An Algebraic Approach to Sharing Analysis of Logic Programs

Michael Codish, Vitaly Lagoon

Department of Mathematics and Computer Science,
Ben-Gurion University of the Negev,
PoB 653 Beer-Sheva 84105, Israel.
E-mail: `codish@cs.bgu.ac.il`, `lagoon@verisity.com`.

Francisco Bueno

Universidad Politécnica de Madrid (UPM),
Facultad de Informática,
28660 Boadilla del Monte, Madrid, Spain.
E-mail: `bueno@fi.upm.es`

Abstract

This paper describes an algebraic approach to the sharing analysis of logic programs based on an abstract domain of *set logic programs*. Set logic programs are logic programs in which the terms are sets of variables and unification is based on an associative, commutative, and idempotent equality theory. All of the basic operations required for sharing analyses, as well as their formal justification, are based on simple algebraic properties of set substitutions and set-based atoms. An ordering on set-based syntactic objects, similar to "less general" on concrete syntactic objects, is shown to reflect the notion of "less sharing" information. The (abstract) unification of a pair of set-based terms corresponds to finding their most general ACI1 unifier with respect to this ordering. The unification of a set of equations between set-based terms is defined exactly as in the concrete case, by solving the equations one by one and repeatedly applying their solutions to the remaining equations. We demonstrate that all of the operations in a sharing analysis have natural definitions which are both correct and optimal.

1 Introduction

Two or more variables in a logic program are said to be *aliased* if in some execution of the program they are bound to terms which contain a common variable. A variable in a logic program is said to be *ground* if it is bound to a

Preprint submitted to Elsevier Preprint 10 November 1998

ground term in every execution of the program. A variable is said to be *linear* if it is bound to a linear term in every execution of the program. Aliasing, groundness and linearity information, often called *sharing* information in the logic programming community, provide the basis for a wide range of program optimizations and other useful applications. Such information can be used to identify circumstances in which the occur check may be safely dispensed with [33,35] or to determine run-time goal independence which can be used to eliminate costly run-time checks in and-parallel execution of logic programs [32,25,24]. In the context of concurrent logic programming languages, sharing information can be used to identify *single-writer* properties (e.g., structures which are constructed by a single process). Though most of these applications focus on aliasing and groundness information, the availability of linearity information is also useful for improving the precision of sharing analyses.

This paper presents a novel algebraic approach for the sharing analysis of logic programs using *set logic programs*. The terms in a set logic program are sets of variables. Atoms contain sets of variables instead of terms. Standard unification is replaced by a suitable unification for sets based on the well studied notion of ACI1-unification [2,29]. Namely, unification in the presence of an associative, commutative, and idempotent equality theory with a unit element (the empty set). Sharing analyses are semantic based and formalized in terms of an abstract domain consisting of set-based atoms and substitutions. The variables in such atoms specify information about possible aliasing and definite groundness and linearity in the corresponding argument positions of concrete atoms. Substitutions are *set substitutions*, which are mappings from variables to sets of variables.

All of the basic operations required for sharing analyses, as well as their formal justification, are based on simple algebraic properties of set substitutions and set-based atoms. The *composition* of set substitutions, *application* of a set substitution to an atom, and the *projection* of a set substitution to a set of relevant variables, all maintain their standard definitions (just as for standard substitutions). An ordering on set-based syntactic objects (similar to "less general" on concrete syntactic objects) reflects the notion of "less sharing". As a consequence, sharing is downwards closed with respect to this ordering. The unification of a pair of set-based terms corresponds to finding their most general ACI1 unifier with respect to this ordering. The unification of a set of equations between set-based terms is defined exactly as in the concrete case, by solving the equations one by one and repeatedly applying their solutions to the remaining equations. We demonstrate that all of the operations in a sharing analysis have natural definitions which are both correct and optimal.

Our approach has several additional advantages over previous proposals for sharing analysis of logic programs: (1) The abstract substitutions in our domain are like substitutions and can hence be applied to other syntactic objects.

This facilitates the implementation, supporting an approach which combines program abstraction (replacing terms by sets of variables) with concrete evaluation (enhanced by ACI1-unification). This approach is derived from ideas presented in [23,14,22] and often termed *abstract compilation*. It has been applied in a variety of applications [20,21,9,11]. To our knowledge, no previous work has provided an abstract compilation scheme for sharing information. (2) Most of the recent work on sharing analyses for logic programs attempts to justify the correctness of the proposed abstract operations (e.g., unification) by mimicking the behavior of a suitable corresponding concrete algorithm (for example as in [7,8]). In contrast, in this paper we focus on algebraic properties (e.g., of a most general unifier). For example, we first characterize the algebraic properties of an abstract most general unifier in the context of a sharing analysis, and then provide an abstract unification algorithm and prove that it computes an object satisfying the required properties.

One of the more widely applied sharing analyses reported in the literature is the so called *set-sharing* analysis due to Jacobs and Langen [25], first implemented by Muthukumar and Hermenegildo [32]. This analysis plays a central role in the And-Parallel Prolog compiler described in [24]. The analysis of Jacobs and Langen as well as many of its extensions are developed within the framework of *abstract interpretation* [16] which provides the basis for a semantic approach to dataflow analysis. In this paper, we show that the domain of set substitutions is isomorphic to the set-sharing domain of Jacobs and Langen and argue that set logic programs provide a natural and intuitive means for describing correct and optimal set-sharing analyses. A contribution of our presentation is thus an optimal sharing analysis for the domain of Jacobs and Langen obtained through the domain isomorphism. To our knowledge, no previous work has provided optimality results for sharing analysis.

The rest of this paper is structured as follows: Section 2 introduces preliminary definitions and presents some properties of (standard) substitutions and atoms which provide the foundation for the proposed sharing analysis. Section 3 presents the syntax of *set logic programs* with which we construct the domains of abstract atoms and abstract substitutions for sharing analysis. The abstract domains are detailed in Section 4 and their operations are described in Section 5. For convenience in presentation, these sections focus on groundness and aliasing information only, which is denoted, in a broad sense, *set-sharing*. Section 6 proves that the domains based on set logic programs are isomorphic to the well known set-sharing domain of Jacobs and Langen. Section 7 provides an example of bottom-up sharing analysis constructed on the basis of our abstract domain and the well-known s-semantics. Section 8 illustrates the extension of the domains with linearity information. Finally, Section 9 concludes. This paper is an extended version of [12].

2 Preliminaries

This section introduces some preliminary definitions and fixes the notation which will be used throughout. In addition, we introduce several properties of substitutions and atoms which provide the background and basic intuition for the set-sharing analyses developed in this paper. Of particular interest are the non-standard orderings on syntactic objects for which set-sharing and linearity information are downwards closed properties.

In the following we assume a familiarity with the standard definitions and notation for logic programs as described in [30,1]. For a set of function symbols Σ and variables $\mathcal{V}$, we let $T(\Sigma, \mathcal{V})$ denote the set of terms constructed using symbols from Σ and variables from $\mathcal{V}$. The set of atoms constructed using predicate symbols from Π and terms from $T(\Sigma, \mathcal{V})$ is denoted by $\mathcal{B}_{\mathcal{V}}$. The set of variables occurring in a syntactic object s is denoted $vars(s)$. We say that a term t is ground, denoted $ground(t)$, if $vars(t) = \emptyset$. The term t is linear, denoted $linear(t)$, if all variables in t have single occurrences.

A *substitution* θ is a mapping from $\mathcal{V}$ to $T(\Sigma, \mathcal{V})$ which acts as the identity function except for a finite set of variables, i.e. its domain $dom(\theta) = \left\{ x \in \mathcal{V} \,\middle|\, x\theta \neq x \right\}$ is finite. The range of a substitution is defined as $range(\theta) = \left\{ x\theta \,\middle|\, x \in dom(\theta) \right\}$. The composition of substitutions θ and ψ is defined as usual and denoted $\theta \circ \psi$. A substitution ψ is *idempotent* if $\psi \circ \psi = \psi$ (or equivalently if $dom(\psi) \cap range(\psi) = \emptyset$). The set of idempotent substitutions is denoted Sub. The empty (identity) substitution is denoted ε. The projection of a substitution θ on a set of variables D, denoted by $\theta\lceil_D$, is defined as usual. A substitution extends to apply to any syntactic object in the usual way.

The standard "less general" ordering on terms is denoted $t_1 \leq t_2$. Recall that a term t_1 is less general than t_2 if there exists a substitution θ such that $t_1 = t_2\theta$. Similarly, $\theta_1 \leq \theta_2$ denotes that a substitution θ_1 is less general than θ_2 which is the case if there exists a substitution θ such that $\theta_1 = \theta_2 \circ \theta$.

A set S is downward-closed with respect to a given order relation $\sqsubseteq$ if $s \in S$ and $s' \sqsubseteq s$ implies $s' \in S$.

A *unifier* of two terms t_1 and t_2 is a substitution θ such that $t_1\theta = t_2\theta$. A unifier θ is said to be a *most general* unifier (mgu) of t_1 and t_2 if $\psi \leq \theta$ for any other unifier ψ of t_1 and t_2. This definition extends for other syntactic objects such as atoms.

We typically consider syntactic objects "modulo the naming of variables". The elements of $\wp(\mathcal{B}_{\mathcal{V}})$, i.e., sets of atoms, modulo variable renaming, are called interpretations. Given an equivalence class (induced by renaming) of syntactic objects and a finite set of variables V, it is always possible to find a representative of the class which contains no variables from V. Let I be a set of (equivalence classes of) syntactic objects and let s be a syntactic object. Then, $A \ll_s I$ denotes that A is a renaming (representative) of an element of I which does not share variables with s. Furthermore, we extend this to specify tuples of renamed apart syntactic objects:

$$\langle A_1, \ldots, A_n \rangle \ll_s I \Leftrightarrow \bigwedge_{i=1}^{n} (A_i \ll_s I) \;\wedge\; \bigwedge_{i \neq j} (vars(A_i) \cap vars(A_j) = \emptyset)$$

Abstract Interpretation: We assume the standard framework of abstract interpretation [16] in which a program analysis is viewed as a non-standard, abstract semantics defined over a domain of data descriptions. An abstract semantics is constructed by replacing operations in a suitable concrete semantics with corresponding abstract operations defined on data descriptions. Program analyses are defined by providing finitely computable abstract interpretations which preserve interesting aspects of program behavior. Formal justification of program analyses is reduced to proving conditions on the relation between data and data descriptions and on the elementary operations defined on the data descriptions. Abstract interpretations are formalized in terms of *Galois insertions.* A Galois insertion is a quadruple (A, α, B, γ) where:

(1) $(A, \sqsubseteq_A)$ and $(B, \sqsubseteq_B)$ are complete lattices of *concrete* and *abstract domains* respectively;
(2) $\alpha : A \to B$ and $\gamma : B \to A$ are monotonic functions called *abstraction* and *concretization functions* respectively; and
(3) $a \sqsubseteq_A \gamma(\alpha(a))$ and $\alpha(\gamma(b)) = b$ for every $a \in A$ and $b \in B$.

Domain Isomorphism: We say that two abstract domains B_1 and B_2 of a concrete domain A are *isomorphic* if they describe the same properties of the concrete domain and are equally precise [17]. From the formal point of view the corresponding Galois insertions $\langle A, \alpha_1, B_1, \gamma_1 \rangle$ and $\langle A, \alpha_2, B_2, \gamma_2 \rangle$ are isomorphic if

(1) there exists a *set isomorphism* between the underlying posets B_1 and B_2 provided by a bijective function $f : B_1 \mapsto B_2$ and its inverse $f^{-1} : B_2 \mapsto B_1$;
(2) there is an *order isomorphism* (also called *order-embedding*) between B_1 and B_2, i.e., for any pair of elements $b_1, b_1' \in B_1$, $b_1 \sqsubseteq_{B_1} b_1'$ implies $f(b_1) \sqsubseteq_{B_2} f(b_1')$, or equivalently for any pair of elements $b_2, b_2' \in B_2$, $b_2 \sqsubseteq_{B_2} b_2'$ implies $f^{-1}(b_2) \sqsubseteq_{B_1} f^{-1}(b_2')$;
(3) the closure operators $\gamma_1 \circ \alpha_1$ and $\gamma_2 \circ \alpha_2$ are equivalent.

Set-Sharing: We say that a set of variables S occurs in a substitution θ through the variable v, if S is (exactly) the set of variables in the domain of θ which are mapped to terms containing v. If S occurs in θ through some variable v then we say that S is a set of variables which *share* under θ. The sharing of sets of variables is usually (informally) understood with respect to a finite *domain of variables of interest* $D \subseteq \mathcal{V}$. We shall assume, without loss of generality, that the domain of interest is the domain of the substitution being considered. As in [25], we provide the following definition for the notion of occurrence:

$$\begin{aligned} &occs : Sub \times \mathcal{V} \to \wp(\mathcal{V}) \\ &occs(\theta, v) = \left\{ \, x \in D \,\middle|\, v \in vars(x\theta) \, \right\}. \end{aligned} \tag{1}$$

The set-sharing of a substitution θ, denoted $\mathcal{A}(\theta)$, is the set of sets of variables which share under θ:

$$\begin{aligned} &\mathcal{A} : Sub \to \wp(\wp(\mathcal{V})) \\ &\mathcal{A}(\theta) = \left\{ \, occs(\theta, v) \,\middle|\, v \in \mathcal{V} \, \right\}. \end{aligned} \tag{2}$$

Observe that a variable $x \in D$ does not occur in any of the sets in $\mathcal{A}(\theta)$ if and only if θ maps x to a ground term.

Similar to the notion of set-sharing for substitutions we consider also the sharing of variables between the argument positions of an atom. We say that a set of (integer) argument positions N occurs in an atom p through the variable v, if N is (exactly) the set of argument positions of p which contain the variable v. If N occurs in p through some v then we say that N is a set of arguments which *share* in p. The following definitions are straightforward extensions of Equations (1) and (2):

$$\begin{aligned} &occs' : \mathcal{B}_{\mathcal{V}} \times \mathcal{V} \to \wp(\mathcal{N}) \\ &occs'(p(t_1, \ldots, t_n), v) = \left\{ \, 1 \leq i \leq n \,\middle|\, v \in vars(t_i) \, \right\}. \end{aligned} \tag{3}$$

The set-sharing for an atom a, denoted $\mathcal{A}'(a)$, is the set of sets of argument positions which share in p:

$$\begin{aligned} &\mathcal{A}' : \mathcal{B}_{\mathcal{V}} \mapsto \wp(\wp(\mathcal{N})) \\ &\mathcal{A}'(a) = \left\{ \, occs'(a, v) \,\middle|\, v \in \mathcal{V} \, \right\}. \end{aligned} \tag{4}$$

In fact, set-sharing for substitutions and for atoms represent the same kind of information considering that an atom $a(t_1, \ldots, t_n)$ can be represented as a pair $a(x_1, \ldots, x_n), \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ where $x_1, \ldots, x_n$ are fresh distinct variables not occurring in $t_1, \ldots, t_n$.

Linearity: Traditionally, information about aliasing of variables is augmented by *linearity*. The linear argument positions in an atom a or a substitution θ are denoted $linearity(a)$ and $linearity(\theta)$ respectively and defined by:

$$
\begin{aligned}
&linearity(p(t_1, \ldots, t_n)) = \left\{ \, 1 \leq i \leq n \, \middle| \, linear(t_i) \, \right\} \\
&linearity(\theta) = \left\{ \, x \in dom(\theta) \, \middle| \, linear(x\theta) \, \right\}
\end{aligned}
\tag{5}
$$

Orderings for Sharing Analysis: Two special types of substitutions are of particular interest for our presentation.

(1) We say that a substitution ψ is an *independent-range* substitution (*ir*-substitution for short) if

$$\forall x, y \in dom(\psi).\ (x \neq y) \ \Rightarrow\ (vars(x\psi) \cap vars(y\psi) = \emptyset). \tag{6}$$

(2) We say that ψ is a *linear* substitution if ψ is an independent-range substitution which maps variables to linear terms.

Independent-range and linear substitutions are of interest as they do not introduce additional set-sharing and non-linearity to the syntactic objects to which they are applied. To formalize this we introduce two partial orders on atoms (and other syntactic objects):

(1) We say that $a_1 \leq_{ir} a_2$ (a_1 precedes a_2 in the ir-ordering) if there exists an ir-substitution ψ on the variables of a_2 such that $a_1 = a_2\psi$. Similarly, $\theta_1 \leq_{ir} \theta_2$, assuming without loss of generality the domain of interest $D = dom(\theta_1) = dom(\theta_2)$ [1], if there exists an ir-substitution ψ on the range of θ_2 such that $\theta_1 = (\theta_2 \circ \psi)\restriction_D$.

(2) We say that $a_1 \leq_{lin} a_2$ (a_1 precedes a_2 in the lin-ordering) if there exists a linear substitution ψ on the variables of a_2 such that $a_1 = a_2\psi$. Similarly, $\theta_1 \leq_{lin} \theta_2$ assuming without loss of generality the domain of interest $D = dom(\theta_1) = dom(\theta_2)$, if there exists a linear substitution ψ on the range of θ_2 such that $\theta_1 = (\theta_2 \circ \psi)\restriction_D$.

[1] We always may extend θ_1 and/or θ_2 with "renaming mappings" of the form $V \mapsto V'$ in such a way that this assumption holds.

The relation between these two orderings and sharing properties is clarified by the following lemma:

Lemma 2.1 *For substitutions θ, θ', such that $dom(\theta) = dom(\theta') = D$ and atoms a, a':*

(1) $\theta \leq_{ir} \theta' \Rightarrow \mathcal{A}(\theta) \subseteq \mathcal{A}(\theta')$
(2) $a \leq_{ir} a' \Rightarrow \mathcal{A}'(a) \subseteq \mathcal{A}'(a')$
(3) $\theta \leq_{lin} \theta' \Rightarrow linearity(\theta) \supseteq linearity(\theta')$
(4) $a \leq_{lin} a' \Rightarrow linearity(a) \supseteq linearity(a')$

PROOF. We prove (1) and (3). The proofs of (2) and (4) are similar.

If $\theta \leq_{ir} \theta'$ then there exists an ir-substitution ψ on the range of θ' such that $\theta = (\theta' \circ \psi)\lceil_D$. Consider a member of $\mathcal{A}(\theta' \circ \psi)$. It is of the form: $occs(\theta' \circ \psi, v) = \left\{ x \in D \,\middle|\, v \in vars(x\theta'\psi) \right\}$ for some variable v. Since ψ is an ir-substitution and $vars(range(\theta')) = dom(\psi)$, there is exactly one variable $v' \in vars(range(\theta'))$ mapped by ψ to a term containing v. Thus, the set of variables occurring in $\theta'\psi$ through v occur in θ' through v', i.e., $occs(\theta' \circ \psi, v) = occs(\theta', v')$. Therefore, $occs(\theta' \circ \psi, v) \in \mathcal{A}(\theta')$, or equivalently $\mathcal{A}(\theta) \subseteq \mathcal{A}(\theta')$.

If $\theta \leq_{lin} \theta'$ then there exists a linear substitution ψ on the range of θ' such that $\theta = (\theta' \circ \psi)\lceil_D$. For a variable $x \in D$, the substitution ψ maps all variables in $x\theta'$ to linear terms which have no variables in common (since ψ is also an ir-substitution). Thus, linearity of $x\theta'$ implies linearity of $x\theta'\psi$. $\square$

Corollary 2.2 *The properties of linearity and (the complement of) set-sharing are downwards closed with respect to $\leq_{lin}$ and $\leq_{ir}$ respectively.*

Note that the orderings $\leq_{ir}$ and $\leq_{lin}$ capture the notions of "less aliasing" and "more linearity", respectively. Observe also that $\mathcal{A}(\theta) \subseteq \mathcal{A}(\theta')$ implies that θ grounds more variables than θ'. Thus, "less sharing" means also (possibly) "more groundness".

Example 1 *Consider the following substitutions:*

$$\begin{aligned}
\theta_1 &= \left\{ A \mapsto P, B \mapsto Q, C \mapsto t(R,R) \right\} \\
\theta_2 &= \left\{ A \mapsto f(X,Y),\ B \mapsto g(Y,Z),\ C \mapsto t(W,V) \right\} \\
\theta_3 &= \left\{ A \mapsto f(F,a),\ B \mapsto g(a,G),\ C \mapsto t(H,H) \right\} \\
\theta_4 &= \left\{ A \mapsto f(a,I),\ B \mapsto g(J,J),\ C \mapsto t(b,b) \right\}
\end{aligned}$$

Assume the domain of variables of interest $D = \{A, B, C\}$. The substitutions θ_1, θ_3 and θ_4 are independent-range but non-linear due to non-linear terms $t(R, R)$ and $t(H, H)$ and $g(J, J)$ in their ranges. The substitution θ_2 is not independent-range since it introduces sharing between A and B through Y. As a result it is also non-linear even though all terms in its range are linear.

Now consider the standard ordering between these substitutions:

- $\theta_3 \leq \theta_1$ *provided by* $\theta_3 = (\theta_1 \circ \psi_{1\to3})\restriction_D$, *where*

$$\psi_{1\to3} = \left\{ P \mapsto f(F, a),\ Q \mapsto g(a, G),\ R \mapsto H \right\}$$

- $\theta_4 \leq \theta_1$ *provided by* $\theta_4 = (\theta_1 \circ \psi_{1\to4})\restriction_D$, *where*

$$\psi_{1\to4} = \left\{ P \mapsto f(a, I),\ Q \mapsto g(J, J),\ R \mapsto b \right\}$$

- $\theta_3 \leq \theta_2$ *provided by* $\theta_3 = (\theta_2 \circ \psi_{2\to3})\restriction_D$, *where*

$$\psi_{2\to3} = \left\{ X \mapsto F,\ Y \mapsto a,\ Z \mapsto G,\ W \mapsto H,\ V \mapsto H \right\}$$

The substitution $\psi_{1\to3}$ is linear; $\psi_{1\to4}$ is independent-range; and $\psi_{2\to3}$ is not an independent-range substitution due to the aliasing of W and V provided by variable H. Thus, in addition to the standard ordering we can say that $\theta_3 \leq_{lin} \theta_1$ and $\theta_4 \leq_{ir} \theta_1$. Of course, $\theta_3 \leq_{lin} \theta_1$ implies also $\theta_3 \leq_{ir} \theta_1$.

Unification with Linear Terms: The following property of most general unifiers when linear terms are involved is useful when designing an abstract unification algorithm in the presence of linearity information.

Let t_1 and t_2 be terms. We say that a variable x in the equation $t_1 = t_2$ is *co-linear* if $vars(t_1) \cap vars(t_2) = \emptyset$ and either $x \in vars(t_2)$ and $linear(t_1)$ or $x \in vars(t_1)$ and $linear(t_2)$.

Lemma 2.3 *Let t and t' be terms. Then, the projection of $mgu(t, t')$ on the co-linear variables of the equation $t = t'$ is a linear substitution.*

PROOF. Let $\bar{t} = \langle t_1, \ldots, t_n \rangle$ and $\bar{t}' = \langle t'_1, \ldots, t'_n \rangle$ be tuples of terms such that $\bar{t}$ is linear and $vars(\bar{t}) \cap vars(\bar{t}') = \emptyset$. We show that the projection of $\theta = mgu(\bar{t}, \bar{t}')$ on $vars(\bar{t}')$ is a linear substitution. Define the depth factor of a tuple $\langle t_1, \ldots, t_n \rangle$ to be a pair (d, n) where d is the maximal depth of a term in $\langle t_1, \ldots, t_n \rangle$. The proof is by induction on the depth factor of $\langle t_1, \ldots, t_n \rangle$ (the linear tuple).

For the base case, the depth factor is $(0,1)$ which means that the unification is of the form $mgu(X, t')$ or of the form $mgu(a, t')$ (i.e. the left argument is either a variable or a constant). In both cases the projection of the most general unifier on $vars(t')$ is trivially a linear substitution.

Assume now that the lemma holds for all tuples of linear terms with a depth factor less than (d, n) (in the standard enumeration for pairs). Consider a unification of the form $\theta = mgu(\langle t_1, \ldots, t_n\rangle, \langle t'_1, \ldots, t'_n\rangle)$ such that the depth factor of $\langle t_1, \ldots, t_n\rangle$ is (d, n). Observe that $\theta = \psi \circ \varphi$ where $\psi = mgu(t_1, t'_1)$ and $\varphi = mgu(\langle t_2, \ldots, t_n\rangle, \langle t'_2, \ldots, t'_n\rangle\psi)$. Note that ψ does not affect the variables in $\langle t_2, \ldots, t_n\rangle$ which does not share any variables with t_1 nor t'_1. Both unification problems have left arguments which are linear and have depth factors which are smaller than (d, n). Hence by the induction hypothesis ψ and φ have linear projections on $vars(t'_1)$ and $vars(\langle t'_2, \ldots, t'_n\rangle)$. This implies that their composition θ has a linear projection on $vars(\langle t'_1, \ldots, t'_n\rangle)$. $\square$

The following example demonstrates that Lemma 2.3 does not hold if we relax the requirement that $vars(t_1) \cap vars(t_2) = \emptyset$.

Example 2 *Consider the unification of the linear term* $t_1 = f(A, B)$ *with* $t_2 = f(g(X, X), A)$ *which involves a common variable* A. *The most general unifier of* t_1 *and* t_2 *is* $\theta = \left\{ A \mapsto g(W, W),\ B \mapsto g(W, W),\ X \mapsto W \right\}$. *Observe that the projection of* θ *on* $vars(t_2)$ *is not linear.*

Note that if both t_1 and t_2 are linear then Lemma 2.3 implies that $mgu(t_1, t_2)$ is a linear substitution.

3 Set Logic Programs

The sharing analyses described in this paper are constructed using a first order language, similar to that of logic programs, which we call *set logic programs*. Intuitively, set logic programs are logic programs in which the terms are sets of variables. This section introduces the syntactic constructs for set logic programs. Namely, the set-based notions of terms, atoms and substitutions. As these form the basis for an abstract domain they are referred to as abstract terms, abstract atoms and abstract substitutions. The definitions and functionality of these entities resemble closely those of the corresponding concrete syntactic elements.

Abstract Terms and Atoms

Syntactically, we assume a set of variables $\mathcal{V}$ and an underlying alphabet $\Sigma^{\oplus} = \{\oplus, \emptyset\}$ consisting of a single binary function symbol $\oplus$ which "glues" elements together and a single constant symbol $\emptyset$ to represent the empty set. Abstract terms, or *set expressions*, are elements of the term algebra $T(\Sigma^{\oplus}, \mathcal{V})$ modulo an equality theory consisting of the following axioms:

$$
\begin{array}{ll}
(x \oplus y) \oplus z = x \oplus (y \oplus z) & (\textit{associativity}) \\
x \oplus y = y \oplus x & (\textit{commutativity}) \\
x \oplus x = x & (\textit{idempotence}) \\
x \oplus \emptyset = x & (\textit{unit element})
\end{array}
$$

This equality theory is sometimes referred to as ACI1 and the corresponding equivalence relation on terms denoted $=_{ACI1}$. This notion of equivalence suggests that abstract terms can be viewed as flat sets of variables. For example, the terms $x_1 \oplus x_2 \oplus x_3$, $x_1 \oplus x_2 \oplus x_3 \oplus \emptyset$, and $x_1 \oplus x_2 \oplus x_3 \oplus x_2$ can each be viewed as representing the set $\{x_1, x_2, x_3\}$ of three variables. In the following we do not distinguish between set expressions and sets of variables, often referring to the set of variables in a term as a set expression. Abstract atoms are entities of the form $p(\tau_1, \ldots, \tau_n)$ where $p/n \in \Pi$ and $\tau_1, \ldots, \tau_n$ are abstract terms.

Abstract Substitutions

Abstract substitutions, or *set substitutions*, are substitutions which map variables of $\mathcal{V}$ to abstract terms from $T(\Sigma^{\oplus}, \mathcal{V})$. We denote the set of idempotent abstract substitutions by $Sub^{\oplus}$. The application of an abstract substitution μ to an abstract term τ is defined as usual by replacing occurrences of each variable x in τ by the abstract term $x\mu$. The standard operations on abstract substitutions such as projection and composition are also defined just as for usual substitutions. Abstract independent-range substitutions and a corresponding partial order on abstract terms, atoms and substitutions are defined as in the concrete case. Namely, an abstract substitution is said to be independent-range if it satisfies the condition of Equation (6). For abstract atoms π_1 and π_2, we say that $\pi_1 \preceq_{ir} \pi_2$ if there exists an abstract independent-range substitution ψ on the variables of π_2 such that $\pi_1 = \pi_2\psi$. Similarly, for abstract substitutions μ_1 and μ_2 such that $D = dom(\mu_1) = dom(\mu_2)$, $\mu_1 \preceq_{ir} \mu_2$, if there exists an independent-range substitution ψ on the range of μ_2 such that $\mu_1 = (\mu_2 \circ \psi)\restriction_D$.

These preorders induce corresponding equivalence relations on abstract atoms and substitutions and partial orders on the equivalence classes. We say that the abstract atoms (or substitutions) π_1 and π_2 are ir-equivalent, denoted by $\pi_1 \approx_{ir} \pi_2$ if $\pi_1 \preceq_{ir} \pi_2$ and $\pi_2 \preceq_{ir} \pi_1$.

Note that similar to the case of concrete syntactic objects, abstract substitutions considered together with predicate names are equivalent to abstract atoms. In Section 6 we use this property for establishing an isomorphism between our domain for sharing analysis (based on atoms) and the domain of Jacobs and Langen (based on substitutions).

The set of abstract atoms modulo ir-equivalence is denoted $[\mathcal{B}_{\mathcal{V}}^{\oplus}]_{\approx_{ir}}$. We often write by abuse of notation $\mathcal{B}_{\mathcal{V}}^{\oplus}$ instead of $[\mathcal{B}_{\mathcal{V}}^{\oplus}]_{\approx_{ir}}$ and denote the equivalence class $[\pi]_{\approx_{ir}}$ by π. We also denote the equivalence of abstract atoms $\pi_1 \approx_{ir} \pi_2$ by equality $\pi_1 = \pi_2$ because the corresponding equivalence classes $[\pi_1]_{\approx_{ir}}$ and $[\pi_2]_{\approx_{ir}}$ are identical in this case.

Intuitively, the orders $\preceq_{ir}$ on abstract atoms and substitutions reflect a notion of "less sharing" similar to the corresponding orders on concrete objects described in Section 2. In fact, it is straightforward to apply the definitions of set-sharing and the results of Lemma 2.1 from Section 2 also to abstract atoms and substitutions.

Observation 1 *The statements of Lemma 2.1 apply also to abstract atoms and substitutions.*

When constructing the abstract domains a stronger result will be obtained: $\pi_1 \preceq_{ir} \pi_2$ implies that the concrete objects *described by* π_1 contain less sharing than the concrete objects *described by* π_2. However, this is better delayed until the appropriate definitions have been introduced.

Example 3 *Consider the following abstract atoms:*

$$\begin{array}{ll} \pi_1 = p(\{A,B\},\{B,C\},\{A,B,D\}), & \pi_2 = p(\{X\},\{X,Y\},\{X,Z\}), \\ \pi_3 = p(\{U\},\{V\},\{U,W\}) & \pi_4 = p(\{F\},\emptyset,\{F\}) \end{array}$$

The first and the third arguments of π_1 share through A, while all three arguments share through B. The second and third arguments of π_1 contain independent variables (C and D respectively) which are not shared with other arguments. In π_2 all three arguments share through X, and in π_3 the first and the third arguments share. In π_4 also the first and the third arguments share, however in contrast to π_3 the third argument contains no independent variables and the second argument is ground. Thus, π_1 contains more set-sharing than each of π_2, π_3 and π_4. In our domain this is captured as $\pi_2 \preceq_{ir} \pi_1$, $\pi_3 \preceq_{ir} \pi_1$ and $\pi_4 \preceq_{ir} \pi_1$. This is because

$\pi_2 = \pi_1\psi_1$, $\pi_3 = \pi_1\psi_2$ and $\pi_4 = \pi_1\psi_3$ where:

$$\psi_1 = \left\{ A \mapsto \emptyset,\ B \mapsto X,\ C \mapsto Y,\ D \mapsto Z \right\},$$
$$\psi_2 = \left\{ A \mapsto U,\ B \mapsto \emptyset,\ C \mapsto V,\ D \mapsto W \right\},$$
$$\psi_3 = \left\{ A \mapsto F,\ B \mapsto \emptyset,\ C \mapsto \emptyset,\ D \mapsto \emptyset \right\}$$

Note also that $\pi_4 \preceq_{ir} \pi_3$ with $\pi_4 = \pi_3\psi_4$ where $\psi_4 = \left\{ U \mapsto F,\ V \mapsto \emptyset,\ W \mapsto \emptyset \right\}$.

The following observation considers the case when $\pi_1 \preceq_{ir} \pi_2$ and $\pi_1 \not\approx_{ir} \pi_2$. It follows that there exists a ground (abstract) substitution ψ such that $\pi_1 = \pi_2\psi$.

Observation 2 *For abstract atoms π_1 and π_2 such that $\pi_1 \not\approx_{ir} \pi_2$ and $\pi_1 \preceq_{ir} \pi_2$ there exists a variable z in $vars(\pi_2)$ such that $\pi_1 \preceq_{ir} \pi_2 \cdot \left\{ z \mapsto \emptyset \right\}$. This is because the variables are meant to represent possible aliasing of the atom arguments. If π_1 and π_2 belong to different ir-equivalence classes, and $\pi_1 \preceq_{ir} \pi_2$, then π_2 represents more sharing than π_1. Therefore, π_2 has at least one variable more than π_1. By grounding this variable π_2 may still represent more sharing, or the same sharing than π_1. A similar result holds for abstract substitutions and implies that if $\mu_1 \not\approx_{ir} \mu_2$ and $\mu_1 \preceq_{ir} \mu_2$ then there exists a variable z in $range(\mu_2)$ such that $\mu_1 \preceq_{ir} \left(\mu_2 \circ \left\{ z \mapsto \emptyset \right\}\right) \upharpoonright_D$, assuming $dom(\mu_1) = dom(\mu_2) = D$.*

Example 4 *Figure 1 depicts the lattice of abstract atoms constructed using a predicate symbol $p/2 \in \Pi$, ordered by the $\preceq_{ir}$ relation. Note that $p(\{A, B\}, \{A, C\})$ is the most general atom (with respect to this ordering) in the lattice, and not $p(\{B\}, \{C\})$. This fact reflects the main difference between $\preceq_{ir}$-ordering and the standard ordering of syntactic objects. In the $\preceq_{ir}$-ordering an atom containing all possible set-sharing is the most general among all comparable atoms. Note that for each pair of abstract atoms connected by an edge, the lower atom can be obtained by applying a ground substitution (which binds a single variable to $\emptyset$) to the upper atom.*

It is important for the sharing analysis and interesting on its own right that the equivalence of abstract atoms partitions $\mathcal{B}^{\oplus}_{\mathcal{V}}$ into a finite number of equivalence classes (assuming of course a finite set Π). This result guarantees finite approximations and terminating analyses in our domain as we will see in the following.

Theorem 3.1 *$[\mathcal{B}^{\oplus}_{\mathcal{V}}]_{\approx_{ir}}$ is finite.*

PROOF. It suffices to prove that for any predicate symbol p/n the number of associated equivalence classes of abstract atoms $[p(\tau_1, \ldots, \tau_n)]_{\approx_{ir}}$ is finite.

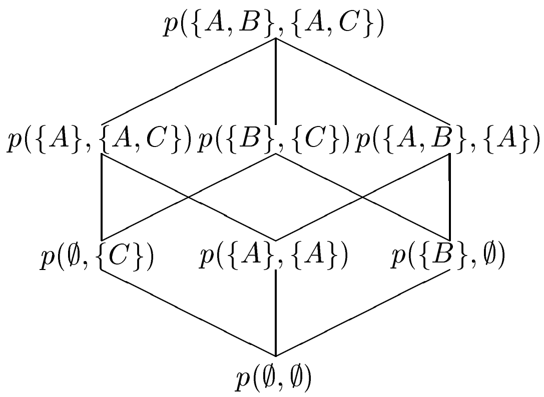

Fig. 1. Abstract atoms constructed using $p/2 \in \Pi$ ordered by $\preceq_{ir}$.

We prove the claim by demonstrating that each equivalence class of the form $[p(\tau_1,\ldots,\tau_n)]_{\approx_{ir}}$ has a representative containing at most 2^n-1 variables. Assume an atom π has more than 2^n-1 distinct variables. Then there are at least two variables x and y occurring in exactly the same set of argument positions of π. Consider the atom $\pi' = \pi \cdot \{x \mapsto \emptyset,\ y \mapsto z\}$ where z is a fresh variable. By construction we have $\pi' \preceq_{ir} \pi$ and it is easy to see that $\pi \preceq_{ir} \pi'$ with $\pi = \pi' \cdot \{z \mapsto x \oplus y\}$. Thus, π and π' are in the same equivalence class and $|vars(\pi')| = |vars(\pi)| - 1$. It follows that for any atom having two or more variables in the same set of argument positions we can find an equivalent atom with a smaller set of variables. So, for any atom constructed using p/n there exists an equivalent atom with all variables occurring in distinct subsets of argument positions, i.e., an atom with at most 2^n-1 variables. $\Box$

Theorem 3.1 demonstrates that for any equivalence class of abstract atoms there exists a "minimal" representative with a bound number of abstract variables. This representative is canonical up to renaming of abstract variables. In the following we assume that such a canonical representative of the corresponding equivalence class is considered.

Example 5 *Note that* $p(X \oplus Y \oplus Xs,\ Ys,\ X \oplus Y \oplus Z) \approx_{ir} p(X' \oplus Xs,\ Ys,\ X' \oplus Z)$ *where the equivalence is provided by the pair of independent-range substitutions:* $\psi_{\preceq_{ir}} = \{X' \mapsto X \oplus Y\}$ *and* $\psi_{\succeq_{ir}} = \{Y \mapsto \emptyset,\ X \mapsto X'\}$. *Note that the first and third arguments share through* X *and* Y*, as well as through* X' *alone. This is redundant, since the variables are meant to represent possible aliasing of the arguments, regardless of the number of variables shared (and of the particular variables shared). Therefore, the atom* $p(X' \oplus Xs,\ Ys,\ X' \oplus Z)$ *(modulo renaming) will be considered the minimal canonical representative of its class. Intuitively, this is so because the set-sharing represented by all atoms in such a class is already present in the above atom, and it has the minimal number of variables.*

The reader familiar with the *Sharing* domain of [25] will observe that the equivalence relation on abstract atoms reflects the same notion as that conveyed by the sharing sets of Jacobs and Langen. This is no coincidence and is elaborated on in Section 6. Note also that from the implementation perspective, abstract atoms should be maintained using minimal representatives of their corresponding equivalence classes.

4 An Abstract Domain for Sharing Analysis

We propose set logic programs as a formal basis for studying sharing properties of logic programs. The sets of variables in the arguments of an abstract atom represent possible set-sharing between corresponding concrete arguments.

Abstraction of Terms, Atoms and Substitutions

The formal relation between concrete and abstract atoms is given in terms of an abstraction function on atoms which replaces the concrete terms in an atom by the set of variables it contains.

$$\begin{aligned} &\sigma : T(\Sigma, \mathcal{V}) \to T(\Sigma^{\oplus}, \mathcal{V}) \\ &\sigma(t) = \begin{cases} \emptyset & \text{if } vars(t) = \emptyset \\ x_1 \oplus \cdots \oplus x_n & \text{if } vars(t) = \{x_1, \ldots, x_n\},\ n > 0 \end{cases} \end{aligned} \tag{7}$$

The abstraction of atoms is obtained by considering the term abstraction separately for each argument of the atom:

$$\begin{aligned} &\sigma : \mathcal{B}_{\mathcal{V}} \to \mathcal{B}_{\mathcal{V}}^{\oplus} \\ &\sigma(p(t_1, \ldots, t_n)) = p(\sigma(t_1), \ldots, \sigma(t_n)) \end{aligned} \tag{8}$$

Example 6 *Consider the concrete atom $p([X, Y|Xs], f(Ys), g(X, Y, Z))$. Its abstraction is:*

$$\sigma(p([X, Y|Xs], f(Ys), g(X, Y, Z)) = p(X \oplus Y \oplus Xs,\ Ys,\ X \oplus Y \oplus Z).$$

Observe that $p(X \oplus Y \oplus Xs,\ Ys,\ X \oplus Y \oplus Z) \approx_{ir} p(X' \oplus Xs,\ Ys,\ X' \oplus Z)$ as explained in Example 5.

We say that an abstract atom π describes a concrete atom a, denoted $\pi \propto a$, if $\sigma(a) \preceq_{ir} \pi$. Observe that $\pi \propto a$ implies that π contains more set-sharing than a.

Lemma 4.1 *Let π and a be abstract and concrete atoms such that $\pi \propto a$.*

Then, π contains more set-sharing than a.

PROOF. First note that by Observation 1 the results of Lemma 2.1 apply also to abstract atoms. If $\pi \propto a$ then by definition $\sigma(a) \preceq_{ir} \pi$ which implies that $\mathcal{A}'(\sigma(a)) \subseteq \mathcal{A}'(\pi)$. However, note that $\mathcal{A}'(\sigma(a)) = \mathcal{A}'(a)$ which implies the claim. $\square$

A substitution is abstracted by abstracting the terms in its range[2]:

$$\begin{aligned} &\sigma : Sub \to Sub^{\oplus} \\ &\sigma(\theta) = \left\{ x \mapsto \sigma(x\theta) \,\middle|\, x \in dom(\theta) \right\}. \end{aligned} \tag{9}$$

We say that an abstract substitution μ describes a concrete substitution θ, denoted $\mu \propto \theta$, if $\sigma(\theta) \preceq_{ir} \mu$.

The following lemma establishes the relation between the abstraction of atoms and substitutions.

Lemma 4.2 *For any atom b and substitution θ: $\sigma(b\theta) = \sigma(b) \cdot \sigma(\theta)$.*

PROOF. Assume that b is of the form $p(t_1, \ldots, t_n)$. For each argument t_i we have $vars(t_i\theta) = vars(\sigma(t_i\theta)) = vars(\sigma(t_i) \cdot \sigma(\theta))$ since the abstractions defined in Equations (7) and (9) preserve the original variables of t and θ. Thus, $\sigma(t_i\theta) = \sigma(t_i) \cdot \sigma(\theta)$ for each t_i, which implies the statement for atoms. $\square$

The Lattice of Abstract Atoms

The domain $[\mathcal{B}^{\oplus}_{\mathcal{V}}]_{\approx_{ir}}$ of abstract atoms forms a lattice w.r.t. the (induced) $\preceq_{ir}$ ordering. The least upper bound of abstract atoms π_1 and π_2 (with respect to $\preceq_{ir}$) is denoted $\pi_1 \sqcup \pi_2$ and can also be characterized by the following theorem.

Theorem 4.3 (least upper bound of abstract atoms)
Let $\pi_1 = p(\tau_1, \ldots, \tau_n)$ and $\pi_2 = p(\tau'_1, \ldots, \tau'_n)$ be (representatives of equivalence classes of) abstract atoms which are renamed apart. Then,

$$\pi_1 \sqcup \pi_2 = p(\tau_1 \oplus \tau'_1, \ldots, \tau_n \oplus \tau'_n).$$

[2] To simplify notation, we denote by σ the abstraction functions for terms, atoms, and substitutions. The intended use will always be clear from the context.

PROOF. Let $\pi = p(\tau_1 \oplus \tau_1', \ldots, \tau_n \oplus \tau_n')$. Observe that π is indeed an upper bound of π_1 and π_2. To demonstrate this we construct an ir-substitution ψ which maps all variables of π_1 to $\emptyset$. Clearly, $\pi\psi = \pi_2$ and thus $\pi_2 \preceq_{ir} \pi$. Similarly $\pi_1 \preceq_{ir} \pi$.

Now, let us prove that π is a least upper bound of π_1 and π_2. Consider an upper bound π' of π_1 and π_2 such that $\pi' \preceq_{ir} \pi$. By contradiction, if $\pi' \neq \pi$, then by Observation 2 there exists at least one variable z in π such that $\pi' \preceq_{ir} \pi \cdot \left\{ z \mapsto \emptyset \right\}$. By construction of π there exists at least one variable z' which occurs in π_1 or π_2 in the same argument positions as z occurs in π. At the same time π' does not contain a variable occurring in the same argument positions as z occurs in π. Thus, either $\pi_1 \not\preceq_{ir} \pi'$ or $\pi_2 \not\preceq_{ir} \pi'$ which means that π' is not an upper bound of π_1 and π_2. The contradiction implies that π is a least upper bound of π_1 and π_2. $\square$

The notion of least upper bound extends to sets of abstract atoms with the same predicate symbol in the natural way and to arbitrary sets by combining all of the atoms with the same predicate symbol. Let $\mathcal{I} \subseteq \mathcal{B}_{\mathcal{V}}^{\oplus}$, then

$$\sqcup\mathcal{I} = \left\{ \sqcup \{p(\tau_1, \ldots, \tau_n) \in \mathcal{I}\} \,\middle|\, p/n \in \Pi \right\}. \tag{10}$$

Abstract Interpretations

An abstract domain for sharing analysis is obtained by considering sets of abstract atoms modulo a suitable notion of equivalence. We view sets of abstract atoms as being downwards-closed with respect to $\preceq_{ir}$: if $a \in \mathcal{I} \subseteq \mathcal{B}_{\mathcal{V}}^{\oplus}$ and $a' \preceq_{ir} a$ then $a' \in \mathcal{I}$. To do this we impose the following ordering on sets of abstract atoms:

$$\mathcal{I}_1 \preceq \mathcal{I}_2 \Leftrightarrow \forall \pi_1 \in \sqcup\mathcal{I}_1 \; \exists \pi_2 \in \sqcup\mathcal{I}_2 : \; \pi_1 \preceq_{ir} \pi_2. \tag{11}$$

This ordering can be lifted up to the quotient of the corresponding equivalence relation:

$$\mathcal{I}_1 \approx \mathcal{I}_2 \;\Leftrightarrow\; (\mathcal{I}_1 \preceq \mathcal{I}_2) \wedge (\mathcal{I}_2 \preceq \mathcal{I}_1). \tag{12}$$

The domain of abstract interpretations is thus the lower powerdomain, or Hoare powerdomain, of (closed sets of elements of) $[\mathcal{B}_{\mathcal{V}}^{\oplus}]_{\approx_{ir}}$ with the $\preceq$ ordering. This domain is the quotient $[\wp([\mathcal{B}_{\mathcal{V}}^{\oplus}]_{\approx_{ir}})]_{\approx}$ of the equivalence relation of Equation 12, which is denoted in the following by an abuse of notation as $\wp(\mathcal{B}_{\mathcal{V}}^{\oplus})$. It is worth noting that for any set $\mathcal{I}$ of abstract atoms, $\sqcup\mathcal{I}$ is an abstract interpretation with minimal cardinality among those equivalent to $\mathcal{I}$

(w.r.t. $\approx$), containing at most one abstract atom for each predicate symbol in Π. These are the canonical representatives of the corresponding equivalence classes.

Lemma 4.4 *$\langle \wp(\mathcal{B}_{\mathcal{V}}^{\oplus}), \preceq \rangle$ is a complete lattice.*

PROOF. If L is a set of downwards-closed sets then $\cap L$ and $\sqcup L$ are downwards-closed, therefore $lub(L) \approx \sqcup L$ and $glb(L) \approx \cap L$. $\square$

The relation between concrete and abstract interpretations is formalized as usual in terms of a pair of abstraction and concretization functions lifted from the abstraction function σ on atoms in the standard way:

$$\begin{array}{ll} \alpha : \wp(\mathcal{B}_{\mathcal{V}}) \to \wp(\mathcal{B}_{\mathcal{V}}^{\oplus}) & \gamma : \wp(\mathcal{B}_{\mathcal{V}}^{\oplus}) \to \wp(\mathcal{B}_{\mathcal{V}}) \\ \alpha(I) = \left\{ \sigma(a) \,\middle|\, a \in I \right\} & \gamma(\mathcal{I}) = \bigcup \left\{ I \,\middle|\, \alpha(I) \preceq \mathcal{I} \right\} \end{array} \tag{13}$$

Theorem 4.5 *$\langle \wp(\mathcal{B}_{\mathcal{V}}), \alpha, \wp(\mathcal{B}_{\mathcal{V}}^{\oplus}), \gamma \rangle$ is a Galois insertion.*

PROOF. It follows immediately from the definitions that α and γ are monotonic. Moreover:

$$\forall I \in \wp(\mathcal{B}_{\mathcal{V}}) : \gamma(\alpha(I)) = \bigcup \left\{ I' \,\middle|\, \alpha(I') \preceq \alpha(I) \right\} \supseteq I,$$

since $I \in \left\{ I' \,\middle|\, \alpha(I') \preceq \alpha(I) \right\}$;

$$\begin{aligned} \forall \mathcal{I} \in \wp(\mathcal{B}_{\mathcal{V}}^{\oplus}) : \ \alpha(\gamma(\mathcal{I})) &= \alpha\left(\bigcup \left\{ I \,\middle|\, \alpha(I) \preceq \mathcal{I} \right\} \right) \approx \\ &\approx \bigcup \left\{ \alpha(I) \,\middle|\, \alpha(I) \preceq \mathcal{I} \right\} \approx \mathcal{I} \end{aligned} \qquad \square$$

5 Abstract Operations for Sharing Analysis

When constructing a semantic based program analysis for logic programs several main operations must be defined: abstract unification, abstract composition, application of abstract substitutions (or projection) and least upper bound. The concrete atoms and substitutions encountered during a computation are described by corresponding abstract atoms and substitutions. Given descriptions of concrete syntactic objects the abstract operations describe the possible results of all corresponding concrete operations.

In our case all of these operations, except for unification, have already been defined and it is straightforward to prove that they are correct and optimal in the context of sharing analysis. These proofs can be found in Appendix A. This section focuses on the definition of abstract unification for sharing analyses.

We distinguish between the unification of abstract terms and that of abstract atoms. For abstract terms we rely on the well-studied notion of ACI1-unification [2]. Intuitively, ACI1-unification provides the basis for the unification of sets of objects. This allows us to formalize in a concise manner the intuition that, upon unification, any variable in one term might match any subset of the variables in the other term.

Recall that an ACI1 unifier of two terms τ_1 and τ_2 is a substitution μ such that $\tau_1\mu =_{ACI1} \tau_2\mu$. In the general case, ACI1-unification is *finitary*. Namely, the unification of τ_1 and τ_2 admits a finite number of "most general" unifiers (in contrast to standard unification which is "*unitary*", i.e., admits at most one most general unifier). In the general case the decision problem for ACI1-unification —whether two terms τ_1 and τ_2 are unifiable — is NP-complete. This can be shown by reducing the ACI-matching problem (which is shown to be NP-complete in [26]) to ACI1-unification as shown in [27].

In our domain we consider a restricted alphabet for ACI1-expressions and consequently, ACI1-unification is far simpler. In our domain there is only one binary function symbol and only one constant. As a consequence, two abstract terms are always unifiable and the underlying decision problem is trivial. Indeed, for any two abstract terms τ_1 and τ_2 the substitution binding the variables of both terms to $\emptyset$ is always a unifier. It turns out that in our case any two abstract terms always have exactly one most general unifier.

There is another important difference between general ACI1-unification and the abstract unification of terms in our domain: we are not interested in the most general ACI1 unifier with respect to the standard instantiation ordering but rather in the most general ACI1 unifier with respect to $\preceq_{ir}$. We denote by $ir\text{-}mgu_{ACI1}(\tau_1, \tau_2)$ the most general ACI1 unifier of τ_1 and τ_2 with respect to this ordering. Note that $ir\text{-}mgu_{ACI1}(\tau_1, \tau_2)$ is not necessarily an independent-range substitution. It only has to be the most general with respect to $\preceq_{ir}$. Moreover, usually this unifier is not an independent-range substitution since it unifies the terms, thus, binding more than one original variable to the same set of variables.

Example 7 *Consider the ACI1-unification of $A \oplus B$ and X.*

$$ir\text{-}mgu_{ACI1}(A \oplus B, X) = \left\{ \begin{array}{l} A \mapsto Z_1 \oplus Z_2,\ B \mapsto Z_2 \oplus Z_3, \\ X \mapsto Z_1 \oplus Z_2 \oplus Z_3 \end{array} \right\}.$$

Note that this unifier is more general than the unifier

$$\mu = \Big\{ A \mapsto Y_1,\ B \mapsto Y_2,\ X \mapsto Y_1 \oplus Y_2 \Big\}$$

since there is an independent-range substitution

$$\psi = \Big\{ Z_1 \mapsto Y_1,\ Z_2 \mapsto \emptyset,\ Z_3 \mapsto Y_2 \Big\}$$

such that $\mu = (ir\text{-}mgu_{ACI1}(A \oplus B, X) \circ \psi)\restriction_{dom(\mu)}$. *Note that the abstract substitution* $\Big\{ A \mapsto \emptyset,\ B \mapsto \emptyset,\ X \mapsto \emptyset \Big\}$ *is also a unifier of these terms. This is the "least general" unifier.*

The following lemma establishes the uniqueness of $ir\text{-}mgu_{ACI1}$ for abstract terms.

Lemma 5.1 *Two abstract terms* $\tau_1, \tau_2 \in T(\Sigma^{\oplus}, \mathcal{V})$ *always have a unique* $ir\text{-}mgu_{ACI1}$.

PROOF. Since τ_1 and τ_2 always unify there always exists at least one most general unifier. Let us show that it is unique. Assume by contradiction that there exist at least two maximal unifiers of τ_1 and τ_2 denoted by μ_1 and μ_2 respectively. Assume without loss of generality that $dom(\mu_1) = dom(\mu_2)$ and that the terms in $range(\mu_1)$ are renamed apart from the terms in $range(\mu_2)$. Consider the substitution $\mu = \lambda x.(x\mu_1 \oplus x\mu_2)$.

Observe that μ is also a unifier of τ_1 and τ_2 since $\tau_1\mu = \tau_1\mu_1 \oplus \tau_1\mu_2 = \tau_2\mu_1 \oplus \tau_2\mu_2 = \tau_2\mu$. Moreover, $\mu_1 \preceq_{ir} \mu$ and $\mu_2 \preceq_{ir} \mu$ since the independent-range substitutions mapping variables of $dom(\mu_1)$ or variables of $dom(\mu_2)$ to $\emptyset$ are obvious. Thus, μ is a more general unifier than μ_1 and μ_2 which contradicts with the assumption that μ_1 and μ_2 are maximal unifiers of τ_1 and τ_2. $\square$

Figure 2 describes a simple algorithm to compute the $ir\text{-}mgu_{ACI1}$ of a pair of abstract terms. The unification procedure consists of two phases. The set S computed in the first phase consists of sets of variables representing all possible sharing in a corresponding unification. The second phase converts S into an abstract substitution by mapping each variable to a set of labels corresponding to those sets of S in which it appears.

Theorem 5.2 *The algorithm depicted in Figure 2 computes a most general ACI1 unifier of* $\tau_1, \tau_2 \in T(\Sigma^{\oplus}, \mathcal{V})$ *with respect to the* $\preceq_{ir}$*-ordering.*

$\underline{\textbf{ir-mgu}_{\mathbf{ACI1}}(\tau_1, \tau_2):}$

$v_1 = vars(\tau_1)$

$v_2 = vars(\tau_2)$

if $(v_1 = \emptyset) \vee (v_2 = \emptyset)$ **then return** $\lambda x \in (v_1 \cup v_2).\ \emptyset$

else

$S = \left\{ s \subseteq (v_1 \cup v_2) \,\middle|\, s \cap v_1 \neq \emptyset,\ s \cap v_2 \neq \emptyset \right\}$

let $S = \{s_1, \quad \ldots, \quad s_k\}$

$Z = \{z_1, \quad \ldots, \quad z_k\}$ // fresh variables

return $\lambda x \in (v_1 \cup v_2).\ \bigoplus_{x \in s_i} z_i$

Fig. 2. ACI1-unification of abstract terms

PROOF. The claim is straightforward for the cases when v_1 or v_2 is empty. Consider the situation when $v_1 \neq \emptyset$ and $v_2 \neq \emptyset$. Denote the output of the algorithm shown on Figure 2 by μ. Clearly μ is a unifier of τ_1 and τ_2 because $\tau_1\mu = \tau_2\mu = z_1 \oplus \ldots \oplus z_k$. Let us show that μ is a most general unifier. Assume by contradiction that there exists an ACI1 unifier μ' of τ_1 and τ_2 which is strictly more general than μ, i.e., $\mu \preceq_{ir} \mu'$ and $\mu \neq \mu'$. Assume without loss of generality that $dom(\mu) = dom(\mu') = v_1 \cup v_2 = D$. Then by Observation 2 there exists a variable $z \in vars(range(\mu'))$ such that $\mu \preceq_{ir} \left(\mu' \circ \left\{ z \mapsto \emptyset \right\}\right) \restriction_D$. Variable z occurs in μ' through some subset of variables from D. Namely, $occs(\mu', z) = \left\{ x \in v_1 \cup v_2 \,\middle|\, z \in vars(x\mu') \right\}$.

Assume $occs(\mu', z) = s_k$, for $s_k \in S$ as computed by the algorithm. Consider the substitution $\mu_k = \mu \circ \left\{ z_i \mapsto \emptyset \,\middle|\, i \neq k \right\}$. This substitution maps all variables of s_k to z_k and all other variables of $dom(\mu)$ to $\emptyset$. Clearly, $\mu_k \preceq_{ir} \mu$, and thus by our initial assumption $\mu_k \preceq_{ir} \mu \circ \left\{ z \mapsto \emptyset \right\}$. However, variables of s_k are not mapped by $\mu \circ \left\{ z \mapsto \emptyset \right\}$ to any common variable and thus, there is no ir-instance of $\mu \circ \left\{ z \mapsto \emptyset \right\}$ having a projection on D equal to μ_k.

If $occs(\mu', z) \neq s_i$ for any i then either $occs(\mu', z) \cap v_1 = \emptyset$ or $occs(\mu', z) \cap v_2 = \emptyset$. In both cases z occurs in only one term of either $\tau_1\mu'$ or $\tau_2\mu'$, and thus, μ' is not a unifier of τ_1 and τ_2.

In any case, the contradiction implies that there is no unifier of τ_1 and τ_2 which is more general than μ. $\square$

The following example demonstrates the algorithm for ACI1-unification of abstract terms.

Example 8 *Consider the evaluation of* $\textit{ir-mgu}_{ACI1}(A \oplus B,\ Y)$. *In the first step, the algorithm computes the non-empty sets of variables* $v_1 = \left\{ A, B \right\}$ *and* $v_2 = \left\{ Y \right\}$. *In the next step,* $S = \left\{ \{A, Y\},\ \{B, Y\}\ , \{A, B, Y\} \right\}$ *and the fresh variables* $Z = \left\{ Z_1,\ Z_2,\ Z_3 \right\}$ *are associated with the corresponding elements of* S. *The final step computes the unifier, by mapping each variable from* $v_1 \cup v_2$ *to a term constructed from the corresponding fresh variables from the set* Z. *For instance, for* A *the corresponding variables are* Z_1 *and* Z_3 *since* A *appears in the first and the third set of* S. *Thus, for* A *the resulting binding is* $A \mapsto Z_1 \oplus Z_3$. *The result of the unification is:*

$$\textit{ir-mgu}_{ACI1}(A \oplus B,\ Y) = \left\{ \begin{array}{l} A \mapsto Z_1 \oplus Z_3,\ B \mapsto Z_2 \oplus Z_3, \\ Y \mapsto Z_1 \oplus Z_2 \oplus Z_3 \end{array} \right\}.$$

In the following we justify the special role that ACI1-unification plays in the formalization of an abstract unification algorithm for sharing analysis. We first discuss the relation between the standard unification of two terms t_1, t_2 and the ACI1-unification of their abstractions $\sigma(t_1), \sigma(t_2)$. The following two lemmata state that ACI1-unification provides a correct and optimal description of the corresponding concrete unification. It is important to note that there is a technical difficulty in stating this argument as we have not given a formal notion of description for terms (but only for other syntactic objects, such as atoms and substitutions). It is inappropriate to do so, because the idea of the description relation is based on the sharing of variables between the terms in a syntactic object and formalized in terms of an appropriate equivalence relation. Observe that an abstract term has no "meaning" on its own. It is only in the context of a more complex syntactic object that the notion of sharing has a meaning. The following example illustrates this point.

Example 9 *Consider the abstract atom* $\pi = p(A \oplus B, B)$ *and abstract term* $\tau = A \oplus B$. *The abstract atom* π *represents a concrete atom of the form* $p(f(A, B), g(B))$ *or of the form* $p([W, X, Y, Z], [X, Y, Z])$ *in which there are some variables in common in the two arguments. But we can not say that* τ *describes the concrete terms* $f(A, B)$ *or* $[W, X, Y, Z]$.

The following lemma states that the unification of concrete terms is approximated by the ACI1-unification of their abstractions. The correctness of ACI1-

unification in this special case is used below to establish correctness of ACI1-unification of terms within a given context, i.e., within atoms.

Lemma 5.3 (ACI1-unification of abstract terms is correct)
For the concrete terms t_1 and t_2:

$$ir\text{-}mgu_{ACI1}(\sigma(t_1), \sigma(t_2)) \propto mgu(t_1, t_2)$$

PROOF. Let $\theta = mgu(t_1, t_2)$ and $\mu = ir\text{-}mgu_{ACI1}(\sigma(t_1), \sigma(t_2))$. Since θ is a unifier of t_1 and t_2 and by Lemma 4.2 we have $\sigma(t_1\theta) = \sigma(t_2\theta) = \sigma(t_1) \cdot \sigma(\theta) = \sigma(t_2) \cdot \sigma(\theta)$. Thus, $\sigma(\theta)$ is an ACI1 unifier of $\sigma(t_1)$ and $\sigma(t_2)$. Since μ is a most general ACI1 unifier of $\sigma(t_1)$ and $\sigma(t_2)$ we have $\sigma(\theta) \preceq_{ir} \mu$, or equivalently, $\mu \propto \theta$. $\square$

Lemma 5.4 (ACI1-unification of abstract terms is optimal)
For abstract terms τ_1 and τ_2 and abstract unifier $\mu = ir\text{-}mgu_{ACI1}(\tau_1, \tau_2)$, and for any μ' which is (strictly) less general than μ, there exist concrete terms t_1 and t_2 such that $\sigma(t_1) = \tau_1$, $\sigma(t_2) = \tau_2$, and $\mu' \not\propto mgu(t_1, t_2)$.

PROOF. The proof is technical and can be found in Appendix A. $\square$

Now let us consider the correctness of ACI1-unification of abstract terms for sharing analysis. We now have to consider the context in which the terms occur, i.e., as arguments of abstract atoms.

Consider a pair of abstract atoms $\pi = p(\tau_1, \ldots, \tau_n)$ and $\pi' = p(\tau'_1, \ldots, \tau'_n)$. We argue that an appropriate (correct and optimal) abstract unification for sharing analysis is obtained by considering the ACI1-unification of the corresponding pairs of abstract terms τ_i and τ'_i. To argue correctness and optimality, each such unification must be considered in the context of the entire set of equations $\left\{ \tau_1 = \tau'_1, \ldots, \tau_n = \tau'_n \right\}$.

Lemma 5.5 (ACI1-unification of abstract terms is correct)
Let $\pi = p(\tau_1, \ldots, \tau_n)$, $\pi' = p(\tau'_1, \ldots, \tau'_n)$, $a = p(t_1, \ldots, t_n)$ and $a' = p(t'_1, \ldots, t'_n)$ such that $\pi \propto a$ and $\pi' \propto a'$. Then for i, $1 \leq i \leq n$:

$$\pi \cdot ir\text{-}mgu_{ACI1}(\tau_i, \tau'_i) \propto a \cdot mgu(t_i, t'_i).$$

PROOF. See Appendix A. $\square$

Now consider the abstract unification of a pair of abstract atoms. The abstract unifier of atoms π_1 and π_2, denoted $mgu^{\mathcal{A}}(\pi_1, \pi_2)$, is defined in terms of the set of equations between the terms in the corresponding argument positions:

$$mgu^{\mathcal{A}}(\mathcal{E}) = \begin{cases} \varepsilon & \text{if } \mathcal{E} = \emptyset \\ \mu \circ mgu^{\mathcal{A}}(\mathcal{E}'\mu) & \text{if } \mathcal{E} = \ \{\tau \doteq \tau'\} \cup \mathcal{E}' \\ & \text{and } \mu = ir\text{-}mgu_{ACI1}(\tau, \tau') \end{cases} \tag{14}$$

Abstract unification is thus defined much the same as in the concrete case. It is parameterized by abstract unification of terms and abstract composition of substitutions.

It is interesting to note that it is possible to define the abstract unification for abstract atoms, similar to the case of abstract terms, as the most general ACI1 unifier of the atoms (with respect to $\preceq_{ir}$). However, this results in a very imprecise (although correct) abstract unification operation for sharing analysis. Indeed, we shall see that $mgu^{\mathcal{A}}$ as defined in Equation (14) is both correct and optimal for our domain.

Example 10 *Consider the unification of the abstract atoms $p(A, B)$ and $p(X, Y)$. The most general ACI1 unifier (with respect to $\preceq_{ir}$) for these atoms is*

$$\zeta = \left\{ \begin{array}{l} A \mapsto Z_1 \oplus Z_2,\ B \mapsto Z_2 \oplus Z_3, \\ X \mapsto Z_1 \oplus Z_2,\ Y \mapsto Z_2 \oplus Z_3 \end{array} \right\}$$

This unifier is correct for sharing analysis since it approximates all possibilities of unification of two atoms with independent arguments. However, ζ is imprecise since it introduces (through Z_2) the possibility that all four arguments (of both atoms) be aliased. Obviously, the concrete unification of atoms with independent arguments does not introduce such an aliasing. Consider now the abstract unification $mgu^{\mathcal{A}}(p(A, B), p(X, Y))$ which is computed by solving the set of equations $\left\{ A = X,\ B = Y \right\}$. The ACI1-unification for the first equation results in $ir\text{-}mgu(A, X) = \left\{ A \mapsto Z,\ X \mapsto Z \right\}$. Applying this result to the rest of the equations gives $\left\{ B = Y \right\}$. Now, $ir\text{-}mgu_{ACI1}(B, Y) = \left\{ B \mapsto W,\ Y \mapsto W \right\}$ and finally

$$\mu = mgu^{\mathcal{A}}(\left\{ A = X,\ B = Y \right\}) = \left\{ A \mapsto Z,\ B \mapsto W,\ X \mapsto Z,\ Y \mapsto W \right\}.$$

This unifier indeed correctly approximates the result of unifying two atoms with independent arguments. Note also that $\mu \preceq_{ir} \zeta$ provided by $\mu = (\zeta \circ \psi)\lceil_{dom(\mu)}$ where

ψ *is the independent-range substitution* $\psi = \left\{ Z_1 \mapsto Z,\ Z_2 \mapsto \emptyset,\ Z_3 \mapsto W \right\}$.

The following illustrates a more complex example of abstract unification.

Example 11 *Consider the unification of the abstract atoms:* $\pi_1 = p(A, A \oplus B, B)$ *and* $\pi_2 = p(X, Y, Z)$. *The unifier* $mgu^{\mathcal{A}}(\pi_1, \pi_2)$ *is computed as defined by Equation 14 by solving the set of equations* $\left\{ A = X,\ A \oplus B = Y,\ B = Z \right\}$. *In each iteration we apply the ir-*mgu_{ACI1} *of the first (upper) equation in the set to the other equations. We also assume that on each iteration the resulting substitution is projected on the set of variables of the original equations, i.e. on the domain of variables of interest.*

$$mgu^{\mathcal{A}}\left(\left\{\begin{array}{l} A = X, \\ A \oplus B = Y, \\ B = Z \end{array}\right\}\right) = \left\{\begin{array}{l} A \mapsto Z_1, \\ X \mapsto Z_1 \end{array}\right\} \circ mgu^{\mathcal{A}}\left(\left\{\begin{array}{l} Z_1 \oplus B = Y, \\ B = Z \end{array}\right\}\right) =$$

$$= \left\{\begin{array}{l} A \mapsto Z_1, \\ X \mapsto Z_1 \end{array}\right\} \circ \left\{\begin{array}{l} Y \mapsto Z_2 \oplus Z_3 \oplus W, \\ Z_1 \mapsto Z_3 \oplus W, \\ B \mapsto Z_2 \oplus W \end{array}\right\} \circ mgu^{\mathcal{A}}\left(\left\{ Z_2 \oplus W = Z \right\}\right) =$$

$$= \left\{\begin{array}{l} A \mapsto Z_1, \\ X \mapsto Z_1 \end{array}\right\} \circ \left\{\begin{array}{l} Y \mapsto Z_2 \oplus Z_3 \oplus W, \\ Z_1 \mapsto Z_3 \oplus W, \\ B \mapsto Z_2 \oplus W \end{array}\right\} \circ \left\{\begin{array}{l} Z \mapsto Z_4 \oplus Z_5 \oplus W', \\ Z_2 \mapsto Z_4 \oplus W', \\ W \mapsto Z_5 \oplus W' \end{array}\right\}.$$

The final result is thus:

$$mgu^{\mathcal{A}}(\pi_1, \pi_2) = \left\{\begin{array}{l} X \mapsto Z_3 \oplus Z_5,\ Y \mapsto Z_3 \oplus Z_4 \oplus Z_5, \\ Z \mapsto Z_4 \oplus Z_5,\ A \mapsto Z_3 \oplus Z_5,\ B \mapsto Z_4 \oplus Z_5 \end{array}\right\}$$

in which W' *collapses to* Z_5 *because of equivalence. Notice that* $mgu^{\mathcal{A}}(\pi_1, \pi_2)$ *indicates the possibility of simultaneous sharing between all variables of the initial atoms (expressed by* Z_5*) as justified for example by considering the unification of the concrete atoms* $p(A, f(A, B), B)$ *with* $p(X, f(Y, Y), Z)$.

Correctness and optimality of abstract unification now follow from the correctness and optimality results of the "atomic" operations used to define the abstract unification of tuples of abstract terms in Equation (14).

Theorem 5.6 (abstract unification is correct for set-sharing)
Let a *and* a' *be concrete atoms such that* $mgu(a, a') = \theta$. *Let* π *and* π' *be*

abstract atoms such that $\pi \propto a$ *and* $\pi' \propto a'$. *Let* $\mu = mgu^{\mathcal{A}}(\pi, \pi')$. *Then* $\pi\mu \propto a\theta$.

PROOF. Let $\pi = p(\tau_1, \ldots, \tau_n)$, $\pi' = p(\tau'_1, \ldots, \tau'_n)$, $a = p(t_1, \ldots, t_n)$ and $a' = p(t'_1, \ldots, t'_n)$. Let $\mu_i = ir\text{-}mgu(\tau_i \cdot (\mu_1 \circ \ldots \circ \mu_{i-1}), \tau'_i \cdot (\mu_1 \circ \ldots \circ \mu_{i-1}))$ and $\theta_i = mgu(t_i \cdot (\theta_1 \circ \ldots \circ \theta_{i-1}), t'_i \cdot (\theta_1 \circ \ldots \circ \theta_{i-1}))$, for $i = 1, \ldots, n$, and $\mu_i = \theta_i = \varepsilon$ for $i = 0$. We prove that $\pi \cdot (\mu_1 \circ \cdots \circ \mu_i) \propto a \cdot (\theta_1 \circ \cdots \circ \theta_i)$ is an invariant of the unification process implied by Equation (14).

It trivially holds at the beginning of the process, since $i = 0$ implies that the invariant is equivalent to $\pi \propto a$. If it holds for $i = k$ then, applying Lemma 5.5, it also holds for $i = k + 1$. Finally, for $i = n$ it implies that $\pi \cdot (\mu_1 \circ \cdots \circ \mu_n) \propto a \cdot (\theta_1 \circ \cdots \circ \theta_n)$, i.e., $\pi \cdot mgu^{\mathcal{A}}(\pi, \pi') \propto a \cdot mgu(a, a')$. □

Theorem 5.7 (abstract unification is optimal for set-sharing)
Let $\mathcal{E}$ *be a set of abstract equations and denote* $\mu = mgu^{\mathcal{A}}(\mathcal{E})$. *There is no unifier* μ' *for* $\mathcal{E}$ *which is more precise than* μ, *i.e., such that* $\mu' \preceq_{ir} \mu$ *and* $\mu \not\preceq_{ir} \mu'$, *which is also correct for set-sharing.*

PROOF. See Appendix A. □

The reader might have noticed that although abstract unification is defined as solving sets of equations, the examples actually consider sequences of equations. The following result, which is a consequence of Theorem 5.7 justifies this.

Corollary 5.8 (abstract unification is confluent)
An abstract unifier for a set of abstract equations is independent of the order in which the equations are solved.

The results of Theorems 5.6 and 5.7 make one of the main points in our presentation. They show that there is a natural ordering (based on independent-range substitutions) for set-sharing analysis for which abstract unification is defined simply by solving a set of equations just as in the concrete case. Correctness and optimality of the abstract operations is a clear consequence of the "algebraic" nature of the abstract domain.

6 Set Logic Programs and Set-Sharing

This section illustrates that the abstract domain based on set logic programs is isomorphic to the well-known *Sharing* domain of Jacobs and Langen [25].

Recall the original definition of the *Sharing* domain which consists of sets of sets of program variables ordered by set inclusion. Sharing information is characterized using the notion of variable occurrences through a substitution, as specified by Equation (1). The elements of the *Sharing* domain are abstract substitutions which are sets of sets of variables and hence we denote $Sharing = \wp(\wp(\mathcal{V}))$.

A set of variables S in an abstract substitution κ indicates the possibility of sharing between these variables. Namely, the possibility that the variables in S occur in a substitution described by κ through some variable. Concrete substitutions are abstracted to elements of the *Sharing* domain using the function $\mathcal{A} : Sub \to \wp(\wp(\mathcal{V}))$ given in Equation (2). The abstraction and concretization functions for the sharing domain are defined as follows:

$$\begin{array}{ll} \alpha^{Sh} : \wp(Sub) \to Sharing & \gamma^{Sh} : Sharing \to \wp(Sub) \\ \alpha^{Sh}(\Theta) = \bigcup \left\{ \mathcal{A}(\theta) \,\middle|\, \theta \in \Theta \right\} & \gamma^{Sh}(\kappa) = \left\{ \theta \in Sub \,\middle|\, \mathcal{A}(\theta) \subseteq \kappa \right\} \end{array} \tag{15}$$

and a Galois insertion is then constructed.

The following example illustrates the description of concrete substitutions by *Sharing* substitutions.

Example 12 *Let $\kappa = \left\{ \{A,B\},\{B,C\},\{A\},\{B\},\{C\},\emptyset \right\}$ be an abstract substitution in the Sharing domain. The substitutions $\theta_1 = \{A \mapsto f(X,Y), B \mapsto g(Y,Z), C \mapsto f(Z,V)\}$ and $\theta_2 = \{A \mapsto f(X), B \mapsto g(Y), C \mapsto f(Z)\}$ are described by κ: In θ_1, X occurs through $\{A\}$, Y occurs through $\{A,B\}$, Z occurs through $\{B,C\}$ and V occurs through $\{C\}$, and in θ_2 there are variables which occur through $\{A\}$, $\{B\}$ and $\{C\}$ — and these occurrences are all specified in κ. Note that the domain of an abstract substitution $\kappa \in Sharing$ must be explicitly specified, as any variable of interest not occurring in κ is considered ground. In contrast, the variables of interest for a set substitution are those in its domain.*

In principle the domain based on set logic programs is formalized in terms of a Galois insertion of *abstract atoms* while the *Sharing* domain is based on a Galois insertion of *abstract substitutions*. The reader should notice that in fact set-sharing analyses, such as those used in [25,32], are actually based on pairs consisting of a concrete atom of the form $p(x_1, \ldots, x_n)$ together with an abstract substitution. Note however, that *Sharing* substitutions cannot be applied to atoms, since they are in fact an encoding of sharing information rather than "true" substitutions. Similarly, an abstract atom $p(\tau_1, \ldots, \tau_n)$ in our domain can also be viewed as a pair $\langle p(\bar{x}), \mu \rangle$, where $\bar{x}$ is a vector of n variables and μ is a set substitution in the form $\{x_1 \mapsto \tau_1, \ldots, x_n \mapsto \tau_n\}$. To facilitate the proof of isomorphism we provide an equivalent definition for our abstract domain defining it as a domain of abstract substitutions:

$$\begin{array}{ll}\alpha^{\oplus} : \wp(Sub) \to Sub^{\oplus} & \gamma^{\oplus} : Sub^{\oplus} \to \wp(Sub) \\ \alpha^{\oplus}(\Theta) = \bigsqcup_{ir} \left\{ \sigma(\theta) \,\middle|\, \theta \in \Theta \right\} & \gamma^{\oplus}(\kappa) = \left\{ \theta \in Sub \,\middle|\, \sigma(\theta) \preceq_{ir} \kappa \right\},\end{array} \tag{16}$$

where $\bigsqcup_{ir}$ denotes a least upper bound of two or more set substitutions with respect to the $\preceq_{ir}$-ordering. The formal construction of a Galois insertion is analogous to that given in Equation (13) and Theorem 4.5.

The following theorem establishes the isomorphism of two representations of sharing information. Namely, that each element in the set $Sub^{\oplus}$ corresponds to an element in $Sharing$ and vice versa.

Lemma 6.1 *There exists a set isomorphism between $Sub^{\oplus}$ and $Sharing$.*

PROOF. Note that the abstraction function $\mathcal{A} : Sub \to Sharing$ extends naturally to a function $\mathcal{A} : Sub^{\oplus} \to Sharing$ viewing sets of variables as ordinary terms. Hence, we prove the lemma demonstrating that $\mathcal{A} : Sub^{\oplus} \to Sharing$ is a bijective function for which an inverse function $\mathcal{A}^{-1} : Sharing \to Sub^{\oplus}$ can be provided.

Let $\kappa = \{S_1, \ldots, S_n\}$ be an element of the $Sharing$ domain defined for a set D of variables of interest. Assume without loss of generality that the domain of substitutions in $Sub^{\oplus}$ is D. Let $\{z_1, \ldots, z_n\}$ be a set of fresh variables, one for each S_i in κ. The inverse function yielding the set substitution μ which corresponds to κ is defined by:

$$\begin{array}{l}\mathcal{A}^{-1} : Sharing \to Sub^{\oplus} \\ \mathcal{A}^{-1}(\kappa) = \left\{ x \mapsto \bigoplus_{x \in S_i} z_i \,\middle|\, x \in D \right\}.\end{array}$$

It is straightforward to see that $\mathcal{A} \circ \mathcal{A}^{-1}$ and $\mathcal{A}^{-1} \circ \mathcal{A}$ correspond to identity functions in $Sub^{\oplus}$ and $Sharing$ respectively. $\square$

Example 13 *Recall the abstract substitution κ of Example 12. Consider the set substitution, $\mu = \{A \mapsto \{X, U\}, B \mapsto \{X, Y, V\}, C \mapsto \{Y, W\}\}$. We have that $\mathcal{A}(\mu) = \left\{ \{A, B\}, \{B, C\}, \{A\}, \{B\}, \{C\}, \emptyset \right\} = \kappa$ and that $\mathcal{A}^{-1}(\kappa) = \{A \mapsto \{X', U'\}$, $B \mapsto \{X', Y', V'\}, C \mapsto \{Y', W'\}\} \approx_{ir} \mu$.*

The following lemma establishes the relation between the ordering of set substitutions and the ordering in the $Sharing$ domain. Namely the fact that the orders of elements in these abstract domains are isomorphic.

Lemma 6.2 (order embedding)
There is an order embedding between $\langle Sub^{\oplus}, \preceq_{ir}\rangle$ *and* $\langle Sharing, \subseteq\rangle$.

PROOF. Let μ_1 and μ_2 be two abstract substitutions and let D be the set of variables of interest. Assume without loss of generality that $dom(\mu_1) = dom(\mu_2) = D$. We prove that $\mu_1 \preceq_{ir} \mu_2 \Leftrightarrow \mathcal{A}(\mu_1) \subseteq \mathcal{A}(\mu_2)$.

($\Rightarrow$) The proof of (1) in Lemma 2.1 applies.
($\Leftarrow$) Given $\mathcal{A}(\mu_1) \subseteq \mathcal{A}(\mu_2)$ we construct an independent-range substitution ψ as follows:

$$\psi = \lambda x. \begin{cases} \emptyset & \text{if } occs(\mu_2, x) \in (\mathcal{A}(\mu_2) \setminus \mathcal{A}(\mu_1)) \\ x & \text{otherwise} \end{cases}$$

The substitution ψ maps to $\emptyset$ all variables of μ_2 which make the set-sharing of μ_2 different to that of μ_1. Thus, $\mathcal{A}(\mu_2 \circ \psi) = \mathcal{A}(\mu_1)$ and consequently, by Lemma 6.1, $\mu_1 = (\mu_2 \circ \psi)\!\upharpoonright_{dom(\mu_1)}$. □

Thus, set substitutions of $Sub^{\oplus}$ and abstract substitutions of $Sharing$ form isomorphic partial orders. Considering the relation of these partial orders to the concrete domain we establish the following result:

Theorem 6.3 (domain isomorphism)

$$\langle \wp(Sub),\ \alpha^{Sh},\ Sharing,\ \gamma^{Sh}\rangle \ \cong\ \langle \wp(Sub),\ \alpha^{\oplus},\ Sub^{\oplus},\ \gamma^{\oplus}\rangle$$

PROOF. Lemma 6.1 and Lemma 6.2 prove that the underlying posets $\langle Sub^{\oplus}, \preceq_{ir}\rangle$ and $\langle Sharing, \subseteq\rangle$ are isomorphic partial orders. It remains to demonstrate that $\gamma^{Sh} \circ \alpha^{Sh}$ and $\gamma^{\oplus} \circ \alpha^{\oplus}$ are equivalent closure operators.

$$\gamma^{Sh} \circ \alpha^{Sh} = \lambda\Theta \in \wp(Sub).\ \left\{ \xi \,\middle|\, \mathcal{A}(\xi) \subseteq \sqcup_{Sh} \left\{ \mathcal{A}(\theta) \,\middle|\, \theta \in \Theta \right\} \right\}.$$

The isomorphism of partial orders $\langle Sharing, \subseteq\rangle$ and $\langle Sub^{\oplus}, \preceq_{ir}\rangle$ implies also the isomorphic behavior of $\sqcup_{Sh}$ and $\sqcup_{ir}$. Thus, the former expression is equivalent to

$$\lambda\Theta \in \wp(Sub).\ \left\{ \xi \,\middle|\, \sigma(\xi) \preceq_{ir} \bigsqcup_{ir} \left\{ \sigma(\theta) \,\middle|\, \theta \in \Theta \right\} \right\} = \gamma^{\oplus} \circ \alpha^{\oplus}. \qquad \square$$

The following example illustrates the isomorphism of the two representations of sharing information.

Example 14 *Recall the abstract substitution* κ *and the concrete substitutions* θ_1 *and* θ_2 *of Example 12. Consider the set substitution of Example 13,* $\mu = \{A \mapsto$

$\{X,U\}, B \mapsto \{X,Y,V\}, C \mapsto \{Y,W\}\}$, which is isomorphic to κ. The substitutions θ_1 and θ_2 are described by μ: X indicates the possible aliasing of A and B, Y indicates that of B and C, and U, V and W the possible presence in A, B and C of variables not shared with other variables. The abstract substitution κ and the set substitution μ also describe the substitutions $\theta_3 = \{A \mapsto f(X), B \mapsto g(X)\}$ and $\theta_4 = \{A \mapsto f(X,Y), B \mapsto g(X,Y), C \mapsto Z\}$.

Observe that the above Theorem 6.3 implies also that the domain of abstract interpretations, i.e., subsets of $\mathcal{B}^{\oplus}_{\mathcal{V}}$ ordered by $\preceq_{ir}$ describe the same sharing information as the elements of $\Pi \times Sharing$.

7 Sharing Analysis with Set Logic Programs

The abstract operations defined in Section 4 (unification, application, least upper bound) provide the building blocks to construct an abstract semantics for the sharing analysis of logic programs. Several sharing analyses have been described using these techniques: A bottom-up approach based on abstraction of the well-known s-semantics [18,19,4] is described in [12]. A top-down approach based on tabulation using XSB is described in [10]. In this section we illustrate as an example a simple bottom-up approach based on an abstract immediate consequences operator $\mathcal{T}_{\mathcal{P}} : \wp(\mathcal{B}^{\oplus}_{\mathcal{V}}) \to \wp(\mathcal{B}^{\oplus}_{\mathcal{V}})$ for set logic programs. For a logic program P the least fixed point of $\mathcal{T}_{\sigma(P)}$ provides the sharing analysis for P.

$$\mathcal{T}_{\mathcal{P}}(\mathcal{I}) = \left\{ h\mu \;\middle|\; \begin{array}{l} c \equiv \; h \leftarrow b_1, \ldots, b_n \in \mathcal{P}, \quad a_1, \ldots, a_n \ll_c \mathcal{I} \\ \mu = mgu^{\mathcal{A}}(\langle b_1, \ldots, b_n\rangle, \langle a_1, \ldots, a_n\rangle) \end{array} \right\}. \tag{17}$$

Let us consider the analysis of the well-known *append* program depicted in Figure 3 (left) using the technique discussed above. The analysis is obtained as a least fixed point of $\mathcal{T}_{\sigma(P)}$ applied to the abstract version of *append*, depicted in Figure 3 (right). In the first iteration of the evaluation we collect

(1) $append([\,], Ys, Ys)$.	(1′) $append(\emptyset, \{Ys\}, \{Ys\})$.
(2) $append([X\|Xs], Ys, [X\|Zs]) \leftarrow$	(2′) $append(\{X, Xs\}, \{Ys\}, \{X, Zs\}) \leftarrow$
$append(Xs, Ys, Zs)$.	$append(\{Xs\}, \{Ys\}, \{Zs\})$.

Fig. 3. The append program and its set based abstraction.

an abstract atom of the form $\pi_1 = append(\emptyset, \{Ys\}, \{Ys\})$ corresponding to fact (1′) in Figure 3, characterizing the set of atoms in which the first argument is ground and the second and third arguments are equal terms. In the second iteration a renaming of π_1 is unified with the body of clause (2′) in

Figure 3. Abstract unification in this case specifies that Xs is bound to $\emptyset$ (a ground term) and that Ys and Zs are bound to the same set (variable Ys'). Consequently the head of clause (2′) under such bindings can be represented as $\pi_2 = append(\{X\}, \{Ys'\}, \{X, Ys'\})$. This abstract atom describes the concrete atoms of the form $append(t_1, t_2, t_3)$ which exhibit sharing between t_1 and t_3 and between t_2 and t_3. Note that $\{\pi_2\} \approx \{\pi_1, \pi_2\}$ since $\pi_1 \preceq_{ir} \pi_2$ with $\pi_1 = \pi_2 \cdot \{X \mapsto \emptyset\}$. An additional iteration results in a new abstract atom of the form $\pi_3 = append(\{X, X'\}, \{Ys'\}, \{X, X', Ys'\})$, which is equivalent to π_2. Thus, the fixed point is reached with

$$lfp(\mathcal{T}_{\sigma(P)}) = \Big\{ append(\{X\}, \{Y\}, \{X, Y\}) \Big\}.$$

This result correctly describes the set of atoms in the non-ground s-semantics of the $append/3$ program which are of the form

$$append([X_1, \ldots, X_n], Ys, [X_1, \ldots, X_n|Ys])$$

in which the variable sharing of the last argument with the first and with the second is evident.

8 Sharing Analysis with Linearity

Traditionally, aliasing and groundness analyses are enhanced with other kinds of information such as linearity and freeness, as first noted by Langen [28] and later studied in numerous works [6,7,15,31,5]. The information about variable linearity and freeness is useful in its own right and can significantly improve the set-sharing information obtained. In [13] the authors demonstrate that linearity information improves the precision of set-sharing analyses and reduces the cost of its computation. The following example illustrates this point.

Example 15 *Reconsider the unification of two abstract terms $A \oplus B$ and Y discussed in Example 7. The abstract unifier*

$$ir\text{-}mgu_{ACI1}(A \oplus B, Y) = \left\{ \begin{array}{l} A \mapsto Z_1 \oplus Z_2,\ B \mapsto Z_2 \oplus Z_3, \\ Y \mapsto Z_1 \oplus Z_2 \oplus Z_3 \end{array} \right\}$$

introduces the possibility that A and B be aliased, which is expressed in the resulting unifier by the variable Z_2. Indeed, the variables A and B may be aliased in the unification of corresponding concrete terms due to the possible non-linearity of the term represented by Y. For instance, in the unification of the concrete terms $f(A, B)$ with $f(Y, Y)$.

Assume now that the abstract term Y is restricted to represent only linear concrete terms. By Lemma 2.3, in any corresponding concrete unification A and B are bound to independent linear terms. Thus, knowing that Y represents a linear term we can compute a more precise abstract unifier. Such an abstract unifier will not introduce aliasing of A and B and thus, it is less general (more precise) than ir-mgu_{ACI1}.

Two things must be done in order to extend the sharing analyses described above with linearity information. First, a suitable notation must be adopted to represent linearity information in the abstract domain elements; and then the abstract operations on these elements must be refined to take into account the new information. This section describes such an extended domain. The new domain, the enhanced operations and their formal justification all remain clean and intuitive. After adopting a simple annotation for linear abstract terms (sets of variables which are designated to represent linear terms), the ordering $\preceq_{ir}$ is refined to an ordering denoted $\preceq_{lin}$. Most of the operations defined on abstract objects extend with ease to consider the new annotations. Abstract unification is the exception. However, also in this case the extended operation remains simple due to Lemma 2.3 which induces constraints on the abstract unifier when linearity information is involved.

Syntax: All of the syntactic constructs for sharing analysis with linearity information remain the same as those described in Section 3 with the single difference that abstract terms are annotated to distinguish between *linear* and possibly *non-linear* set expressions. An *annotated abstract term* τ is a set expression of the form $\{\sigma\}$ or $\{\!|\sigma|\!\}$, where $\sigma = x_1 \oplus \cdots \oplus x_n$, often denoted $\{x_1, \ldots, x_n\}$ and $\{\!|x_1, \ldots, x_n|\!\}$ respectively. An abstract term is said to be *linear* if it is of the form $\{\!|\sigma|\!\}$ or of the form $\emptyset$. Abstract atoms and substitutions maintain their definitions from Section 3, with the only difference that they involve annotated abstract terms. We say that two syntactic objects π_1 and π_2 are *equivalent up to annotation* denoted $\pi_1 =_{ann} \pi_2$, if they are equal up to the annotation of the abstract terms they contain.

Example 16 *Abstract atoms $\pi_1 = p(\{\!|A, B|\!\}, \{B\})$ and $\pi_2 = p(\{A, B\}, \{\!|B|\!\})$ are equal up to annotation and thus we write $\pi_1 =_{ann} \pi_2$. Observe that these atoms describe the same set-sharing, because they contain the same variables in the same argument positions.*

The notion of an independent-range abstract substitution also maintains its definition. We say that an abstract substitution is *linear* if it is an abstract independent-range substitution which maps variables to linear abstract terms. So, the main difference is that we can distinguish linear terms in abstract entities.

Application and Composition: For an abstract term τ of the form $\{\!|\sigma|\!\}$ or $\tau = \{\sigma\}$ and an abstract substitution μ we define the application of μ to τ as follows:

$$\tau\mu = \begin{cases} \{\!|\sigma\mu|\!\} & \text{if } \tau \text{ and the projection of } \mu \text{ on } \sigma \text{ are linear} \\ \{\sigma\mu\} & \text{otherwise} \end{cases} \tag{18}$$

Example 17 *Consider applications of an abstract substitution $\mu = \{W \mapsto \{\!|A|\!\},$ $X \mapsto \{\!|B,C|\!\}$, $Y \mapsto \{D,E\}$, $Z \mapsto \{\!|F|\!\}\}$ to abstract terms. Let $\tau = \{\!|X,Y|\!\}$. Then $\tau\mu = \{B,C,D,E\}$ because the non-annotated set expression σ corresponding to τ is $X \oplus Y$ and the projection of μ on σ is $\mu_\sigma = \left\{ X \mapsto \{\!|B,C|\!\},\ Y \mapsto \{D,E\} \right\}$, which is non-linear.*

The application of μ to the following abstract terms τ yields:

τ	$\{\!\|W,X\|\!\}$	$\{W\}$	$\{\!\|W\|\!\}$	$\{\!\|W,Y\|\!\}$	$\{\!\|W,Z\|\!\}$
$\tau\mu$	$\{\!\|A,B,C\|\!\}$	$\{A\}$	$\{\!\|A\|\!\}$	$\{A,D,E\}$	$\{\!\|A,F\|\!\}$

The composition of annotated set substitutions is defined in terms of application as illustrated by the following example.

Example 18 *Consider a composition of two abstract substitutions in the annotated domain:*

$$\begin{aligned} \mu_1 &= \left\{ X \mapsto \{\!|A,B|\!\},\ Y \mapsto \{B,C\} \right\}, \\ \mu_2 &= \left\{ A \mapsto \{W,V\},\ C \mapsto \{\!|Z|\!\} \right\}. \end{aligned}$$

The composition $\mu_1 \circ \mu_2$ is computed using the definition of Equation (18) for application of μ_2 to the terms in the range of μ_1. Thus, $\{\!|A,B|\!\} \cdot \mu_2 = \{W,V,B\}$ and $\{B,C\} \cdot \mu_2 = \{B,Z\}$. Hence

$$\mu_1 \circ \mu_2 = \left\{ X \mapsto \{W,V,B\},\ Y \mapsto \{B,Z\},\ A \mapsto \{W,V\},\ C \mapsto \{\!|Z|\!\} \right\}.$$

The Abstract Domain: We introduce an ordering $\preceq_{lin}$ on annotated abstract objects, similar to the ordering $\leq_{lin}$ given in Section 2. This ordering reflects both set-sharing and linearity information in abstract objects.

For abstract atoms π_1 and π_2, we say that π_1 precedes π_2 in the $\preceq_{lin}$-ordering if π_2 contains more set-sharing information and more non-linearity than π_1. Formally, we say that $\pi_1 \preceq_{lin} \pi_2$ if $linearity(\pi_1) \supseteq linearity(\pi_2)$ and if there exists an independent-range abstract substitution ψ on the variables of π_2 such that

$\pi_1 =_{ann} \pi_2\psi$. Similarly, for abstract substitutions μ_1 and μ_2, $\mu_1 \preceq_{lin} \mu_2$, assuming without loss of generality the domain of interest $dom(\mu_1) = dom(\mu_2) = D$, if there exists an independent-range abstract substitution ψ on the range of μ_2 such that $\mu_1 =_{ann} (\mu_2 \circ \psi)\lceil_D$ and if $linearity(\mu_1) \supseteq linearity(\mu_2)$.

Example 19 *Let $\pi_1 = p(\{\!|A, B|\!\}, \{B\})$ and $\pi_2 = p(\{X, Y\}, \{Y, Z\})$. Let us demonstrate that $\pi_1 \preceq_{lin} \pi_2$. The independent-range substitution*

$$\psi = \left\{ X \mapsto \{\!|A|\!\},\ Y \mapsto \{\!|B|\!\},\ Z \mapsto \emptyset \right\}$$

applied to π_2 results in $p(\{A, B\}, \{B\})$ which is equal up to annotation to π_1. Observe also that $linearity(\pi_1) = \{1\}$ and $linearity(\pi_2) = \emptyset$ and thus, $linearity(\pi_1) \supseteq linearity(\pi_2)$. Therefore, both requirements for $\pi_1 \preceq_{lin} \pi_2$ are satisfied.

Observe that if $\pi_1 \preceq_{lin} \pi_2$ then there exists a linear abstract substitution ψ on the variables of π_2 such that $\pi_1 = \pi_2\psi$. Observe also that the linearity information in abstract atoms is downwards closed with respect to $\preceq_{lin}$.

Similar to the construction in Section 3, we let $\preceq_{lin}$ induce an equivalence relation on the sets of abstract atoms and substitutions and corresponding partial orders on equivalence classes. We denote $\pi_1 \approx_{lin} \pi_2$ if $\pi_1 \preceq_{lin} \pi_2$ and $\pi_2 \preceq_{lin} \pi_1$.

The relation between abstract and concrete atoms and substitutions is formalized as a Galois insertion the construction of which is completely analogous to that given in Section 4. We elaborate only that the abstraction of a term is formalized by:

$$\sigma(t) = \begin{cases} \emptyset & \text{if } vars(t) = \emptyset \\ \{\!|x_1 \oplus \ldots \oplus x_n|\!\} & \text{if } vars(t) = \{x_1, \ldots, x_n\} \\ & \text{and } linear(t) \\ \{x_1 \oplus \ldots \oplus x_n\} & \text{if } vars(t) = \{x_1, \ldots, x_n\} \\ & \text{and not } linear(t) \end{cases} \tag{19}$$

This definition is the straightforward extension of Equation (7) enhanced to specify the linearity information in a concrete term. The abstraction for substitutions is defined in the similar way.

Example 20

(1) $\sigma([X, X|Xs]) = \{X, Xs\}$;

(2) $\sigma(tree(X, Left, Right)) = \{\!|X, Left, Right|\!\}$;

(3) $\sigma(X) = \{\!|X|\!\}$;

(4) $\sigma([\ ]) = \emptyset$.

We say that an abstract atom (or substitution) π describes an atom (or a substitution) a, denoted $\pi \propto a$ if $\sigma(a) \preceq_{lin} \pi$.

Example 21

(1) $p(\{\!| X |\!\}, \{\!| Y |\!\}, \{X, Y\}) \propto p([X_1, X_2], Ys, [X_1, X_2|Ys])$ *and*
$p(\{\!| X |\!\}, \{\!| Y |\!\}, \{X, Y\}) \propto p([\,], Ys, Ys)$, *but*
$p(\{\!| X |\!\}, \{\!| Y |\!\}, \{X, Y\}) \not\propto p([X_1, X_1], Ys, [X_1, X_1|Ys])$ *since the first argument is not linear;*

(2) $p(\{X\}) \propto p(X)$ *and* $p(\{X\}) \propto p(a)$*;*

(3) $p(\{\!| X |\!\}) \propto p(X)$ *and* $p(\{\!| X |\!\}) \propto p(a)$*;*

(4) $p(\emptyset) \propto p(a)$ *but* $p(\emptyset) \not\propto p(X)$*.*

The operations on abstract atoms and substitutions with linearity information are straightforward extensions of the definitions in Section 5. For the operations of application, projection, composition and lub, this involves a straightforward case analysis. We only present here the definition for abstract lub.

Least Upper Bound: The least upper bound of two abstract atoms is based on the notion of union of two abstract terms.

Let τ_1 and τ_2 be two abstract terms with σ_1 and σ_2 being the corresponding ACI1-expressions, i.e., $\tau_i = \{\!| \sigma_i |\!\}$ or $\tau_i = \{\sigma_i\}$ for $i = 1, 2$. The union of τ_1 and τ_2, denoted by $\tau_1 \cup \tau_2$ is defined as:

$$\tau_1 \cup \tau_2 = \begin{cases} \emptyset & \text{if } \tau_1 = \emptyset \text{ and } \tau_2 = \emptyset \\ \{\!| \sigma_1 \oplus \sigma_2 |\!\} & \text{if } linear(\tau_1) \text{ and } linear(\tau_2) \\ \{\sigma_1 \oplus \sigma_2\} & \text{otherwise} \end{cases} \tag{20}$$

Example 22

(1) $\{A, B\} \cup \{\!| A, C |\!\} = \{A, B, C\}$ (2) $\{\!| A, B |\!\} \cup \{\!| C |\!\} \cup \emptyset = \{\!| A, B, C |\!\}$

The least upper bound for atoms (with respect to $\preceq_{lin}$) can be characterized by the following result, the proof of which is similar to that of Theorem 4.3.

Theorem 8.1 *For the abstract atoms* $\pi_1 = p(\tau_1, \ldots, \tau_n)$ *and* $\pi_2 = p(\tau'_1, \ldots, \tau'_n)$*:*

$$\pi_1 \sqcup \pi_2 = p(\tau_1 \cup \tau'_1, \ldots, \tau_n \cup \tau'_n).$$

Example 23

$$
\begin{array}{lll}
\textit{(1)} & \pi_1 & = p(\{A\},\ \{\!|B|\!\}, \{A,B\}) \\
& \pi_2 & = p(\{\!|Y|\!\}, \{\!|Y|\!\},\ \{\!|Y|\!\}) \\
\hline
& \pi_1 \sqcup \pi_2 & = p(\{A,Y\}, \{\!|B,Y|\!\}, \{A,B,Y\})
\end{array}
$$

$$
\begin{array}{lll}
\textit{(2)} & \pi_1 & = p(\ \emptyset,\ \ \{\!|Y|\!\}, \{\!|Y|\!\}) \\
& \pi_2 & = p(\{A\}, \{\!|B|\!\}, \{A,B\}) \\
\hline
& \pi_1 \sqcup \pi_2 & = p(\{A\}, \{\!|B,Y|\!\}, \{A,B,Y\}).
\end{array}
$$

Observe that $p(\{A\}, \{\!|B,Y|\!\}, \{A,B,Y\}) \approx_{lin} p(\{A\}, \{\!|B|\!\}, \{A,B\})$.

Abstract Unification: As illustrated by Example 15 abstract unification can give more precise results for set-sharing when linearity information is present. To formalize this we recall Lemma 2.3 which imposes additional constraints about linearity information for concrete unification problems. In particular, we recall that the most general unifier of two terms t_1 and t_2 is guaranteed to have a linear projection on the co-linear variables of t_1 and t_2. As a consequence, the abstract unifier for a pair of (annotated) abstract terms τ_1 and τ_2 can safely be chosen as their most general ACI1 unifier (with respect to $\preceq_{lin}$) which has a linear projection on their co-linear variables. Observe that if neither τ_1 nor τ_2 is annotated as linear then this boils down to the definition of $ir\text{-}mgu_{ACI1}$ from Section 5.

The algorithm depicted in Figure 4 computes the most general abstract unifier of two annotated abstract terms. It is based on the algorithm of ACI1-unification (Figure 2). For the cases when one term is linear the algorithm computes an annotated most general (with respect to $\preceq_{lin}$) ACI1 unifier with a linear projection on the second term. If two abstract terms are linear then the algorithm computes a most general ACI1 unifier with two linear projections. The case of unification of two non-linear terms is analogous to the unification performed in the algorithm of Figure 2. The set S computed in a first phase consists of sets of variables representing all possible sharing in a corresponding (concrete) unification. The formation of this set relies on the result of Lemma 2.3. For instance, if one of the terms is linear (second "else"), then each variable of the second term appears in exactly one member of S. This ensures that the projection of the abstract unifier on each of the co-linear variables (v_2) is a linear substitution.

Correctness of this algorithm is based on Theorem 5.2 with the additional restrictions on linearity provided by Lemma 2.3. Its optimality can also be proven, using the same principle as in the proof of Lemma 5.4; the complete proof can be found in Appendix A.

$\underline{\textbf{lin-mgu}_{\textbf{ACI1}}(\tau_1, \tau_2) :}$

$v_1 = vars(\tau_1)$

$v_2 = vars(\tau_2)$

if $(v_1 = \emptyset) \vee (v_2 = \emptyset)$ **then return** $\lambda x \in (v_1 \cup v_2).\ \emptyset$

else if τ_1 is linear **and** τ_2 is linear **and** $v_1 \cap v_2 = \emptyset$ **then**

$$S = \left\{ \ \{u, v\} \ \middle| \ u \in v_1,\ v \in v_2 \right\}$$

else if τ_1 is linear **and** $v_1 \cap v_2 = \emptyset$ **then**

$$S = \left\{ \ \{v\} \cup s \ \middle| \ v \in v_2,\ s \subseteq v_1,\ s \neq \emptyset \right\}$$

else if τ_2 is linear **and** $v_1 \cap v_2 = \emptyset$ **then**

$$S = \left\{ \ \{v\} \cup s \ \middle| \ v \in v_1,\ s \subseteq v_2,\ s \neq \emptyset \right\}$$

else

$$S = \left\{ \ s \subseteq (v_1 \cup v_2) \ \middle| \ s \cap v_1 \neq \emptyset,\ s \cap v_2 \neq \emptyset \right\}$$

let $S = \{s_1, \ \ldots, \ s_k\}$

$Z = \{z_1, \ \ldots, \ z_k\}$ // fresh variables

$$\textbf{return} \ \ \lambda x \in (v_1 \cup v_2). \begin{cases} \{\!| \bigoplus_{x \in s_i} z_i |\!\} & \text{if } x \text{ is co-linear} \\ \{ \bigoplus_{x \in s_i} z_i \} & \text{otherwise} \end{cases}$$

Fig. 4. Abstract unification of annotated terms

Example 24 *Let us demonstrate how the unification algorithm shown in Figure 4 is applied to compute a precise unifier for the abstract terms from Example 15. In this example we assumed the Y represents only linear terms, which means that in the annotated domain we consider the unification of $\{\!|Y|\!\}$ with $\{A, B\}$. Since both abstract terms are non-empty and have no variables in common, the set S computed by the algorithm is $S = \left\{ \{A, Y\},\ \{B, Y\} \right\}$. The members of S are labeled by fresh variables Z_1 and Z_2 respectively. Thus, the unifier computed by the algorithm is $\mu = \left\{ A \mapsto \{\!|Z_1|\!\},\ B \mapsto \{\!|Z_2|\!\},\ Y \mapsto \{Z_1, Z_2\} \right\}$. Note that μ does not introduce aliasing between A and B, which indeed cannot occur in the unification of corresponding concrete terms as discussed in Example 15.*

Unification of abstract atoms is defined as usual by incremental unification of corresponding abstract terms.

$$
mgu^{\mathcal{A}}_{lin}(\mathcal{E}) = \begin{cases} \varepsilon & \text{if } \mathcal{E} = \emptyset \\ \mu \circ mgu^{\mathcal{A}}_{lin}(\mathcal{E}'\mu) & \text{if } \mathcal{E} = \{\tau = \tau'\} \cup \mathcal{E}' \\ & \text{and } \mu = \mathit{lin\text{-}mgu}_{ACI1}(\tau, \tau') \end{cases} \tag{21}
$$

Example 25 *In the following examples, adapted from [8], we solve at each step the first (upper) equation in the set and apply the result to the other equations:*

(1) Consider the unification of abstract atoms: $\pi_1 = p(\{\!|A|\!\}, \{\!|A, B|\!\}, \{\!|B|\!\})$ *and* $\pi_2 = p(\{X\}, \{\!|Y|\!\}, \{Z\})$*.*

$$
mgu^{\mathcal{A}}_{lin}\left(\left\{\begin{array}{l} \{\!|A|\!\} = \{X\}, \\ \{\!|A, B|\!\} = \{\!|Y|\!\}, \\ \{\!|B|\!\} = \{Z\} \end{array}\right\}\right) =
$$

$$
= \left\{\begin{array}{l} A \mapsto \{Z_1\}, \\ X \mapsto \{\!|Z_1|\!\} \end{array}\right\} \circ mgu^{\mathcal{A}}_{lin}\left(\left\{\begin{array}{l} \{Z_1, B\} = \{\!|Y|\!\}, \\ \{\!|B|\!\} = \{Z\} \end{array}\right\}\right) =
$$

$$
= \left\{\begin{array}{l} A \mapsto \{Z_1\}, \\ X \mapsto \{\!|Z_1|\!\} \end{array}\right\} \circ \left\{\begin{array}{l} B \mapsto \{\!|Z_2|\!\}, \\ Y \mapsto \{Z_2, Z_3\}, \\ Z_1 \mapsto \{\!|Z_3|\!\} \end{array}\right\} \circ mgu^{\mathcal{A}}_{lin}(\{\!|Z_2|\!\} = \{Z\}) =
$$

$$
= \left\{\begin{array}{l} A \mapsto \{Z_1\}, \\ X \mapsto \{\!|Z_1|\!\} \end{array}\right\} \circ \left\{\begin{array}{l} B \mapsto \{\!|Z_2|\!\}, \\ Y \mapsto \{Z_2, Z_3\}, \\ Z_1 \mapsto \{\!|Z_3|\!\} \end{array}\right\} \circ \left\{\begin{array}{l} Z \mapsto \{\!|Z_4|\!\}, \\ Z_2 \mapsto \{Z_4\} \end{array}\right\}
$$

which gives:

$$
mgu^{\mathcal{A}}(\pi_1, \pi_2) = \left\{\begin{array}{l} X \mapsto \{\!|Z_3|\!\},\ Y \mapsto \{Z_4, Z_3\},\ Z \mapsto \{\!|Z_4|\!\} \\ A \mapsto \{Z_3\},\ B \mapsto \{Z_4\} \end{array}\right\}.
$$

Notice that there is no aliasing between A *and* B *and that* X *and* Z *are bound to linear terms.*

(2) Consider the unification of abstract atoms: $\pi_1 = p(\{\!|A|\!\}, \{\!|A, B|\!\}, \{\!|B|\!\})$ *and*

$\pi_2 = p(\{X\},\{Y\},\{Z\})$.

$$mgu^{\mathcal{A}}_{lin}\left(\left\{\begin{array}{l} \{\!|A|\!\} = \{X\}, \\ \{\!|A,B|\!\} = \{Y\}, \\ \{\!|B|\!\} = \{Z\} \end{array}\right\}\right) =$$

$$= \left\{\begin{array}{l} A \mapsto \{Z_1\}, \\ X \mapsto \{\!|Z_1|\!\} \end{array}\right\} \circ mgu^{\mathcal{A}}_{lin}\left(\left\{\begin{array}{l} \{Z_1, B\} = \{Y\}, \\ \{\!|B|\!\} = \{Z\} \end{array}\right\}\right) =$$

$$= \left\{\begin{array}{l} A \mapsto \{Z_1\}, \\ X \mapsto \{\!|Z_1|\!\} \end{array}\right\} \circ \left\{\begin{array}{l} Y \mapsto \{Z_2, Z_3, W\}, \\ Z_1 \mapsto \{Z_3, W\}, \\ B \mapsto \{Z_2, W\} \end{array}\right\} \circ mgu^{\mathcal{A}}_{lin}\left(\{Z_2, W\} = \{Z\}\right) =$$

$$= \left\{\begin{array}{l} A \mapsto \{Z_1\}, \\ X \mapsto \{\!|Z_1|\!\} \end{array}\right\} \circ \left\{\begin{array}{l} Y \mapsto \{Z_2, Z_3, W\}, \\ Z_1 \mapsto \{Z_3, W\}, \\ B \mapsto \{Z_2, W\} \end{array}\right\} \circ \left\{\begin{array}{l} Z \mapsto \{Z_4, Z_5, W'\}, \\ Z_2 \mapsto \{Z_4, W'\}, \\ W \mapsto \{Z_5, W'\} \end{array}\right\}$$

which gives:

$$mgu^{\mathcal{A}}_{lin}(\pi_1, \pi_2) = \left\{\begin{array}{l} X \mapsto \{Z_3, Z_5\},\ Y \mapsto \{Z_3, Z_4, Z_5\}, \\ Z \mapsto \{Z_4, Z_5\},\ A \mapsto \{Z_3, Z_5\},\ B \mapsto \{Z_4, Z_5\} \end{array}\right\}$$

in which W' collapses to Z_5 because of equivalence. Notice that there is (possible) aliasing between A and B and that X and Z are bound to non-linear terms.

Again, the correctness and optimality (and confluence) of the abstract unification for sharing analysis follow naturally. The proofs are similar to those of theorems 5.6 and 5.7.

9 Conclusion

We have described an algebraic approach for the sharing analysis of logic programs based on an abstract domain of *set logic programs*. The main advantage of this approach is that the specification of the abstract unification algorithm relies on the well-studied notion of ACI1-unification. The justification of the abstract operations needed to define a sharing analysis all follow a clear and intuitive argument based on simple algebraic properties of set substitutions and set-based atoms. We have given full proofs of correctness and optimality for these operations and we have proven that the well-known set-sharing domain of Jacobs and Langen is isomorphic to our domain. We do not know if the abstract operations defined by Jacobs and Langen are optimal (the authors have not proven this). But, in case they are not, then this paper provides optimal abstract operations for the set-sharing domain via the domain isomorphism. Another advantage of our approach is the simplicity in which it is extended with linearity information. Finally we note that the approach described in this paper facilitates implementation based on abstract compilation — be it in a top-down or in a bottom-up approach.

We close the paper with two proposals for future work:

(1) It would be interesting to cast the algebraic framework demonstrated in this paper in terms of a *generalized constraint system*, following [22]. This is for example the approach in [34], where (groundness and type) analyses are designed as constraint solving. In particular, it would be interesting to consider the implementation of the approach described here using recent developments in set logic programming.

(2) It would be interesting to investigate the application of our approach for pair sharing analysis based on the results of [3]. In that paper, the authors prove that the set sharing domain is over complex for the analysis of pair sharing (i.e. to answer the question: “do variables X and Y share?”. In particular the authors show how to avoid using the expensive star-closure operation of set sharing and hence to obtain polynomial abstract algorithms.

Appendix A: Proofs

We first discuss the operations of abstract application, composition and lub. The role of abstract application is to extract the sharing information expressed by an abstract substitution which is relevant for a given (abstract) atom or other syntactic object.

Lemma A.1 (application of an abstract substitution is correct) *Let a, θ, π and μ be concrete and abstract atoms and substitutions such that $\pi \propto a$ and $\mu \propto \theta$. Then $\pi\mu \propto a\theta$.*

PROOF.

$$\begin{array}{lll} \sigma(a\theta) & = & \text{[by Lemma 4.2]} \\ \sigma(a)\cdot\sigma(\theta) & \preceq_{ir} & \text{[because } \mu \propto \theta\text{]} \\ \sigma(a)\mu & \preceq_{ir} & \text{[because } \pi \propto a\text{]} \\ \pi\mu & \Rightarrow & \text{[by definition of } \propto\text{]} \end{array}$$

$$\pi\mu \propto a\theta \qquad \Box$$

Lemma A.2 (application of an abstract substitution is optimal) *Let a and π be concrete and abstract atoms such that $\sigma(a) = \pi$, and θ and μ be concrete and abstract substitutions such that $\sigma(\theta) = \mu$. There is no abstract atom π' (not equivalent to $\pi\mu$) such that $\pi' \propto a\theta$ and $\pi' \preceq_{ir} \pi\mu$.*

PROOF. By Lemma 4.2 we have $\sigma(a\theta) = \sigma(a)\cdot\sigma(\theta) = \pi\mu$. If $\pi' \propto a\theta$ then $\sigma(a\theta) \preceq_{ir} \pi'$, and therefore $\pi\mu \preceq_{ir} \pi'$. $\Box$

Correctness and optimality of composition for abstract substitutions are implied by the corresponding results for abstract application as established by the following two lemmata.

Lemma A.3 (composition of abstract substitutions is correct) *Let θ_1 and θ_2 be concrete substitutions, and μ_1 and μ_2 abstract substitutions such that $\mu_1 \propto \theta_1$ and $\mu_2 \propto \theta_2$. Then assuming $dom(\mu_1) = dom(\theta_1)$ and $dom(\mu_2) = dom(\theta_2)$, $(\mu_1 \circ \mu_2) \propto (\theta_1 \circ \theta_2)$.*

PROOF. Let $\bar{x}$ be a tuple of variables of interest. Lemma A.1 implies that $\bar{x}\mu_1 \propto \bar{x}\theta_1$ and $(\bar{x}\mu_1)\mu_2 \propto (\bar{x}\theta_1)\theta_2$. Thus $\bar{x}\cdot(\mu_1 \circ \mu_2) \propto \bar{x}\cdot(\theta_1 \circ \theta_2)$, which implies the lemma statement. $\Box$

Lemma A.4 (composition of abstract substitutions is optimal)
Let θ_1 and θ_2 be concrete substitutions, and μ_1 and μ_2 abstract substitutions such that $\mu_1 \propto \theta_1$ and $\mu_2 \propto \theta_2$. There is no abstract substitution μ' (not equivalent to $\mu_1 \circ \mu_2$) such that $\mu' \propto (\theta_1 \circ \theta_2)$ and $\mu' \preceq_{ir} (\mu_1 \circ \mu_2)$.

PROOF. Follows from the optimality of abstract application established by Lemma A.1 similarly as the correctness of abstract composition (Lemma A.3) follows from the correctness of abstract application. $\square$

Lemma A.5 (abstract lub is correct)
For abstract atoms π_1 and π_2 and concrete atoms a_1 and a_2,

$$(\pi_1 \propto a_1) \wedge (\pi_2 \propto a_2) \;\Rightarrow\; ((\pi_1 \sqcup \pi_2) \propto a_1) \wedge ((\pi_1 \sqcup \pi_2) \propto a_2).$$

PROOF. Since $\pi_1 \sqcup \pi_2$ is an upper bound of π_1 and π_2 with respect to $\preceq_{ir}$ we have $\pi_1 \preceq_{ir} (\pi_1 \sqcup \pi_2)$ and $\pi_2 \preceq_{ir} (\pi_1 \sqcup \pi_2)$. Because $\pi_1 \propto a_1$ and $\pi_2 \propto a_2$ we have $\sigma(a_1) \preceq_{ir} \pi_1$ and $\sigma(a_2) \preceq_{ir} \pi_2$. Thus, $\sigma(a_1) \preceq_{ir} (\pi_1 \sqcup \pi_2)$ and $\sigma(a_2) \preceq_{ir} (\pi_1 \sqcup \pi_2)$, i.e., $\pi_1 \sqcup \pi_2 \propto a_1$ and $\pi_1 \sqcup \pi_2 \propto a_2$. $\square$

Lemma A.6 (abstract lub is optimal)
For abstract atoms π_1 and π_2 and concrete atoms a_1 and a_2, such that $\pi_1 \propto a_1$ and $\pi_2 \propto a_2$, there is no abstract atom π' (not equivalent to $\pi_1 \sqcup \pi_2$) such that $\pi' \propto a_1$, $\pi' \propto a_2$, and $\pi' \preceq_{ir} \pi_1 \sqcup \pi_2$.

PROOF. Straightforward since $\pi_1 \sqcup \pi_2$ is a *least* upper bound of π_1 and π_2 with respect to $\preceq$ as established by Theorem 4.3. $\square$

Similar results for the operations on abstract atoms and substitutions with linearity information can be obtained. The above proofs are on the whole easy to enhance for this purpose. To justify correctness and optimality we have only to focus on the added linearity information, which involves a straightforward case analysis, which we omit here.

We now consider abstract unification. We first claim that ACI1-unification of a single equation in a context (of a set of equations for the unification of atoms) is well defined. Namely, that it does not depend on the particular representative element.

Observation A.7 *Consider abstract terms τ_1, τ_2, ρ_1, and ρ_2, such that $vars(\tau_1) \subseteq vars(\rho_1)$ and $vars(\tau_2) \subseteq vars(\rho_2)$, and denote $\mu = ir\text{-}mgu_{ACI1}(\tau_1, \tau_2)$ and $\mu' = ir\text{-}mgu_{ACI1}(\rho_1, \rho_2)$. Consider the abstract substitutions ψ and φ:*

$$\psi = \left\{ \; x \mapsto \emptyset \;\middle|\; x \in (vars(\rho_1) \setminus vars(\tau_1)) \cup (vars(\rho_2) \setminus vars(\tau_2)) \; \right\},$$

$$\varphi = \left\{ \, x \mapsto \emptyset \,\middle|\, \exists y \in range(\mu').\ (y\varphi = \emptyset) \wedge (x \in vars(y\mu')) \,\right\}.$$

Then, $\mu' \circ \varphi$ is a renamed instance of $\psi \circ \mu$. To justify this observation, let us consider the case when $\rho_1 = \tau_1 \oplus z$ (where z is a fresh variable) and $\rho_2 = \tau_2$. So $\psi = \left\{ \, z \mapsto \emptyset \,\right\}$ and $\varphi = \left\{ \, x \mapsto \emptyset \,\middle|\, x \in vars(z\mu') \,\right\}$.

Consider the set S in the evaluation of μ' following the algorithm of Figure 2. The variables in $z\mu'$ are the labels for the sets in S which contain z. Thus, $\mu'\varphi$ is the same as μ except that it maps z to $\emptyset$. In other words, $\mu' \circ \varphi$ is a renamed instance of $\psi \circ \mu$.

Lemma A.8 (unification of abstract terms is well defined)
Let $\mathcal{E}$ be an equivalence class of abstract equations with representative elements $\hat{\mathcal{E}} = \langle \tau_1 = \tau'_1, \ldots, \tau_n = \tau'_n \rangle$ and $\tilde{\mathcal{E}} = \langle \rho_1 = \rho'_1, \ldots, \rho_n = \rho'_n \rangle$ such that $\hat{\mathcal{E}} \approx_{ir} \tilde{\mathcal{E}}$. Then

$$\hat{\mathcal{E}} \cdot ir\text{-}mgu_{ACI1}(\tau_i, \tau'_i) \approx_{ir} \tilde{\mathcal{E}} \cdot ir\text{-}mgu_{ACI1}(\rho_i, \rho'_i).$$

PROOF. We may assume without loss of generality that $\hat{\mathcal{E}}$ is a "minimal" context for the abstract unification, i.e., a context constructed from two tuples of abstract terms minimized as in the proof of Theorem 3.1. Since the lemma trivially holds for contexts which are renamed instances of one another, we may assume that $vars(\tau_i) \subseteq vars(\rho_i)$ and $vars(\tau'_i) \subseteq vars(\rho'_i)$ for any i and that variables from $vars(\rho_i) \setminus vars(\tau_i)$ and from $vars(\rho'_i) \setminus vars(\tau'_i)$ do not occur in $\hat{\mathcal{E}}$. Thus, a ground substitution ϕ providing $\hat{\mathcal{E}} = \tilde{\mathcal{E}}\varphi$ is:

$$\psi = \left\{ \, x \mapsto \emptyset \,\middle|\, x \in (vars(\rho_i) \setminus vars(\tau_i)) \cup (vars(\rho'_i) \setminus vars(\tau'_i)) \,\right\}.$$

Let us denote $ir\text{-}mgu_{ACI1}(\tau_i, \tau'_i)$ by $\hat{\mu}$ and $ir\text{-}mgu_{ACI1}(\rho_i, \rho'_i)$ by $\tilde{\mu}$. Then

$$\begin{array}{lll}
\tilde{\mathcal{E}}\tilde{\mu} & = & [\text{because } \tilde{\mathcal{E}} \approx_{ir} \hat{\mathcal{E}} \text{ and } vars(\hat{\mathcal{E}}) \subseteq vars(\tilde{\mathcal{E}})] \\
\hat{\mathcal{E}}\tilde{\mu} & \approx_{ir} & [\text{because } \mathcal{E} \text{ has no occurrences of variables in } dom(\psi)] \\
\hat{\mathcal{E}}\psi\tilde{\mu} & \approx_{ir} & [\text{by Observation A.7}] \\
\hat{\mathcal{E}}\psi\hat{\mu} & \approx_{ir} & [\text{because } \mathcal{E} \text{ has no occurrences of variables of } dom(\psi)] \\
\hat{\mathcal{E}}\hat{\mu} & \square &
\end{array}$$

We now prove the correctness of ACI1-unification of abstract terms in the context of a set of equations between abstract terms.

Observation A.9 *Consider abstract terms τ_1, τ_2, ρ_1, and ρ_2, such that $vars(\tau_1) \subseteq vars(\rho_1)$ and $vars(\tau_2) \subseteq vars(\rho_2)$, and denote $\mu = ir\text{-}mgu_{ACI1}(\tau_1, \tau_2)$ and $\mu' =$*

ir-mgu$_{ACI1}(\rho_1, \rho_2)$. *Consider the abstract substitution:*

$$\psi = \left\{ x \mapsto \emptyset \,\middle|\, x \in (vars(\rho_1) \setminus vars(\tau_1)) \cup (vars(\rho_2) \setminus vars(\tau_2)) \right\}.$$

Then $\mu' \circ \psi$ *is equivalent to* μ. *To justify this consider again the unification of* $\rho_1 = \tau_1 \oplus z$ *with* $\rho_2 = \tau_2$, *as in Observation A.7.*

It is easy to see that the same variable sets share in μ' *and in* μ, *except for the occurrence of variable* z. *Since* $\mu' \circ \left\{ z \mapsto \emptyset \right\}$ *makes* z *ground —and thus it cannot share through any variable—, then the set-sharing of* μ *and* $\mu' \circ \left\{ z \mapsto \emptyset \right\}$ *is the same, i.e.,* $\mathcal{A}(\mu' \circ \left\{ z \mapsto \emptyset \right\}) = \mathcal{A}(\mu)$, *and consequently, by Lemma 6.1,* $\mu' \circ \left\{ z \mapsto \emptyset \right\} \approx_{ir} \mu$.

Lemma 5.5 (ACI1-unification of abstract terms is safe)
Let $\pi = p(\tau_1, \ldots, \tau_n)$, $\pi' = p(\tau_1', \ldots, \tau_n')$, $a = p(t_1, \ldots, t_n)$ *and* $a' = p(t_1', \ldots, t_n')$ *such that* $\pi \propto a$ *and* $\pi' \propto a'$. *Then for* i, $1 \le i \le n$:

$$\pi \cdot ir\text{-}mgu_{ACI1}(\tau_i, \tau_i') \propto a \cdot mgu(t_i, t_i').$$

PROOF. Assume without loss of generality that π and π' are representative elements such that $vars(\tau_i) \supseteq vars(t_i)$, $vars(\tau_i') \supseteq vars(t_i')$, and variables from $vars(\tau_i) \setminus vars(t_i)$ and from $vars(\tau_i') \setminus vars(t_i')$ do not occur in a.

Let $\zeta = ir\text{-}mgu_{ACI1}(\sigma(t_i), \sigma(t_i'))$, $\mu = ir\text{-}mgu_{ACI1}(\tau_i, \tau_i')$, and $\theta = mgu(t_i, t_i')$. From Observation A.9 there is a ground substitution ψ such that $\zeta \circ \psi \approx_{ir} \mu$. Also, from Lemma 5.3, $\sigma(a\theta) \preceq_{ir} \sigma(a)\zeta$.

Since $\pi \propto a$ we have $\sigma(a) \preceq_{ir} \pi$, and therefore, $\sigma(a)\mu \preceq_{ir} \pi\mu$. Since $\zeta \circ \psi \approx_{ir} \mu$ we have $\sigma(a) \cdot (\zeta \circ \psi) \preceq_{ir} \pi\mu$. Since variables of $dom(\psi)$ do not occur in a we get $\sigma(a)\zeta \preceq_{ir} \pi\mu$. Since $\sigma(a\theta) \preceq_{ir} \sigma(a)\zeta$, and by transitivity of $\preceq_{ir}$, it follows $\sigma(a\theta) \preceq_{ir} \pi\mu$. $\square$

As we have seen, the above result is instrumental in the proof of correctness of abstract unification for set-sharing analysis (Theorem 5.6). The above proof can be easily enhanced for the case of including linearity information. The corresponding lemma leads us to a correctness result for sharing analysis of the abstract unification with linearity, in the same way as that of Theorem 5.6. We now turn our attention to optimality.

Lemma 5.4 (ACI1-unification of abstract terms is optimal)
For abstract terms τ_1 *and* τ_2 *and abstract unifier* $\mu = ir\text{-}mgu_{ACI1}(\tau_1, \tau_2)$, *and for any* μ' *which is (strictly) less general than* μ, *there exist concrete terms* t_1 *and* t_2 *such that* $\sigma(t_1) = \tau_1$, $\sigma(t_2) = \tau_2$, *and* $\mu' \not\propto mgu(t_1, t_2)$.

PROOF. If at least one of τ_1 and τ_2 equals to $\emptyset$ then μ binds all variables found in τ_1 and τ_2 to $\emptyset$, thus, precisely approximating the result of concrete unification when one or both terms are ground. For this case μ is trivially optimal.

Now consider the case when both τ_1 and τ_2 have variables. Assume by contradiction that there exists μ' which is more precise than μ such that $\mu' \propto mgu(t_1, t_2)$. Hence, $\mu' \preceq_{ir} \mu$ and $\mu \neq \mu'$, and by Observation 2, there exists a variable $z \in vars(range(\mu))$ such that $\mu' \preceq \mu \circ \left\{ z \mapsto \emptyset \right\}$. We assume without loss of generality that z is a fresh variable not occurring in τ_1 and τ_2.

Let (without loss of generality) $occs(\mu, z) \cap vars(\tau_1) = \{x_1, \ldots, x_m\}$ and $occs(\mu, z) \cap vars(\tau_2) = \{y_1, \ldots, y_p\}$. Since μ is a unifier these sets are surely nonempty. Denote the variables in τ_1 which are not in $occs(\mu, z)$ by $\{x_{m+1}, \ldots, x_n\}$ and the variables in τ_2 which are not in $occs(\mu, z)$ by $\{y_{p+1}, \ldots, y_q\}$. Let us construct the following concrete terms:

$$
\begin{aligned}
t_1 &= s(e(x_1, \ldots, x_m), f(x_1, \ldots, x_1), g(x_{m+1}, \ldots, x_n), \quad h(a, \ldots, a) \quad) \\
t_2 &= s(\ e(y_1, \ldots, y_1),\ f(y_1, \ldots, y_p), \quad g(a, \ldots, a), \quad h(y_{p+1}, \ldots, y_q)).
\end{aligned}
$$

Clearly, $\sigma(t_1) = \tau_1$ and $\sigma(t_2) = \tau_2$. Note that:

$$
mgu(t_1, t_2) = \left\{ \begin{array}{l} x_1 \mapsto w, \ldots, x_m \mapsto w, y_1 \mapsto w, \ldots, y_p \mapsto w, \\ x_{m+1} \mapsto a, \ldots, x_n \mapsto a, y_{p+1} \mapsto a, \ldots, y_q \mapsto a \end{array} \right\}
$$

and

$$
\sigma(mgu(t_1, t_2)) = \left\{ \begin{array}{l} x_1 \mapsto w, \ldots, x_m \mapsto w, y_1 \mapsto w, \ldots, y_p \mapsto w, \\ x_{m+1} \mapsto \emptyset, \ldots, x_n \mapsto \emptyset, y_{p+1} \mapsto \emptyset, \ldots, y_q \mapsto \emptyset \end{array} \right\}.
$$

It is easy to see that $\mu \propto \theta$, or equivalently $\mu \circ \psi = \sigma(\theta)$ with ψ the following independent-range substitution:

$$
\psi = \lambda x. \begin{cases} w & \text{if } x = z \\ \emptyset & \text{otherwise.} \end{cases}
$$

Let us demonstrate now that $\mu' \not\propto \theta$. Assume that there exists an independent-range substitution ψ' satisfying $\mu' \circ \psi' = \sigma(\theta)$. This substitution is of the form:

$$
\psi' = \lambda x. \begin{cases} w & \text{if } x = y \\ \emptyset & \text{otherwise} \end{cases}
$$

for some variable $y \in range(\mu')$. Note that since ψ' is an independent-range substitution, only one variable in the range of μ' can be mapped by ψ' to w. We may assume that both μ and μ' are in their "minimal" form, i.e., each variable in these substitutions occurs in a distinct set of terms in their ranges. Consequently, μ' has no variable z' such that $occs(\mu, z) = occs(\mu', z')$. It follows that for any choice of y in the above ψ', the unifiers $\mu \circ \psi$ and $\mu' \circ \psi'$ are different. Thus, for any independent-range substitution ψ', $\mu' \circ \psi' \neq \sigma(\theta)$. Therefore, μ' is not a correct abstract unifier. The contradiction implies that μ is a most precise abstract unifier of τ_1 and τ_2. $\square$

Theorem 5.7 (abstract unification is optimal for set-sharing)
Let $\mathcal{E}$ be a set of abstract equations and denote $\mu = mgu^{\mathcal{A}}(\mathcal{E})$. There is no unifier μ' for $\mathcal{E}$ which is more precise than μ, i.e., such that $\mu' \preceq_{ir} \mu$ and $\mu \not\preceq_{ir} \mu'$, which is also correct for set-sharing.

PROOF. Assume by contradiction that there exists another unifier μ' for $\mathcal{E}$ such that $\mu' \preceq_{ir} \mu$ and $\mu \not\preceq_{ir} \mu'$. Thus, by Observation 2 there exists a variable $z \in vars(range(\mu))$ such that $\mu' \preceq_{ir} \mu \circ \left\{ z \mapsto \emptyset \right\}$. Let $\mathcal{E} = \left\{ e_1, \ldots, e_n \right\}$. By Equation (14), $\mu = \mu_1 \circ \mu_2 \circ \cdots \circ \mu_n$, where $\mu_1 = ir\text{-}mgu_{ACI1}(e_1)$ and $\mu_i = ir\text{-}mgu_{ACI1}(e_i \mu^{i-1})$, and $\mu^{i-1} = \mu_1 \circ \mu_2 \circ \cdots \circ \mu_{i-1}$, for $i = 2, \ldots, n$.

Let μ_k be the first substitution such that $z \in vars(range(\mu_k))$. Let $\mathcal{E}_k = \left\{ e_{k+1}, \ldots, e_n \right\}$. Thus, on step k of the resolution in Equation (14) we have $\mu = \mu^{k-1} \circ \mu_k \circ mgu^{\mathcal{A}}\left(\mathcal{E}_k \mu^{k-1} \mu_k\right)$.

Note that z appears in the range of μ and thus, the steps from $k+1$ to n do not compute any bindings for z. Therefore, applying the substitution $\left\{ z \mapsto \emptyset \right\}$ to μ at step k is equivalent to applying it at the end of the resolution process. Since $\mu' \preceq_{ir} \mu \cdot \left\{ Z \mapsto \emptyset \right\}$, there must hold one of the following possibilities:

(1) $\mu' \preceq_{ir} \mu^{k-1} \circ \left(\mu_k \circ \left\{ z \mapsto \emptyset \right\}\right) \circ mgu^{\mathcal{A}}\left(\mathcal{E}_k \cdot \left(\mu_k \circ \left\{ z \mapsto \emptyset \right\}\right)\right)$,
if $\left\{ z \mapsto \emptyset \right\}$ is applied to $ir\text{-}mgu(e_k)$, i.e., to μ_k;

(2) $\mu' \preceq_{ir} \mu^{k-1} \circ \mu_k \circ mgu^{\mathcal{A}}\left((\mathcal{E}_k \mu_k) \cdot \left\{ z \mapsto \emptyset \right\}\right)$,
if $\left\{ z \mapsto \emptyset \right\}$ is applied to the result of application of μ_k to $\mathcal{E}_k$;

(3) $\mu' \preceq_{ir} \mu^{k-1} \circ \mu_k \circ \left\{ z \mapsto \emptyset \right\} \circ mgu(\mathcal{E}_k \mu_k)$,
if $\left\{ z \mapsto \emptyset \right\}$ is applied to the result of the composition of μ^{k-1} with μ_k.

If one of these possibilities holds then the corresponding "atomic" operation on step k, i.e., ACI1-unification, application, or composition, admits a more precise result.

But this contradicts one of the Lemmas 5.4, A.2 and A.4 establishing the optimality of all operations used in the abstract unification. □

Again, for the case of including linearity similar optimality results are obtained. We include here the prove of the lemma for the optimality of abstract unification of terms (with linearity). A result similar to Theorem 5.7 for abstract unification of atoms is obtained using this lemma, much in the same way as in the case of Theorem 5.7.

Lemma A.10 (optimality of $lin\text{-}mgu_{ACI1}$)
For abstract terms τ_1 and τ_2 and abstract unifier $\mu = lin\text{-}mgu_{ACI1}(\tau_1, \tau_2)$, and for any μ' which is (strictly) less general than μ, there exist concrete terms t_1 and t_2 such that $\sigma(t_1) = \tau_1$, $\sigma(t_2) = \tau_2$, and $\mu' \not\propto mgu(t_1, t_2)$.

PROOF. First we consider whether the terms being unified have variables in common or not. If they have, we have seen in Example 8 that any possible set-sharing can appear during the unification of these terms. In this case $lin\text{-}mgu_{ACI1}$ defaults to $ir\text{-}mgu_{ACI1}$, and therefore the proof is a case of Lemma 5.4, except for the annotation of terms in the range of the unifier. Moreover, the terms variables in this case can be bound to non-linear terms, which is also demonstrated in Example 8. But this is exactly what $lin\text{-}mgu_{ACI1}$ does in this case.

If the terms do not have variables in common then either (1) they are both non-linear, (2) one of them is linear but the other is not, or (3) both are linear. If both are non-linear $lin\text{-}mgu_{ACI1}$ defaults again to $ir\text{-}mgu_{ACI1}$, and the claim follows directly from Lemma 5.4. Let us now prove the optimality of unification of abstract terms for the cases when $lin\text{-}mgu_{ACI1}$ is more precise than $ir\text{-}mgu_{ACI1}$, i.e., (2) and (3).

We prove (2); the prove of (3) is similar. Assume without loss of generality that τ_1 is non-linear and τ_2 is linear. Let $\mu = lin\text{-}mgu_{ACI1}(\tau_1, \tau_2)$ and $\theta = mgu(t_1, t_2)$. Assume by contradiction that there exists a more precise unifier μ' of τ_1 and τ_2 such that $\mu' \propto \theta$. The substitution μ' can be more precise than μ if it exhibits less set-sharing and/or more linearity than μ.

Let us consider linearity first. The projection of μ on variables of τ_2 is a linear substitution (since by $lin\text{-}mgu_{ACI1}$ μ is linear, and τ_2 also is). Observe that if the projection of μ' on the variables of τ_2 is a non-linear substitution then $linearity(\mu) \supseteq linearity(\mu')$ and thus, μ' is not more precise than μ. Thus, both projections of μ and μ' on the variables of τ_2 are linear substitutions. Because of this, w.r.t. linearity only τ_1 needs be considered.

If $linearity(\mu') \supset linearity(\mu)$ then μ' maps some variables of τ_1 to linear terms. In this situation a contradiction is easily obtained by constructing a linear concrete term t_1 and a non-linear term t_2 such that $\sigma(t_1) = \tau_1$, $\sigma(t_2) = \tau_2$, and the unification of t_1 with t_2 binds all variables of t_1 to non-linear terms. It follows that if μ' is a correct unifier then $linearity(\mu) \supseteq linearity(\mu')$ and consequently, if μ' is an optimal unifier then $linearity(\mu) = linearity(\mu')$.

Now let us consider the case when μ' is more precise than μ because it introduces less set-sharing. In this case there exists at least one variable z in the domain of μ such that $\mu' \preceq_{lin} \mu \circ \left\{ z \mapsto \emptyset \right\}$. Assume that

$$\tau_1 = x_1 \oplus \ldots \oplus x_n,$$
$$\tau_2 = y_1 \oplus \ldots \oplus y_p,$$

such that (without loss of generality) $occs(\mu, z) \cap vars(\tau_1) = \{x_1, \ldots, x_m\}$, $m \leq n$, and $occs(\mu, z) = \{y_1\}$ (recall that according to $lin\text{-}mgu_{ACI1}$ on Figure 4 only one variable of τ_2 occurs through each variable in the range of μ). Consider the following concrete terms:

$$t_1 = f(x_1,\ \ldots,\ x_m,\ g,\ \ \ldots,\ g\ \ x_{m+1},\ \ldots,\ x_n)$$
$$t_2 = f(y_1,\ \ldots,\ y_1,\ y_2,\ \ldots,\ y_p,\ \ g,\ \ \ \ldots,\ g).$$

As we can see, $\sigma(t_1) = \tau_1$, $\sigma(t_2) = \tau_2$ and t_1 is linear. The unifier θ of t_1 and t_2 binds all variables $x_1, \ldots, x_m$ to some variable w and binds all other variables to ground terms. It is easy to see that $\mu \propto \theta$, observing that $\mu \circ \psi = \sigma(\theta)$ where

$$\psi = \lambda x. \begin{cases} \{\!|w|\!\} & \text{if } x = z \\ \emptyset & \text{otherwise.} \end{cases}$$

The rest of the proof is the same as for Lemma 5.4. We demonstrate that $\mu' \not\propto \theta$ by showing that there is no linear substitution ψ' for which $\sigma(\theta) = \mu' \circ \psi'$, and thus, μ' is not a correct unifier. From this contradiction we conclude that μ is an optimal abstract unifier of τ_1 and τ_2. $\square$

References

[1] K.R. Apt. Introduction to Logic Programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Model and Semantics, pages 495–574. Elsevier, Amsterdam and The MIT Press, Cambridge, 1990.

[2] F. Baader and J. Siekmann. Unification theory. In C. Hogger, D. Gabbay, and J. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 2, pages 41–126. Oxford Science Publications, 1994.

[3] R. Bagnara, P. Hill, and E. Zaffanella. Set-sharing is redundant for pair-sharing. In P. Van Hentenryck, editor, *Proceedings of the Fourth International Static Analysis Symposium*, volume 1302 of *LNCS*, pages 53–67. Springer Verlag, September 1997.

[4] A. Bossi, M. Gabbrielli, G. Levi, and M. Martelli. The s-semantics approach: Theory and applications. *Journal of Logic Programming*, 19/20:149–198, July 1994.

[5] M. Bruynooghe and M. Codish. Freeness, Sharing, Linearity and Correctness — all at Once. In P. Cousot, M. Falaschi, G. Filè, and A. Rauzy, editors, *Third International Workshop on Static Analysis WSA'93*, volume 724 of *LNCS*, pages 153–164. Springer Verlag, September 1993.

[6] M. Codish, D. Dams, G. Filè, and M. Bruynooghe. Freeness Analysis for Logic Programs – and Correctness. In David S. Warren, editor, *Proceedings of the Tenth International Conference on Logic Programming*, pages 116–131. The MIT Press, June 1993.

[7] M. Codish, D. Dams, G. Filé, and M. Bruynooghe. On the design of a correct freeness analysis for logic programs. *Journal of Logic Programming*, 28(3):181–206, September 1996.

[8] M. Codish, D. Dams, and E. Yardeni. Derivation and Safety of an Abstract Unification Algorithm for Groundness and Aliasing Analysis. In Koichi Furukawa, editor, *Proceedings of the Eighth International Conference on Logic Programming*, pages 79–93. The MIT Press, 1991.

[9] M. Codish and B. Demoen. Analyzing logic programs using "PROP"-ositional logic programs and a magic wand. *Journal of Logic Programming*, 25(3):249–274, December 1995.

[10] M. Codish, B. Demoen, and K. Sagonas. General purpose semantic based analysis using XSB. Technical report, Ben-Gurion University of the Negev, January 1997. `ftp://ftp.cs.bgu.ac.il/pub/people/codish/absxsb.ps`.

[11] M. Codish and V. Lagoon. Type dependencies for logic programs using ACI-unification. *Journal of Theoretical Computer Science*. (accepted for publication).

[12] M. Codish, V. Lagoon, and F. Bueno. An algebraic approach to sharing analysis of logic programs. In P. Van Hentenryck, editor, *Proceedings of the Fourth International Static Analysis Symposium*, volume 1302 of *LNCS*, pages 68–82. Springer Verlag, September 1997.

[13] 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, January 1995.

[14] P. Codognet and G. Filè. Computations, Abstractions and Constraints. In *Proc. Fourth IEEE Int'l Conference on Computer Languages*. IEEE Press, 1992.

[15] A. Cortesi and G. Filé. Abstract Interpretation of Logic Programs: an Abstract Domain for Groundness, Sharing, Freeness and Compoundness Analysis. In P. Hudak and N. D. Jones, editors, *Proceedings of the ACM SIGPLAN Symposium on partial evaluation and semantics based program manipulation, PEPM'91*, number 26 in Sigplan notices, pages 52–61. ACM, 1991.

[16] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc., Fourth ACM Symp. on Principles of Programming Languages*, pages 238–252. ACM Press, January 1977.

[17] P. Cousot and R. Cousot. Systematic Design of Program Analysis Frameworks. In *Proc. Sixth ACM Symp. Principles of Programming Languages*, pages 269–282. ACM Press, 1979.

[18] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. Declarative Modeling of the Operational Behavior of Logic Languages. *Theoretical Computer Science*, 69(3):289–318, 1989.

[19] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. A Model-Theoretic Reconstruction of the Operational Semantics of Logic Programs. *Information and Computation*, 102(1):86–113, 1993.

[20] T. Frühwirth, E. Shapiro, M. Vardi, and E. Yardeni. Logic programs as types for logic programs. In *Proceedings of Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 300–309. IEEE Computer Society Press, July 1991.

[21] J. Gallagher and D. A. de Waal. Fast and Precise Regular Approximation of Logic Programs. In Pascal Van Hentenryck, editor, *Proceedings of the Eleventh International Conference on Logic Programming*, pages 599–613. The MIT Press, 1994.

[22] R. Giacobazzi, S. Debray, and G. Levi. Generalized Semantics and Abstract Interpretation for Constraint Logic Programs. *Journal of Logic Programming*, 25(3):191–248, 1995.

[23] M. V. Hermenegildo, R. Warren, and S. K. Debray. Global flow analysis as a practical compilation tool. *Journal of Logic Programming*, 13(4):349–366, August 1992.

[24] M.V. Hermenegildo and K.J. Greene. &-Prolog and its Performance: Exploiting Independent And-Parallelism. In David H. D. Warren and Peter Szeredi, editors, *Seventh International Conference on Logic Programming*, pages 253–268. The MIT Press, June 1990.

[25] D. Jacobs and A. Langen. Static Analysis of Logic Programs for Independent and Parallelism. *Journal of Logic Programming*, 13(2/3):291–314, 1992.

[26] D. Kapur and P. Narendran. Complexity of unification problems with associative-commutative operators. *Journal of Automated Reasoning*, 9(2):261–288, October 1992.

[27] V. Lagoon. Logic program analysis using set logic programs. Master's thesis, Ben-Gurion University of the Negev (Israel), 1998.

[28] A. Langen. *Static analysis for independent And-parallelism in logic programs.* PhD thesis, Univ. of Southern California, 1990.

[29] P. Lincoln and J. Christian. Adventures in associative-commutative unification. *Journal of Symbolic Computation*, 8:217–240, 1989. Also appears in *Unification*, edited by Claude Kirchner (Academic, 1990), pages 393–416.

[30] J. W. Lloyd. *Foundations of Logic Programming.* Springer Verlag, 1987. Second edition.

[31] K. Muthukumar and M. Hermenegildo. Combined Determination of Sharing and Freeness of Program Variables Through Abstract Interpretation. In Koichi Furukawa, editor, *Proceedings of the Eighth International Conference on Logic Programming*, pages 49–63. The MIT Press, 1991.

[32] K. Muthukumar and M. Hermenegildo. Compile-time Derivation of Variable Dependency Using Abstract Interpretation. *Journal of Logic Programming*, 13(2/3):315–347, July 1992.

[33] D. A. Plaisted. The occur-check problem in Prolog. In *International Symposium on Logic Programming*, pages 272–280. Atlantic City, IEEE Computer Society, February 1984.

[34] C. Ramakrishnan, I. Ramakrishnan, and R. Sekar. A symbolic constraint solving framework for analysis of logic programs. In *Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, pages 12–23. ACM Press, 1995.

[35] H. Søndergaard. An Application of Abstract Interpretation of Logic Programs: Occur Check Reduction. In *European Symposium on Programming*, volume 213 of *LNCS*, pages 327–338. Springer Verlag, 1986.