova, Italy; email: catuscia@disi.unige.it.

1. INTRODUCTION

Abstract interpretation is a general theory, introduced by Cousot and Cousot [1977; 1979], for describing the relationship among semantics of programming languages at different levels of abstraction. In this framework, program analysis is defined as nonstandard program semantics, obtained from the standard one by substituting its domain of computation, called *concrete*, (and the basic operations on it) with an *abstract domain* (and corresponding abstract operations). The concrete and the abstract domains are always complete lattices, where the ordering relations describe the relative precision of the denotations, the top elements representing no information. For example, assume that the concrete domain is the powerset $\wp(\mathbb{Z})$ of integer numbers. This may be the case whenever we perform a static analysis on variables assuming integer values. Possible abstract domains are depicted in the picture below. The interpretation of their elements is straightforward: for instance, $0+$ represents the set of nonnegative integers whereas $-$ represents the set of negative integers.



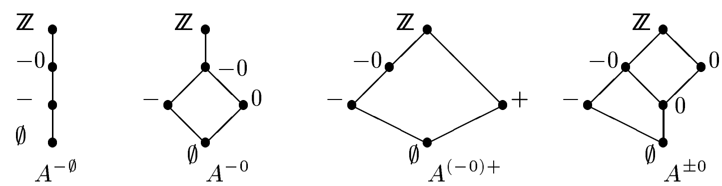
The relation of abstraction between abstract domains is traditionally formalized in terms of *Galois insertions* (cf. Cousot and Cousot [1977]). A domain D is an abstraction of C, if there exists a monotone abstraction function $\alpha : C \to D$, which is part of a Galois insertion. This happens whenever α is surjective, i.e., all the elements of D are approximations of elements in C, and if $c \in C$ is such an element, then $\alpha(c) \in D$ is the least (unique) element in D which corresponds to an object in C approximating the meaning of c. This correspondence is defined in terms of an adjoint function $\gamma : D \to C$, providing a representation in C for the elements of D. The relation between domains established by a Galois insertion ensures that the abstract domain contains only the best (viz., most precise) approximations of the elements of the concrete one. This is the case of A^+ and *Sign* in the example above. In this case, $-0 \in Sign$ is abstracted into $\mathbb{Z}$ of A^+, which corresponds to the least set in A^+ containing nonpositive numbers, while $0 \in Sign$ is approximated by $0+ \in A^+$, which corresponds to the least set in A^+ containing 0.

Already in their early works Cousot and Cousot [1977; 1979], have pointed out the importance of incrementally designing abstract domains. Richer domains can be obtained by combining simpler ones or by lifting them by systematically adding new information. The first kind of operations are known as *domain combinators*, while the latter ones are known as *domain completions*. Both operations are devoted to enhance the expressive power of domains and have been called *domain refinements* (cf. Filé et al. [1996]). All these domain operations provide high-level facilities to tune the analysis in accuracy and cost. Among these operations for domain refinement, *reduced product* [Cousot and Cousot 1979] is probably the most common and widely known one. It has been included as an important tool for design aid in many modern systems for program analysis, like for instance in System Z [Yi and Harrison 1993]. In Codish et al. [1995] the cost/precision tradeoff between using two separate abstract domains and using their reduced product has been experimentally evaluated for the analysis of logic programs in the system PLAI.

Reduced product corresponds to cartesian product of domains, where equivalent tuples are identified, viz., reduced. Let us use the abstract domains introduced above in order to illustrate reduced product. The reduced product of A^+ and A^- is *Sign*. Observe that *Sign* has two new elements with respect to A^+ and A^-: 0 and $\emptyset$. These elements are obtained by combining, by conjunction, which is set intersection of the corresponding sets of integers, respectively, $0+$ with -0 and $+$ with $-$. In this example, reduction is necessary only for identifying distinct pairs of elements denoting $\emptyset$, as, for instance, the pairs $\langle +, - \rangle$ and $\langle +, -0 \rangle$.

The Problem: Domain Decomposition in Abstract Interpretation. A natural question that arises in this setting is whether it is possible to define the *inverse of 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. Recall that in the setting of abstract interpretation a domain D is more abstract than C, if C contains all the information of D, by simplifying: $D \subseteq C$, up to isomorphic representation of domain objects. In the example above, for instance, A^+ is more abstract than *Sign*.

Because of the peculiar structure of domains in abstract interpretation, the above problem cannot be solved by simply considering the set-theoretic complement $C \setminus D$ of D into C. This is because the result would not be in general an abstract domain. In our example, for instance, $Sign \setminus A^+ = \{-0, -, 0, \emptyset\}$ is not an abstract domain: in particular, $Sign \setminus A^+$ does not contain any correct (with respect to the standard meaning of signs as sets of integers) denotation representing no information at all, namely denoting the set of all integers. In contrast, it is easy to verify that the domains A^- and *Sign* above, and the domains $A^{-\emptyset}$, A^{-0}, $A^{(-0)+}$, and $A^{\pm 0}$ depicted below, are all and only those abstractions of *Sign* which, once combined with A^+ by reduced product, give *Sign* back.



It is worth noting that A^- is contained in (viz., more abstract than) all these domains. Hence, in this sense, $A^- = Sign \sim A^+$. The main question is therefore whether for any two domains C and D as above, the domain $C \sim D$ always exists. A positive answer to this question would fill one gap of abstract interpretation theory, providing a methodology for systematic domain decomposition. This is important for several reasons. First, it can be useful for designing new abstract domains and for minimizing their space requirement, yielding compact representations for complex domains as tuples of simpler (i.e., more abstract) factors. This can be particularly interesting for those domains that were not originally designed through composition, which is the case for most of the abstract domains used in practical analyses. Furthermore, decomposing domains allows to decompose also their verification. In fact, instead of proving some property for a complex domain, it may be convenient to verify it for the simpler components of a decomposition of this domain. Clearly, this is a viable technique only provided that the considered property is preserved under reduced product. Finally, domain decomposition helps to understand the internal structure of complex domains, which, as already observed, are in most cases defined as a whole and not by composition of elementary domains. Such an operation would allow to show for instance that $Sign \sim A^+ = A^-$ and $Sign \sim A^- = A^+$, and thus that $\langle A^+, A^- \rangle$ provides a minimal way of decomposing $Sign$, in the sense that no component can be simplified without changing the other one.

The Idea: Pseudocomplementation for Domain Decomposition. In order to solve the problem of inverting reduced product in a general setting, i.e., for any domain in abstract interpretation, we need a formal framework, where properties of abstract domains can be studied independently from the representation of their objects. In fact, the definition of abstraction in terms of Galois insertions suffers from being dependent on the way domain objects are represented.

The closure operator approach to abstract interpretation provides a very useful mathematical framework for studying abstract domains and in particular the above operation for domain decomposition. The equivalence between Galois insertions and closure operators is well known in lattice theory (cf. Birkhoff [1967]). Cousot and Couost [1979] applied this equivalence, observing that every abstraction of a 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* $\sqcap$ on the complete lattice $uco(C)$ of all (upper) closure operators on C. Closure operators play here the role of approximating operations. The intuition behind this construction is simple. A closure operator $\rho : C \to C$ is a function which is monotonic, idempotent (i.e., $\rho(\rho(c)) = \rho(c)$), and extensive (i.e., $\rho(c) \geq c$). Monotonicity ensures that the abstraction monotonically approximates domain objects. Idempotence ensures that the approximation is performed all at once, while extensivity captures the situation where the approximation of an object c contains all the information in c. Closure operators capture therefore the essence of the process of abstraction. Moreover, they have the advantage of simplicity, e.g., one single function, and specify abstractions independently from isomorphic representation of domain objects. Hence, they provide precisely the right high-level setting which is needed when reasoning about properties of abstract domains. For example, it is easy to verify that A^+ and A^- correspond, up to object names, to the image of $Sign$ under the following closure operators:

$$\rho_{A^+}(x) = \begin{cases} x & \text{if } x \in \{+, 0+, \mathbb{Z}\} \\ 0+ & \text{if } x = 0 \\ + & \text{if } x = \emptyset \\ \mathbb{Z} & \text{if } x \in \{-, -0\}; \end{cases} \qquad \rho_{A^-}(x) = \begin{cases} x & \text{if } x \in \{-, -0, \mathbb{Z}\} \\ -0 & \text{if } x = 0 \\ - & \text{if } x = \emptyset \\ \mathbb{Z} & \text{if } x \in \{+, 0+\}. \end{cases}$$

The lattice of abstract interpretations of a domain C is therefore isomorphic to the lattice $uco(C)$ of closure operators on C. The bottom element in $uco(C)$ corresponds to the identical abstraction, which leaves C unchanged, while the top element corresponds to the straightforward abstraction, which forgets about the whole structure of C, mapping C into its top element. In this setting, asking whether $C \sim D$ exists for any C and D (with D abstracting C) is equivalent to ask whether $uco(C)$ is *pseudocomplemented*. Being a pseudocomplement of D in $uco(C)$ for a domain (closure operator) $C \sim D$ means that whenever there exists another domain X, such that $D \sqcap X = C$, then X is always more concrete than $C \sim D$. Therefore, in the context of abstract interpretation, the lattice-theoretic notion of pseudocomplementation [Birkhoff 1967] captures precisely the intended meaning of inverting reduced product. In the example above, it is easy to see that $A^+ \sqcap A^- = Sign$, and for any domain X such that $A^+ \sqcap X = Sign$, e.g., $X = A^{-0}$, X contains $Sign \sim A^+ = A^-$.

The attentive reader may wonder why we have defined the operation $\sim$ as the pseudocomplement instead of the complement, which would additionally require that the least upper bound of $C \sim D$ and D in $uco(C)$ is the top element. This is the case in our example, where the most abstract domain (viz., the top closure) $\{\mathbb{Z}\}$ is the only common abstraction of A^+ and A^-. Although some closure (domain) may have accidentally a complement, this is not true in general, i.e., for any closure. This would clearly correspond to requiring that $uco(C)$ is *complemented*, but Dwinger [1954] and, successively, Morgado [1962] 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 in general are not complete chains. On the contrary, the requirement that $uco(C)$ is pseudocomplemented can be met in most cases of interest. In fact, recently, Giacobazzi et al. [1996] have given a sufficient condition for $uco(C)$ to be pseudocomplemented. In the present article, we point out that this condition is satisfied by most of the known concrete and abstract domains for semantics and analysis of programming languages.

Structure of the Article. On the basis of the result in Giacobazzi et al. [1996], we define the operation of pseudocomplementation of abstract domains. For the sake of simplicity, this operation will be simply called *complementation*. We study the basic properties of complementation for abstract domain decomposition, and we provide a constructive iterative characterization of complements. We introduce the notion of *minimal decomposition* for an abstract domain, and we provide an iterative method to construct minimal decompositions, i.e., decompositions of domains involving the most abstract factors. The usefulness of complementation for domain decomposition is illustrated by means of several examples. First, we consider some general properties of domains which can be verified *compositionally* on the simpler factors of their decompositions. Then, we apply domain decomposition to specific well-known domains for program analysis. Cousot and Cousot's [1976; 1977] domain for *integer interval* analysis of programs with integer variables provides the first simple, but useful, example for explaining some subtle technical points of the complementation. More complex examples of the use of complementation for domain decomposition are given by considering a domain for aliasing analysis in logic programming, notably *Sharing*, introduced by Jacobs and Langen [1989; 1992], and Cousot and Cousot's [1994] domain for *comportment* analysis for higher-order functional languages.

The domain for comportment analysis was originally obtained by disjunctive completion of a simpler domain of basic comportments for functions. We prove that the domain of basic comportments can be decomposed by complementation as reduced product of Mycroft's [1980; 1981] domains for *termination* and *strictness* analysis. This provides a reduction of the lattice-structure of comportments as well as an intuitive interpretation of comportment analysis as the space of relations between the domains for termination and strictness. For the sake of simplicity, the domain of comportment analysis is considered for the case of functional basic types only.

We use complementation for decomposing *Sharing* into a component expressing the ability of *Sharing* to compute ground dependency information and a component expressing the remaining information of *Sharing*. Cortesi et al. [1992] 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* for ground dependency analysis. *Def* was introduced by Marriott and Søndergaard [1993], as an adaptation of Dart's [1988; 1991] work on groundness in deductive databases. As expectable, the complement of *Def* relative to *Sharing*, called *Sharing*$^+$, captures precisely variable aliasing and no ground dependency information. It is worth mentioning that *Sharing*$^+$ corresponds to a simple and elegant closure operator on *Sharing*.

The rest of the article is organized as follows. Section 2 contains some preliminaries about lattice theory and abstract interpretation. Section 3 introduces the notion of complementation of abstract domains and studies its properties. In this section, we also give the constructive fixpoint characterization of complementation of closure operators, which is the basis for an iterative construction of complements of abstract domains. Section 4 gives systematic ways to decompose domains and exemplifies how the notion of complementation actually works. In Section 5 we give some examples of domain properties which can be verified on the simpler factors of their decompositions. Section 6 presents complements of some abstractions of Cousot and Cousot's integer interval domain. Section 7 provides decompositions of Cousot and Cousot's domain for comportment analysis. Section 8 describes the application of the complement to the domain *Sharing*. Section 9 concludes.

2. PRELIMINARIES

Throughout the article, we will assume familiarity with the basic notions of lattice theory (e.g., see Birkhoff [1967], Davey and Priestley [1990], and Grätzer [1978]) and abstract interpretation [Cousot and Cousot 1977; 1979]. Now, we briefly introduce some notation and recall some well-known notions.

2.1 Mathematical Notation

Let C and D be sets. The powerset of C is denoted by $\wp(C)$, and the cardinality of C by $|C|$. The set-difference between C and D is denoted by $C \setminus D$. If f is a function defined on C and $D \subseteq C$ then $f(D) = \{f(x) \mid x \in D\}$. By $g \circ f$ we denote the composition of the functions f and g, i.e., $\forall x.\ (g \circ f)(x) = g(f(x))$. The set D equipped with a partial order $\preceq$ is denoted by $\langle D, \preceq \rangle$. If D is a poset, we usually denote by $\preceq_D$ the corresponding partial order. A complete lattice D with partial order $\preceq$, greatest lower bound (*glb*) $\wedge$, least upper bound (*lub*) $\vee$, top element $\top = \wedge\emptyset = \vee D$, and bottom element $\bot = \vee\emptyset = \wedge D$ is denoted by $\langle D, \preceq, \wedge, \vee, \top, \bot\rangle$. When D is a lattice, $\vee_D$, $\wedge_D$, $\top_D$, and $\bot_D$ denote the corresponding basic operators and elements. In the following, we will often abuse notation by denoting lattices with their poset notation. We use $C \cong D$ to denote that the ordered structures C and D are isomorphic.

2.2 Galois Connections and Closure Operators

We recall the notions of Galois connection and insertion and closure operator, which are fundamental in the standard theory of abstract interpretation [Cousot and Cousot 1977; 1979]. 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. We also recall that the above definition of G.c. is equivalent to that of adjunction: (γ, D, C, α) is an *adjunction* if $\forall c \in C. \forall d \in D.\ \alpha(c) \preceq_D d \Leftrightarrow c \preceq_C \gamma(d)$. Abstract interpretations are traditionally specified in terms of Galois insertions. 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 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. Furthermore, if C is not an abstraction of D, then we say that D is a *strict* abstraction of C. Note that, if (γ, D, C, α) is a G.i., then the concretization and abstraction mappings, γ and α, are 1-1 and onto, respectively. In this case, each value of the abstract domain D is useful in the representation of the concrete domain C, as all the elements of D represent distinct members of C.

An (*upper*) *closure operator*, or simply a *closure*, on the poset $\langle L, \preceq\rangle$ is a monotonic, idempotent, and extensive (viz., $\forall x \in L.\ x \preceq \rho(x)$) operator $\rho : L \to L$. If $\langle L, \preceq, \wedge, \vee, \top, \bot\rangle$ is a complete lattice then each closure operator ρ is uniquely determined by the set of its fixpoints, which is its image $\rho(L)$. A set $X \subseteq L$ is the set of fixpoints of a closure operator iff X is a *Moore-family* of L, i.e., $\top \in X$ and X is meet-closed (viz., for any $Y \subseteq X$, $\wedge Y \in X$). 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 sublattice of L, since the *lub* in $\rho(L)$ might be different from that in L. Hence, in the following, a closure operator ρ will often denote the set of its fixpoints $\rho(L)$. Being a set, the set of fixpoints of a closure is often denoted with capital Latin letters. Denoting closures by sets will be particularly convenient when closure operators will denote domains. In the following, we will keep Greek letters only as denotations for closures in lattice-theoretic notions and results and will use capital Latin ones to emphasize their role as domains. However, viewing closures as functions is also important in abstract interpretation, because they define the abstraction function. Hence, in the following, we will keep this soft ambiguity by using both notations and will leave to the reader to distinguish their use as functions or sets, according to the context. Ward [1942, Theorem 4.2] proved that if L is a complete lattice, then $\langle uco(L), \sqsubseteq, \sqcap, \sqcup, \lambda x.\top, \lambda x.x \rangle$ is a complete lattice, where for every $\rho, \eta \in uco(L)$, $\{\rho_i\}_{i \in I} \subseteq uco(L)$, and $x \in L$:

—$\rho \sqsubseteq \eta$ iff $\forall x \in L.\ \rho(x) \preceq \eta(x)$, or, equivalently, $\rho \sqsubseteq \eta$ iff $\eta(L) \subseteq \rho(L)$;

—$(\sqcap_{i \in I} \rho_i)(x) = \wedge_{i \in I} \rho_i(x)$;

—$(\sqcup_{i \in I} \rho_i)(x) = x \Leftrightarrow \forall i \in I.\ \rho_i(x) = x$;

—$\lambda x.\top$ is the top element, whereas $\lambda x.x$ is the bottom element.

2.3 Abstract Interpretation and Closure Operators: Reduced Product Revisited

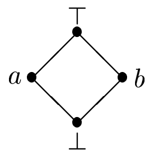
A key point in abstract interpretation theory [Cousot and Cousot 1979] is the equivalence between the Galois insertion and closure operator approach to the design of abstract domains. Usually, the Galois insertion approach is the most used. In this case, D is an abstraction of C if there exist α and γ such that (γ, D, C, α) is a Galois insertion. It is well known since Cousot and Cousot [1979] 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 $\rho = \gamma \circ \alpha$ on C. On the other hand, if ρ is a closure on C and $\iota : \rho(C) \to D$ is an isomorphism of complete lattices (with inverse ι^{-1}) then $(\iota^{-1}, D, C, \iota \circ \rho)$ is a G.i. By the above equivalence, it is not restrictive, and often more convenient, to use the closure operator approach to reason about abstract properties up to isomorphic representations of abstract domains. Thus, in the rest of the article, we will feel free to use this approach most of the time, and whenever we will say that D is an abstraction of C (or C a concretization of D), we will mean that $D \cong \rho(C)$ for some closure $\rho \in uco(C)$. It is well known [Cousot and Cousot 1979] 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 $D_1 \sqsubseteq D_2$. Therefore, to compare domains with regard to their precision, we will only speak about abstractions between them and will use $\sqsubseteq$ to relate both nonhomogeneous domains, i.e., domains which are not Moore-families of the same concrete domain, and homogeneous domains, i.e., closure operators on a concrete domain. Further, because we will be independent from object representations in domains, 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 least upper bound $(\sqcup_{i \in I} \rho_i)(C)$ is the most concrete among the domains which are abstractions of all the D_i, i.e., D is the least common abstraction of all the D_i. The interpretation of the *glb* operation on $uco(C)$ is twofold. First, any domain D isomorphic to the greatest lower bound $(\sqcap_{i\in I}\rho_i)(C)$ is (isomorphic to) the well-known *reduced product* [Cousot and Cousot 1979] of all the domains D_i. Further, the *glb* D, and hence reduced product, is the most abstract among the domains (abstracting C) which are more concrete than every D_i. Thus, we will denote reduced product of abstract domains by the *glb* symbol $\sqcap$.

3. (PSEUDO)COMPLEMENTS OF ABSTRACT DOMAINS

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 possible, in general, to define the lattice-theoretic complement of abstract domains, i.e., for any domain D such that $C \sqsubseteq D$, a domain $\bar{D}$ such that $D \sqcap \bar{D} = C$ and $D \sqcup \bar{D} = \{\top_C\}$. This follows from results by Dwinger [1954] and Morgado [1962], saying that $uco(C)$ is complemented (or a Boolean algebra) iff C is a complete well-ordered chain. A similar condition, on arbitrary complete chains, is also necessary and sufficient for $uco(C)$ to be distributive [Dwinger 1954]. Both these conditions are clearly too restrictive for abstract interpretation of programming languages. The following example shows this problem in a simple finite lattice.

Example 3.1. Consider the following finite lattice C.

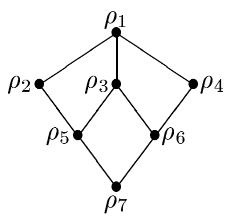


The closure operators (or equivalently abstract interpretations) on C are the following:

$$\rho_1 = \{\top\},\ \rho_2 = \{\top, a\},\ \rho_3 = \{\top, \bot\},\ \rho_4 = \{\top, b\},$$

$$\rho_5 = \{\top, a, \bot\},\ \rho_6 = \{\top, b, \bot\},\ \rho_7 = \{\top, a, b, \bot\}.$$

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 $uco(C)$.

The idea of this article is to use a different and somehow weaker notion of complementation in $uco(C)$ as a systematic approach for domain complementation in abstract interpretation. Indeed, while in general $uco(C)$ is not complemented, it is always *pseudocomplemented*, as proved in Giacobazzi et al. [1996]. We recall the lattice-theoretic notion of pseudocomplement.

Definition 3.2. Let $\langle L, \preceq, \wedge, \bot \rangle$ be a meet semilattice with bottom. The *pseudocomplement* 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 pseudocomplement, we say that L is *pseudocomplemented.*

Pseudocomplement is in this sense weaker than complement because, in general, if L is a lattice, the least upper bound between x and x^* is not the top element of L. It is worth noting that if the pseudocomplement of $x \in L$ exists then it is unique, and in a complete lattice L, it is always defined as (cf. Grätzer [1978])

$$x^* = \vee\{y \in L \mid x \wedge y = \bot\}. \tag{†}$$

Example 3.3. Consider the lattice $uco(C)$ of closure operators of Example 3.1. It is easy to verify the following pseudocomplements 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 $\rho_7^* = \rho_1$.

In the following, we assume that L is a complete lattice.

Definition 3.4. L is *meet-continuous* if, for any chain $C \subseteq L$ and for each $x \in L$, $x \wedge (\vee C) = \vee_{y \in C}(x \wedge y)$.

Remark 3.5. It is worth noting that meet-continuity 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 meet-continuous, but not necessarily complete inf-distributive (which for a finite lattice amounts to be distributive). Moreover complete Heyting algebras, continuous, algebraic, arithmetic, completely distributive, and Boolean complete lattices are always meet-continuous [Gierz et al. 1980, p. 96].

This notion of meet-continuity is central in the following result.

Theorem 3.6 [Giacobazzi et al. 1996]. *If L is meet-continuous then $uco(L)$ is pseudocomplemented.*

By the above remark, the following corollary is immediate.

Corollary 3.7. *If L either satisfies the ascending chain condition, or it is a complete Heyting algebra, or continuous, or algebraic, or arithmetic, or completely distributive, or a Boolean lattice, then $uco(L)$ is pseudocomplemented.*

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

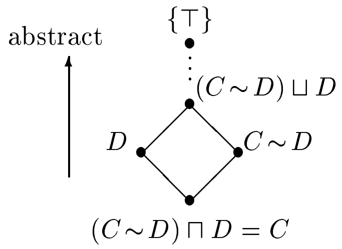
Suppose D is an abstraction of the complete lattice C, and assume that C is meet-continuous. Exploiting Theorem 3.6 we can give the following definition of the complement of abstract domains.

Definition 3.8. The *complement* of D relative to (or in) C is the complete lattice $C \sim D$ given by the set of fixpoints of the pseudocomplement D^* (in $uco(C)$) of D.

It is immediate to observe that if $C \sqsubseteq D$ then $C \sqsubseteq C \sim D$. Therefore, the complement $C \sim D$ is an abstraction of C. By equation (†), the pseudocomplement of D is expressible as

$$D^* = \sqcup\{E \in uco(C) \mid D \sqcap E = C\}.$$

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. The following picture shows the relation between the different domains involved in complementation.



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 3.8, we fix in C the representation of the complement $C \sim D$, being $C \sim D$ a closure on C. Hence, $C \sim D$ is often intended as 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 technical result is recalled from Cousot [1978, Theorem 4.2.0.4.7] and provides a simple way to generalize domain complementation to arbitrary abstractions in the lattice of abstract interpretations of a given domain. For the sake of completeness, we present a sketch of the proof.

Proposition 3.9 [Cousot 1978]. *Let L be a complete lattice and $\eta \in uco(L)$. Then, $uco(\eta) \cong \ \uparrow\eta = \{\rho \in uco(L) \mid \eta \sqsubseteq \rho\}$.*

Proof Sketch. Let L be a complete lattice and $\eta \in uco(L)$. Since η is a complete lattice, $uco(\eta)$ is also a complete lattice. Clearly, the principal filter $\uparrow\eta$ of $uco(L)$ is a complete (sub)lattice too, and $uco(\eta)$ and $\uparrow\eta$ are isomorphic complete lattices. Indeed, the isomorphism is given by the mappings $\Psi : uco(\eta) \to \uparrow\eta$ such that $\Psi(\mu) = \mu \circ \eta$, and $\Phi : \uparrow\eta \to uco(\eta)$ such that $\forall x \in \eta.\ \Phi(\rho)(x) = \rho(x)$. $\square$

By this result, we can apply Theorem 3.6 to arbitrary pairs of elements in the lattice of abstract interpretations. In general, if C is a concrete domain, D an abstraction of C (i.e., $D \in uco(C)$), and E an abstraction of D (i.e., $E \in uco(D)$), then while the computation of $D \sim E$ requires that D is meet-continuous, the domain C can be merely a complete lattice. The complement $D \sim E$ is given in this case by the set of fixpoints of the pseudocomplement $E^* = \sqcup\{X \in uco(D) \mid E \sqcap X = D\}$, which is a closure operator on D. Proposition 3.9 says that $uco(D)$ and $\{X \in uco(C) \mid D \sqsubseteq X\}$ are isomorphic complete lattices. Hence, the complement $D \sim E$ can be equivalently obtained as a pseudocomplement on the more concrete domain C, using the isomorphism of Proposition 3.9. In fact, if, by the isomorphism of Proposition 3.9, we view E as a closure on C (viz., $E = E \circ D \in uco(C)$), then $E^* = \sqcup\{X \in uco(C) \mid (E \circ D) \sqcap X = D\}$, which corresponds precisely to the expected intuitive meaning of $D \sim E$ as closure on C.

Meet-continuity of the concrete domain of reference plays a central role in the existence of the complement (cf. Theorem 3.6). Giacobazzi et al. [1996] observed that meet-continuity is preserved by continuous closures ($\eta \in uco(L)$ is continuous if for any chain $C \subseteq L$, $\eta(\vee C) = \vee \eta(C)$).

Proposition 3.10 [Giacobazzi et al. 1996]. *If L is meet-continuous, and $\eta \in uco(L)$ is continuous, then $\eta(L)$ is meet-continuous.*

Therefore, this result provides a sufficient condition on the closure defining an abstract domain D in order to verify the meet-continuity of D.

The following algebraic properties of the complement operation $\sim$ on abstract domains can be easily derived from similar properties of pseudocomplemented lattices (see Birkhoff [1967], Frink [1962], Grätzer [1978], and Varlet [1963]). Note that (j) includes one of De Morgan's laws.

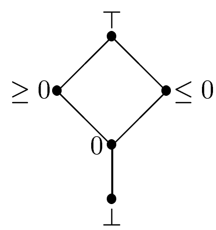
Proposition 3.11. *Let C be a meet-continuous lattice, $C \sqsubseteq D, E$, and $C \sqsubseteq D_i$ for each $i \in I$. Define $\top$ as the most abstract interpretation of C. Then,*

- (a) $D \sqsubseteq C \sim (C \sim D)$;
- (b) $(D \sqsubseteq E) \Rightarrow (C \sim E) \sqsubseteq (C \sim D)$;
- (c) $(C \sim D) = C \sim (C \sim (C \sim D))$;
- (d) $(D \sqsubseteq (C \sim E)$ *and* $D \sqsubseteq E) \Leftrightarrow (D = C)$;
- (e) $((C \sim D) \sqsubset (C \sim E)) \Leftrightarrow (C \sim (C \sim E) \sqsubset C \sim (C \sim D))$;
- (f) $(C \sim D = \top) \Leftrightarrow (D = C)$;
- (g) $C \sim \top = C$ *and* $C \sim C = \top$;
- (h) $(C \sim (D \sqcap E)) = C \sim ((C \sim (C \sim D)) \sqcap (C \sim (C \sim E)))$;
- (i) $C \sim (C \sim (D \sqcap E)) = ((C \sim (C \sim D)) \sqcap (C \sim (C \sim E)))$;
- (j) $(C \sim (D \sqcap E)) = C \sim (C \sim ((C \sim D) \sqcup (C \sim E)))$ *and* $(C \sim (D \sqcup E)) = (C \sim D) \sqcap (C \sim E)$;
- (k) $\{(C \sim D) \mid C \sqsubseteq D\}$ *is a Moore-family of the abstract interpretations of C, containing C and such that* $\sqcap_{i \in I}(C \sim D_i) = (C \sim (C \sim (\sqcap_{i \in I}(C \sim D_i))))$.

Proof. The proofs of these algebraic properties can be found in Frink [1962] and Grätzer [1978]. Varlet [1963] also contains a survey on the above standard algebraic properties of pseudocomplement. In particular, the proof of (a), (b), and (c) can be found in the proof of Glivenko's Theorem [Grätzer 1978, Theorem 4, p. 49]. Their proof can also be found by observing that they correspond precisely to Eqs. (8), (9), and (11) in Frink [1962], respectively. The proof of (e), (g), (i), and (j) can be found in Frink [1962], corresponding to Eqs. (10), (15), (18), and (19), respectively. Part (d) is immediate by definition of pseudocomplement, while (f) is immediate from (a). Parts (h) and (k) can be derived as corollaries of Glivenko's Theorem [Grätzer 1978, Theorem 4, p. 49]. Also, (h) corresponds precisely to Eq. (17) in Frink [1962]. □

There exists a wide class of abstract domains for which we can always compute the complement (cf. Corollary 3.7). Indeed, the overwhelming majority of the abstract domains used as the 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, meet-continuity can be checked for it. As a remarkable example, we show later in Section 6 that the abstract lattice of intervals of integer numbers introduced in Cousot and Cousot [1977] to analyze the values of an integer variable does not satisfy the ascending chain condition and is not distributive, but it is meet-continuous.

Example 3.12. Let us consider the typical example of the rule of signs [Cousot and Cousot 1977; 1979] given by the lattice C depicted below. The concrete domain is $\wp(\mathbb{Z})$ (ordered with set-theoretic inclusion $\subseteq$). The concretization and abstraction maps are the obvious ones.



It is easy to verify that all the possible abstractions of this domain, i.e., all the closures on C, are the following:

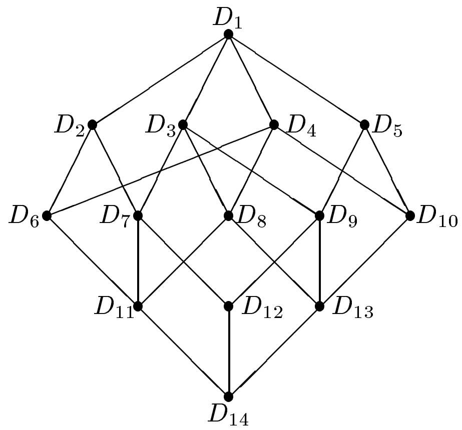
$$D_1 = \{\top\},\ D_2 = \{\top, \geq 0\},\ D_3 = \{\top, 0\},\ D_4 = \{\top, \perp\},\ D_5 = \{\top, \leq 0\},$$

$$D_6 = \{\top, \geq 0, \perp\},\ D_7 = \{\top, \geq 0, 0\},\ D_8 = \{\top, 0, \perp\},\ D_9 = \{\top, \leq 0, 0\},$$

$$D_{10} = \{\top, \leq 0, \perp\},\ D_{11} = \{\top, \geq 0, 0, \perp\},\ D_{12} = \{\top, \leq 0, \geq 0, 0\},$$

$$D_{13} = \{\top, \leq 0, 0, \perp\},\ D_{14} = D.$$

Since C is a finite lattice, by Corollary 3.7 $uco(C)$ is a pseudocomplemented lattice. In fact, $uco(C)$ is the lattice depicted below, and it is simple to verify the pseudocomplementation of $uco(C)$ straight from its Hasse diagram.

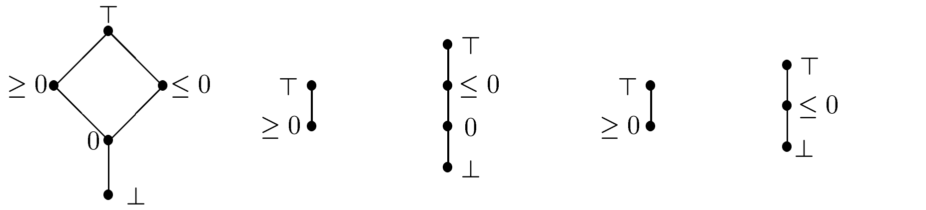


Indeed, the pseudocomplements, i.e., the complements relative to C of the above abstractions, are as follows.

$$D_1^* = D_{14},\ D_2^* = D_{10},\ D_3^* = D_{14},\ D_4^* = D_{12},\ D_5^* = D_6,\ D_6^* = D_5,\ D_7^* = D_{10},$$

$$D_8^* = D_{12},\ D_9^* = D_6,\ D_{10}^* = D_2,\ D_{11}^* = D_5,\ D_{12}^* = D_4,\ D_{13}^* = D_2,\ D_{14}^* = D_1.$$

This example shows how complementation can work for domain decomposition. Suppose C has been incrementally designed by reduced product of the domains D_2 and D_{13}. The complement $(D_2 \sqcap D_{13}) \sim D_2$ is just the domain corresponding to the closure $D_2^* = D_{10}$. In this case, $(D_2 \sqcap D_{13}) \sim D_2$ is a strict abstraction of D_{13}. It is worth noting that we can safely substitute D_{13} by $(D_2 \sqcap D_{13}) \sim D_{13}$, getting a more concise representation of the product (see the figure below).



$$C \quad = \quad D_2 \quad \sqcap \quad D_{13} \quad = \quad D_2 \quad \sqcap\ ((D_2 \sqcap D_{13}) \sim D_2)$$

Domain decomposition by complementation will be considered later in Section 4.

3.1 Construction of Complements

In this section, we present a method to generate the complement $C \sim D$, when C is a finite set. We will build $C \sim D$ as the saturation point of an increasing sequence of closure operators.

In the following, given a complete lattice $\langle C, \preceq \rangle$, and $X \subseteq C$, we will denote by $maxs(X)$ the set of maximal elements of X, viz., the objects $x \in X$ such that $\forall y \in X.\ x \preceq y \Rightarrow x = y$, and by $Cl(X)$ the greatest (w.r.t. $\sqsubseteq$) closure operator containing X. Observe that, if X, Y are (sets of fixpoints of) closure operators on C, then $Cl(X \cup Y) = X \sqcap Y = \{x \wedge y \mid x \in X \text{ and } y \in Y\}$, and that $Cl(X)$ can be computed by adding to X all the greatest lower bounds of subsets of X.

Let $\{X_n\}_{n\in\mathbb{N}}$ be the following family:

$$\begin{aligned}
X_0 &= \{\top\} \\
X_1 &= Cl(X_0 \cup maxs(C \setminus Cl(X_0 \cup D))) \\
&\ \vdots \\
X_n &= Cl(X_{n-1} \cup maxs(C \setminus Cl(X_{n-1} \cup D))) \\
&\ \vdots
\end{aligned}$$

The following proposition shows some properties of X_n, for $n \in \mathbb{N}$.

Proposition 3.1.1. *In the family $\{X_n\}_{n\in\mathbb{N}}$ defined as above, we have that*

(1) $\forall n \in \mathbb{N}.\ X_n \in uco(C)$,

(2) $\forall n \in \mathbb{N}.\ X_n \subseteq X_{n+1}$,

(3) $\forall n \in \mathbb{N}. \forall Y \in uco(C). (Y \sqcap D = C \Rightarrow X_n \subseteq Y)$.

PROOF. Points (1) and (2) are obvious by definition. Let us now prove (3), by induction with respect to n.

$(n = 0)$. Obvious.

$(n > 0)$. Assume $Y \in uco(C)$ and $Y \sqcap D = C$. Let $x \in X_n$. Then there exists $X \subseteq X_{n-1} \cup maxs(C \setminus Cl(X_{n-1} \cup D))$ such that $x = \wedge X$. Let $X' \subseteq X_{n-1}$, $X'' \subseteq maxs(C \setminus Cl(X_{n-1} \cup D))$ such that $X = X' \cup X''$. By inductive hypothesis, $X' \subseteq Y$. It is therefore sufficient to show that $X'' \subseteq Y$ and then use the fact that Y is closed with respect to the operation of greatest lower bound. Consider $z \in X''$. Since $Y \sqcap D = C$, there exist $y \in Y, y' \in D$ such that $z = y \wedge y'$. It turns out that $y \notin Cl(X_{n-1} \cup D)$, as otherwise we would have $z \in Cl(X_{n-1} \cup D)$. Hence, $y \preceq z$, since z is a maximal of the complement of the set $Cl(X_{n-1} \cup D)$. But from $z = y \wedge y'$, it follows also that $z \preceq y$; hence $z = y \in Y$.

This concludes the proof. □

If C is finite, then from the above proposition, we derive that the sequence $\{X_n\}_{n \in \mathbb{N}}$ converges in a finite number of steps to the complement $C \sim D$.

COROLLARY 3.1.2. *If C is finite, then there exists $\bar{n}$ such that $X_{\bar{n}+1} = X_{\bar{n}}$ and $X_{\bar{n}} = C \sim D$.*

PROOF. From Proposition 3.1.1, point (2), we have that $X_{\bar{n}+1} = X_{\bar{n}}$ for some $\bar{n} \leq |C|$. By definition of $X_{\bar{n}+1}$, this implies that $Cl(X_{\bar{n}} \cup D) = C$, i.e., $X_{\bar{n}} \sqcap D = C$. The rest follows from Proposition 3.1.1, points (1) and (3). □

The above corollary constitutes a proof of Theorem 3.6, in the finite case. If C is infinite, then the construction can be extended so as to obtain a transfinite sequence, by defining $X_\alpha = Cl(X_{\alpha-1} \cup maxs(C \setminus Cl(X_{\alpha-1} \cup D)))$ for α successor ordinal, and $X_\alpha = Cl(\bigcup_{\beta<\alpha} X_\beta)$, for α limit ordinal. It is possible to show that $X_{\bar{\alpha}} = C \sim D$, where $\bar{\alpha}$ is the smallest ordinal such that $|\bar{\alpha}| > |C|$. For the detailed (nonconstructive) proof of Theorem 3.6, however, we refer to Giacobazzi et al. [1996].

Example 3.1.3. Consider the domains C and D_2 of Example 3.12. Let us compute the complement $C \sim D_2$ using the method described above. It is simple to verify that the sequence of all the X_n is as follows: $X_0 = \{\top\}$, $X_1 = \{\top, \leq 0\}$, $X_2 = \{\top, \leq 0, \bot\}$, $X_3 = X_2$. Thus, the construction of the X_n generates the complement in three steps.

4. COMPLEMENTS TO DECOMPOSE ABSTRACT DOMAINS

Often, abstract domains for analysis are incrementally designed using reduced product of simpler domains (e.g., in logic program analysis see Codish et al. [1995], Muthukumar and Hermenegildo [1991], and Sundararajan and Conery [1992]). This introduces modularity in domain design, which is helpful both to design domain-dependent abstract operations and to simplify proofs of properties for complex domains for analysis. The inverse operation of *domain decomposition* does not exist in the standard theory of abstract interpretation. This would be clearly helpful to achieve modularity from domains which are not originally designed as reduced product.

Definition 4.1. A (*conjunctive*) *decomposition* for a domain D is any tuple of domains $\langle D_i \rangle_{i \in I}$ such that $D = \sqcap_{i \in I} D_i$.

Obviously, with $\sqcap$ being the greatest lower bound operation on domains, 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 trivial decompositions always exist. In fact, for any domain D, $\langle D \rangle$ and the pair $\langle D, \top \rangle$ (and $\langle \top, D \rangle$) are evidently (straightforward) decompositions of D.

Example 4.2. Consider Example 3.12. $\langle D_1, D_2 \rangle$ and $\langle D_1, C \sim D_1 \rangle$ are both binary decompositions for C.

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 products of more abstract domains. The case of the domain *Sharing*, designed by Jacobs and Langen [1989; 1992] for aliasing analysis of logic programs, 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 8.

It is well known (cf. Ward [1942]) that $uco(C)$, for any complete lattice C, is dual-atomistic. Recall that a complete lattice L is (*dual-*) *atomistic* if every element of L is the join (meet) of the (dual-)atoms in L preceding (following) it. A (*dual-*)*atom* is a nonbottom (top) element $a \in L$ such that for any $b \in L$, $\perp \prec b \preceq a \;\Rightarrow\; a = b$ ($a \preceq b \prec \top \;\Rightarrow\; a = b$). Hence, each closure which is different from the top closure $\lambda x.\top$ is the infimum of the set of dual-atoms following it. It is therefore always possible to decompose any abstract domain in terms of a (possibly infinite) family of *basic domains*. If C is a complete lattice, we call *basic domains* those abstract domains which correspond to dual-atoms in $uco(C)$. The dual-atoms in $uco(C)$ are all and only those closures φ_x, for $x \in C$, with $x \neq \top$, such that $\varphi_x(C) = \{x, \top\}$ [Ward 1942], namely the functions

$$\varphi_x(y) = \begin{cases} x & \text{if } y \preceq_C x; \\ \top & \text{otherwise.} \end{cases}$$

Therefore, the basic abstract domains for a concrete domain C are all and only the two-element domains $\{\top, x\}$, for $x \in C \setminus \{\top\}$. This decomposition is straightforward, and corresponds, in most cases, to enumerating the elements of the domain.

Example 4.3. Consider the domain C of Example 3.12. Its basic abstract domains are those associated with the dual-atoms D_2, D_3, D_4, and D_5. However, it is immediate to verify that $\langle D_2, D_4, D_5 \rangle$ is a decomposition for C as well, i.e., $C = D_2 \sqcap D_4 \sqcap D_5$, involving only three dual-atoms. The corresponding basic domains are the two-point lattices $\{\top, \geq 0\}$, $\{\top, \leq 0\}$, and $\{\top, \perp\}$, respectively.

The following definition introduces a natural preordering relation between decompositions of a given domain.

Definition 4.4. If $\langle D_i \rangle_{i \in I}$ and $\langle D_j \rangle_{j \in J}$ are decompositions of D, then $\langle D_i \rangle_{i \in I}$ is *space better* than $\langle D_j \rangle_{j \in J}$ if $\sum_{i \in I} |D_i| \leq \sum_{j \in J} |D_j|$.

Thus, the above definition allows to compare decompositions with respect to their "space complexity." This definition gives rise to space minimal decompositions, where a decomposition $\langle D_i \rangle_{i \in I}$ for a domain D is *space minimal* if D has no decomposition strictly space better than $\langle D_i \rangle_{i \in I}$.

On the other hand, it is possible to compare decompositions of fixed length in a pointwise manner, exploiting the order of precision between abstract domains.

Definition 4.5. If $\langle D_i \rangle_{i\in I}$ and $\langle E_i \rangle_{i\in I}$ are decompositions of D (of the same length), then $\langle D_i \rangle_{i\in I}$ is *better* than $\langle E_i \rangle_{i\in I}$ if $\forall i \in I.\ E_i \sqsubseteq D_i$.

It is immediate to observe that this definition actually induces a partial ordering. However, least decompositions (w.r.t. this order) of a given fixed length in general do not exist. In fact, for any domain D, with $|D| > 1$, $\langle D, \top \rangle$ and $\langle \top, D \rangle$ are uncomparable minimal binary decompositions of D. Moreover, as observed in Filé and Ranzato [1996], it is simple to verify that $\langle D_i \rangle_{i\in I}$ is a minimal decomposition of D if for all $k \in I$, $(D_k \sqsubset E_k) \Rightarrow (C \sqsubset (\sqcap_{i\in I\setminus\{k\}} D_i) \sqcap E_k)$.

As the following straightforward lemma states, complementation naturally induces minimal decompositions.

LEMMA 4.6 [FILÉ AND RANZATO 1996]. *$\langle D_i \rangle_{i\in I}$ is a decomposition of D such that for any $k \in I$, $D \sim (\sqcap_{i\in I\setminus\{k\}} D_i) = D_k$ iff it is minimal.*

In particular, note that if $D \sqsubseteq E$ then, using the above lemma and Proposition 3.11 part (c), $\langle D \sim E, D \sim (D \sim E)\rangle$ always yields a minimal binary decomposition for D. The role played by complementation for the decompositions is particularly important whenever we deal with an abstract domain built by reduced product of simpler domains. Suppose that $D = \sqcap_{i\in I} D_i$ and $|I| = n$. Then, complementation allows to improve on the representation of D. In fact, for any $k \in I$,

$$\langle D_1, \ldots, D_{k-1}, D \sim (\sqcap_{i\in I\setminus\{k\}} D_i), D_{k+1}, \ldots, D_n \rangle \qquad (*)$$

is a decomposition both better and space better than the starting one $\langle D_i \rangle_{i\in I}$. Moreover, each one of these decompositions can be further improved by iterating this process, applying complementation to the other components, until a minimal decomposition is reached. In this case, we get a minimal decomposition if, by iterating this kind of application of the complementation, we get a domain decomposition where each component has been considered in at least one step of $(*)$. The following proposition proves the correctness of the algorithm in Figure 1. This algorithm is nondeterministic, since the function *choose* selects an arbitrary element from an input set.

PROPOSITION 4.7. *The algorithm in Figure 1 correctly computes a minimal decomposition for D.*

PROOF. It is immediate to observe that the algorithm terminates in n steps. Let $I = \{1, ..., n\}$, and assume that $\mathbf{D} = \langle A_1, ..., A_n \rangle$ is the decomposition given as output by the algorithm. Note that for any $i \in I$, we have that $A_i = D \sim E_i$, for a suitable domain E_i, since, for each component of the input decomposition, a step $(*)$ has been applied. Note also that, by construction, for any $j \in I$, $D \sim (\sqcap_{i\in I\setminus\{j\}} D_i) \sqsubseteq D \sim E_j$. By Lemma 4.6, it is enough to verify that for any $k \in I$, $D \sim (\sqcap_{i\in I\setminus\{k\}} D \sim E_i) = D \sim E_k$. Obviously, for any $i \in I$, $D_i \sqsubseteq D \sim E_i$. Then, with fixed $k \in I$, we have that $\sqcap_{i\in I\setminus\{k\}} D_i \sqsubseteq \sqcap_{i\in I\setminus\{k\}} D \sim E_i$. Then, by Proposition 3.11 part (b), we get $D \sim (\sqcap_{i\in I\setminus\{k\}} D \sim E_i) \sqsubseteq D \sim (\sqcap_{i\in I\setminus\{k\}} D_i) \sqsubseteq D \sim E_k$. On the other hand, $D \sim E_k \sqsubseteq D \sim (\sqcap_{i\in I\setminus\{k\}} D \sim E_i)$ evidently holds, hence concluding the proof. $\square$

Input: A decomposition for D as an array $\mathbf{D}$ of n domains.
Output: A minimal decomposition for D in $\mathbf{D}$.

```
begin
 J := {1,...,n};
 repeat
    k := choose(J);
    J := J \ {k};
    A := ⊤;
    for i := 1 to n do
       if i ≠ k then A := A ⊓ D[i];
    D[k] := D ∼ A
 until J = ∅
end
```

Fig. 1. An algorithm for minimal decompositions.

For instance, if $D = D_1 \sqcap D_2 \sqcap D_3$ then by calling $E = D \sim (D_1 \sqcap D_2)$,

$$\langle D \sim ((D \sim (D_1 \sqcap E)) \sqcap E), D \sim (D_1 \sqcap E), E \rangle$$

is a minimal decomposition both better and space better than $\langle D_1, D_2, D_3 \rangle$. Clearly, if $D = \sqcap_{i \in I} D_i$ and $|I| = n$, then by this process we can get at most $n!$ different minimal decompositions of D, all better and space better than the starting one. In the binary case, if we start from a decomposition $\langle D_1, D_2 \rangle$ of D then we get the two minimal decompositions: $\langle D \sim D_2, D \sim (D \sim D_2) \rangle$ and $\langle D \sim (D \sim D_1), D \sim D_1 \rangle$.

5. COMPOSITIONAL VERIFICATION OF DOMAIN PROPERTIES

One of the advantages of abstract domain decompositions lies in checking properties of abstract domains *compositionally*, by checking them on domain factors. Instead of proving properties for general domains, one can prove properties for more abstract (and simple) factors, provided that these properties are preserved under composition, which is in our case reduced product.

In the following sections, we consider two examples of domain properties which are particularly important in abstract interpretation and which can be verified on the factors of domain decompositions. This may greatly simplify their test for complex domains.

5.1 Dual-Atomicity

Atomicity and dual-atomicity are important lattice-theoretic properties which can allow efficient domain implementation. The intuition behind atomicity in abstract domains is simple: atoms represent *primitive* properties for program analysis.[1] (Dual-)atomistic domains can therefore be generated by considering only their (dual-)atoms. It turns out that dual-atomicity can be verified on the factors of domain decomposition. We need first some preliminary definitions. Given a complete lattice L, we say that two subsets $A, B \subseteq L$ are not comparable in L if for any $a \in A, b \in B$, $a \not\leq b$ and $a \not\geq b$. A closure operator (or equivalently an abstract

[1]This observation, in the context of program analysis, has been done first by Nielson [1985].

domain) is (dual-)atomistic if the set of its fixpoints is a (dual-)atomistic lattice. We denote by $DAtom(L)$ the set of dual-atoms of L.

PROPOSITION 5.1.1. *Let $\{\rho_i\}_{i\in I}$ be a set of dual-atomistic closure operators on a complete lattice L, such that $DAtom(\rho_i)$ and $DAtom(\rho_j)$ are not comparable in L, for any $i, j \in I$ with $i \neq j$. If $K \subseteq I$ then $\sqcap_{k\in K}\rho_k$ is dual-atomistic.*

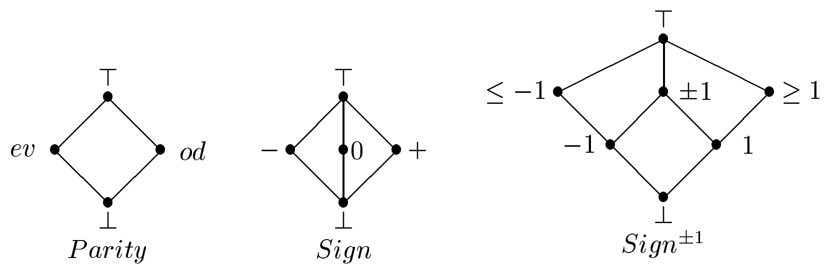
PROOF. Trivial if $K = \emptyset$. Thus, assume that $\emptyset \neq K \subseteq I$. We observe first that any element $x \in \sqcap_{k\in K}\rho_k$ is such that there exists $Y \subseteq \cup_{k\in K} DAtom(\rho_k)$ for which $x = \wedge Y$. This is immediate because by definition of reduced product $x = \wedge_{k\in K} y_k$, where, for any $k \in K$, $y_k \in \rho_k$. Therefore, because $\{\rho_k\}_{k\in K}$ is a set of dual-atomistic closure operators, we have that for any $k \in K$, there exist $Y_k \subseteq DAtom(\rho_k)$ such that $y_k = \wedge Y_k$. Hence, $x = \wedge_{k\in K}(\wedge Y_k)$ for some $Y_k \subseteq DAtom(\rho_k)$, and $k \in K$. Thus, in order to conclude, we have to prove that $\cup_{k\in K} DAtom(\rho_k) = DAtom(\sqcap_{k\in K}\rho_k)$.

($\subseteq$). Assume that $x \in DAtom(\rho_j)$ for some $j \in K$, but $x \notin DAtom(\sqcap_{k\in K}\rho_k)$. By definition of dual-atom, there exists $y \in \sqcap_{k\in K}\rho_k$, $y \neq \top$, such that $x \prec y$. Because any $y \in \sqcap_{k\in K}\rho_k$ can be generated as *glb* of elements in $\cup_{k\in K} DAtom(\rho_k)$, then there exist $k \in K$ and $z \in DAtom(\rho_k)$ (with $z \neq \top$), such that $y \preceq z$. Clearly $k \neq j$; otherwise x cannot be a dual-atom in ρ_j. Moreover, if $k \neq j$ then we also have a contradiction because, by hypothesis, $DAtom(\rho_k)$ and $DAtom(\rho_j)$ are not comparable sets, and therefore $x \not\prec z$.

($\supseteq$). Assume that $x \in DAtom(\sqcap_{k\in K}\rho_k)$, but $x \notin \cup_{k\in K} DAtom(\rho_k)$. Since x can be generated as *glb* of elements in $\cup_{k\in K} DAtom(\rho_k)$, there exist $k \in K$ and $y \in DAtom(\rho_k)$ (with $y \neq \top$), such that $x \prec y$. For any $k \in K$, $DAtom(\rho_k) \subseteq \sqcap_{k\in K}\rho_k$, and therefore $y \in \sqcap_{k\in K}\rho_k$, which contradicts the fact that x was an atom.

This concludes the proof. □

Let us now consider an example. Consider the domains for *parity*, *sign*, and ± 1-*interval* analysis, respectively $Parity$, $Sign$, and $Sign^{\pm 1}$, for data-flow analysis of integer variables, depicted below. In $Sign^{\pm 1}$, which is a strict abstraction of the Cousot and Cousot [1976; 1977] lattice of intervals later considered in Section 6, ± 1 represents the closed interval $[-1, 1]$, while ≤ -1 and ≥ 1 represent the intervals $(-\infty, -1]$ and $[1, +\infty)$, respectively. In the abstract domain $Parity$, od, and ev represent the set of odd and even numbers, respectively. The concrete domain is $\langle \wp(\mathbb{Z}), \subseteq \rangle$. By these identifications, the concretization and abstraction maps for these domains are the obvious ones.



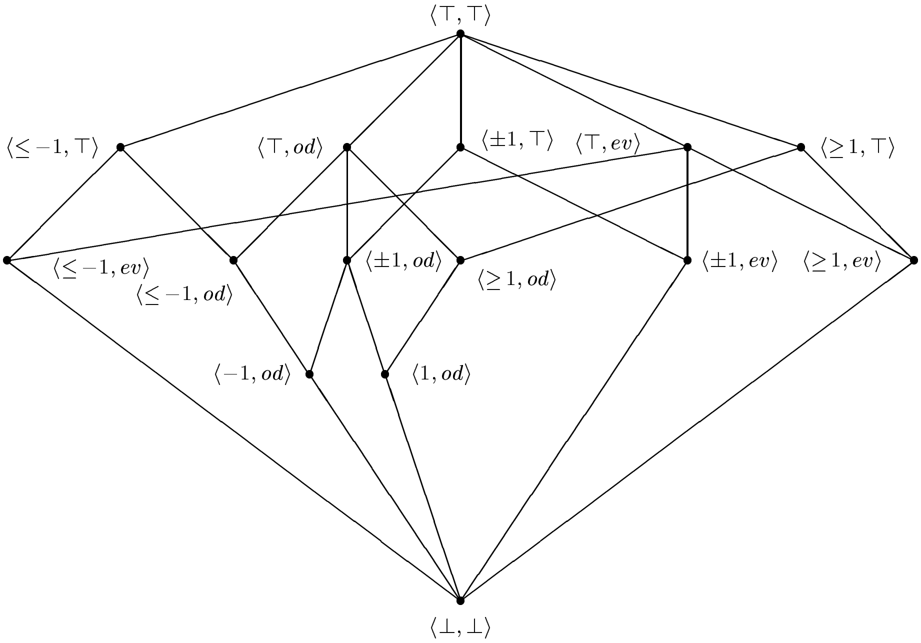


Fig. 2. $Sign^{\pm 1} \sqcap Parity$.

$Sign^{\pm 1}$ and $Parity$ are dual-atomistic lattices, where both $DAtom(Sign^{\pm 1}) = \{\leq -1, \pm 1, \geq 1\}$ and $DAtom(Parity) = \{od, ev\}$ are not comparable in $\wp(\mathbb{Z})$. Therefore, by Proposition 5.1.1, the reduced product $Sign^{\pm 1} \sqcap Parity$ depicted in Figure 2 is a dual-atomistic lattice. It is worth noting that the reduced product $Sign^{\pm 1} \sqcap Parity$ provides also a characterization for 0, which is $\langle \pm 1, ev \rangle$. Because of dual-atomicity, any element in $Sign^{\pm 1} \sqcap Parity$ can be represented as a set of atoms in $DAtom(Sign^{\pm 1}) \cup DAtom(Parity) = \{\leq -1, \pm 1, \geq 1, ev, od\}$. For example, $\langle +1, od \rangle$ can be equivalently represented by the dual-atoms $\{\pm 1, \geq 1\}$.

The condition in Proposition 5.1.1, specifying that dual-atoms of a conjunctive decomposition have to be uncomparable, is essential to maintain dual-atomicity by reduced product. Consider the domains $Sign$ and $Parity$ above. The domain $Sign$ is different from that in Example 3.12, as it does not characterize possibly null or negative (positive) values. Note that while $Sign$ is clearly both atomistic and dual-atomistic, the domain C in Example 3.12 is neither atomistic nor dual-atomistic. Note however that the sets of dual-atoms of $Sign$ and $Parity$ contain comparable elements in $\wp(\mathbb{Z})$. In this case, $DAtom(Sign) = \{+, 0, -\}$ and $DAtom(Parity) = \{ev, od\}$, and 0 is contained in ev. Indeed, the reduced product $Sign \sqcap Parity$, which is depicted in Figure 3, is not dual-atomistic, because $\langle 0, ev \rangle$ cannot be constructed as meet of dual-atoms in $DAtom(Sign \sqcap Parity) = \{\langle -, \top \rangle, \langle \top, od \rangle, \langle +, \top \rangle, \langle \top, ev \rangle\}$.

5.2 Domain Completeness

Traditionally, abstract interpretation is intended to create *sound* approximations of concrete semantics (cf. Cousot and Cousot [1977]). If C is a concrete domain, and

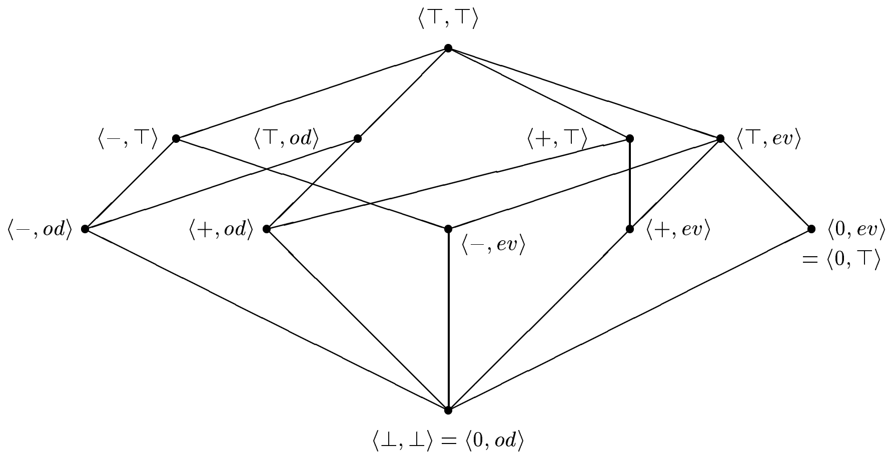


Fig. 3. $Sign \sqcap Parity$.

$T_P : C \to C$ is a monotone semantic operation for a program P, then the soundness criterion for an abstraction given by a G.i. (γ, D, C, α), and an abstract semantic operator $T_P^\sharp : D \to D$, is $\alpha \circ T_P \preceq_D T_P^\sharp \circ \alpha$. This ensures that the least-fixpoint abstract semantics $lfp(T_P^\sharp)$ approximates $lfp(T_P)$, i.e., $\alpha(lfp(T_P)) \preceq_D lfp(T_P^\sharp)$ (cf. Cousot and Cousot [1977]). *Completeness* is the dual relation $T_P^\sharp \circ \alpha \preceq_D \alpha \circ T_P$. Because soundness is required in most abstract interpretations, in the following we abuse terminology and say that (γ, D, C, α) and $T_P^\sharp$ are complete w.r.t. T_P if $\alpha \circ T_P = T_P^\sharp \circ \alpha$.

Completeness is recurrent in the relations between (concrete) semantics of programming languages at different levels of abstraction (cf. Comini and Levi [1994], Cousot and Cousot [1992b], and Giacobazzi [1996]). In the context of program analysis, it has been studied by Cousot and Cousot [1977], Mycroft [1993], and Cortesi et al. [1996]. This condition ensures that $\alpha(lfp(T_P)) = lfp(T_P^\sharp)$. Hence, in analysis, complete abstract interpretations represent, in a sense, an ideal situation, where no loss of precision is introduced in the analysis by using abstract operations.

Completeness can be made a property of domains, by making this notion independent on the choice for $T_P^\sharp$. It is well known [Cousot and Cousot 1977; 1979] that, given a concrete semantic operation T_P, any G.i. naturally defines an abstract semantic operation for T_P, which is its *best correct approximation* in D, viz., $\alpha \circ T_P \circ \gamma$. Hence, because γ is always a 1-1 function in a G.i., and by the correspondence between G.i.'s and closure operators (see Section 2.3), we can define a notion of completeness for closure operators relatively to any monotonic function as follows.

Definition 5.2.1. Let C be a complete lattice and $f : C \to C$ be a monotone function. Then, $\rho \in uco(C)$ is *complete* w.r.t. f if $\rho \circ f = \rho \circ f \circ \rho$.

Example 5.2.2. The lattice C in Example 3.12 for sign analysis is complete with respect to integer multiplication. This proves that the rule of sign is complete (see Mycroft [1993]).

Completeness can be verified compositionally, by verifying it on the factors of a domain decomposition. Let C be a complete lattice, and let $f : C \to C$ be a monotone function. Define $\Gamma(f) = \{\rho \in uco(C) \mid \rho \circ f = \rho \circ f \circ \rho\}$ as the set of complete domains w.r.t. f.

PROPOSITION 5.2.3. *If* $\{\rho_i\}_{i\in I} \subseteq \Gamma(f)$ *then* $\sqcap_{i\in I}\rho_i \in \Gamma(f)$.

PROOF. Let $\{\rho_i\}_{i\in I} \subseteq \Gamma(f)$. Then, by definition, for any $i \in I$ we have $\rho_i \circ f = \rho_i \circ f \circ \rho_i$. Let $x \in C$. Clearly, by extensivity of closure operators, $\wedge_{i\in I}\rho_i(f(x)) \preceq_C \wedge_{i\in I}\rho_i(f(\wedge_{i\in I}\rho_i(x)))$. Moreover, by monotonicity we have $\wedge_{i\in I}\rho_i(f(\wedge_{i\in I}\rho_i(x))) \preceq_C \wedge_{i\in I}\rho_i(f(\rho_i(x)))$. By completeness, we have $\wedge_{i\in I}\rho_i(f(\rho_i(x))) = \wedge_{i\in I}\rho_i(f(x))$. Hence, $\wedge_{i\in I}\rho_i(f(x)) = \wedge_{i\in I}\rho_i(f(\wedge_{i\in I}\rho_i(x)))$, which concludes the proof. $\square$

Clearly, both the straightforward abstractions given by the identity closure C and the top closure $\{\top_C\}$ are complete. Hence, $\Gamma(f) \in uco(uco(C))$.

6. IMPERATIVE PROGRAMMING: DECOMPOSING INTEGER INTERVAL DOMAIN

In this section, we apply complementation to some abstractions of the standard lattice of integer intervals, introduced by Cousot and Cousot [1976; 1977] as an abstract domain for data-flow analysis of (imperative) programs with variables assuming integer values. The lattice of integer intervals is particularly important because it provides a typical example of abstract domain for analysis which is meet-continuous, but it is neither distributive nor does it satisfy the ascending chain condition.

6.1 Domains for Integer Variable Analysis

For simplicity, we consider a single integer variable to analyze (the generalization is straightforward), and therefore the domain of concrete denotations is the powerset of the integers, $\wp(\mathbb{Z})$, ordered by subset inclusion.

The abstract domain I of integer intervals is defined as follows:

$$\begin{aligned} I \;=\; & \{[a,b] \mid a,b \in \mathbb{Z}, a \leq b\} \cup \{(-\infty,b] \mid b \in \mathbb{Z}\} \cup \\ & \{[a,+\infty) \mid a \in \mathbb{Z}\} \cup \{(-\infty,+\infty)\} \cup \{\bot\}, \end{aligned}$$

where the ordering $\leq$ on $\mathbb{Z}$ is the natural ordering on the integer numbers. The ordering relation $\leq_I$ on intervals is defined as follows:

—for any $x \in I$, $\bot \leq_I x \leq_I (-\infty,+\infty)$,
—for any $a,b,c,d \in \mathbb{Z}$, $[a,b] \leq_I [c,d] \Leftrightarrow c \leq a \;\;\&\;\; b \leq d$,
—for any $b,d \in \mathbb{Z}$, $(-\infty,b] \leq_I (-\infty,d] \Leftrightarrow b \leq d$,
—for any $a,b,d \in \mathbb{Z}$, $[a,b] \leq_I (-\infty,d] \Leftrightarrow b \leq d$,
—for any $a,c \in \mathbb{Z}$, $[a,+\infty) \leq_I [c,+\infty) \Leftrightarrow c \leq a$,
—for any $a,b,c \in \mathbb{Z}$, $[a,b] \leq_I [c,+\infty) \Leftrightarrow c \leq a$.

This domain I is depicted in Figure 4. Notice that the top element in I is the interval $(-\infty,+\infty)$. As pointed out in Cousot and Cousot [1977], it turns out that $\langle I, \leq_I\rangle$ is a complete lattice.

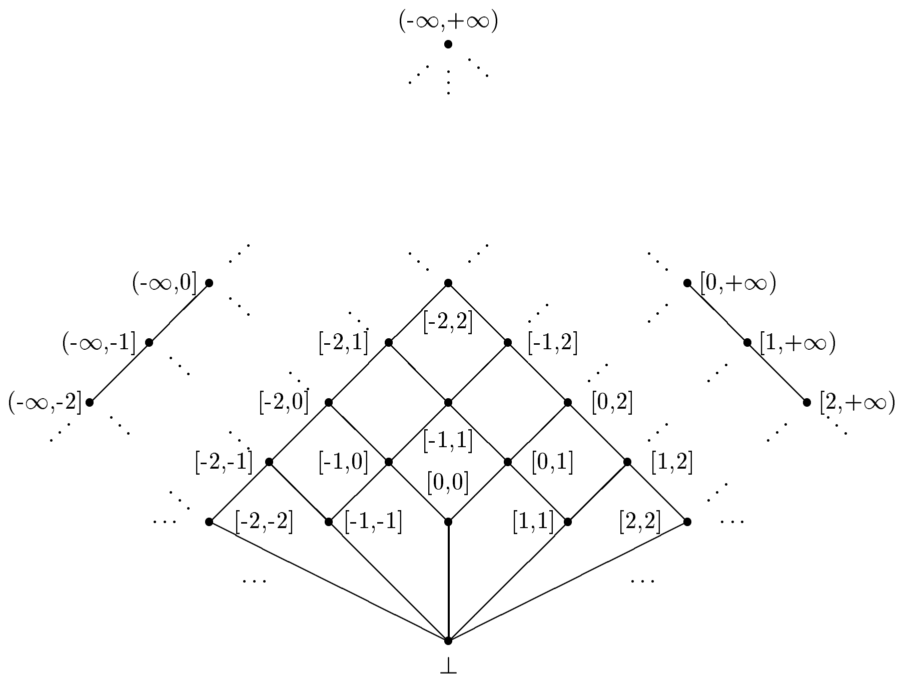


Fig. 4. The abstract domain I.

I enjoys a Galois insertion with $\langle \wp(\mathbb{Z}), \subseteq \rangle$, which is determined by the following mappings:

$$\alpha_I(S) = \begin{cases} \bot & \text{if } S = \emptyset \\ [a, b] & \text{if } min(S) = a \text{ and } max(S) = b \\ (-\infty, b] & \text{if } \nexists min(S) \text{ and } max(S) = b \\ [a, +\infty) & \text{if } min(S) = a \text{ and } \nexists max(S) \\ (-\infty, +\infty) & \text{if } \nexists min(S) \text{ and } \nexists max(S); \end{cases}$$

$$\gamma_I(x) = \begin{cases} \emptyset & \text{if } x = \bot \\ \{z \in \mathbb{Z} \mid a \leq z \leq b\} & \text{if } x = [a, b] \\ \{z \in \mathbb{Z} \mid z \leq b\} & \text{if } x = (-\infty, b] \\ \{z \in \mathbb{Z} \mid a \leq z\} & \text{if } x = [a, +\infty) \\ \mathbb{Z} & \text{if } x = (-\infty, +\infty). \end{cases}$$

Cousot and Cousot [1977] introduced some abstractions of the domain of the intervals. We recall these domains in Figure 5.

An abstraction of the intervals I is given by the domain I_{CS} depicted in Figure 5. This domain is obtained by identifying any integer number z by the interval $[z, z]$, the elements $-$ and $+$ by (-∞,0] and [0,+∞), respectively, and the element $\top$ by (-∞,+∞). It is clear that I_{CS} is an abstraction of I, since it corresponds to a Moore-family of elements of I. I_{CS} is an appropriate domain for *constant propagation* and *sign* analysis.

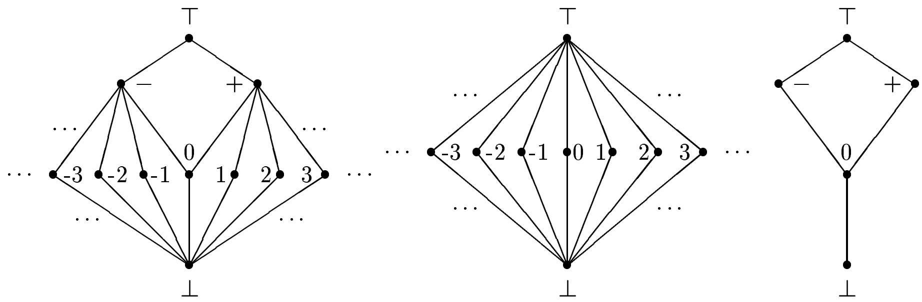


Fig. 5. The abstract domains I_{CS}, I_C, and I_S.

A further abstraction of I_{CS}, and hence of I, is given by the domain I_C also depicted in Figure 5. I_C is clearly a Moore-family of elements of I_{CS} and therefore an abstraction of I_{CS}.

Finally, a third abstraction of I_{CS}, incomparable with I_C, is the domain I_S also given in Figure 5. Also in this case, I_S is obviously a Moore-family of I_{CS} and hence an abstraction of it.

The domain I_C is the standard lattice for *constant propagation* analysis [Kam and Ullman 1977; Kildall 1973], namely used to detect program expressions computing the same value on all executions of the program, while I_S is used for *sign* analysis [Cousot and Cousot 1977]. I_{CS} is an enrichment of both I_C and I_S. Indeed, it is immediate to observe that I_{CS} is exactly the reduced product of I_C and I_S.

LEMMA 6.1.1. $I_{CS} = I_C \sqcap I_S$.

6.2 The Complements

Evidently, the lattice of intervals I does not satisfy the ascending chain condition. Moreover I is not distributive; in fact, $(A \vee B) \wedge C$ differs from $(A \wedge C) \vee (B \wedge C)$, with $A = [-1,-1]$, $B = [-1,0]$, and $C = [1,1]$. Nevertheless, complementation is equally possible, since a simple direct inspection of the Hasse diagram of I reveals that I is meet-continuous, and hence Theorem 3.6 is applicable. Thus, we can compute the three complements $I \sim I_{CS}$, $I \sim I_C$, and $I \sim I_S$. First, we will compute $I \sim I_C$ and $I \sim I_S$, and then $I \sim I_{CS}$ will be a simple consequence of these two.

The complement of constant propagation relative to intervals is somehow surprising; in fact, it turns out that this complement is precisely the lattice of intervals itself.

PROPOSITION 6.2.1. $I \sim I_C = I$.

PROOF. Trivially, $I_C \sqcap I = I$. $I \sim I_C$ must contain the top $(-\infty,+\infty)$. Also, every proper interval (namely, each interval x such that $|\gamma_I(x)| > 1$) must belong to $I \sim I_C$, since the closure on I associated with the abstraction of I_C maps each of them into $\top$. Consequently, $I \sim I_C$ contains clearly $\perp$ and each $[z,z]$, for $z \in \mathbb{Z}$, since every $[z,z]$ is the meet of two proper intervals (viz., $[z,z] = [z-1,z] \wedge [z,z+1]$). Thus, we conclude that $I \sim I_C = I$. □

The complement of the sign domain I_S relative to the interval domain I is given

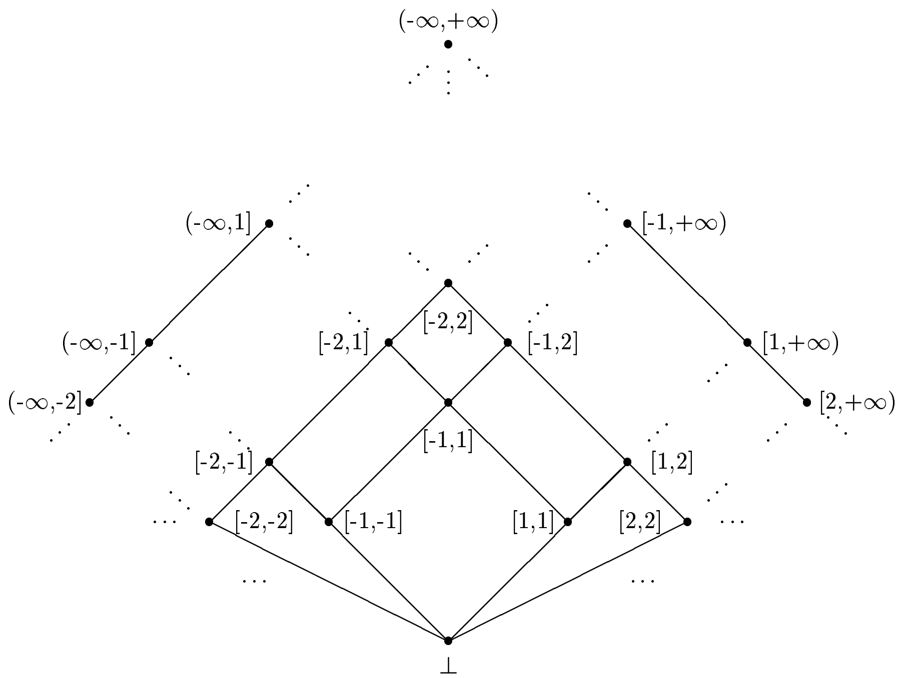


Fig. 6. The complement $I \sim I_S$.

by the Hasse diagram of Figure 6. Thus, the elements of this complement are exactly given by

$$I \sim I_S = I \setminus (\{[z,0] \in I \mid z \in \mathbb{Z}\} \cup \{[0,z] \in I \mid z \in \mathbb{Z}\} \cup \{(-\infty,0],[0,+\infty)\}).$$

Hence, this domain $I \sim I_S$ is not able to represent those intervals having 0 as an extreme. In this sense, $I \sim I_S$ captures interval information but not sign.

Proposition 6.2.2. *$I \sim I_S$ is the domain depicted in Figure 6.*

Proof. First of all, note that the domain of Figure 6, let us call it D, is indeed an abstraction of I, being closed by *glb*. Also, $I_S \sqcap D = I$: in fact, it is enough to observe that each element $[z,0]$, for -$\infty < z \leq$-1, can be obtained by reduced product as $[z,0] = (\text{-}\infty,0] \wedge [z,1]$ (analogously for each $[0,z]$, for $1 \leq z < +\infty$). Finally, D is actually the complement $I \sim I_S$. This follows because each element of D must belong to $I \sim I_S$; otherwise there is no way to recover it by reduced product. ☐

Finally, we compute the complement of the domain I_{CS} relative to the interval lattice I. It turns out that this complement is precisely the above-computed $I \sim I_S$, which is depicted in Figure 6.

Proposition 6.2.3. *$I \sim I_{CS} = I \sim I_S$.*

Proof. By Lemma 6.1.1, we know that $I_{CS} = I_C \sqcap I_S$. Then, exploiting Proposition 3.11 parts (e) and (j) and Proposition 6.2.1, we get the following equalities:

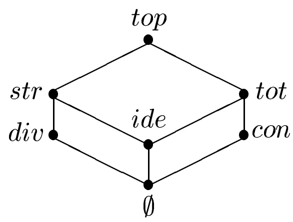
$$\begin{aligned} I \sim I_{CS} &= I \sim (I_C \sqcap I_S) \\ &= I \sim (I \sim ((I \sim I_C) \sqcup (I \sim I_S))) \\ &= I \sim (I \sim (I \sqcup (I \sim I_S))) \\ &= I \sim (I \sim (I \sim (I \sim I_S))) \\ &= I \sim I_S. \end{aligned}$$

This concludes the proof. □

7. FUNCTIONAL PROGRAMMING: DECOMPOSING COMPORTMENTS

In this application, we consider complements relative to the lattice of *comportment analysis*, designed by Cousot and Cousot [1994] in order to generalize Mycroft's *strictness* and *termination* analysis [Mycroft 1980; 1981], Wadler and Hughes' *projection* analysis [Wadler and Hughes 1987], and Hunt's *PER* analysis [Hunt 1990]. This provides a decomposition of the lattice of comportments into sensible factors, which gives a better comprehension of its structure.

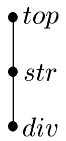
The comportment analysis applies to higher-order monomorphically typed lazy functional programming languages. To illustrate Cousot and Cousot's comportment analysis, we consider abstract interpretation of a simply typed lambda calculus with basic types β. Denote D^τ the domain of values of a type τ, and by $\perp$ its bottom element. For simplicity, we consider abstractions of functional basic types $\beta \to \beta$ (i.e., elements in $D^{\beta\to\beta} = D^\beta \to D^\beta$, the lattice of total continuous 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 basic types $\beta \to \beta$.



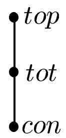
Basic comportments $\mathcal{B}_\mathcal{C}$

The meaning of basic comportments in $\mathcal{B}_\mathcal{C}$ is given in Table I, 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 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 $\mathcal{T}$

Table I. Basic Comportment Analysis $\mathcal{B}_{\mathcal{C}}$

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	$\gamma^{\beta\to\beta}(\emptyset)$	$= \emptyset$

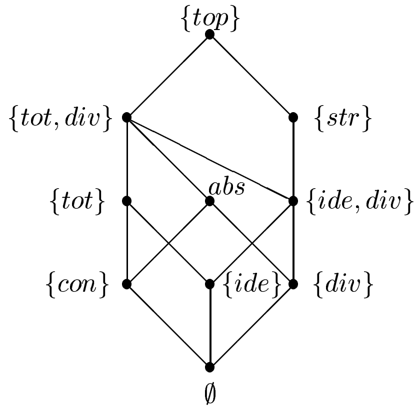
By an inspection of the lattice of basic comportments $\mathcal{B}_{\mathcal{C}}$, we notice that the complement of the strictness (termination) domain relative to $\mathcal{B}_{\mathcal{C}}$ is the termination (strictness) analysis.

PROPOSITION 7.1. *$\mathcal{B}_{\mathcal{C}} \sim \mathcal{S} = \mathcal{T}$ and $\mathcal{B}_{\mathcal{C}} \sim \mathcal{T} = \mathcal{S}$*

PROOF. Note that $\mathcal{B}_{\mathcal{C}}$ is a finite lattice. Hence, by the iterative method in Section 3.1 applied to $\mathcal{B}_{\mathcal{C}} \sim \mathcal{S}$, we get $X_0 = \{top\}$, $X_1 = \{top, tot\}$, and $X_2 = X_3 = \{top, tot, con\} = \mathcal{T}$. The proof for $\mathcal{B}_{\mathcal{C}} \sim \mathcal{T}$ is analogous. $\square$

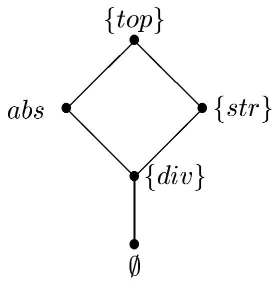
Hence, by Lemma 4.6, $\langle \mathcal{S}, \mathcal{T} \rangle$ is a minimal decomposition for the lattice of basic comportments. In particular, the identity information *ide* as well as $\emptyset$ can be both constructed by conjunction of strictness and termination values (*str* and *tot* for *ide* and *div* and *con* for $\emptyset$). Therefore, the lattice of basic comportments $\mathcal{B}_{\mathcal{C}}$ is precisely the reduced product of strictness and termination. As we will show later on, in this example the identity information will be always definable as the conjunction of factors involving strictness information (i.e., in factorizations of analysis involving strictness, such as strictness or projection analysis), even though the lattice of comportments will be lifted at a powerset level.

As proved by Cousot and Cousot [1994], more precise comportment properties for higher-order functional languages can be characterized by lifting the domain of basic comportments to its disjunctive completion. Disjunctive completion is here used to mimic the collecting semantics construction. Collecting semantics are defined by "collecting" in sets the possible output values corresponding to a given set of possible input values, as defined by the standard semantics of the language. Hence, in order to exploit sets of values, Cousot and Cousot considered a powerset completion of the abstract domain, which corresponds, at the level of abstract domains, to the collecting semantics construction. The abstraction of sets of functions in $D^{\beta\to\beta}$ yields a corresponding abstract domain for comportments which can be systematically derived by reduction of a powerset completion of the lattice of basic comportments $\mathcal{B}_{\mathcal{C}}$. 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 (disjunctive) comportment analysis. It is obtained by (e.g., antichain) powerset completion and reduction (viz., sets of basic comportments denoting the same object in $\wp(D^{\beta\to\beta})$ are identified). The new element *abs* corresponds here to the set of basic comportments $\{con, div\}$ and represents *absence*.

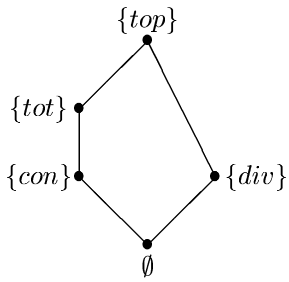


Comportments $\mathcal{C}$

As shown in Cousot and Cousot [1994], this lattice generalizes projection $\mathcal{P}$ and dual-projection $\mathcal{D}_{\mathcal{P}}$ depicted respectively below, as well as the above strictness $\mathcal{S}$ and termination $\mathcal{T}$ analyses (in the latter case, the concretization in $\mathcal{C}$ of an element x is the singleton $\{x\}$).



Projection $\mathcal{P}$



Dual-projection $\mathcal{D}_{\mathcal{P}}$

Some interesting properties of comportment analysis can be studied by looking at the complements of projection, dual-projection, strictness, and termination in $\mathcal{C}$.

PROPOSITION 7.2.

(1) $\mathcal{C} \sim \mathcal{P} = \{\{top\}, \{tot, div\}, \{tot\}\}$,
(2) $\mathcal{C} \sim \mathcal{S} = \{\{top\}, \{tot, div\}, \{tot\}, abs, \{con\}\}$, *and*
(3) $\mathcal{C} \sim \mathcal{T} = \mathcal{C} \sim \mathcal{D}_{\mathcal{P}} = \{\{top\}, \{tot, div\}, \{str\}, abs, \{ide, div\}, \{div\}\}$.

PROOF. We only include the proof for

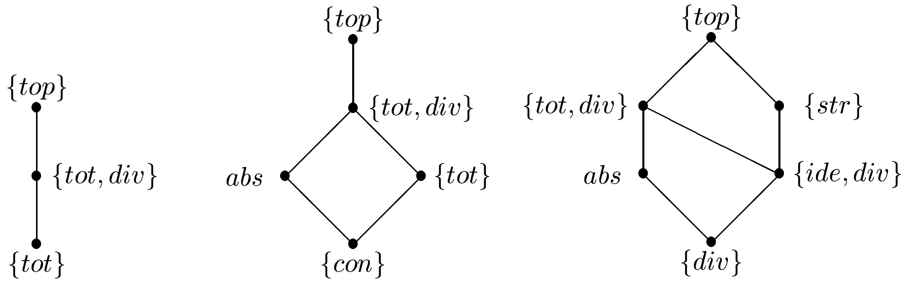
$$\mathcal{C} \sim \mathcal{T} = \mathcal{C} \sim \mathcal{D}_{\mathcal{P}} = \{\{top\}, \{tot, div\}, \{str\}, abs, \{ide, div\}, \{div\}\}$$

since the other proofs are similar. Because $\mathcal{C}$ is a finite domain, by applying the iterative method in Section 3.1, we obtain

—$X_0 = \{\{top\}\}$,
—$X_1 = \{\{top\}, \{tot, div\}, \{str\}, \{ide, div\}\}$, and
—$X_2 = X_3 = \{\{top\}, \{tot, div\}, \{str\}, \{ide, div\}, abs, \{div\}\}$.

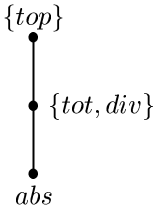
The equality $\mathcal{C} \sim \mathcal{T} = \mathcal{C} \sim \mathcal{D}_{\mathcal{P}}$ is a consequence of the fact that $\{div\}$ is always obtained by combining the maximal elements abs and $\{ide, div\}$ which are both neither included in $\mathcal{T}$, nor in $\mathcal{D}_{\mathcal{P}}$. $\square$

The complements $\mathcal{C} \sim \mathcal{P}$, $\mathcal{C} \sim \mathcal{S}$, and $\mathcal{C} \sim \mathcal{T}$ are depicted below.



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

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 reduced product involving $\mathcal{C} \sim \mathcal{P}$ and $\mathcal{P}$. The identity and convergence information is therefore redundant for the decomposition $\langle \mathcal{P}, \mathcal{C} \sim \mathcal{P} \rangle$. $\mathcal{C} \sim \mathcal{S}$ is a domain for totality analysis. This domain characterizes precisely the nonstrictness comportments. It is worth noting 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. As observed in Proposition 7.2, $\mathcal{C} \sim \mathcal{T} = \mathcal{C} \sim \mathcal{D}_{\mathcal{P}}$. This domain characterizes exactly the nonterminating (or divergent) comportments. Note that both $\langle \mathcal{P}, \mathcal{C} \sim \mathcal{P} \rangle$ and $\langle \mathcal{S}, \mathcal{C} \sim \mathcal{S} \rangle$ provide binary decompositions of the lattice of comportments, which are strictly space better than $\langle \mathcal{T}, \mathcal{C} \sim \mathcal{T} \rangle$. A domain for nonterminating and nonstrictness comportments can be further obtained as the complement of strictness relative to $\mathcal{C} \sim \mathcal{T}$, or equivalently as the complement of termination relative to $\mathcal{C} \sim \mathcal{S}$, i.e., $(\mathcal{C} \sim \mathcal{S}) \sim \mathcal{T} = (\mathcal{C} \sim \mathcal{T}) \sim \mathcal{S} = (\mathcal{C} \sim \mathcal{D}_{\mathcal{P}}) \sim \mathcal{S}$, as depicted below. We omit the proofs for this complement, since it is similar to that in Proposition 7.2.



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

Note that $\langle \mathcal{S}, \mathcal{T}, (\mathcal{C} \sim \mathcal{S}) \sim \mathcal{T} \rangle$ is a decomposition for the lattice of comportments $\mathcal{C}$. Hence, by iterating complementation of these factors, by Proposition 4.7, we get the following minimal decomposition for $\mathcal{C}$.

Proposition 7.3. *$\langle \{\{top\}, \{str\}\}, \{\{top\}, \{tot\}\}, \{\{top\}, \{tot, div\}, abs\} \rangle$ is a minimal decomposition for $\mathcal{C}$.*

Proof. We iterate the application of complementation, as suggested in Proposition 4.7, to the input decomposition $\langle \mathcal{S}, \mathcal{T}, (\mathcal{C} \sim \mathcal{S}) \sim \mathcal{T} \rangle$. It is worth noting that $((\mathcal{C} \sim \mathcal{S}) \sim \mathcal{T}) \sqcap \mathcal{T} = (\mathcal{C} \sim \mathcal{S})$, and because $(\mathcal{C} \sim \mathcal{S}) \sim \mathcal{T} = (\mathcal{C} \sim \mathcal{T}) \sim \mathcal{S}$, then $((\mathcal{C} \sim \mathcal{S}) \sim \mathcal{T}) \sqcap \mathcal{S} = (\mathcal{C} \sim \mathcal{T})$. Moreover, by Proposition 7.1, $\mathcal{S} \sqcap \mathcal{T} = \mathcal{B}_{\mathcal{C}}$. Hence, by applying the iterative method in Section 3.1, we get

$\mathcal{C} \sim (\mathcal{C} \sim \mathcal{S})$. $X_0 = \{\{top\}\}$, $X_1 = \{\{top\}, \{str\}\}$, which is a fixpoint;

$\mathcal{C} \sim (\mathcal{C} \sim \mathcal{T})$. $X_0 = \{\{top\}\}$, $X_1 = \{\{top\}, \{tot\}\}$, which is a fixpoint;

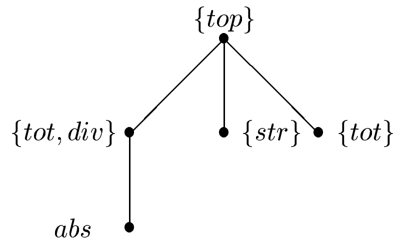
$\mathcal{C} \sim \mathcal{B}_{\mathcal{C}}$. $X_0 = \{\{top\}\}$, $X_1 = \{\{top\}, \{tot, div\}\}$, $X_2 = \{\{top\}, \{tot, div\}, abs\}$ which is a fixpoint. Hence, $\mathcal{C} \sim \mathcal{B}_{\mathcal{C}} = (\mathcal{C} \sim \mathcal{S}) \sim \mathcal{T}$.

The proof follows by Proposition 4.7. □

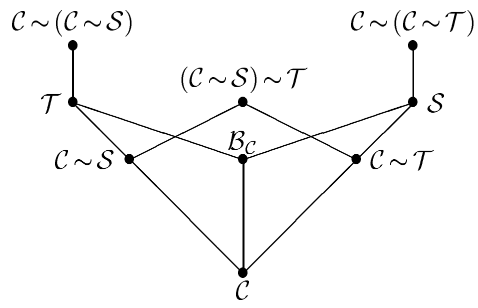
Therefore, the whole domain of comportments $\mathcal{C}$ can be represented, more concisely, by the following decomposition:

$$\langle \{\{top\}, \{str\}\}, \{\{top\}, \{tot\}\}, \{\{top\}, \{tot, div\}, abs\} \rangle$$

which is just a tuple of chains, corresponding to the data structure below.



The above decomposition shows the logical relation between comportment and strictness and termination analysis in abstract interpretation of functional languages. The following diagram summarizes the relation (in $uco(\mathcal{C})$) between some of the complements obtained above, involving strictness and termination only.



As far as dual-projection is concerned, we have shown that the comportment analysis cannot be obtained as the reduced product of projection and dual-projection, because 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 decomposition of the domain of comportments. Finally, the factorization of the domain of basic comportments into the pair $\langle \mathcal{S}, \mathcal{T} \rangle$ proves that the domain of comportments is indeed (isomorphic to) the space of all relations between strictness and termination (at least for the case of the functional basic types). This clarifies the relationship between the domain of comportments and the domains for termination and strictness analysis.

8. LOGIC PROGRAMMING: A DECOMPOSITION FOR *SHARING*

In this section, we apply complementation to the case of *Sharing*, a well-known domain for variable aliasing and groundness analysis of logic programs introduced by Jacobs and Langen [1989; 1992]. In Cortesi et al. [1992], it has been shown that *Sharing* represents, in addition to variable sharing, ground dependency. We prove that *Sharing* enjoys a Galois insertion (not only a Galois connection as proved in Cortesi et al. [1992]) with the domain *Def* [Marriott and Søndergaard 1993]. Further, we investigate what is left of *Sharing* once we remove *Def*, i.e., the complement of *Def* with respect to *Sharing*. This domain, on the one hand, must represent variable independency and sharing and, on the other hand, must disregard ground dependency. We show that such a domain is characterized by a simple closure operator on *Sharing*.

8.1 Notation on Substitutions

Let *Var* be a countable set of variables $x, y, z, \ldots$, and let $\mathcal{A}$ be an alphabet of constant and function symbols. A substitution σ on $(\mathcal{A}, \mathit{Var})$ is a function mapping each $x \in \mathit{Var}$ to a term $\sigma(x)$ built on the variables of *Var* and on the symbols of $\mathcal{A}$, and such that $\sigma(x) \neq x$ holds only for a finite number of variables x. We denote a substitution by the list of its nontrivial bindings, i.e., $\sigma = \{x/\sigma(x) \mid \sigma(x) \neq x\}$. Given a term t, we denote by $var(t)$ the set of variables which occur in t. Given two substitutions σ and θ, the composition $\theta \circ \sigma$ is defined as the substitution which maps each x into $\theta(\sigma(x))$, where $\theta(t)$ is the term obtained from t by replacing each $y \in var(t)$ by $\theta(y)$. A substitution σ is *idempotent* if $\sigma \circ \sigma = \sigma$. We denote by *Subst* the set of idempotent substitutions. It is possible to define the relation $\leq$ of instantiation on *Subst* (actually a preorder) in the usual way: if $\sigma, \theta \in \mathit{Subst}$ then $\sigma \leq \theta$ iff there exists a substitution δ (possibly nonidempotent) such that $\sigma = \delta \circ \theta$. In this case, we say that σ is an *instance* of θ. The set of all instances of σ is denoted by Δ_σ. The concrete domain of computation of a logic program is the set $\wp(\mathit{Subst})$ ordered by set inclusion. In the following, we illustrate the domains *Sharing* and *Def* as abstractions of $\wp(\mathit{Subst})$ with respect to a given finite set of *variables of interest* $\mathit{VI} \subseteq \mathit{Var}$.

8.2 The Domain *Sharing*

The abstract domain *Sharing* is defined as the set

$$\{S \subseteq \wp(\mathit{VI}) \mid S \neq \emptyset \Rightarrow \emptyset \in S\}$$

ordered with respect to set inclusion. For instance, if $\mathit{VI} = \{x, y\}$, then *Sharing* is the domain illustrated in Figure 7.

Jacobs and Langen [1989; 1992] proved that *Sharing* enjoys a Galois insertion into the concrete domain $\wp(\mathit{Subst})$. We recall briefly the construction of the mappings. For $x \in \mathit{Var}$ and $\sigma \in \mathit{Subst}$, let $\mathit{share}(\sigma, x)$ be the set of variables of interest whose images under σ contain the variable x, i.e., $\mathit{share}(\sigma, x) = \{y \in \mathit{VI} \mid x \in var(\sigma(y))\}$.

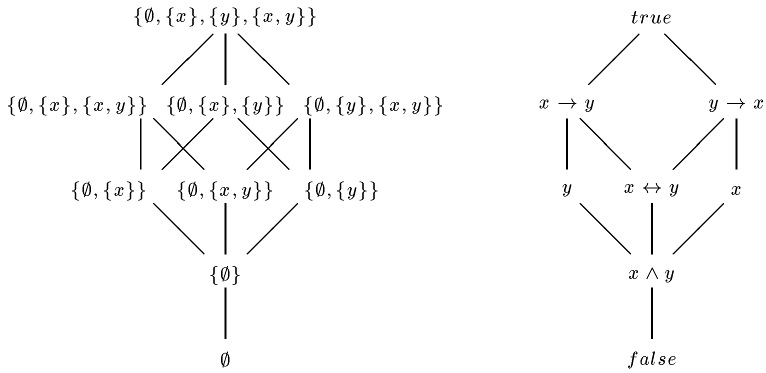


Fig. 7. The domains *Sharing* and *Def* for VI=$\{x, y\}$.

The abstraction and concretization functions between $\wp(Subst)$ and *Sharing* are then defined as follows. For $\Sigma \in \wp(Subst)$ and $S \in Sharing$

$$\begin{array}{rcl} \alpha_{Sharing}(\Sigma) & = & \{share(\sigma, x) \mid \sigma \in \Sigma,\ x \in Var\}, \\ \gamma_{Sharing}(S) & = & \{\sigma \in Subst \mid \alpha_{Sharing}(\{\sigma\}) \subseteq S\}. \end{array}$$

Intuitively, $\alpha_{Sharing}$ extracts the *sharing information* from substitutions, i.e., for a substitution σ, we have that $y_1, y_2, \ldots$ share under σ (i.e., their images under σ share a variable) iff there exists $A \in \alpha_{Sharing}(\{\sigma\})$ such that $y_1, y_2, \ldots \in A$. As a particular case, we obtain also the ground information: y is ground in σ (i.e., its image under σ is a ground term) iff for every $A \in \alpha_{Sharing}(\{\sigma\})$, $y \notin A$.

For example, let $VI = \{x, y, z, w\}$, and consider the set $\{\emptyset, \{w\}, \{y, z, w\}\} \in$ *Sharing*. This set represents the substitutions under which x is ground and y, z, and w may share. In particular, $\sigma_1 = \{x/a, y/b, z/c\}$ and $\sigma_2 = \{x/b, y/v, z/v, w/v\}$ (where a, b, c are constant symbols and where v is a variable) satisfy these properties. Therefore, $\{\sigma_1, \sigma_2\} \subseteq \gamma_{Sharing}(\{\emptyset, \{w\}, \{y, z, w\}\})$.

8.3 The Domain *Def*

The domain *Def* was introduced by Marriott and Søndergaard [1993] to analyze the ground dependencies induced by substitutions on a given set of variables of interest VI. We recall briefly its construction.

Assume that $VI = \{x_1, \ldots, x_n\}$. Given the Boolean domain $B = \{true, false\}$, we say that the Boolean function $f : B^n \rightarrow B$ is *positive* iff $f(true, \ldots, true) = true$. Given an interpretation $m : VI \rightarrow B$, and an n-ary boolean function f, we say that m *is a model* of f (notation $m \models f$) iff $f(m(x_1), \ldots, m(x_n)) = true$. By $Models(f)$ we denote the set of models of a given f. We also say that f *validates* f' (notation $f \models f'$) iff all the models of f are models also of f'. Given two interpretations m_1, m_2, the conjunction $m_1 \wedge m_2$ is the pointwise extension of the logical conjunction, i.e., $\forall x.\ (m_1 \wedge m_2)(x) = m_1(x) \wedge m_2(x)$. We will represent an interpretation also as a set a variables, i.e., the set of elements of VI which are mapped into *true*. For instance, $\{x, y\}$ represents the interpretation which assigns *true* to x and y and *false* to all the other variables of VI. Clearly, in this representation conjunction corresponds to set intersection.

The domain *Def* consists of all positive n-ary Boolean functions whose models are closed under model conjunction, plus the bottom element *false* (the constant function which always returns *false*), ordered by $\models$.

Obviously, n-ary Boolean functions can be represented by means of propositional formulae on VI.[2] It is possible to show that each formula of *Def* is equivalent to *false* or to a conjunction of definite (propositional) clauses. It turns out that *Def* is a (finite) lattice (e.g., Armstrong et al. [1994]), where the *glb* is simply logical conjunction, while the *lub* is $f_1 \vee_{Def} f_2 = \wedge\{f \in Def \mid f_1 \models f \text{ and } f_2 \models f\}$. For $VI = \{x, y\}$, *Def* is depicted in Figure 7.

We now recall the Galois insertion of *Def* into $\wp(Subst)$. For $\sigma \in Subst$ the formula that expresses the ground dependencies of σ is

$$gdep(\sigma) = \exists_{\overline{VI}}.\ \wedge \{x \leftrightarrow \wedge var(\sigma(x)) \mid \sigma(x) \neq x\}$$

where $\exists_{\overline{VI}}$ is the existential quantification over the variables of noninterest, i.e., the variables in $Var \setminus VI$. The interpretation which specifies which variables in VI are bound by σ to ground terms, and which are not, is

$$ground(\sigma) = \{x \in VI \mid var(\sigma(x)) = \emptyset\}.$$

The abstraction and concretization maps are as follows [Marriott and Søndergaard 1993]. For any $\Sigma \in \wp(Subst)$ and $f \in Def$

$$\begin{aligned} \alpha_{Def}(\Sigma) &= \vee_{Def}\{gdep(\sigma) \mid \sigma \in \Sigma\}, \\ \gamma_{Def}(f) &= \{\sigma \in Subst \mid \forall \sigma' \leq \sigma.\ ground(\sigma') \models f\}. \end{aligned}$$

These two mappings form a Galois insertion of *Def* into $\wp(Subst)$ [Armstrong et al. 1994; Marriott and Søndergaard 1993].

As an example, assume $VI{=}\{x, y, z, w\}$. 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 $\sigma_2 = \{x/a, y/v, z/v, w/u\}$ satisfy these properties. Thus, $\{\sigma_1, \sigma_2\} \subseteq \gamma_{Def}(x \wedge (y \leftrightarrow z))$.

The following result relates a substitution with its abstraction in *Def*. It will be useful later.

LEMMA 8.3.1. *For any* $\sigma \in Subst$, $Models(\alpha_{Def}(\{\sigma\})) = \{ground(\sigma') | \sigma' \in \Delta_\sigma\}$.

PROOF. The $\supseteq$ inclusion follows by definition of α_{Def}. For the other direction, we show that for any $m \in Models(\alpha_{Def}(\{\sigma\}))$, there exists a substitution $\sigma_m = \delta \circ \sigma \in \Delta_\sigma$ such that $ground(\sigma_m) = m$. Since $m \models \exists_{\overline{VI}}.\ \wedge \{x \leftrightarrow \wedge var(\sigma(x)) \mid \sigma(x) \neq x\}$, there exists m' that extends m on all the variables in σ (also those not in VI), such that $m' \models \wedge\{x \leftrightarrow \wedge var(\sigma(x)) \mid \sigma(x) \neq x\}$. Using m' we define δ as follows:

$$\forall x \in Var.\ \delta(x) = \begin{cases} a & \text{if } x \in m' \\ x & \text{otherwise.} \end{cases}$$

[2] The formulae of propositional logic defined by taking as propositional variables the elements of VI (see also Armstrong et al. [1994] and Marriott and Søndergaard [1993]).

It is easy to see that for any $x \in Var$, $\sigma_m(x)$ is ground $\Leftrightarrow x \in m'$. In fact

$$\begin{array}{lcll}\sigma_m(x) \text{ is ground} & \Leftrightarrow & \delta(\sigma(x)) \text{ is ground} & \\ & \Leftrightarrow & m' \supseteq Var(\sigma(x)) & \\ & \Leftrightarrow & x \in m' & (\text{since } m' \models x \leftrightarrow \wedge Var(\sigma(x))).\end{array}$$

It suffices now to observe that $ground(\sigma_m) = m' \cap VI = m$. $\square$

By this lemma, the following fact about the abstraction in *Def* of sets of substitutions is immediate. Here, the following notation is used: if A is any set and $X \subseteq \wp(A)$, then $cli(X)$ is the least superset of X that is closed under intersection of its elements.

Corollary 8.3.2. *For any* $\Sigma \subseteq Subst$,

$$\alpha_{Def}(\Sigma) = cli(\{ground(\sigma') \mid \sigma' \in \Delta_\sigma,\ \sigma \in \Sigma\}).$$

8.4 Relation between *Def* and *Sharing*

In Cortesi et al. [1992], it has been shown that there is a Galois connection between *Def* and *Sharing*. We prove here something more, namely the existence of a Galois insertion.

The abstraction function which maps an element S of *Sharing* into a formula of *Def* capturing its ground dependency information is defined as follows:

$$\alpha(S) = f, \text{ such that } Models(f) = \{\overline{\cup B} \mid B \neq \emptyset \text{ and } B \subseteq S\},$$

where $\overline{\cup B}$ is the set-theoretic complement of $\cup B$ with respect to *VI*, i.e., $\overline{\cup B} = VI \setminus \cup B$. It is not difficult to see that, according to the above definition, α is well-defined, i.e., $\alpha(S) \in Def$ for all $S \in Sharing$. In fact, $Y = \{\overline{\cup B} \mid B \neq \emptyset \text{ and } B \subseteq S\}$ is closed under intersection (i.e., conjunction of models):

$$\forall m_1 = \overline{\cup B_1}, m_2 = \overline{\cup B_2} \in Y.\ m_1 \cap m_2 = \overline{\cup(B_1 \cup B_2)} \in Y.$$

For instance, if $VI = \{x, y, z, w\}$ and $S = \{\emptyset, \{x, y\}, \{x, z\}\}$, then $\alpha(S)$ is the formula in *Def* with the following models:

$$\begin{array}{lcl}\{\emptyset\} \subseteq S & \xrightarrow{\text{gives the model}} & \{x, y, z, w\} \\ \{\{x, y\}\} \subseteq S & \xrightarrow{\text{gives the model}} & \{z, w\} \\ \{\{x, z\}\} \subseteq S & \xrightarrow{\text{gives the model}} & \{y, w\} \\ \{\{x, y\}, \{x, z\}\} \subseteq S & \xrightarrow{\text{gives the model}} & \{w\}.\end{array}$$

From this, one sees that $\alpha(S) = w \wedge (x \leftrightarrow (y \wedge z))$, which expresses that for every $\sigma \in \gamma_{Sharing}(S)$ the variable w is ground in σ, and x is ground in σ iff also y and z are ground in σ. Also observe that $\alpha(\emptyset) = false$.

In Cortesi et al. [1992], an abstraction function from *Sharing* to *Def* was given that looks different from the α described above. However, it is easy to show that the two functions coincide. The abstraction map $\mathcal{C}$ was given as follows:

$$\mathcal{C}(S) = \begin{cases} false & \text{if } S = \emptyset \\ \wedge\{\wedge W \to x \mid \{x\}, W \subseteq VI,\ \forall A \in S.\ x \in A \Rightarrow W \cap A \neq \emptyset\} & \text{otherwise.}\end{cases}$$

Lemma 8.4.1. $C = \alpha$.

PROOF. The case of $S = \emptyset$ is trivial. It is easy to see that $\forall S \in \mathit{Sharing} \setminus \{\emptyset\}.\ \alpha(S) \models \mathcal{C}(S)$: assume that m is a model of $\alpha(S)$ and that $m \not\models \mathcal{C}(S)$. Then, there is a definite formula $\wedge W \rightarrow x$ implied by $\mathcal{C}(S)$ such that $W \subseteq m$ and $x \notin m$. Since $m = \overline{\cup X}$, for some $X \subseteq S$, this means that there is $A \in X$ (and thus in S) such that $x \in A$ and $W \cap A = \emptyset$, but this contradicts the hypothesis that $\mathcal{C}(S) \models \wedge W \rightarrow x$.

For the other direction, assume that there is a model m of $\mathcal{C}(S)$ that is not a model of $\alpha(S)$. Let $X = \{A \in S \mid A \subseteq \overline{m}\}$. By the hypothesis that $m \not\models \alpha(S)$, it must be that $\overline{\cup X} \supset m$, and thus there exists $y \in \overline{\cup X} \setminus m$. It is obvious that $m \not\models \wedge m \rightarrow y$. We will now show that $\mathcal{C}(S) \models \wedge m \rightarrow y$, finding a contradiction. To this end, it suffices to observe that all $A \in S$ such that $A \cap m = \emptyset$ are in X, and thus, since $y \in \overline{\cup X}$, y is in none of these sets. □

The concretization function from *Def* into *Sharing*, adjoint to α, is defined as follows:

$$\gamma(f) = \{\overline{m} \subseteq VI \mid m \models f\}.$$

The functions α and γ are obviously monotonic. Indeed, they form a Galois insertion.

THEOREM 8.4.2. *α and γ form a Galois insertion of Def into Sharing.*

PROOF. We show the following points: (i) $\forall S \in \mathit{Sharing}.\ S \subseteq \gamma(\alpha(S))$; (ii) $\forall f \in \mathit{Def}.\ f = \alpha(\gamma(f))$.

(i) If $A \in S \in \mathit{Sharing}$, by definition of α, $\overline{A} \models \alpha(S)$, and thus, by definition of γ, $A \in \gamma(\alpha(S))$.

(ii) It suffices to observe that, since each formula $f \in \mathit{Def}$ has its set of models closed under intersection, $\gamma(f)$ is an element of *Sharing* closed under union. From this and the definition of α, the desired relation follows.

This concludes the proof. □

The domain *Def* can therefore be represented as the closure operator $\gamma \circ \alpha$ on *Sharing*. From the proof of the above theorem, it will not be surprising that this closure operator is as follows.

LEMMA 8.4.3. *The closure operator $\gamma \circ \alpha$ is the function that, for any $S \in \mathit{Sharing}$, gives the closure of S under set union. Formally, $\gamma \circ \alpha = \mathit{clu}$, where*

$$\mathit{clu}(S) = \{A \mid \exists A_1, \ldots, A_n \in S.\ A = A_1 \cup \ldots \cup A_n\}.$$

We show now that this insertion of *Def* into *Sharing* is *coherent* with the insertions of *Sharing* into $\wp(\mathit{Subst})$, and of *Def* into $\wp(\mathit{Subst})$. More precisely, we show that $\alpha_{\mathit{Def}} = \alpha \circ \alpha_{\mathit{Sharing}}$ and that $\gamma_{\mathit{Def}} = \gamma_{\mathit{Sharing}} \circ \gamma$. The following lemma is useful for this purpose. It is interesting to observe the similarity of this result with Lemma 8.3.1 and Corollary 8.3.2.

LEMMA 8.4.4. *Let $\Sigma \subseteq \mathit{Subst}$ and $\alpha_{\mathit{Sharing}}(\Sigma) = S$. Then,*

$$\mathit{cli}(\{\mathit{ground}(\sigma') \mid \sigma' \in \Delta_\sigma,\ \sigma \in \Sigma\}) = \{\overline{\cup X} \mid X \subseteq S\}.$$

PROOF.

($\subseteq$). Let $\sigma' \in \Delta_\sigma$ and $\sigma \in \Sigma$. We define

$$X = \{share(\sigma, y) \mid y \in Var,\ var(\sigma'(y)) \neq \emptyset\}.$$

We want to show that $ground(\sigma') = \overline{\cup X}$. If $\sigma'(y)$ is not ground, then all the variables x in $share(\sigma, y)$ are not ground in σ', i.e., $var(\sigma'(x))$ contains $var(\sigma'(y))$; conversely, if $x \notin \cup X$, then all variables in $var(\sigma(x))$ are made ground by σ', and thus x is also ground in σ'. Let us consider now two substitutions σ_1 and σ_2 that are instances of (possibly distinct) substitutions in Σ. By the above argument, there are $X_i \subseteq S, i = 1, 2$, such that $ground(\sigma_i) = \overline{\cup X_i}$; from this, $ground(\sigma_1) \cap ground(\sigma_2) = \overline{\cup(X_1 \cup X_2)}$.

($\supseteq$). By the argument just used, it suffices to show the following:

$$\forall A \in S . \exists \sigma \in \Sigma, \sigma' \in \Delta_\sigma .\ ground(\sigma') = \overline{A}.$$

By definition of $\alpha_{Sharing}$, there must be $y \in Var$ and $\sigma \in \Sigma$ such that $A = share(\sigma, y)$. The desired instance of σ is $\sigma' = \delta \circ \sigma$, with δ as follows:

$$\forall x \in Var.\ \delta(x) = \begin{cases} a & \text{if } share(\sigma, x) \neq share(\sigma, y), \\ x & \text{otherwise.} \end{cases}$$

It is easy to see that $ground(\sigma') = \overline{A}$.

This concludes the proof. □

We can now prove the coherency of α and γ.

THEOREM 8.4.5. *$\alpha_{Def} = \alpha \circ \alpha_{Sharing}$ and $\gamma_{Def} = \gamma_{Sharing} \circ \gamma$.*

PROOF. Since the composition of Galois insertions produces a Galois insertion [Cousot and Cousot 1992a], $\alpha \circ \alpha_{Sharing}$ and $\gamma_{Sharing} \circ \gamma$ form a Galois insertion of *Def* into $\wp(Subst)$. From this and the well-known fact that in a Galois insertion one of the two functions uniquely determines the other one [Cousot and Cousot 1992a], it suffices to show only one of the two relations in order to prove the other one too. We will show that $\alpha_{Def} = \alpha \circ \alpha_{Sharing}$. For any $\Sigma \in \wp(Subst)$,

$$\begin{aligned} \alpha_{Def}(\Sigma) &= cli(\{ground(\sigma') \mid \sigma' \in \Delta_\sigma, \sigma \in \Sigma\}) && \text{(by Corollary 8.3.2)} \\ &= \{\overline{\cup X} \mid X \subseteq \alpha_{Sharing}(\Sigma)\} && \text{(by Lemma 8.4.4)} \\ &= \alpha(\alpha_{Sharing}(\Sigma)) && \text{(by definition of } \alpha\text{).} \end{aligned}$$

This concludes the proof. □

As a consequence of previous results, we get the following corollary.

COROLLARY 8.4.6.

(i) $\alpha = \alpha_{Def} \circ \gamma_{Sharing}$;

(ii) $\gamma = \alpha_{Sharing} \circ \gamma_{Def}$.

PROOF. (i): From $\alpha_{Def} = \alpha \circ \alpha_{Sharing}$, shown in Theorem 8.4.5, one obtains, composing both sides with $\gamma_{Sharing}$,

$$\alpha_{Def} \circ \gamma_{Sharing} = \alpha \circ \alpha_{Sharing} \circ \gamma_{Sharing}.$$

It suffices now to observe that $\alpha_{Sharing} \circ \gamma_{Sharing}$ is the identity, because the two functions form a Galois insertion. Point (ii) is shown similarly. □

8.5 *Sharing*$^+$: The Complement of *Def* w.r.t. *Sharing*

In previous sections, we have shown that *Def* corresponds to the closure operator *clu* on *Sharing*, where for any $S \in \mathit{Sharing}$, $\mathit{clu}(S)$ is the closure of S under set union. We now compute the complement of *Def* with respect to *Sharing* by using the methodology illustrated in Section 3.1. We will call such a domain *Sharing*$^+$. In the following, we will use the representation of *Def* as (the set of fixpoints of) *clu*, i.e., $\mathit{Def} = \{S \in \mathit{Sharing} \mid S = \mathit{clu}(S)\}$.

We construct a chain $X_0, X_1, X_2, \ldots$ of subsets of *Sharing* following the definitions in Section 3.1. According to such definitions, we have that

$$X_0 = \{\wp(\mathit{VI})\}.$$

In order to construct X_1, consider the set $X = \mathit{maxs}(\mathit{Sharing} \setminus \mathit{Def})$. The set X_1 is defined as the closure of $X_0 \cup X$ under the *glb* on *Sharing*, namely under set intersection. Let us analyze X. For each $S \in X$, S is not closed under set union (otherwise $S \in \mathit{Def}$), and there exists only a nonempty set $A \subseteq \mathit{VI}$ such that $A \notin S$, i.e., $S = \wp(\mathit{VI}) \setminus \{A\}$ with $A \neq \emptyset$ (otherwise S would not be maximal). Furthermore, since S is not closed under union, A cannot be a singleton. Hence, we have

$$X = \{\wp(\mathit{VI}) \setminus \{A\} \mid A \subseteq \mathit{VI} \text{ and } |A| \geq 2\}.$$

Observe now that every element of *Sharing* which contains the empty set and all singletons of *VI* either is $\wp(\mathit{VI})$, or it can be obtained by set intersection of suitable elements of X. If we define $\mathit{singletons}(\mathit{VI}) = \{\{x\} \mid x \in \mathit{VI}\}$, we therefore have

$$X_1 = \{S \in \mathit{Sharing} \mid \mathit{singletons}(\mathit{VI}) \subseteq S\}.$$

We show now that the closure of $X_1 \cup \mathit{Def}$ under set intersection coincides with *Sharing*, which implies that we have already reached the limit of the construction, i.e., $X_2 = X_1$. Let $S \in \mathit{Sharing}$. If $S = \emptyset$ then $S \in \mathit{Def}$. Otherwise, observe that $S \cup \mathit{singletons}(\mathit{VI}) \in X_1$, $\mathit{clu}(S) \in \mathit{Def}$, and

$$\begin{aligned}(S \cup \mathit{singletons}(\mathit{VI})) \cap \mathit{clu}(S) &= (S \cap \mathit{clu}(S)) \cup (\mathit{singletons}(\mathit{VI}) \cap \mathit{clu}(S)) \\ &= S \cup (\mathit{singletons}(\mathit{VI}) \cap S) \\ &= S.\end{aligned}$$

We can therefore conclude that $X_2 = X_1$. Thus, defining

$$\mathit{Sharing}^+ = \{S \in \mathit{Sharing} \mid \mathit{singletons}(\mathit{VI}) \subseteq S\}$$

we have proved the following theorem.

THEOREM 8.5.1. *$\mathit{Sharing} \sim \mathit{Def} = \mathit{Sharing}^+$.*

The closure operator on *Sharing* corresponding to *Sharing*$^+$ is, of course, the closure under union with $\mathit{singletons}(\mathit{VI})$. Formally, denoting such operator by *cls*, we have $\mathit{cls}(S) = S \cup \mathit{singletons}(\mathit{VI})$. For instance, if $\mathit{VI} = \{x, y\}$, then *Sharing*$^+$ is the simple domain depicted below.

$$\begin{array}{c} \{\emptyset, \{x\}, \{y\}, \{x, y\}\} \\ | \\ \{\emptyset, \{x\}, \{y\}\} \end{array}$$

Let us try to interpret this result. If a set $S \in \mathit{Sharing}$ contains the singleton $\{x\}$, then S gives no ground dependency information concerning x, i.e., it represents all those substitutions where the groundness of x does not depend on the groundness of any other variable in VI. Thus, the presence of singletons prevents expressing ground dependencies. This intuition is confirmed by the easy observation that for any $S \in \mathit{Sharing}^+$, $\alpha(S) = \mathit{true}$. Apart from the ground dependencies, however, for $S \in \mathit{Sharing}$, all S' such that $S \setminus \mathit{singletons}(\mathit{VI}) \subseteq S' \subseteq S \cup \mathit{singletons}(\mathit{VI})$ do provide the same sharing information as S. This is because for any $y, z \in \mathit{VI}$ and any substitution σ, the fact that y and z share under σ is represented in *Sharing* by a set of cardinality at least two. These observations are coherent with the fact that $\mathit{Sharing}^+$ is what remains of *Sharing* once *Def* is removed from it.

9. RELATED AND FURTHER WORK

In this article, we have introduced the notion of complementation in abstract interpretation. Although our interest in this work is mainly concerned with abstract interpretation for program analysis, the same notion of complementation can be applied in any field where abstract interpretation theory is used. In particular, complementation can also be used for semantics related by abstract interpretation. Cousot and Cousot [1992b] proved that abstract interpretation can be used to systematically design hierarchies of semantics. In this case, 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 Comini and Levi [1994], and Giacobazzi [1996], where hierarchies of collecting semantics are designed by abstracting SLD resolution. The interest in complementation is therefore evident 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. A preliminary report on this research is in Giacobazzi and Ranzato [1996].

Very recently, Filé and Ranzato [1996] have stated a new sufficient lattice-theoretic condition on the complete lattice L that guarantees the existence of pseudocomplements of closure operators on L. The relationship with the condition of meet-continuity of L in Theorem 3.6 is, to the best of our knowledge, not yet known in the lattice-theoretic literature. Also, Filé and Ranzato [1996] provided a practical systematic methodology, based on standard lattice-theoretic notions, to compute complements, having some analogy with that presented in Section 3.1. Further work will be devoted to understand the precise relationship between the two approaches. Another recent related work is in Marchiori [1996], where the author considers decompositions of some domains for the analysis of logic programs characterized by means of first-order formulae.

ACKNOWLEDGMENTS

We are grateful to the anonymous referees and to the TOPLAS Associate Editor Sam Kamin, for their helpful comments. This article has been conceived while Roberto Giacobazzi and Francesco Ranzato were visiting *LIX, Laboratoire d'Informatique, École Polytechnique*, Paris. They thank *LIX*, and in particular Radhia Cousot, for the kind hospitality.

REFERENCES

ARMSTRONG, T., MARRIOTT, K., SCHACHTE, P., AND SØNDERGAARD, H. 1994. Boolean functions for dependency analysis: Algebraic properties and efficient representation. In *Proceedings of the 1st International Static Analysis Symposium* (*SAS '94*), B. Le Charlier, Ed. Lecture Notes in Computer Science, vol. 864. Springer-Verlag, Berlin, 266–280.

BIRKHOFF, G. 1967. *Lattice Theory.* AMS Colloquium Publication, 3rd edition. AMS, Providence, R.I.

CODISH, M., MULKERS, A., BRUYNOOGHE, M., GARCÍA DE LA BANDA, M., AND HERMENEGILDO, M. 1995. Improving abstract interpretations by combining domains. *ACM Trans. Program. Lang. Syst. 17,* 1, 28–44.

COMINI, M. AND LEVI, G. 1994. An algebraic theory of observables. In *Proceedings of the 1994 International Logic Programming Symposium* (*ILPS '94*), M. Bruynooghe, Ed. The MIT Press, Cambridge, Mass., 172–186.

CORTESI, A., FILÉ, G., AND WINSBOROUGH, W. 1992. Comparison of abstract interpretations. In *Proceedings of the 19th International Colloquium on Automata, Languages and Programming* (*ICALP '92*), W. Kuich, Ed. Lecture Notes in Computer Science, vol. 623. Springer-Verlag, Berlin, 521–532.

CORTESI, A., FILÉ, G., AND WINSBOROUGH, W. 1996. Optimal groundness analysis using propositional logic. *J. Logic Program. 27,* 2, 137–167.

COUSOT, P. 1978. Méthodes Itératives de Construction et d'Approximation de Points Fixes d'Opérateurs Monotones sur un Treillis, Analyse Sémantique des Programmes. Ph.D. thesis, Université Scientifique et Médicale de Grenoble, Grenoble, France.

COUSOT, P. AND COUSOT, R. 1976. Static determination of dynamic properties of programs. In *Proceedings of the 2nd International Symposium on Programming.* Dunod, Paris, 106–130.

COUSOT, P. AND COUSOT, R. 1977. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the 4th ACM Symposium on Principles of Programming Languages* (*POPL '77*). ACM Press, New York, 238–252.

COUSOT, P. AND COUSOT, R. 1979. Systematic design of program analysis frameworks. In *Conference Record of the 6th ACM Symposium on Principles of Programming Languages* (*POPL '79*). ACM Press, New York, 269–282.

COUSOT, P. AND COUSOT, R. 1992a. Abstract interpretation and application to logic programs. *J. Logic Program. 13,* 2-3, 103–179.

COUSOT, P. AND COUSOT, R. 1992b. Inductive definitions, semantics and abstract interpretation. In *Conference Record of the 19th ACM Symposium on Principles of Programming Languages* (*POPL '92*). ACM Press, New York, 83–94.

COUSOT, P. AND COUSOT, R. 1994. Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and per analysis of functional languages). In *Proceedings of the IEEE International Conference on Computer Languages* (*ICCL '94*). IEEE Computer Society Press, Los Alamitos, Calif., 95–112.

DART, P. 1988. Dependency analysis and query interfaces for deductive databases. Ph.D. thesis, Univ. of Melbourne, Melbourne, Australia.

DART, P. 1991. On derived dependencies and connected databases. *J. Logic Program. 11,* 2, 163–188.

DAVEY, B. A. AND PRIESTLEY, H. A. 1990. *Introduction to Lattices and Order.* Cambridge University Press, Cambridge, U.K.

DWINGER, P. 1954. On the closure operators of a complete lattice. *Indagationes Mathematicæ 16,* 560–563.

FILÉ, G. AND RANZATO, F. 1996. Complementation of abstract domains made easy. In *Proceedings of the 1996 Joint International Conference and Symposium on Logic Programming* (*JICSLP '96*), M. Maher, Ed. The MIT Press, Cambridge, Mass., 348–362.

FILÉ, G., GIACOBAZZI, R., AND RANZATO, F. 1996. A unifying view of abstract domain design. *ACM Comput. Surv. 28,* 2, 333–336.

FRINK, O. 1962. Pseudo-complements in semi-lattices. *Duke Math. J. 29,* 505–514.

GIACOBAZZI, R. 1996. "Optimal" collecting semantics for analysis in a hierarchy of logic program semantics. In *Proceedings of the 13th International Symposium on Theoretical Aspects of Computer Science (STACS '96)*, C. Puech and R. Reischuk, Eds. Lecture Notes in Computer Science, vol. 1046. Springer-Verlag, Berlin, 503–514.

GIACOBAZZI, R. AND RANZATO, F. 1996. Complementing logic program semantics. In *Proceedings of the 5th International Conference on Algebraic and Logic Programming (ALP '96)*, M. Hanus and M. Rodríguez Artalejo, Eds. Lecture Notes in Computer Science. Springer-Verlag, Berlin, 238–253.

GIACOBAZZI, R., PALAMIDESSI, C., AND RANZATO, F. 1996. Weak relative pseudo-complements of closure operators. *Algebra Universalis 36,* 3, 405–412.

GIERZ, G., HOFMANN, K. H., KEIMEL, K., LAWSON, J. D., MISLOVE, M., AND SCOTT, D. S. 1980. *A Compendium of Continuous Lattices.* Springer-Verlag, Berlin.

GRÄTZER, G. 1978. *General Lattice Theory.* Birkhäuser Verlag, Basel, Switzerland.

HUNT, S. 1990. PERs generalize projections for strictness analysis. In *Proceedings of the 1990 Glasgow Functional Programming Workshop*, S. Peyton Jones, G. Hutton, and C. K. Holst, Eds. Workshops in Computing. Springer-Verlag, Berlin, 156–168.

JACOBS, D. AND LANGEN, A. 1989. Accurate and efficient approximation of variable aliasing in logic programs. In *Proceedings of the 1989 North American Conference on Logic Programming (NACLP '89)*, E. L. Lusk and R. A. Overbeek, Eds. The MIT Press, Cambridge, Mass., 154–165.

JACOBS, D. AND LANGEN, A. 1992. Static analysis of logic programs for independent AND-parallelism. *J. Logic Program. 13,* 2-3, 154–165.

KAM, J. B. AND ULLMAN, J. D. 1977. Monotone data flow analysis frameworks. *Acta Informatica 7,* 305–317.

KILDALL, G. A. 1973. A unified approach to global program optimization. In *Conference Record of the 1st ACM Symposium on Principles of Programming Languages (POPL '73)*. ACM Press, New York, 194–206.

MARCHIORI, E. 1996. Prime factorizations of abstract domains using first order logic. In *Proceedings of the 5th International Conference on Algebraic and Logic Programming (ALP '96)*, M. Hanus and M. Rodríguez Artalejo, Eds. Lecture Notes in Computer Science. Springer-Verlag, Berlin, 209–223.

MARRIOTT, K. AND SØNDERGAARD, H. 1993. Precise and efficient groundness analysis for logic programs. *ACM Lett. Program. Lang. Syst. 2,* 1-4, 181–196.

MORGADO, J. 1962. Note on complemented closure operators of complete lattices. *PortugaliæMathematica 21,* 3, 135–142.

MUTHUKUMAR, K. AND HERMENEGILDO, M. 1991. Combined determination of sharing and freeness of program variables through abstract interpretation. In *Proceedings of the 8th International Conference on Logic Programming (ICLP '91)*, K. Furukawa, Ed. The MIT Press, Cambridge, Mass., 49–63.

MYCROFT, A. 1980. The theory and practice of transforming call-by-need into call-by-value. In *Proceedings of the 4th International Symposium on Programming*, B. Robinet, Ed. Lecture Notes in Computer Science, vol. 83. Springer-Verlag, Berlin, 270–281.

MYCROFT, A. 1981. Abstract interpretation and optimising transformations for applicative programs. Ph.D. thesis, CST-15/81, Univ. of Edinburgh, Edinburgh, Scotland.

MYCROFT, A. 1993. Completeness and predicate-based abstract interpretation. In *Proceedings of the ACM Symposium on Partial Evaluation and Program Manipulation (PEPM '93)*. ACM Press, New York, 179–185.

NIELSON, F. 1985. Tensor products generalize the relational data flow analysis method. In *Proceedings of the 4th Hungarian Computer Science Conference*, M. Arató, I. Kátai, and L. Varga, Eds. 211–225.

SUNDARARAJAN, R. AND CONERY, J. 1992. An abstract interpretation scheme for groundness, freeness, and sharing analysis of logic programs. In *Proceedings of the 12th Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS '92)*, R. Shyamasundar, Ed. Lecture Notes in Computer Science, vol. 652. Springer-Verlag, Berlin, 203–216.

VARLET, J. 1963. Contribution à l'étude des treillis pseudo-complémentés et des treillis de Stone. *Mémoires de la Société Royale des Sciences de Liège 8,* 4, 1–71.

WADLER, P. AND HUGHES, R. J. M. 1987. Projections for strictness analysis. In *Proceedings of the Conference on Functional Programming Languages and Computer Architecture* (*FPCA '87*), G. Kahn, Ed. Lecture Notes in Computer Science, vol. 274. Springer-Verlag, Berlin, 385–407.

WARD, M. 1942. The closure operators of a lattice. *Ann. Math. 43,* 2, 191–196.

YI, K. AND HARRISON, W. L. 1993. Automatic generation and management of interprocedural program analyses. In *Conference Record of the 20th ACM Symposium on Principles of Programming Languages* (*POPL '93*). ACM Press, New York, 246–259.

Received November 1995; revised June 1996; accepted October 1996