Precise Pair-Sharing Analysis of Logic Programs

Vitaly Lagoon and Peter J. Stuckey
Department of Computer Science and Software Engineering
University of Melbourne
3010, Australia
{lagoon,pjs}@cs.mu.oz.au

June 18, 2002

Abstract

The paper presents a novel approach to pair-sharing analysis of logic programs. The pair-sharing domain ASub of Søndergaard is known to be more efficient than the set-sharing domain Sharing of Jacobs and Langen and gains accuracy because of linearity tracking. However, it is less accurate because of weaker groundness information, and the fact that it loses track of where new groundness eliminates sharing. In this paper we present a new domain which inherits the advantages of both ASub and Sharing and is uniformly more accurate in terms of pair-sharing than each of the two domains.

The proposed domain expresses pair-sharing in terms of existence of *traversable paths* in *relation graphs* derived from program constraints. Each edge of a relation graph has an attached *groundness formula* which defines under what groundness conditions it still causes sharing. This allows the domain to maintain information about when groundness can eliminate sharing. Relation graphs are augmented by separate groundness information (usually Def or Pos). The groundness analysis and groundness formulae can be represented using efficient ROBDD methods.

1 Introduction

Two or more variables in a logic program are said to *share* 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 ground term in every execution of the program. A variable is said to be *linear* if it is bound to a linear term (a term where each variable occurs at most once) in every execution of the program. Sharing, groundness and linearity information 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 [15, 16] or to determine run-time goal independence which can be used to eliminate costly run-time checks in and-parallel execution of logic programs [14, 11, 10]. 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 sharing and groundness information, the availability of linearity information is also useful for improving the precision of sharing analyses.

Abstract descriptions in the Sharing [11] domain carry a great deal of information. They capture not only the possibility of some variables to share but also the ways these variables are shared. For instance, the sharing set XYZ denotes not only the fact that X, Y and Z can share with each other but also that they can share *as a triple*, i.e., by pointing to the same non-ground structure. This extra information is irrelevant for many practical applications of sharing analysis. Normally, the most useful kind of information is whether two variables *can possibly share*, or whether they are *definitely independent*.

Acknowledging the complexity and redundancy associated with Sharing various schemes have been proposed for capturing only *possible pair-sharing* or, equivalently *definite pairwise independence*.

Bagnara *et al.* propose in [3] a domain for pair-sharing analysis derived from Sharing. The abstract descriptions in the proposed domain are considered modulo pair-sharing equivalence. The computational advantage of the domain over Sharing comes from the fact that the costly star-union operation used in the abstract unification in Sharing can be replaced in case of pair-sharing by a cheaper binary union. The authors show that their domain is equally precise as Sharing for pair-sharing analysis. Though potentially more efficient, the proposed domain still suffers from the same problems as Sharing. Firstly, the expressive power of the domain for groundness and groundness dependencies is the same as in Sharing, i.e. equivalent to Def. Secondly, despite the $O(n^2)$ limit for the number of possible sharing pairs for n variables, the domain still admits exponential representations. This is again because of poor handling of groundness in Sharing, which is stored explicitly in sets of models of the corresponding Def formula.

Consider the constraint $X = f(U, V, \_) \wedge X = f(W, Z, \_) \wedge W = Z$, the Sharing description generated by analysis is (writing the sharing sets as sequences):

$$\{X, XU, XV, XUV, XWZ, XUWZ, XVWZ, XUVWZ\}$$

Note that if X is subsequently grounded, we remove all sets containing X, and hence all sets containing both U and V, thus we eliminate sharing between them. If we ground W we do not remove the sharing of U and V because of the set XUV.

The original pair sharing analysis domain ASub of Søndergaard [16] has a much more efficient $O(n^2)$ representation and also maintains linearity information which substantially improves the accuracy of the pair-sharing information obtained. Its weaknesses are: its simple handling of groundness, and its inability to eliminate sharing using new groundness. It is easy to fix the first problem by replacing the weak groundness by a separate Def

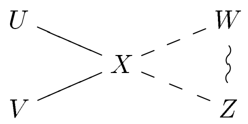


Figure 1: Relation graph for $X = f(U, V, \_),\ X = f(W, Z, \_),\ W = Z$.

or Pos groundness analysis. But the second problem is implicit in the representation.

Consider the constraint $X = f(U, V, \_) \wedge X = f(W, Z, \_) \wedge W = Z$, the pair-sharing component of the ASub description resulting from analysis is

$$\{\ XX, XU, XV, XW, XZ, UU, UV, UW, UZ, VV, VW, VZ, WZ\ \}$$

When we ground X or W we remove all pairs containing them, but the sharing between U and V and their possible non-linearity remains. Note however that we know that W and Z are linear unlike the case for Sharing.

In our approach we avoid this by representing pair sharing not as sets of pairs, but as the paths in a graph. The graph maintains the information about how a pair of variable are able to share. Later groundness information may eliminate variables on the path between two variables, and hence eliminate their sharing.

Essentially a relation graph is a simple abstraction of the concrete constraints where each primitive constraint is mapped to one or more edges in the graph, which represent the direct sharing that the primitive constraint engenders. Paths in the graph represent the sharing resulting from the interaction of primitive constraints. In order to avoid too many paths, the edges resulting from a single primitive constraint are marked as incompatible so that we cannot traverse them in succession to create sharing. A traversable path in the graph is one where each edge is compatible to the previous edge in the path.

The relation graph for $X = f(U, V, \_) \wedge X = f(W, Z, \_) \wedge W = Z$ is shown in Figure 1 using same line styles to mark the pairs of incompatible edges. An example of a traversable path is $\langle U, X, W, Z, X, V\rangle$, which illustrates that U and V share. Note that this path is not simple since it visits X twice, but there is no simple traversable path from U to V. Note that if we ground X or W this path disappears and U and V no longer share.

The relation graph also encodes linearity information in the same way as ASub. A variable v is possibly non-linear if there is a traversable path from v to v. For example U is possibly non-linear in Figure 1 because of the traversable path $\langle U, X, W, Z, X, U\rangle$ while W is linear since there is no traversable path beginning and ending with W. Again the non-linearity of U disappears if we ground X or W.

Note that even compared to the reduced product of Sharing and ASub or the weaker Sharing+Linearity, the relation graph approach is more accurate. For example, in the reduced product the sharing between U and V does not disappear on grounding W.

The rest of the paper is structured as follows. In the next section we give preliminary definitions of sharing concepts. In Section 3 we introduce the relation graph description, and in Section 4 we define the operations to build an abstract domain from these descriptions. In Section 5 we prove it is uniformly more accurate than Sharing. In Section 6 we show how the domain can be implemented efficiently, and give results from our prototype implementation. Finally in Section 7 we conclude. Proofs can be found in the Appendix.

2 Preliminaries

In the following we assume familiarity with the standard definitions and notation for logic programs [12, 1] and abstract interpretation [7]. A reasonable level of familiarity with the Sharing domain [11] is helpful when reading Section 5.

We assume the rules of logic programs are in *canonical form*. That is, rule heads and calls are normalized such that parameters are distinct variables, unifications are broken into sequences of *primitive unification constraints* and nested terms are flattened by introducing intermediate variables. Primitive unification constraints are elements of the set $\mathcal{C}^1$ and can be either in the form $x = y$, where both x and y are variables (from some set $\mathcal{V}$), or in the form $x = f(y_1, \ldots, y_n)$, where $f/n \in \Sigma$, $n \geq 0$ and $x, y_1, \ldots, y_n$ are distinct variables. The set of conjunctions of primitive constraints is denoted by $\mathcal{C}$. A body of a rule in canonical form is a conjunction of elementary unification constraints and procedure calls in the form $p(x_1, \ldots, x_n)$, where $p/n \in \Pi$ is a predicate symbol and parameters $x_1, \ldots, x_n$ are distinct variables.

Set-Sharing: The domain Sharing due to Jacobs and Langen [11] is the best-known domain for set-sharing analysis. Analyses based on Sharing approximate the sets of variables in the domain of interest which can share under the corresponding concrete substitution, or equivalently, a program constraint.

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 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 interest D which are mapped to terms containing v. Following [11], we define the notion of occurrence as:

$$\begin{aligned} &\text{occs} : Sub \times \mathcal{V} \to \wp(\mathcal{V}) \\ &\text{occs}(\theta, v) = \{\, x \in D \mid v \in \text{vars}(x\theta) \,\}. \end{aligned}$$

If $S \subseteq D$ occurs in θ through some variable v then we say that S is a set of variables which *share* under θ. Sets of variables which share under θ form a *set-sharing description*. Set-sharing descriptions are members of the Sharing domain which is a domain of sets of sets of variables ordered by set inclusion: $\mathsf{Sharing} = \langle \wp(\wp(\mathcal{V})), \subseteq \rangle$. Function $\text{Sh}(\theta)$ computes a sharing description of θ:

$$\begin{aligned} &\text{Sh} : Sub \to \mathsf{Sharing} \\ &\text{Sh}(\theta) = \{\, \text{occs}(\theta, v) \mid v \in \mathcal{V} \,\}. \end{aligned}$$

Observe that a variable $x \in D$ is absent from every set in $\text{Sh}(\theta)$ if and only if θ maps x to a ground term.

For each $\kappa \in \mathsf{Sharing}$ and each $T \in \wp(\mathcal{V})$, the operation extracting the relevant component of κ with respect to T is defined as:

$$\mathrm{rel}(\kappa, T) = \{\, S \in \kappa \mid S \cap T \neq \emptyset \,\}$$

The *closure under union* operator (*star-union*) is defined as:

$$(\cdot)^* : \mathsf{Sharing} \to \mathsf{Sharing}$$
$$\kappa^* = (\mathsf{FIX})\,\{\, S_1 \cup S_2 \mid S_1, S_2 \in \kappa \,\}$$

The *pairwise union* function is defined as:

$$\uplus : \mathsf{Sharing} \times \mathsf{Sharing} \to \mathsf{Sharing}$$
$$\kappa_1 \uplus \kappa_2 = \{\, S_1 \cup S_2 \mid S_1 \in \kappa_1,\ S_2 \in \kappa_2 \,\}$$

The function $\mathrm{amgu}_{\mathrm{Sh}}$ captures the effect of a primitive constraint $x = t$ on an element of the $\mathsf{Sharing}$ domain. Let x be a variable and t a term such that $x \notin \mathrm{vars}(t)$. Let $\alpha = \mathrm{rel}(\kappa, \{x\})$, $\beta = \mathrm{rel}(\kappa, \mathrm{vars}(t))$. Then:

$$\mathrm{amgu}_{\mathrm{Sh}}(\kappa, x, t) = \kappa \setminus (\alpha \cup \beta) \ \cup \ (\alpha \uplus \beta)^*$$

Pair-Sharing: Pair-sharing is the property often referred to as the "end-user image" of sharing analysis. Pair-sharing captures only the most interesting bit of sharing information, namely whether two variables can possibly share, or equivalently whether they are definitely independent. The pair-sharing of a substitution θ is defined as:

$$\mathrm{Psh} : \mathit{Sub} \to \wp(\mathcal{V} \times \mathcal{V})$$
$$\mathrm{Psh}(\theta) = \left\{\ \langle x, y \rangle \ \middle|\ \begin{array}{l} x, y \in D,\ x \neq y, \\ \mathrm{vars}(x\theta) \cap \mathrm{vars}(y\theta) \neq \emptyset \end{array} \right\}. \tag{1}$$

The definition of Equation (1) computes all (symmetric) pairs of variables which are aliased by the application of θ. Alternatively, a pair-sharing description of a substitution θ can be extracted from its set-sharing description $\kappa = \mathrm{Sh}(\theta)$:

$$\mathrm{Psh} : \mathsf{Sharing} \to \wp(\mathcal{V} \times \mathcal{V})$$
$$\mathrm{Psh}(\kappa) = \left\{\ \langle x, y \rangle \ \middle|\ \begin{array}{l} x, y \in D, x \neq y, \\ \exists s \in \kappa : \{x, y\} \subseteq s \end{array} \right\} \tag{2}$$

Bagnara *et al.* [3] propose a domain for pair-sharing analysis SS^ρ derived from $\mathsf{Sharing}$. The descriptions in SS^ρ are sets of sets of variables considered modulo the pair-sharing equivalence. This equivalence is based on the canonical form of descriptions produced by the following closure operator:

$$\rho(\kappa) = \kappa \cup \{\, s \mid \forall x, y \in s\ \exists t \in \kappa :\ \{x, y\} \subseteq s \Rightarrow \{x, y\} \subseteq t \,\}$$

This operator completes κ with all sets which can be added without changing the pair-sharing information expressed by κ. Two descriptions are equivalent if the corresponding ρ-closures are identical. The descriptions are ordered with respect to the set-inclusion of their ρ-closures, i.e. $\kappa_1 \preceq_{SS^\rho} \kappa_2 \Leftrightarrow \rho(\kappa_1) \subseteq \rho(\kappa_2)$. The addition of a new constraint $x = t$ to a description κ in SS^ρ does not require the star-closure. Instead, assuming that α and β are the same as in the definition of $\mathrm{amgu}_{\mathrm{Sh}}$, the corresponding amgu_{SS^ρ} is computed as:

$$\mathrm{amgu}_{SS^\rho}(\kappa, x, t) = \kappa \setminus (\alpha \cup \beta) \ \cup \ (\alpha \uplus \beta) \uplus (\alpha \uplus \beta).$$

The authors prove that although the domain SS^ρ is a proper abstraction of Sharing and the unifications in SS^ρ are performed at a lower cost, the two domains are equally precise in capturing pair-sharing.

Linearity: A term t is called *linear* if each variable of $\mathrm{vars}(t)$ occurs in t only once. A variable x in the domain of a substitution θ is *linear* if θ maps x to a linear term.

The pair-sharing sets resulting from Equations (1) and (2) do not consider the reflexive pairs in the form $\langle x, x\rangle$. Since we are interested in the pair-sharing of *different variables*, the $\langle x, x\rangle$ pairs apparently make no sense for pair-sharing analysis. However, considering such pairs provides an elegant way of capturing linearity together with pair-sharing in one set of pairs of variables. The following function $\mathrm{Psh}_{\mathcal{L}}$ computes both the pair-sharing and the linearity of a substitution θ:

$$\begin{aligned}&\mathrm{Psh}_{\mathcal{L}} : Sub \to \wp(\mathcal{V} \times \mathcal{V})\\ &\mathrm{Psh}_{\mathcal{L}}(\theta) = \mathrm{Psh}(\theta) \cup \{\, \langle x, x\rangle \mid x \in D, x\theta \text{ is non-linear} \,\}\,.\end{aligned} \tag{3}$$

The idea of handling both pair-sharing and linearity as a single set of pairs is due to Søndergaard [16].

Note that the information of variables linearity cannot be extracted from a set-sharing description of a substitution. The original Sharing domain discards all information about linearity of variables of interest. However, the analysis presented in this work is concerned with *definite linearity* as opposed to *possible non-linearity*. Thus, it is always safe to assume that all variables are non-linear in the pair-sharing approximations extracted from set-sharing descriptions. The following function definition achieves this by removing the $x \neq y$ requirement from the definition of Equation 2:

$$\begin{aligned}&\mathrm{Psh}_{\mathcal{L}} : \mathsf{Sharing} \to \wp(\mathcal{V} \times \mathcal{V})\\ &\mathrm{Psh}_{\mathcal{L}}(\kappa) = \{\, \langle x, y\rangle \mid x, y \in D,\ \exists s \in \kappa : \{x, y\} \subseteq s \,\}\end{aligned} \tag{4}$$

Note that for any κ the result of $\mathrm{Psh}_{\mathcal{L}}(\kappa)$ includes all pairs in the form $\langle x, x\rangle$ where $x \in D$ and x is not ground.

3 Relation Graphs

The formalism of *relation graphs* provides a straightforward abstraction for program constraints. Informally, a relation graph is a graph with its nodes corresponding to the constraint variables and its edges corresponding to pairs of variables linked by primitive constraints.

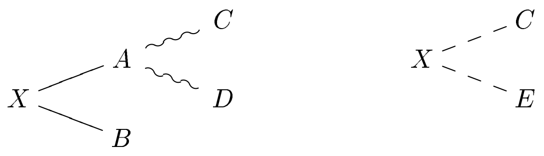


Figure 2: Relation graphs for $X = f(A, B) \wedge A = g(C, D)$ and $X = f(E, C)$.

For a constraint in the form $C : x_1 = t_1 \wedge \cdots \wedge x_n = t_n$ we define a *relation graph* $G = (V, E, I)$ with a set of vertices $V \subseteq \mathcal{V}$, a set of labeled undirected edges $E \subseteq \mathcal{V} \times \mathcal{V} \times \mathbb{N}$, where each label is a unique number, and a set of pairs of incompatible edges $I \subseteq E \times E$. Unique labels are used in order to distinguish between multiple edges connecting the same pair of vertices. The set of relation graphs is denoted by $\mathcal{G}$. For a primitive constraint in the form $x = t$ the relation graph $G = (V, E, I)$ is provided by the following function $\sigma^1 : \mathcal{C}^1 \to \mathcal{G}$:

$$
\begin{aligned}
&\sigma^1([x = t]) = (V, E, I), \text{ where} \\
&\qquad V = \{x\} \cup \mathrm{vars}(t), \\
&\qquad E = \{\ (x, y, k)\ \mid y \in \mathrm{vars}(t),\ k \text{ is a unique id}\,\}, \\
&\qquad I = E \times E
\end{aligned}
\tag{5}
$$

For a conjunction of primitive constraints the relation graph is provided by the following recursive function $\sigma : \mathcal{C} \to \mathcal{G}$ based on σ^1:

$$
\begin{aligned}
\sigma(\mathit{true}) &= (\emptyset, \emptyset, \emptyset) \\
\sigma([x = t] \wedge C) &= (V \cup V_C, E \cup E_C, I \cup I_C), \text{ where} \\
&\qquad (V_C, E_C, I_C) = \sigma(C), \\
&\qquad (V, E, I) = \sigma^1([x = t]).
\end{aligned}
$$

Note that all edges of a relation graph corresponding to the same primitive constraint are defined as incompatible. Note also that each edge is incompatible to itself. The notion of edge incompatibility is central in restricting the paths in relation graphs to only those relevant for pair-sharing and linearity analysis. We call such paths *traversable*.

We say that a non-empty path $\langle v_0, \ldots, v_k \rangle$ from $u = v_0$ to $v = v_k$ in a relation graph G is traversable if any two successive edges contained in this path are compatible, i.e. if we have $\forall i \in \{0, \ldots, n-1\} :\ \langle e_i, e_{i+1} \rangle \notin I$. We use the notation $u \looparrowright_G v$ to denote the existence of a traversable path between u and v in the graph G. By abuse of notation we write $u \looparrowright v$ when the graph is clear from the context. Sometimes we also use the notation $u {\looparrowright}^{\rho} v$ which is read as "a traversable path ρ connecting u with v". Note that in contrast to the standard connectivity relation on graph "$\looparrowright$" is neither reflexive nor transitive. However "$\looparrowright$" is symmetric meaning that if x is linked to y by an traversable path then y is linked to x.

Example 1 *Relation graphs $\sigma(C_1)$ and $\sigma(C_2)$ where C_1 is $X = f(A,B) \wedge A = g(C,D)$ and C_2 is $X = f(E,C)$ are given in Figure 2. Example traversable paths in $\sigma(C_1)$ are $\langle B, X\rangle$ and $\langle X, A, C\rangle$. Note however that there is no traversable path from B to C.*

The fundamental relation between traversable paths of a graph and the pair-sharing and linearity of the corresponding constraint is given by the following theorem.

Theorem 1 *For a constraint C we have:*

$$\mathrm{Psh}_{\mathcal{L}}(C) \subseteq \left\{ \langle x, y\rangle \;\middle|\; x \looparrowright_{\sigma(C)} y \right\}.$$

Note that Theorem 1 does not take into account the fact that a solution of the constraint C may ground, i.e., bind to fully instantiated terms some variables of c. Moreover, the equivalence of the pair-sharing characteristics of two descriptions does not extend to their "more ground" instances. Consider the following graphs.

$$A \longrightarrow B \rightsquigarrow C \qquad\qquad A \longrightarrow C \rightsquigarrow B$$

Although all pairs share for both graphs, grounding B in the first graph eliminates $\langle A, C\rangle$ sharing, while it does not eliminate it from the second graph.

In order to reason about pair-sharing with respect to groundness of variables we introduce the concept of *groundness model.* A set S is a groundness model of a constraint C if there exists a solution of C which grounds precisely the set S of variables.

It is clear that variables that become ground cannot share with other variables. The intuition also suggests that pair-sharing cannot be propagated through the ground variables. Thus, for a particular groundness model S of a constraint C we can obtain tighter upper bounds for pair-sharing than those of Theorem 1 if we exclude the traversable paths that pass through the nodes of S. We consider separately the pair-sharing of C under each of its groundness models. The pair-sharing in each case can be viewed as a pair-sharing imposed by the combined substitution $\theta \circ \psi$, where θ is a most general solution of C and ψ is a substitution mapping variables of S to ground terms, i.e., $\mathrm{dom}(\psi) = S$ and $\mathrm{vars}(\mathrm{range}(\psi)) = \emptyset$. We use the denotation $\mathrm{Psh}_{\mathcal{L}}(C, S)$ for the pair-sharing imposed by a most general solution of C which grounds S. Let $G \setminus S$ be the relation graph defined by eliminating from G all nodes in S and all edges attached to those nodes. The following theorem establishes the correctness of the intuition that partially ground traversable paths do not represent pair sharing.

Theorem 2 *For a constraint C and a groundness model S of C we have:*

$$\mathrm{Psh}_{\mathcal{L}}(C, S) \subseteq \left\{ \langle x, y\rangle \;\middle|\; x \looparrowright_{\sigma(C)\setminus S} y \right\}.$$

Naturally, reasoning about pair-sharing of variables with respect to their groundness has to be based on some form of groundness approximation. We assume a function $\phi : \wp(\mathcal{V}) \to \{\mathsf{True}, \mathsf{False}\}$ which characterizes the groundness models of C. Rather than choose a particular domain for groundness, we simply assume that the approximation of groundness is correct in the usual sense of abstract interpretation. Later in this work we show how the known techniques of groundness analysis such as the ones based on the domains of definite (Def) and positive (Pos) Boolean formulas [13, 2] can be linked to our framework.

In Section 4 we compare abstract objects based on their amount of pair-sharing. It is normally our interest to verify if two abstract descriptions define the same pair-sharing with respect to the same groundness model. So, we define the characteristic pair-sharing of *triplets* $\langle x, y, s\rangle$, where x and y are variables which can share under the groundness model s for a graph G and groundness function ϕ as:

$$\chi_{\mathrm{Psh}_{\mathcal{L}}(G,\phi)}(x, y, s) = \phi(s) \wedge x \looparrowright_{G \setminus s} y \ . \tag{6}$$

Different descriptions may involve different sets of "non-interesting" variables. Thus, the next step necessary for allowing comparison of pair-sharing of different descriptions is to restrict the characteristics of pair-sharing and groundness to a domain D of variables of interest as follows:

$$\chi_{\mathrm{gr}(\phi,D)}(d) = \exists e \subseteq (\mathcal{V} \setminus D) : \ \phi(d \cup e), \tag{7}$$

$$\chi_{\mathrm{Psh}_{\mathcal{L}}(G,\phi,D)}(x, y, d) = \exists e \subseteq (\mathcal{V} \setminus D) : \chi_{\mathrm{Psh}_{\mathcal{L}}(G,\phi)}(x, y, d \cup e), \tag{8}$$

The above formulas are used as building blocks for reasoning about the order and equivalence of abstract objects developed in the next section.

4 The Abstract Domain

In the previous section we showed how we can capture the pair-sharing of a constraint C by the corresponding relation graph $\sigma(C)$ together with the groundness description ϕ. In this section we define a (class of) abstract domain(s) Ω_{gr} of *abstract constraints* in the form $\langle G, \phi\rangle$ where $G = (V, E, I)$ is a relation graph and ϕ is a description from abstract groundness domain gr, represented as a characteristic function $\phi : \wp(\mathcal{V}) \to \{\mathsf{True}, \mathsf{False}\}$. We define the abstraction function, and the abstract operations of conjunction, disjunction, projection and extension for Ω.

The abstraction function α of Ω_{gr} is simply defined as $\alpha(C) = \langle \sigma(C), \alpha_{\mathsf{gr}} C\rangle$, where α_{gr} is the groundness domain abstraction function. For now we treat α_{gr} as a generic component of our construction assuming only its correctness. On practice however, α_{gr} is fixed by the domain of groundness analysis which is normally either Def or Pos [2].

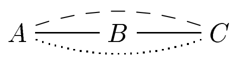
Elements of Ω_{gr} are ordered with respect to the amount of pair-sharing and groundness they define for variables of the domain of interest D. The abstract order relation denoted by $\preceq_D$ is defined as follows:

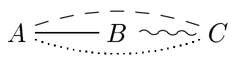
$$
\begin{aligned}
&\langle G_1, \phi_1 \rangle \ \preceq_D \ \langle G_1, \phi_2 \rangle \ \equiv \\
&\quad \forall d \subseteq D : (\chi_{\mathrm{gr}(\phi_1, D)}(d) \Leftarrow \chi_{\mathrm{gr}(\phi_2, D)}(d)) \ \wedge \\
&\quad \forall x, y \in D : \forall d \subseteq D : (\chi_{\mathrm{Psh}_{\mathcal{L}}(G_1, \phi_1, D)}(x, y, d) \Rightarrow \chi_{\mathrm{Psh}_{\mathcal{L}}(G_2, \phi_2, D)}(x, y, d)).
\end{aligned}
$$

Namely, for two abstract constraints $\mu_1, \mu_2 \in \Omega_{\mathsf{gr}}$ the relation $\mu_1 \preceq_D \mu_2$ means that μ_1 defines more groundness and less pair-sharing than μ_2. The equivalence of two abstract constraints μ_1 and μ_2, denoted by $\mu_1 \approx_D \mu_2$ is naturally defined as:

$$
\mu_1 \approx_D \mu_2 \equiv \mu_1 \preceq_D \mu_2 \ \wedge \ \mu_2 \preceq_D \mu_1. \tag{9}
$$

The following example shows that the equivalence of two pair-sharing descriptions as defined by Equation (9) does not imply the isomorphism of the corresponding graphs.





Although the graphs are non-isomorphic, they define the same pair-sharing for the same (pairs of) variables of interest for any groundness model.

Clearly two different graphs can represent the same pair-sharing. Moreover, since the graphs can contain "non-interesting" nodes (existentially quantified variables), there are infinitely many graphs corresponding to the same pair-sharing for the same domain of interest. However, this infinity does not lead to the infinity of the domain we are constructing. From our point of view the two graphs above are indistinguishable because they both represent the same equivalence class of Ω_{gr} as induced by the equivalence relation of Equation (9). The following theorem establishes that the number of such equivalence classes is finite.

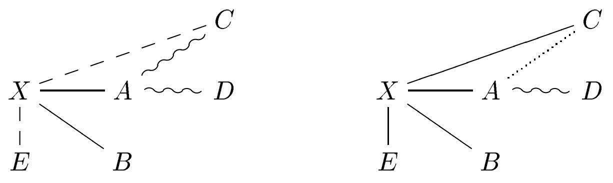
Theorem 3 *The set $[\Omega_{\mathsf{gr}}]_{\approx}$ is finite for a finite domain of interest D.*

4.1 Abstract Conjunction

The operation of *abstract conjunction*, $\curlywedge$, approximates the pair-sharing imposed by the concrete conjunction of unification constraints performed during the program run. Because the definition of relation graphs is effectively conjunctive this is simple to define in our domain.

$$
\begin{aligned}
&\langle (V_1, E_1, I_1), \phi_1 \rangle \curlywedge \langle (V_2, E_2, I_2), \phi_2 \rangle \ = \\
&\qquad\qquad \langle (V_1 \cup V_2, E_1 \cup E_2, I_1 \cup I_2), \ (\phi_1 \curlywedge_{\mathsf{gr}} \phi_2) \rangle
\end{aligned}
$$

The conjunction is just the union of the two relation graphs, together with the abstract conjunction of the groundness information. We assume $\curlywedge_{\mathsf{gr}}$ is a correct abstract conjunction for groundness domain gr. Note that every edge from the first relation graph is compatible with every edge from the second relation graph.



(a) (b)

Figure 3: Relation graphs for (a) $\sigma(C_1) \curlywedge \sigma(C_2)$ and (b) $\sigma(C_1) \curlyvee \sigma(C_2)$.

Theorem 4 *The abstract conjunction operation $\curlywedge$ is correct.*

Example 2 *The abstract conjunction of the two graphs $\sigma(C_1)$ and $\sigma(C_2)$ of Example 1 is shown in Figure 3(a). Note that in the conjunction X is non-linear because of the path $\langle X, C, A, X\rangle$, but E is not non-linear.*

4.2 Abstract Disjunction

The operation of *abstract disjunction*, $\curlyvee$, of two abstract constraints approximates the result of disjunction of the corresponding concrete constraints. It is defined as follows:

$$\langle (V_1, E_1), \phi_1\rangle \curlyvee \langle (V_2, E_2, I_2), \phi_2\rangle = \\ \langle (V_1 \cup V_2, E_1 \cup E_2, I_1 \cup I_2 \cup E_1 \times E_2),\ (\phi_1 \curlyvee_{\mathsf{gr}} \phi_2)\rangle$$

The disjunction is just the union of the two relation graphs *except* that every edge in the first graph is incompatible with every edge in the second graph (using $E_1 \times E_2$). This construction guarantees that a traversable path can never cross from G_1 to G_2 (or vice versa). This guarantees that the set of traversable paths in the resulting graph G is exactly the union of all the paths of the original graphs. Again we assume $\curlyvee_{\mathsf{gr}}$ is a correct abstract disjunction for groundness domain gr.

Theorem 5 *The abstract disjunction operation $\curlyvee$ is correct and optimal.*

Example 3 *The abstract disjunction of the two graphs $\sigma(C_1)$ and $\sigma(C_2)$ of Example 1 is shown in Figure 3(b). The dotted arc $\langle A, C\rangle$ is incompatible with arc $\langle X, C\rangle$ and $\langle A, D\rangle$ but not $\langle X, A\rangle$. Note how no new traversable paths are introduced.*

4.3 Abstract Projection and Extension

The operation of *abstract projection* of an abstract constraint μ onto a (smaller) set of variables of interest D' is trivial, since we simply need to change the set of variables of interest D to be D'. For safety we rename the variables in $D \setminus D'$ away from any variables used elsewhere.

The operation of *abstract extension* of an abstract constraint μ onto a (larger) set of variables of interest D' is also trivial, since again we simply need to change the set of variables of interest to D'.

5 Comparison with ASub and Sharing

It is straightforward to show that the domain Ω_{gr} is uniformly more accurate than ASub (assuming the groundness gr is at least as strong as that of ASub). Essentially the result holds because whenever there is a traversable path $x \looparrowright_\mu y$ then we can show the pair $\langle x, y\rangle$ exists in the corresponding ASub description. If Con is the simple groundness domain of [16], then we have.

Theorem 6 *The pair-sharing analysis in Ω_{Con} is at least as precise as the pair-sharing analysis based on* ASub*.*

The comparison with Sharing is much less straightforward. In order to show that Ω is at least as precise as Sharing we require an adequate approximation of groundness. For a fair comparison we show that Ω_{Def} is at least as precise as Sharing, since Def is the groundness captured by Sharing.

Instead of comparing Ω_{Def} with Sharing directly we base our argument on the comparison of Ω_{Def} with SS^ρ described in Section 2. Bagnara *et al.* [3] demonstrate that the domain SS^ρ and Sharing are equivalently precise for pair-sharing and groundness analysis. Thus, establishing that Ω_{Def} is more precise than SS^ρ also provides the result for Sharing.

First we demonstrate that Ω_{Def} is more expressive than SS^ρ by proving that the latter is a proper abstraction of the former. Assume a description $\mu = \langle G, \phi\rangle$ in Ω_{Def} where ϕ is a Def-formula. Then the corresponding description in SS^ρ is produced by the following function of abstraction $\alpha_\Omega : \Omega_{\mathsf{Def}} \to SS^\rho$:

$$\alpha_\Omega(\langle G, \phi\rangle) = \left\{ s \;\middle|\; \begin{array}{l} \exists t \subseteq D :\ s = D \setminus t, \\ \forall x, y \in s,\ x \neq y : \chi_{\mathrm{Psh}_{\mathcal{L}}(G,\phi,D)}(x, y, t) \end{array} \right\} \tag{10}$$

According to Equation (6), t is a model of ϕ. Thus, α_Ω of Equation (10) collects inversions of groundness models which contain only valid sharing pairs. Let us now justify the correctness of this abstraction.

Theorem 7 *A description $\mu = \langle G, \phi\rangle$ in Ω_{Def} and the corresponding abstraction $\alpha_\Omega(\mu)$ in SS^ρ express the same groundness and the same pair-sharing for the same domain of interest D.*

The following lemmas establish that Ω_{def} is more accurate than Sharing for conjunction and disjunction.

Lemma 8 *For two descriptions $\mu \in \Omega_{\mathsf{Def}}$ and $\kappa \in SS^\rho$ over the same domain of interest D we have:*

$$\alpha_\Omega(\mu) \preceq_{SS^\rho} \kappa \ \Rightarrow\ \alpha_\Omega(\mu \curlywedge \alpha([x = t])) \preceq_{SS^\rho} \mathrm{amgu}_{SS^\rho}(\kappa, x, t)$$

Lemma 9 *For the descriptions* $\mu_1, \mu_2 \in \Omega_{\mathsf{Def}}$ *and* $\kappa_1, \kappa_2 \in SS^\rho$ *over the same domain of interest* D *we have:*

$$(\alpha_\Omega(\mu_1) \preceq_{SS^\rho} \kappa_1) \wedge (\alpha_\Omega(\mu_2) \preceq_{SS^\rho} \kappa_2) \Rightarrow \alpha_\Omega(\mu_1 \curlyvee \mu_2) \preceq_{SS^\rho} (\kappa_1 \cup \kappa_2)$$

The cases for projection and extension are straightforward. This leads to:

Theorem 10 *The pair-sharing analysis in* Ω_{Def} *is at least as precise as the pair-sharing analysis based on* Sharing.

6 Experimental Results

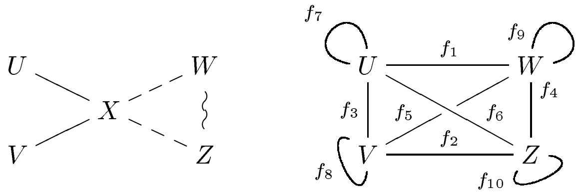
The first impression of the domain Ω_{gr} may be that it is cumbersome and impractical. The equivalence check of Equation (9) required by fixpoint computations seemingly involves enumerating all traversable paths in relation graphs. In addition, the abstract operations seem to keep all intermediate variables in the abstract descriptions, thus leading to huge relation graphs maintained through the computation. We now discuss a method for effectively handling both problems: enumeration of traversable paths and projection of "non-interesting" variables, by taking a transitive closures of relation graphs annotated by Boolean formulas.

6.1 The Transitively Closed Form

The pair-sharing closure of a relation graph $G = (V, E, I)$ is a relation graph $G' = (V, E')$ such that all edges in E' are treated as pairwise-incompatible and each traversable path in G corresponds to an edge in E'. The edges in the set E' are in the form (u, v, ϕ), where ϕ is a Boolean formula encoding the groundness models under which the corresponding pair-sharing can occur. Figure 4 demonstrates the transitive closure of the relation graph from the introduction of this paper (Figure 1). Here we assume that the domain of interest does not include X and thus, X is removed from the closure. The resulting graph together with the annotations (f_1 to f_{10}) is shown on the right side of Figure 4. Note that some edges are annotated by False. This means that the corresponding traversable path is nonexistent. Naturally, in the implementation of the analyzer these edges can be removed.

The transitively closured pair-sharing representations are initially from the results of pair-sharing abstraction of primitive constraints (Equation (5)) by annotating all edges of the graph. An edge connecting two nodes u and v is annotated by $\bar{u} \wedge \bar{v}$, thus, expressing the requirement of non-groundness of the endpoints.

An abstract semantics approximating the pair-sharing of a logic program is evaluated through a finite sequence of abstract conjunctions, abstract disjunctions and projections, given respectively by the operators ABSTR-CONJ ABSTR-DISJ, ABSTR-PROJ. These operators described in Sections 6.2–6.4 all maintain their inputs and outputs as transitively closed pair-sharing descriptions.



f_1	:	$\overline{U} \wedge \overline{W}$	f_6	:	$\overline{U} \wedge \overline{Z}$
f_2	:	$\overline{V} \wedge \overline{Z}$	f_7	:	$\overline{U} \wedge \overline{W} \wedge \overline{Z}$
f_3	:	$\overline{U} \wedge \overline{V} \wedge \overline{W} \wedge \overline{Z}$	f_8	:	$\overline{V} \wedge \overline{W} \wedge \overline{Z}$
f_4	:	$\overline{W} \wedge \overline{Z}$	f_9	:	False
f_5	:	$\overline{V} \wedge \overline{W}$	f_{10}	:	False

Figure 4: Path computation and elimination of X.

The representations based on the transitive closures of relation graphs are canonical. For the same domain of interest two equivalent representations will have equivalent annotations for each edge of the closure graph. Moreover, two equivalent representations would be *identical* if we choose to maintain groundness annotations of the edges as ROBDDs [4].

6.2 Abstract Conjunction

Algorithm 1 together with the supplementary functions of Algorithms 2 and 3 implements the operation of abstract conjunction of two pair-sharing descriptions. The descriptions are given respectively as two pairs $\langle G_1, \xi_1 \rangle$ and $\langle G_2, \xi_2 \rangle$, where G_1 and G_2 are transitively closed relation graphs and ξ_1 and ξ_2 are Pos formulae representing the corresponding groundness approximations. The result of ABSTR-CONJ is a new pair $\langle G, \xi \rangle$ such that G is also transitively closed.

The computation of ABSTR-CONJ computes alternating paths in the graph made up from merging the two descriptions. In order to maintain the notion of edge compatibility throughout the computation we assume that the endpoints of all edges are colored either in black or in white. For an edge e based on the endpoints u and v we denote the corresponding colors by $col[e_u]$ and $col[e_v]$ respectively. For a self-edge e based on a node v the colors of the two endpoints are denoted as $col[e_{v1}]$ and $col[e_{v2}]$ respectively. We assume that two adjacent edges are compatible if and only if their adjacent endpoints have *different* colors. Initially the function ABSTR-CONJ colors all edges of G_1 in white and all edges of G_2 as black, thus, making any pair of edges originating from the same graph incompatible and making any pair of adjacent edges from different graphs compatible.

The computation of the transitive closure of both G_1 and G_2 consists in incrementally adding the edges of G_2 to G_1 using ADD-EDGE function shown in Algorithms 2. This function adds a new edge e to a set of edges E together with all transitive paths which result from the addition of e. There are five

```
Func ABSTR-CONJ(⟨G_1, ξ_1⟩,⟨G_2, ξ_2⟩)
  let G_1 = (V_1, E_1)
  let G_2 = (V_2, E_2)
  ξ ← ξ_1 ∧ ξ_2
  V ← V_1; E ← ∅
  for all e = (u, v, φ) ∈ E_1 do
    col[e_u] ← WHITE
    col[e_v] ← WHITE
    E ← E ∪ (u, v, φ ∧ ξ)
  for all e = (u, v, φ) ∈ E_2 do
    col[e_u] ← BLACK
    col[e_v] ← BLACK
    ADD-EDGE(E, e, ξ)
  G ← JOIN-MULTI-EDGES((V, E))
  return ⟨G, ξ⟩
```

Algorithm 1: Conjunction of pair-sharing descriptions.

cases (encoded by the corresponding "for all" loops) when such new paths may appear. Note that each edge in the conjunction has the new groundness ξ conjoined to its groundness description.

Note that in general the result of ADD-EDGE may contain more than one edge connecting the same a pair of nodes. We use function JOIN-MULTI-EDGES in order to collapse all multiple edges of the graph G into single ones using disjunction to join the groundness parts.

6.3 Abstract Disjunction

The operation of abstract disjunction of two transitively closed graphs G_1 and G_2 and the corresponding Pos descriptions ξ_1 and ξ_2 is given by the function ABSTR-DISJ of Algorithm 4. The operation trivially collects all the paths existing in the two graphs *without adding any new paths* and unites the multi-edges corresponding to the same paths.

6.4 Abstract Projection

Function ABSTR-PROJ of Algorithm 6.4 projects a transitively closed relation graph G on the domain of interest D. That step is accomplished by disregarding all edges with one or two endpoints outside of D. The groundness annotations of the remaining edges are projected on the domain of interest as usual i.e., by quantifying existentially all "non-interesting" variables.

6.5 Complexity

The algorithms of abstract conjunction, disjunction and projection rely on the Boolean operations of conjunction, disjunction, existential quantification and

Func $\textsc{Add-Edge}(E, e, \xi)$
  let $e = (u, v, \phi)$
  $E' \leftarrow E \cup \{(u, v, \phi \wedge \xi)\}$
  for all $e' = (x, u, \phi') \in E'$ **do**
    if $col[e_u] \neq col[e'_u]$ **then**
      $e^\dagger \leftarrow (x, v, \phi_0 \wedge \phi_1)$
      $col[e^\dagger_v] \leftarrow col[e_v]$
      $col[e^\dagger_x] \leftarrow col[e'_x]$
      $E' \leftarrow E' \cup \{e^\dagger\}$
  for all $e' = (v, w, \phi') \in E'$ **do**
    if $col[e_v] \neq col[e'_v]$ **then**
      $e^\dagger \leftarrow (u, w, \phi_0 \wedge \phi_1)$
      $col[e^\dagger_u] \leftarrow col[e_u]$
      $col[e^\dagger_w] \leftarrow col[e'_w]$
      $E' \leftarrow E' \cup \{e^\dagger\}$
  for all $e' = (x, u, \phi') \in E', e'' = (v, w, \phi'') \in E'$ **do**
    if $(col[e'_u] \neq col[e_u]) \wedge (col[e_v] \neq col[e''_v])$ **then**
      $e^\dagger \leftarrow (x, w, \phi \wedge \phi' \wedge \phi'')$
      $col[e^\dagger_x] \leftarrow col[e'_x]$
      $col[e^\dagger_w] \leftarrow col[e''_w]$
      $E' \leftarrow E' \cup \{e^\dagger\}$
  for all $e' = (v, v, \phi') \in E'$ **do**
    if $(col[e_v] \neq col[e'_{v1}]) \wedge (col[e_v] \neq col[e'_{v2}])$ **then**
      $e^\dagger \leftarrow (u, u, \phi \wedge \phi')$
      $col[e^\dagger_{u1}] \leftarrow col[e_u]$
      $col[e^\dagger_{u2}] \leftarrow col[e_u]$
      $E' \leftarrow E' \cup \{e^\dagger\}$
  for all $e' = (u, u, \phi') \in E'$ **do**
    if $(col[e_u] \neq col[e'_{u1}]) \wedge (col[e_v] \neq col[e'_{u2}])$ **then**
      $e^\dagger \leftarrow (v, v, \phi \wedge \phi')$
      $col[e^\dagger_{v1}] \leftarrow col[e_v]$
      $col[e^\dagger_{v2}] \leftarrow col[e_v]$
      $E' \leftarrow E' \cup \{e^\dagger\}$
  return E'

Algorithm 2: Add new edge to the closure

```
Func JOIN-MULTI-EDGES(G)
  let G = (V, E)
  E' ← ∅
  for all e ∈ E do
    let e = (u, v, φ)
    if ∃e' ∈ E' : e' = (u, v, φ') then
      E' ← (E' \ {e'}) ∪ {(u, v, φ ∨ φ')}
    else
      E' ← E' ∪ {e}
  return (V, E')
```

Algorithm 3: Join multi-edges into a single edge

```
Func ABSTR-DISJ(⟨G1, ξ1⟩,⟨G2, ξ2⟩)
  let G1 = (V1, E1)
  let G2 = (V2, E2)
  G ← JOIN-MULTI-EDGES((V1 ∪ V2, E1 ∪ E2))
  return ⟨G, ξ1 ∨ ξ2⟩
```

Algorithm 4: Disjunction of pair-sharing descriptions.

a satisfiability test. Thus, a) the theoretical worst-case complexity of each of these algorithms is exponential and b) in practice the performance of any implementation of Ω will heavily depend on the performance of the corresponding Boolean solver integrated into the analyzer.

In our implementation described in the following section all Boolean combinations and satisfiability tests are performed using ROBDDs [4]. We use the ROBDDs library implemented by Peter Schachte.[1] Although the worst-case complexity of Boolean operations is exponential, thanks to ROBDD data structure and the nature of our analysis, exponential-size ROBDDs almost never occur in practice. This claim is supported by our experimental evaluations discussed in the following section. In all cases our prototype implementation demonstrates adequate performance both in terms of run-time and in terms of memory taken by ROBDDs.

Since the cost of each individual operation in ROBDD is low it actually makes sense to estimate the cost of abstract operations *in the number of calls to Boolean functions*. Let us estimate the complexity of ABSTR-CONJ algorithm. This complexity and the actual performance of the implementation are the major factors influencing the performance of any pair-sharing analyzer based on Ω. Assume that ABSTR-CONJ joins two transitively closed graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ into a new graph $G = (V, E)$. The algorithm makes $|E_2|$ calls to ADD-EDGE. Using graph representation based on adjacency matrix the "for all" loops of ADD-EDGE make at most $O(|V|)$ calls to Boolean functions (conjunction and satisfiability checks). Thus, the total cost of ABSTR-CONJ is $O(|V| \cdot |E_2|)$ Boolean operations which in worst case is equivalent to $O(|V|^3)$

[1] `http://www.cs.mu.OZ.AU/~schachte`

Func ABSTR-PROJ($\langle G, \xi \rangle, D$)
  let $G = (V, E)$
  $\xi' \leftarrow \exists(\text{vars}(\xi) \backslash D) : \xi$
  $V' \leftarrow V \cap D$
  $E' \leftarrow \emptyset$
  for all $e \in E$ **do**
    let $e = (u, v, \phi)$
    if $\{u, v\} \subseteq D$ **then**
      $\phi' \leftarrow \exists(\text{vars}(\phi) \backslash D) : \phi$
      $E' \leftarrow E' \cup \{(u, v, \phi')\}$
  return $\langle (V', E'), \xi' \rangle$

Algorithm 5: Existential quantification for pair-sharing.

Boolean operations, especially taking into account the fact that all graphs are transitively closed.

The cost of both ABSTR-DISJ and of ABSTR-PROJ estimates at $O(|V|^2)$ Boolean operations assuming that we use adjacency matrix for representing the graphs.

6.6 Goal-Independent and Differential Analysis

García de la Banda *et al.* showed [8] that goal-independent analysis is uniformly more accurate than goal-dependent for order-independent and upwards closed domains. It clear from the definition of conjunction in Ω_{Pos} that it is order independent. Hence by Theorem 6.2 of [8], we know that goal-independent analysis and simple differential analysis are equivalent. But even though Ω_{Pos} is not upwards closed, we can extend the result of Theorem 6.3 of [8] holds for domain Ω_{Pos}, that is goal-independent analysis is uniformly more accurate than standard goal-dependent analysis. We assume the definitions of $gi_{\mathcal{D}}$, $sdiff_{\mathcal{D}}$ and $std_{\mathcal{D}}$ of [8].

Theorem 11 *For description domain Ω_{Pos} for all programs P and goals G,*

$$gi_{\Omega_{\mathsf{Pos}}}(P, G) = sdiff_{\Omega_{\mathsf{Pos}}}(P, G) \preceq_{\Omega_{\mathsf{Pos}}} std_{\Omega_{\mathsf{Pos}}}(P, G).$$

6.7 Experiments

In order to validate experimentally the ideas presented in this work a prototype implementation of the pair-sharing analyzer has been developed. The implementation is based on a goal-dependent Prolog meta-interpreter integrated with the algorithm of abstract conjunction (Algorithm 1) implemented in C and the ROBDD library. The Prolog code is developed under GNU Prolog v1.2.1. This implementation is indeed a prototype. It does not contain optimizations normally found in Prolog meta-interpreters, such as those based on call graph analysis. Moreover, we chose to avoid managing foreign (non-Prolog) data from within our Prolog code. So, Boolean formulas are normally kept as Prolog terms and converted to ROBDDs only when simplification or existential quantification is required. The result of the corresponding operation is converted back to a Prolog term. Naturally, it is easier to develop and debug the code following this strategy. However, the price is paid in the code performance since the same Boolean formulae are re-interpreted and converted to ROBDDs and back to Prolog terms many times.

Table 6.7 shows the performance of results of running our prototype analyzer on the PLAI test suite.[2] The computation times has been taken on a 900MHz Pentium III machine with 512Mb of RAM running Linux 2.4. The "Vars" column gives the size of a benchmark measured in a number of variables in all calls and answers discovered during the analysis. The "Time" column gives the time of analysis in seconds. The "BDD" column gives the maximum number of ROBDD nodes allocated during the test run (which includes garbage produced during ROBDD construction). We ran two different versions of the analysis for Ω_{Pos}, the standard goal-dependent analysis (Ω_{Pos}), and a differential analysis ($\Omega^{\mathrm{diff}}_{\mathsf{Pos}}$). Note that the differential analysis is not the simple differential analysis [8], but an improved differential analysis where the downwards closed groundness component is passed into the differential component. See [8] for the definition of a similar differential analysis for ASub.

The accuracy results are also shown in Table 6.7, we give the number of sharing pairs and the number of non-linear variables discovered by the analysis in the calling and answer patterns for Ω_{Pos}. We give the same information for $\Omega^{\mathrm{diff}}_{\mathsf{Pos}}$. Although uniformly more accurate the differential analysis $\Omega^{\mathrm{diff}}_{\mathsf{Pos}}$ only improves the information in one case "witt". But for the difficult benchmarks ("qplan" and "witt") it give substantial execution benefits, because most conjunctions involve smaller operands. For goals with no initial groundness information, we would expect $\Omega^{\mathrm{diff}}_{\mathsf{Pos}}$ to show substantially more improvement in accuracy.

We then give accuracy results for ASub and $\mathsf{Sharing} \otimes \mathsf{ASub}$, omitting entries which do not differ from those of Ω_{Pos}. The analysis for ASub and $\mathsf{Sharing} \otimes \mathsf{ASub}$ was performed using the CIAO-Prolog preprocessor "ciaopp" (v0.7) [9]. In this case the statistics about a programs pair-sharing is extracted from the set-sharing and linearity annotations produced by "ciaopp". All results are identical for these two domains, except "qplan" and "witt" where "ciaopp" runs out of memory (denoted by ∞) on $\mathsf{Sharing} \otimes \mathsf{ASub}$. Note that ASub in "ciaopp" betters Ω_{Pos} for "witt" because the analysis framework uses pattern matching among calls and heads to improve accuracy.

Next we show the same statistics for $\mathsf{ASub} \otimes \mathsf{Pos}$. We mimicked this domain using our implementation of Ω_{Pos}. This was achieved by discarding non-groundness annotations of edges in the transitive closure (see 6.1). It is interesting to note that for the considered set of benchmarks the addition of a stronger Pos groundness contributes more to the precision of the analysis than the addition of a stronger set-sharing. The analysis in $\mathsf{ASub} \otimes \mathsf{Pos}$ is strictly more precise than in ASub and $\mathsf{Sharing} \otimes \mathsf{ASub}$ for six benchmarks of the suite

[2] `ftp://clip.dia.fi.upm.es/pub/software/benchmarks/`

Program	Vars	Performance				Accuracy							
		Ω_{Pos}		$\Omega^{\mathrm{diff}}_{\mathsf{Pos}}$		Ω_{Pos}		$\Omega^{\mathrm{diff}}_{\mathsf{Pos}}$		$\mathsf{ASub/Sharing} \otimes \mathsf{ASub}$		$\mathsf{ASub} \otimes \mathsf{Pos}$	
		Time	BDD	Time	BDD	Pairs	NLin	Pairs	NLin	Pairs	NLin	Pairs	NLin
aiakl	44	0.2	312	0.2	350	19	0			20			
bid	96	0.1	418	0.1	414	12	40			13		13	
boyer	122	0.1	496	0.2	496	36	19			38	24		
browse	28	0	72	0	121	0	0						
deriv	10	0	116	0	172	0	0						
fib	4	0	72	0	73	0	0						
grammar	34	0	20	0	20	4	0						
hanoiapp	16	0	96	0	93	0	0						
mmatrix	18	0	119	0	116	0	0						
occur	14	0	53	0	111	0	0						
peephole	76	0.5	511	0.6	486	19	24			21	36		
progeom	58	0	263	0	801	0	0						
qplan	288	19.0	78357	11.7	75738	193	196			$193/\infty$	$199/\infty$	197	
qsortapp	18	0	68	0	137	0	0						
query	14	0	68	0	68	0	0						
rdtok	104	0.1	149	0.1	149	8	8				9		
read	130	0.2	464	0.2	1597	0	0						
serialize	28	0.1	311	0.1	410	5	3			8	6		
tak	8	0	115	0	117	0	0						
warplan	139	1.2	2285	1.2	1203	84	106				109		
witt	190	50.8	390104	24.3	393736	379	326	377	321	$348/\infty$	$320/\infty$		
zebra	34	0	1143	0	1143	0	0						

Table 1: Experimental results for Ω_{Pos}

and is equally precise for the rest. However, Ω_{Pos} still improves the results of $\mathsf{ASub} \otimes \mathsf{Pos}$ in two cases (3 cases using $\Omega_{\mathsf{Pos}}^{\mathsf{diff}}$).

7 Conclusion

The relation graph approach to pair-sharing Ω is accurate and practical. A full implementation should be substantially faster than the prototype we illustrate here. There are a number of interesting questions that remain in relating Ω to Sharing and Sharing+Linearity. We also conjecture that the abstract operations of Ω are all optimal.

References

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

[2] T. Armstrong, K. Marriott, P. Schachte, and H. Søndergaard. Boolean functions for dependency analysis: Algebraic properties and efficient representation. In *1st International Symposium on Static Analysis*, volume 864 of *LNCS*, pages 266–280. Springer-Verlag, 1994.

[3] R. Bagnara, P. Hill, and E. Zaffanella. Set-sharing is redundant for pair-sharing. In *4th International Symposium on Static Analysis*, volume 1302 of *LNCS*, pages 53–67. Springer-Verlag, 1997.

[4] R. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[5] A. Cortesi, G. Filé, R. Giacobazzi, and C. Palamidessi. Complementation in abstract interpretation. In *2nd International Symposium on Static Analysis*, volume 983 of *LNCS*, pages 100–117. Springer-Verlag, September 1995.

[6] A. Cortesi, G. Filé, R. Giacobazzi, C. Palamidessi, and F. Ranzato. Complementation in abstract interpretation. *ACM TOPLAS*, 19(1):7–47, 1997.

[7] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM Symp. on Principles of Programming Languages*, pages 238–252, 1977.

[8] M. García de la Banda, K. Marriott, P.J. Stuckey, and H. Søndergaard. Differential methods in logic program analysis. *Journal of Logic Programming*, 35(1):1–37, 1998.

[9] M. Hermenegildo, F. Bueno, D. Cabeza, M. Carro, M. García de la Banda, and G. Puebla P. López-García. The CIAO multi-dialect compiler and system: An experimentation workbench for future (C)LP systems. In *Parallelism and Implementation of Logic and Constraint Logic Programming*, pages 65–85. Nova Science, 1999.

[10] M. Hermenegildo and K. Greene. &-Prolog and its performance: Exploiting independent And-Parallelism. In D. H. D. Warren and P. Szeredi, editors, *Proceedings of the 7th Int. Conf. on Logic Programming*, pages 253–268. MIT Press, 1990.

[11] D. Jacobs and A. Langen. Static analysis of logic programs for independent AND-parallelism. *JLP*, 13(2 & 3):291–314, 1992.

[12] J. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 2nd edition, 1988.

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

[14] K. Muthukumar and M. Hermenegildo. Compile-time derivation of variable dependency using Abstract Interpretation. *JLP*, 13(1, 2, 3 and 4):315–347, 1992.

[15] D. A. Plaisted. The occur-chack problem in Prolog. In *Int. Symp. on Logic Programming*, pages 272–280. IEEE Computer Society Press, 1984.

[16] H. Søndergaard. An application of abstract interpretation of logic programs: Occur check reduction. In B. Robinet and R. Wilhelm, editors, *Proceedings of European Symp. on Programming*, volume 213 of *LNCS*, pages 327–338. Springer-Verlag, 1986.

A Proofs

Theorem 1 *For a constraint C we have:*

$$\mathrm{Psh}_{\mathcal{L}}(C) \subseteq \left\{ \langle x, y\rangle \;\middle|\; x \looparrowright_{\sigma(C)} y \right\}$$

Proof. The proof is by induction on the number of constraints in C. The induction hypothesis is that (i) if x and y share in C there is a traversable path $x \looparrowright_{\sigma(C)} y$, and (ii) if x is non-linear in C there is a traversable path $x \looparrowright_{\sigma(C)} x$.

The base case is clear. Let us assume that the claim of the theorem holds for some number n of conjuncts and prove the step for $n+1$. Let $C = C' \wedge c$, where c is an equation.

Suppose x and y do not share in C', but share in C. We first consider the case that c is of the form $u = v$. Clearly the only way this constraint can cause sharing between x and y is if either (a) x shares with u and y shares with v (or vice versa) or (b) x and y both share with u and v is non-linear. Consider case (a). By induction there are traversable paths $x \looparrowright^{\rho_1}_{\sigma(C')} u$, and $v \looparrowright^{\rho_2}_{\sigma(C')} y$, and hence clearly there is a traversable path $x \looparrowright^{\rho}_{\sigma(C)} y$ where $\rho = \rho_1 \cdot \rho_2$ since the edge connecting u, v is new and compatible with all previous edges.

Consider case (b). Again by induction there are traversable paths $x \looparrowright^{\rho_1}_{\sigma(C')} u$, and $u \looparrowright^{\rho_2}_{\sigma(C')} y$, and $v \looparrowright^{\rho_3}_{\sigma(C')} v$, hence clearly there is a traversable path $x \looparrowright^{\rho}_{\sigma(C)} y$ where $\rho = \rho_1 \cdot \rho_3 \cdot \rho_2$ because the edge connecting u, v is new.

The case for c of the form $v = f(u, w)$ is similar. The only way this can cause sharing for x and y is (a) x shares with v and y shares with w.l.o.g. w, (b) y shares with v and x shares with w.l.o.g u, (c) x shares with u and y shares with w and v is non-linear, or (d) x shares with v and y shares with v and u shares with w. Cases (a) and (b) are identical to case (a) above. Case (c) is identical to case(b) above. For case (d) we have, by induction $x \looparrowright^{\rho_1}_{\sigma(C')} v$, and $v \looparrowright^{\rho_2}_{\sigma(C')} y$, and $u \looparrowright^{\rho_3}_{\sigma(C')} v$, and hence clearly there is a traversable path $x \looparrowright^{\rho}_{\sigma(C)} y$ where $\rho = \rho_1 \cdot \rho_3 \cdot \rho_2$ because the edges connecting v, u and w, v are new.

Suppose x is linear in C' but nonlinear in C. Then if c is of the form $u = v$ the only possibility that is makes x non-linear is w.l.o.g. that x shares with u and v is non-linear. Hence by induction there is a traversable path $x \looparrowright^{\rho_1}_{\sigma(C')} u$, and $v \looparrowright^{\rho_2}_{\sigma(C')} v$, and hence clearly there is a traversable path $x \looparrowright^{\rho}_{\sigma(C)} x$ where $\rho = \rho_1 \cdot \rho_2 \cdot \rho_1^{-1}$ because the edge connecting u, v is new.

Similarly if c if of the form $v = f(u, w)$ either (a) x shares with v and w.l.o.g. u is non-linear, (b) x shares with u and v is non-linear, or (c) x shares with v and u shares with w. The first two cases are similar to the case for $u = v$. For case c, by induction we have traversable paths $x \looparrowright^{\rho_1}_{\sigma(C')} v$ and $u \looparrowright^{\rho_2}_{\sigma(C')} w$, and hence clearly there is a traversable path $x \looparrowright^{\rho}_{\sigma(C)} x$ where $\rho = \rho_1 \cdot \rho_2 \cdot \rho_1^{-1}$ because the edges connecting v, u and w, v is new. □

Theorem 2 *For a constraint C and a groundness model S of C we have:*

$$\mathrm{Psh}_{\mathcal{L}}(C, S) \subseteq \left\{ \langle x, y\rangle \;\middle|\; x \looparrowright_{\sigma(C)\setminus S} y \right\}.$$

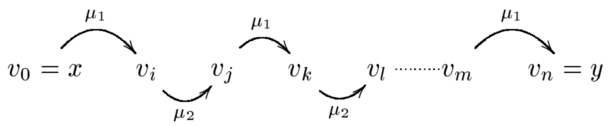


Figure 5: Illustration to the proof of Theorem 4.

PROOF. Consider the most general solution θ_S of C which grounds precisely the set $S = \{v_1, \ldots, v_n\}$, $S \subseteq \mathrm{vars}(C)$. Assume that θ_S maps each variable $v_i \in S$ to a ground terms g_i. Thus, θ_S is equivalent to the most general solution of a new constraint $C' = C \cdot \{\, v_i \mapsto g_i \mid i = 1 \ldots n \,\}$. Consequently, by Theorem 1 we have: $\mathrm{Psh}_{\mathcal{L}}(C') \subseteq \left\{\, \langle x, y \rangle \;\middle|\; x \leftrightarrow_{\sigma(C')} y \,\right\}$. The proof is complete by observing that the relation graph $\sigma(C')$ is exactly $\sigma(C) \setminus S$ since $\mathrm{vars}(C') = \mathrm{vars}(C) \setminus S$. Technically the theorem cannot be used as stated because the form of the constraints in C' is not a sequence of primitive unification constraints, but the proof can be straightforwardly modified to handle these new forms of equations. $\square$

Theorem 3 *The set $[\Omega_{\mathsf{gr}}]_{\approx}$ is finite for a finite domain of interest D.*
PROOF. According to Equation (9) two descriptions $\langle G_1, \phi_1 \rangle$ and $\langle G_2, \phi_2 \rangle$ belong to different equivalence classes, i.e. are not equivalent if and only if:

$$\begin{aligned}
&\exists d \subseteq D : \neg \left(\chi_{\mathrm{gr}(\phi_1, D)}(d) \Leftrightarrow \chi_{\mathrm{gr}(\phi_2, D)}(d) \right) \;\vee \\
&\exists x, y \in D \exists d \subseteq D :\; \neg\left(\chi_{\mathrm{Psh}_{\mathcal{L}}(G_1, \phi_1, D)}(x, y, d) \Leftrightarrow \chi_{\mathrm{Psh}_{\mathcal{L}}(G_2, \phi_2, D)}(x, y, d)\right)
\end{aligned}$$

Since the set D is finite, there is only a finite number of choices of d, x and y satisfying the above formula. Thus, for the domain D we can distinguish only between a finite number of "classes of interest" induced by the equivalence of Equation (9). $\square$

We say that an abstract constraint $\mu \in \mathcal{C}^{\alpha}$ *approximates* the concrete constraint C (based on the same domain of interest D) if $\alpha(C) \preceq_D \mu$. The *concretization* of an abstract constraint μ is a set containing all concrete constraints approximated by G:

$$\gamma(\mu) = \{\, C \mid \alpha(C) \preceq_D \mu \,\}. \tag{11}$$

Theorem 4 *The abstract conjunction operation is correct.*
PROOF. Given two descriptions $\mu_1 = \langle (V_1, E_1, I_1), \phi_1 \rangle$ and $\mu_2 = \langle (V_2, E_2, I_2), \phi_2 \rangle$ Assume two concrete constraints $C_1 \in \gamma(\mu_1)$ and $C_2 \in \gamma(\mu_2)$. We have to show that $C_1 \wedge C_2 \in \gamma(\mu_1 \curlywedge \mu_2)$, which by Equation (11) is the same as showing that $\alpha(C_1 \wedge C_2) \preceq_D (\mu_1 \curlywedge \mu_2)$. The latter is equivalent to showing that

$$\left\langle \sigma(C_1 \wedge C_2), \chi_{\mathrm{gr}(C_1 \wedge C_2)} \right\rangle \preceq_D \left\langle (V_1 \cup V_2,\; E_1 \cup E_2,\; I_1 \cup I_2),\; (\phi_1 \curlywedge_{\mathsf{gr}} \phi_2) \right\rangle.$$

Assuming $C = C_1 \wedge C_2$ and $G = (V_1 \cup V_2,\ E_1 \cup E_2,\ I_1 \cup I_2)$ we have to demonstrate that the following two conditions are satisfied:

$$\forall d \subseteq D : \left(\chi_{\mathrm{gr}(C,D)}(d) \Leftarrow \left(\chi_{\mathrm{gr}(\phi_1 \curlywedge_{\mathsf{gr}} \phi_2, D)}(d)\right)\right) \qquad (\dagger)$$

$$\forall x, y \in D,\ d \subseteq D : \left(\chi_{\mathrm{Psh}_{\mathcal{L}}(C,D)}(x,y,d) \ \Rightarrow\ \chi_{\mathrm{Psh}_{\mathcal{L}}(G,D)}(x,y,d)\right) \qquad (\ddagger)$$

Equation (†) is, in fact, a correctness claim of abstract conjunction used in the groundness analysis. This holds by assumption.

Now since $C_1 \in \gamma(\mu_1)$ we have that both (a) $\chi_{\mathrm{gr}(\mu_1)} \subseteq \chi_{\mathrm{gr}(C_1)}$ and (b) for all $d \in \chi_{\mathrm{gr}(\mu_1)}$ $x \looparrowright_{\sigma(C_1)\setminus d} y$ implies $x \looparrowright_{G_1\setminus d} y$. Similarly for C_2 and G_2.

We need to show that $\forall x, y, \forall d \in \phi_1 \curlywedge_{\mathsf{gr}} \phi_2$ that $x \looparrowright_{\sigma(C_1 \wedge C_2)\setminus d} y$ implies $x \looparrowright_{G_1 \cup G_2 \setminus d} y$. By the downwards closure of groundness information, we have that $d \in \chi_{\mathrm{gr}(\mu_1)} \subseteq \chi_{\mathrm{gr}(C_1)}$ and similarly $d \in \chi_{\mathrm{gr}(C_2)}$.

Suppose that $x \looparrowright_{\sigma(C_1 \wedge C_2)\setminus d} y$, then there must exist a traversable path of the form show in Figure 5 where $v_i \looparrowright_{\sigma(C_1)\setminus d} u_{i+1}$ and $u_i \looparrowright_{\sigma(C_2)\setminus d} v_i$. Hence there are traversable paths: $v_i \looparrowright_{G_1\setminus d} u_{i+1}$ and $u_i \looparrowright_{G_2\setminus d} v_i$, and hence there is a traversable path of the same form shown in Figure 5 $x \looparrowright_{G_1 \cup G_2 \setminus d} y$. □

Lemma 5 *Assume a substitution θ which grounds a set $t \subseteq \mathrm{dom}(\theta)$ of variables and two variables $x, y \in \mathrm{dom}(\theta) \setminus t$ such that x and y are independent (do not share) under θ. Then there exist two instances of θ which ground respectively the sets of variables $t_x \subseteq \mathrm{dom}(\theta)$ and $t_y \subseteq \mathrm{dom}(\theta)$ such that $x \in t_x$, $y \notin t_x$, $y \in t_y$, $x \notin t_y$ and $t = t_x \cap t_y$.*

PROOF. The two instances of θ_x and θ_y of θ are constructed as follows:

$$\begin{aligned}\theta_x &= \theta \circ \{\, v \mapsto a \mid v \in \mathrm{vars}(x\theta) \,\}, \\ \theta_y &= \theta \circ \{\, v \mapsto a \mid v \in \mathrm{vars}(y\theta) \,\}.\end{aligned}$$

where a is some fixed constant. The substitution θ_x grounds all variables in the range of $x\theta$ thus, grounding x and possibly some other variables. Similarly, θ_y grounds y and possibly some other variables. Consider the respective sets t_x and t_y of variables grounded by the substitutions:

$$\begin{aligned}t_x &= t \cup \{\, v \mid \mathrm{vars}(v\theta) \subseteq \mathrm{vars}(x\theta) \,\}, \\ t_y &= t \cup \{\, v \mid \mathrm{vars}(v\theta) \subseteq \mathrm{vars}(y\theta) \,\}.\end{aligned}$$

Namely, in the case of θ_x, if some variable v is mapped to a subset of $\mathrm{vars}(x\theta)$ then θ_x grounds v (as it grounds all variables of $x\theta$). The same applies to θ_y. Now observe that the independence of x and y in θ means that the corresponding sets $\mathrm{vars}(x\theta)$ and $\mathrm{vars}(y\theta)$ are disjoint. Consequently, all subsets of $\mathrm{vars}(x\theta)$ and $\mathrm{vars}(y\theta)$ are also disjoint. This in turn implies that the groundness sets added to t by θ_x and θ_y are disjoint. Thus, we have $x \in t_x$, $y \notin t_x$, $y \in t_y$, $x \notin t_y$ and $t = t_x \cap t_y$. □

Theorem 7 *A description $\mu = \langle G, \phi \rangle$ in Ω_{Def} and the corresponding abstraction $\alpha_\Omega(\mu)$ in SS^ρ express the same groundness and the same pair-sharing for the same domain of interest D.*

PROOF. Let us first observe that for any correct abstraction of $\mu \in \Omega_{\mathsf{Def}}$ to $\kappa \in SS^\rho$ the result κ may contain only sets in the form $s = D \setminus t$, where t satisfies the groundness description $\chi_{\mathrm{gr}(\mu)}$. Indeed, the presence of a set s in κ means that some concrete substitution (constraint) θ approximated by κ binds all members of s to terms containing a common variable v. Now assume an instance of θ which grounds all variables in range(θ) except v. Clearly, this instance leaves non-ground only and precisely the set s of variables of interest. Thus, $D \setminus s$ is a groundness model of θ, i.e., a model of $\chi_{\mathrm{gr}(\mu)}$.

Equation (10) restricts the result $\kappa = \alpha_\Omega(\mu)$ to only those inverse models of $\chi_{\mathrm{gr}(\mu)}$ which consist of variables which can all share pairwise. Let us prove that (1) κ describes the same groundness as $\chi_{\mathrm{gr}(\mu)}$ and (2) all pair-sharing of μ is preserved in κ. In order to prove (1) assume a model t of $\chi_{\mathrm{gr}(\mu)}$ such that two variables $x, y \in D \setminus t$ cannot share in μ. Lemma 5 implies that for any concrete substitution θ described by μ there exist two instances of θ with the groundness models t_x and t_y such that $t = t_x \cap t_y$. Thus, t_x and t_y are also models of $\chi_{\mathrm{gr}(\mu)}$. Since $\chi_{\mathrm{gr}(\mu)}$ is a down-closed Def-formula, the model t is redundant. Consequently, since the groundness of $\kappa = \alpha_\Omega(\mu)$ is interpreted as a Def-formula, the set $D \setminus t$ can be omitted without affecting the groundness expressed by κ. Thus, the result of Equation (10) defines the same groundness information in SS^ρ (also in $\mathsf{Sharing}$) as $\chi_{\mathrm{gr}(\mu)}$.

Let us prove (2). In order to derive a contradiction assume that the pair-sharing between some variables u and v possible under a groundness model t of μ is not preserved in κ. In that case the corresponding sharing set $s = D \setminus t$ is removed from κ, which means that it contains at least one pair $\langle x, y\rangle$ of independent (under the groundness t) variables. Consider a substitution θ approximated by μ such that $\mathrm{dom}(\theta) = D$, θ grounds the set t of variables, u and v share in θ, and x and y are independent. According to Lemma 5 there exist two instances θ_x and θ_y of θ which ground respectively the sets t_x and t_y of variables with the following properties: $x \in t_x$, $y \notin t_x$, $y \in t_y$, $x \notin t_y$ and $t = t_x \cap t_y$. Note that t_x and t_y are models of $\chi_{\mathrm{gr}(\mu)}$ and thus, the corresponding sharing sets $s_x = D \setminus t_x$ and $s_y = D \setminus t_y$ may potentially appear in κ. If u and v appear together in s_x (s_y) then according to our initial assumption this set is also removed from κ. Otherwise the pair-sharing $\langle u, v\rangle$ would be possible under the groundness t_x (t_y) and consequently under t. In that case we can re-iterate our argument replacing the initial s by its proper subset s_x (s_y) and assuming another pair $\langle x, y\rangle$ of variables which cannot share in μ under t_x (t_y). Finally, this process reaches the state when none of the sets s_x and s_y contains the pair $\langle u, v\rangle$. However, this pair is contained in $s = s_x \cup s_y$ and thus, we assume (without loss of generality) that $u \in s_x$, $v \notin s_x$ and $v \in s_y$, $u \notin s_y$.

Let us re-consider the substitution θ approximated by μ and its instances θ_x and θ_y. As we have shown above grounding of x in θ_x implies grounding of v (since $v \notin s_x$ we have $v \in t_x$). Similarly, grounding of y involves grounding of u. Thus, we have $\mathrm{vars}(v\theta) \subseteq \mathrm{vars}(x\theta)$ and $\mathrm{vars}(u\theta) \subseteq \mathrm{vars}(y\theta)$. Now since, u and v pair-share in θ we have $\mathrm{vars}(u\theta) \cap \mathrm{vars}(v\theta) \neq \emptyset$. Consequently $\mathrm{vars}(x\theta) \cap \mathrm{vars}(y\theta) \neq \emptyset$ which clearly contradicts the initial assumption that x and y are independent in θ. The contradiction means that there always exists a set $s' \subseteq s$

which contains both u and v but includes no pair of independent variables. $\square$

Lemma 8 *For two descriptions $\mu \in \Omega_{\mathsf{Def}}$ and $\kappa \in SS^\rho$ over the same domain of interest D we have:*

$$\alpha_\Omega(\mu) \preceq_{SS^\rho} \kappa \;\Rightarrow\; \alpha_\Omega(\mu \curlywedge \alpha([x = r])) \preceq_{SS^\rho} \mathrm{amgu}_{SS^\rho}(\kappa, x, r)$$

PROOF. Assume that $\mu = \langle G, \phi_\mu \rangle$, where ϕ_μ is a Def-formula. Let us also assume $\mu' = \mu \curlywedge \alpha([x = r])$ and $\kappa' = \mathrm{amgu}_{SS^\rho}(\kappa, x, r)$.

Let us prove first that the groundness description of μ' is at least as precise as the groundness description of κ'. Assume that ϕ_κ is a Def-formula expressing the groundness of κ.[3] The relation $\alpha_\Omega(\mu) \preceq_{SS^\rho} \kappa$ yields the corresponding relation $\phi_\mu \to \phi_\kappa$ between the groundness descriptions. The groundness descriptions in both Ω_{Def} and SS^ρ are computed as Def-formulas, explicitly in case of "$\curlywedge$" and implicitly in case of amgu_{SS^ρ}. The respective groundness descriptions of the results are $\phi_{\mu'} = \phi_\mu \wedge \phi$ and $\phi_{\kappa'} = \phi_\kappa \wedge \phi$ where $\phi = (x \leftrightarrow \bigwedge \mathrm{vars}(r))$ is a groundness description of the constraint $x = r$. Clearly, this gives us we have $\phi_{\mu'} \to \phi_{\kappa'}$.

Let us prove now that all pair-sharing defined by μ' is preserved in κ'. Assume a pair of variables of interest $\langle u, v \rangle$ occurring in the same set s of $\alpha_\Omega(\mu')$. The inverse $t = D \setminus s$ corresponds to a groundness model of μ' which allows a fully non-ground traversable path between u and v in μ'. The groundness model t is a also a model of μ. There are two cases: either (1) the pair-sharing $\langle u, v \rangle$ exists in μ or (2) the pair-sharing $\langle u, v \rangle$ is a result of abstract conjunction. Consider the case (1). Since t is also a model of μ, the same traversable path connecting u with v exists also in μ. In this case either $\alpha_\Omega(\mu)$ includes the sharing set $s = D \setminus t$ or it includes a set $s' = D \setminus t'$, where $t' \supset t$ and $\langle x, y \rangle \subseteq s'$ (see the proof of Theorem 7). In any case $\alpha_\Omega(\mu)$ includes a fully non-ground (in both μ and μ') set containing the pair $\langle u, v \rangle$. Consequently, κ also includes a set with these properties. Observe that the operation amgu_{SS^ρ} preserves all pair-sharing contained in fully non-ground sets. Thus, the pair-sharing $\langle u, v \rangle$ is contained also in κ'.

Consider the case (2). Without loss of generality we can divide it into the following three sub-cases: (2.1) u and v both share with x in μ (but not with each other) (2.2) u shares with $w \in \mathrm{vars}(r)$ and v shares with $z \in \mathrm{vars}(r)$ in μ and (2.3) u shares with x and v shares with $y \in \mathrm{vars}(r)$. Cases (2.1) and (2.2) assume that the pair-sharing $\langle u, v \rangle$ is enabled by the non-linearity of x or r respectively which results from the abstract conjunction. Similarly to the case (1) we show that κ contains: (2.1) two sets s_1 and s_2 such that $\{u, x\} \subseteq s_1$ and $\{v, x\} \subseteq s_2$, (2.2) two sets s_1 and s_2 such that $\{u, w\} \subseteq s_1$ and $\{v, z\} \subseteq s_2$ and (2.3) two sets s_1 and s_2 such that $\{u, x\} \subseteq s_1$ and $\{v, y\} \subseteq s_2$. Assume $\alpha = \mathrm{rel}(\kappa, \{x\})$ and $\beta = \mathrm{rel}(\kappa, \mathrm{vars}(r))$. It is easy to see that for all three cases the operation $\mathrm{amgu}_{\mathsf{PSh}}$ takes the union $s_1 \cup s_2$ in one of its steps: (2.1) during $\alpha \uplus \alpha$, (2.2) during $\beta \uplus \beta$ and (2.3) during $(\alpha \uplus \alpha) \uplus (\beta \uplus \beta)$. Thus, the pair-sharing $\langle u, v \rangle$ found in μ' is also contained in κ'. $\square$

[3] The interested reader is referred to [5, 6] for the discussion of how ϕ_κ is constructed from κ.

Lemma 9

$$\alpha_\Omega(\mu_1) \preceq_{SS^\rho} \kappa_1 \wedge \alpha_\Omega(\mu_2) \preceq_{SS^\rho} \kappa_2 \;\Rightarrow\; \alpha_\Omega(\mu_1 \curlyvee \mu_2) \preceq_{SS^\rho} (\kappa_1 \cup \kappa_2)$$

PROOF. Straightforward since each pair of variables which share in $(\mu_1 \curlyvee \mu_2)$ already shares either in μ_1 or in μ_2. Thus, $\alpha_\Omega(\mu_1 \curlyvee \mu_2) = \alpha_\Omega(\mu_1) \cup \alpha_\Omega(\mu_2)$. $\square$

We assume the definitions of $gi_{\mathcal{D}}$, $sdiff_{\mathcal{D}}$ and $std_{\mathcal{D}}$ of [8].

Theorem 11 *For description domain Ω_{Pos} for all programs P and goals G, $gi_{\Omega_{\mathsf{Pos}}}(P,G) \preceq_{\Omega_{\mathsf{Pos}}} std_{\Omega_{\mathsf{Pos}}}(P,G)$.*

PROOF. It is straightforward to show that Ω_{Pos} is order independent and conjunction is commutative, hence we have by Theorem 6.2 of [8] that $gi_{\Omega_{\mathsf{Pos}}}(P,G) = sdiff_{\Omega_{\mathsf{Pos}}}(P,G)$. We now show that $sdiff_{\Omega_{\mathsf{Pos}}}(P,G) \preceq std_{\Omega_{\mathsf{Pos}}}(P,G)$.

The proof is by induction on the Kleene sequence of computing the least fixpoint of the general goal-dependent analysis, maintaining the following invariant: at each program point where the description of the differential semantics is (D_{in}, D_δ) and the description given by the standard semantics is D_{std}, we have that $D_{in} \curlywedge D_\delta \preceq D_{std}$.

Again since Pos is condensing it is clear that the groundness information at each program point is such that $gr_{in} \wedge gr_\delta = gr_{std}$.

The proof proceeds as for Theorem 6.2 of [8] except for the point at which a return from call is performed. Consider the program fragment

$$\begin{array}{l} \ldots, [1], p(\bar{v}), [4]. \ldots \\ p(\bar{v}) \;:-\; B_1[2] \\ p(\bar{v}) \;:-\; B_2[3] \end{array}$$

We assume for simplicity that p is defined by only two rules, the extension to an arbitrary number is straightforward.

Let (D^x_{in}, D^x_δ) and D^x_{std} be the differential and standard descriptions at program point $[x]$. By induction we have that $D^x_{in} \curlywedge D^x_\delta \preceq D^x_{std}$ for $x \in \{1,2,3\}$.

Now we have that $D^2_{in} = D^3_{in} = Aproj(\bar{v}, D^1_{in} \curlywedge D^1_\delta)$ by definition of the simple differential. Hence $gr^2_{in} = gr^3_{in} = \bar{\exists}_{\bar{v}} gr^1_{std}$.

For each edge $(u,v,\psi) \in D^x_\delta$ there exists edge $(u,v,\psi') \in D^x_{std}$ such that $\psi \wedge gr^x_{in} \to \psi'$ for $x \in \{2,3\}$ because each edge $(u,v,\psi) \in D^x_\delta$ creates an edge $(u,v,\psi \wedge gr^x_{in}) \in D^x_{in} \curlywedge D^x_\delta$. Hence each edge $(u,v,\bar{\exists}_{\bar{v}}\psi) \in Aproj(\bar{v}, D^x_\delta)$ there exists edge $(u,v,\bar{\exists}_{\bar{v}}\psi') \in Aproj(\bar{v}, D^x_{std})$ such that $\bar{\exists}_{\bar{v}}\psi \wedge gr^x_{in} \to \bar{\exists}_{\bar{v}}\psi'$ for $x \in \{2,3\}$ since gr^x_{in} only involves variables $\bar{v}$.

Hence $(\bar{\exists}_{\bar{v}} gr^1_{std}, \emptyset) \curlywedge Aproj(\bar{v} D^x_\delta) \preceq Aproj(\bar{v}, D^x_{std})$ for $x \in \{2,3\}$. Now we have $D^4_{std} = D^1_{std} \curlywedge (Aproj(\bar{v}, D^2_{std}) \curlyvee Aproj(\bar{v}, D^3_{std}))$ and $D^4_\delta = D^1_\delta \curlywedge (Aproj(\bar{v}, D^2_\delta) \curlyvee Aproj(\bar{v}, D^3_\delta))$. We want to show that $D^1_{in} \curlywedge D^4_\delta \preceq D^4_{std}$.

$$
\begin{array}{ll}
 & D^1_{in} \curlywedge D^4_\delta \\
= & (D^1_{in} \curlywedge D^1_\delta) \curlywedge (Aproj(\bar{v}, D^2_\delta) \curlyvee Aproj(\bar{v}, D^3_\delta)) \\
\preceq & D^1_{std} \curlywedge (Aproj(\bar{v}, D^2_\delta) \curlyvee Aproj(\bar{v}, D^3_\delta)) \\
= & D^1_{std} \curlywedge (\bar{\exists}_{\bar{v}} gr^1_{std}, \emptyset) \curlywedge (Aproj(\bar{v}, D^2_\delta) \curlyvee Aproj(\bar{v}, D^3_\delta)) \\
= & D^1_{std} \curlywedge (((\bar{\exists}_{\bar{v}} gr^1_{std}, \emptyset) \curlywedge Aproj(\bar{v}, D^2_\delta)) \curlyvee \\
 & \qquad ((\bar{\exists}_{\bar{v}} gr^1_{std}, \emptyset) \curlywedge Aproj(\bar{v}, D^3_\delta))) \\
\preceq & D^1_{std} \curlywedge (Aproj(\bar{v}, D^2_{std}) \curlyvee Aproj(\bar{v}, D^3_{std})) \\
= & D^4_{std}
\end{array}
$$

The first step by the definition of D^4_δ and associativity of $\curlywedge$, the second step by the invariant applied to program point [1] and the monotonicity of $\curlywedge$, the third step because any the groundness information $\bar{\exists}_{\bar{v}} gr^1_{std}$ is implied by D^1_{std}, the fourth step because $\curlywedge$ where there are no edges distributes over $\curlyvee$ (Pos is condensing), the fifth step is argued above, and the final step is by the definition of D^4_{std}.

$\square$