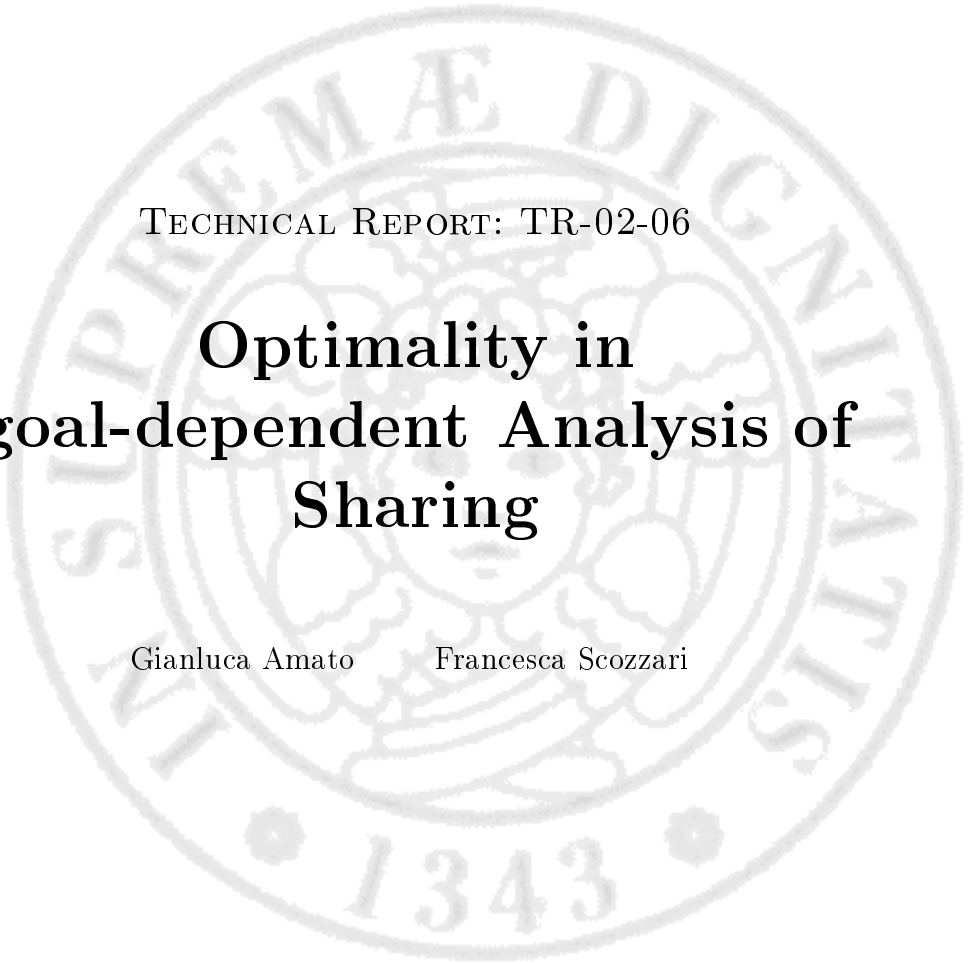
Università di Pisa
Dipartimento di Informatica



Technical Report: TR-02-06

Optimality in goal-dependent Analysis of Sharing

Gianluca Amato Francesca Scozzari

May 7, 2002

address: Corso Italia 40, 56125 Pisa, Italy. tel: +39 050 2212700 fax: +39 050 2212726

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` [19, 9, 1] 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, 22]) , 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. to the concrete operator, the common solution is to design a new correct (and possibly optimal) abstract operator. But, as we show with many examples, we think that incorrectness is mainly due to a non-standard 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.

Futhermore, 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 which is 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 $\alpha(c) \leq a \iff c \leq \gamma(a)$. 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 $\mathcal{V}$. Given a term t, by $vars(t)$ we denote the set of variables occurring in t and with $uvars(t)$ the subset of $vars(t)$ whose elements only appear once in t. We will abuse notation and apply $vars$ and $uvars$ to any syntactic object with the obvious meaning (e.g. , $uvars(t(x, y) = t(y, z)) = \{x, y, z\}$). With ϵ we denote the empty substitution, while $\{x_1/t_1, \ldots, x_n/t_n\}$ denotes a substitution θ with $\theta(x_i) = t_i \neq x_i$. We denote by $vars(\theta)$ the set $\mathrm{dom}(\theta) \cup \mathrm{range}(\theta)$ and, given $U \in \wp_f(\mathcal{V})$, we denote by $\theta|_U$ the 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, let us briefly recall the definitions of the concrete domain and operators used in [7].

3.1 Concrete Domain and Operations

The concrete domain is $\texttt{Rsub} = (\wp(\mathit{Subst}) \times \wp_f(\mathcal{V})) \cup \{\top_{\mathrm{Rs}}, \bot_{\mathrm{Rs}}\}$. `Rsub` is partially ordered as follows: $\top_{\mathrm{Rs}}$ is the top element, $\bot_{\mathrm{Rs}}$ is the bottom element and $\langle \Sigma_1, U_1 \rangle \sqsubseteq_{\mathrm{Rs}} \langle \Sigma_2, U_2 \rangle$ if and only if $U_1 = U_2$ and $\Sigma_1 \subseteq \Sigma_2$. `Rsub` is a complete lattice w.r.t. $\sqsubseteq_{\mathrm{Rs}}$. The least upper bound of `Rsub` is denoted by $\sqcup_{\mathrm{Rs}}$. The concrete projection $\pi_{\mathrm{Rs}} : \texttt{Rsub} \times \wp_f(\mathcal{V}) \to \texttt{Rsub}$ is defined as follows:

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

The concrete unification is $\mathbf{U}_{Rs} : \texttt{Rsub} \times \texttt{Rsub} \times \mathit{Subst} \to \texttt{Rsub}$ such that:

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

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 input 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 operator in the abstract domain will only be defined under these conditions.

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

$$A^* = \{ \bigcup T \mid \emptyset \neq T \subseteq A \}$$

– 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* $\mathbf{bin} : \wp(\wp_f(\mathcal{V})) \times \wp(\wp_f(\mathcal{V})) \to \wp(\wp_f(\mathcal{V}))$:

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

Fig. 1. Auxiliary operators

3.2 Abstract Domain and Operations

Now let us briefly recall the definition of the abstract domain `Sharing` [16, 7]:

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

The abstraction function $\alpha_{\mathrm{Sh}} : \texttt{Rsub} \to \texttt{Sharing}$ is defined as follows:

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

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. `xyz` represents $\{x, y, z\}$). The abstract operators do behave exactly as the concrete ones on $\top_{Sh}$ and $\bot_{Sh}$, while the other cases are the following:

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

where $\mathbf{u}_{Sh} : \wp(\wp_f(\mathcal{V})) \times \mathit{Subst} \to \wp(\wp_f(\mathcal{V}))$ is defined by induction, by using the auxiliary operators in Fig. 1.

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 [20] and the domain *ESubst* in [16]. It is possible to formalize this idea by defining a preorder $\preceq$ over `Rsub` such that:

$$[\Sigma, U] \preceq [\Sigma', U] \iff \forall \theta \in \Sigma. \exists \theta' \in \Sigma', \rho \in Ren. \forall x \in U. \theta(x) = \rho(\theta'(x)) \quad (1)$$

and the corresponding equivalence relation $\sim$. All the elements in an equivalence class modulo $\sim$ are essentially the same object as long as the actual name of existential variables is ignored.

Proposition 1. *$\sim$ is a congruence w.r.t. π_{Rs} and $\sqcup_{\mathrm{Rs}}$, while α_{Sh} assume the same value over all the elements of the same equivalence class.*

However, the same does not happen with $\mathbf{U}_{Rs}$, since the concrete operator used for unification 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 $vars(\sigma_1) \cap vars(\sigma_2) = \emptyset$. This unification can lead to counterintuitive results. For instance, consider the following concrete unification:

$$\mathbf{U}_{Rs}([\{\{x/y\}\}, \{x\}], [\{\epsilon\}, \{y\}], \epsilon) = [\{\{x/y\}\}, \{x, y\}] \quad (2)$$

Being $vars(\epsilon) = \emptyset$, 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 it was. On the contrary,

$$\mathbf{U}_{Rs}([\{\{x/z\}\}, \{x\}], [\{\epsilon\}, \{y\}], \epsilon) = [\{\{x/z\}\}, \{x, y\}] \quad (3)$$

and $[\{\{x/y\}\}, \{x, y\}] \not\sim [\{\{x/z\}\}, \{x, y\}]$, and this also causes the incorrectness of $\mathbf{U}_{Sh}$. Actually, consider the equation 2. If we compute:

$$\begin{array}{l} \alpha_{Sh}([\{\{x/y\}\}, \{x\}]) = [\{\mathtt{x}\}, \{x\}] \\ \alpha_{Sh}([\{\epsilon\}, \{y\}] \qquad\quad = [\{\mathtt{y}\}, \{y\}] \end{array} \quad (4)$$

by using the abstract unification operator $\mathbf{U}_{Sh}$ we have:

$$\begin{array}{c} \mathbf{U}_{Sh}(\alpha_{Sh}([\{\{x/y\}\}, \{x\}]), \alpha_{Sh}([\{\epsilon\}, \{y\}]), \epsilon) \\ = \mathbf{U}_{Sh}([\{\mathtt{x}\}, \{x\}], [\{\mathtt{y}\}, \{y\}], \epsilon) = [\{\mathtt{x}, \mathtt{y}\}, \{x, y\}] \ . \end{array} \quad (5)$$

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

$$\alpha_{Sh}([\{\{x/y\}\}, \{x, y\}]) = [\{\mathtt{xy}\}, \{x, y\}] \not\sqsubseteq_{Sh} [\{\mathtt{x}, \mathtt{y}\}, \{x, y\}] \ . \quad (6)$$

This counterexample proves that the abstract unification operator is not correct w.r.t. the concrete unification. Note, however, that:

$$\alpha_{Sh}([\{\{x/z\}\}, \{x, y\}]) = [\{\mathtt{x}, \mathtt{y}\}, \{x, y\}]). \quad (7)$$

For the curious reader, if we look at the proof of Theorem 6.3 in [7] it appears that the problem is in the base case of the inductive reasoning. Here, it is stated that given $[A_1, U_1]$ and $[A_2, U_2]$ in `Sharing` with $U_1 \cap U_2 = \emptyset$, $\sigma_i \in \gamma_{Sh}([A_i, U_i])$ for $i \in \{1, 2\}$ with $vars(\sigma_1) \cap vars(\sigma_2) = \emptyset$, then $[\{\rho_0\}, U_0] \in \gamma_{\mathrm{Sh}}([R_0, U_0])$ where $\rho_0 = \sigma_1 \cup \sigma_2$, $U_0 = U_1 \cup U_2$ and $R_0 = A_1 \cup A_2$. However, the substitutions $\sigma_1 = \{x/y\} \in \gamma_{Sh}([\{\mathtt{x}\}, \{x\}])$ and $\sigma_2 = \epsilon \in \gamma_{Sh}[\{\mathtt{y}\}, \{y\}]$ of the previous example make the statement false.

3.4 Toward a Result of Correctness

In the previous section we have shown that the abstract unification operator $\mathbf{U}_{Sh}$ proposed in [7] is not correct w.r.t. the concrete one $\mathbf{U}_{Rs}$. Of course, it would be possible to design a correct and optimal abstract unification for $\mathbf{U}_{Rs}$, but we strongly believe that the real anomaly is in the choice of the concrete operator. Any meaningful semantics for logic programs should perform some kind of renaming in order to avoid variables clashes such as that in Eq. (2).

There are several ways to approach the problem. We could define a new concrete operator $\mathbf{U}^*_{\mathrm{Rs}}$ enforcing a stronger applicability condition in such a way to ensure correctness and optimality of $\mathbf{U}_{Sh}$ w.r.t. $\mathbf{U}^*_{\mathrm{Rs}}$. A natural choice would be the following one.

$$\begin{aligned}\mathbf{U}^*_{\mathrm{Rs}}([\Sigma_1, U_1], [\Sigma_2, U_2], \delta) = [\{\mathrm{mgu}(\sigma_1, \sigma_2, \delta) \mid \sigma_1 \in \Sigma_1, \sigma_2 \in \Sigma_2,\\ (U_1 \cup vars(\sigma_1)) \cap (U_2 \cup vars(\sigma_2)) = \emptyset\}, U_1 \cup U_2] \quad (8)\end{aligned}$$

This operator differs from the previous one since the original condition $vars(\sigma_1) \cap vars(\sigma_2) = \emptyset$ is replaced by the stronger condition $(U_1 \cup vars(\sigma_1)) \cap (U_2 \cup vars(\sigma_2)) = \emptyset$ which chooses σ_1 – resp. σ_2 – renamed apart from U_2 – resp. U_1 – and each other. Actually, if we replace $\mathbf{U}_{Rs}$ with $\mathbf{U}^*_{\mathrm{Rs}}$ in the Theorem 6.3 of [7], the base case works, and therefore [7] correctly proves that $\mathbf{U}_{Sh}$ is correct and optimal w.r.t. $\mathbf{U}^*_{\mathrm{Rs}}$. However, this causes another problem. It holds that:

$$\mathbf{U}^*_{\mathrm{Rs}}([\{x/z\}, \{x\}], [\{y/z\}, \{y\}], \epsilon) = \emptyset \quad (9)$$

but the intuitive result which comes from the unification between ex-equations would be $[\{x/z_1, y/z_2\}, \{x, y\}]$ where $z_1 \neq z_2$. So we must be sure that we have enough variants in Σ_1 and Σ_2 for each substitution and this probably requires some change to the other operators and to the semantics of the language.

In order to solve this problem, we could just perform a renaming before doing the unification, instead of choosing substitutions which are renamed apart. Actually, in [9] the authors recognize that "in a typical semantic construction, renaming is performed" and in [8] actually define a more complex unification with renamings. However, the corresponding abstract operators are given only for groundness analysis. The concrete unification $\mathbf{U}'_{\mathrm{Rs}} : \mathtt{Rsub} \times \mathtt{Rsub} \times \mathsf{Atoms} \times \mathsf{Atoms} \to \mathtt{Rsub}$ defined in [8] is the following:

$$\begin{aligned}&\mathbf{U}'_{\mathrm{Rs}}([\Sigma_1, U_1], [\Sigma_2, U_2], A_1, A_2) =\\ &\quad = \mathbf{U}_{Rs}([\rho_1(\Sigma_1), \rho_1(U_1)], [\rho_2(\Sigma_2), U_2], \mathrm{mgu}(\rho_1(A_1) = A_2))\end{aligned} \quad (10)$$

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 $\rho_1 : \mathcal{V} \to V_1$ and $\rho_2 : \mathcal{V} \to V_2$ 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. The definition of $\mathbf{U}'_{\mathrm{Rs}}$ allows us to extend the Prop. 1 to the unification operator.

Proposition 2. $\sim$ *is a congruence for* $\mathbf{U}'_{\mathrm{Rs}}$.

We can now define the abstract unification $\mathbf{U}'_{Sh}$ corresponding to $\mathbf{U}'_{\mathrm{Rs}}$ as follows:

$$\begin{aligned}\mathbf{U}'_{Sh}([S_1, U_1], [S_2, U_2], A_1, A_2) \\ = \mathbf{U}_{Sh}([\rho_1(S_1), \rho_1(U_1)], [S_2, U_2], \mathrm{mgu}(\rho_1(A_1) = A_2))\end{aligned} \tag{11}$$

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. It is easy to show that:

$$\begin{aligned}\mathbf{U}'_{\mathrm{Rs}}([\Sigma_1, U_1], [\Sigma_2, U_2], A_1, A_2) \\ = \mathbf{U}^*_{\mathrm{Rs}}([\rho_1(\Sigma_1), \rho_1(U_1)], [\rho_2(\Sigma_2), U_2], \mathrm{mgu}(\rho_1(A_1) = A_2))\end{aligned} \tag{12}$$

Since $\mathbf{U}_{Sh}$ is correct and optimal w.r.t. $\mathbf{U}^*_{\mathrm{Rs}}$, it turns out 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. $\mathbf{U}'_{Sh}$ *is correct and optimal w.r.t.* $\mathbf{U}'_{\mathrm{Rs}}$.

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

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. On the contrary, a semantics using $\pi_{\mathrm{Rs}}, \sqcup_{\mathrm{Rs}}$ and $\mathbf{U}'_{\mathrm{Rs}}$ has been defined in [8]. We will briefly recall the relevant definitions.

4.1 Semantics

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 $\mathcal{X}$. A *denotation* is an element in the set of monotonic maps:

$$\mathcal{D}en = \mathsf{Atoms} \to \mathcal{X} \to \mathcal{X} \quad . \tag{13}$$

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

Fig. 2. Semantic functions.

We have the following semantic functions:

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

whose definitions are given in Fig. 2 by means of the following operators:

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

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

$$\mathbf{U}_{\mathcal{X}} = \mathbf{U}'_{\mathrm{Rs}}$$
$$\pi_{\mathcal{X}}(\langle \Sigma_1, U_1\rangle, \langle \Sigma_2, U_2\rangle) = \pi_{\mathrm{Rs}}([\Sigma_1, U_1], U_2)$$
$$id_{\mathcal{X}}(cl) = [\{\epsilon\}, vars(cl)]$$

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

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 $\langle \alpha'_{Sh}, \gamma'_{Sh}\rangle$: $\mathsf{Atoms} \to \mathtt{Rsub} \to \mathtt{Rsub} \leftrightharpoons \mathsf{Atoms} \to \mathtt{Sharing} \to \mathtt{Sharing}$.

4.2 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 $[\{\epsilon\}, vars(H \leftarrow B)]$. Therefore, a specialized version of $\mathbf{U}'_{Sh}$ could improve the precision of the analysis.

Example 1. Assume, without loss of generality, that $(\rho_1, \rho_2) = Apart(U_2)$ and ρ_1 restricted to $\{x, y, z\}$ is the identity. Then,

$$\mathbf{U}'_{Sh}([\{\mathtt{xy}, \mathtt{yz}\}, \{x, y, z\}], id_{Sh}[\![p(u, v, w) \leftarrow \lambda]\!], p(x, y, z), p(u, v, w)) = [\{\mathtt{xyuv}, \mathtt{yzvw}, \mathtt{xyzuvw}\}, \{x, y, z, u, v, w\}] \ . \quad (14)$$

However, since we know that u, v and w are free in $id_{\mathrm{Rs}}[\![p(u, v, w) \leftarrow \lambda]\!]$, following [14] when considering the binding y/v in $\mathbf{u}_{Sh}$ we can avoid to compute the star unions, 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 $[\{\mathtt{uv}, \mathtt{vw}, \mathtt{uvw}\}, \{u, v, w\}]$ in the first case and $[\{\mathtt{uv}, \mathtt{vw}\}, \{u, v, w\}]$ in the latter, with an obvious gain of precision.

Example 2. Let us consider the following unification.

$$\mathbf{U}'_{Sh}([\{\mathtt{xy}, \mathtt{xz}\}, \{x, y, z\}], id_{Sh}[\![p(t(u, v), h, k) \leftarrow]\!], p(x, y, z), p(t(u, v), h, k)) = [\mathbf{bin}(\{\mathtt{xyh}, \mathtt{xzk}, \mathtt{xyzhk}\}, \{\mathtt{u}, \mathtt{v}, \mathtt{uv}\}), \{x, y, z, h, k, u\}] \ . \quad (15)$$

Here, considering that 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 $[\{\mathbf{bin}(\{\mathtt{xyh}, \mathtt{xzk}\}, \{\mathtt{u}, \mathtt{v}, \mathtt{uv}\}), \{x, y, z, h, k, u\}]$. 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 $[\{\epsilon\}, vars(H \leftarrow B)]$. Note that we are not proposing of embedding freeness and linearity information inside the domain, but only to use all the information coming from the syntax of clauses.

4.3 The Refined Forward Unification

Our first step is to change the definition of the semantic function for clauses in the following way:

$$\begin{aligned} &\mathcal{C}[\![H \leftarrow B]\!]dAx = \pi_{\mathcal{X}}(\mathbf{U}_{\mathcal{X}}(x', x, H, A), x) \\ &\quad \text{where } x' = \mathcal{B}[\![B]\!]d(\pi_{\mathcal{X}}(\mathbf{U}^f_{\mathcal{X}}(x, H \leftarrow B, A, H), id[\![H \leftarrow B]\!])) \end{aligned} \quad (16)$$

by using a new operator $\mathbf{U}^f_{\mathcal{X}} : \mathcal{X} \times \mathsf{Clauses} \times \mathsf{Atoms} \times \mathsf{Atoms} \to \mathcal{X}$. For the concrete semantics, we instantiate $\mathbf{U}^f_{\mathcal{X}}$ with $\mathbf{U'}^f_{\mathrm{Rs}}$ defined as:

$$\mathbf{U'}^f_{\mathrm{Rs}}([\Sigma, U], cl, A_1, A_2) = \mathbf{U}'_{\mathrm{Rs}}([\Sigma, U], id_{\mathrm{Rs}}[\![cl]\!], A_1, A_2) \ , \tag{17}$$

so that nothing changes. However when we move to the abstract domain `Sharing`, directly abstracting $\mathbf{U'}^f_{\mathrm{Rs}}$ gives more precise results that 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. 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:*

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

if $x \in U$

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

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

provided $U_1 \cap U_2 = \emptyset$ *and* $\mathit{vars}(\theta) \subseteq U_1 \cup U_2$. *For all the other cases, the result of* $\mathbf{U}^f_{Sh}$ *is* $\bot_{Sh}$.

We can now define the forward abstract unification with renamings $\mathbf{U'}^f_{Sh} : \mathsf{Sharing} \times \mathsf{Clauses} \times \mathsf{Atoms} \times \mathsf{Atoms} \to \mathtt{Sharing}$ as follows:

$$\mathbf{U'}^f_{Sh}([S_1, U_1], cl, A_1, A_2) = \mathbf{U}^f_{Sh}([\rho_1(S_1), \rho_1(U_1)], \mathit{vars}(cl), \mathrm{mgu}(\rho_1(A_1) = A_2)) \tag{18}$$

where $(\rho_1, \rho_2) = Apart(vars(cl))$ provided $vars(A_1) \subseteq U_1$ and $vars(A_2) \subseteq vars(cl)$, $\bot_{Sh}$ otherwise.

Using $\mathbf{U'}^f_{Sh}$ instead of $\mathbf{U}'_{Sh}$ gives the improvements in precision we have discussed in the Examples 1 and 2. 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 influenced by the proof of the analogous theorem for $\mathbf{U}_{Sh}$ and $\mathbf{U}^*_{\mathrm{Rs}}$ 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 once for all before the actual analysis begins, since they depend from the syntax of the program alone. 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.

It is worth noting that this operator introduces some new optimizations which, up to our knowledge, are not used even in more complex domains for sharing analysis which include linearity and freeness information. The next example shows one of these possible optimizations.

Example 3. Let us consider the following unification.

$$\mathbf{U}'_{Sh}([\{\mathtt{xw}, \mathtt{xz}, \mathtt{yw}, \mathtt{yz}\}, \{x, y, w, z\}], id_{Sh}[\![p(f(u, h), f(u, k), s, t) \leftarrow]\!], \\ p(x, y, w, z), p(f(u, h), f(u, k), s, t)) \ . \quad (19)$$

By applying the optimizations suggested from the unification algorithm in presence of linearity and freeness information [14], we obtain either

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

or

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

respectively taking into account the binding $x/f(u, h)$ before or after the binding $y/f(u, k)$. When we project over $\{u, h, k, s, t\}$, we obtain either the sharing group `stk` or `sth`.

The difference between the two computations is due to the fact that when we consider the first of the two bindings, assume it is $x/f(u, h)$, the term $f(u, h)$ is linear and independent from x. However, when the second binding is considered,

this does not hold anymore, since we are not sure that u is linear and independent from y. However, 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. Therefore, we obtain

$$\{\mathtt{ywsk}, \mathtt{yztk}\} \cup \mathbf{bin}(\{\mathtt{yws}, \mathtt{yzt}\}^*, \mathbf{bin}(\{\mathtt{xws}, \mathtt{xzt}\}, \{\mathtt{u}, \mathtt{uh}\})^*)$$
$$\cup\, \mathbf{bin}(\mathbf{bin}(\{\mathtt{yws}, \mathtt{yzt}\}^*, \mathbf{bin}(\{\mathtt{xws}, \mathtt{xzt}\}, \{\mathtt{u}, \mathtt{uh}\})^*), \{k\}) \cup \{\mathtt{xwsh}, \mathtt{xzth}\} \quad (20)$$

and when we project over $\{u, h, k, s, t\}$, both the sharing groups `sth` and `stk` do not appear. The result does not change by permuting the order of the bindings.

4.4 Matching and Backward Unification

In this section we study some optimizations for the computation of the exit substitution. When we compute $\mathbf{U}^*_{\mathrm{Rs}}(x', x, \delta)$, 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 [20, Section 5.5]. Following this idea, we define a new operator for concrete backward unification given as:

$$\mathbf{U}^b_{\mathrm{Rs}}([\Sigma_1, U_1], [\Sigma_2, U_2], \delta) = [\{\mathrm{mgu}(\sigma_1, \sigma_2, \delta) \mid \sigma_1 \in \Sigma_1, \sigma_2 \in \Sigma_2,$$
$$(vars(\sigma_1) \cup U_1) \cap (vars(\sigma_2) \cup U_2) = \emptyset, \exists \theta.\, \sigma_2 \leq (\theta \circ \mathrm{mgu}(\sigma_1, \delta))|_{U_2}\}, U_1 \cup U_2] \ .$$

We also define the version with renamings $\mathbf{U'}^b_{\mathrm{Rs}}$ as it has been done for $\mathbf{U}'_{\mathrm{Rs}}$. It turns out that $\sim$ is a congruence for $\mathbf{U}'_{\mathrm{Rs}}$. We instantiate $\mathbf{U}_{\mathcal{X}}$ with $\mathbf{U'}^b_{\mathrm{Rs}}$ in the semantic definition of Eq. (16).

The idea of using a refined operator for computing the exit substitution is not new. For example 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 [22] use a refined operator for backward unification. However, both their operators are not optimal and quite inaccurate. 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 operation 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*

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

As before, $\mathbf{U'}^b_{Sh}$ is obtained from $\mathbf{U}^b_{Sh}$ by introducing the necessary renamings.

Example 4. 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 path. 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

$$\mathbf{U'}^b_{Sh}([S_1, U_1], [S_2, U_2], \delta) = [\{\mathtt{xyuv}, \mathtt{yzvw}\}, U_1 \cup U_2] \ .$$

Now, 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.

Theorem 3. $\mathbf{U'}^b_{Sh}$ *is correct and optimal w.r.t.* $\mathbf{U'}^b_{\mathrm{Rs}}$.

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 5. Actually, the example is trivial and consists of a program with just one clause `p(u,v,w)` $\leftarrow$. Consider the goal $p(x, y, z)$ with calling substitution $\{\mathtt{xy}, \mathtt{yz}\}$. Using our abstract operators, we obtain the entry substitution $\{\mathtt{uv}, \mathtt{vw}\}$ (see Example 1) and the success substitution $\{\mathtt{xy}, \mathtt{yz}\}$ (see Example 4). Therefore we prove that x and z are independent. Note that 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`.

5 Related Works

We consider here other improvements to the standard analyses based on `Sharing` and their relationships with our proposal. It turns out that 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. Furthermore, the definition of $\mathbf{U}^f_{Sh}$ sheds new light on the abstract unification in the presence of freeness and linearity information.

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, since they do not provide any information as far as pair sharing is concerned. Given $S \in \wp(\wp_f(\mathcal{V}))$ and $B \in \wp_f(\mathcal{V})$, B is *redundant* for S when $B \times B = \bigcup\{B' \times B' \mid B' \in S, B' \subsetneq B\}$. Now, although our forward unification is more precise than the standard unification, it can 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 1, 2 and 3 give improvements which are still significant in `PSD`. The same holds for backward unification in Example 4. It would be interesting to examine more 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. [21, 14]), optimal operators for these domains have never been developed. Actually, the standard mgu in `SFL` [22, 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 3). 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, [23, 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 three are the major contributions of this paper.

- We provide a result of 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.

– We suggest a new idea for treating freeness and linearity information which can also be used in more powerful domains such as `SFL`.

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. Actually, in [20] 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 [21, 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, 22] 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. Boolean functions for dependency analysis: Algebraic properties and efficient representation. In B. Le Charlier, editor, *Proc. Static Analysis Symposium, SAS'94*, volume 864 of *Lecture Notes in Computer Science*, pages 266–280. Springer-Verlag, 1994.
2. R. Bagnara, P. Hill, and E. Zaffanella. Set-sharing is redundant for pair-sharing. *Theoretical Computer Science*, 2002. To appear.
3. R. Bagnara, E. Zaffanella, and P. M. Hill. Enhanced sharing analysis techniques: A comprehensive evaluation. In M. Gabbrielli and F. Pfenning, editors, *Proceedings of the 2nd International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming*, pages 103–114, Montreal, Canada, 2000. ACM Press.
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. In *Static Analysis Symposium*, pages 68–82, 1997.
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. *Journal of Logic Programming*, 38(3):371–386, 1999.
8. A. Cortesi, G. Filè, and W. W. Winsborough. Optimal groundness analysis using propositional formulas. Technical Report 94/11, Department of Mathematics, University 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. Sixth ACM Symp. Principles of Programming Languages (POPL '79)*, pages 269–282, New York, 1979. ACM Press.

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. *Journal of Logic Programming*, 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. Technical Report 11-01, Computing Laboratory, University of Kent at Canterbury, August 2001. To appear in Theory and Practice of Logic Programming.
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. Technical Report 4-95*, University of Kent, Computing Laboratory, University of Kent, Canterbury, UK, March 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 and H. Søndergaard. Abstract Interpretation of Logic Programs: the Denotational Approach. In A. Bossi, editor, *Proc. Fifth Italian Conference on Logic Programming*, pages 399–425, 1990.
20. K. Marriott, H. Søndergaard, and N. D. Jones. Denotational abstract interpretation of logic programs. *ACM Transactions on Programming Languages and Systems*, 16(3):607–648, May 1994.
21. K. Muthukumar and M. Hermenegildo. Combined Determination of Sharing and Freeness of Program Variables through Abstract Interpretation. In K. Furukawa, editor, *Proceedings of the 8th International Conference on Logic Programming*, pages 49–63, Paris, 1991. The MIT Press.
22. K. Muthukumar and M. V. Hermenegildo. Compile-time derivation of variable dependency using abstract interpretation. *Journal of Logic Programming*, 13(2&3):315–347, 1992.
23. E. Zaffanella. *Correctness, Precision and Efficiency in the Sharing Analysis of Real Logic Languages.* PhD thesis, School of Computing, University of Leeds, Leeds, U.K., 2001. Available at http://www.cs.unipr.it/~zaffanella/.