Article:
Hill, P.M., Zaffanella, E. and Bagnara, R. (2004) A correct, precise and efficient integration of set-sharing, freeness and linearity for the analysis of finite and rational tree languages. Theory and Practice of Logic Programming, 4 (3). pp. 289-323. ISSN 1471-0684

https://doi.org/10.1017/S1471068403001868

TLP **4** (3): 289–323, 2004. © 2004 Cambridge University Press

DOI: 10.1017/S1471068403001868 Printed in the United Kingdom

*A correct, precise and efficient integration of set-sharing, freeness and linearity for the analysis of finite and rational tree languages**

PATRICIA M. HILL
School of Computing, University of Leeds, Leeds, UK
(*e-mail:* `hill@comp.leeds.ac.uk`)

ENEA ZAFFANELLA, ROBERTO BAGNARA
Department of Mathematics, University of Parma, Parma, Italy
(*e-mail:* `{zaffanella,bagnara}@cs.unipr.it`)

Abstract

It is well known that freeness and linearity information positively interact with aliasing information, allowing both the precision and the efficiency of the sharing analysis of logic programs to be improved. In this paper, we present a novel combination of set-sharing with freeness and linearity information, which is characterized by an improved abstract unification operator. We provide a new abstraction function and prove the correctness of the analysis for both the finite tree and the rational tree cases. Moreover, we show that the same notion of redundant information as identified in Bagnara *et al.* (2000) and Zaffanella *et al.* (2002) also applies to this abstract domain combination: this allows for the implementation of an abstract unification operator running in polynomial time and achieving the same precision on all the considered observable properties.

KEYWORDS: abstract interpretation, logic programming, abstract unification, rational trees, set-sharing, freeness, linearity

1 Introduction

Even though the set-sharing domain is, in a sense, remarkably precise, more precision is attainable by combining it with other domains. In particular, freeness and linearity information has received much attention by the literature on sharing analysis (recall that a variable is said to be free if it is not bound to a non-variable term; it is linear if it is not bound to a term containing multiple occurrences of another variable).

* This work has been funded by MURST projects “Automatic Program Certification by Abstract Interpretation”, “Abstract Interpretation, type systems and control-flow analysis”, and “Automatic Aggregate- and Number-Reasoning for Computing: from Decision Algorithms to Constraint Programming with Multisets, Sets, and Maps”; by the Integrated Action Italy-Spain “Advanced Development Environments for Logic Programs”; by the University of Parma’s FIL scientific research project (ex 60%) “Pure and applied mathematics”; and by the UK’s Engineering and Physical Sciences Research Council (EPSRC) under grant M05645.

As argued informally by Søndergaard (1986), the mutual interaction between linearity and aliasing information can improve the accuracy of a sharing analysis. This observation has been formally applied in Codish *et al.* (1991) to the specification of the abstract mgu operator for the domain ASub. In his PhD thesis, Langen (1990) proposed a similar integration with linearity, but for the set-sharing domain. He has also shown how the aliasing information allows to compute freeness with a good degree of accuracy (however, freeness information was not exploited to improve aliasing). King (1994) has also shown how a more refined tracking of linearity allows for further precision improvements.

The synergy attainable from a bi-directional interaction between aliasing and freeness information was initially pointed out by Muthukumar and Hermenegildo (1991, 1992). Since then, several authors considered the integration of set-sharing with freeness, sometimes also including additional explicit structural information (Codish *et al.*, 1993; Codish *et al.*, 1996; Filé, 1994 King and Soper, 1994).

Building on the results obtained in Søndergaard (1986), Codish *et al.* (1991) and Muthukumar and Hermenegildo (1991), but independently from Langen (1990), Hans and Winkler (1992) proposed a combined integration of freeness and linearity information with set-sharing. Similar combinations have been proposed (Bruynooghe and Codish, 1993; Bruynooghe *et al.*, 1994a, 1994b). From a more pragmatic point of view, Codish *et al.* (1993, 1996) integrate the information captured by the domains of Søndergaard (1986) and Muthukumar and Hermenegildo (1991) by performing the analysis with both domains at the same time, exchanging information between the two components at each step.

Most of the above proposals differ in the carrier of the underlying abstract domain. Even when considering the simplest domain combinations where explicit structural information is ignored, there is no general consensus on the specification of the abstract unification procedure. From a theoretical point of view, once the abstract domain has been related to the concrete one by means of a Galois connection, it is always possible to specify the best correct approximation of each operator of the concrete semantics. However, empirical observations suggest that sub-optimal operators are likely to result in better complexity/precision trade-offs (Bagnara *et al.*, 2000). As a consequence, it is almost impossible to identify "the right combination" of variable aliasing with freeness and linearity information, at least when practical issues, such as the complexity of the abstract unification procedure, are taken into account.

Given this state of affairs, we will now consider a domain combination whose carrier is essentially the same as specified by Langen (1990) and Hans and Winkler (1992). (The same domain combination was also considered by Bruynooghe *et al.* (1994a, 1994b), but with the addition of compoundness and explicit structural information.) The novelty of our proposal lies in the specification of an improved abstract unification procedure, better exploiting the interaction between sharing and linearity. As a matter of fact, we provide an example showing that all previous approaches to the combination of set-sharing with freeness and linearity are not uniformly more precise than the analysis based on the ASub domain (Codish *et al.*, 1991; King, 2000; Søndergaard, 1986), whereas such a property is enjoyed by our proposal.

By extending the results of Hill *et al.* (2002) to this combination, we provide a new abstraction function that can be applied to any logic language computing on domains of syntactic structures, with or without the occurs-check; by using this abstraction function, we also prove the correctness of the new abstract unification procedure. Moreover, we show that the same notion of redundant information as identified in Bagnara *et al.* (2002) and Zaffanella *et al.* (2002) also applies to this abstract domain combination. As a consequence, it is possible to implement an algorithm for abstract unification running in polynomial time and still obtain the same precision on all the considered observables: groundness, independence, freeness and linearity.

This paper is based on Zaffanella (2001, Chapter 6), the PhD thesis of the second author. In section 2, we define some notation and recall the basic concepts used later in the paper. In section 3, we present the domain *SFL* that integrates set-sharing, freeness and linearity. In section 4, we show that *SFL* is uniformly more precise than the domain ASub, whereas all the previous proposals for a domain integrating set-sharing and linearity fail to satisfy such a property. In section 5, we show that the domain *SFL* can be simplified by removing some redundant information. In section 6, we provide an experimental evaluation using the China analyzer (Bagnara, 1997). In section 7, we discuss some related work. Section 8 concludes with some final remarks.

The proofs of the results stated here are not included, but all of them are available in an extended version of this paper (Hill *et al.*, 2003).

2 Preliminaries

For a set S, $\wp(S)$ is the powerset of S. The cardinality of S is denoted by $\#S$ and the empty set is denoted by $\emptyset$. The notation $\wp_{\mathrm{f}}(S)$ stands for the set of all the *finite* subsets of S, while the notation $S \subseteq_{\mathrm{f}} T$ stands for $S \in \wp_{\mathrm{f}}(T)$. The set of all finite sequences of elements of S is denoted by S^*, the empty sequence by ϵ, and the concatenation of $s_1, s_2 \in S^*$ is denoted by $s_1 . s_2$.

2.1 Terms and trees

Let *Sig* denote a possibly infinite set of function symbols, ranked over the set of natural numbers. Let *Vars* denote a denumerable set of variables, disjoint from *Sig*. Then *Terms* denotes the free algebra of all (possibly infinite) terms in the signature *Sig* having variables in *Vars*. Thus a term can be seen as an ordered labeled tree, possibly having some infinite paths and possibly containing variables: every inner node is labeled with a function symbol in *Sig* with a rank matching the number of the node's immediate descendants, whereas every leaf is labeled by either a variable in *Vars* or a function symbol in *Sig* having rank 0 (a constant). It is assumed that *Sig* contains at least two distinct function symbols, with one of them having rank 0.

If $t \in \mathit{Terms}$ then $\mathrm{vars}(t)$ and $\mathrm{mvars}(t)$ denote the set and the multiset of variables occurring in t, respectively. We will also write $\mathrm{vars}(o)$ to denote the set of variables occurring in an arbitrary syntactic object o.

Suppose $s, t \in \mathit{Terms}$: s and t are *independent* if $\mathrm{vars}(s) \cap \mathrm{vars}(t) = \emptyset$; we say that variable y *occurs linearly in* t, more briefly written using the predication $\mathrm{occ\_lin}(y, t)$, if y occurs exactly once in $\mathrm{mvars}(t)$; t is said to be *ground* if $\mathrm{vars}(t) = \emptyset$; t is *free* if $t \in \mathit{Vars}$; t is *linear* if, for all $y \in \mathrm{vars}(t)$, we have $\mathrm{occ\_lin}(y, t)$; finally, t is a *finite term* (or *Herbrand term*) if it contains a finite number of occurrences of function symbols. The sets of all ground, linear and finite terms are denoted by GTerms, LTerms and HTerms, respectively.

2.2 Substitutions

A *substitution* is a total function $\sigma : \mathit{Vars} \to \mathit{HTerms}$ that is the identity almost everywhere; in other words, the *domain* of σ,

$$\mathrm{dom}(\sigma) \stackrel{\mathrm{def}}{=} \{\, x \in \mathit{Vars} \mid \sigma(x) \neq x \,\},$$

is finite. Given a substitution $\sigma : \mathit{Vars} \to \mathit{HTerms}$, we overload the symbol 'σ' so as to denote also the function $\sigma : \mathit{HTerms} \to \mathit{HTerms}$ defined as follows, for each term $t \in \mathit{HTerms}$:

$$\sigma(t) \stackrel{\mathrm{def}}{=} \begin{cases} t, & \text{if } t \text{ is a constant symbol;} \\ \sigma(t), & \text{if } t \in \mathit{Vars}; \\ f\bigl(\sigma(t_1), \ldots, \sigma(t_n)\bigr), & \text{if } t = f(t_1, \ldots, t_n). \end{cases}$$

If $t \in \mathit{HTerms}$, we write $t\sigma$ to denote $\sigma(t)$. Note that, for each substitution σ and each finite term $t \in \mathit{HTerms}$, if $t\sigma \in \mathit{Vars}$, then $t \in \mathit{Vars}$.

If $x \in \mathit{Vars}$ and $t \in \mathit{HTerms} \setminus \{x\}$, then $x \mapsto t$ is called a *binding*. The set of all bindings is denoted by Bind. Substitutions are denoted by the set of their bindings, thus a substitution σ is identified with the (finite) set

$$\{\, x \mapsto x\sigma \mid x \in \mathrm{dom}(\sigma) \,\}.$$

We denote by $\mathrm{vars}(\sigma)$ the set of variables occurring in the bindings of σ. We also define $\mathrm{range}(\sigma) \stackrel{\mathrm{def}}{=} \bigcup \{\, \mathrm{vars}(x\sigma) \mid x \in \mathrm{dom}(\sigma) \,\}$.

A substitution is said to be *circular* if, for $n > 1$, it has the form

$$\{x_1 \mapsto x_2, \ldots, x_{n-1} \mapsto x_n, x_n \mapsto x_1\},$$

where $x_1, \ldots, x_n$ are distinct variables. A substitution is in *rational solved form* if it has no circular subset. The set of all substitutions in rational solved form is denoted by RSubst. A substitution σ is *idempotent* if, for all $t \in \mathit{Terms}$, we have $t\sigma\sigma = t\sigma$. Equivalently, σ is idempotent if and only if $\mathrm{dom}(\sigma) \cap \mathrm{range}(\sigma) = \emptyset$. The set of all idempotent substitutions is denoted by ISubst and $\mathit{ISubst} \subset \mathit{RSubst}$.

The composition of substitutions is defined in the usual way. Thus $\tau \circ \sigma$ is the substitution such that, for all terms $t \in \mathit{HTerms}$,

$$t(\tau \circ \sigma) = t\sigma\tau$$

and has the formulation

$$\tau \circ \sigma = \{\, x \mapsto x\sigma\tau \mid x \in \mathrm{dom}(\sigma) \cup \mathrm{dom}(\tau), x \neq x\sigma\tau \,\}. \tag{1}$$

As usual, σ^0 denotes the identity function (i.e. the empty substitution) and, when $i > 0$, σ^i denotes the substitution $(\sigma \circ \sigma^{i-1})$.

For each $\sigma \in \mathit{RSubst}$ and $s \in \mathit{HTerms}$, the sequence of finite terms

$$\sigma^0(s), \sigma^1(s), \sigma^2(s), \ldots$$

converges to a (possibly infinite) term, denoted $\sigma^\infty(s)$ (Intrigila and Zilli, 1996; King, 2000). Therefore, the function $\mathrm{rt} \colon \mathit{HTerms} \times \mathit{RSubst} \to \mathit{Terms}$ such that

$$\mathrm{rt}(s, \sigma) \stackrel{\mathrm{def}}{=} \sigma^\infty(s)$$

is well defined. Note that, in general, this function is not a substitution: while having a finite domain, its "bindings" $x \mapsto \mathrm{rt}(x, \sigma)$ can map a domain variable x into a term $\mathrm{rt}(x, \sigma) \in \mathit{Terms} \setminus \mathit{HTerms}$. However, as the name of the function suggests, the term $\mathrm{rt}(x, \sigma)$ is granted to be *rational*, meaning that it can only have a finite number of distinct subterms and hence, be finitely represented.

Example 1

Consider the substitutions

$$\begin{aligned}
\sigma_1 &= \{x \mapsto f(z), y \mapsto a\} && \in \mathit{ISubst}, \\
\sigma_2 &= \{x \mapsto f(y), y \mapsto a\} && \in \mathit{RSubst} \setminus \mathit{ISubst}, \\
\sigma_3 &= \{x \mapsto f(x)\} && \in \mathit{RSubst} \setminus \mathit{ISubst}, \\
\sigma_4 &= \{x \mapsto f(y), y \mapsto f(x)\} && \in \mathit{RSubst} \setminus \mathit{ISubst}, \\
\sigma_5 &= \{x \mapsto y, y \mapsto x\} && \notin \mathit{RSubst}.
\end{aligned}$$

Note that there are substitutions, such as σ_2, that are not idempotent and nonetheless define finite trees only; namely, $\mathrm{rt}(x, \sigma_2) = f(a)$. Similarly, there are other substitutions, such as σ_4, whose bindings are not explicitly cyclic and nonetheless define rational trees that are infinite; namely, $\mathrm{rt}(x, \sigma_4) = f(f(f(\cdots)))$. Finally note that the 'rt' function is not defined on $\sigma_5 \notin \mathit{RSubst}$.

2.3 Equality theories

An *equation* is of the form $s = t$ where $s, t \in \mathit{HTerms}$. Eqs denotes the set of all equations. A substitution σ may be regarded as a finite set of equations, that is, as the set $\{\, x = t \mid (x \mapsto t) \in \sigma \,\}$. We say that a set of equations e is in *rational solved form* if $\{\, s \mapsto t \mid (s = t) \in e \,\} \in \mathit{RSubst}$. In the rest of the paper, we often write a substitution $\sigma \in \mathit{RSubst}$ to denote a set of equations in rational solved form (and vice versa). As is common in research work involving equality, we overload the symbol '=' and use it to denote both equality and to represent syntactic identity. The context makes it clear what is intended.

Let $\{r, s, t, s_1, \ldots, s_n, t_1, \ldots, t_n\} \subseteq \mathit{HTerms}$. We assume that any equality theory T over *Terms* includes the *congruence axioms* denoted by the following schemata:

$$s = s, \tag{2}$$

$$s = t \leftrightarrow t = s, \tag{3}$$

$$r = s \wedge s = t \rightarrow r = t, \tag{4}$$

$$s_1 = t_1 \wedge \cdots \wedge s_n = t_n \rightarrow f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n). \tag{5}$$

In logic programming and most implementations of Prolog it is usual to assume an equality theory based on syntactic identity. This consists of the congruence axioms together with the *identity axioms* denoted by the following schemata, where f and g are distinct function symbols or $n \neq m$:

$$f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n) \rightarrow s_1 = t_1 \wedge \cdots \wedge s_n = t_n, \tag{6}$$

$$\neg\bigl(f(s_1, \ldots, s_n) = g(t_1, \ldots, t_m)\bigr). \tag{7}$$

The axioms characterized by schemata (6) and (7) ensure the equality theory depends only on the syntax. The equality theory for a non-syntactic domain replaces these axioms by ones that depend instead on the semantics of the domain and, in particular, on the interpretation given to functor symbols.

The equality theory of Clark (1978), denoted $\mathscr{FT}$, on which pure logic programming is based, usually called the *Herbrand* equality theory, is given by the congruence axioms, the identity axioms, and the axiom schema

$$\forall z \in \mathit{Vars} : \forall t \in (\mathit{HTerms} \setminus \mathit{Vars}) : z \in \operatorname{vars}(t) \rightarrow \neg(z = t). \tag{8}$$

Axioms characterized by the schema (8) are called the *occurs-check axioms* and are an essential part of the standard unification procedure in SLD-resolution.

An alternative approach used in some implementations of logic programming systems, such as Prolog II, SICStus and Oz, does not require the occurs-check axioms. This approach is based on the theory of rational trees (Colmerauer, 1982, 1984), denoted $\mathscr{RT}$. It assumes the congruence axioms and the identity axioms together with a *uniqueness axiom* for each substitution in rational solved form. Informally speaking these state that, after assigning a ground rational tree to each variable which is not in the domain, the substitution uniquely defines a ground rational tree for each of its domain variables. Note that being in rational solved form is a very weak property. Indeed, unification algorithms returning a set of equations in rational solved form are allowed to be much more “lazy” than one would expect. We refer the interested reader elsewhere (Jaffar *et al.*, 1987; Keisu, 1994; Maher, 1988) for details on the subject.

In the sequel we use the expression “equality theory” to denote any consistent, decidable theory T satisfying the congruence axioms. We also use the expression “syntactic equality theory” to denote any equality theory T also satisfying the identity axioms.

We say that a substitution $\sigma \in \mathit{RSubst}$ is *satisfiable* in an equality theory T if, when interpreting σ as an equation system in rational solved form,

$$T \vdash \forall\bigl(\mathit{Vars} \setminus \operatorname{dom}(\sigma)\bigr) : \exists \operatorname{dom}(\sigma) . \sigma.$$

Let $e \in \wp_{\mathrm{f}}(Eqs)$ be a set of equations in an equality theory T. A substitution $\sigma \in RSubst$ is called a *solution for e in T* if σ is satisfiable in T and $T \vdash \forall(\sigma \rightarrow e)$; we say that e is satisfiable if it has a solution. If $\mathrm{vars}(\sigma) \subseteq \mathrm{vars}(e)$, then σ is said to be a *relevant* solution for e. In addition, σ is a *most general solution for e in T* if $T \vdash \forall(\sigma \leftrightarrow e)$. In this paper, a most general solution is always a relevant solution of e. When the theory T is clear from the context, the set of all the relevant most general solutions for e in T is denoted by $\mathrm{mgs}(e)$.

Example 2
Let $e = \{g(x) = g(f(y)), f(x) = y, z = g(w)\}$ and

$$\sigma = \{x \mapsto f(y), y \mapsto f(x), z \mapsto g(w)\}.$$

Then, for any syntactic equality theory T, we have $T \vdash \forall(\sigma \leftrightarrow e)$. Since $\sigma \in RSubst$, then σ and hence e is satisfiable in $\mathcal{RT}$. Intuitively, whatever rational tree t_w is assigned to the parameter variable w, there exist rational trees t_x, t_y and t_z that, when assigned to the domain variables x, y and z, will turn σ into a set of trivial identities; namely, let t_x and t_y be both equal to the infinite rational tree $f(f(f(\cdots)))$, which is usually denoted by f^ω, and let t_z be the rational tree $g(t_w)$. Thus σ is a relevant most general solution for e in $\mathcal{RT}$. In contrast,

$$\tau = \{x \mapsto f(y), y \mapsto f(x), z \mapsto g(f(a))\}$$

is just a relevant solution for e in $\mathcal{RT}$. Also observe that, for any equality theory T,

$$T \vdash \forall\Big(\sigma \rightarrow \{x = f(f(x))\}\Big)$$

so that σ does not satisfy the occurs-check axioms. Therefore, neither σ nor e are satisfiable in the Herbrand equality theory $\mathcal{FT}$ Intuitively, there is no finite tree t_x such that $t_x = f(f(t_x))$.

We have the following useful result regarding 'rt' and satisfiable substitutions that are equivalent with respect to any given syntactic equality theory.

Proposition 3
Let $\sigma, \tau \in RSubst$ be satisfiable in the syntactic equality theory T and suppose that $T \vdash \forall(\sigma \leftrightarrow \tau)$. Then

$$\mathrm{rt}(y,\sigma) \in Vars \iff \mathrm{rt}(y,\tau) \in Vars, \tag{9}$$

$$\mathrm{rt}(y,\sigma) \in GTerms \iff \mathrm{rt}(y,\tau) \in GTerms, \tag{10}$$

$$\mathrm{rt}(y,\sigma) \in LTerms \iff \mathrm{rt}(y,\tau) \in LTerms. \tag{11}$$

2.4 Galois connections and upper closure operators

Given two complete lattices $(C, \leqslant_C)$ and $(A, \leqslant_A)$, a *Galois connection* is a pair of monotonic functions $\alpha : C \rightarrow A$ and $\gamma : A \rightarrow C$ such that

$$\forall c \in C : c \leqslant_C \gamma\big(\alpha(c)\big), \qquad \forall a \in A : \alpha\big(\gamma(a)\big) \leqslant_A a.$$

The functions α and γ are said to be the abstraction and concretization functions, respectively. A *Galois insertion* is a Galois connection where the concretization function γ is injective.

An *upper closure operator* (uco) $\rho : C \to C$ on the complete lattice $(C, \leqslant_C)$ is a monotonic, idempotent and extensive[1] self-map. The set of all uco's on C, denoted by $\mathrm{uco}(C)$, is itself a complete lattice. For any $\rho \in \mathrm{uco}(C)$, the set $\rho(C)$, i.e. the image under ρ of the lattice carrier, is a complete lattice under the same partial order $\leqslant_C$ defined on C. Given a Galois connection, the function $\rho \stackrel{\mathrm{def}}{=} \gamma \circ \alpha$ is an element of $\mathrm{uco}(C)$. The presentation of abstract interpretation in terms of Galois connections can be rephrased by using uco's. In particular, the partial order $\sqsubseteq$ defined on $\mathrm{uco}(C)$ formalizes the intuition of an abstract domain being more precise than another one; moreover, given two elements $\rho_1, \rho_2 \in \mathrm{uco}(C)$, their reduced product (Cousot and Cousot 1979), denoted $\rho_1 \sqcap \rho_2$, is their glb on $\mathrm{uco}(C)$.

2.5 The set-sharing domain

The set-sharing domain of Jacobs and Langen (Jacobs and Langen 1989), encodes both aliasing and groundness information. Let $VI \subseteq_{\mathrm{f}} Vars$ be a fixed and finite set of variables of interest. An element of the set-sharing domain (a *sharing set*) is a set of subsets of VI (the *sharing groups*). Note that the empty set is not a sharing group.

Definition 4

(The *set-sharing* lattice) Let $SG \stackrel{\mathrm{def}}{=} \wp(VI) \setminus \{\varnothing\}$ be the set of *sharing groups*. The set-sharing lattice is defined as $SH \stackrel{\mathrm{def}}{=} \wp(SG)$, ordered by subset inclusion.

The following operators on SH are needed for the specification of the abstract semantics.

Definition 5

(Auxiliary operators on SH) For each $sh, sh_1, sh_2 \in SH$ and each $V \subseteq VI$, we define the following functions:
the *star-union* function $(\cdot)^\star : SH \to SH$, is defined as

$$sh^\star \stackrel{\mathrm{def}}{=} \{\, S \in SG \mid \exists n \geqslant 1 \,.\, \exists S_1, \ldots, S_n \in sh \,.\, S = S_1 \cup \cdots \cup S_n \,\};$$

the extraction of the *relevant component of sh with respect to V* is encoded by $\mathrm{rel} : \wp(VI) \times SH \to SH$ defined as

$$\mathrm{rel}(V, sh) \stackrel{\mathrm{def}}{=} \{\, S \in sh \mid S \cap V \neq \varnothing \,\};$$

the *irrelevant component of sh with respect to V* is thus defined as

$$\overline{\mathrm{rel}}(V, sh) \stackrel{\mathrm{def}}{=} sh \setminus \mathrm{rel}(V, sh);$$

[1] Namely, $c \leqslant_C \rho(c)$ for each $c \in C$.

the *binary union* function $\mathrm{bin}\colon SH \times SH \to SH$ is defined as

$$\mathrm{bin}(sh_1, sh_2) \stackrel{\mathrm{def}}{=} \{\, S_1 \cup S_2 \mid S_1 \in sh_1, S_2 \in sh_2 \,\};$$

the *self-bin-union* operation on SH is defined as

$$sh^2 \stackrel{\mathrm{def}}{=} \mathrm{bin}(sh, sh);$$

the *abstract existential quantification* function $\mathrm{aexists}\colon SH \times \wp(VI) \to SH$ is defined as

$$\mathrm{aexists}(sh, V) \stackrel{\mathrm{def}}{=} \{\, S \setminus V \mid S \in sh, S \setminus V \neq \varnothing \,\} \cup \{\, \{x\} \mid x \in V \,\}.$$

In Bagnara *et al.* (1997, 2002), it was shown that the domain SH contains many elements that are redundant for the computation of the actual *observable* properties of the analysis, definite groundness and definite independence. The following formalization of these observables is a rewording of the definitions provided in Zaffanella *et al.* (1999, 2002).

Definition 6

(The observables of SH**)** The *groundness* and *independence* observables (on SH) $\rho_{Con}, \rho_{PS} \in \mathrm{uco}(SH)$ are defined, for each $sh \in SH$, by

$$\rho_{Con}(sh) \stackrel{\mathrm{def}}{=} \{\, S \in SG \mid S \subseteq \mathrm{vars}(sh) \,\},$$
$$\rho_{PS}(sh) \stackrel{\mathrm{def}}{=} \{\, S \in SG \mid (P \subseteq S \wedge \#P = 2) \implies (\exists T \in sh \,.\, P \subseteq T) \,\}.$$

Note that, as usual in sharing analysis domains, definite groundness and definite independence are both represented by encoding possible non-groundness and possible pair-sharing information.

The abstract domain PSD (Bagnara *et al.*, 2002; Zaffanella *et al.*, 2002) is the simplest abstraction of the domain SH that still preserves the same precision on groundness and independence.

Definition 7

(The *pair-sharing dependency* lattice PSD**)** The operator $\rho_{PSD} \in \mathrm{uco}(SH)$ is defined, for each $sh \in SH$, by

$$\rho_{PSD}(sh) \stackrel{\mathrm{def}}{=} \left\{\, S \in SG \;\middle|\; \forall y \in S : S = \bigcup \{\, U \in sh \mid y \in U \subseteq S \,\} \,\right\}.$$

The *pair-sharing dependency* lattice is $PSD \stackrel{\mathrm{def}}{=} \rho_{PSD}(SH)$.

In the following example we provide an intuitive interpretation of the approximation induced by the three upper closure operators of Definitions 6 and 7.

Example 8

Let $VI = \{v, w, x, y, z\}$ and consider[2] $sh = \{vx, vy, xy, xyz\}$. Then

$$\begin{aligned}
\rho_{Con}(sh) &= \{v, vx, vxy, vxyz, vxz, vy, vyz, vz, x, xy, xyz, xz, y, yz, z\},\\
\rho_{PS}(sh) &= \{v, vx, vxy, vy, w, x, xy, xyz, xz, y, yz, z\},\\
\rho_{PSD}(sh) &= \{vx, vxy, vy, xy, xyz\}.
\end{aligned}$$

When observing $\rho_{Con}(sh)$, the only information available is that variable w does not occur in a sharing group; intuitively, this means that w is definitely ground. All the other information encoded in sh is lost; for instance, in sh variables v and z never occur in the *same* sharing group (i.e. they are definitely independent), while this happens in $\rho_{Con}(sh)$.

When observing $\rho_{PS}(sh)$, it should be noted that two distinct variables occur in the same sharing group if and only if they were also occurring together in a sharing group of sh, so that the definite independence information is preserved (e.g. v and z keep their independence). On the other hand, all the variables in VI occur as singletons in $\rho_{PS}(sh)$ whether or not they are known to be ground; for instance, $\{w\}$ occurs in $\rho_{PS}(sh)$ although w does not occur in any sharing group in sh.

By noting that $\rho_{PSD}(sh) \subset \rho_{Con}(sh) \cap \rho_{PS}(sh)$, it follows that $\rho_{PSD}(sh)$ preserves both the definite groundness and the definite independence information of sh; moreover, as the inclusion is strict, $\rho_{PSD}(sh)$ encodes other information, such as variable covering (the interested reader is referred to (Bagnara *et al.*, 2002; Zaffanella *et al.*, 2002) for a more formal discussion).

2.6 *Variable-idempotent substitutions*

One of the key concepts used in Hill *et al.* (2003) for the proofs of the correctness results stated in this paper is that of variable-idempotence. For the interested reader, we provide here a brief introduction to variable-idempotent substitutions, although these are not referred to elsewhere in the paper.

The definition of idempotence requires that repeated applications of a substitution do not change the syntactic structure of a term and idempotent substitutions are normally the preferred form of a solution to a set of equations. However, in the domain of rational trees, a set of solvable equations does not necessarily have an idempotent solution (for instance, in Example 2, the set of equations e has no idempotent solution). On the other hand, several abstractions of terms, such as the ones commonly used for sharing analysis, are only interested in the set of variables occurring in a term and not in the concrete structure that contains them. Thus, for applications such as sharing analysis, a useful way to relax the definition of idempotence is to ignore the structure of terms and just require that the repeated application of a substitution leaves the set of variables in a term invariant.

[2] In this and all the following examples, we will adopt a simplified notation for a set-sharing element sh, omitting inner braces. For instance, we will write $\{xy, xz, yz\}$ to denote $\{\{x, y\}, \{x, z\}, \{y, z\}\}$.

Definition 9
(Variable-idempotence) A substitution $\sigma \in RSubst$ is *variable-idempotent*[3] if and only if for all $t \in HTerms$ we have

$$\mathrm{vars}(t\sigma\sigma) = \mathrm{vars}(t\sigma).$$

The set of variable-idempotent substitutions is denoted $VSubst$.

As any idempotent substitution is also variable-idempotent, we have $ISubst \subset VSubst \subset RSubst$.

Example 10
Consider the following substitutions which are all in $RSubst$.

$$\begin{aligned}
\sigma_1 &= \{x \mapsto f(y)\} && \in ISubst \subset VSubst,\\
\sigma_2 &= \{x \mapsto f(x)\} && \in VSubst \setminus ISubst,\\
\sigma_3 &= \{x \mapsto f(y,z), y \mapsto f(z,y)\} && \in VSubst \setminus ISubst,\\
\sigma_4 &= \{x \mapsto y, y \mapsto f(x,y)\} && \notin VSubst.
\end{aligned}$$

3 The domain *SFL*

The abstract domain *SFL* is made up of three components, providing different kinds of sharing information regarding the set of variables of interest VI: the first component is the set-sharing domain SH of Jacobs and Langen (1989); the other two components provide freeness and linearity information, each represented by simply recording those variables of interest that are known to enjoy the corresponding property.

Definition 11
(The domain *SFL*) Let $F \stackrel{\mathrm{def}}{=} \wp(VI)$ and $L \stackrel{\mathrm{def}}{=} \wp(VI)$ be partially ordered by reverse subset inclusion. The abstract domain SFL is defined as

$$SFL \stackrel{\mathrm{def}}{=} \{\, \langle sh, f, l\rangle \mid sh \in SH, f \in F, l \in L \,\}$$

and is ordered by $\leqslant_s$, the component-wise extension of the orderings defined on the sub-domains. With this ordering, SFL is a complete lattice whose least upper bound operation is denoted by alub_s. The bottom element $\langle \emptyset, VI, VI\rangle$ will be denoted by $\bot_s$.

3.1 The abstraction function

When the concrete domain is based on the theory of finite trees, idempotent substitutions provide a finitely computable *strong normal form* for domain elements,

[3] This definition, which is the same as that originally provided in Hill *et al.* (1998), is slightly stronger than the one adopted in Hill *et al.* (2002), which disregarded the domain variables of the substitution. The adoption of this stronger definition allows for some simplifications in the correctness proofs for freeness and linearity.

meaning that different substitutions describe different sets of finite trees.[4] In contrast, when working on a concrete domain based on the theory of rational trees, substitutions in rational solved form, while being finitely computable, no longer satisfy this property: there can be an infinite set of substitutions in rational solved form all describing the same set of rational trees (i.e. the same element in the "intended" semantics). For instance, the substitutions

$$\sigma_n = \{x \mapsto \overbrace{f(\cdots f(}^{n} x) \cdots)\},$$

for $n = 1, 2, \ldots$, all map the variable x into the same infinite rational tree f^{ω}.

Ideally, a strong normal form for the set of rational trees described by a substitution $\sigma \in \mathit{RSubst}$ can be obtained by computing the limit σ^{∞}. The problem is that σ^{∞} can map domain variables to infinite rational terms and may not be in RSubst.

This poses a non-trivial problem when trying to define "good" abstraction functions, since it would be really desirable for this function to map any two equivalent concrete elements to the same abstract element. As shown in Hill *et al.* (2002), the classical abstraction function for set-sharing analysis (Cortesi and Filé, 1999; Jacobs and Langen, 1989), which was defined only for substitutions that are idempotent, does not enjoy this property when applied, as it is, to arbitrary substitutions in rational solved form. In Hill *et al.* (1998, 2002), this problem is solved by replacing the sharing group operator 'sg' of Jacobs and Langen (1989) by an occurrence operator, 'occ', defined by means of a fixpoint computation. However, to simplify the presentation, here we define 'occ' directly by exploiting the fact that the number of iterations needed to reach the fixpoint is bounded by the number of bindings in the substitution.

Definition 12

(Occurrence operator) For each $\sigma \in \mathit{RSubst}$ and $v \in \mathit{Vars}$, the *occurrence operator* $\mathrm{occ}\colon \mathit{RSubst} \times \mathit{Vars} \to \wp_{\mathrm{f}}(\mathit{Vars})$ is defined as

$$\mathrm{occ}(\sigma, v) \stackrel{\mathrm{def}}{=} \{\, y \in \mathit{Vars} \mid n = \#\sigma, v \in \mathrm{vars}(y\sigma^n) \setminus \mathrm{dom}(\sigma) \,\}.$$

For each $\sigma \in \mathit{RSubst}$, the operator $\mathrm{ssets}\colon \mathit{RSubst} \to \mathit{SH}$ is defined as

$$\mathrm{ssets}(\sigma) \stackrel{\mathrm{def}}{=} \{\, \mathrm{occ}(\sigma, v) \cap \mathit{VI} \mid v \in \mathit{Vars} \,\} \setminus \{\varnothing\}.$$

The operator 'ssets' is introduced for notational convenience only.

Example 13

Let

$$\sigma = \{x_1 \mapsto f(x_2), x_2 \mapsto g(x_3, x_4), x_3 \mapsto x_1\},$$
$$\tau = \{x_1 \mapsto f(g(x_3, x_4)), x_2 \mapsto g(x_3, x_4), x_3 \mapsto f(g(x_3, x_4))\}.$$

[4] As usual, this is modulo the possible renaming of variables.

Then $\mathrm{dom}(\sigma) = \mathrm{dom}(\tau) = \{x_1, x_2, x_3\}$ so that $\mathrm{occ}(\sigma, x_i) = \mathrm{occ}(\tau, x_i) = \emptyset$, for $i = 1$, 2, 3 and $\mathrm{occ}(\sigma, x_4) = \mathrm{occ}(\tau, x_4) = \{x_1, x_2, x_3, x_4\}$. As a consequence, supposing that $VI = \{x_1, x_2, x_3, x_4\}$, we obtain $\mathrm{ssets}(\sigma) = \mathrm{ssets}(\tau) = \{VI\}$.

In a similar way, it is possible to define suitable operators for groundness, freeness and linearity. As all ground trees are linear, a knowledge of the definite groundness information can be useful for proving properties concerning the linearity abstraction. Groundness is already encoded in the abstraction for set-sharing provided in Definition 12; nonetheless, for both a simplified notation and a clearer intuitive reading, we now explicitly define the set of variables that are associated to ground trees by a substitution in *RSubst*.

Definition 14
(Groundness operator) The *groundness operator* $\mathrm{gvars}\colon RSubst \to \wp_{\mathrm{f}}(Vars)$ is defined, for each $\sigma \in RSubst$, by

$$\mathrm{gvars}(\sigma) \stackrel{\mathrm{def}}{=} \{\, y \in \mathrm{dom}(\sigma) \mid \forall v \in Vars : y \notin \mathrm{occ}(\sigma, v) \,\}.$$

Example 15
Consider $\sigma \in RSubst$ where

$$\sigma = \{x_1 \mapsto x_2, x_2 \mapsto f(a), x_3 \mapsto x_4, x_4 \mapsto f(x_2, x_4)\}.$$

Then $\mathrm{gvars}(\sigma) = \{x_1, x_2, x_3, x_4\}$. Observe that $x_1 \in \mathrm{gvars}(\sigma)$ although $x_1\sigma \in Vars$. Also, $x_3 \in \mathrm{gvars}(\sigma)$ although $\mathrm{vars}(x_3\sigma^i) = \{x_2, x_4\} \neq \emptyset$ for all $i \geqslant 2$.

As for possible sharing, the definite freeness information can be extracted from a substitution in rational solved form by observing the result of a bounded number of applications of the substitution.

Definition 16
(Freeness operator) The *freeness operator* $\mathrm{fvars}\colon RSubst \to \wp(Vars)$ is defined, for each $\sigma \in RSubst$, by

$$\mathrm{fvars}(\sigma) \stackrel{\mathrm{def}}{=} \{\, y \in Vars \mid n = \#\sigma, y\sigma^n \in Vars \,\}.$$

As $\sigma \in RSubst$ has no circular subset, $y \in \mathrm{fvars}(\sigma)$ implies $y\sigma^n \in Vars \setminus \mathrm{dom}(\sigma)$.

Example 17
Let $VI = \{x_1, x_2, x_3, x_4, x_5\}$ and consider $\sigma \in RSubst$ where

$$\sigma = \{x_1 \mapsto x_2, x_2 \mapsto f(x_3), x_3 \mapsto x_4, x_4 \mapsto x_5\}.$$

Then $\mathrm{fvars}(\sigma) \cap VI = \{x_3, x_4, x_5\}$. Thus $x_1 \notin \mathrm{fvars}(\sigma)$ although $x_1\sigma \in Vars$. Also, $x_3 \in \mathrm{fvars}(\sigma)$ although $x_3\sigma \in \mathrm{dom}(\sigma)$.

As in previous cases, the definite linearity information can be extracted by observing the result of a bounded number of applications of the considered substitution.

Definition 18
(Linearity operator) The *linearity operator* $\mathrm{lvars}\colon RSubst \to \wp(Vars)$ is defined, for each $\sigma \in RSubst$, by

$$\mathrm{lvars}(\sigma) \stackrel{\mathrm{def}}{=} \{\, y \in Vars \mid n = \#\sigma, \forall z \in \mathrm{vars}(y\sigma^n) \setminus \mathrm{dom}(\sigma) : \mathrm{occ\_lin}(z, y\sigma^{2n}) \,\}.$$

In the next example we consider the extraction of linearity from two substitutions. The substitution σ shows that, in contrast with the case of set-sharing and freeness, for linearity we may need to compute up to $2n$ applications, where $n = \#\sigma$; the substitution τ shows that, when observing the term $y\tau^{2n}$, multiple occurrences of domain variables have to be disregarded.

Example 19
Let $VI = \{x_1, x_2, x_3, x_4\}$ and consider $\sigma \in RSubst$ where

$$\sigma = \{x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto f(x_1, x_4)\}.$$

Then $\mathrm{lvars}(\sigma) \cap VI = \{x_4\}$. Observe that $x_1 \notin \mathrm{lvars}(\sigma)$. This is because $x_4 \notin \mathrm{dom}(\sigma)$, $x_1\sigma^3 = f(x_1, x_4)$ so that $x_4 \in \mathrm{vars}(x_1\sigma^3)$ and $x_1\sigma^6 = f\bigl(f(x_1, x_4), x_4\bigr)$ so that $\mathrm{occ\_lin}(x_4, x_1\sigma^6)$ does not hold. Note also that $\mathrm{occ\_lin}(x_4, x_1\sigma^i)$ holds for $i = 3$, 4, 5.

Consider now $\tau \in RSubst$ where

$$\tau = \{x_1 \mapsto f(x_2, x_2), x_2 \mapsto f(x_2)\}.$$

Then $\mathrm{lvars}(\tau) \cap VI = VI$. Note that we have $x_1 \in \mathrm{lvars}(\tau)$ although, for all $i > 0$, $x_2 \in \mathrm{dom}(\tau)$ occurs more than once in the term $x_1\tau^i$.

The occurrence, groundness, freeness and linearity operators are invariant with respect to substitutions that are equivalent in the given syntactic equality theory.

Proposition 20
Let $\sigma, \tau \in RSubst$ be satisfiable in the syntactic equality theory T and suppose that $T \vdash \forall(\sigma \leftrightarrow \tau)$. Then

$$\mathrm{ssets}(\sigma) = \mathrm{ssets}(\tau), \tag{12}$$
$$\mathrm{gvars}(\sigma) = \mathrm{gvars}(\tau), \tag{13}$$
$$\mathrm{fvars}(\sigma) = \mathrm{fvars}(\tau), \tag{14}$$
$$\mathrm{lvars}(\sigma) = \mathrm{lvars}(\tau). \tag{15}$$

Moreover, these operators precisely capture the intended properties over the domain of rational trees.

Proposition 21
If $\sigma \in RSubst$ and $y, v \in Vars$ then

$$y \in \mathrm{occ}(\sigma, v) \iff v \in \mathrm{vars}\bigl(\mathrm{rt}(y, \sigma)\bigr), \tag{16}$$
$$y \in \mathrm{gvars}(\sigma) \iff \mathrm{rt}(y, \sigma) \in GTerms, \tag{17}$$
$$y \in \mathrm{fvars}(\sigma) \iff \mathrm{rt}(y, \sigma) \in Vars, \tag{18}$$
$$y \in \mathrm{lvars}(\sigma) \iff \mathrm{rt}(y, \sigma) \in LTerms. \tag{19}$$

It follows from (16) and (18) that any free variable necessarily shares (at least, with itself). Also, as $Vars \cup GTerms \subset LTerms$, it follows from (17), (18) and (19) that any variable that is either ground or free is also necessarily linear. Thus we have the following corollary.

Corollary 22
If $\sigma \in RSubst$, then

$$\begin{aligned} \mathrm{fvars}(\sigma) &\subseteq \mathrm{vars}\bigl(\mathrm{ssets}(\sigma)\bigr), \\ \mathrm{fvars}(\sigma) \cup \mathrm{gvars}(\sigma) &\subseteq \mathrm{lvars}(\sigma). \end{aligned}$$

We are now in position to define the abstraction function mapping rational trees to elements of the domain *SFL*.

Definition 23
(The abstraction function for *SFL***)** For each substitution $\sigma \in RSubst$, the function $\alpha_s : RSubst \to SFL$ is defined by

$$\alpha_s(\sigma) \stackrel{\mathrm{def}}{=} \langle \mathrm{ssets}(\sigma), \mathrm{fvars}(\sigma) \cap VI, \mathrm{lvars}(\sigma) \cap VI \rangle.$$

The concrete domain $\wp(RSubst)$ is related to *SFL* by means of the *abstraction function* $\alpha_s : \wp(RSubst) \to SFL$ such that, for each $\Sigma \in \wp(RSubst)$,

$$\alpha_s(\Sigma) \stackrel{\mathrm{def}}{=} \mathrm{alub}_s\{\, \alpha_s(\sigma) \mid \sigma \in \Sigma \,\}.$$

Since the abstraction function α_s is additive, the concretization function is given by the adjoint (Cousot and Cousot 1977)

$$\gamma_s\bigl(\langle sh, f, l\rangle\bigr) \stackrel{\mathrm{def}}{=} \{\, \sigma \in RSubst \mid \mathrm{ssets}(\sigma) \subseteq sh, \mathrm{fvars}(\sigma) \supseteq f, \mathrm{lvars}(\sigma) \supseteq l \,\}.$$

With Definition 23 and Proposition 20, one of our objectives is fulfilled: substitutions in *RSubst* that are equivalent have the same abstraction.

Corollary 24
Let $\sigma, \tau \in RSubst$ be satisfiable in the syntactic equality theory T and suppose $T \vdash \forall(\sigma \leftrightarrow \tau)$. Then $\alpha_s(\sigma) = \alpha_s(\tau)$.

Observe that the Galois connection defined by the functions α_s and γ_s is not a Galois insertion since different abstract elements are mapped by γ_s to the same set of concrete computation states. To see this it is sufficient to observe that, by Corollary 22, any abstract element $d = \langle sh, f, l\rangle \in SFL$ such that $f \nsubseteq \mathrm{vars}(sh)$, as is the case for the bottom element $\bot_s$, satisfies $\gamma_s(d) = \gamma_s(\bot_s) = \varnothing$; thus, all such d's will represent the semantics of those program fragments that have no successful computations. Similarly, by letting $V = \bigl(VI \setminus \mathrm{vars}(sh)\bigr) \cup f$, it can be seen that, for any l' such that $V \cup l = V \cup l'$, we have, again by Corollary 22, $\gamma_s(d) = \gamma_s\bigl(\langle sh, f, l'\rangle\bigr)$.

Of course, by taking the abstract domain as the subset of *SFL* that is the co-domain of α_s, we would have a Galois insertion. However, apart from the simple cases shown above, it is somehow difficult to *explicitly* characterize such a set. For instance, as observed in (Filé 1994), if

$$d = \bigl\langle \{xy, xz, yz\}, \{x, y, z\}, \{x, y, z\} \bigr\rangle \in SFL$$

we have $\gamma_s(d) = \gamma_s(\bot_s) = \varnothing$. It is worth stressing that these "spurious" elements do not compromise the correctness of the analysis and, although they can affect the precision of the analysis, they rarely occur in practice (Bagnara *et al.*, 2000; Zaffanella, 2001).

3.2 The abstract operators

The specification of the abstract unification operator on the domain *SFL* is rather complex, since it is based on a very detailed case analysis. To achieve some modularity, that will be also useful when proving its correctness, in the next definition we introduce several auxiliary abstract operators.

Definition 25

(Auxiliary operators in *SFL*) Let $s, t \in HTerms$ be finite terms such that $\mathrm{vars}(s) \cup \mathrm{vars}(t) \subseteq VI$. For each $d = \langle sh, f, l \rangle \in SFL$ we define the following predicates:

s and t are *independent in* d if and only if $\mathrm{ind}_d : HTerms^2 \to Bool$ holds for (s, t), where

$$\mathrm{ind}_d(s,t) \stackrel{\mathrm{def}}{=} \Bigl(\mathrm{rel}\bigl(\mathrm{vars}(s), sh\bigr) \cap \mathrm{rel}\bigl(\mathrm{vars}(t), sh\bigr) = \varnothing\Bigr);$$

t is *ground in* d if and only if $\mathrm{ground}_d : HTerms \to Bool$ holds for t, where

$$\mathrm{ground}_d(t) \stackrel{\mathrm{def}}{=} \bigl(\mathrm{vars}(t) \subseteq VI \setminus \mathrm{vars}(sh)\bigr);$$

$y \in \mathrm{vars}(t)$ *occurs linearly (in* t*) in* d if and only if $\mathrm{occ\_lin}_d : VI \times HTerms \to Bool$ holds for (y, t), where

$$\mathrm{occ\_lin}_d(y,t) \stackrel{\mathrm{def}}{=} \mathrm{ground}_d(y) \vee \Bigl(\mathrm{occ\_lin}(y,t) \wedge (y \in l) \wedge \forall z \in \mathrm{vars}(t) : \bigl(y \neq z \implies \mathrm{ind}_d(y,z)\bigr)\Bigr);$$

t is *free in* d if and only if $\mathrm{free}_d : HTerms \to Bool$ holds for t, where

$$\mathrm{free}_d(t) \stackrel{\mathrm{def}}{=} (t \in f);$$

t is *linear in* d if and only if $\mathrm{lin}_d : HTerms \to Bool$ holds for t, where

$$\mathrm{lin}_d(t) \stackrel{\mathrm{def}}{=} \forall y \in \mathrm{vars}(t) : \mathrm{occ\_lin}_d(y,t).$$

The function $\mathrm{share\_with}_d : HTerms \to \wp(VI)$ yields the set of variables of interest that may share with the given term. For each $t \in HTerms$,

$$\mathrm{share\_with}_d(t) \stackrel{\mathrm{def}}{=} \mathrm{vars}\Bigl(\mathrm{rel}\bigl(\mathrm{vars}(t), sh\bigr)\Bigr).$$

The function $\mathrm{cyclic}_x^t : SH \to SH$ strengthens the sharing set sh by forcing the coupling of x with t. For each $sh \in SH$ and each $(x \mapsto t) \in Bind$,

$$\mathrm{cyclic}_x^t(sh) \stackrel{\mathrm{def}}{=} \overline{\mathrm{rel}}\bigl(\{x\} \cup \mathrm{vars}(t), sh\bigr) \cup \mathrm{rel}\bigl(\mathrm{vars}(t) \setminus \{x\}, sh\bigr).$$

As a first correctness result, we have that the auxiliary operators correctly approximate the corresponding concrete properties.

Theorem 26
Let $d \in SFL$, $\sigma \in \gamma_S(d)$ and $y \in VI$. Let also $s, t \in HTerms$ be two finite terms such that $\mathrm{vars}(s) \cup \mathrm{vars}(t) \subseteq VI$. Then

$$\mathrm{ind}_d(s,t) \implies \mathrm{vars}\bigl(\mathrm{rt}(s,\sigma)\bigr) \cap \mathrm{vars}\bigl(\mathrm{rt}(t,\sigma)\bigr) = \varnothing; \tag{20}$$

$$\mathrm{ind}_d(y,t) \iff y \notin \mathrm{share\_with}_d(t); \tag{21}$$

$$\mathrm{free}_d(t) \implies \mathrm{rt}(t,\sigma) \in Vars; \tag{22}$$

$$\mathrm{ground}_d(t) \implies \mathrm{rt}(t,\sigma) \in GTerms; \tag{23}$$

$$\mathrm{lin}_d(t) \implies \mathrm{rt}(t,\sigma) \in LTerms. \tag{24}$$

Example 27
Let $VI = \{v, w, x, y, z\}$ and consider the abstract element $d = \langle sh, f, l \rangle \in SFL$, where

$$sh = \{v, wz, xz, z\}, \qquad f = \{v\}, \qquad l = \{v, x, y, z\}.$$

Then, by applying Definition 25, we obtain the following:

- $\mathrm{ground}_d(x)$ does not hold whereas $\mathrm{ground}_d\bigl(h(y)\bigr)$ holds.
- $\mathrm{free}_d(v)$ holds but $\mathrm{free}_d\bigl(h(v)\bigr)$ does not hold.
- Both $\mathrm{ind}_d(w,x)$ and $\mathrm{ind}_d\bigl(f(w,y), f(x,y)\bigr)$ hold whereas $\mathrm{ind}_d(x,z)$ does not hold; note that, in the second case, the two arguments of the predicate do share y, but this does not affect the independence of the corresponding terms, because y is definitely ground in the abstract element d.
- Let $t = f(w, x, x, y, y, z)$; then $\mathrm{occ\_lin}_d(w,t)$ does not hold because $w \notin l$; $\mathrm{occ\_lin}_d(x,t)$ does not hold because x occurs more than once in t; $\mathrm{occ\_lin}_d(y,t)$ holds, even though y occurs twice in t, because y is definitely ground in d; $\mathrm{occ\_lin}_d(z,t)$ does not hold because both x and z occur in term t and, as observed in the point above, $\mathrm{ind}_d(x,z)$ does not hold.
- For the reasons given in the point above, $\mathrm{lin}_d(t)$ does not hold; in contrast, $\mathrm{lin}_d\bigl(f(y,y,z)\bigr)$ holds.
- $\mathrm{share\_with}_d(w) = \{w, z\}$ and $\mathrm{share\_with}_d(x) = \{x, z\}$; thus, both w and x may share one or more variables with z; since we observed that w and x are definitely independent in d, this means that the set of variables that w shares with z is disjoint from the set of variables that x shares with z.
- Let $t = f(w,z)$; then

  $$\begin{aligned}\mathrm{cyclic}_z^t(sh) &= \overline{\mathrm{rel}}\bigl(\{w,z\}, sh\bigr) \cup \mathrm{rel}\bigl(\{w\}, sh\bigr)\\ &= \{v\} \cup \{wz\}\\ &= sh \setminus \{xz, z\}.\end{aligned}$$

  An intuitive explanation of the usefulness of this operator is deferred until after the introduction of the abstract mgu operator (see also Example 31).

We now introduce the abstract mgu operator, specifying how a single binding affects each component of the domain SFL in the context of a syntactic equality theory T.

Definition 28
(amgu_s) The function $\mathrm{amgu}_s : \mathit{SFL} \times \mathit{Bind} \to \mathit{SFL}$ captures the effects of a binding on an element of *SFL*. Let $d = \langle sh, f, l\rangle \in \mathit{SFL}$ and $(x \mapsto t) \in \mathit{Bind}$, where $\{x\} \cup \mathrm{vars}(t) \subseteq \mathit{VI}$. Let also

$$sh' \stackrel{\mathrm{def}}{=} \mathrm{cyclic}_x^t(sh_- \cup sh''),$$

where

$$sh_x \stackrel{\mathrm{def}}{=} \mathrm{rel}\bigl(\{x\}, sh\bigr), \qquad sh_t \stackrel{\mathrm{def}}{=} \mathrm{rel}\bigl(\mathrm{vars}(t), sh\bigr),$$
$$sh_{xt} \stackrel{\mathrm{def}}{=} sh_x \cap sh_t, \qquad sh_- \stackrel{\mathrm{def}}{=} \overline{\mathrm{rel}}\bigl(\{x\} \cup \mathrm{vars}(t), sh\bigr),$$

$$sh'' \stackrel{\mathrm{def}}{=} \begin{cases} \mathrm{bin}(sh_x, sh_t), & \text{if } \mathrm{free}_d(x) \vee \mathrm{free}_d(t); \\ \mathrm{bin}\bigl(sh_x \cup \mathrm{bin}(sh_x, sh_{xt}^\star), \\ \qquad sh_t \cup \mathrm{bin}(sh_t, sh_{xt}^\star)\bigr), & \text{if } \mathrm{lin}_d(x) \wedge \mathrm{lin}_d(t); \\ \mathrm{bin}(sh_x^\star, sh_t), & \text{if } \mathrm{lin}_d(x); \\ \mathrm{bin}(sh_x, sh_t^\star), & \text{if } \mathrm{lin}_d(t); \\ \mathrm{bin}(sh_x^\star, sh_t^\star), & \text{otherwise.} \end{cases}$$

Letting $S_x \stackrel{\mathrm{def}}{=} \mathrm{share\_with}_d(x)$ and $S_t \stackrel{\mathrm{def}}{=} \mathrm{share\_with}_d(t)$, we also define

$$f' \stackrel{\mathrm{def}}{=} \begin{cases} f, & \text{if } \mathrm{free}_d(x) \wedge \mathrm{free}_d(t); \\ f \setminus S_x, & \text{if } \mathrm{free}_d(x); \\ f \setminus S_t, & \text{if } \mathrm{free}_d(t); \\ f \setminus (S_x \cup S_t), & \text{otherwise;} \end{cases}$$

$$l' \stackrel{\mathrm{def}}{=} \bigl(\mathit{VI} \setminus \mathrm{vars}(sh')\bigr) \cup f' \cup l'',$$

where

$$l'' \stackrel{\mathrm{def}}{=} \begin{cases} l \setminus (S_x \cap S_t), & \text{if } \mathrm{lin}_d(x) \wedge \mathrm{lin}_d(t); \\ l \setminus S_x, & \text{if } \mathrm{lin}_d(x); \\ l \setminus S_t, & \text{if } \mathrm{lin}_d(t); \\ l \setminus (S_x \cup S_t), & \text{otherwise.} \end{cases}$$

Then

$$\mathrm{amgu}_s\bigl(d, x \mapsto t\bigr) \stackrel{\mathrm{def}}{=} \begin{cases} \bot_s, & \text{if } d = \bot_s \vee \bigl(T = \mathcal{FT} \wedge x \in \mathrm{vars}(t)\bigr); \\ \langle sh', f', l'\rangle & \text{otherwise.} \end{cases}$$

The next result states that the abstract mgu operator is a correct approximation of the concrete one.

Theorem 29
Let $d \in \mathit{SFL}$ and $(x \mapsto t) \in \mathit{Bind}$, where $\{x\} \cup \mathrm{vars}(t) \subseteq \mathit{VI}$. Then, for all $\sigma \in \gamma_s(d)$ and $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$ in the syntactic equality theory T, we have $\tau \in \gamma_s\bigl(\mathrm{amgu}_s(d, x \mapsto t)\bigr)$.

We now highlight the similarities and differences of the operator amgu_s with respect to the corresponding ones defined in the "classical" proposals for the integration of set-sharing with freeness and linearity, such as Bruynooghe *et al.* (1994a, 1995), Hans and Winkler (1992) and Langen (1990). Note that, when comparing our domain with the proposal in Bruynooghe *et al.* (1994a), we deliberately ignore all those enhancements that depend on properties that cannot be represented in *SFL* (i.e. compoundness and explicit structural information).

- In the computation of the set-sharing component, the main difference can be observed in the second, third and fourth cases of the definition of sh'': here we omit one of the star-unions even when the terms x and t possibly share. In contrast, in Bruynooghe *et al.* (1994a, 1995), Hans and Winkler (1992) and Langen (1990), the corresponding star-union is avoided only when $\mathrm{ind}_d(x,t)$ holds. Note that when $\mathrm{ind}_d(x,t)$ holds in the second case of sh'', then we have $sh_{xt} = \emptyset$; thus, the whole computation for this case reduces to $sh'' = \mathrm{bin}(sh_x, sh_t)$, as was the case in the previous proposals.
- Another improvement on the set-sharing component can be observed in the definition of sh': the cyclic_x^t operator allows the set-sharing description to be further enhanced when dealing with *explicitly cyclic bindings*, i.e. when $x \in \mathrm{vars}(t)$. This is the rewording of a similar enhancement proposed in Bagnara (1997) for the domain *Pos* in the context of groundness analysis. Its net effect is to recover some groundness and sharing dependencies that would have been unnecessarily lost when using the standard operators. When $x \notin \mathrm{vars}(t)$, we have $\mathrm{cyclic}_x^t(sh_- \cup sh'') = sh_- \cup sh''$.
- The computation of the freeness component f' is the same as specified in Bruynooghe *et al.* (1994a) and Hans and Winkler (1992) and is more precise than the one defined in Langen (1990).
- The computation of the linearity component l' is the same as specified in Bruynooghe *et al.* (1994a), and is more precise than those defined in Hans and Winkler (1992) and Langen (1990).

In the following examples we show that the improvements in the abstract computation of the sharing component allow, in particular cases, to derive better information than that obtainable by using the classical abstract unification operators.

Example 30
Let $VI = \{x, x_1, x_2, y, y_1, y_2, z\}$ and $\sigma \in RSubst$ such that

$$\sigma \stackrel{\mathrm{def}}{=} \{x \mapsto f(x_1, x_2, z), y \mapsto f(y_1, z, y_2)\}.$$

By Definition 23, we have $d \stackrel{\mathrm{def}}{=} \alpha_s(\{\sigma\}) = \langle sh, f, l\rangle$, where

$$sh = \{xx_1, xx_2, xyz, yy_1, yy_2\}, \qquad f = VI \setminus \{x, y\}, \qquad l = VI.$$

Consider the binding $(x \mapsto y) \in Bind$. In the concrete domain, we compute (a substitution equivalent to) $\tau \in \mathrm{mgs}(\sigma \cup \{x = y\})$, where

$$\tau = \{x \mapsto f(y_1, y_2, y_2), y \mapsto f(y_1, y_2, y_2), x_1 \mapsto y_1, x_2 \mapsto y_2, z \mapsto y_2\}.$$

Note that $\alpha_s(\{\tau\}) = \langle sh_\tau, f_\tau, l_\tau \rangle$, where $sh_\tau = \{xx_1yy_1, xx_2yy_2z\}$, so that the pairs of variables $P_x = \{x_1, x_2\}$ and $P_y = \{y_1, y_2\}$ keep their independence.

When evaluating the sharing component of $\mathrm{amgu}_s(d, x \mapsto y)$, using the notation of Definition 28, we have

$$sh_x = \{xx_1, xx_2, xyz\}, \qquad sh_t = \{xyz, yy_1, yy_2\},$$
$$sh_{xt} = \{xyz\}, \qquad sh_- = \emptyset.$$

Since both $\mathrm{lin}_d(x)$ and $\mathrm{lin}_d(y)$ hold, we apply the second case of the definition of sh'' so that

$$\begin{aligned}
sh_x \cup \mathrm{bin}(sh_x, sh_{xt}^\star) &= \{xx_1, xx_1yz, xx_2, xx_2yz, xyz\},\\
sh_t \cup \mathrm{bin}(sh_t, sh_{xt}^\star) &= \{xyy_1z, xyy_2z, xyz, yy_1, yy_2\},\\
sh'' &= \mathrm{bin}\bigl(sh_x \cup \mathrm{bin}(sh_x, sh_{xt}^\star), sh_t \cup \mathrm{bin}(sh_t, sh_{xt}^\star)\bigr)\\
&= \{xx_1yy_1, xx_1yy_1z, xx_1yy_2, xx_1yy_2z, xx_1yz,\\
&\qquad xx_2yy_1, xx_2yy_1z, xx_2yy_2, xx_2yy_2z, xx_2yz,\\
&\qquad xyy_1z, xyy_2z, xyz\}.
\end{aligned}$$

Finally, as the binding is not cyclic, we obtain $sh' = sh''$. Thus amgu_s captures the fact that pairs P_x and P_y keep their independence.

In contrast, since $\mathrm{ind}_d(x, y)$ does not hold, all of the classical definitions of abstract unification would have required the star-closure of both sh_x and sh_t, resulting in an abstract element including, among the others, the sharing group $S = \{x, x_1, x_2, y, y_1, y_2\}$. Since $P_x \cup P_y \subset S$, this independence information would have been unnecessarily lost.

Similar examples can be devised for the third and fourth cases of the definition of sh'', where only one side of the binding is known to be linear. The next example shows the precision improvements arising from the use of the cyclic_x^t operator.

Example 31

Let $VI = \{x, x_1, x_2, y\}$ and $\sigma \stackrel{\mathrm{def}}{=} \{x \mapsto f(x_1, x_2)\}$. By Definition 23, we have $d \stackrel{\mathrm{def}}{=} \alpha_s(\{\sigma\}) = \langle sh, f, l \rangle$, where

$$sh = \{xx_1, xx_2, y\}, \qquad f = VI \setminus \{x\}, \qquad l = VI.$$

Let $t = f(x, y)$ and consider the cyclic binding $(x \mapsto t) \in Bind$. In the concrete domain, we compute (a substitution equivalent to) $\tau \in \mathrm{mgs}(\sigma \cup \{x = t\})$, where

$$\tau = \{x \mapsto f(x_1, x_2), x_1 \mapsto f(x_1, x_2), y \mapsto x_2, \}.$$

Note that if we further instantiate τ by grounding y, then variables x, x_1 and x_2 would become ground too. Formally we have $\alpha_s(\{\tau\}) = \langle sh_\tau, f_\tau, l_\tau \rangle$, where $sh_\tau = \{xx_1x_2y\}$. Thus, as observed above, y covers x, x_1 and x_2. When abstractly evaluating the binding, we compute

$$sh_x = \{xx_1, xx_2\}, \qquad sh_t = \{xx_1, xx_2, y\},$$
$$sh_{xt} = sh_x, \qquad sh_- = \emptyset.$$

Since both $\mathrm{lin}_d(x)$ and $\mathrm{lin}_d(t)$ hold, we apply the second case of the definition of sh'', so that

$$\begin{aligned}
sh_x \cup \mathrm{bin}(sh_x, sh^\star_{xt}) &= sh^\star_x = \{xx_1, xx_1x_2, xx_2\},\\
sh_t \cup \mathrm{bin}(sh_t, sh^\star_{xt}) &= \{xx_1, xx_1x_2, xx_1x_2y, xx_1y, xx_2, xx_2y, y\},\\
sh'' &= \mathrm{bin}\bigl(sh_x \cup \mathrm{bin}(sh_x, sh^\star_{xt}), sh_t \cup \mathrm{bin}(sh_t, sh^\star_{xt})\bigr)\\
&= \{xx_1, xx_1x_2, xx_1x_2y, xx_1y, xx_2, xx_2y\}.
\end{aligned}$$

Thus, as $x \in \mathrm{vars}(t)$, we obtain

$$\begin{aligned}
sh' &= \mathrm{cyclic}^t_x(sh_- \cup sh'')\\
&= \overline{\mathrm{rel}}\bigl(\{x\} \cup \mathrm{vars}(t), sh''\bigr) \cup \mathrm{rel}\bigl(\mathrm{vars}(t) \setminus \{x\}, sh''\bigr)\\
&= \varnothing \cup \mathrm{rel}\bigl(\{y\}, sh''\bigr)\\
&= \{xx_1x_2y, xx_1y, xx_2y\}.
\end{aligned}$$

Note that, in the element $sh_- \cup sh'' = sh''$ (which is the abstract element that would have been computed when not exploiting the cyclic^t_x operator) variable y covers none of variables x, x_1 and x_2. Thus, by applying the cyclic^t_x operator, this covering information is restored.

The full abstract unification operator aunify_s, capturing the effect of a sequence of bindings on an abstract element, can now be specified by a straightforward inductive definition using the operator amgu_s.

Definition 32
(aunify_s**)** The operator $\mathrm{aunify}_s : SFL \times Bind^* \to SFL$ is defined, for each $d \in SFL$ and each sequence of bindings $bs \in Bind^*$, by

$$\mathrm{aunify}_s(d, bs) \stackrel{\mathrm{def}}{=} \begin{cases} d, & \text{if } bs = \epsilon;\\ \mathrm{aunify}_s\bigl(\mathrm{amgu}_s(d, x \mapsto t), bs'\bigr), & \text{if } bs = (x \mapsto t) \,.\, bs'. \end{cases}$$

Note that the second argument of aunify_s is a *sequence* of bindings (i.e. it is not a substitution, which is a *set* of bindings), because amgu_s is neither commutative nor idempotent, so that the multiplicity and the actual order of application of the bindings can influence the overall result of the abstract computation. The correctness of the aunify_s operator is simply inherited from the correctness of the underlying amgu_s operator. In particular, any reordering of the bindings in the sequence bs still results in a correct implementation of aunify_s.

The 'merge-over-all-path' operator on the domain SFL is provided by alub_s and is correct by definition. Finally, we define the abstract existential quantification operator for the domain SFL, whose correctness does not pose any problem.

Definition 33
($\mathrm{aexists}_s$**)** The function $\mathrm{aexists}_s : SFL \times \wp_{\mathrm{f}}(VI) \to SFL$ provides the *abstract existential quantification* of an element with respect to a subset of the variables of interest. For each $d \stackrel{\mathrm{def}}{=} \langle sh, f, l\rangle \in SFL$ and $V \subseteq VI$,

$$\mathrm{aexists}_s\bigl(\langle sh, f, l\rangle, V\bigr) \stackrel{\mathrm{def}}{=} \langle \mathrm{aexists}(sh, V), f \cup V, l \cup V\rangle.$$

The intuition behind the definition of the abstract operator $\mathrm{aexists}_s$ is the following. As explained in section 2, any substitution $\sigma \in RSubst$ can be interpreted, under the given equality theory T, as a first-order logical formula; thus, for each set of variables V, it is possible to consider the (concrete) existential quantification $\exists V \mathrel{.} \sigma$. The goal of the abstract operator $\mathrm{aexists}_s$ is to provide a correct approximation of such a quantification starting from any correct approximation for σ.

Example 34
Let $VI = \{x, y, z\}$ and $\sigma = \{x \mapsto f(v_1, v_2), y \mapsto g(v_2, v_3), z \mapsto f(v_1, v_1)\}$, so that, by Definition 23,

$$d = \alpha_s\bigl(\{\sigma\}\bigr) = \langle\{xy, xz, y\}, \varnothing, \{x, y\}\rangle.$$

Let $V = \{y, z\}$ and consider the concrete element corresponding to the logical formula $\exists V \mathrel{.} \sigma$. Note that $T \vdash \forall(\tau \leftrightarrow \exists V \mathrel{.} \sigma)$, where $\tau = \{x \mapsto f(v_1, v_2)\}$. By applying Definition 33, we obtain

$$\mathrm{aexists}_s(d, V) = \langle\{x, y, z\}, \{y, z\}, \{x, y, z\}\rangle = \alpha_s\bigl(\{\tau\}\bigr).$$

It is worth stressing that such an operator does not affect the set VI of the variables of interest. In particular, the abstract element $\mathrm{aexists}_s(d, V)$ still has to provide correct information about variables y and z. Intuitively, since all the occurrences of y and z in $\exists V \mathrel{.} \sigma$ are bound by the existential quantifier, the two variables of interest are un-aliased, free and linear.

Note that an abstract *projection* operator, i.e. an operator that actually modifies the set of variables of interest, is easily specified by composing the operator $\mathrm{aexists}_s$ with an operator that simply removes, from all the components of *SFL and* from the set of variables of interest VI, those variables that have to be projected out.

4 A formal comparison between *SFL* and ASub

As we have already observed, Example 30 shows that the abstract domain *SFL*, when equipped with the abstract mgu operator introduced in section 3.2, can yield results that are strictly more precise than all the classical combinations of set-sharing with freeness and linearity information. In this section we show that the same example has another interesting, unexpected consequence, since it can be used to formally prove that all the classical combinations of set-sharing with freeness and linearity, including those presented in Bagnara *et al.* (2000), Bruynooghe *et al.* (1994a), Hans and Winkler (1992) and Langen (1990), are not *uniformly* more precise than the abstract domain ASub (Sondergaard, 1986), which is based on pair-sharing.

To formalize the above observation, we now introduce the ASub domain and the corresponding abstract semantics operators as specified in Codish *et al.* (1991). The elements of the abstract domain ASub have two components: the first one is a set of variables that are known to be definitely ground; the second one encodes both possible pair-sharing and possible non-linearity into a single relation defined on the set of variables. Intuitively, when $x \neq y$ and $(x, y) \in VI^2$ occurs in the second component, then x and y may share a variable; when $(x, x) \in VI^2$ occurs in the second component, then x may be non-linear. The second component always encodes a symmetric relation; thus, for notational convenience and without any loss of generality (King, 2000), we will represent each pair (x, y) in such a relation as the sharing group $S = \{x, y\}$, which will have cardinality 1 or 2, depending on whether $x = y$ or not, respectively.

Definition 35
(The domain $\mathsf{ASub}_\perp$) The abstract domain $\mathsf{ASub}_\perp$ is defined as $\mathsf{ASub}_\perp \stackrel{\mathrm{def}}{=} \{\perp_{\mathsf{ASub}}\} \cup \mathsf{ASub}$, where

$$\mathsf{ASub} \stackrel{\mathrm{def}}{=} \left\{ \langle G, R \rangle \in \wp(VI) \times SH \;\middle|\; \begin{array}{l} G \cap \mathrm{vars}(R) = \varnothing, \\ \forall S \in R : 1 \leqslant \#S \leqslant 2 \end{array} \right\}.$$

For $i \in \{1, 2\}$, let $\kappa_i = \langle G_i, R_i \rangle \in \mathsf{ASub}$. Then

$$\kappa_1 \preceq_{\mathsf{ASub}} \kappa_2 \quad \stackrel{\mathrm{def}}{\iff} \quad G_1 \supseteq G_2 \wedge R_1 \subseteq R_2.$$

The partial order $\preceq_{\mathsf{ASub}}$ is extended on $\mathsf{ASub}_\perp$ by letting $\perp_{\mathsf{ASub}}$ be the bottom element.

Let $u, v \in VI$ and $\kappa = \langle G, R \rangle \in \mathsf{ASub}$. Then $u \stackrel{\kappa}{\longleftrightarrow} v$ is a shorthand for the condition $\{u, v\} \in R$, whereas $u \stackrel{\kappa}{\Longleftrightarrow} v$ is a shorthand for $u = v \vee \{u, v\} \in R$.

It is well-known that the domain $\mathsf{ASub}_\perp$ can be obtained by a further abstraction of any domain such as SFL that is based on set-sharing and enhanced with linearity information. The following definition formalizes this abstraction.

Definition 36
($\alpha_{\mathsf{ASub}} : SFL \to \mathsf{ASub}_\perp$) Let $d = \langle sh, f, l \rangle \in SFL$. Then

$$\alpha_{\mathsf{ASub}}(d) \stackrel{\mathrm{def}}{=} \begin{cases} \perp_{\mathsf{ASub}}, & \text{if } d = \perp_S; \\ \langle G, R \rangle, & \text{otherwise}; \end{cases}$$

where

$$\begin{aligned} G &\stackrel{\mathrm{def}}{=} \{\, x \in VI \mid x \notin \mathrm{vars}(sh) \,\}, \\ R &\stackrel{\mathrm{def}}{=} \{\, \{x\} \subseteq VI \mid x \in \mathrm{vars}(sh) \wedge x \notin l \,\} \\ &\qquad \cup \{\, \{x, y\} \subseteq VI \mid x \neq y \wedge \exists S \in sh \,.\, \{x, y\} \subseteq S \,\}. \end{aligned}$$

The definition of abstract unification in Codish *et al.* (1991) is based on a few auxiliary operators. The first of these introduces the concept of abstract multiplicity for a term under a given abstract substitution, therefore modeling the notion of definite groundness and definite linearity.

Definition 37
(Abstract multiplicity) Let $\kappa = \langle G, R \rangle \in \mathsf{ASub}$ and let $t \in HTerms$ be a term such that $\mathrm{vars}(t) \subseteq VI$. We say that $y \in \mathrm{vars}(t)$ *occurs linearly (in t) in κ* if and only if $\mathrm{occ\_lin}_\kappa : VI \times HTerms \to Bool$ holds for (y, t), where

$$\mathrm{occ\_lin}_\kappa(y, t) \stackrel{\mathrm{def}}{=} y \in G \vee \bigl(\mathrm{occ\_lin}(y, t) \wedge \forall z \in \mathrm{vars}(t) : \{y, z\} \notin R\bigr).$$

We say that t *has abstract multiplicity* m *in* κ if and only if $\chi_\kappa(t) = m$, where $\chi_\kappa : \mathit{HTerms} \to \{0,1,2\}$ is defined as follows:

$$\chi_\kappa(t) \stackrel{\text{def}}{=} \begin{cases} 0, & \text{if } \mathrm{vars}(t) \subseteq G; \\ 1, & \text{if } \forall y \in \mathrm{vars}(t) : \mathrm{occ\_lin}_\kappa(y,t); \\ 2, & \text{otherwise.} \end{cases}$$

For any binding $x \mapsto t$, the function $\chi_\kappa : \mathit{Bind} \to \{0\} \cup \{1,2\}^2$ is defined as follows

$$\chi_\kappa(x \mapsto t) \stackrel{\text{def}}{=} \begin{cases} 0, & \text{if } \chi_\kappa(x) = 0 \text{ or } \chi_\kappa(t) = 0; \\ \bigl(\chi_\kappa(x), \chi_\kappa(t)\bigr), & \text{otherwise.} \end{cases}$$

It is worth noting that, modulo a few insignificant differences in notation, the multiplicity operator χ_κ defined above corresponds to the abstract multiplicity operator $\chi^{\mathscr{A}}$, which was introduced in Codish *et al.* (1991, Definition 3.4) and provided with an executable specification in King (2000, Definition 4.3). Similarly, the next definition corresponds to Codish *et al.* (1991, Definition 4.3).

Definition 38

(Sharing caused by an abstract equation) For each $\kappa \in \mathsf{ASub}$ and $(x \mapsto t) \in \mathit{Bind}$, where $V_x = \{x\}$ and $V_t = \mathrm{vars}(t)$ are such that $V_x \cup V_t \subseteq \mathit{VI}$, the function $\mathrm{soln} : \mathsf{ASub} \times \mathit{Bind} \to \mathsf{ASub}$ is defined as follows

$$\mathrm{soln}(\kappa, x \mapsto t) \stackrel{\text{def}}{=} \begin{cases} \langle V_x \cup V_t, \varnothing \rangle, & \text{if } \chi_\kappa(x \mapsto t) = 0; \\ \langle \varnothing, \mathrm{bin}(V_x, V_t) \rangle, & \text{if } \chi_\kappa(x \mapsto t) = (1,1); \\ \langle \varnothing, \mathrm{bin}(V_x, V_x \cup V_t) \rangle, & \text{if } \chi_\kappa(x \mapsto t) = (1,2); \\ \langle \varnothing, \mathrm{bin}(V_x \cup V_t, V_t) \rangle, & \text{if } \chi_\kappa(x \mapsto t) = (2,1); \\ \langle \varnothing, \mathrm{bin}(V_x \cup V_t, V_x \cup V_t) \rangle, & \text{if } \chi_\kappa(x \mapsto t) = (2,2); \end{cases}$$

where the function $\mathrm{bin} : \wp(\mathit{VI})^2 \to \mathit{SH}$, for each $V, W \subseteq \mathit{VI}$, is defined as follows

$$\mathrm{bin}(V, W) \stackrel{\text{def}}{=} \bigl\{\, \{v, w\} \subseteq \mathit{VI} \bigm| v \in V, w \in W \,\bigr\}.$$

The next definition corresponds to Codish *et al.* (1991, Definition 4.5).

Definition 39

(Abstract composition) Let $\kappa, \kappa' \in \mathsf{ASub}$, where $\kappa = \langle G, R \rangle$ and $\kappa' = \langle G', R' \rangle$. Then $\kappa \circ \kappa' \stackrel{\text{def}}{=} \langle G'', R'' \rangle$, where

$$G'' \stackrel{\text{def}}{=} G \cup G',$$

$$R'' \stackrel{\text{def}}{=} \left\{ \{u, v\} \in \mathit{SH} \;\middle|\; \begin{array}{l} \{u, v\} \cap G'' = \varnothing, \\ \bigl(u \stackrel{\kappa}{\longleftrightarrow} v\bigr) \lor \bigl(\exists x, y \,.\, u \stackrel{\kappa}{\Longleftrightarrow} x \stackrel{\kappa'}{\longleftrightarrow} y \stackrel{\kappa}{\Longleftrightarrow} v\bigr) \end{array} \right\}.$$

We are now ready to define the abstract mgu operator for the domain $\mathsf{ASub}_\bot$. This operator can be viewed as a specialization of Codish *et al.* (1991, Definition 4.6) for the case when we have to abstract a single binding.

Definition 40
(Abstract mgu **for** $\mathsf{ASub}_\perp$**)** Let $\kappa \in \mathsf{ASub}_\perp$ and $(x \mapsto t) \in \mathit{Bind}$, where $\{x\} \cup \mathrm{vars}(t) \subseteq \mathit{VI}$. Then

$$\mathrm{amgu}_{\mathsf{ASub}}(\kappa, x \mapsto t) \stackrel{\mathrm{def}}{=} \begin{cases} \perp_{\mathsf{ASub}}, & \text{if } \kappa = \perp_{\mathsf{ASub}}; \\ \kappa \circ \mathrm{soln}(\kappa, x \mapsto t), & \text{otherwise.} \end{cases}$$

By repeating the abstract computation of Example 30 on the domain ASub, we provide a formal proof that all the classical approaches based on set-sharing are not uniformly more precise than the pair-sharing domain ASub.

Example 41
Consider the substitutions $\sigma, \tau \in \mathit{RSubst}$ and the abstract element $d \in \mathit{SFL}$ as introduced in Example 30.

By Definition 36, we obtain $\kappa = \alpha_{\mathsf{ASub}}(d) = \langle \varnothing, R \rangle$, where

$$R = \{xx_1, xx_2, xy, xz, yy_1, yy_2, yz\}.$$

When abstractly evaluating the binding $x \mapsto y$ according to Definition 40, we compute the following:

$$\begin{aligned} \chi_\kappa(x \mapsto y) &= (1, 1), \\ \mathrm{soln}(\kappa, x \mapsto y) &= \langle \varnothing, \{xy\} \rangle, \\ \mathrm{amgu}_{\mathsf{ASub}}(\kappa, x \mapsto y) &= \kappa \circ \mathrm{soln}(\kappa, x \mapsto y) = \langle \varnothing, R'' \rangle, \end{aligned}$$

where

$$R'' = R \cup \{x, xy_1, xy_2, x_1y, x_1y_1, x_1y_2, x_1z, x_2y, x_2y_1, x_2y_2, x_2z, y, y_1z, y_2z, z\}.$$

Note that $\{x_1, x_2\} \notin R''$ and $\{y_1, y_2\} \notin R''$, so that these pairs of variables keep their independence. In contrast, as observed in Example 30, the operators in Bagnara *et al.* (2000), Bruynooghe *et al.* (1994a), Hans and Winkler (1992) and Langen (1990) will fail to preserve the independence of these pairs.

We now show that the abstract domain *SFL*, when equipped with the operators introduced in section 3.2, is uniformly more precise than the domain ASub. In particular, the following theorem states that the abstract operator amgu_S of Definition 28 is uniformly more precise than the abstract operator $\mathrm{amgu}_{\mathsf{ASub}}$.

Theorem 42
Let $d \in \mathit{SFL}$ and $\kappa \in \mathsf{ASub}_\perp$ be such that $\alpha_{\mathsf{ASub}}(d) \preceq_{\mathsf{ASub}} \kappa$. Let also $(x \mapsto t) \in \mathit{Bind}$, where $\{x\} \cup \mathrm{vars}(t) \subseteq \mathit{VI}$. Then

$$\alpha_{\mathsf{ASub}}\bigl(\mathrm{amgu}_S(d, x \mapsto t)\bigr) \preceq_{\mathsf{ASub}} \mathrm{amgu}_{\mathsf{ASub}}(\kappa, x \mapsto t).$$

Similar results can be stated for the other abstract operators, such as the abstract existential quantification $\mathrm{aexists}_S$ and the merge-over-all-path operator alub_S. It is worth stressing that, when sequences of bindings come into play, the specification provided in Codish *et al.* (1991, Definition 4.7) requires that the *grounding* bindings (i.e. those bindings such that $\chi_\kappa(x \mapsto t) = 0$) are evaluated before the non-grounding ones. Clearly, if we want to lift the result of Theorem 42 so that it also applies to the operator aunify_S, the same evaluation strategy has to be adopted when computing on the domain SFL; this improvement is well-known (Langen, 1990, pp. 66–67) and already exploited in most implementations of sharing analysis (Bagnara *et al.*, 2000).

5 SFL_2: Eliminating redundancies

As done in Bagnara *et al.* (2002) and Zaffanella *et al.* (2002) for the plain set-sharing domain SH, even when considering the richer domain SFL it is natural to question whether it contains redundancies with respect to the computation of the observable properties.

It is worth stressing that the results presented in Bagnara *et al.* (2002) and Zaffanella *et al.* (2002) cannot be simply inherited by the new domain. The concept of "redundancy" depends on both the starting domain and the given observables: in the SFL domain both of these have changed. First, as can be seen by looking at the definition of amgu_S, freeness and linearity positively interact in the computation of sharing information: *a priori* it is an open issue whether or not the "redundant" sharing groups can play a role in such an interaction. Secondly, since freeness and linearity information can be themselves usefully exploited in a number of applications of static analysis (e.g. in the optimized implementation of concrete unification or in occurs-check reduction), these properties have to be included in the observables.

We will now show that the domain SFL can be simplified by applying the same notion of redundancy as identified in Bagnara *et al.* (2002). Namely, in the definition of SFL it is possible to replace the set-sharing component SH by PSD without affecting the precision on groundness, independence, freeness and linearity. In order to prove such a claim, we now formalize the new observable properties.

Definition 43

(The observables of SFL) The (overloaded) *groundness* and *independence* observables $\rho_{Con}, \rho_{PS} \in \mathrm{uco}(SFL)$ are defined, for each $\langle sh, f, l\rangle \in SFL$, by

$$\rho_{Con}\big(\langle sh, f, l\rangle\big) \stackrel{\mathrm{def}}{=} \langle \rho_{Con}(sh), \varnothing, \varnothing\rangle,$$
$$\rho_{PS}\big(\langle sh, f, l\rangle\big) \stackrel{\mathrm{def}}{=} \langle \rho_{PS}(sh), \varnothing, \varnothing\rangle;$$

the *freeness* and *linearity* observables $\rho_F, \rho_L \in \mathrm{uco}(SFL)$ are defined, for each $\langle sh, f, l\rangle \in SFL$, by

$$\rho_F\big(\langle sh, f, l\rangle\big) \stackrel{\mathrm{def}}{=} \langle SG, f, \varnothing\rangle,$$
$$\rho_L\big(\langle sh, f, l\rangle\big) \stackrel{\mathrm{def}}{=} \langle SG, \varnothing, l\rangle.$$

The overloading of ρ_{PSD} working on the domain SFL is the straightforward extension of the corresponding operator on SH: in particular, the freeness and linearity components are left untouched.

Definition 44

(Non-redundant *SFL*) For each $\langle sh, f, l\rangle \in SFL$, the operator $\rho_{PSD} \in \mathrm{uco}(SFL)$ is defined by

$$\rho_{PSD}\big(\langle sh, f, l\rangle\big) \stackrel{\mathrm{def}}{=} \langle \rho_{PSD}(sh), f, l\rangle.$$

This operator induces the lattice $SFL_2 \stackrel{\mathrm{def}}{=} \rho_{PSD}(SFL)$.

As proved in Zaffanella *et al.* (2002), we have that $\rho_{PSD} \sqsubseteq (\rho_{Con} \sqcap \rho_{PS})$; by the above definitions, it is also clear that $\rho_{PSD} \sqsubseteq (\rho_F \sqcap \rho_L)$; thus, ρ_{PSD} is more precise than the reduced product $(\rho_{Con} \sqcap \rho_{PS} \sqcap \rho_F \sqcap \rho_L)$. Informally, this means that the domain SFL_2 is able to *represent* all of our observable properties without precision losses.

The next theorem shows that ρ_{PSD} is a congruence with respect to the aunify_S, alub_S and $\mathrm{aexists}_S$ operators. This means that the domain SFL_2 is able to *propagate* the information on the observables as precisely as *SFL*, therefore providing a completeness result.

Theorem 45

Let $d_1, d_2 \in SFL$ be such that $\rho_{PSD}(d_1) = \rho_{PSD}(d_2)$. Then, for each sequence of bindings $bs \in Bind^*$, for each $d' \in SFL$ and $V \in \wp(VI)$,

$$\begin{aligned} \rho_{PSD}\big(\mathrm{aunify}_S(d_1, bs)\big) &= \rho_{PSD}\big(\mathrm{aunify}_S(d_2, bs)\big), \\ \rho_{PSD}\big(\mathrm{alub}_S(d_1, d')\big) &= \rho_{PSD}\big(\mathrm{alub}_S(d_2, d')\big), \\ \rho_{PSD}\big(\mathrm{aexists}_S(d_1, V)\big) &= \rho_{PSD}\big(\mathrm{aexists}_S(d_2, V)\big). \end{aligned}$$

Finally, by providing the minimality result, we show that the domain SFL_2 is indeed the generalized quotient (Cortesi *et al.*, 1998; Giacobazzi *et al.*, 1998) of *SFL* with respect to the reduced product $(\rho_{Con} \sqcap \rho_{PS} \sqcap \rho_F \sqcap \rho_L)$.

Theorem 46

For each $i \in \{1, 2\}$, let $d_i = \langle sh_i, f_i, l_i\rangle \in SFL$ be such that $\rho_{PSD}(d_1) \neq \rho_{PSD}(d_2)$. Then there exist a sequence of bindings $bs \in Bind^*$ and an observable property $\rho \in \{\rho_{Con}, \rho_{PS}, \rho_F, \rho_L\}$ such that

$$\rho\big(\mathrm{aunify}_S(d_1, bs)\big) \neq \rho\big(\mathrm{aunify}_S(d_2, bs)\big).$$

As far as the implementation is concerned, the results proved in Bagnara *et al.* (2002) for the domain *PSD* can also be applied to SFL_2. In particular, in the definition of amgu_S every occurrence of the star-union operator can be safely replaced by the self-bin-union operator. As a consequence, it is possible to provide an implementation where the time complexity of the amgu_S operator is bounded by a polynomial in the number of sharing groups of the set-sharing component.

The following result provides another optimization that can be applied when both terms x and t are definitely linear, but none of them is definitely free (i.e. when we compute sh'' by the second case stated in Definition 28).

Theorem 47
Let $sh \in SH$ and $(x \mapsto t) \in Bind$, where $\{x\} \cup \mathrm{vars}(t) \subseteq VI$. Let $sh_{-} \stackrel{\mathrm{def}}{=} \overline{\mathrm{rel}}\bigl(\{x\} \cup \mathrm{vars}(t), sh\bigr)$, $sh_x \stackrel{\mathrm{def}}{=} \mathrm{rel}\bigl(\{x\}, sh\bigr)$, $sh_t \stackrel{\mathrm{def}}{=} \mathrm{rel}\bigl(\mathrm{vars}(t), sh\bigr)$, $sh_{xt} \stackrel{\mathrm{def}}{=} sh_x \cap sh_t$, $sh_W \stackrel{\mathrm{def}}{=} \mathrm{rel}(W, sh)$, where $W = \mathrm{vars}(t) \setminus \{x\}$, and

$$sh^{\diamond} \stackrel{\mathrm{def}}{=} \mathrm{bin}\bigl(sh_x \cup \mathrm{bin}(sh_x, sh_{xt}^{\star}), sh_t \cup \mathrm{bin}(sh_t, sh_{xt}^{\star})\bigr).$$

Then it holds

$$\rho_{PSD}\bigl(\mathrm{cyclic}_x^t(sh_{-} \cup sh^{\diamond})\bigr) = \begin{cases} \rho_{PSD}\bigl(sh_{-} \cup \mathrm{bin}(sh_x, sh_t)\bigr), & \text{if } x \notin \mathrm{vars}(t); \\ \rho_{PSD}\bigl(sh_{-} \cup \mathrm{bin}(sh_x^2, sh_W)\bigr), & \text{otherwise.} \end{cases}$$

Therefore, even when terms x and t possibly share (i.e. when $sh_{xt} \neq \emptyset$), by using SFL_2 we can avoid the expensive computation of at least one of the two inner binary unions in the expression for $sh^{\diamond}$.

6 Experimental evaluation

Example 30 shows that an analysis based on the new abstract unification operator can be strictly more precise than one based on the classical proposal. However, that example is artificial and leaves open the question as to whether or not such a phenomenon actually happens during the analysis of real programs and, if so, how often. This was the motivation for the experimental evaluation we describe in this section. We consider the abstract domain $Pos \times SFL_2$ (Bagnara *et al.*, 2001), where the non-redundant version SFL_2 of the domain SFL is further combined, as described in (Bagnara *et al.*, 2001, Section 4), with the definite groundness information computed by *Pos* and compare the results using the (classical) abstract unification operator of Bagnara *et al.* (2002, Definition 4) with the (new) operator amgu_S given in Definition 28. Taking this as a starting point, we experimentally evaluate eight variants of the analysis arising from all possible combinations of the following options:

1. the analysis can be goal independent or goal dependent;
2. the set-sharing component may or may not have widening enabled (Zaffanella *et al.*, 1999);
3. the abstract domain may or may not be upgraded with structural information using the Pattern$(\cdot)$ operator (see Bagnara *et al.* (2000, 2001, Section 5)).

The experiments have been conducted using the China analyzer (Bagnara, 1997) on a GNU/Linux PC system. China is a data-flow analyzer for (constraint) logic programs performing bottom-up analysis and deriving information on both call-patterns and success-patterns by means of program transformations and optimized fixpoint computation techniques. An abstract description is computed for the call- and success-patterns for each predicate defined in the program. The benchmark suite, which is composed of 372 logic programs of various sizes and complexity, can be considered representative.

The precision results for the goal independent comparisons are summarized in Table 1. For each benchmark, precision is measured by counting the number of independent pairs as well as the numbers of definitely ground, free and linear

Table 1. *Classical Pos × SFL₂ versus enhanced one: precision*

Goal Independent	Without Widening				With Widening			
	w/o SI		with SI		w/o SI		with SI	
Prec. class	I	L	I	L	I	L	I	L
$5 < p \leqslant 10$	—	2	—	2	—	2	—	2
$2 < p \leqslant 5$	—	—	—	—	—	—	—	1
$0 < p \leqslant 2$	5	5	9	6	6	6	12	8
same precision	357	355	337	338	366	364	360	361
unknown	10	10	26	26	—	—	—	—

variables detected. For each variant of the analysis, these numbers are then compared by computing the relative precision improvements and expressing them using percentages. The benchmark suite is then partitioned into several precision equivalence classes and the cardinalities of these classes are shown in Table 1. For example, when considering a goal independent analysis without structural information and without widenings, the value 5 found at the intersection of the row labeled '$0 < p \leqslant 2$' with the column labeled 'I' should be read: "for five benchmarks there has been a (positive) increase in the number of independent pairs of variables which is less than or equal to two percent." Note that we only report on independence and linearity (in the columns labeled 'I' and 'L', respectively), because no differences have been observed for groundness and freeness. The precision class labeled 'unknown' identifies those benchmarks for which the analyses timed-out (the time-out threshold was fixed at 600 seconds). Hence, for goal independent analyses, a precision improvement affects from 1.6% to 3% of the benchmarks, depending on the considered variant.

When considering the goal dependent analyses, we obtain a single, small improvement, so that no comparison tables are included here: the improvement, affecting linearity information, can be observed when the abstract domain includes structural information.

With respect to differences in the efficiency, the introduction of the new abstract unification operator has no significant effect on the computation time: small differences (usually improvements) are observed on as many as 6% of the benchmarks for the goal independent analysis without structural information and without widenings; other combinations register even less differences.

We note that it is not surprising that the precision and efficiency improvements occur very rarely since the abstract unification operators behave the same except under very specific conditions: the two terms being unified must not only be definitely linear, but also possibly non-free and share a variable.

7 Related work

Sharing information has been shown to be important for finite-tree analysis (Bagnara *et al.*, 2001). This aims at identifying those program variables that, at a particular program point, cannot be bound to an infinite rational tree (in other words, they are necessarily bound to acyclic terms). This novel analysis is irrelevant for those logic languages computing over a domain of finite trees, while having several applications for those (constraint) logic languages that are explicitly designed to compute over a domain including rational trees, such as Prolog II and its successors (Colmerauer, 1982, 1990), SICStus Prolog (Swedish Institute of Computer Science, Programming Systems Group, 1995), and Oz (Smolka and Treinen, 1994). The analysis specified in Bagnara *et al.* (2001) is based on a parametric abstract domain $H \times P$, where the H component (the Herbrand component) is a set of variables that are known to be bound to finite terms, while the parametric component P can be any domain capturing aliasing, groundness, freeness and linearity information that is useful to compute finite-tree information. An obvious choice for such a parameter is the domain combination *SFL*. It is worth noting that, in Bagnara *et al.* (2001), the correctness of the finite-tree analysis is proved by *assuming* the correctness of the underlying analysis on the parameter P. Thus, thanks to the results shown in this paper, the proof for the domain $H \times SFL$ can now be considered complete.

Codish *et al.* (2001) describe an algebraic approach to the sharing analysis of logic programs that is based on *set logic programs*. A set logic program is a logic program in which the terms are sets of variables and standard unification is replaced by a suitable unification for sets, called *ACI1-unification* (unification in the presence of an associative, commutative, and idempotent equality theory with a unit element). The authors show that the domain of *set-substitutions*, with a few modifications, can be used as an abstract domain for sharing analysis. They also provide an isomorphism between this domain and the set-sharing domain *SH* of Jacobs and Langen. The approach using set logic programs is also generalized to include linearity information, by suitably annotating the set-substitutions, and the authors formally state the optimality of the corresponding abstract unification operator $\mathit{lin\text{-}mgu}_{ACI1}$ (Lemma A.10 in the Appendix of Codish *et al.* (2000)). However, this operator is very similar to the classical combinations of set-sharing with linearity (Bruynooghe *et al.*, 1994a; Hans and Winkler, 1992; Langen, 1990): in particular, the precision improvements arising from this enhancement are only exploited when the two terms being unified are definitely independent. As we have seen in this paper, such a choice results in a sub-optimal abstract unification operator, so that the optimality result cannot hold. By looking at the proof of Lemma A.10 in Codish *et al.* (2000), it can be seen that the case when the two terms possibly share a variable is dealt with by referring to an example:[5] this one is supposed to show that all the possible sharing groups can be generated. However, even our improved operator correctly characterizes the given example, so that the proof is wrong. It should be stressed that the amgu_S operator presented in this paper, though remarkably precise, is not meant to subsume all of the proposals for an improved sharing analysis that appeared in the recent literature (for a thorough experimental evaluation of many of these proposals, the reader is referred elsewhere (Bagnara *et al.*, 2000; Zaffanella,

[5] The proof refers to Example 8, which however has nothing to do with the possibility that the two terms share; we believe that Example 2 was intended.

2001)). In particular, it is not difficult to show that our operator is not the optimal approximation of concrete unification.

In a very recent paper, Howe and King (2003) consider the domain *SFL* and propose three optimizations to improve both the precision and the efficiency of the (classical) abstract unification operator. The first optimization is based on the same observation we have made in this paper, namely that the independence check between the two terms being unified is not necessary for ensuring the correctness of the analysis. However, the proposed enhancement does not fully exploit this observation, so that the resulting operator is strictly less precise than our amgu_s operator (even when the operator cyclic_x^t does not come into play). In fact, the first optimization of Howe and King (2003) is not uniformly more precise than the classical proposals. The following example illustrates this point.

Example 48

Let $VI = \{x, y, z_1, z_2, z_3\}$, $(x \mapsto y) \in \mathit{Bind}$ and $d \stackrel{\mathrm{def}}{=} \langle sh, \varnothing, VI \rangle$, where $sh = \{xz_1, xz_2, xz_3, yz_1, yz_2, yz_3\}$.

Since x and y are linear and independent, amgu_s as well as all the classical abstract unification operators will compute $d_1 = \langle sh_1, \varnothing, \{x, y\} \rangle$, where

$$sh_1 \stackrel{\mathrm{def}}{=} \mathrm{bin}(sh_x, sh_y) = \{xyz_1, xyz_1z_2, xyz_1z_3, xyz_2, xyz_2z_3, xyz_3\}.$$

In contrast, a computation based on (Howe and King,, 2003, Definition 3.2), results in the less precise abstract element $d_2 = \langle sh_2, \varnothing, \{x, y\} \rangle$, where

$$sh_2 \stackrel{\mathrm{def}}{=} \mathrm{bin}(sh_x^\star, sh_y) \cap \mathrm{bin}(sh_x, sh_y^\star) = sh_1 \cup \{xyz_1z_2z_3\}.$$

The second optimization shown in Howe and King (2003) is based on the enhanced combination of set-sharing and freeness information, which was originally proposed in Filé (1994). In particular, the authors propose a slightly different precision enhancement, less powerful as far as precision is concerned, which however seems to be amenable for an efficient implementation. The third optimization in Howe and King (2003) exploits the combination of the domain *SFL* with the groundness domain *Pos*.

8 Conclusion

In this paper we have introduced the abstract domain *SFL*, combining the set-sharing domain *SH* with freeness and linearity information. While the carrier of *SFL* can be considered standard, we have provided the specification of a new abstract unification operator, showing examples where this operator achieves more precision than the classical proposals. The main contributions of this paper are the following:

- we have defined a precise abstraction function, mapping arbitrary substitutions in rational solved form into their *most precise* approximation on *SFL*;
- using this abstraction function, we have provided the mandatory proof of *correctness* for the new abstract unification operator, *for both finite-tree and rational-tree languages*;

- we have formally shown that the domain *SFL* is *uniformly* more precise than the domain ASub; we have also provided an example showing that all the classical approaches to the combinations of set-sharing with freeness and linearity fail to satisfy this property;
- we have shown that, in the definition of *SFL*, we can replace the set-sharing domain *SH* by its non-redundant version *PSD*. As a consequence, it is possible to implement an algorithm for abstract unification running in *polynomial time* and still obtain the same precision on all the considered observables, that is groundness, independence, freeness and linearity.

Acknowledgements

We recognize the hard work required to review technical papers such as this one and would like to express our real gratitude to the Journal referees for their critical reading and constructive suggestions for preparing this improved version.

References

Bagnara, R. 1997. Data-flow analysis for constraint logic-based languages. PhD thesis, Dipartimento di Informatica, Università di Pisa, Pisa, Italy. Report TD-1/97.

Bagnara, R., Gori, R., Hill, P. M. and Zaffanella, E. 2001. Finite-tree analysis for constraint logic-based languages. In: P. Cousot, Ed. *Static Analysis: 8th International Symposium, SAS 2001, Lecture Notes in Computer Science 2126*, pp. 165–184. Springer-Verlag.

Bagnara, R., Hill, P. M. and Zaffanella, E. 1997. Set-sharing is redundant for pair-sharing. In: P. Van Hentenryck, Ed. *Static Analysis: Proceedings of the 4th International Symposium, Lecture Notes in Computer Science 1302*, pp. 53–67. Springer-Verlag.

Bagnara, R., Hill, P. M. and Zaffanella, E. 2000. Efficient structural information analysis for real CLP languages. In: M. Parigot and A. Voronkov, Eds. *Proceedings of the 7th International Conference on Logic for Programming and Automated Reasoning (LPAR 2000), Lecture Notes in Artificial Intelligence 1955*, pp. 189–206. Springer-Verlag.

Bagnara, R., Hill, P. M. and Zaffanella, E. 2002. Set-sharing is redundant for pair-sharing. *Theoretical Computer Science 277*, 1–2, 3–46.

Bagnara, R., Zaffanella, E., Gori, R. and Hill, P. M. 2001. Boolean functions for finite-tree dependencies. In: R. Nieuwenhuis and A. Voronkov, Eds. *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2001), Lecture Notes in Artificial Intelligence 2250*, pp. 579–594. Springer-Verlag.

Bagnara, R., Zaffanella, E. and Hill, P. M. 2000. Enhanced sharing analysis techniques: A comprehensive evaluation. In: M. Gabbrielli and F. Pfenning, Eds. *Proceedings of the 2nd International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming*, pp. 103–114. Association for Computing Machinery, Montreal, Canada.

Bagnara, R., Zaffanella, E. and Hill, P. M. 2001. Enhanced sharing analysis techniques: A comprehensive evaluation. Submitted for publication. (Available at `http://www.cs.unipr.it/~bagnara/`.)

Bruynooghe, M. and Codish, M. 1993. Freeness, sharing, linearity and correctness – All at once. In: P. Cousot, M. Falaschi, G. Filé and A. Rauzy, Eds. *Static Analysis, Proceedings of the Third International Workshop, Lecture Notes in Computer Science 724*, pp. 153–164. Springer-Verlag. (An extended version is available as Technical Report CW 179, Department of Computer Science, K.U. Leuven, September 1993.)

Bruynooghe, M., Codish, M. and Mulkers, A. 1994a. Abstract unification for a composite domain deriving sharing and freeness properties of program variables. In: F. S. de Boer and M. Gabbrielli, Eds. *Verification and Analysis of Logic Languages, Proceedings of the W2 Post-Conference Workshop, International Conference on Logic Programming*, pp. 213–230. Santa Margherita Ligure, Italy.

Bruynooghe, M., Codish, M. and Mulkers, A. 1994b. A composite domain for freeness, sharing, and compoundness analysis of logic programs. Technical Report CW 196, Department of Computer Science, K.U. Leuven, Belgium.

Bruynooghe, M., Codish, M. and Mulkers, A. 1995. Abstracting unification: A key step in the design of logic program analyses. In: J. van Leeuwen, Ed. *Computer Science Today: Recent Trends and Developments, Lecture Notes in Computer Science 1000*, pp. 406–425. Springer-Verlag, Berlin.

Clark, K. L. 1978. Negation as failure. In: H. Gallaire and J. Minker, Eds. *Logic and Databases*, pp. 293–322. Plenum Press, France.

Codish, M., Dams, D., Filé, G. and Bruynooghe, M. 1993. Freeness analysis for logic programs – and correctness? In: D. S. Warren, Ed. *Logic Programming: Proceedings of the Tenth International Conference on Logic Programming*, pp. 116–131. MIT Press Series in Logic Programming. MIT Press. (An extended version is available as Technical Report CW 161, Department of Computer Science, K.U. Leuven, December 1992.)

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

Codish, M., Dams, D. and Yardeni, E. 1991. Derivation and safety of an abstract unification algorithm for groundness and aliasing analysis. *International Conference on Logic Programming*, pp. 79–93.

Codish, M., Lagoon, V. and Bueno, F. 2000. An algebraic approach to sharing analysis of logic programs. *Journal of Logic Programming 42*, 2, 111–149.

Codish, M., Mulkers, A., Bruynooghe, M., Garcìa de la Banda, M. and Hermenegildo, M. 1993. Improving abstract interpretations by combining domains. In: *Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, pp. 194–205. ACM Press. (Also available as Technical Report CW 162, Department of Computer Science, K.U. Leuven, December 1992.)

Codish, M., Mulkers, A., Bruynooghe, M., Garcìa de la Banda, M. and Hermenegildo, M. 1995. Improving abstract interpretations by combining domains. *ACM Transactions on Programming Languages and Systems 17*, 1, 28–44.

Colmerauer, A. 1982. Prolog and infinite trees. In: K. L. Clark and S. Å. Tärnlund, Eds. *Logic Programming, APIC Studies in Data Processing*, Vol. 16, pp. 231–251. Academic Press.

Colmerauer, A. 1984. Equations and inequations on finite and infinite trees. *Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS'84)*, pp. 85–99. Tokyo, Japan.

Colmerauer, A. 1990. An introduction to Prolog-III. *Communications of the ACM 33*, 7, 69–90.

Cortesi, A. and Filé, G. 1999. Sharing is optimal. *Journal of Logic Programming 38*, 3, 371–386.

Cortesi, A., Filé, G. and Winsborough, W. 1998. The quotient of an abstract interpretation for comparing static analyses. *Theoretical Computer Science 202*, 1&2, 163–192.

Cousot, P. and Cousot, R. 1977. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. *Proceedings of the Fourth Annual ACM Symposium on Principles of Programming Languages*, pp. 238–252. ACM Press.

Cousot, P. and Cousot, R. 1979. Systematic design of program analysis frameworks. *Proceedings of the Sixth Annual ACM Symposium on Principles of Programming Languages*, pp. 269–282. ACM Press, New York.

Filé, G. 1994. Share × Free: Simple and correct. Technical Report 15, Dipartimento di Matematica, Università di Padova.

Furukawa, K., Ed. 1991. *Logic Programming: Proceedings of the Eighth International Conference on Logic Programming*. MIT Press Series in Logic Programming. MIT Press.

Giacobazzi, R., Ranzato, F. and Scozzari, F. 1998. Complete abstract interpretations made constructive. In: J. Gruska and J. Zlatuska, Eds. *Proceedings of 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS'98), Lecture Notes in Computer Science 1450*, pp. 366–377. Springer-Verlag.

Hans, W. and Winkler, S. 1992. Aliasing and groundness analysis of logic programs through abstract interpretation and its safety. Technical Report 92-27, Technical University of Aachen (RWTH Aachen).

Hill, P. M., Bagnara, R. and Zaffanella, E. 1998. The correctness of set-sharing. In: G. Levi, Ed. *Static Analysis: Proceedings of the 5th International Symposium, Lecture Notes in Computer Science 1503*, pp. 99–114. Springer-Verlag.

Hill, P. M., Bagnara, R. and Zaffanella, E. 2002. Soundness, idempotence and commutativity of set-sharing. *Theory and Practice of Logic Programming 2*, 2, 155–201.

Hill, P. M., Bagnara, R. and Zaffanella, E. 2003. On the analysis of set-sharing, freeness and linearity for finite and rational tree languages. Technical Report 2003.08, School of Computing, University of Leeds. (Available at `http://www.comp.leeds.ac.uk/research/pubs/reports.shtml`.)

Howe, J. M. and King, A. 2003. Three optimisations for sharing. *Theory and Practice of Logic Programming 3*, 2, 243–257.

Intrigila, B. and Venturini Zilli, M. 1996. A remark on infinite matching vs infinite unification. *Journal of Symbolic Computation 21*, 3, 2289–2292.

Jacobs, D. and Langen, A. 1989. Accurate and efficient approximation of variable aliasing in logic programs. In: E. L. Lusk and R. A. Overbeek, Eds. *Logic Programming: Proceedings of the North American Conference*, pp. 154–16. MIT Press Series in Logic Programming. MIT Press.

Jaffar, J., Lassez, J.-L. and Maher, M. J. 1987. Prolog-II as an instance of the logic programming scheme. In: M. Wirsing, Ed. *Formal Descriptions of Programming Concepts III*, pp. 275–299. North-Holland.

Keisu, T. 1994. Tree constraints. PhD thesis, The Royal Institute of Technology, Stockholm, Sweden. (Also available in the SICS Dissertation Series: SICS/D-16-SE.)

King, A. 1994. A synergistic analysis for sharing and groundness which traces linearity. In: D. Sannella, Ed. *Proceedings of the Fifth European Symposium on Programming, Lecture Notes in Computer Science 788*, pp. 363–378. Springer-Verlag.

King, A. 2000. Pair-sharing over rational trees. *Journal of Logic Programming 46*, 1–2, 139–155.

King, A. and Soper, P. 1994. Depth-k sharing and freeness. In: P. Van Hentenryck, Ed. *Logic Programming: Proceedings of the Eleventh International Conference on Logic Programming*, pp. 553–568. MIT Press Series in Logic Programming. MIT Press.

Langen, A. 1990. Advanced techniques for approximating variable aliasing in logic programs. PhD thesis, Computer Science Department, University of Southern California.

Maher, M. J. 1988. Complete axiomatizations of the algebras of finite, rational and infinite trees. *Proceedings, Third Annual Symposium on Logic in Computer Science*, pp. 348–357. IEEE Press.

Muthukumar, K. and Hermenegildo, M. 1991. Combined determination of sharing and freeness of program variables through abstract interpretation. *International Conference on Logic Programming*, pp. 49–63.

Muthukumar, K. and Hermenegildo, M. 1992. Compile-time derivation of variable dependency using abstract interpretation. *Journal of Logic Programming 13*, 2&3, 315–347.

Smolka, G. and Treinen, R. 1994. Records for logic programming. *Journal of Logic Programming 18*, 3, 229–258.

Søndergaard, H. 1986. An application of abstract interpretation of logic programs: Occur check reduction. In: B. Robinet and R. Wilhelm, Eds. *Proceedings of the 1986 European Symposium on Programming, Lecture Notes in Computer Science 213*, pp. 327–338. Springer-Verlag.

Swedish Institute of Computer Science, Programming Systems Group 1995. *SICStus Prolog User's Manual*, release 3 #0 ed. Swedish Institute of Computer Science, Programming Systems Group.

Zaffanella, E. 2001. Correctness, precision and efficiency in the sharing analysis of real logic languages. PhD thesis, School of Computing, University of Leeds, Leeds, U.K. (Available at `http://www.cs.unipr.it/~zaffanella/`.)

Zaffanella, E., Bagnara, R. and Hill, P. M. 1999. Widening Sharing. In: G. Nadathur, Ed. *Principles and Practice of Declarative Programming, Lecture Notes in Computer Science 1702*, pp. 414–431. Springer-Verlag.

Zaffanella, E., Hill, P. M. and Bagnara, R. 1999. Decomposing non-redundant sharing by complementation. In: A. Cortesi and G. Filé, Eds. *Static Analysis: Proceedings of the 6th International Symposium, Lecture Notes in Computer Science 1694*, pp. 69–84. Springer-Verlag.

Zaffanella, E., Hill, P. M. and Bagnara, R. 2002. Decomposing non-redundant sharing by complementation. *Theory and Practice of Logic Programming 2*, 2, 233–261.