Centrum voor Wiskunde en Informatica
REPORT*RAPPORT*

Design of abstract domains using first-order logic

E. Marchiori

Computer Science/Department of Interactive Systems

CS-R9633 1996

Report CS-R9633
ISSN 0169-118X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

Design of Abstract Domains Using First-Order Logic

Elena Marchiori
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
and
University of Leiden
P.O. Box 9512, 2300 RA Leiden, The Netherlands
e-mail: elena@cwi.nl

Abstract

In this paper we propose a simple framework based on first-order logic, for the design and decomposition of abstract domains for static analysis. An assertion language is chosen that specifies the properties of interest, and abstract domains are defined to be suitably chosen sets of assertions. Composition and decomposition of abstract domains is facilitated by their logical specification in first-order logic. In particular, the operations of reduced product and disjunctive completion are formalized in this framework. Moreover, the notion of (conjunctive) factorization of sets of assertions is introduced, that allows one to decompose domains in 'disjoint' parts. We illustrate the use of this framework by studying typical abstract domains for ground-dependency and aliasing analysis in logic programming.

AMS Subject Classification (1991): 68Q40, 68N17, 03B70
CR Subject Classification (1991): D.2.10, F.3.1, D.1.6
Keywords & Phrases: Abstract domains, static analysis, first-order logic

1 Introduction

In the theory of abstract interpretation [3], abstract domains are (computer) representations of properties. The semantics of an abstract domain is given by a function called concretization, that maps elements of the abstract domain into elements of a 'concrete domain'. Two fundamental aspects of the study of abstract domains are the investigation of representations supporting efficient implementations, and the comparative study of the properties represented by abstract domains. This paper is concerned with the latter aspect.

In the standard approach the design phase is not clearly distinguished from the representation one. In general, once the concrete domain is chosen, a representation of an abstract domain is directly defined by means of a suitable lattice structure equipped with a concretization function. Then the image of the abstract domain under its concretization function is used to study its properties, as well as for comparing the domain with other ones (w.r.t. the same concrete domain). There are two equivalent techniques for the study and comparison of the properties represented by abstract domains (cf. [3]): *Galois connections*, where an abstract domain is a complete lattice together with a Galois connection (or insertion) that relates the abstract domain with the 'concrete' one; and *closure operators*, where an abstract domain is

identified with the image of a suitable closure operator on the 'concrete' domain. For instance, Galois connections are used in [3] to define the notion of domain abstraction for embedding one domain into another one, and in [9] to define the notion of domain quotient for extracting from a domain the part that describes a given property. Closure operators have been used in two recent works [6, 14] to define the notions of domain (pseudo)-complementation and disjunctive basis, for studying the 'inverse' of the operators of reduced product and of disjunctive completion. In particular, in [6] a notion of domain decomposition is introduced, and pseudo-complementation is used for decomposing abstract domains.

The aim of this paper is to introduce a framework for the design and study of abstract domains. We adopt the setting of first-order logic, because it is a familiar formalism for specifying as well as for reasoning about properties. The idea is to define abstract domains in logical form, and then to use 'isomorphic' copies in some other lattice structures as representations for the implementation. In order to define a specific instance of the framework, one has to make two choices: 1) the set, say V, of syntactic objects (generally a subset of the variables of the considered program) to be analyzed; 2) the first-order assertion language, say $\mathcal{L}$, for specifying the properties of V one wants to study. Thus V is (identified with) a set of the variables of $\mathcal{L}$. An abstract domain is defined to be a suitably chosen set of assertions, whose free variables are contained in V. In this way, only the information on the objects of interest (i.e., of V) is taken into account.

Composition of abstract domains can be performed at the logical level in the following way: the reduced product of two domains consists of all the conjunctions of their assertions, and the disjunctive completion of a domain consists of all the disjunctions of its assertions. Moreover, in order to decompose abstract domains, we introduce the notion of (conjunctive) factorization of sets of assertions, where domains are factorized in 'disjoint' parts. It is worth noting that this desirable property is not guaranteed in the decompositions obtained using the method of [6].

Computer representations of abstract domains are defined in the expected way, i.e., they have to respect (i.e., be isomorphic to) their specification in the assertion language.

This framework can be embedded in the standard one based on abstract interpretation (under the assumption that the assertion language is enough expressive): the concrete domain consists of suitably chosen sets of valuations, and the concretization function of a domain maps an assertion into the set of valuations that satisfy it.

The benefits of using a logical framework as the one we propose can be summarized as follows. The two phases of design and computer representation of abstract domains are neatly separated, where the design phase is performed at the logical level. Moreover, the choice of the assertion language allows one to focus only on the abstract domains that describe the properties of interest, that are those expressible in that language. This is not the case for the standard methods above mentioned, where all possible abstract domains (on the concrete domain) are taken into account.

We illustrate this approach by considering typical abstract domains for ground-dependency and aliasing analysis in logic programming. The fragment $\mathcal{L}$ of a first-order assertion language introduced in [19] (actually, a slight extension of this) is used. Logical descriptions of various abstract domains are given: *Def* [10] and *Pos* [21, 22] for ground-dependency analysis; *Sharing* [15] and *ASub* [26] for aliasing analysis. Maximal factorizations for these domains are obtained by inspecting the structure of the assertions in the abstract domains, and they are used for analyzing and comparing the abstract domains. Moreover, we study the disjunctive completion of these domains.

The paper is organized as follows. The next section introduces a methodology for the design and decomposition of abstract domains using first-order logic. Section 3 presents an assertion language for the design of typical abstract domains for logic programming. Section 4 contains a comparative study of various abstract domains for logic programming. Section 5 discusses some related work. Finally, in Section 6 we conclude with a discussion on other applications and on future work. A preliminary version of this paper appeared in [20].

2 Abstract Domains in Assertion Form

We show in this section how first-order logic can be used for the design of abstract domains for abstract interpretation. The approach is based on the seminal work of the Cousots [3]. First, a first-order assertion language $\mathcal{L}$ is chosen, in order to describe the properties of interest. Next, abstract domains (on $\mathcal{L}$) are defined as suitably chosen sets of assertions of $\mathcal{L}$. Finally, (efficient) computer representations of abstract domains are defined as usual, i.e., as isomorphic copies of their logical specification in $\mathcal{L}$. In order to decompose abstract domains, the notion of (conjunctive) factorization on $\mathcal{L}$ is introduced, where abstract domains are decomposed in pairwise 'disjoint' parts.

Here and in the sequel $\mathcal{L}$ denotes a generic assertion language. We assume that the semantics of the predicates in $\mathcal{L}$ is fixed according to their intended meaning, by a given structure denoted by $\mathcal{M}$. Assertions are indicated by ϕ, ψ. As already mentioned, abstract domains represent properties of some syntactic objects, usually a subset of the variables of the considered program. Thus, the definition of abstract domain we give is parametric with respect to a set V of syntactic objects. We adopt the following convenient assumptions:

1. V is (identified with) a set of distinct variables of $\mathcal{L}$;
2. in the definition of abstract domain, only the set of assertions of $\mathcal{L}$ whose free variables are contained in V is considered, denoted by $A(\mathcal{L}, V)$;
3. assertions with the same meaning are identified.

The first two assumptions ensure that only the information on the objects of interest (i.e., of V) is taken into account. The last assumption amounts to consider equivalence classes of assertions of $A(\mathcal{L}, V)$, where $[\phi]$ denotes all the assertions that are logically equivalent to ϕ. For simplicity, in the sequel the squares in $[\phi]$ are often omitted.

Definition 2.1 (Abstract Domain on $\mathcal{L}$) An abstract domain (on $\mathcal{L}$), denoted by $\mathcal{A}$ (possibly subscripted), is a set of assertions of $A(\mathcal{L}, V)$ containing *false*, and closed under conjunction and variance[1]. □

One can characterize an abstract domain (on $\mathcal{L}$) by means of Galois connections according to the standard approach in the following way. In order to define the concrete domain for the Galois connections, we introduce an equivalence relation, based on the observation that valuations mapping the variables of V into the same object can be identified, since the free variables of the assertions in a domain are contained in V. Let $\sim_V$ be the relation on valuations s.t. $\sigma \sim_V \sigma'$ if $\sigma(x) = \sigma'(x)$ for every $x \in V$. Clearly $\sim_V$ is an equivalence relation.

[1]Recall that a variant of an assertion ϕ is any assertion $\phi\rho$ obtained by applying to ϕ a function ρ that renames the variables of ϕ

Consider the set $Val/_V$ of those equivalence classes w.r.t. $\sim_V$ that are closed under variance w.r.t. V[2]. Then the concrete domain $Conc_V$ is the family of subsets of $Val/_V$. Consider now a domain $\mathcal{D}$ on $\mathcal{L}$. Its concretization function γ maps an assertion ϕ into the set of $Conc_V$ consisting of the equivalence classes of valuations that satisfy ϕ. It is easy to check that γ induces a Galois connection (actually a Galois insertion) (γ, α) of $\mathcal{D}$ into $Conc_V$[3]. We shall see at the end of this section that also the converse holds, under a suitable assumption on the assertion language.

Example 2.2 A simple abstract domain for the study of the sign of program variables assuming integer values is given in [3]. For a considered set V of program variables, this domain can be specified in our formalism as follows: $\mathcal{L}$ contains the constants and function symbols of the program, and the unary predicates $\geq$, $\leq$; $\mathcal{M}$ maps terms into integers according to their intended interpretation, and specify the semantics of $\geq$, $\leq$ in the expected way. Then the abstract domain for the study of the sign of the variables in V can be described by the set $Sign_V$ of assertions that are conjunctions of atoms of the form $x \geq 0$, or $x \leq 0$, with x in V. □

We conclude with an observation on the lattice structure of our concrete domain.

Proposition 2.3 *The set $Conc_V$ is an algebraic complete lattice with intersection and union as meet and join, respectively.*

It is worth noticing that in the standard framework [3], the concrete domain is a complete lattice, but it is not in general algebraic. The property of algebraicity of $Conc_V$ simplifies the study of abstract domains, as we shall see in the sequel. In the sequel, for simplicity, we shall write valuations instead of equivalence classes of valuations. Moreover, we shall often avoid to mention the element *false* when specifying the set of assertions of an abstract domain.

In order to improve the precision of the static analysis of logic programs, various operators on abstract domains have been introduced. Two fundamental operators are the reduced-product and the disjunctive completion ([3]). In the following two subsections we discuss the correspondent of these operators on $\mathcal{L}$.

2.1 Reduced Product

The reduced product of two domains is obtained from the cardinal product of the domains by identifying pairs of elements whose conjunction represent the same information. We can characterize this notion in the logical framework on $\mathcal{L}$ as follows.

Definition 2.4 The *reduced product of two domains* $\mathcal{A}_1$, $\mathcal{A}_2$ ($\mathcal{A}_1 \wedge \mathcal{A}_2$) is the set $\{[\phi_1 \wedge \phi_2] \mid \phi_1 \in \mathcal{A}_1,\ \phi_2 \in \mathcal{A}_2\}$.

[2]The notion of variant w.r.t. V of a set d of valuations is defined in the expected way: let ρ be a substitution that renames the variables of V with other variables of V. Then a variance of d is obtained by applying ρ to the domain of every valuation

[3]Recall that (γ, α) is a Galois connection if: a) γ, α are monotonic functions (i.e., $S \subseteq S'$ implies $\alpha(S) \Rightarrow \alpha(S')$, and $\phi \Rightarrow \phi'$ implies $\gamma(\phi) \subseteq \gamma(\phi')$); b) $S \subseteq \gamma(\alpha(S))$ for every S in $Conc_V$; and c) $\alpha(\gamma(\phi)) \Rightarrow \phi$ for every ϕ in $\mathcal{D}$. If in the condition c) we have $\Leftrightarrow$ instead of $\Rightarrow$ then (γ, α) is a Galois insertion

The notion of reduced product can be used to define the concept of (conjunctive) domain decomposition. For instance, in [6] a definition of decomposition of a domain D is given, as a set of domains whose reduced product yields D. Here we consider a stronger notion of decomposition (in $\mathcal{L}$) where the factors have to be pairwise 'disjoint'. A comparison with the work in [6] is postponed to the Section 5.

Here and in the sequel, the notation $\mathcal{A}_1 = \mathcal{A}_2$ is used, meaning that $\mathcal{A}_1$ and $\mathcal{A}_2$ contain the same equivalence classes.

Definition 2.5 (Conjunctive Factorization on $\mathcal{L}$) The set $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ of abstract domains is a *(conjunctive) factorization* of $\mathcal{A}$ if the following conditions hold:
(a) If $n > 1$ then $\mathcal{A}_i \neq \{true, false\}$, for $i \in [1, n]$;
(b) $\mathcal{A}_i \cap \mathcal{A}_j = \{true, false\}$ for every $i \neq j$;
(c) $\mathcal{A}_1 \wedge \ldots \wedge \mathcal{A}_n = \mathcal{A}$.

We call $\mathcal{A}$ *reduced* if it has only one factorization. Moreover, a factorization of $\mathcal{A}$ is *maximal* if $\mathcal{A}_i$ is $\wedge$-reduced, for $i \in [1, n]$. □

It follows from the definition that an abstract domain has always a factorization (e.g., $\{\mathcal{A}\}$). Moreover, if $\mathcal{A}$ is $\wedge$-reduced then $\{\mathcal{A}\}$ is its only factorization, and it is maximal.

Example 2.6 It is easy to check that $\{Sign_{\leq 0}, Sign_{\geq 0}\}$ is a maximal factorization of $Sign_V$, where $Sign_{\leq 0}$ is the set of assertions that are conjunctions of atoms of the form $x \leq 0$, with x in V, and where $Sign_{\geq 0}$ is defined analogously.

□

2.2 Disjunctive Completion

The disjunctive completion of a domain is obtained from the powerset of the domain by identifying sets whose disjunction represent the same information. We can characterize this notion in the logical framework on $\mathcal{L}$ as follows.

Definition 2.7 The *disjunctive completion of* $\mathcal{A}$ ($\vee\mathcal{A}$) is the set $\{[\phi_1 \vee \ldots \vee \phi_n] \mid n \geq 0,\ \phi_1, \ldots, \phi_n \in \mathcal{A}\}$.

The operator of disjunctive completion has been thoroughly investigated in [13], where the inverse of this operator, called least disjunctive basis, is introduced. We can introduce a similar notion in our framework as follows.

Definition 2.8 A domain $\mathcal{A}$ *$\vee$-reduced (in $\mathcal{L}$)* if for every $\mathcal{A}'$ s.t. $\vee\mathcal{A}' = \vee\mathcal{A}$ we have that $\mathcal{A} \subseteq \mathcal{A}'$.

It is easy to check that the domain $Sign_V$ introduced in Example 2.2 is $\vee$-reduced.

Call an abstract domain $\mathcal{A}$ *$\vee$-closed* if $\vee\mathcal{A} = \mathcal{A}$. Every $\vee$-closed abstract domain is the disjunctive completion of a $\vee$-reduced abstract domain.

Proposition 2.9 *Suppose that $\mathcal{A}$ is $\vee$-closed. Then there exists a $\vee$-reduced domain $\mathcal{A}'$ s.t. $\vee\mathcal{A}' = \mathcal{A}$.*

A similar result is proven in [14] (Theorem 4.5) under the hypothesis that the concrete domain is (dual-)algebraic. Here this hypothesis is implied by Proposition 2.3.

Moreover, the following result follows directly.

Proposition 2.10 *$\mathcal{A}$ is $\vee$-closed if and only if it is closed under disjunction.*

A similar result is given in [13] (Theorem 3.7) under the assumption that the concrete domain is a completely meet-distributive lattice. Here Proposition 2.3 allows us to drop this assumption.

In the following section we formalize the notion of domain representation in the logical framework.

2.3 Domain Representation

The benefit of using this first-order framework is that the definition, decomposition and comparison of abstract domains can be performed in a uniform and familiar setting. However, (computer) representations of abstract domains for their efficient manipulation ([11]) often need different lattice structures (see, e.g., [2] for ground-dependency analysis). Therefore the notion of representation of an abstract domain is defined as follows. First, we need some preliminary terminology. The following notion of embedding of an abstract domain into $\mathcal{L}$ is used. Here and in the the sequel $\mathcal{D}$ denotes an abstract domain (on any complete lattice) and $\gamma_{\mathcal{D}}$ denotes its concretization function (cf. [3]).

Definition 2.11 (Embedding) An *embedding* of $\mathcal{D}$ in $\mathcal{L}$ is an injective mapping $\epsilon_{\mathcal{D}} : \mathcal{D} \to \mathcal{L}$ s.t. for every D in $\mathcal{D}$, α is in $\gamma_{\mathcal{D}}(D)$ if and only if $\epsilon(D)$ is true under α. □

Thus an embedding of a domain into $\mathcal{L}$ consists of the (equivalence classes of the) assertions ϕ_D characterizing the sets $\gamma_{\mathcal{D}}(D)$ of valuations, with D in $\mathcal{D}$. The following result is an easy consequence of the definition of concretization function ([3]).

Proposition 2.12 *The image $\epsilon_{\mathcal{D}}(\mathcal{D})$ of an embedding is an abstract domain on $\mathcal{L}$.*

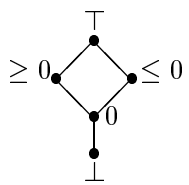
Proof. $\epsilon_{\mathcal{D}}$ is an isomorphism of $\mathcal{D}$ into $\epsilon_{\mathcal{D}}(\mathcal{D})$. □

We can now formalize the concept of representation domain.

Definition 2.13 (Representation Domain) $\mathcal{D}$ is a *representation* of $\mathcal{A}$ (or equivalently $\mathcal{A}$ and $\mathcal{D}$ are *isomorphic*, denoted by $\mathcal{A} \simeq \mathcal{D}$) if there exists an *embedding* $\epsilon_{\mathcal{D}}$ s.t. $\mathcal{A} = \epsilon_{\mathcal{D}}(\mathcal{D})$. □

The definition of representation domain clarifies the role of domains in assertion form, as those used in the design phase, in contrast to the representation domains used in the (efficient) implementation. We conclude with an example.

Example 2.14 Suppose $V = \{x\}$. Then a representation of $Sign_V$ is the familiar lattice pictured below



2.4 Relation with the Standard Approach

We conclude this section with a discussion on the relationship of our framework with the standard approach based on closure operators (or equivalently on Galois connections). We have already shown that our notion of abstract domain (Definition 2.1) is consistent with the original definition (cf. [3]). We shall give here the same result in terms of closure operators. Moreover, we shall see that a full equivalence of our framework with the standard one holds only under the assumptions that the assertion language is enough expressive.

Recall that $Val/_V$ denotes the set (of equivalence classes) of valuations that are closed under variance w.r.t V, and $Val/_V$ denotes $2^{Val/_V}$. Let us start by giving few preliminaries on closure operators (the reader interested to this subject is referred to e.g. [12]). Let X be a set. An *upper closure operator on* X is a function $c : 2^X \to 2^X$ that is extensive ($S \subseteq c(S)$), monotonic ($S \subseteq S'$ implies $c(S) \subseteq c(S')$) and idempotent ($c(c(S)) = c(S)$). An important characterization of closure operators that we shall use is given in terms of intersection structures ($\bigcap$ structures). An *intersection structure (on X)* is a non-empty family of subsets of X which is closed under intersection. Moreover, it is called *topped* if it contains X. For every closure operator c on X, the family $\mathcal{S}_c$ of those sets S s.t. $c(S) = S$ is a topped $\bigcap$ structure. Vice versa, for every topped $\bigcap$ structure $\mathcal{S}$ (on X), the formula $c_{\mathcal{S}}(S) = \bigcap_{S' \in \mathcal{S}, S' \subseteq S} S'$ defines a closure operator on X. Moreover, the closure operator induced by the topped $\bigcap$ structure $\mathcal{S}_c$ is c itself, and, similarly, the $\bigcap$ structure induced by the closure operator $c_{\mathcal{S}}$ is $\mathcal{S}$.

The importance of this result relies on the fact that, if we identify an assertion with the set of valuations under which it is true, then an abstract domain on $\mathcal{L}$ is a topped algebraic[4] intersection structure on $Val/_V$, hence it induces a closure operator on $Val/_V$ defined as above.

In the standard approach, also the vice versa holds, i.e., the lattice of abstract domains is isomorphic to the lattice of upper closure operators. This result does not hold with our notion of abstract domain, because (the topped intersection structure induced by) a closure operator is an abstract domain on $\mathcal{L}$ only if it can be described by means of a set of assertions of $\mathcal{L}$. However, if one assumes that the assertion language allows to describe all the subsets of $Val/_V$, then the lattice of abstract domains (according to Definition 2.1) is isomorphic to the lattice of upper closure operators on $Val/_V$.

Formally, call $\mathcal{L}$ *complete w.r.t. V* if for every subset S of $Val/_V$ there there exists an assertion ϕ on $\mathcal{L}$ s.t. ϕ is true under all the valuations of S and under no other one. Then we have the following result.

Proposition 2.15 *Suppose that $\mathcal{L}$ is complete w.r.t. V. Then the set of abstract domains on $\mathcal{L}$ is isomorphic to the set of (upper) closure operators on $Val/_V$.*

3 Abstract Domains for Logic Programming

In this section, we show how a slight extension of the first-order assertion language $\mathcal{L}$ introduced in [19] can be used for the design and decomposition of typical abstract domains for the static analysis of logic programs.

[4] A $\bigcap$ structure $\mathcal{S}$ is algebraic if it is closed under the union of directed subfamilies. $\mathcal{S}$ is a direct subfamily if, for every finite subset $\mathcal{T}$ of $\mathcal{S}$, there exists a S in $\mathcal{S}$ s.t. $T \subseteq S$ for every $T \in \mathcal{T}$

Term properties, like groundness and sharing, have been identified as crucial when analyzing the run-time behaviour of logic programs. For instance, ground-dependency analysis can be used for compile optimization, by using matching instead of unification when it is known that at a given program point a variable is bound to a ground term every time the execution reaches that point. Information on the sharing among variables in a logic program is useful for important optimizations, like and-parallelism. The assertion language here considered allows to express properties of terms, like groundness, freeness, linearity, sharing, covering and independency. Informally, a term is *ground* if it does not contain variables, it is *free* if it is a variable, and it is *linear* if every variable occurs in it at most once. Moreover, a set of terms *share* if they have at least one common variable, while they are *independent* if they do not share. Finally, a term is *covered* by a set of terms if the set of its variables is contained in the union of the sets of variables of the terms in that set. For instance, the term $f(x, y)$ is covered by the set $\{g(x), g(y)\}$.

In order to define $\mathcal{L}$, a countable set *Var* of *(logical) variables* is used, denoted by v, x, y, z, possibly subscripted. Here and in the sequel, S represents a finite set of logical variables, and $|S|$ its cardinality. Moreover, the notation $S \subset S'$ indicates that S is a proper subset of S'.

Definition 3.1 (The Assertion Language) Let $\mathcal{L}'$ be the smallest set F of formulas containing atoms of the form $var(x)$, $ground(x)$, $linear(x)$, $share(S)$, and s.t. if ϕ_1 and ϕ_2 are in F then $\neg\phi_1$ and $\phi_1 \wedge \phi_2$ are also in F. The assertion language $\mathcal{L}$ consists of all the formulas of the form $\forall x_1, \ldots, x_n(\phi)$, with $\phi \in \mathcal{L}'$, and $n \geq 0$. □

The formula $\phi \vee \psi$ is used as a shorthand for $\neg(\neg\phi \wedge \neg\psi)$, $\phi \Rightarrow \psi$ denotes $\neg\phi \vee \psi$, and $\phi \Leftrightarrow \psi$ stands for $(\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$. Moreover, the propositional constants *true* and *false* are assumed to be in $\mathcal{L}$, where *true* is identified with the conjunction over the empty set of assertions $\wedge\emptyset$ and *false* with $\vee\emptyset$. In the sequel, the notation $share(x, y)$ is used as shorthand of $share(\{x, y\})$, with x, y distinct.

Observe that only a weak form of universal quantification is allowed, where $\forall$ does not occur in the scope of any $\neg$. For instance, $\forall z\,(var(z) \wedge \neg share(\{z, x\}))$ is in $\mathcal{L}$, but $\neg\forall z\,(var(z) \wedge \neg share(\{z, x\}))$ is not in $\mathcal{L}$.

The meaning of assertions in $\mathcal{L}$ is specified by means of the following structure $\mathcal{M}$. Let *OVar* be the set of *(object) variables*, here identified for simplicity with *Var*, and let *Fun* be a set of *functors with rank* (constants are identified with functors of rank 0). In the following, $occ(x, \tau)$ denotes the number of occurrences of the variable x in the term τ, and $OVar(\tau)$ the set of (object) variables occurring in τ.

Definition 3.2 The structure $\mathcal{M}$ contains the universe $\mathcal{U}$ consisting of the *(object) terms* built on *OVar* and *Fun*. Moreover, for each predicate symbol p of $\mathcal{L}$, $\mathcal{M}$ contains a predicate in $\mathcal{U}$, also denoted by p, with the following semantics:

$$
\begin{array}{lll}
\mathcal{M} \models var(\tau) & \text{if} & \tau \in OVar \\
\mathcal{M} \models ground(\tau) & \text{if} & OVar(\tau) = \emptyset \\
\mathcal{M} \models linear(\tau) & \text{if} & occ(x, \tau) = 1 \text{ for every } x \text{ in } OVar(\tau) \\
\mathcal{M} \models share(\{\tau_1, \ldots, \tau_n\}) & \text{if} & \bigcap_{i=1}^{n} OVar(\tau_i) \neq \emptyset
\end{array}
$$

□

Example 3.3 The assertion $\neg share(\{x, y, z\}) \vee share(\{x, y\})$ is valid in $\mathcal{M}$. In fact, for every valuation α, if $OVar(x\alpha) \cap OVar(y\alpha) \neq \emptyset$ then $\mathcal{M} \models share(\{x, y\})\alpha$, otherwise $\mathcal{M} \models \neg share(\{x, y, z\})\alpha$. □

Note that even if *share* is not first-order (its argument is a set), it can be expressed in first-order logic by means of a family of first-order predicates $share_n$ of rank n, with $n \geq 0$. The set of valid (in $\mathcal{M}$) assertions of $\mathcal{L}$ has been characterized by means of a complete and decidable theory $\mathcal{T}$, by means of a simple axiomatization (see [19]).

The completeness and decidability of $\mathcal{T}$ provides an automatic tool for proving properties of some elements of an abstract domain, in the following way. In order to prove that an element ϕ of a domain satisfies a property P, specified in $\mathcal{L}$ by means of the assertion ψ, it is sufficient to check the validity of the implication $\phi \Rightarrow \psi$.

In order to use $\mathcal{L}$ for the static analysis of logic programs, it is necessary to assume that $\mathcal{U}$ contains the constants and function symbols of the considered class of programs. Moreover, we adopt the notation of the previous section: V denotes the set of (logical) variables representing the considered (program) variables, and $A(\mathcal{L}, V)$ the set of assertions of $\mathcal{L}$ whose free variables are contained in V. Therefore substitutions are identified with (equivalence classes of) valuations. For instance, the substitution $\{x_1/t_1, \ldots, x_n/t_n\}$ is identified with the set of valuations mapping $x_1, \ldots, x_n$ into the object terms $\tau_1, \ldots, \tau_n$ obtained by replacing the variables of the t_i's with the corresponding object variables.

An abstract domain (on $\mathcal{L}$) is specified according to Definition 2.1. Observe that we obtain a more specific notion of abstract domain than the original one (cf. [3]), because of the choice of the assertion language, and because of the condition of closure under variance. For instance, $\{ground(x), true, false\}$ would represent an abstract domain in the original definition, but it is not a legal one in our definition (unless $V = \{x\}$). The condition of closure under variance w.r.t. V has been implicitly assumed in the literature on abstract interpretation of logic programs, but it has not been taken into account when reasoning about these domains using the standard techniques based on Galois insertions or closure operator (cf. [6]).

We conclude this section with a simple example.

Example 3.4 Consider the abstract domain *Con* introduced by Mellish [24] and used in early mode and groundness analyzers [18]. *Con* consists of the bottom element $\bot$, and of the sets $S = \{x_1, \ldots, x_n\}$ of variables of V, with concretization function mapping $\bot$ into $\emptyset$ and $\gamma_{Con}(S) = \{\sigma \mid OVar(x\sigma) = \emptyset \text{ for all } x \in S\}$.

Let $\mathcal{A}_{Con}$ be the set of assertions that are conjunctions of atoms of the form $ground(x)$, with x in V. It is easy to show that $\mathcal{A}_{Con}$ satisfies Definition 2.1, and that *Con* is a representation of $\mathcal{A}_{Con}$, by considering the embedding ϵ_{Con} that maps $\bot$ into *false* and a set $\{x_1, \ldots, x_n\}$ into the assertion $ground(x_1) \wedge \ldots \wedge ground(x_n)$. □

4 Abstract Domains for Ground-Dependency and Aliasing

This section contains a comparative analysis of various abstract domains for the static analysis of logic programs, namely *Def*, *Pos*, *Sharing* and *ASub*. Each of these domains is shown to be the representation of an abstract domain on $\mathcal{L}$. These logical characterizations in $\mathcal{L}$ of the domains are used for deriving their maximal conjunctive factorizations, for studying and comparing the original domains, as well as for defining new ones. Moreover, composite domains that use *Sharing* and *ASub*, called *equations systems*, are investigated. We deal with the disjunctive completion of these domains in the last subsection.

4.1 *Def* in Logical Form

The abstract domain *Def* was introduced by Marriott and Søndergaard for ground-dependency analysis in [22], based on previous work by Dart ([10]) on groundness analysis in deductive databases. We show that *Def* can be factorized into two reduced domains, describing groundness and covering, respectively.

First, we recall the definition of *Def*. *Def* is the largest class of positive boolean functions whose models are closed under intersection, augmented with the bottom element *false*. Recall that a boolean function F is positive if $F(true, \ldots, true) = true$. Here boolean functions are represented by (equivalence classes of) propositional formulas, as e.g. in [22]. In order to define the concretization function γ_{Def}, substitutions are viewed as truth assignments as follows. For a substitution σ, the truth assignment $grounds\sigma$ maps a propositional variable x to *true* iff $x\sigma$ is ground, and to *false* otherwise. Moreover, the notion of instance σ' of a substitution σ is used, meaning that σ' is obtained by composing σ with some substitution. The concretization function γ_{Def} maps an element F of *Def* into the set $\gamma_{Def}(F)$ of those substitutions σ s.t. for every instance σ' of σ, F under the truth assignment $grounds\sigma'$ is *true*. Intuitively, $\gamma_{Def}(F)$ extracts the 'monotonic' (in the sense that its truth is preserved under instantiation) information described by the propositional formula F.

Consider the following abstract domain $\mathcal{A}_{Def}$ on $\mathcal{L}$.

Definition 4.1 $\mathcal{A}_{Def}$ is the set of assertions that are conjunctions of formulas of the form $\forall z\,(var(z) \wedge share(z,x) \Rightarrow share(z,y_1) \vee \ldots \vee share(z,y_n))$, with $n \geq 0$, where $x, y_1, \ldots, y_n$ are in V, and z is a fresh variable. □

We show that *Def* is a representation of $\mathcal{A}_{Def}$, and provide a maximal factorization of $\mathcal{A}_{Def}$.

First, *Def* is characterized in logical form by means of the following transformation. We use the representation of an element F in *Def* as a conjunction of formulas, called definite clauses, of the form $y_1 \wedge \ldots \wedge y_n \rightarrow x$ with $n \geq 0$ (see [10, 2]).

Definition 4.2 The transformation $\epsilon_{Def} : Def \rightarrow \mathcal{L}$ maps F into ϕ_F, defined as follows:

- $\phi_F = \forall z\,(var(z) \wedge share(z,x) \Rightarrow share(z,y_1) \vee \ldots \vee share(z,y_n))$ if $F = y_1 \wedge \ldots \wedge y_n \rightarrow x$.
- $\phi_F = \phi_{F_1} \wedge \ldots \wedge \phi_{F_k}$ if $F = F_1 \wedge \ldots \wedge F_k$, $k \geq 0$, and all the F_i's are definite clauses.

□

Observe that, for $n = 0$ we obtain the assertion $\forall z\,(var(z) \wedge share(z,x) \Rightarrow false)$, that is equivalent to $ground(x)$.

Example 4.3 The element $x \wedge (y \leftrightarrow w)$ is mapped by ϵ_{Def} into the assertion $ground(x) \wedge \forall z\,(var(z) \wedge share(z,w) \Rightarrow share(z,y)) \wedge \forall z\,(var(z) \wedge share(z,y) \Rightarrow share(z,w))$. □

Next, the transformation of Definition 4.2 is shown to be correct.

Lemma 4.4 ϵ_{Def} *is an embedding of* Def *into* $\mathcal{L}$.

Proof. Let F be an element of Def. Let σ be a substitution. We show that σ is in $\gamma_{Def}(F)$ if and only if $\phi_F\sigma$ is true.

Suppose that σ is in $\gamma_{Def}(F)$. Then F under $groundsσ$ is true. For every conjunct $y_1 \wedge \ldots \wedge y_n \rightarrow x$ of F, we have to prove prove that the corresponding conjunct ψ in ϕ_F is true under σ. If $x\sigma$ is ground then $\psi\sigma$ is readily true. Otherwise, suppose by contradiction that $x\sigma$ contains a variable v which does not occur in any term $y_i\sigma$, for $i \in [1, n]$. Consider the substitution θ with domain the set of variables occurring in all the $y_i\sigma$'s, and mapping all variables into ground terms. Since v is not in the domain of θ, we have that $\sigma' = \sigma\theta$ grounds all the y_i's but does not ground x, hence F is false under $groundsσ'$. So σ would not be in $\gamma_{Def}(F)$.

Vice versa, suppose that ϕ_F is true under σ. In order to prove that σ is in $\gamma_{Def}(F)$, we have to show that F under $groundsσ'$ is true, for every instance σ' of σ. Let $\psi = \forall z\,(var(z) \wedge share(z, x) \Rightarrow share(z, y_1) \vee \ldots \vee share(z, y_n))$ be a conjunct of ϕ_F. From ψ true under σ we have that: $OVar(x\sigma) \subseteq \cup_{i=1}^{n} OVar(y_i\sigma)$; moreover, for every instance σ' of σ we have that $OVar(x\sigma') \subseteq \cup_{i=1}^{n} OVar(y_i\sigma')$. So if $x\sigma'$ is not ground then at least one of the $y_i\sigma'$ is not ground. Hence $y_1 \wedge \ldots \wedge y_n \rightarrow x$ is true under $groundsσ'$. □

Finally, using the above Lemma we can prove that Def is a representation of $\mathcal{A}_{Def}$.

Theorem 4.5 $\mathrm{Def} \simeq \mathcal{A}_{\mathrm{Def}}$.

Proof. By Lemma 4.4 F in Def can be characterized by the assertion ϕ_F in $\mathcal{A}_{Def}$.

Vice versa, consider a ϕ in $\mathcal{A}_{Def}$. Consider F defined as the conjunction of definite clauses, s.t. $y_1 \wedge \ldots \wedge y_n \rightarrow x$ occurs in F iff the conjunct $\forall z\,(var(z) \wedge share(z, x) \Rightarrow share(z, y_1) \vee \ldots \vee share(z, y_n))$ occurs in ϕ. It is easy to check that F is in Def, and that $\epsilon_{Def}(F)$ is equivalent to ϕ. □

In order to analyze Def and to compare it with other abstract domains, a maximal factorization of $\mathcal{A}_{Def}$ is given. To this end, we use the following domains. For every $|V| \geq n \geq 0$, consider the domain $\mathcal{A}_{Def^n}$ consisting of the conjunctions of formulas of the form $\forall z\,(var(z) \wedge share(z, x) \Rightarrow share(z, y_1) \vee \ldots \vee share(z, y_n))$, with $y_1, \ldots, y_n$ distinct variables of V. The following result holds.

Lemma 4.6 $\{\mathcal{A}_{\mathrm{Def}^n} \mid n \in [0, |V|]\}$ *is a maximal factorization of* $\mathcal{A}_{\mathrm{Def}}$.

Proof.
First, we prove that every $\mathcal{A}_{Def^n}$ is reduced. For $n = 0$ the result is immediate. For $n > 0$, observe that assertions of the form $\forall z\,(var(z) \wedge share(z, x) \Rightarrow share(z, y_1) \vee \ldots \vee share(z, y_n))$ s.t. x is not in $\{y_1, \ldots, y_n\}$ cannot be decomposed in $\phi_1 \wedge \phi_2$, with ϕ_1, ϕ_2 in $\mathcal{L}$ not equivalent to *true*, because of the presence of the $\forall$ operator. Then, there is only one conjunctive factorization of $\mathcal{A}_{Def^n}$.

Next, we have to check that conditions (a),(b) and (c) of the definition of factorization are satisfied.
(a) Notice that for every $n \geq 0$, the element $\forall z\,(var(z) \wedge share(z, x) \Rightarrow share(z, y_1) \vee \ldots \vee share(z, y_n))$ is true under the valuation that maps all the variables of V into ground terms, but is false under the valuation that maps all the variables of V into distinct variables.
(b) Consider $n, m \geq 0$, and suppose that $n > m$. We have to show that $\mathcal{A}_{Def^m} \cap \mathcal{A}_{Def^n} = \{true, false\}$. By contradiction, assume that ϕ is in the intersection but is neither *true* nor

false. Then, from ϕ in $\mathcal{A}_{Def^n}$, it is a non-empty conjunction of assertions, each of them of the form $\psi = \forall z\,(var(z) \wedge share(z,x) \Rightarrow share(z,y_1) \vee \ldots \vee share(z,y_n))$ with $x \notin \{y_1,\ldots,y_n\}$. But ϕ is also in $\mathcal{A}_{Def^m}$. Moreover, every $\mathcal{A}_{Def^i}$ is reduced. So ψ is equivalent to $\forall z\,(var(z) \wedge share(z,x) \Rightarrow share(z,y_1) \vee \ldots \vee share(z,y_m))$. It is easy to build a valuation σ under which ψ is both true and false: 1) if y is in $\{y_{m+1},\ldots,y_n\}$ then $OVar(y\sigma) = OVar(x\sigma) \neq \emptyset$; 2) $OVar(y_i\sigma) = \emptyset$ for every $i \in [1,m]$.
(c) Follows easily by the definition of $\mathcal{A}_{Def}$.

□

Let $\mathcal{A}_{Def^+} = \wedge_{n\in[1,|V|]}\mathcal{A}_{Def^n}$. A representation of $\mathcal{A}_{Def^+}$ is provided by the set Def^+ of positive boolean functions that can be represented as conjunctions of clauses $y_1 \wedge \ldots \wedge y_n \rightarrow x$, with $n \geq 1$, plus the bottom element *false*, with concretization function the one of Def. Then by Lemma 4.6 it follows that Def is (isomorphic to) the reduced-product of the domain *Con* and Def^+.

It has been recently shown in [6] that Def characterizes the ground-dependency information on V described by the domain *Sharing*. We shall see that this result is easily derived from the logical descriptions of these domains.

4.2 *Pos* in Logical Form

In order to study ground-dependency analysis, the abstract domain *Pos* was introduced by Marriott and Søndergaard [21, 22], consisting of the positive boolean functions, plus the bottom element *false*, with concretization function equal to γ_{Def}.

Consider the following abstract domain $\mathcal{A}_{Pos}$. In the sequel $Q(z,y_1,\ldots,y_n)$ denotes the assertion $share(z,y_1) \vee \ldots \vee share(z,y_n)$.

Definition 4.7 $\mathcal{A}_{Pos}$ is the set of assertions that are conjunctions of formulas of the form $\forall z\,(var(z) \wedge share(z,x_1) \Rightarrow Q(z,y_1,\ldots,y_n)) \vee \ldots \vee \forall z\,(var(z) \wedge share(z,x_m) \Rightarrow Q(z,y_1,\ldots,y_n))$, with $m \geq 1$, and $n \geq 0$, where $x_1,\ldots,x_m,y_1,\ldots,y_n$ are in V, and z is a fresh variable. □

We show that *Pos* is a representation of $\mathcal{A}_{Pos}$, and provide a maximal factorization (on $\mathcal{L}$) of $\mathcal{A}_{Pos}$.

First, *Pos* is characterized in logical form by means of the following transformation. We use the representation of an element F of *Pos* as a conjunction of clauses, of the form $y_1 \wedge \ldots \wedge y_n \rightarrow x_1 \vee \ldots \vee x_m$, $m \geq 1$, $n \geq 0$ (cf. [2]).

Definition 4.8 The transformation $\epsilon_{Pos} : Pos \rightarrow \mathcal{L}$ maps F into ϕ_F, defined as follows:

- $\phi_F = \forall z\,(var(z) \wedge share(z,x_1) \Rightarrow Q(z,y_1,\ldots,y_n)) \vee \ldots \vee \forall z\,(var(z) \wedge share(z,x_m) \Rightarrow Q(z,y_1,\ldots,y_n))$ if $F = y_1 \wedge \ldots \wedge y_n \rightarrow x_1 \vee \ldots \vee x_m$.
- $\phi_F = \phi_{F_1} \wedge \ldots \wedge \phi_{F_k}$ if $F = F_1 \wedge \ldots \wedge F_k$, $k \geq 0$, and all the F_i's are clauses. □

It is easy to check that the above transformation restricted to the elements of Def coincides with ϵ_{Def}.

Example 4.9 The element $x \vee y$ is mapped by ϵ_{Pos} into the assertion $\forall z\,(var(z) \wedge share(z,x) \Rightarrow false) \vee \forall z\,(var(z) \wedge share(z,y) \Rightarrow false)$, equivalent to $ground(x) \vee ground(y)$. □

Next, the transformation of Definition 4.8 is shown to be correct.

Lemma 4.10 *ϵ_{Pos} is an embedding of* Pos *into $\mathcal{L}$.*

Proof. Let F be an element of Pos, and σ a substitution. We prove that σ is in $\gamma_{Pos}(F)$ if and only if $\phi_F\alpha$ is true.

Suppose that σ is in $\gamma_{Pos}(F)$. For every conjunct $y_1 \wedge \ldots \wedge y_n \rightarrow x_1 \vee \ldots \vee x_m$ be a conjunct of F we prove that the corresponding conjunct ψ in ϕ_F is true under α. If $x_i\sigma$ is ground for at least one $i \in [1, m]$, then $\psi\alpha$ is true. Otherwise, we proceed per contradiction. Suppose that every $x_i\alpha$ contains a variable v_i which does not occur in $\cup_{j\in[1,n]} OVar(y_j\sigma)$. Consider the substitution θ with domain $\cup_{j\in[1,n]} OVar(y_j\sigma)$, mapping every variable in the domain into a ground term. Observe that $v_1, \ldots, v_m$ are not in the domain of θ. Consider $\sigma' = \sigma\theta$: it grounds all the y_j's but does not ground any of the x_i's. Hence F is false under $grounds\sigma'$. Contradiction.

Vice versa, suppose that ϕ_F is true under σ. In order to prove that σ is in $\gamma_{Pos}(F)$, we have to show that F under $grounds\sigma'$ is true, for every instance σ' of σ. Consider a conjunct $\psi = \forall z\,(var(z) \wedge share(z, x_1) \Rightarrow Q(z, y_1, \ldots, y_n)) \vee \ldots \vee \forall z\,(var(z) \wedge share(z, x_m) \Rightarrow Q(z, y_1, \ldots, y_n))$ of ϕ_F: it is true under σ, therefore $OVar(x_j\sigma) \subseteq \cup_{i=1}^{n} OVar(y_i\sigma)$ for at least one $j \in [1, m]$. For every instance σ' of σ we have that $OVar(x_j\sigma') \subseteq \cup_{i=1}^{n} OVar(y_i\sigma')$. Then $y_1 \wedge \ldots \wedge y_n \rightarrow x_1 \vee \ldots \vee x_m$ is true under $grounds\sigma'$, because if $x_j\sigma'$ is not ground then at least one of the $y_i\sigma'$ is not ground.

□

Finally, using Lemma 4.10, we can prove that Pos is a representation of $\mathcal{A}_{Pos}$.

Theorem 4.11 $\mathrm{Pos} \simeq \mathcal{A}_{\mathrm{Pos}}$.

Proof. From Lemma 4.10 it follows that F in Pos can be characterized by the assertion ϕ_F in $\mathcal{A}_{Pos}$.

Vice versa, consider a ϕ in $\mathcal{A}_{Pos}$. Let F be the conjunction of definite clauses, s.t. $y_1 \wedge \ldots \wedge y_n \rightarrow x_1 \vee \ldots \vee x_m$ occurs in F iff the conjunct $\forall z\,(var(z) \wedge (share(z, x_1) \vee \ldots \vee share(z, x_m)) \Rightarrow share(z, y_1) \vee \ldots \vee share(z, y_n))$ occurs in ϕ. It is easy to check that F is in Pos, and that $\epsilon_{Pos}(F)$ is equivalent to ϕ. □

In order to give a maximal factorization of $\mathcal{A}_{Pos}$, we use the decomposition of $\mathcal{A}_{Def}$, and the following domains. For every $|V| \geq n \geq 0$ and $|V| \geq m \geq 2$, consider the domain ${\mathcal{A}_{Pos}}^{m,n}$ consisting of the conjunctions of formulas of the form $\forall z\,(var(z) \wedge share(z, x_1) \Rightarrow Q(z, y_1, \ldots, y_n)) \vee \ldots \vee \forall z\,(var(z) \wedge share(z, x_m) \Rightarrow Q(z, y_1, \ldots, y_n))$ with $x_1, \ldots, x_m$ and $y_1, \ldots, y_n$ distinct variables of V. The following result holds.

Lemma 4.12 *$\{{\mathcal{A}_{\mathrm{Def}}}^n, {\mathcal{A}_{\mathrm{Pos}}}^{m,n} \mid n \in [0, |V|], m \in [2, |V|]\}$ is a maximal factorization of $\mathcal{A}_{\mathrm{Pos}}$.*

Proof. The proof is similar to the one of Lemma 4.6. So, we only show that the domains ${\mathcal{A}_{Pos}}^{m,n}$'s are 'disjoint'. The proof is by contradiction.
Let $(m_1, n_1) \neq (m_2, n_2)$ Assume that ϕ is in the intersection but is neither *true* nor *false*. ϕ is in ${\mathcal{A}_{Pos}}^{m_1,n_1}$, so it contains a conjunct $\psi = \forall z\,(var(z) \wedge share(z, x_1) \Rightarrow Q(z, y_1, \ldots, y_{n_1})) \vee \ldots \vee \forall z\,(var(z) \wedge share(z, x_{m_1}) \Rightarrow Q(z, y_1, \ldots, y_{n_1}))$ where $\{x_1, \ldots, x_{m_1}\} \cap \{y_1, \ldots, y_{n_1}\} = \emptyset$.

But ϕ is also in $\mathcal{A}_{Pos^{m_2,n_2}}$. Suppose $m_2 > m_1$ (the proof for the other case is analogous). Then every conjunct ψ_i of ϕ contains one variable occurring free on the left-hand side of $\Rightarrow$, say w_i, that does not belong to $\{x_1, \ldots, x_{m_1}\}$. Consider a valuation σ s.t.: 1) $OVar(w_i\sigma) = \emptyset$ for every w_i; 2) all the other variables are mapped into distinct variables. Then $\psi\sigma$ is false. However, from condition 1) every ψ_i is true under σ. Contradiction. The proof for the other case,namely when $m_1 = m_2$ and $n_1 > n_2$ (or $n_2 > n_1$) is similar to the proof of (b) of Lemma 4.6, where one replaces x by $x_1, \ldots, x_{m_1}$.

□

Let $\mathcal{A}_{Pos^\vee} = \wedge_{n\in[0,|V|],m\in[2,|V|]}\mathcal{A}_{Pos^{m,n}}$. A representation of $\mathcal{A}_{Pos^\vee}$ is provided by the set $Pos^\vee$ of positive boolean functions that can be represented as conjunctions of clauses $y_1 \wedge \ldots \wedge y_n \rightarrow x_1 \vee \ldots \vee x_m$, with $n \geq 0, m \geq 2$, plus the bottom element *false*, with concretization function the one of *Pos*. Then by Lemma 4.12 it follows that *Pos* is (isomorphic to) the reduced-product of the domains *Con*, Def^+ and $Pos^\vee$. It has been shown in [8] that *Def* is properly contained in *Pos*. Lemma 4.12 characterizes logically the other part of *Pos*.

4.3 *Sharing* in Logical Form

In order to study information on the possible sharing among abstract variables, an abstract domain extensively used in abstract interpretation is the domain *Sharing* by Jacobs and Langen [15]. *Sharing* is the set of sets $\Delta \in 2^{2^V}$ s.t. if $\Delta \neq \emptyset$ then $\emptyset \in \Delta$. Its concretization function $\gamma_{Sharing}$ maps an element Δ of *Sharing* into the set $\gamma_{Sharing}(\Delta)$ of those substitutions σ whose approximation set $A(\sigma)$ is an element of Δ. The approximation set $A(\sigma)$ consists of all the sets $occ(\sigma, x) = \{v \mid v \text{ in the domain of } \sigma \text{ s.t. } x \text{ occurs in } v\sigma\}$, for all the variables x occurring in the range of σ.

Consider the following abstract domain $\mathcal{A}_{Sharing}$.

Definition 4.13 $\mathcal{A}_{Sharing}$ is the set of assertions of $\mathcal{L}$ that are conjunctions of formulas of the form $\forall z\,(var(z) \wedge share(z, x_1) \wedge \ldots \wedge share(z, x_m) \Rightarrow share(z, y_1) \vee \ldots \vee share(z, y_n))$ with $m \geq 1$, $n \geq 0$, where $x_1, \ldots, x_m, y_1, \ldots, y_k$ are in V, and z is a fresh variable. □

We show that *Sharing* is a representation of $\mathcal{A}_{Sharing}$, and provide a maximal factorization (on $\mathcal{L}$) of $\mathcal{A}_{Sharing}$.

First, *Sharing* is characterized in logical form by means of the following transformation. In the sequel, for the sake of simplicity, we write $share(x, S)$ instead of $share(\{x\} \cup S)$.

Definition 4.14 The transformation $\epsilon_{Sharing}$ maps Δ into the assertion

$$\phi_\Delta = \bigwedge_{S\subseteq V} \forall z\,(var(z) \wedge share(z, S) \Rightarrow share(z, S_1) \vee \ldots \vee share(z, S_k)),$$

with $\{S_1, \ldots, S_k\} = \{S' \mid S' \in \Delta \text{ s.t. } S \subseteq S'\}$.

□

Let ϕ_S denote the conjunct of ϕ_Δ corresponding to the subset S of V.

Observe that if S is not contained in any set of Δ, then ϕ_S is the assertion $\forall z\,(var(z) \wedge share(z, S) \Rightarrow false)$, which says that the variables of S can only be bound to terms sharing no variables. If S is a singleton, say $S = \{x\}$, then ϕ_S describes information on ground-dependency for x. Indeed, it is not difficult to see that in this case ϕ_S can be rewritten

into an assertion of $\mathcal{A}_{Def}$. The other assertions ϕ_S, for S not singleton and $k > 0$, describe information about sharing of sets containing at least three variables.

Example 4.15 Consider $\Delta = \{\emptyset, \{x\}, \{x, y\}, \{y, z\}\}$, and $V = \{x, y, z\}$. Then ϕ_Δ is (equivalent to) $\neg share(x, z) \wedge \neg share(\{x, y, z\}) \wedge \forall v (var(v) \wedge share(v, y) \Rightarrow share(v, z) \vee share(v, x)) \wedge \forall v (var(v) \wedge share(v, z) \Rightarrow share(v, y))$. □

Next, the correctness of this transformation is shown.

Lemma 4.16 $\epsilon_{\text{Sharing}}$ *is an embedding of* Sharing *into* $\mathcal{L}$.

Proof. Let Δ an element of *Sharing*, and let σ a substitution. We show that σ is in $\gamma(\Delta)$ if and only if ϕ_Δ is true under σ.

Suppose that σ is in $\gamma(\Delta)$. By definition of $\gamma(\Delta)$, for every $S \subseteq V$, if the terms of $S\sigma$ share at least one variable v then $S \subseteq occ(\sigma, v)$, and $occ(\sigma, v)$ is in Δ. Hence ϕ_S is true under σ.

Vice versa, suppose that ϕ_Δ is true under σ. In order to prove that σ is in $\gamma(\Delta)$, we have to show that for every v in the range of σ, $occ(\sigma, v)$ is in Δ. By hypothesis $\phi_{occ(\sigma,v)} = \forall z (var(z) \wedge share(z, occ(\sigma, v)) \Rightarrow share(z, S_1) \vee \ldots \vee share(z, S_k))$ is true under σ. Moreover $k \geq 1$, since v occurs in every term of $occ(\sigma, v)\sigma$. Observe that $occ(\sigma, v)$ is the biggest set S of variables in V s.t. v occurs in $x\sigma$, for every $x \in S$. Then $occ(\sigma, v) = S_i$ for some $i \in [1, k]$. □

Finally, Lemma 4.16 is used to prove that *Sharing* is a representation of $\mathcal{A}_{Sharing}$.

Theorem 4.17 Sharing $\simeq \mathcal{A}_{\text{Sharing}}$.

Proof. Consider Δ in *Sharing*. By Lemma 4.16, it is characterized by the assertion ϕ_Δ of the form $\wedge_{S \subseteq V} \forall z (var(z) \wedge share(z, S) \Rightarrow share(z, S_1) \vee \ldots \vee share(z, S_k))$. It is not difficult to prove that ϕ_Δ is equivalent to the conjunction of the formulas $\forall z (var(z) \wedge share(z, x_1) \wedge \ldots \wedge share(z, x_m) \Rightarrow share(z, y_1) \vee \ldots \vee share(z, y_k))$, for all $(y_1, \ldots, y_k)$ occurring in $S_1 \times \ldots \times S_k$. The proof consists of a manipulation of the assertion by means of standard first-order logic equivalences, together with the equivalence of $share(z, S)$ and $\wedge_{x \in S} share(z, x)$. Thus ϕ_Δ is in $\mathcal{A}_{Sharing}$.

Vice versa, it is easy to prove that a ϕ in $\mathcal{A}_{Sharing}$ is equivalent to the assertion $\epsilon_{Sharing}(\Delta)$ for a suitable Δ. ϕ is manipulated, by means of standard first-order logic equivalences, together with the equivalence of $share(z, S)$ and $\wedge_{x \in S} share(z, x)$, in order to obtain an assertion of the form $\wedge_{S \subseteq V} \forall z (var(z) \wedge share(z, S) \Rightarrow share(z, S_1) \vee \ldots \vee share(z, S_k))$. This assertion is $\epsilon_{Sharing}(\Delta)$, for Δ consisting of the sets S_i that occur in the right hand side of the implications. □

In order to give a maximal factorization of $\mathcal{A}_{Sharing}$, we use the following domains. For every $|V| \geq n \geq 0$ and $|V| \geq m \geq 1$, consider the domain $\mathcal{A}_{Sharing^{m,n}}$ consisting of the conjunctions of formulas of the form $\forall z (var(z) \wedge share(z, x_1) \wedge \ldots \wedge share(z, x_m) \Rightarrow share(z, y_1) \vee \ldots \vee share(z, y_n))$, with $x_1, \ldots, x_m$ and $y_1, \ldots, y_n$ distinct variables of V. The following result holds.

Lemma 4.18 $\{\mathcal{A}_{\text{Sharing}^{m,n}} \mid n \in [0, |V|], m \in [1, |V|]\}$ *is a maximal factorization of* $\mathcal{A}_{\text{Sharing}}$.

Proof. The proof is similar to the one for the decomposition of *Pos*. □

Consider the abstract domain $Sharing^+$ introduced in [6], containing as elements the empty set, and the sets Δ^+ of the form $\Delta \cup T$, with Δ in *Sharing* and $T = \{\{x\} \mid x \in V\} \cup \{\emptyset\}$. One can prove that $Sharing^+$ is a representation of $\bigwedge_{m\geq 2, n\geq 0} \mathcal{A}_{Sharing}{}^{m,n}$. Moreover, *Def* is a representation of $\bigwedge_{n\geq 0} \mathcal{A}_{Sharing}{}^{1,n}$. Therefore, by Lemma 4.18 it follows that *Sharing* is (isomorphic to) the reduced product of $Sharing^+$,Def^+ and *Con*.

4.4 *ASub* in Logical Form

The pair-sharing domain *ASub* was introduced by Søndergaard [26] for sharing and linearity analysis. Its elements are pairs (G, R) where the first component is a subset of V, and the second one is a symmetric binary relation on V, s.t. $(G \times V) \cap R = \emptyset$. Moreover, the element $\bot$, representing the empty set of substitutions, is in *ASub*. Its concretization function γ_{ASub} maps an element (G, R) of *ASub* into the set of substitutions σ s.t. for all (x, y) in V: (i) x in G implies $x\sigma$ ground; (ii) x, y distinct and $OVar(x\sigma) \cap OVar(y\sigma) \neq \emptyset$ implies (x, y) in R; (iii) $(x, x) \notin R$ implies $x\sigma$ linear.

Consider the following abstract domain $\mathcal{A}_{ASub}$.

Definition 4.19 $\mathcal{A}_{ASub}$ is the set of assertions that are conjunctions of literals of the form $ground(x)$, $\neg share(x, y)$, and $linear(x)$, with x, y in V. □

We show that *ASub* is a representation of $\mathcal{A}_{ASub}$, and provide a maximal factorization of $\mathcal{A}_{ASub}$.

First, *ASub* is characterized in logical form by means of the following transformation.

Definition 4.20 The transformation ϵ_{ASub} maps $\bot$ into *false*, and (G, R) into the assertion $\phi_{(G,R)} = \phi_1 \wedge \phi_2 \wedge \phi_3$, where:

1. ϕ_1 is the conjunction of the atoms $ground(x)$, for all x in G.
2. ϕ_2 is the conjunction of the literals $\neg share(x, y)$, for all (x, y) not in R with x, y distinct.
3. ϕ_3 is the conjunction of the atoms $linear(x)$, for all (x, x) not in R. □

Assertions ϕ_1, ϕ_2 and ϕ_3 characterize *ASub* in logical form, by means of its information on groundness, independence, and linearity, respectively.

Example 4.21 Consider the element (G, R) of *ASub*, with $G = \{x\}$ and $R = \{(y, z), (z, z), (z, w)\}$ and suppose that $V = \{x, y, z, w\}$. Then $\phi_{(G,R)}$ is (equivalent to) $ground(x) \wedge linear(y) \wedge linear(w) \wedge \neg share(y, w)$. □

Next, this transformation is shown to be correct.

Lemma 4.22 ϵ_{ASub} *is an embedding of* ASub *into* $\mathcal{L}$.

Proof. Let (G, R) be a pair-sharing. Let σ be a substitution. We show that σ is in $\gamma_{ASub}(G, R)$ if and only if $\phi_{(G,R)}$ is true under σ.

Suppose that σ is in $\gamma_{ASub}(G, R)$. We have to show that each conjunct ψ of $\phi_{(G,R)}$ is true under σ. We distinguish three cases according to those of Definition 4.20. If $\psi = ground(x)$

then by 1. x is in G, hence by (i) $ground(x)$ is true under σ. If $\psi = \neg share(x, y)$ then by 2. $x \neq y$ and (x, y),(y, x) not in R, hence by (ii) $OVar(x\sigma) \cap OVar(y\sigma) = \emptyset$. If $\psi = linear(x)$ then by 3. (x, x) is not in R, hence by (iii) $x\sigma$ is linear.

Vice versa, suppose that $\phi_{(G,R)}$ is true under σ. We show that σ satisfies (i)–(iii). From 1. it follows that (i) holds. Assume now that $x \neq y$ and $OVar(x\sigma) \cap OVar(y\sigma) \neq \emptyset$. Then from 2. it follows that (x, y) is in R. Finally, (iii) follows from 3. □

Lemma 4.22 is used to prove that *ASub* is a representation of $\mathcal{A}_{ASub}$.

Theorem 4.23 ASub $\simeq \mathcal{A}_{\text{ASub}}$.

Proof. We have already shown in Lemma 4.22 that (G, R) in *ASub* can be characterized by the assertion $\phi_{(G,R)}$ in $\mathcal{A}_{ASub}$.

Vice versa, consider a ϕ in $\mathcal{A}_{ASub}$. The pair (G, R) is defined as follows: x is in G if there is a conjunct of ϕ of the form $ground(x)$; (x, y) is in R if $\neg share(x, y)$ does not occur in ϕ; and (x, x) is in R if $linear(x)$ occurs in ϕ. It is easy to check that (G, R) is in *ASub*. □

In order to give a maximal factorization of $\mathcal{A}_{ASub}$, the domain $\mathcal{A}_{Linear}$ is used, consisting of the conjunctions of atoms the form $linear(x)$, with x in V.

Lemma 4.24 $\{\mathcal{A}_{\text{Sharing}^{m,0}}, \mathcal{A}_{\text{Linear}} \mid m \in [1, 2]\}$ *is a maximal factorization of* $\mathcal{A}_{\text{ASub}}$.

Proof. It is sufficient to show that $\mathcal{A}_{Linear}$ satisfies (a), since the rest of the proof is similar to the one of Lemma 4.16. Consider the element $linear(x)$ of $\mathcal{A}_{Linear}$. Then the valuation that maps x into a ground term satisfies the assertion, while the one mapping x into the term $f(y, y)$ does not satisfy the assertion. Observe that the last result is based on the assumption that *Fun* contains one functor of rank greater or equal than 2. □

4.5 Abstract Equations Systems

A recent proposal, called abstract equation systems (cf. [4, 25]), considers composite domains defined using *Sharing* or *ASub*. In this proposal, elementary properties are specified by means of a lattice *An* of annotations. For instance, the authors consider the annotations lattice consisting of the three elements $f \leq l \leq a$, where f means free, l stands for linear, and a stands for any term. Moreover, a sharing component Δ is used, which is either *Sharing* or *ASub*. It is easy to characterize abstract equation systems in $\mathcal{L}$. In abstract equations systems, the distinction between abstract variables and variables is used. This corresponds to the distinction between logical and object variables used in our logical framework. Each annotation of *An*, augmented with the bottom element $\perp$, corresponds to a reduced abstract domain on $\mathcal{L}$. For example, l corresponds to the abstract domain $\mathcal{A}_{Linear}$ introduced in the previous section, f to $\mathcal{A}_{Free}$, consisting of the conjunctions of atoms of the form $var(x)$, and a to $\{true, false\}$.

4.6 Disjunctive Completions

The logical characterizations on $\mathcal{L}$ of the domains for ground-dependency and aliasing show that there is no 'disjunctive' information incorporated into these domains, except for $\mathcal{A}_{Pos}$. This is formalized in the following result.

Proposition 4.25 *The abstract domains* $\mathcal{A}_{\mathrm{Def}}$, $\mathcal{A}_{\mathrm{Sharing}}$ *and* $\mathcal{A}_{\mathrm{Asub}}$ *are* $\vee$*-reduced. Moreover,* $\vee\mathcal{A}_{\mathrm{Def}} = \mathcal{A}_{\mathrm{Pos}}$.

Proof. The first result follows from the fact that the disjunctive normal form of any assertion in one of the three considered abstract domains is equal to its conjunctive normal form.

The inclusion $\vee\mathcal{A}_{Def} \subseteq \vee\mathcal{A}_{Pos}$ follows from Lemma 4.12. The converse inclusion follows by observing that an assertion of $\mathcal{A}_{Pos}$ is a conjunction of assertions in $\vee\mathcal{A}_{Def}$, and by computing the disjunctive normal form. □

A similar result on *Pos* has already been given in [14], using the approach based on closure operators. In [13] it is shown that the disjunctive completion of *Pos* is strictly better than *Pos*. This result is considered somewhat surprising: indeed, one would expect that an element $\{F_1, \ldots, F_k\}$ of the disjunctive completion can be represented by the propositional formula $F_1 \vee \ldots \vee F_k$. This confusion is caused by the fact that the interpretation of a formula given by the concretization function is not equal to the interpretation in propositional logic, while we are used to interpret the logical connectives according to their standard semantics (in first-order logic). This confusion does not arise when one uses the logical framework introduced in this paper for the design and the reasoning phase: indeed, it is easy to show that $\mathcal{A}_{Pos}$ is not closed under disjunction, hence (by Proposition 2.10) it is not equivalent to its disjunctive completion.

5 Related Work

The standard techniques used for the comparative study of the properties represented by abstract domains are based on two equivalent approaches: Galois connections and closure operators.

In the original approach ([3]), comparison of abstract domains is performed by means of the notion of abstraction, where an abstract domain is more abstract than another one if there is a Galois insertion from the first into the latter. This notion is weakened in [9], where the comparison is defined w.r.t. a given property, by means of the notion of quotient of one abstract domain w.r.t. another one, describing the part of the former abstract domain that is useful for computing the information described by the latter one. A similar analysis is possible by means of our framework, where the domain and the property are first specified in the logic, and next factorized. Then the reduced product of the common factors of the domains corresponds to the quotient of the domain w.r.t. the property.

The approach based on closure operators has been employed in two recent works [6, 14], that investigate the 'inverse' of the two important operators on abstract domains, namely the reduced product and the disjunctive completion, respectively. In [6], this approach is used for investigating domain complementation in abstract interpretation, a kind of inverse of the reduced product. The authors formalize the concept of decomposition of an abstract domain, as a set of abstract domains whose reduced product yields the initial abstract domain and use the notion of pseudo-complement for decomposing abstract domains. The distinguishing feature of our approach from this work is the use of an assertion language for describing the properties of interest, and the explicit role of the set of considered program variables (or more in general of syntactic objects). As a consequence, domains are decomposed by inspecting the form of their assertions. The relative definition of conjunctive factorization is rather intuitive, since it resembles the notion of factorization of integers into pairwise prime

factors: it is always applicable, and the resulting factors are 'disjoint'. This is not the case for the method based on the notion of psuedo-complement: the notion of domain decomposition is introduced, that amounts to condition (c) of Definition 2.5, and the pseudo-complement $D \sim C$ of a domain D w.r.t. another domain C is used to provide the (binary) decomposition $(D \sim C, C)$ of D, where the factors are not necessarily 'disjoints'.

In [14], the approach based on closure operators is used for investigating the inverse of the operation of disjunctive completion. They introduce the notion of disjunctive basis for an abstract domain as the most abstract domain inducing the same disjunctive completion, and study the disjunctive basis of typical abstract domains used in abstract interpretation of functional and logic programming. In this paper we have introduced a similar notion, called $\vee$-reduced domain (on $\mathcal{L}$). The main difference is that here $\mathcal{L}$ determines the granularity of the $\vee$-reduced domains, while in [14] all the closure operators on the concrete domain are considered. Moreover, the fact that $\mathcal{L}$ is a first-order assertion language guarantees that the disjunctive completion of a domain is equal to the disjunctive completion of a $\vee$-reduced domain.

The abstract domains analyzed in Section 4 have been extensively studied in previous work. In [8] it is proven that the part of *Sharing* describing groundness dependencies is contained in *Pos*. In [6] this result is strengthen by showing that this part coincides with *Def*, and that $Sharing^+$ is the pseudo-complement of *Def* in *Sharing*. In this paper these results are directly derived from the logical characterization of *Sharing*. Moreover, we have obtained the finest (in $\mathcal{L}$) decomposition of *Sharing*. Finally, the factors of this decomposition have been used for other purposes, e.g. for comparing the expressiveness of the abstract domains.

The classes of Boolean functions used to represent *Def* and *Pos* have been analyzed in [7, 2]. The difference from these works is that they focus on the representation, while we focus on the design and reasoning, by considering a syntactic characterization of $\gamma_{Def}(Def)$ in first-order logic.

6 Conclusion

In this paper a simple framework based on first-order logic has been proposed for the design of abstract domains for static analysis. The correspondent of the operations of reduced product and disjoint completion of abstract domains have been defined in the logical framework. Moreover, the notion of conjunctive factorization has been introduced, for decomposing abstract domains in 'disjoint' parts. The usefulness of this framework has been illustrated by analyzing typical abstract domains used in abstract interpretation of logic programs.

The framework can also be used for defining operators on abstract domains. For instance, an important operator in abstract interpretation of logic programs is the 'projection' away from the variables that are not in V. The projection operation corresponds to existential quantification ([23]). As one would expect, all the domains considered in Section 4 are closed under existential quantification. The existential closure $\exists x\,(\phi)$ w.r.t. a variable x of a domain ϕ in $\mathcal{A}_{\mathcal{D}}$ (where $\mathcal{D}$ stands for one of the domains considered in Section 4) is the domain obtained from ϕ by deleting all the conjuncts containing at least one free occurrence of x. For example, if $\phi = ground(w) \wedge \forall v(var(v) \wedge share(v, y) \Rightarrow share(v, z) \vee share(v, x))$ then $\exists x(\phi)$ is (equivalent to) $ground(w)$.

Finally, abstract domains in logical form can be used for proving the correctness and optimality of a representation. For instance, in [5] a function called *Reduce* is used which

yields the minimal representative of an element of the reduced product of $ASub$ and $Sharing$, given an arbitrary representative. $Reduce : ASub \times Sharing \rightarrow ASub \times Sharing$ maps an element $\langle (G, R), \Delta \rangle$ into $\langle (G', R'), \Delta' \rangle$ where $\Delta' = \{S \in \Delta \mid S \cap G = \emptyset, Pairs(S) \subseteq R\}$, $R' = R \cap (\cup_{S \in \Delta'} S \times S)$, $G' = \{x \in V \mid x \notin S \text{ for every } S \in \Delta'\}$, and $Pairs(S) = \{(x, y) \in S \times S \mid x, y \text{ distinct}\}$.

The logical representation of these domains can be used to prove that this definition is correct and optimal, (i.e. it provides the minimal representation). In fact, correctness amounts to prove that the following is an equivalence (in $\mathcal{M}$):

$$\epsilon_{ASub}(G, R) \wedge \epsilon_{Sharing}(\Delta) \Leftrightarrow \epsilon_{ASub}(G', R') \wedge \epsilon_{Sharing}(\Delta').$$

Optimality amounts to prove the following two conditions:

1. for every x in V:

   $\epsilon_{ASub}(G', R') \Rightarrow ground(x)$ iff $\epsilon_{Sharing}(\Delta') \Rightarrow ground(x)$;

2. for every distinct variables x, y of V:

   $\epsilon_{ASub}(G', R') \Rightarrow \neg share(x, y)$ iff $\epsilon_{Sharing}(\Delta') \Rightarrow \neg share(x, y)$.

The proof of the above statements is not difficult, using the definitions of $\epsilon_{Sharing}$ and ϵ_{ASub}.

We conclude by mentioning some interesting topics for future work. The specific framework for logic programming could be applied for proving the correctness of abstract unification algorithms. This could be done by describing the unification by means of a suitable predicate transformer on $\mathcal{L}$, in the style of [25], and by defining a transformation which reduces the considered abstract unification algorithm to an instance of the predicate transformer. However, this is not an easy task, for it is already difficult to design a specific correct abstract unification algorithm (see e.g. [4]).

Another interesting topic that seems worth of investigation, is the study of the relationship between abstract interpretations and proof methods. This topic has been tackled in the functional programming setting, where a domain-theoretic approach is used in [17] for proving that strictness analysis by abstract interpretation and non-standard type inference are equivalent. For logic programming, our framework could be used for defining a program logic for the comparison of data-driveness analysis using type inference (cf. e.g. [1]) and abstract interpretation (cf. [23]).

Acknowledgements

This work was partially supported by the Netherlands Computer Science Research Foundation, with financial support from the Netherlands Organisation for Scientific Research 'NWO'. I would like to thank Livio Colussi, Tino Cortesi, Gilberto Filé, Maurizio Gabbrielli, Massimo Marchiori, and Catuscia Palamidessi for stimulating discussions on the subject of this paper; and Jan Rutten for his encouragement.

References

[1] K.R. Apt and E. Marchiori. Reasoning about Prolog programs: from Modes through types to assertions. In Formal Aspects of Computing, vol. 6A, pag. 743-764, 1994.

[2] T. Armstrong, K. Marriott, P. Schachte and H. Søndergaard. Boolean Functions for Dependency Analysis: Algebraic Properties and Efficient Representation. In B. Le Charlier ed., *Static Analysis: Proceedings of the First International Symposium*, Springer-Verlag, LNCS 864, pp. 266-280, 1994.

[3] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Conference Record of the 6th ACM Symposium on Principles of Programming Languages (POPL '79)*, pages 269–282. ACM Press, 1979.

[4] M. Codish, D. Dams, G. Filé and M. Bruynooghe. Freeness analysis for logic programs - and correctness?. *Proceedings of the 10th International Conference on Logic Programming (ICLP '93)*, ed. D.S. Warren, The MIT Press, pag. 116–131, 1993.

[5] M. Codish, A. Mulkers, M. Bruynooghe, M. García de la Banda, and M. Hermenegildo. Improving abstract interpretations by combining domains. *ACM Transactions on Programming Languages and Systems*, 17(1):28–44, 1995.

[6] A. Cortesi, G. Filé, R. Giacobazzi, C. Palamidessi, and F. Ranzato. Complementation in abstract interpretation. In A. Mycroft, editor, *Proceedings of the 2nd International Static Analysis Symposium (SAS '95)*, LNCS Vol. 983, pages 100–117. Springer-Verlag, 1995.

[7] A. Cortesi, G. Filé, and W. Winsborough. Prop revised: propositional formula as abstract domain for groundness analysis. In G. Kahn, editor, *Proceedings of the 6th Annual IEEE Symposium on Logic in Computer Science (LICS '91)*, pages 322–327, 1991.

[8] A. Cortesi, G. Filé, and W. Winsborough. Comparison of abstract interpretations. In W. Kuich, editor, *Proceedings of the 19th International Colloquium on Automata, Languages and Programming (ICALP '92)*, LNCS Vol. 623, pages 521–532. Springer-Verlag, 1992.

[9] A. Cortesi, G. Filé, and W. Winsborough. The quotient of an abstract interpretation. Technical Report 12/94, Dipartimento di Matematica Pura ed Applicata, Università di Padova, 1994.

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

[11] S. Debray. On the Complexity of Dataflow Analysis of Logic Programs. *Proceedings of the 19th International Coll. on Automata, Languages and Programming (ICALP '92)*, Springer Verlag, pag. 509-520, LNCS 623, 1992.

[12] B.A. Davey and H.A. Priestley. Introduction to Lattices and Order. Cambridge University Press, 1990.

[13] G. Filé and F. Ranzato. Improving Abstract Interpretation by Systematic Lifting to the Powerset. In M. Bruynooghe, editor, *Proceedings of the 1994 International Symposium on Logic Programming (ILPS'94)*, The MIT Press, pages 655-669, 1994.

[14] R. Giacobazzi and F. Ranzato. Compositional Optimization of Disjunctive Abstract Interpretations. In H.R. Nielson, editor, *Proceedings of the 1996 European Symposium on Programming (ESOP'96)*, Springer-Verlag, LNCS 1058, pages 141-155, 1996.

[15] D. Jacobs and A. Langen. Accurate and efficient approximation of variable aliasing in logic programs. In E.L. Lusk and R.A. Overbeek, editors, *Proceedings of the 1989 North American Conference on Logic Programming (NACLP '89)*, pages 154–165. The MIT Press, 1989.

[16] D. Jacobs and A. Langen. Static analysis of logic programs for independent AND-parallelism. *Journal of Logic Programming*, 13(2,3):154–165, 1992.

[17] T.P. Jensen. Strictness Analysis in Logical Form. *Proceedings of the Conference on Functional Programming Languages and Computer Architectures*, Springer Verlag, pag. 352-366, LNCS 523, 1991.

[18] N.D. Jones and H. Søndergaard. A Semantics-Based Framework for the Abstract Interpretation of Prolog. *Abstract Interpretation of Declarative Languages*, eds. S. Abramsky and C. Hankin, Ellis Horwood, Chichester, U.K., pag. 123-142, 1987.

[19] E. Marchiori. A Logic for Variable Aliasing in Logic Programs. *Proceedings of the 4th International Conference on Algebraic and Logic Programming (ALP '94)*, eds. G. Levi and M. Rodriguez-Artalejo, Springer Verlag, pag. 287-304, LNCS 850, 1994.

[20] E. Marchiori. Prime Factorizations of Abstract Domains Using First-Order Logic. *Proceedings of the 5th International Conference on Algebraic and Logic Programming (ALP '96)*, Springer Verlag, to appear, 1996.

[21] K. Marriott and H. Søndergaard. Notes for a tutorial on abstract interpretation of logic programs. *North American Conference on Logic Programming*, 1989.

[22] K. Marriott and H. Søndergaard. Precise and efficient groundness analysis for logic programs. *ACM Letters on Programming Languages and Systems*, 2(1–4):181–196, 1993.

[23] K. Marriott, H. Søndergaard and N.D. Jones. Denotational abstract interpretation of logic programs. *ACM Transactions on Programming Languages and Systems*, ACM-TOPLAS, 16(3):607–648, 1994.

[24] C. Mellish. Some Global Optimizations for a Prolog Compiler. *The Journal of Logic Programming*, 2(1):43–66, 1985.

[25] A. Mulkers, W. Simoens, G. Janssens and M. Bruynooghe. On the Practicality of Abstract Equation Systems. *Proceedings of the International Conference on Logic Programming (ICLP '95)*, ed. L. Sterling, MIT Press, pag. 781–795, 1995.

[26] H. Søndergaard. An Application of Abstract Interpretation of Logic Programs: Occur Check Reduction. *Proceedings of the European Symposium on Programming (ESOP '86)*, eds. B. Robinet and R. Wilhelm, Springer Verlag, pag. 327-338, LNCS 213, 1986.