Optimality in goal-dependent Analysis of Sharing

Gianluca Amato[1] and Francesca Scozzari[2]

[1] Dipartimento di Matematica e Informatica, Università di Udine.
[2] Dipartimento di Informatica, Università di Pisa.
amato@dimi.uniud.it, scozzari@di.unipi.it

Abstract. We cope with the problem of correctness and optimality for logic programs analysis by abstract interpretation. We refine the goal-dependent framework appeared in [7] by fixing a result of correctness and introducing two specialized operators for forward and backward unification. We provide the best correct abstractions of the concrete operators in the case of set-sharing analysis. We show that the precision of the overall analysis is strictly improved and that, in some cases, we gain precision w.r.t. more complex domains involving linearity and freeness information.

1 Introduction

In the field of static analysis by abstract interpretation [10, 11], the most interesting (and studied) properties for logic programs are arguably groundness and sharing. Groundness analysis aims at discovering ground variables in the answer substitutions, while the goal of (set) sharing analysis is to detect sets of variables which share a common variable. While the results on groundness analysis seem to converge toward the domain `Pos` [1, 9] which is commonly recognized as the optimal domain for detecting groundness property, the same did not happen for the sharing property. The problem of finding a "good" domain for sharing analysis is still open. Among the various proposals, we find new domains for more efficient and/or more precise analyses [2, 15], combinations of domains (e.g. including freeness and/or linearity information [14, 21]), and many techniques for improving the abstract operators used in the analysis [17, 14]. The common starting point of most of these proposals is the domain `Sharing` by Langen [18, 16], slightly modified in [7] where all the abstract operators are proved to be correct and optimal. Actually, the proof of correctness of the abstract unification operator in [7] has a flaw and we give a counterexample in this paper. When an abstract operator is not correct w.r.t. the concrete operator, the common solution is to design a new correct (and possibly optimal) abstract operator. But we think that, in this case, the incorrectness is mainly due to a wrong definition of the concrete unification operator. In fact, such an operator does not perform the necessary renamings in order to avoid variable clashes, which is common to almost any semantics for logic programs. Following this intuition, we propose to modify the concrete semantics, by using a more "intuitive" concrete unification, which performs the necessary renamings, and define the corresponding optimal abstract unification operator.

Once the correctness of the framework has been asserted, we can cope with the problem of precision. The choice of the authors of using a unique unification operator for performing both forward and backward unification leads to a significant loss of precision. The forward unification computes the entry substitution by unifying the calling substitution with the head of the clauses. The backward unification computes the success substitution from the calling substitution, the exit substitution and the head of the clauses. It is immediate to verify that the forward unification can be computed as a backward unification with an empty exit substitution. But, at the abstract level, this stratagem leads to a loss of precision. Therefore we define a specialized operator which is able to exploit the particular characteristics of the forward unification. We also prove that this operator is correct and optimal w.r.t. the concrete one.

Furthermore, we propose a different backward concrete unification which makes use of the matching operation instead of the standard unification. This choice is not new and has been already suggested in [14, 17]. We show that using a matching operator leads to a significant augment of precision already at the concrete level. We design a backward abstract unification operator, correct and optimal w.r.t. the new concrete operator, which leads to a strictly more precise analysis. Finally, we compare our results with other techniques for improving precision and efficiency of sharing analysis.

2 Notations

Given a set A, let $\wp(A)$ be the set of subsets of A and $\wp_f(A)$ be the set of finite subsets of A. Given two posets A and B, we denote by $A \to B$ the space of monotonic functions from A to B ordered pointwise. When an order is not specified, we assume the least informative order ($x \leq y \iff x = y$). Given A, C complete lattices, a Galois Insertion [10] $\langle \alpha, \gamma \rangle : C \leftrightarrows A$ is given by a pair of maps $\alpha : C \to A$, $\gamma : A \to C$ such that $\alpha(c) \leq a \iff c \leq \gamma(a)$ and α is onto. We say that an abstract operator $f^\alpha : A \to A$ is *correct* w.r.t. a concrete operator $f : C \to C$ when $\alpha \circ f \leq_A f^\alpha \circ \alpha$ and it is *optimal* when $\alpha \circ f = f^\alpha \circ \alpha$. In this case f^α is called the *best correct approximation* of f.

Let $\mathcal{V}$ be a countable set of variables and Term be the set of terms built from $\mathcal{V}$ and a fixed signature. Given a term t, we denote by $\mathit{vars}(t)$ the set of variables occurring in t and by $\mathit{uvars}(t)$ the subset of $\mathit{vars}(t)$ whose elements only appear once in t (e.g., $\mathit{uvars}(t(x,y) = t(y,z)) = \{x, z\}$). We will abuse the notation and apply vars and uvars to any syntactic object with the obvious meaning. We denote by ϵ the empty substitution and by $\{x_1/t_1, \ldots, x_n/t_n\}$ a substitution θ with $\theta(x_i) = t_i \neq x_i$. We denote by $\mathit{vars}(\theta)$ the set $\mathrm{dom}(\theta) \cup \mathrm{range}(\theta)$ and, given $U \in \wp_f(\mathcal{V})$, we denote by $\theta|_U$ the projection of θ over U, i.e. the only substitution such that $\theta|_U(x) = \theta(x)$ if $x \in U$ and $\theta|_U(x) = x$ otherwise. Given θ_1 and θ_2 two substitutions with disjoint domains, we denote by $\theta_1 \uplus \theta_2$ the substitution θ such that the domain $\mathrm{dom}(\theta) = \mathrm{dom}(\theta_1) \cup \mathrm{dom}(\theta_2)$ and $\theta(x) = \theta_i(x)$ if $x \in \mathrm{dom}(\theta_i)$, for $i \in \{1, 2\}$. The application of a substitution θ to a term t is denoted by $t\theta$ or $\theta(t)$. Given two substitutions θ and δ, their composition, denoted by $\theta \circ \delta$ is given by $(\theta \circ \delta)(x) = \theta(\delta(x))$. Instantiation induces a preorder on substitutions: θ is more general than δ, denoted by $\delta \leq \theta$, if there exists σ such that $\sigma \circ \theta = \delta$. The set of idempotent substitutions is denoted by *Subst*, while *Ren* denotes the set of all the renamings (i.e. invertible substitutions). Any idempotent substitution σ is an mgu (most general unifier) of the corresponding set of equations $\mathrm{Eq}(\sigma) = \{x = \theta(x) \mid x \in \mathrm{dom}(\sigma)\}$. In the following, we will abuse the notation and denote by $\mathrm{mgu}(\sigma_1, \ldots, \sigma_n)$, when it exists, the substitution $\mathrm{mgu}(\mathrm{Eq}(\sigma_1) \cup \ldots \cup \mathrm{Eq}(\sigma_n))$.

3 Correctness in the Cortesi-Filè Framework

Cortesi and Filè define in [7] an abstract domain for recovering sharing information based upon a variant of the domain `Sharing` by Jacobs and Langen [16] and give a set of optimal abstract operators. However, the proof has a flaw, namely the fact that the abstract unification $\mathbf{U}_{Sh}$ is not correct w.r.t. the concrete one $\mathbf{U}_{Rs}$. To show why this happens, and since this will be useful in the rest of the paper, we briefly recall the definitions of the domains and operators from [7].

3.1 Concrete Domain and Operations

The concrete domain is $\mathtt{Rsub} = (\wp(\mathit{Subst}) \times \wp_f(\mathcal{V})) \cup \{\top_{\mathrm{Rs}}, \bot_{\mathrm{Rs}}\}$, which is a complete lattice, partially ordered as follows: $\top_{\mathrm{Rs}}$ is the top element, $\bot_{\mathrm{Rs}}$ is the bottom element and $[\Sigma_1, U_1] \sqsubseteq_{\mathrm{Rs}} [\Sigma_2, U_2]$ if and only if $U_1 = U_2$ and $\Sigma_1 \subseteq \Sigma_2$. An object $[\Sigma, U]$ is a set of substitution Σ where the set of variables of interest U is explicitly provided. The least upper bound of `Rsub` is denoted by $\sqcup_{\mathrm{Rs}}$ and collects substitutions coming from different computational paths. The concrete projection $\pi_{\mathrm{Rs}} : \mathtt{Rsub} \times \wp_f(\mathcal{V}) \to \mathtt{Rsub}$ is defined as follows:

$$\pi_{\mathrm{Rs}}(\top_{\mathrm{Rs}}, U_2) = \top_{\mathrm{Rs}} \qquad\qquad \pi_{\mathrm{Rs}}(\bot_{\mathrm{Rs}}, U_2) = \bot_{\mathrm{Rs}}$$
$$\pi_{\mathrm{Rs}}([\Sigma_1, U_1], U_2) = [\Sigma_1, U_1 \cap U_2]$$

It restricts the variables of interest of a set of substitutions. Finally, the concrete unification is $\mathbf{U}_{Rs} : \mathtt{Rsub} \times \mathtt{Rsub} \times \mathit{Subst} \to \mathtt{Rsub}$ such that:

$$\begin{aligned}
\mathbf{U}_{Rs}(\bot_{\mathrm{Rs}}, \xi, \delta) =& \mathbf{U}_{Rs}(\xi, \bot_{\mathrm{Rs}}, \delta) = \bot_{\mathrm{Rs}} \\
\mathbf{U}_{Rs}(\xi, \top_{\mathrm{Rs}}, \delta) =& \mathbf{U}_{Rs}(\top_{\mathrm{Rs}}, \xi, \delta) = \top_{\mathrm{Rs}} \qquad \text{if } \xi \neq \bot_{\mathrm{Rs}} \\
\mathbf{U}_{Rs}([\Sigma_1, U_1], [\Sigma_2, U_2], \delta) =& [\{\mathrm{mgu}(\sigma_1, \sigma_2, \delta) \mid \sigma_1 \in \Sigma_1, \sigma_2 \in \Sigma_2, \\
& \mathit{vars}(\sigma_1) \cap \mathit{vars}(\sigma_2) = \emptyset\}, U_1 \cup U_2]
\end{aligned}$$

Although it is well defined for all the values of the domain, the use of $\mathbf{U}_{Rs}([\Sigma_1, U_1], [\Sigma_2, U_2], \delta)$ is restricted only to those values such that $U_1 \cap U_2 = \emptyset$ and $\mathit{vars}(\delta) \subseteq U_1 \cup U_2$, since this is the only use in the semantics. Also, the corresponding abstract operators will be defined under these conditions only. It is worth noting that the authors use the condition $\mathit{vars}(\sigma_1) \cap \mathit{vars}(\sigma_2) = \emptyset$ in order to avoid variables clashes between the two chosen substitutions.

3.2 Abstract Domain and Operations

Now we briefly recall the definition of the abstract domain $\mathtt{Sharing}$ [16, 7].

$$\mathtt{Sharing} = \{[A,U] \mid A \subseteq \wp(U), (A \neq \emptyset \Rightarrow \emptyset \in A), U \in \wp_f(\mathcal{V})\} \cup \{\top_{Sh}, \bot_{Sh}\} \ .$$

Intuitively, an abstract object $[A,U]$ describes the relations between the variables in U: if $S \in A$ then the variables in S are allowed to share a common variable. For instance, $[\{\{x,y\},\{z\},\emptyset\},\{x,y,z\}]$ represents the substitutions where x and y may possibly share, while z is independent from both x and y.

The domain is ordered like $\mathtt{Rsub}$, with $\top_{Sh}$ and $\bot_{Sh}$ as the greatest and least element respectively, and $[A_1,U_1] \sqsubseteq_{Sh} [A_2,U_2]$ iff $A_1 = A_2$ and $U_1 \subseteq U_2$. The abstraction function $\alpha_{\mathrm{Sh}} : \mathtt{Rsub} \to \mathtt{Sharing}$ is defined as follows:

$$\alpha_{\mathrm{Sh}}(\bot_{\mathrm{Rs}}) = \bot_{Sh} \qquad \alpha_{\mathrm{Sh}}(\top_{\mathrm{Rs}}) = \top_{Sh}$$
$$\alpha_{\mathrm{Sh}}([\Sigma,U]) = [\{occ(\sigma,y) \cap U \mid y \in \mathcal{V}, \sigma \in \Sigma\}, U]$$

where $occ(\sigma,y) = \{z \in \mathcal{V} \mid y \in vars(\sigma(z))\}$. We call *sharing group* an element of $\wp_f(\mathcal{V})$. To ease the notation, often we will write a sharing group as the sequence of its elements in any order (e.g. $\mathtt{xyz}$ represents $\{x,y,z\}$) and we omit the empty set when clear from the context. The abstract operators do behave exactly as the concrete ones on $\top_{Sh}$ and $\bot_{Sh}$, while the other cases are the following:

$$[A_1,U_1] \sqcup_{Sh} [A_2,U_2] = \begin{cases} [A_1 \cup A_2, U_2] & \text{if } U_1 = U_2 \\ \top_{Sh} & \text{otherwise} \end{cases}$$
$$\pi_{Sh}([A,U],V) = [\{B \cap V \mid B \in A\}, V \cap U]$$
$$\mathbf{U}_{Sh}([A_1,U_1],[A_2,U_2],\delta) = [\mathbf{u}_{Sh}(A_1 \cup A_2, \delta), U_1 \cup U_2]$$
$$\mathbf{u}_{Sh}(A,\epsilon) = A$$
$$\mathbf{u}_{Sh}(A,\{x/t\} \uplus \theta) = \mathbf{u}_{Sh}(A \setminus (\mathbf{rel}(A,\{x\}) \cup \mathbf{rel}(A, vars(t))) \cup \mathbf{bin}(\mathbf{rel}(A,\{x\})^*, \mathbf{rel}(A, vars(t))^*), \theta).$$

where $\mathbf{u}_{Sh} : \wp(\wp_f(\mathcal{V})) \times Subst \to \wp(\wp_f(\mathcal{V}))$ is defined by induction, by using the following auxiliary operators:

- the *closure under union* (or *star union*) $\cdot^* : \wp(\wp_f(\mathcal{V})) \to \wp(\wp_f(\mathcal{V}))$

$$A^* = \{\bigcup T \mid \emptyset \neq T \in \wp_f(A)\}$$

- the *extraction of relevant components* $\mathbf{rel} : \wp(\wp_f(\mathcal{V})) \times \wp_f(\mathcal{V}) \to \wp(\wp_f(\mathcal{V}))$:

$$\mathbf{rel}(A,V) = \{T \in A \mid T \cap V \neq \emptyset\}$$

- the *binary union* $\mathbf{bin} : \wp(\wp_f(\mathcal{V})) \times \wp(\wp_f(\mathcal{V})) \to \wp(\wp_f(\mathcal{V}))$:

$$\mathbf{bin}(A,B) = \{T_1 \cup T_2 \mid T_1 \in A, T_2 \in B\}$$

In the rest of the paper, we will abuse the notation and write $\mathbf{rel}(A,o)$ for $\mathbf{rel}(A, vars(o))$, where o is any syntactic object.

3.3 Problems in Correctness

In an object $[\Sigma, U] \in \texttt{Rsub}$, all the variables which do appear in Σ and not in U are thought as they were existentially quantified. This means that it does not matter what these variables really are, but only their relationships with other variables in the same substitution. The same idea also applies to ex-equations [19] and the domain *ESubst* in [16]. However, this intuition does not apply to the concrete operator for unification $\mathbf{U}_{Rs}$, since it does not perform any renaming. Actually, $\mathbf{U}_{Rs}$ only checks that σ_1 and σ_2 do not have variables in common, without considering their sets of variables of reference U_1 and U_2; namely it checks that $\mathit{vars}(\sigma_1) \cap \mathit{vars}(\sigma_2) = \emptyset$. This unification can lead to counterintuitive results.

Example 1. Consider the following concrete unification:

$$\mathbf{U}_{Rs}([\{\{x/y\}\}, \{x\}], [\{\epsilon\}, \{y\}], \epsilon) = [\{\{x/y\}\}, \{x, y\}] \ . \tag{1}$$

Being $\mathit{vars}(\epsilon) = \emptyset$, the concrete unification operator allows us to unify $\{x/y\}$ with ϵ without renaming the variable y, which is not a variable of interest in the first element but it is treated as if it was, and this also causes the incorrectness of $\mathbf{U}_{Sh}$. If we consider Eq. (1) and compute the result on the abstract side by using the abstract unification operator $\mathbf{U}_{Sh}$ we have:

$$\begin{aligned} &\mathbf{U}_{Sh}(\ \alpha_{Sh}([\{\{x/y\}\}, \{x\}]), \alpha_{Sh}([\{\epsilon\}, \{y\}]), \epsilon) \\ =\ &\mathbf{U}_{Sh}(\quad [\{\mathtt{x}\}, \{x\}], \quad [\{\mathtt{y}\}, \{y\}], \quad \epsilon) = [\{\mathtt{x}, \mathtt{y}\}, \{x, y\}] \ . \end{aligned} \tag{2}$$

This is not a correct approximation of the concrete result, since:

$$\alpha_{Sh}([\{\{x/y\}\}, \{x, y\}]) = [\{\mathtt{xy}\}, \{x, y\}] \not\sqsubseteq_{Sh} [\{\mathtt{x}, \mathtt{y}\}, \{x, y\}] \ . \tag{3}$$

This counterexample proves that the abstract unification operator $\mathbf{U}_{Sh}$ proposed in [7] is not correct w.r.t. the concrete one $\mathbf{U}_{Rs}$. This problem can be solved by introducing a stronger check on variable clashes, namely by replacing the condition $\mathit{vars}(\sigma_1) \cap \mathit{vars}(\sigma_2) = \emptyset$ with $(\mathit{vars}(\sigma_1) \cup U_1) \cap (\mathit{vars}(\sigma_2) \cup U_2) = \emptyset$ in the definition of $\mathbf{U}_{Rs}$, thus obtaining the following operator.

$$\begin{aligned} \mathbf{U}_{Rs}([\Sigma_1, U_1], [\Sigma_2, U_2], \delta) = [\{\mathrm{mgu}(\sigma_1, \sigma_2, \delta) \mid \sigma_1 \in \Sigma_1, \sigma_2 \in \Sigma_2, \\ (\mathit{vars}(\sigma_1) \cup U_1) \cap (\mathit{vars}(\sigma_2) \cup U_2) = \emptyset\}, U_1 \cup U_2] \ , \end{aligned} \tag{4}$$

In the rest of the paper, we will refer to $\mathbf{U}_{Rs}$ as given by the above definition. Moreover, when we design domains and operators for developing static analyses, we need to be sure that the concrete operators are powerful enough to allow the definition of the semantics of the programming language we are interested in. In the case of logic programming, this semantics will be some sensible abstraction of the SLD-derivations [6], such as computed answers. We think that the operators $\pi_{\mathrm{Rs}}, \sqcup_{\mathrm{Rs}}$ and $\mathbf{U}_{Rs}$ do not allow for the definition of such a semantics without using some additional operator which performs renamings. Actually, in [9] the authors recognize that "in a typical semantic construction, renaming is performed" and in [8] they define a more complex unification with renamings. However, the corresponding abstract operators are given only for groundness analysis. We recall the definition of the concrete operator. Let Atoms be the syntactic categories for atoms. The concrete unification $\mathbf{U}'_{\mathrm{Rs}} : \mathtt{Rsub} \times \mathtt{Rsub} \times \mathsf{Atoms} \times \mathsf{Atoms} \rightarrow \mathtt{Rsub}$ defined in [8] is the following:

$$\begin{aligned} &\mathbf{U}'_{\mathrm{Rs}}([\Sigma_1, U_1], [\Sigma_2, U_2], A_1, A_2) = \\ &\qquad = \mathbf{U}_{Rs}([\rho_1(\Sigma_1), \rho_1(U_1)], [\rho_2(\Sigma_2), U_2], \mathrm{mgu}(\rho_1(A_1) = A_2)) \end{aligned} \tag{5}$$

where $(\rho_1, \rho_2) = \mathit{Apart}(U_2)$ provided $\mathit{vars}(A_1) \subseteq U_1$ and $\mathit{vars}(A_2) \subseteq U_2$, $\bot_{\mathrm{Rs}}$ otherwise. We still need to define *Apart*. Given $U_2 \in \wp_f(\mathcal{V})$, take a partition $\{V_1, V_2\}$ of $\mathcal{V}$ such that V_1 and V_2 are infinite and $U_2 \subseteq V_2$. Then $\mathit{Apart}(U_2) = (\rho_1, \rho_2)$ where $\rho_1 : \mathcal{V} \rightarrow V_1$ and $\rho_2 : \mathcal{V} \rightarrow V_2$ are bijections such that, for each $x \in U_2$, $\rho_2(x) = x$. We apply such bijections to syntactic objects as if they were substitutions. Intuitively, the maps ρ_1 and ρ_2 are used to rename the variables in Σ_1 and Σ_2 in such a way that $\rho_1(\Sigma_1)$ and $\rho_2(\Sigma_2)$ are renamed apart each others, and variables in U_2 are not renamed.

The definition of $\mathbf{U}'_{\mathrm{Rs}}$ allows us to define the abstract unification $\mathbf{U}'_{Sh}$ corresponding to $\mathbf{U}'_{\mathrm{Rs}}$ by performing the necessary renamings, as suggested by the concrete operator.

$$\begin{aligned} &\mathbf{U}'_{Sh}([S_1, U_1], [S_2, U_2], A_1, A_2) \\ &\qquad\qquad = \mathbf{U}_{Sh}([\rho_1(S_1), \rho_1(U_1)], [S_2, U_2], \mathrm{mgu}(\rho_1(A_1) = A_2)) \end{aligned} \tag{6}$$

where $(\rho_1, \rho_2) = \mathit{Apart}(U_2)$ provided $\mathit{vars}(A_1) \subseteq U_1$ and $\mathit{vars}(A_2) \subseteq U_2$, $\bot_{Sh}$ otherwise. Note that we do not need to apply ρ_2 to S_2 since $\mathit{vars}(S_2) \subseteq U_2$ and we now $\rho_2|_{U_2}$ is the identity. It is easy to show that $\mathbf{U}'_{Sh}$ is the best correct operator induced by the concrete unification $\mathbf{U}'_{\mathrm{Rs}}$, as shown by the next theorem.

Theorem 1. *$\mathbf{U}'_{Sh}$ (respectively $\mathbf{U}_{Sh}$) is correct and optimal w.r.t. $\mathbf{U}'_{\mathrm{Rs}}$ (respectively $\mathbf{U}_{Rs}$).*

This result fixes the correctness problem in the framework of [7]. The following sections will be devoted to examine several improvements concerning the precision of the resulting semantics.

4 Forward and Backward Unification

A semantics using $\pi_{\mathrm{Rs}}, \sqcup_{\mathrm{Rs}}$ and $\mathbf{U}'_{\mathrm{Rs}}$ has been defined in [8]. In the following, we briefly recall the relevant definitions and show some possible improvements for the analysis which arise from specializing the unification operator in two different contexts.

Let Atoms, Clauses, Body and Progs be the syntactic categories for atoms, clauses, bodies and programs respectively. The semantics is parametric with respect to a complete lattice $\mathcal{X}$, where a *denotation* is an element in the set of monotonic maps:

$$\mathcal{D}en = \mathsf{Atoms} \rightarrow \mathcal{X} \rightarrow \mathcal{X} \ . \tag{7}$$

We have the following semantic functions:

$$\begin{aligned}\mathcal{P} &: \mathsf{Progs} \to \mathcal{D}en\\ \mathcal{C} &: \mathsf{Clauses} \to \mathcal{D}en \to \mathcal{D}en\\ \mathcal{B} &: \mathsf{Body} \to \mathcal{D}en \to \mathcal{X} \to \mathcal{X}\end{aligned}$$

$$\begin{aligned}&\mathcal{P}[\![P]\!] = \mathit{lfp}\,\lambda d.\left(\bigsqcup_{cl \in P} \mathcal{C}[\![cl]\!]d\right)\\ &\mathcal{C}[\![H \leftarrow B]\!]dAx = \pi_{\mathcal{X}}(\mathbf{U}_{\mathcal{X}}(x', x, H, A), x)\\ &\quad \text{where } x' = \mathcal{B}[\![B]\!]d(\pi_{\mathcal{X}}(\mathbf{U}_{\mathcal{X}}(x, id[\![H \leftarrow B]\!], A, H), id[\![H \leftarrow B]\!]))\\ &\mathcal{B}[\![\lambda]\!]dx = x\\ &\mathcal{B}[\![A : B]\!]dx = \mathcal{B}[\![B]\!]d(dAx)\end{aligned}$$

whose definitions are given by means of the following operators:

$$\begin{aligned}\mathbf{U}_{\mathcal{X}} &: \mathcal{X} \times \mathcal{X} \times \mathsf{Atoms} \times \mathsf{Atoms} \to \mathcal{X}\\ \pi_{\mathcal{X}} &: \mathcal{X} \times \mathcal{X} \to \mathcal{X}\\ id_{\mathcal{X}} &: \mathsf{Clauses} \to \mathcal{X}\end{aligned}$$

The instantiation of $\mathcal{X}$ to `Rsub` is obtained by defining:

$$\begin{aligned}\mathbf{U}_{\mathcal{X}} &= \mathbf{U}'_{\mathrm{Rs}}\\ \pi_{\mathcal{X}}([\Sigma_1, U_1], [\Sigma_2, U_2]) &= \pi_{\mathrm{Rs}}([\Sigma_1, U_1], U_2)\\ id_{\mathcal{X}}(cl) &= [\{\epsilon\}, \mathit{vars}(cl)]\end{aligned}$$

while for $\mathcal{X} = \mathtt{Sharing}$ we replace $\mathbf{U}'_{\mathrm{Rs}}$ and π_{Rs} with $\mathbf{U}'_{Sh}$ and π_{Sh} and we define $id_{\mathcal{X}}(cl) = \alpha_{Sh}([\{\epsilon\}, \mathit{vars}(cl)]) = [\{\{x\} \mid x \in \mathit{vars}(cl)\}, \mathit{vars}(cl)]$.

It is routine to check that all the semantic functions are continuous, the least fixpoint in the definition of $\mathcal{P}$ does exist and the abstract semantics over `Sharing` is correct w.r.t. the concrete one over `Rsub`, assuming the standard lifting [10] of the Galois connection $\langle \alpha_{Sh}, \gamma_{Sh} \rangle$ to $\langle \alpha'_{Sh}, \gamma'_{Sh} \rangle$: $\mathsf{Atoms} \to \mathtt{Rsub} \to \mathtt{Rsub} \leftrightarrows \mathsf{Atoms} \to \mathtt{Sharing} \to \mathtt{Sharing}$.

4.1 Forward Unification

The concrete unification $\mathbf{U}'_{\mathrm{Rs}}$ is used in two different contexts:

- as a *forward unification* to compute the collecting *entry substitution* $\pi_{\mathrm{Rs}}(\mathbf{U}'_{\mathrm{Rs}}(x, id[\![H \leftarrow B]\!], A, H), id[\![H \leftarrow B]\!])$ from the collecting *call substitution* x;
- as a *backward unification* to compute the collecting *answer substitution* $\pi_{\mathrm{Rs}}(\mathbf{U}'_{\mathrm{Rs}}(x', x, H, A), x)$ from the collecting *exit substitution* x'.

Although $\mathbf{U}'_{Sh}$ is optimal w.r.t. $\mathbf{U}'_{\mathrm{Rs}}$, this is not the case for a specialized version of $\mathbf{U}'_{\mathrm{Rs}}$, as used in in the forward unification, where the second argument is always of the form $[\{\epsilon\}, \mathit{vars}(H \leftarrow B)]$. Therefore, a specialized version of $\mathbf{U}'_{Sh}$ could improve the precision of the analysis.

Example 2. Assume, without loss of generality, that $(\rho_1, \rho_2) = \mathit{Apart}(U_2)$ and ρ_1 restricted to $\{x, y, z\}$ is the identity. Then, we have that:

$$\mathbf{U}'_{Sh}([\{\mathtt{xy}, \mathtt{yz}\}, \{x, y, z\}], id_{Sh}[\![p(u, v, w) \leftarrow]\!], p(x, y, z), p(u, v, w)) = \\ [\{\mathtt{xyuv}, \mathtt{yzvw}, \mathtt{xyzuvw}\}, \{x, y, z, u, v, w\}] \ . \quad (8)$$

If we compute the projection on the variables $\{u, v, w\}$ we obtain the entry substitution $[\{\mathtt{uv}, \mathtt{vw}, \mathtt{uvw}\}, \{u, v, w\}]$. However, we know that u, v, w are free in $id_{\mathrm{Rs}}[\![p(u, v, w) \leftarrow]\!]$. Following [14], we can avoid to compute the star unions when considering the binding y/v in $\mathbf{u}_{Sh}$, obtaining the smaller result $[\{\mathtt{xyuv}, \mathtt{yzvw}\}, \{x, y, z, u, v, w\}]$. If we now compute the projection on the variables $\{u, v, w\}$ we obtain the entry substitution $[\{\mathtt{uv}, \mathtt{vw}\}, \{u, v, w\}]$, with an obvious gain of precision.

Example 3. Let us consider the following unification.

$$\mathbf{U}'_{Sh}([\{\mathtt{xy}, \mathtt{xz}\}, \{x, y, z\}], id_{Sh}[\![p(t(u, v), h, k) \leftarrow]\!], p(x, y, z), p(t(u, v), h, k)) = \\ [\mathbf{bin}(\{\mathtt{xyh}, \mathtt{xzk}, \mathtt{xyzhk}\}, \{\mathtt{u}, \mathtt{v}, \mathtt{uv}\}), \{x, y, z, h, k, u\}] \ . \quad (9)$$

Since the term $t(u, v)$ is linear and independent from x, following [14] we can avoid to compute the star union over $\{\mathtt{xy}, \mathtt{xz}\}$, obtaining the abstract object $[\mathbf{bin}(\{\mathtt{xyh}, \mathtt{xzk}\}, \{\mathtt{u}, \mathtt{v}, \mathtt{uv}\}), \{x, y, z, h, k, u\}]$. If we project on $\{h, k, u, v\}$ we obtain $\mathbf{bin}(\{\mathtt{h}, \mathtt{k}\}, \{\mathtt{u}, \mathtt{v}, \mathtt{uv}\})$ against $\mathbf{bin}(\{\mathtt{h}, \mathtt{k}, \mathtt{hk}\}, \{\mathtt{u}, \mathtt{v}, \mathtt{uv}\})$. With the new improvement, we are able to prove the independence of h from k.

These examples show that, when computing forward abstract unification as a specialized version of the abstract unification, there is a loss of precision. In fact, such a forward abstract unification operator is not optimal. We now show that it is possible to design an optimal operator for forward unification which is able to exploit the information of linearity and freeness coming from the fact that the second argument is always of the form $[\{\epsilon\}, \mathit{vars}(H \leftarrow B)]$. Note that we are not proposing to embed freeness and linearity information inside the domain, but only to use all the information coming from the syntax of clauses.

4.2 The Refined Forward Unification

Our first step is to change the definition of the semantic function for clauses by introducing a new operator for forward unification $\mathbf{U}^f_{\mathcal{X}} : \mathcal{X} \times \mathsf{Clauses} \times \mathsf{Atoms} \times \mathsf{Atoms} \to \mathcal{X}$, in the following way:

$$\begin{aligned} &\mathcal{C}[\![H \leftarrow B]\!]dAx = \pi_{\mathcal{X}}(\mathbf{U}_{\mathcal{X}}(x', x, H, A), x) \\ &\quad \text{where } x' = \mathcal{B}[\![B]\!]d(\pi_{\mathcal{X}}(\mathbf{U}^f_{\mathcal{X}}(x, H \leftarrow B, A, H), id[\![H \leftarrow B]\!])) \end{aligned} \quad (10)$$

For the concrete semantics, we simply instantiate $\mathbf{U}^f_{\mathcal{X}}$ with $\mathbf{U}'^f_{\mathrm{Rs}}$ defined as:

$$\mathbf{U}'^f_{\mathrm{Rs}}([\Sigma, U], cl, A_1, A_2) = \mathbf{U}'_{\mathrm{Rs}}([\Sigma, U], id_{\mathrm{Rs}}[\![cl]\!], A_1, A_2) \ , \quad (11)$$

so that nothing changes. However when we move to the abstract domain $\mathtt{Sharing}$, directly abstracting $\mathbf{U}'^f_{\mathrm{Rs}}$ gives more precise results than abstracting $\mathbf{U}'_{\mathrm{Rs}}$ in $\mathbf{U}'_{Sh}$ and composing it with id_{Sh}.

Reasoning according to this rule, we could think that a better approximation could be reached by abstracting $\pi_{\mathrm{Rs}}(\mathbf{U'}^f_{\mathrm{Rs}}(x, H \leftarrow B, A, H), id[\![H \leftarrow B]\!])$ as a whole. However, since π_{Rs} is complete [7], this does not happen. Studying the direct abstraction of this composition would still be useful to find a direct implementation which is more efficient than computing $\mathbf{U'}^f_{\mathrm{Rs}}(x, H \leftarrow B, A, H)$ and projecting later, but we do not consider this problem here.

Following the approach of [16, 7], we first define an operator $\mathbf{U}^f_{Sh}$ which does not perform renamings.

Definition 1. *We define the* forward abstract unification *without renamings* $\mathbf{U}^f_{Sh}$: $\mathtt{Sharing} \times \wp_f(\mathcal{V}) \times \mathit{Subst} \to \mathtt{Sharing}$ *as:*

$$\mathbf{U}^f_{Sh}([S_1, U_1], U_2, \theta) = [\mathbf{u}^f_{Sh}(S_1 \cup \{\{x\} \mid x \in U_2\}, U_2, \theta), U_1 \cup U_2]$$

where $\mathbf{u}^f_{Sh} : \wp(\wp_f(\mathcal{V})) \times \wp_f(\mathcal{V}) \to \wp(\wp_f(\mathcal{V}))$ *is defined as:*

$$\mathbf{u}^f_{Sh}(S, U, \epsilon) = S$$

$$\begin{aligned}\mathbf{u}^f_{Sh}(S, U, \{x/t\} \uplus \delta) = \mathbf{u}^f_{Sh}(&(S \setminus (\mathbf{rel}(S,t) \cup \mathbf{rel}(S,x))) \cup \\ &\mathbf{bin}(\mathbf{rel}(S,x), \mathbf{rel}(S,t)), U \setminus \{x\}, \delta)\end{aligned}$$

if $x \in U$

$$\begin{aligned}\mathbf{u}^f_{Sh}(S, U, \{x/t\} \uplus \delta) = \mathbf{u}^f_{Sh}(&(S \setminus (\mathbf{rel}(S,t) \cup \mathbf{rel}(S,x))) \cup \\ &\mathbf{bin}(\mathbf{rel}(S,x), \mathbf{rel}(S,Y)^*) \cup \\ &\mathbf{bin}(\mathbf{rel}(S,x)^*, \mathbf{rel}(S,Z)^*) \cup \\ &\mathbf{bin}(\mathbf{bin}(\mathbf{rel}(S,x)^*, \mathbf{rel}(S,Z)^*), \mathbf{rel}(S,Y)^*), \\ &U \setminus \mathit{vars}(\{x/t\}), \delta)\end{aligned}$$

if $x \notin U$, $Y = \mathit{uvars}(t) \cap U$, $Z = \mathit{vars}(t) \setminus Y$.

The domain is restricted to the case $U_1 \cap U_2 = \emptyset$ *and* $\mathit{vars}(\theta) \subseteq U_1 \cup U_2$.

The idea is simply to carry on, in the second argument of $\mathbf{u}_{Sh}$, the set of variables which are definitively free and to apply the optimizations for the abstract unification with linear terms and free variables [14]. Actually, while the case for $x \in U$ is standard, the case for $x \notin U$ exploits some optimizations which are not found in the literature. When $Z = \emptyset$, we obtain:

$$(S \setminus (\mathbf{rel}(S,t) \cup \mathbf{rel}(S,x))) \cup \mathbf{bin}(\mathbf{rel}(S,x), \mathbf{rel}(S,Y)^*) \ ,$$

which is the standard result when the term t is linear and independent from x. However, when $Z \neq \emptyset$, the standard optimizations do not apply, since t cannot be proved to be linear and independent from x, and we should obtain the following standard result:

$$(S \setminus (\mathbf{rel}(S,t) \cup \mathbf{rel}(S,x))) \cup \mathbf{bin}(\mathbf{rel}(S,x)^*, \mathbf{rel}(S,t)^*) \ .$$

However, we are able to avoid some star unions by distinguishing the variables in t which are "linear and independent" (the set Y) from the others (the set Z), and observing that two sharing groups in $\mathbf{rel}(S,x)$ may be merged together only under the effect of the unification with some variable in Z. We will come back later to this topic.

We can now define the forward abstract unification with renamings $\mathbf{U'}^{f}_{Sh}$: $\mathtt{Sharing} \times \mathsf{Clauses} \times \mathsf{Atoms} \times \mathsf{Atoms} \to \mathtt{Sharing}$ by exploiting the operator $\mathbf{U}^{f}_{Sh}$ previously defined. We only need to introduce the necessary renamings, as done for the concrete case:

$$\mathbf{U'}^{f}_{Sh}([S_1, U_1], cl, A_1, A_2) = \mathbf{U}^{f}_{Sh}([\rho_1(S_1), \rho_1(U_1)], \mathit{vars}(cl), \mathrm{mgu}(\rho_1(A_1) = A_2)) \tag{12}$$

with $(\rho_1, \rho_2) = \mathit{Apart}(\mathit{vars}(cl))$ provided that $\mathit{vars}(A_1) \subseteq U_1$ and $\mathit{vars}(A_2) \subseteq \mathit{vars}(cl)$, $\bot_{Sh}$ otherwise. Using $\mathbf{U'}^{f}_{Sh}$ instead of $\mathbf{U}'_{Sh}$ gives the improvements in precision we have discussed in the Examples 2 and 3. Moreover, it is not possible to do better then $\mathbf{U'}^{f}_{Sh}$ if we want to remain correct over all the conditions, as the following theorem proves.

Theorem 2. *$\mathbf{U'}^{f}_{Sh}$ is correct and optimal w.r.t. $\mathbf{U'}^{f}_{\mathrm{Rs}}$.*

The proof of this theorem is very similar to the proof of the analogous theorem for $\mathbf{U}_{Sh}$ and $\mathbf{U}_{Rs}$ which can be found in [7].

Since $\mathbf{U}^{f}_{Sh}$ generates less sharing groups then $\mathbf{U}_{Sh}$ and since checking whether a variable is in U is easy, we can expect an improvement in the efficiency of the analysis by replacing $\mathbf{U}_{Sh}$ with $\mathbf{U}^{f}_{Sh}$ in the computation of the entry substitution. If computing Y and Z at each step of $\mathbf{u}^{f}_{Sh}$ seems difficult, it is always possible to precompute these values before the actual analysis begins, since they depend from the syntax of the program only. Moreover, in the definition of $\mathbf{u}^{f}_{Sh}$, when $x \in U$ we can replace $\mathbf{rel}(S, x)$ with $\{\{x\}\}$, since θ is an idempotent substitution and $x \notin U_1$. Finally, from the result of optimality, it immediately follows that $\mathbf{U'}^{f}_{Sh}$ yields the same result, regardless of the choice of the mgu, and the result of the algorithm for computing $\mathbf{u}^{f}_{Sh}$ is independent from the ordering of the bindings. We have said before that this operator introduces new optimizations which, up to our knowledge, are not used even in more complex domains for sharing analysis which include linearity and freeness information. We give here one example which shows their effects.

Example 4. Let us consider the following unification.

$$\begin{aligned}\mathbf{U}'_{Sh}([\{\mathtt{xw}, \mathtt{xz}, \mathtt{yw}, \mathtt{yz}\}, \{x, y, w, z\}], \mathit{id}_{Sh}[\![p(f(u,h), f(u,k), s, t) \leftarrow]\!],\\ p(x, y, w, z), p(f(u,h), f(u,k), s, t))\ .\end{aligned} \tag{13}$$

By applying the optimizations suggested from the unification algorithm in presence of linearity and freeness information [14], we may start from the abstract object $S = \{\mathtt{xw}, \mathtt{xz}, \mathtt{yw}, \mathtt{yz}, \mathtt{s}, \mathtt{t}, \mathtt{u}, \mathtt{h}\}$ and process the bindings one at a time, keeping in mind that s, t, u and h are initially free. This means that in the binding $x/f(u,h)$, the term $f(u,h)$ is linear, and therefore we can avoid to compute the star union in $\mathbf{rel}(S, x)$, thus obtaining:

$$\begin{aligned}\{\mathtt{s}, \mathtt{t}, \mathtt{yw}, \mathtt{yz}\} \cup \mathbf{bin}(\{\mathtt{xw}, \mathtt{xz}\}, \{\mathtt{u}, \mathtt{h}, \mathtt{uh}\}) =\\ \{\mathtt{s}, \mathtt{t}, \mathtt{yw}, \mathtt{yz}, \mathtt{xwu}, \mathtt{xwh}, \mathtt{xzu}, \mathtt{xzh}, \mathtt{xwuh}, \mathtt{xzuh}\}\ .\end{aligned}$$

However, after this unification, the variable u can be bound to a non-linear term. Therefore, when we consider the binding $y/f(u,k)$, according to [14], we are forced to compute all the star unions, obtaining:

$$\{\mathtt{s},\mathtt{t}\} \cup \mathbf{bin}(\{\mathtt{yw},\mathtt{yz}\}^*, (\{\mathtt{k}\} \cup \mathbf{bin}(\{\mathtt{xw},\mathtt{xz}\},\{\mathtt{u},\mathtt{uh}\}))^*) \cup \{\mathtt{xwh},\mathtt{xzh}\} \ .$$

Finally, in the bindings w/t and z/s we may omit all the star unions since t and s are free, and we get the final result:

$$\mathbf{bin}(\{\mathtt{yws},\mathtt{yzt}\}^*, (\{\mathtt{k}\} \cup \mathbf{bin}(\{\mathtt{xws},\mathtt{xzt}\},\{\mathtt{u},\mathtt{uh}\}))^*) \cup \{\mathtt{xwsh},\mathtt{xzth}\} \ .$$

When we project over $\{u,h,k,s,t\}$, we obtain the sharing group $\mathtt{stk}$. However, when we consider the second binding, we know that k is free and independent from y, and this is enough to apply a new optimization. In fact, k can share with more than one sharing group related to y only if k shares with u. If we compute the abstract unification with our algorithm we obtain:

$$\begin{aligned}&\{\mathtt{ywsk},\mathtt{yztk}\} \cup \mathbf{bin}(\{\mathtt{yws},\mathtt{yzt}\}^*, \mathbf{bin}(\{\mathtt{xws},\mathtt{xzt}\},\{\mathtt{u},\mathtt{uh}\})^*)\\ &\cup \mathbf{bin}(\mathbf{bin}(\{\mathtt{yws},\mathtt{yzt}\}^*, \mathbf{bin}(\{\mathtt{xws},\mathtt{xzt}\},\{\mathtt{u},\mathtt{uh}\})^*), \{k\}) \cup \{\mathtt{xwsh},\mathtt{xzth}\}\end{aligned} \qquad (14)$$

and when we project over $\{u,h,k,s,t\}$, the sharing group $\mathtt{stk}$ does not appear. The result does not change by permuting the order of the bindings. If we consider the binding $y/f(u,k)$ before $x/f(u,h)$, with the standard operators we get:

$$\mathbf{bin}(\{\mathtt{xws},\mathtt{xzt}\}^*, (\{\mathtt{h}\} \cup \mathbf{bin}(\{\mathtt{yws},\mathtt{yzt}\},\{\mathtt{u},\mathtt{uk}\}))^*) \cup \{\mathtt{ywsk},\mathtt{yztk}\}$$

and, when we project over $\{u,h,k,s,t\}$, we obtain the sharing group $\mathtt{sth}$, which does not appear in our result.

4.3 Matching and Backward Unification

In this section we study some optimizations for the computation of the exit substitution. When we compute $\mathbf{U}_{Rs}(x',x,\delta)$, we essentially unify all pairs σ' and σ, elements of x' and x, with δ. However, we could consider only the pairs in which σ' is an instance of $\mathrm{mgu}(\sigma,\delta)$ w.r.t. the variable of interest of x'. If this does not hold, then σ' cannot be a success substitution corresponding to the calling substitution σ, and therefore we are unifying two objects which pertain to different computational paths, with an obvious loss of precision, already at the concrete level. This problem has been pointed out in [19, Section 5.5].

Formally speaking, we write $\theta \preceq_V \sigma$ to denote that θ is an instance of σ w.r.t. the set of variables V, and we define:

$$\theta \preceq_V \sigma \iff \exists \eta.\theta \leq (\eta \circ \sigma)|_V \ .$$

Equivalently, it holds $\theta \preceq_V \sigma \iff \exists \eta.\forall v \in V.\theta(v) = \eta(\sigma(v))$. Intuitively, if $\theta \preceq_V \sigma$ it means that some existential variables have been instantiated in σ to obtain θ. For example $\{x/a\} \preceq_{\{x\}} \{x/y\}$ since we may take $\eta = \{y/a\}$

and $(\{y/a\} \circ \{x/y\})|_{\{x\}} = \{x/a, y/a\}|_{\{x\}} = \{x/a\}$. Note that $\{x/a\} \not\preceq \{x/y\}$ according to the standard instantiation ordering.

Following this idea, we define a new operator for concrete backward unification given as follows.

$$\mathbf{U}^b_{\mathrm{Rs}}([\Sigma_1, U_1], [\Sigma_2, U_2], \delta) = [\{\mathrm{mgu}(\sigma_1, \sigma_2, \delta) \mid \sigma_1 \in \Sigma_1, \sigma_2 \in \Sigma_2,$$
$$(\mathit{vars}(\sigma_1) \cup U_1) \cap (\mathit{vars}(\sigma_2) \cup U_2) = \emptyset, \sigma_2 \preceq_{U_2} \mathrm{mgu}(\sigma_1, \delta)\}, U_1 \cup U_2] \ .$$

We also define the version with renamings $\mathbf{U'}^b_{\mathrm{Rs}}$ as it has been done for $\mathbf{U}'_{\mathrm{Rs}}$.

$$\begin{aligned}&\mathbf{U'}^b_{\mathrm{Rs}}([\Sigma_1, U_1], [\Sigma_2, U_2], A_1, A_2) = \\ &\quad = \mathbf{U}^b_{\mathrm{Rs}}([\rho_1(\Sigma_1), \rho_1(U_1)], [\rho_2(\Sigma_2), U_2], \mathrm{mgu}(\rho_1(A_1) = A_2))\end{aligned} \tag{15}$$

where $(\rho_1, \rho_2) = \mathit{Apart}(U_2)$, and we instantiate $\mathbf{U}_{\mathcal{X}}$ with $\mathbf{U'}^b_{\mathrm{Rs}}$ in the semantic definition of Eq. (10).

The idea of using a refined operator for computing the exit substitution is not new. Both [14], working in the operational framework of [4], and [17] in a denotational framework similar to ours, propose an abstract operator which is correct w.r.t. $\mathbf{U}^b_{\mathrm{Rs}}$. Also [21] use a refined algorithm for backward unification, although it is not presented in algebraic form. However, none of these operators is optimal (see Example 5). We now want to define the optimal abstract operator $\mathbf{U'}^b_{Sh}$ corresponding to $\mathbf{U'}^b_{\mathrm{Rs}}$. This is accomplished by composing the forward unification operator $\mathbf{U}^f_{Sh}$ with a new auxiliary operator match_{Sh}.

Definition 2. *Given* $[S_1, U_1], [S_2, U_2] \in \mathtt{Sharing}$ *with* $U_2 \subseteq U_1$, *we define*

$$\mathsf{match}_{Sh}([S_1, U_1], [S_2, U_2]) = [S'_1 \cup \{X \in (S''_1)^* \mid X \cap U_2 \in S_2\}, U_1]$$

where $S'_1 = \{B \in S_1 \mid B \cap U_2 = \emptyset\}$ *and* $S''_1 = S_1 \setminus S'_1$, *and*

$$\mathbf{U}^b_{Sh}([S_1, U_1], [S_2, U_2], \delta) = \mathsf{match}_{Sh}(\mathbf{U}^f_{Sh}([S_1, U_1], U_2, \delta), [S_2, U_2]) \ .$$

The idea is to collect only the sharing groups whose projection over U_2 belongs to S_2. As before, $\mathbf{U'}^b_{Sh}$ is obtained from $\mathbf{U}^b_{Sh}$ by introducing the necessary renamings.

Example 5. Let $U_1 = \{x, y, z\}$, $U_2 = \{u, v, w\}$, $\delta = \{x/u, y/v, z/w\}$, $\Sigma_1 = \{\{y/t(x, z, z)\}, \{y/t(x, x, z)\}\}$, $\Sigma_2 = \{\{v/t(u, w, w)\}, \{v/t(u, u, w)\}\}$. If we compute $[\Sigma, U_1 \cup U_2] = \mathbf{U}'_{\mathrm{Rs}}([\Sigma_1, U_1], [\Sigma_2, U_2], \delta)$, assuming $(\rho_1, \rho_2) = \mathit{Apart}(U_2)$ such that $\rho_1|_{U_1} = \epsilon$, we obtain $\theta = \{y/t(x, x, x), z/x, u/x, v/t(x, x, x), w/x\} \in \Sigma$. Given $[S_1, U_1] = \alpha_{\mathrm{Sh}}([\Sigma_1, U_1])$, $[S_2, U_2] = \alpha_{\mathrm{Sh}}([\Sigma_2, U_2])$, $S_1 = \{\mathtt{xy}, \mathtt{yz}\}$ and $S_2 = \{\mathtt{uv}, \mathtt{vw}\}$, we obtain $[S, U_1 \cup U_2] = \mathbf{U}'_{Sh}([S_1, U_1], [S_2, U_2], \delta)$ and $\mathtt{xyzuvw} \in S$.

However, note that θ is obtained by unifying $\sigma_1 = \{y/t(x, z, z)\}$ with $\sigma_2 = \{v/t(u, u, w)\}$, and that $\sigma_2(v) = t(u, u, w)$ is not an instance of $(\mathrm{mgu}(\sigma_1, \delta))(v) = t(x, z, z)$. Therefore, σ_1 and σ_2 do pertain to different computational paths. If we compute $[\Sigma', U_1 \cup U_2] = \mathbf{U'}^b_{\mathrm{Rs}}([\Sigma_1, U_1], [\Sigma_2, U_2], \delta)$ we obtain:

$$\Sigma' = \{\{y/t(x, z, z), u/x, v/t(x, z, z), w/z\}, \{y/t(x, x, z), u/x, v/t(x, x, z), w/z\}\}$$

which does not contain θ. In the abstract domain, we have:

$$\mathbf{U}'^{b}_{Sh}([S_1, U_1], [S_2, U_2], p(x, y, z), p(u, v, w)) = [\{\mathtt{xyuv}, \mathtt{yzvw}\}, U_1 \cup U_2] \enspace .$$

After the unification we know that x and z are independent. On the contrary, the operators defined in [17] and [14] cannot establish this property. The algorithm in [21] computes the same result in this particular example, but since their matching is partially performed by first projecting the sharing information on the term positions of the calling atom and of the clause head, this does not hold in general. For example, the algorithm in [21] states that x and z may possibly share when the unification is performed between the calling atom $p(t(x, y, z))$ and the head $p(t(u, v, w))$, where t is a function symbol and p a unary predicate.

We can prove that we have defined the best correct abstraction of the backward concrete unification.

Theorem 3. $\mathbf{U}^{b}_{Sh}$ *(respectively* $\mathbf{U}'^{b}_{Sh}$*) is correct and optimal w.r.t.* $\mathbf{U}^{b}_{\mathrm{Rs}}$ *(respectively* $\mathbf{U}'^{b}_{\mathrm{Rs}}$*).*

It is worth noting that, in order to obtain the optimality result, it is necessary to use the matching match_{Sh} as given in Def. 2 and the forward unification $\mathbf{U}^{f}_{Sh}$ as given in Def. 1. The combination of these two operators allows us to prove the optimality of $\mathbf{U}'^{b}_{Sh}$. It is now easy to give an example of a program which can be analyzed with a better precision w.r.t. the original framework in [7].

Example 6. Consider the trivial program with just one clause `p(u,v,w)` $\leftarrow$ and the goal $p(x, y, z)$ with calling substitution $\{\mathtt{xy}, \mathtt{yz}\}$. Using our abstract operators, we obtain the entry substitution $\{\mathtt{uv}, \mathtt{vw}\}$ and the success substitution $\{\mathtt{xy}, \mathtt{yz}\}$ (see Ex. 2 and 5), thus proving that x and z are independent. If we replace either $\mathbf{U}'^{b}_{Sh}$ or $\mathbf{U}'^{f}_{Sh}$ with $\mathbf{U}'_{Sh}$, then the success substitution will contain the sharing group `xyz`. In fact, as shown in Ex. 2, the entry substitution in the latter case would be $[\{\mathtt{uv}, \mathtt{vw}, \mathtt{uvw}\}, \{u, v, w\}]$. If we compute the success substitution we obtain:

$$\pi_{Sh}(\mathbf{U}'_{Sh}([\{\mathtt{uv}, \mathtt{vw}, \mathtt{uvw}\}, \{u, v, w\}], [\{\mathtt{xy}, \mathtt{yz}\}, \{x, y, z\}], p(u, v, w), p(x, y, z)),$$
$$\{x, y, z\}) = [\{\mathtt{xy}, \mathtt{yz}, \mathtt{xyz}\}, \{x, y, z\}]$$

which contains the sharing group `xyz`.

Note that the improvement in the previous example is obtained with a program in *head normal form*. Usually, when programs are in head normal form, the forward and backward unification may be replaced by renamings, which are complete and do not cause any loss in precision. However, there is the need of an unification operator for the explicit constraints which appear in the body of the clauses. In general, the analyses we obtain in our framework are more precise than those which can be obtained by using the standard domain `Sharing` by translating the same program in head normal form.

Example 7. Consider again Ex. 6 and the program `p(u,f(s),w)` $\leftarrow$ which is not in head normal form. Using our abstract operators, we obtain the success substitution $\{\mathtt{xy}, \mathtt{yz}\}$, as in Ex. 6. If we normalize the program, we obtain the clause `p(u,v,w)` $\leftarrow$ `v=f(s)`. The entry substitution obtained from $\{\mathtt{xy}, \mathtt{yz}\}$ by simply renaming the variables x, y, z in u, v, w and introducing the new variable s is $\{\mathtt{uv}, \mathtt{vw}, \mathtt{s}\}$. By using the standard operator for unification, when applying the binding $v/f(s)$ we obtain $\{\mathtt{uvs}, \mathtt{vws}, \mathtt{uvws}\}$, and thus the success substitution will contain the sharing group `xyz`, so resulting in a loss of precision. It is worth noting that it would still be possible to use our improved forward abstract unification in a normalized program by enlarging the set of variables of interest only when new variables are effectively met, instead of adding all the variables which appear in the body of a clause once for all when the entry substitution is computed. In the example we are considering, the variable s can be introduced when unifying the abstract object $\{\mathtt{uv}, \mathtt{vw}\}$ with $v/f(s)$. Since $\mathbf{U}^f_{Sh}([\{\mathtt{uv}, \mathtt{vw}\}, \{u, v, w\}], \{s\}, \{v/f(s)\}) = [\{\mathtt{uvs}, \mathtt{vws}\}, \{u, v, w, s\}]$, we still obtain as success substitution $\{\mathtt{xy}, \mathtt{yz}\}$, thus proving that x and z are independent.

5 Related Works

We consider here other improvements to the standard analyses based on `Sharing` and their relationships with our proposal.

Forward/Backward Unification and `PSD`**.** Although the usual goal of sharing analyses is to discover the pairs of variables which may possibly share, `Sharing` is a domain that keeps track of set-sharing information. In [2] the authors propose a new domain, called `PSD`, which is the complete shell [13] of pair sharing w.r.t. `Sharing`. They recognize that, in an abstract object $[S, U]$, some sharing groups in S may be *redundant* as far as pair sharing is concerned. Although our forward unification is more precise than the standard unification, it could be the case that they have the same precision in `PSD`. This would mean that $\mathbf{U}^f_{Sh}([S_1, U_1], U_2, \delta)$ and $\mathbf{U}_{Sh}([S_1, U_1], [\{\{x\} \mid x \in U_2\}, U_2], \delta)$ only differ for redundant sharing groups. However, this is not the case, and Examples 2, 3 and 4 give improvements which are still significant in `PSD`. The same holds for backward unification in Example 5. It would be interesting to examine in details the behavior of our unification operators in the domain `PSD`, since it is not clear whether it is still complete w.r.t. pair-sharing when our specialized operators are used.

Domains with Freeness and Linearity. Although the use of freeness and linearity information has been pursued in several papers (e.g. [20, 14]), optimal operators for these domains have never been developed. Actually, the standard mgu in `SFL` [21, 14, 3], when unifying with a binding $\{x/t\}$ where neither x nor t are linear, does compute all the star unions. In $\mathbf{u}^f_{Sh}$, however, we apply an optimization which is able to avoid some sharing groups (see e.g. Example 4). This optimization could be integrated in a domain which explicitly contains freeness and linearity information. Actually [3] includes some optimizations for the standard abstract unification of `SFL` which are similar to ours, in the case of a binding $\{x/t\}$ with x linear. In addition, [22, 15] propose to remove the check for independence between x and t. We think it should be possible to devise an optimal abstract unification for an enhanced domain including linearity information, by combining these improvements with our results.

Another Optimality Proof. In [5] the authors provide an alternative approach to the analysis of sharing by using set logic programs and ACI1 unification. They define abstract operators which are proved to be correct and optimal, and examine the relationship between set substitutions and `Sharing`, proving that they are essentially isomorphic. However, they do not extend this correspondence to the abstract operators, so that a proof of optimality of $\mathbf{U}'_{Sh}$ w.r.t. $\mathbf{U}'_{\mathrm{Rs}}$ starting from their results should be feasible but it is not immediate. Moreover, since they provide a goal-independent analysis, they do not have different operators for forward and backward unification.

6 Conclusions

We think that this paper makes three major contributions.

- We provide a result of correctness and optimality for the abstract unification in `Sharing`, which corrects the one presented in [7].
- We propose a refined framework with specialized operators for forward and backward unification. We provide the corresponding abstract operators for sharing analysis which are proved to be correct and optimal. The obtained analysis is shown to be strictly more precise than the original one.
- Our definition of $\mathbf{U}^f_{Sh}$ sheds new light on the abstract unification in the presence of freeness and linearity information, suggesting new optimizations which can also be used in more powerful domains such as `SFL`.

Our idea of specialized operators for forward and backward unification is orthogonal to most of other proposals for improving precision and/or efficiency of the analysis. To the best of our knowledge, this is the first work which optimizes the abstract forward unification for sharing analysis by using a specialized operator. In [19] the concrete *unify* operator is essentially our $\mathbf{U}^f_{Rs}$, but the abstract operator is given only for groundness analysis, where specializing the forward unification gives no gain in precision. In other works about goal-dependent analysis, such as [20, 14], the algorithm used for computing the entry substitution is simply the standard unification. This is also the first work where a specialized backward unification operator is proved to be optimal, although matching has been used in several papers [14, 17, 21] to improve backward unification. To the best of our knowledge, all the abstract operators proposed so far for `Sharing` were not optimal. Matching, however, does not remove some imprecisions of goal-dependent versus goal-independent analysis which have been pointed out in [12]. As a future work, we think that our results could be easily generalized for designing optimal unification operators for a domain including linearity information. Moreover, the problem of efficiently implementing the backward unification could be addressed.

References

1. T. Armstrong, K. Marriott, P. Schachte, and H. Søndergaard. Two classes of Boolean functions for dependency analysis. *SoCP*, 31(1):3–45, 1998.
2. R. Bagnara, P. Hill, and E. Zaffanella. Set-sharing is redundant for pair-sharing. *Theoretical Computer Science*, 277(1-2):3–46, 2002.
3. R. Bagnara, E. Zaffanella, and P. M. Hill. Enhanced sharing analysis techniques: A comprehensive evaluation. In *Proc. PPDP*, pages 103–114, 2000.
4. M. Bruynooghe. A practical framework for the abstract interpretation of logic programs. *Journal of Logic Programming*, 10(1/2/3&4):91–124, 1991.
5. M. Codish, V. Lagoon, and F. Bueno. An algebraic approach to sharing analysis of logic programs. *The Journal of Logic Programming*, 41(2):110–149, 2000.
6. M. Comini, G. Levi, and M. C. Meo. A theory of observables for logic programs. *Information and Computation*, 169, 2001.
7. A. Cortesi and G. Filè. Sharing is optimal. *JLP*, 38(3):371–386, 1999.
8. A. Cortesi, G. Filè, and W. W. Winsborough. Optimal groundness analysis using propositional formulas. TR 94/11, Dept. of Mathematics, Univ. of Padova, 1994.
9. A. Cortesi, G. Filé, and W. W. Winsborough. Optimal groundness analysis using propositional logic. *Journal of Logic Programming*, 27(2):137–167, 1996.
10. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. ACM POPL*, pages 269–282, 1979.
11. P. Cousot and R. Cousot. Abstract Interpretation and Applications to Logic Programs. *Journal of Logic Programming*, 13(2 & 3):103–179, 1992.
12. M. G. de La Banda, K. Marriott, P. Stuckey, and H. Søndergaard. Differential methods in logic program analysis. *JLP*, 37(1):1–37, Apr. 1998.
13. R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretations complete. *Journal of the ACM*, 47(2):361–416, 2000. ISSN 1084-6654.
14. W. Hans and S. Winkler. Aliasing and groundness analysis of logic programs through abstract interpretation and its safety. Technical Report 92–27, Technical University of Aachen (RWTH Aachen), 1992.
15. J. Howe and A. King. Three Optimisations for Sharing. TR 11-01, Computing Laboratory, Univ. of Kent at Canterbury, August 2001. To appear in TPLP.
16. D. Jacobs and A. Langen. Static Analysis of Logic Programs for Independent AND Parallelism. *Journal of Logic Programming*, 13(2 & 3):291–314, 1992.
17. A. King and M. Longley. Abstract matching can improve on abstract unification. TR 4-95*, Computing Laboratory, Univ. of Kent, Canterbury, UK, 1995.
18. A. Langen. *Static Analysis for Independent And-parallelism in Logic Programs.* PhD thesis, University of Southern California, Los Angeles, California, 1990.
19. K. Marriott, H. Søndergaard, and N. D. Jones. Denotational abstract interpretation of logic programs. *ACM TOPLAS*, 16(3):607–648, May 1994.
20. K. Muthukumar and M. Hermenegildo. Combined Determination of Sharing and Freeness of Program Variables through Abstract Interpretation. In *Proc. ICLP*, pp. 49–63, 1991.
21. K. Muthukumar and M. V. Hermenegildo. Compile-time derivation of variable dependency using abstract interpretation. *JLP*, 13(2&3):315–347, 1992.
22. E. Zaffanella. *Correctness, Precision and Efficiency in the Sharing Analysis of Real Logic Languages.* PhD thesis, School of Computing, University of Leeds, U.K., 2001. Available at http://www.cs.unipr.it/~zaffanella/.