Enhanced Sharing Analysis Techniques: A Comprehensive Evaluation

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

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

Abstract

Sharing, an abstract domain developed by D. Jacobs and A. Langen for the analysis of logic programs, derives useful aliasing information. It is well-known that a commonly used core of techniques, such as the integration of Sharing with freeness and linearity information, can significantly improve the precision of the analysis. However, a number of other proposals for refined domain combinations have been circulating for years. One feature that is common to these proposals is that they do not seem to have undergone a thorough experimental evaluation even with respect to the expected precision gains. In this paper we experimentally evaluate: helping Sharing with the definitely ground variables found using *Pos*, the domain of positive Boolean formulas; the incorporation of explicit structural information; a full implementation of the reduced product of Sharing and *Pos*; the issue of reordering the bindings in the computation of the abstract mgu; an original proposal for the addition of a new mode recording the set of variables that are deemed to be ground or free; a refined way of using linearity to improve the analysis; the recovery of hidden information in the combination of Sharing with freeness information. Finally, we discuss the issue of whether tracking compoundness allows the computation of more sharing information.

KEYWORDS: Abstract Interpretation; Logic Programming; Sharing Analysis; Experimental Evaluation.

1 Introduction

In the execution of a logic program, two variables are *aliased* or *share* at some program point if they are bound to terms that have a common variable. Conversely, two variables are *independent* if they are bound to terms that have no variables in common. Thus by providing information about possible variable aliasing, we also provide information about definite variable independence. In logic programming,

∗ The work of the first and second authors has been partly supported by MURST projects "Certificazione automatica di programmi mediante interpretazione astratta" and "Interpretazione astratta, sistemi di tipo e analisi control-flow."
† This work was partly supported by EPSRC under grant GR/M05645.

a knowledge of the possible aliasing (and hence definite independence) between variables has some important applications.

Information about variable aliasing is essential for the efficient exploitation of AND-parallelism (Bueno et al. 1994; Bueno et al. 1999; Chang et al. 1985; Hermenegildo and Greene 1990; Hermenegildo and Rossi 1995; Jacobs and Langen 1992; Muthukumar and Hermenegildo 1992). Informally, two atoms in a goal are executed in parallel if, by a mixture of compile-time and run-time checks, it can be guaranteed that they do not share any variable. This implies the absence of *binding conflicts* at run-time: it will never happen that the processes associated to the two atoms try to bind the same variable.

Another significant application is *occurs-check reduction* (Crnogorac et al. 1996; Søndergaard 1986). It is well-known that many implemented logic programming languages (e.g., almost all Prolog systems) omit the *occurs-check* from the unification procedure. Occurs-check reduction amounts to identifying the unifications where such an omission is safe, and, for this purpose, information on the possible aliasing of program variables is crucial.

Aliasing information can also be used indirectly in the computation of other interesting program properties. For instance, the precision with which freeness information can be computed depends, in a critical way, on the precision with which aliasing can be tracked (Bruynooghe et al. 1994a; Codish et al. 1993; Filé 1994; King and Soper 1994; Langen 1990; Muthukumar and Hermenegildo 1991).

In addition to these well-known applications, a recent line of research has shown that aliasing information can be exploited in *Inductive Logic Programming* (ILP). Several optimizations have been proposed for speeding up the refinement of inductively defined predicates in ILP systems (Blockeel et al. 2000; Santos Costa et al. 2000). It has been observed that the applicability of some of these optimizations, formulated in terms of syntactic conditions on the considered predicate, could be recast as tests on variable aliasing (Blockeel et al. 2000, Appendix D).

Sharing, a domain due to D. Jacobs and A. Langen (Jacobs and Langen 1989; Jacobs and Langen 1992; Langen 1990), is based on the concept of *set-sharing*. An element of the Sharing domain, which is a set of *sharing-groups* (i.e., a set of sets of variables), represents information on groundness,[1] groundness dependencies, possible aliasing, and more complex *sharing-dependencies* among the variables that are involved in the execution of a logic program (Bagnara et al. 1997; Bagnara et al. 2002; Bueno et al. 1994; Bueno et al. 1999).

Even though Sharing is quite precise, it is well-known that more precision is attainable by combining it with other domains. Nowadays, nobody would seriously consider performing sharing analysis without exploiting the combination of aliasing information with groundness and linearity information. As a consequence, expressions such as 'sharing information', 'sharing domain' and 'sharing analysis' usually capture groundness, aliasing, linearity and quite often also freeness. Notice that

[1] A variable is *ground* if it is bound to a term containing no variables, it is *compound* if it is bound to a non-variable term, it is *free* if it is not compound, it is *linear* if it is bound to a term that does not contain multiple occurrences of a variable.

this idiom is nothing more than a historical accident: as we will see in the sequel, compoundness and other kinds of structural information could also be included in the collective term 'sharing information'.

As argued informally by H. Søndergaard (Søndergaard 1986), linearity information can be suitably exploited to 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 ASub, a sharing domain based on the concept of *pair-sharing* (i.e., aliasing and linearity information is encoded by a set of pairs of variables). A similar integration with linearity for the domain Sharing was proposed by Langen in his PhD thesis (Langen 1990). The synergy attainable from the integration between aliasing and freeness information was pointed out by K. Muthukumar and M. Hermenegildo (Muthukumar and Hermenegildo 1992). Building on these works, W. Hans and S. Winkler (Hans and Winkler 1992) proposed a combined integration of freeness and linearity information with sharing, but small variations (such as the one we will present as the starting point for our work) have been developed by M. Bruynooghe et al. (Bruynooghe and Codish 1993; Bruynooghe et al. 1994a).

There have been a number of other proposals for more refined combinations which have the potential for improving the precision of the sharing analysis over and above that obtainable using the classical combinations of Sharing with linearity and freeness. These include the implementation of more powerful abstract semantic operators (since it is well-known that the commonly used ones are sub-optimal) and/or the integration with other domains. Not one of these proposals seem to have undergone a thorough experimental evaluation, even with respect to the expected precision gains. The goal of this paper is to systematically study these enhancements and provide a uniform theoretical presentation together with an extensive experimental evaluation that will give a strong indication of their impact on the accuracy of the sharing information.

Our investigation is primarily from the point of view of precision. Reasonable efficiency is also clearly of interest but this has to be secondary to the question as to whether precision is significantly improved: only if this is established, should better implementations be researched. One of the investigated enhancements is the integration of explicit structural information in the sharing analysis and an important contribution of this paper is that it shows both the feasibility and the positive impact of this combination.

Note that, regardless of its practicality, any feasible sharing analysis technique that offers good precision may be valuable. While inefficiency may prevent its adoption in production analyzers, it can help in assessing the precision of the more competitive techniques.

The present paper, which is an improved and extended version of (Bagnara et al. 2000), is structured as follows. In Section 2, we define some notation and recall the definitions of the domain Sharing and its standard integration with freeness and linearity information denoted as *SFL*. In Section 3, we briefly describe the China analyzer, the benchmark suite and the methodology we follow in the experimental evaluations. In each of the next seven sections, we describe and experimentally evaluate different enhancements and precision optimizations for the domain *SFL*. Section 4 considers a simple combination of *Pos* with *SFL*; Section 5 investigates the effect of including explicit structural information by means of the Pattern($\cdot$) construction; Section 6 discusses possible heuristics for reordering the bindings so as to maximize the precision of *SFL*; Section 7 studies the implementation of a more precise combination between *Pos* and *SFL*; Section 8 describes a new mode 'ground or free' to be included in *SFL*; Section 9 and Section 10 study the possibility of improving the exploitation of the linearity and freeness information already encoded in *SFL*. In Section 11 we discuss (without an experimental evaluation) whether compoundness information can be useful for precision gains. Section 12 concludes with some final remarks.

2 Preliminaries

For any set S, $\wp(S)$ denotes the powerset of S. For ease of presentation, we assume there is a finite set of variables of interest denoted by VI. If t is a syntactic object then $vars(t)$ and $mvars(t)$ denote the set and the multiset of variables in t, respectively. If a occurs more than once in a multiset M we write $a \Subset M$. We let $Terms$ denote the set of first-order terms over VI. $Bind$ denotes the set of equations of the form $x = t$ where $x \in VI$ and $t \in Terms$ is distinct from x. Note that we do not impose the occurs-check condition $x \notin vars(t)$, since we target the analysis of Prolog and CLP systems possibly omitting this check. The following simplification of the standard definitions for the Sharing domain (Cortesi and Filé 1999; Hill et al. 1998; Jacobs and Langen 1992) assumes that the set of variables of interest is always given by VI.[2]

Definition 1
(The *set-sharing* domain SH.) The set SH is defined by

$$SH \stackrel{\mathrm{def}}{=} \wp(SG),$$

where the set of *sharing-groups* SG is given by

$$SG \stackrel{\mathrm{def}}{=} \wp(VI) \setminus \{\varnothing\}.$$

SH is ordered by subset inclusion. Thus the lub and glb of the domain are set union and intersection, respectively.

Definition 2
(Abstract operations over SH.) The *abstract existential quantification* on SH causes an element of SH to "forget everything" about a subset of the variables of

[2] Note that, during the analysis process, the set of variables of interest may expand (when solving the body of a clause) and contract (when abstract descriptions are projected onto the variables occurring in the head of a clause). However, at any given time the set of variables of interest is fixed. By consistently denoting this set by VI, we simplify the presentation, since we can omit the set of variables of interest to which an abstract description refers.

interest. It is encoded by the binary function aexists: $SH \times \wp(VI) \to SH$ such that, for each $sh \in SH$ and $V \in \wp(VI)$,

$$\operatorname{aexists}(sh, V) \stackrel{\text{def}}{=} \bigl\{\, S \setminus V \bigm| S \in sh, S \setminus V \neq \varnothing \,\bigr\} \cup \bigl\{\, \{x\} \bigm| x \in V \,\bigr\}.$$

For each $sh \in SH$ and each $V \in \wp(VI)$, the extraction of the *relevant component of sh with respect to V* is given by the function rel: $\wp(VI) \times SH \to SH$ defined as

$$\operatorname{rel}(V, sh) \stackrel{\text{def}}{=} \{\, S \in sh \mid S \cap V \neq \varnothing \,\}.$$

For each $sh \in SH$ and each $V \in \wp(VI)$, the function $\overline{\operatorname{rel}}$: $\wp(VI) \times SH \to SH$ gives the *irrelevant component of sh with respect to V*. It is defined as

$$\overline{\operatorname{rel}}(V, sh) \stackrel{\text{def}}{=} sh \setminus \operatorname{rel}(V, sh).$$

The function $(\cdot)^\star$: $SH \to SH$, also called *star-union*, is given, for each $sh \in SH$, by

$$sh^\star \stackrel{\text{def}}{=} \left\{\, S \in SG \;\middle|\; \exists n \geq 1 \;.\; \exists T_1, \ldots, T_n \in sh \;.\; S = \bigcup_{i=1}^{n} T_i \,\right\}.$$

For each $sh_1, sh_2 \in SH$, the function bin: $SH \times SH \to SH$, called *binary union*, is given by

$$\operatorname{bin}(sh_1, sh_2) \stackrel{\text{def}}{=} \{\, S_1 \cup S_2 \mid S_1 \in sh_1, S_2 \in sh_2 \,\}.$$

We also use the *self-bin-union* function sbin: $SH \to SH$, which is given, for each $sh \in SH$, by

$$\operatorname{sbin}(sh) \stackrel{\text{def}}{=} \operatorname{bin}(sh, sh).$$

The function amgu: $SH \times \mathit{Bind} \to SH$ captures the effect of a binding on an element of SH. Assume $(x = t) \in \mathit{Bind}$, $sh \in SH$, $V_x = \{x\}$, $V_t = \mathit{vars}(t)$, and $V_{xt} = V_x \cup V_t$. Then

$$\operatorname{amgu}(sh, x = t) \stackrel{\text{def}}{=} \overline{\operatorname{rel}}(V_{xt}, sh) \cup \operatorname{bin}\bigl(\operatorname{rel}(V_x, sh)^\star, \operatorname{rel}(V_t, sh)^\star\bigr). \tag{1}$$

We now briefly recall the standard integration of set-sharing with freeness and linearity information. These properties are each represented by a set of variables, namely those variables that are bound to terms that definitely enjoy the given property. These sets are partially ordered by *reverse* subset inclusion so that the lub and glb operators are given by set intersection and union, respectively.

Definition 3
(The domain *SFL*.) Let $F \stackrel{\text{def}}{=} \wp(VI)$ and $L \stackrel{\text{def}}{=} \wp(VI)$ be partially ordered by reverse subset inclusion. The domain *SFL* is defined by the Cartesian product

$$SFL \stackrel{\text{def}}{=} SH \times F \times L$$

ordered by the component-wise extension of the orderings defined on the three subdomains.

A complete definition would explicitly deal with the set of variables of interest VI. We could even define an equivalence relation on *SFL* identifying the bottom element

$\bot \stackrel{\mathrm{def}}{=} \langle \varnothing, \mathit{VI}, \mathit{VI} \rangle$ with all the elements corresponding to an impossible concrete computation state: for example, elements $\langle sh, f, l \rangle \in \mathit{SFL}$ such that $f \not\subseteq \mathit{vars}(sh)$ (because a free variable does share with itself) or $\mathit{VI} \setminus \mathit{vars}(sh) \not\subseteq l$ (because variables that cannot share are also linear). Note however that these and other similar spurious elements rarely occur in practice and cannot compromise the correctness of the results.

In a bottom-up abstract interpretation framework, such as the one we focus on, abstract unification is the only critical operation. Besides unification, the analysis depends on the 'merge-over-all-paths' operator, corresponding to the lub of the domain, and the abstract projection operator, which can be defined in terms of an abstract existential quantification operator.

Definition 4

(Abstract operations over *SFL*.) The *abstract existential quantification* on *SFL* is encoded by the binary function aexists: $\mathit{SFL} \times \wp(\mathit{VI}) \to \mathit{SFL}$ such that, for each $d = \langle sh, f, l \rangle \in \mathit{SFL}$ and $V \in \wp(\mathit{VI})$,

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

For each $d = \langle sh, f, l \rangle \in \mathit{SFL}$, we define the following predicates. The predicate $\mathit{ind}_d \colon \mathit{Terms} \times \mathit{Terms} \to \mathit{Bool}$ expresses definite independence of terms. Two terms $s, t \in \mathit{Terms}$ are *independent in* d if and only if $\mathit{ind}_d(s, t)$ holds, where

$$\mathit{ind}_d(s, t) \stackrel{\mathrm{def}}{=} \Big(\mathrm{rel}\big(\mathit{vars}(s), sh\big) \cap \mathrm{rel}\big(\mathit{vars}(t), sh\big) = \varnothing \Big).$$

A term $t \in \mathit{Terms}$ is *free in* d if and only if the predicate $\mathit{free}_d \colon \mathit{Terms} \to \mathit{Bool}$ holds for t, that is,

$$\mathit{free}_d(t) \stackrel{\mathrm{def}}{=} \big(\exists x \in \mathit{VI} \mathrel{.} x = t \land x \in f \big).$$

A term $t \in \mathit{Terms}$ is *linear in* d if and only if $\mathit{lin}_d(t)$, where $\mathit{lin}_d \colon \mathit{Terms} \to \mathit{Bool}$ is given by

$$\begin{aligned} \mathit{lin}_d(t) \stackrel{\mathrm{def}}{=} {} & \big(\mathit{vars}(t) \subseteq l \big) \\ & \land \big(\forall x, y \in \mathit{vars}(t) : x = y \lor \mathit{ind}_d(x, y) \big) \\ & \land \Big(\forall x \in \mathit{vars}(t) : x \Subset \mathit{mvars}(t) \Rightarrow x \notin \mathit{vars}(sh) \Big). \end{aligned}$$

The function amgu: $\mathit{SFL} \times \mathit{Bind} \to \mathit{SFL}$ captures the effects of a binding on an element of *SFL*. Let $(x = t) \in \mathit{Bind}$ and $d = \langle sh, f, l \rangle \in \mathit{SFL}$. Let also $V_x = \{x\}$, $V_t = \mathit{vars}(t)$, $V_{xt} = V_x \cup V_t$, $R_x = \mathrm{rel}(V_x, sh)$ and $R_t = \mathrm{rel}(V_t, sh)$. Then

$$\mathrm{amgu}\big(d, x = t\big) \stackrel{\mathrm{def}}{=} \langle sh', f', l' \rangle,$$

where

$$
\begin{aligned}
sh' &\stackrel{\mathrm{def}}{=} \overline{\mathrm{rel}}(V_{xt}, sh) \cup \mathrm{bin}\bigl(S_x, S_t\bigr); \\
S_x &\stackrel{\mathrm{def}}{=} \begin{cases} R_x, & \text{if } \mathit{free}_d(x) \lor \mathit{free}_d(t) \lor \bigl(\mathit{lin}_d(t) \land \mathit{ind}_d(x,t)\bigr); \\ R_x^\star, & \text{otherwise;} \end{cases} \\
S_t &\stackrel{\mathrm{def}}{=} \begin{cases} R_t, & \text{if } \mathit{free}_d(x) \lor \mathit{free}_d(t) \lor \bigl(\mathit{lin}_d(x) \land \mathit{ind}_d(x,t)\bigr); \\ R_t^\star, & \text{otherwise;} \end{cases} \\
f' &\stackrel{\mathrm{def}}{=} \begin{cases} f, & \text{if } \mathit{free}_d(x) \land \mathit{free}_d(t); \\ f \setminus \mathit{vars}(R_x), & \text{if } \mathit{free}_d(x); \\ f \setminus \mathit{vars}(R_t), & \text{if } \mathit{free}_d(t); \\ f \setminus \mathit{vars}(R_x \cup R_t), & \text{otherwise;} \end{cases} \\
l' &\stackrel{\mathrm{def}}{=} \bigl(\mathit{VI} \setminus \mathit{vars}(sh')\bigr) \cup f' \cup l''; \\
l'' &\stackrel{\mathrm{def}}{=} \begin{cases} l \setminus \bigl(\mathit{vars}(R_x) \cap \mathit{vars}(R_t)\bigr), & \text{if } \mathit{lin}_d(x) \land \mathit{lin}_d(t); \\ l \setminus \mathit{vars}(R_x), & \text{if } \mathit{lin}_d(x); \\ l \setminus \mathit{vars}(R_t), & \text{if } \mathit{lin}_d(t); \\ l \setminus \mathit{vars}(R_x \cup R_t), & \text{otherwise.} \end{cases}
\end{aligned}
$$

This specification of the abstract unification operator is equivalent (modulo the lack of the explicit structural information provided by *abstract equation systems*) to that given in (Bruynooghe et al. 1994a), provided $x \notin \mathit{vars}(t)$. Indeed, as done in all the previous papers on the subject, in (Bruynooghe et al. 1994a) it is assumed that the analyzed language does perform the occurs-check. As a consequence, whenever considering a *definitely cyclic binding*, that is a binding $x = t$ such that $x \in \mathit{vars}(t)$, the abstract operator can detect the definite failure of the concrete computation and thus return the bottom element of the domain. Such an improvement would not be safe in our case, since we also consider languages possibly omitting the occurs-check. However, when dealing with definitely cyclic bindings, the specification given by the previous definition can still be refined as follows.

Definition 5

(Improvement for definitely cyclic bindings.) Consider the specification of the abstract operations over *SFL* given in Definition 4. Then, whenever $x \in \mathit{vars}(t)$, the computation of the new sharing component sh' can be replaced by the following.[3]

$$sh' \stackrel{\mathrm{def}}{=} \overline{\mathrm{rel}}(V_{xt}, sh) \cup \mathrm{bin}\bigl(S_x, \mathit{CS}_t\bigr),$$

[3] Note that, in this special case, it also holds that $\mathit{free}_d(t) = \mathrm{false}$ and $\mathit{ind}_d(x,t) = (R_x = \varnothing)$.

where

$$CS_t \stackrel{\text{def}}{=} \begin{cases} CR_t, & \text{if } \mathit{free}_d(x); \\ CR_t^\star, & \text{otherwise;} \end{cases}$$

$$CR_t \stackrel{\text{def}}{=} \mathrm{rel}\bigl(\mathit{vars}(t) \setminus \{x\}, \mathit{sh}\bigr).$$

This enhancement, already implemented in the CHINA analyzer, is the rewording of a similar one proposed in (Bagnara 1997a) for the domain *Pos* in the context of groundness analysis. Its net effect is to recover some groundness and sharing dependencies that are unnecessarily lost when using the standard operators.

The domain *SH* captures *set-sharing*. However, the property we wish to detect is *pair-sharing* and, for this, it has been shown in (Bagnara et al. 2002) that *SH* includes unwanted redundancy. The same paper introduces an upper-closure operator ρ on *SH* and the domain $\mathit{PSD} \stackrel{\text{def}}{=} \rho(\mathit{SH})$, which is the weakest abstraction of *SH* that is *as precise as SH* as far as tracking groundness and pair-sharing is concerned.[4] A notable advantage of *PSD* is that we can replace the star-union operation in the definition of the amgu by self-bin-union without loss of precision. In particular, in (Bagnara et al. 2002) it is shown that

$$\mathrm{amgu}(\mathit{sh}, x = t) =_\rho \overline{\mathrm{rel}}(V_{xt}, \mathit{sh}) \cup \mathrm{bin}\Bigl(\mathrm{sbin}\bigl(\mathrm{rel}(V_x, \mathit{sh})\bigr), \mathrm{sbin}\bigl(\mathrm{rel}(V_t, \mathit{sh})\bigr)\Bigr), \qquad (2)$$

where the notation $\mathit{sh}_1 =_\rho \mathit{sh}_2$ means $\rho(\mathit{sh}_1) = \rho(\mathit{sh}_2)$.

It is important to observe that the complexity of the amgu operator on *SH* (1) is exponential in the number of sharing-groups of *sh*. In contrast, the operator on *PSD* (2) is $O\bigl(|\mathit{sh}|^4\bigr)$. Moreover, checking whether a fixpoint has been reached by testing $\mathit{sh}_1 =_\rho \mathit{sh}_2$ has complexity $O\bigl(|\mathit{sh}_1|^3 + |\mathit{sh}_2|^3\bigr)$. Practically speaking, very often this makes the difference between thrashing and termination of the analysis in reasonable time.

The above observations on *SH* and *PSD* can be generalized to apply to the domain combinations *SFL* and $\mathit{SFL}_2 \stackrel{\text{def}}{=} \mathit{PSD} \times F \times L$. In particular, SFL_2 achieves the same precision as *SFL* for groundness, pair-sharing, freeness and linearity and the complexity of the corresponding abstract unification operator is polynomial. For this reason, all the experimental work in this paper, with the exception of part of the one described in Section 7, has been conducted using the SFL_2 domain.

3 Experimental Evaluation

Since the main purpose of this paper is to provide an experimental measure of the precision gains that might be achieved by enhancing a standard sharing analysis with several new techniques we found in the literature, it is clear that the implementation of the various domain combinations was a major part of the work. However, so as to adapt these assorted proposals into a uniform framework and provide a fair

[4] The name *PSD*, which stands for *Pair-Sharing Dependencies*, was introduced in (Zaffanella et al. 1999). All previous papers, including (Bagnara et al. 2002), denoted this domain by SH^ρ.

comparison of their results, a large amount of underlying conceptual work was also required. For instance, almost all of the proposed enhancements were designed for systems that perform the occurs-check and some of them were developed for rather different abstract domains: besides changing the representation of the domain elements, such a situation usually requires a reconsideration of the specification of the abstract operators.

All the experiments have been conducted using the China analyzer (Bagnara 1997a) on a GNU/Linux PC system equipped with an AMD Athlon clocked at 700 MHz and 256 MB of RAM. China is a data-flow analyzer for CLP($\mathcal{H}_\mathcal{N}$) languages (i.e., ISO Prolog, CLP($\mathcal{R}$), `clp(FD)` and so forth), $\mathcal{H}_\mathcal{N}$ being an extended Herbrand system where the values of a numeric domain $\mathcal{N}$ can occur as leaves of the terms. China, which is written in C++, performs bottom-up analysis 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 using a sophisticated chaotic iteration strategy proposed in (Bourdoncle 1993a; Bourdoncle 1993b).[5]

A major point of the experimental evaluation is given by the test-suite, which is probably the largest one ever reported in the literature on data-flow analysis of (constraint) logic programs. The suite comprises all the programs we have access to (i.e., everything we could find by systematically dredging the Internet): more than 330 programs, 24 MB of code, 800 K lines. Besides classical benchmarks, several real programs of respectable size are included, the largest one containing 10063 clauses in 45658 lines of code. The suite also comprises a few synthetic benchmarks, which are artificial programs explicitly constructed to stress the capabilities of the analyzer and of its abstract domains with respect to precision and/or efficiency.

Because of the exponential complexity of the base domain *SFL*, a data-flow analysis that includes this domain will only be practical if it incorporates widening operators (Zaffanella et al. 1999).[6] However, since almost none of the investigated combinations come with specialized widening operators, for a fair assessment of the precision improvements we decided to disable all the widenings available in our *SFL* implementation. As a consequence, there are a few benchmarks for which the analysis does not terminate in reasonable time or absorbs memory beyond acceptable limits, so that a precision comparison is not possible. Note however that the motivations behind this choice go beyond the simple observation that widening operators affect the precision of the analysis: the problem is also that, if we use the widenings defined and tuned for our implementation of the domain *SFL*, the results would be biased. In fact, the definition of a good widening for an analysis domain normally depends on both the representation and the implementation of the domain. In other words, different implementations even of the same domain

[5] China uses the recursive fixpoint iteration strategy on the weak topological ordering defined by partitioning of the call graph into strongly-connected subcomponents (Bourdoncle 1993b).

[6] Note that we use the term 'widening operator' in its broadest sense: any mechanism whereby, in the course of the analysis, an abstract description is substituted by one that is less precise.

will require different tunings of the widening operators (or even, possibly, brand new widenings). This means that adopting the same widening operators for all the domain combinations would weaken, if not invalidate, any conclusions regarding the relative benefits of the investigated enhancements. On the other hand, the definition of a new specialized widening operator for each one of the considered domain combinations, besides being a formidable task, would also be wasted effort as the number of benchmark programs for which termination cannot be obtained within reasonable time is really small.

For space reasons, the experimental results are only summarized here. The interested reader can find more information (including a description of the constantly growing benchmark suite and detailed results for each benchmark) at the URI `http://www.cs.unipr.it/China/`. Indeed, given the high number of benchmark programs and the many domain combinations considered,[7] even finding a concise, meaningful and practical way to summarize the results has been a non-trivial task.

For each benchmark, precision is measured by counting the number of independent pairs (the corresponding columns are labeled 'I' in the tables) as well as the numbers of definitely ground (labeled 'G'), free ('F') and linear ('L') variables detected by each abstract domain. The results obtained for different analyses are compared by computing the relative precision improvements or degradations on each of these quantities and expressing them using percentages. The "overall" ('O') precision improvement for the benchmark is also computed as the maximum improvement on all the measured quantities.[8] The benchmark suite is then partitioned into several precision equivalence classes: the cardinalities of these classes are expressed again using percentages. For example, when looking at the precision results reported in Table 1 for goal-dependent analysis, the value 2.3 that can be found at the intersection of the row labeled '$0 < p \leq 2$' with the column labeled 'G' is to be read as follows: "for 2.3 percent of the benchmarks the increase in the number of ground variables is less than or equal to 2 percent." The precision class labeled 'unknown' identifies those benchmarks for which a precision comparison was not possible, because one or both of the analyses was timed-out (for all comparisons, the time-out threshold is 600 seconds). In summary, a precision table gives an approximation of the distribution of the programs in the benchmark suite with respect to the obtained precision gains.

For a rough estimate of the efficiency of the different analyses, for each comparison we provide two tables that summarize the times taken by the fixpoint computations. It should be stressed that these by no means provide a faithful account of the

[7] We compute the results of 40 different variations of the static analysis, which are then used to perform 36 comparisons. The results are computed over 332 programs for goal-independent analyses and over 221 programs for goal-dependent analyses. This difference in the number of benchmarks considered comes from the fact that many programs either are not provided with a set of entry goals or use constructs such as `call(G)` where `G` is a term whose principal functor is not known. In these cases the analyzer recognizes that goal-dependent analysis is pointless, since no call-patterns can be excluded.

[8] When computing this "overall" result for a benchmark, the presence of even a single precision loss for one of the measures overrides any precision improvement computed on the other components.

intrinsic computational cost of the tested domain combinations. Besides the lack of widenings, which have a big impact on performance as can be observed by the results reported in (Zaffanella et al. 1999), the reader should not forget that, for ease of implementation, having targeted at precision we traded efficiency whenever possible. Therefore, these tables provide, so to speak, upper-bounds: refined implementations can be expected to perform at least as well as those reported in the tables.

As done for the precision results, the timings are summarized by partitioning the suite into equivalence classes and reporting the cardinality of each class using percentages. In the first table we consider the distribution of the *absolute time differences*, that is we measure the slow-down and speed-up due to the incorporation of the considered enhancement. Note that the class called 'same time' actually comprises the benchmarks having a time difference below a given threshold, which is fixed at 0.1 seconds. In the second table we show the distribution of the *total fixpoint computation times*, both for the base analysis (in the columns labeled '%1') and for the enhanced one (in the columns labeled '%2'); the columns labeled 'Δ' show how much each total time class grows or shrinks due to the inclusion of the considered combination.

4 A Simple Combination with Pos

It is well-known that the domain Sharing (and thus also *SFL*) keeps track of ground dependencies. More precisely, Sharing contains *Def*, the domain of definite Boolean functions (Armstrong et al. 1998), as a proper subdomain (Cortesi et al. 1992; Zaffanella et al. 1999). However, we consider here the combination of *SFL* with *Pos*, the domain of positive Boolean functions (Armstrong et al. 1998). There are several good reasons to couple *SFL* with *Pos*:

1. *Pos* is strictly more expressive than *Def* in that it can represent (positive) disjunctive groundness dependencies that arise in the analysis of Prolog programs (Armstrong et al. 1998). The ability to deal with disjunctive dependencies is also needed for the precise approximation of the constraints of some CLP languages: for example, when using the finite domain solver of SICStus Prolog, the user can write disjunctive constraints such as '`X #= 4 #\/ Y #= 6`'.
2. The increased precision on groundness propagates to the *SFL* component. It can be exploited to remove redundant sharing groups and to identify more linear variables, therefore having a positive impact on the computation of the amgu operator of the *SFL* domain. Moreover, when dealing with sequences of bindings, the added groundness information allows them to be usefully reordered. In fact, while it has been proved that Sharing alone is commutative, meaning that the result of the analysis does not depend on the ordering in which the bindings are executed (Hill et al. 1998), the domain *SFL* does not enjoy this property. In particular, even for the simpler combination of Sharing with linearity it is known since (Langen 1990, pp. 66-67) that better results

are obtained if the *grounding bindings* are considered before the others.[9] As an example, consider the sequences of unifications $\bigl(f(X, X, Y) = A,\ X = a\bigr)$ and $\bigl(X = a,\ f(X, X, Y) = A\bigr)$ (Langen 1990, p. 66). The combination with *Pos* is clearly advantageous in this respect.

3. Besides being useful for improving precision on other properties, disjunctive dependencies also have a few direct applications, such as occurs-check reduction. As observed in (Crnogorac et al. 1996), if the groundness formula $x \vee y$ holds, the unification $x = y$ is occurs-check free, even when neither x nor y are definitely linear.
4. Detecting the set of definitely ground variables through *Pos* and exploiting it to simplify the operations on *SFL* can improve the efficiency of the analysis. In particular this is true if the set of ground variables is readily available, as is the case, for instance, with the GER implementation of *Pos* (Bagnara and Schachte 1999).
5. The combination with *Pos* is essential for the application of a powerful widening technique on *SFL* (Zaffanella et al. 1999). This is very important, since analysis based on *SFL* is not practical without widenings.
6. In the context of the analysis of CLP programs, the notions of "ground variable" and the notion of "variable that cannot share a common variable with other variables" are distinct. A numeric variable in, say, CLP($\mathcal{R}$), cannot share with other numerical variables (not in the sense of interest in this paper) but is not ground unless it has been constrained to a unique value. Thus the analysis of CLP programs with *SFL* alone either will lose precision on pair-sharing (if arithmetic constraints are abstracted into "sharings" among numeric variables in order to approximate the groundness of the latter) or will be imprecise on the groundness of numeric variables (because only Herbrand constraints take part in the construction of sharing-sets). In the first alternative, as we have already noted, the precision with which groundness of numeric variables can be tracked will also be limited. Since groundness of numeric variables is important for a number of applications (e.g., compiling equality constraints down to assignments or tests in some circumstances), we advocate the use of *Pos* and *SFL* at the same time.

Thus, as a first technique to enhance the precision of sharing analysis, we consider the simple propagation of the set of definitely ground variables from the *Pos* component to the *SFL* component.[10] We denote this domain by $Pos \times SFL$.

As noted above, the GER implementation of (Bagnara and Schachte 1999), besides being the fastest implementation of *Pos* known to date, is the natural candidate for this combination, since it provides constant-time access to the set G of the definitely ground variables. Note that the widenings on the *Pos* component

[9] A binding $x = t$ is *grounding* with respect to an abstract description if, in all the concrete computation states approximated by the abstract description, either the variable x is ground or all the variables in t are ground. For example, when considering an abstract description $sh \in SH$, the binding $x = t$ is grounding if $\mathrm{rel}(\{x\}, sh) = \varnothing$ or $\mathrm{rel}(vars(t), sh) = \varnothing$.

[10] A more precise combination will be considered in Section 7.

Prec. class	Goal Independent					Goal Dependent				
	O	I	G	F	L	O	I	G	F	L
$5 < p \leq 10$	—	—	—	—	—	0.5	—	0.5	—	—
$2 < p \leq 5$	0.3	—	0.3	—	—	—	—	—	—	—
$0 < p \leq 2$	0.6	0.6	0.6	—	0.6	3.2	3.6	2.3	—	2.7
same precision	95.8	96.1	95.8	96.7	96.1	92.8	92.8	93.7	96.4	93.7
unknown	3.3	3.3	3.3	3.3	3.3	3.6	3.6	3.6	3.6	3.6

Time difference class	% benchmarks	
	Goal Ind.	Goal Dep.
degradation > 1	2.7	6.8
$0.5 <$ degradation ≤ 1	1.5	0.5
$0.2 <$ degradation ≤ 0.5	3.0	0.9
$0.1 <$ degradation ≤ 0.2	5.7	5.0
both timed out	3.3	3.6
same time	81.6	81.9
$0.1 <$ improvement ≤ 0.2	—	0.5
$0.2 <$ improvement ≤ 0.5	0.9	0.5
$0.5 <$ improvement ≤ 1	0.3	—
improvement > 1	0.9	0.5

Total time class	Goal Ind.			Goal Dep.		
	%1	%2	Δ	%1	%2	Δ
timed out	3.3	3.3	—	3.6	3.6	—
$t > 10$	8.4	9.0	0.6	7.2	7.2	—
$5 < t \leq 10$	0.6	0.3	-0.3	1.4	1.4	—
$1 < t \leq 5$	6.6	7.5	0.9	3.2	3.6	0.5
$0.5 < t \leq 1$	3.3	2.7	-0.6	5.4	5.4	—
$0.2 < t \leq 0.5$	7.2	8.4	1.2	10.4	13.1	2.7
$t \leq 0.2$	70.5	68.7	-1.8	68.8	65.6	-3.2

Table 1. SFL_2 *versus* $\mathit{Pos} \times \mathit{SFL}_2$.

have been retained. The reason for this choice is that they fire for only a few benchmarks and, when coming into play, they rarely affect the precision of the groundness analysis: by switching them off we would only obtain a few more time-outs.

In the *SFL* component, the set G of definitely ground variables is used

- to reorder the sequence of bindings in the abstract unification so as to handle the grounding ones first;
- to eliminate the sharing groups containing at least one ground variable; and
- to recover from previous linearity losses.

The experimental results for $\mathit{Pos} \times \mathit{SFL}$ are compared with those obtained for the domain *SFL* considered in isolation and reported in Table 1. It can be observed that a precision improvement is observed in all of the measured quantities but freeness, affecting up to 3.6% of the programs.

Note that there is a small discrepancy between these results and those of (Bagnara et al. 2000) where more improvements were reported. The reason is that the current *SFL* implementation uses an enhanced abstract unification operator, fully exploiting the anticipation of the grounding bindings even on the base domain *SFL* itself. In contrast, in the earlier *SFL* implementation used for the results in (Bagnara et al. 2000), only the *syntactically* grounding bindings were anticipated.[11]

As for the timings, even if the figures in the tables seem to contradict what we claimed in point 4 above, a closer inspection of the detailed results reveals that this is only due to a very unfortunate interaction between the increased precision given by *Pos* and the absence of widening operators on *SFL*. This state of affairs forces the analyzer to compute a few, but very expensive, further iterations in the fixpoint computation.

Because of the reasons detailed above, we believe *Pos* should be part of the global domain employed by any "production analyzer" for CLP languages. That is why, for the remaining comparisons, unless otherwise stated, this simple combination with the *Pos* domain is always included.

5 Tracking Explicit Structural Information

A way of increasing the precision of almost any analysis domain is by enhancing it with structural information. For mode analysis, this idea dates back to (Janssens and Bruynooghe 1992). A more general technique was proposed in (Cortesi et al. 1994), where the generic structural domain $\mathtt{Pat}(\Re)$ was introduced. A similar proposal, tailored to sharing analysis, is due to (Bruynooghe et al. 1994a), where *abstract equation systems* are considered. In the experimental evaluation the $\mathrm{Pattern}(\cdot)$ construction (Bagnara 1997a; Bagnara 1997b; Bagnara et al. 2000) is used. This is similar to $\mathtt{Pat}(\Re)$ and correctly supports the analysis of languages omitting the occurs-check in the unification procedure as well as those that do not.

The construction $\mathrm{Pattern}(\cdot)$ upgrades a domain $\mathcal{D}$ (which must support a certain set of basic operations) with structural information. The resulting domain, where structural information is retained to some extent, is usually much more precise than $\mathcal{D}$ alone. There are many occasions where these precision gains give rise to consistent speed-ups. The reason for this is twofold. First, structural information has the potential of pruning some computation paths on the grounds that they cannot be followed by the program being analyzed. Second, maintaining a tuple of terms with many variables, each with its own description, can be cheaper than computing a description for the whole tuple (Bagnara et al. 2000). Of course, there is also a price to be paid: in the analysis based on $\mathrm{Pattern}(\mathcal{D})$, the elements of

[11] A binding $x = t$ is syntactically grounding if $\mathit{vars}(t) = \varnothing$. This "syntactic" definition differs from the "semantic" one provided before in that it does not depend on the information provided by an abstract description.

$\mathcal{D}$ that are to be manipulated are often bigger (i.e., there are more variables of interest) than those that arise in analyses that are simply based on $\mathcal{D}$.

When comparing the precision results, the difference in the number of variables tracked by the two analyses poses a non-trivial problem. How can we provide a *fair* measure of the precision gain? There is no easy answer to such a question. The approach chosen is simple though unsatisfactory: at the end of the analysis, first throw away all the structural information in the results and then calculate the cardinality of the usual sets. In other words, we only measure how the explicit structural information in Pattern($\mathcal{D}$) improves the precision on $\mathcal{D}$ itself, which is only a tiny part of the real gain in accuracy. As shown by the following example, this solution greatly underestimates the precision improvement coming from the integration of structural information.

Consider a simple but not trivial Prolog program: `mastermind`.[12] Consider also the only direct query for which it has been written, '`?- play.`', and focus the attention on the procedure `extend_code/1`. A standard goal-dependent analysis of the program with the $Pos \times SFL$ domain cannot say anything on the successes of `extend_code/1`. If the analysis is performed with Pattern($Pos \times SFL$) the situation changes radically. Here is what such a domain allows China to derive:[13]

```
extend_code([([A|B],C,D)|E]) :-
  list(B), list(E),
  (functor(C,_,1);integer(C)),
  (functor(D,_,1);integer(D)),
  ground([C,D]), may_share([[A,B,E]]).
```

This means: "during any execution of the program, whenever `extend_code/1` succeeds it will have its argument bound to a term of the form `[([A|B],C,D)|E]`, where `B` and `E` are bound to list cells (i.e., to terms whose principal functor is either `'.'/2` or `[]/0`); `C` and `D` are ground and bound to a functor of arity 1 or to an integer; and pair-sharing may only occur among `A`, `B`, and `E`". Once structural information has been discarded, the analysis with Pattern($Pos \times SFL$) only specifies that `extend_code/1` may succeed. Thus, according to our approach to the precision comparison, explicit structural information gives no improvements in the analysis of `extend_code/1` (which is far from being a fair conclusion).

Of course, structural information is very valuable in itself. For example, when exploited for optimized compilation it allows for enhanced clause indexing and simplified unification. Several other semantics-based program manipulation techniques (such as debugging, program specialization, and verification) benefit from this kind of information. However, the value of this extra precision could only be measured from the point of view of the target application of the analysis.

[12] This program which implements the game "Mastermind" was rewritten by H. Koenig and T. Hoppe after code by M. H. van Emden and available at `http://www.cs.unipr.it/China/Benchmarks/Prolog/mastermind.pl`.

[13] Some extra groundness information obtained by the analysis has been omitted for simplicity: this says that, if `A` and `B` turn out to be ground, then `E` will also be ground.

Prec. class	Goal Independent					Goal Dependent				
	O	I	G	F	L	O	I	G	F	L
$p > 20$	7.5	2.7	3.9	2.1	3.3	6.3	1.4	3.6	1.8	3.6
$10 < p \leq 20$	3.9	2.1	2.7	—	2.4	2.7	2.3	1.4	—	2.7
$5 < p \leq 10$	4.5	1.8	2.7	2.4	2.4	1.8	0.9	2.3	0.9	1.4
$2 < p \leq 5$	7.5	6.0	3.9	2.7	5.1	2.7	3.2	1.4	1.8	2.3
$0 < p \leq 2$	7.8	9.0	6.6	6.9	12.0	2.3	4.5	1.8	1.8	5.0
same precision	61.7	71.7	73.5	79.2	67.8	74.2	78.3	80.1	84.2	75.1
unknown	6.6	6.6	6.6	6.6	6.6	9.5	9.5	9.5	9.5	9.5
$p < 0$	0.3	—	—	—	0.3	0.5	—	—	—	0.5

Time diff. class	% benchmarks	
	Goal Ind.	Goal Dep.
degradation > 1	11.7	17.6
$0.5 <$ degradation ≤ 1	1.2	0.9
$0.2 <$ degradation ≤ 0.5	3.6	4.1
$0.1 <$ degradation ≤ 0.2	1.5	4.1
both timed out	3.3	3.6
same time	70.8	66.5
$0.1 <$ improvement ≤ 0.2	0.9	0.5
$0.2 <$ improvement ≤ 0.5	1.5	—
$0.5 <$ improvement ≤ 1	0.6	0.5
improvement > 1	4.8	2.3

Total time class	Goal Ind.			Goal Dep.		
	%1	%2	Δ	%1	%2	Δ
timed out	3.3	6.6	3.3	3.6	9.5	5.9
$t > 10$	9.0	8.4	-0.6	7.2	8.6	1.4
$5 < t \leq 10$	0.3	1.5	1.2	1.4	1.8	0.5
$1 < t \leq 5$	7.5	6.6	-0.9	3.6	5.0	1.4
$0.5 < t \leq 1$	2.7	3.3	0.6	5.4	3.2	-2.3
$0.2 < t \leq 0.5$	8.4	10.2	1.8	13.1	13.6	0.5
$t \leq 0.2$	68.7	63.3	-5.4	65.6	58.4	-7.2

Table 2. $\mathit{Pos} \times \mathit{SFL}_2$ *versus* $\mathrm{Pattern}(\mathit{Pos} \times \mathit{SFL}_2)$.

Thus the precision of the domain $\mathit{Pos} \times \mathit{SFL}$ has been compared with that obtained using the domain $\mathrm{Pattern}(\mathit{Pos} \times \mathit{SFL})$ and the results reported in Table 2. It can be seen that, for goal-independent analysis, on one third of the benchmarks compared there is a precision improvement in at least one of the measured quantities; the same happens for one sixth of the benchmarks in the case of goal-dependent analysis. Moreover, the increase in precision can be considerable, as testified by the percentages of benchmarks falling in the higher precision classes.

The reader may be surprised, as the authors were, to see that in some cases the precision actually decreased.[14] Indeed, to the best of our knowledge, this possibility has escaped all previous research work investigating this kind of abstract domain enhancement, including (Cortesi et al. 1994; Bruynooghe et al. 1994a; Bagnara 1997a). The reason for these precision losses lies in a subtle interaction between the explicit structural information and the underlying abstract unification operator.

When using the base domain $Pos \times SFL$, the abstract evaluation of a single syntactic binding, such as $x = f(y, z)$, directly corresponds to a single application of the amgu operator. In contrast, when computing on $\mathrm{Pattern}(Pos \times SFL)$, it may well happen that the computed abstract description already contains the information that variable x is bound to a term, such as $f\bigl(g(w), w\bigr)$. As a consequence, after peeling the principal functor $f/2$, the abstract computation should proceed by evaluating, on the base domain $Pos \times SFL$, the set of bindings $\bigl\{y = g(w), z = w\bigr\}$. Here the problem is that, as already noted, the amgu operator on the base domain $Pos \times SFL$ is not commutative. While this improvement in the data used by the abstract computation very often allows for a corresponding increase in the precision of the result, in rare situations it may happen that a sub-optimal ordering of the bindings is chosen, incurring a precision loss.

It should be noted that such a negative interaction with the explicit structural information is only possible when the underlying domain implements non-commutative abstract operators. In particular, this phenomenon could not be observed when computing on $\mathrm{Pattern}(SH)$ or $\mathrm{Pattern}(Pos)$.

One issue that should be resolved is whether the improvements provided by explicit structural information subsume those previously obtained for the simple combination with *Pos*. Intuitively, it would seem that this cannot happen, since these two enhancements are based on different kinds of information: while the $\mathrm{Pattern}(\cdot)$ construction encodes some *definite* structural information, the precision gain due to using *Pos* rather than just *Def* only stems from *disjunctive* groundness dependencies. However, the impact of these techniques on the overall analysis is really intricate and some overlapping cannot be excluded *a priori*: for instance, both techniques affect the ordering of bindings in the computation of abstract unification on *SFL*. In order to provide some experimental evidence for this qualitative reasoning, the precision results are computed for the simpler domain $\mathrm{Pattern}(SFL)$ and then compared with those obtained for the domain $\mathrm{Pattern}(Pos \times SFL)$. Since the main differences between Tables 1 and 3 can be explained by discrepancies in the numbers of programs that timed-out, these results confirm our expectations that these two enhancements are effectively orthogonal.

Similar experimental evaluations, but based on the abstract equation systems of (Bruynooghe et al. 1994a), were reported by A. Mulkers et al. in (Mulkers et al. 1994; Mulkers et al. 1995). Here a depth-k abstraction (replacing all subterms occurring at a depth greater than or equal to k with fresh abstract variables) is conducted on a small benchmark suite (19 programs) for values of k between 0 and 3. The domain they employed was not suitable for the analysis of real programs and, in

[14] This happens for the program `attractions2` in the case of goal-independent analysis and for the program `semi` in the case of goal-dependent analysis.

Prec. class	Goal Independent					Goal Dependent				
	O	I	G	F	L	O	I	G	F	L
$5 < p \leq 10$	—	—	—	—	—	0.5	—	0.5	—	—
$2 < p \leq 5$	0.3	—	0.3	—	—	—	0.5	—	—	—
$0 < p \leq 2$	—	—	—	—	—	3.2	3.2	2.7	—	2.7
same precision	93.1	93.4	93.1	93.4	93.4	86.4	86.4	86.9	90.0	87.3
unknown	6.6	6.6	6.6	6.6	6.6	10.0	10.0	10.0	10.0	10.0

Time diff. class	% benchmarks	
	Goal Ind.	Goal Dep.
degradation > 1	5.7	7.7
$0.5 <$ degradation ≤ 1	2.4	0.5
$0.2 <$ degradation ≤ 0.5	3.6	5.4
$0.1 <$ degradation ≤ 0.2	5.4	2.7
both timed out	6.6	9.5
same time	75.6	73.8
$0.1 <$ improvement ≤ 0.2	—	—
$0.2 <$ improvement ≤ 0.5	0.6	—
$0.5 <$ improvement ≤ 1	—	—
improvement > 1	—	0.5

Total time class	Goal Ind.			Goal Dep.		
	%1	%2	Δ	%1	%2	Δ
timed out	6.6	6.6	—	10.0	9.5	-0.5
$t > 10$	8.1	8.4	0.3	7.7	8.6	0.9
$5 < t \leq 10$	1.5	1.5	—	2.3	1.8	-0.5
$1 < t \leq 5$	5.1	6.6	1.5	4.5	5.0	0.5
$0.5 < t \leq 1$	3.9	3.3	-0.6	3.2	3.2	—
$0.2 < t \leq 0.5$	7.2	10.2	3.0	10.9	13.6	2.7
$t \leq 0.2$	67.5	63.3	-4.2	61.5	58.4	-3.2

Table 3. Pattern(SFL_2) *versus* Pattern($\mathit{Pos} \times \mathit{SFL}_2$).

fact, even the analysis of a modest-sized program like `ann` could only be carried out with depth-0 abstraction (i.e., without any structural information). Such a problem in finding practical analyzers that incorporated structural information with sharing analysis was not unique to this work: there was at least one other previous attempt to evaluate the impact of structural information on sharing analysis that failed because of combinatorial explosion [A. Cortesi, personal communication, 1996].

What makes the more realistic experimentation now possible is the adoption of the non-redundant domain *PSD*, where the exponential star-union operation is replaced by the quadratic self-bin-union. Note that, even if biased by the absence of widenings, the timings reported in Table 2 show that the Pattern($\cdot$) construction is computationally feasible. Indeed, as demonstrated by the results reported in

(Bagnara et al. 2000), an analyzer that incorporates a carefully designed structural information component, besides being more precise, can also be very efficient.

The results obtained in this section demonstrate that there is a relevant amount of sharing information that is not detected when using the classical set-sharing domains. Therefore, in order to provide an experimental evaluation that is as systematic as possible, in all of the remaining experiments the comparison is performed both with and without explicit structural information.

6 Reordering the Non-Grounding Bindings

As already explained in Section 4, the results of abstract unification on *SFL* may depend on the order in which the bindings are considered and will be improved if the grounding bindings are considered first. This heuristic, which has been used for all the experiments in this paper, is well-known: in the literature all the examples that illustrate the non-commutativity of the abstract mgu on *SFL* use a grounding binding. However, as observed in Section 5, the problem is more general than that.

To illustrate this, suppose that $\mathit{VI} = \{u, v, w, x, y, z\}$ is the set of relevant variables, and consider the *SFL* element[15]

$$d \stackrel{\text{def}}{=} \bigl\langle \{vy, wy, xy, yz\}, \varnothing, \{u, x, z\} \bigr\rangle,$$

where no variable is free and u, x, and z are linear with the bindings $v = w$ and $x = y$. Then, applying amgu to these bindings in the given ordering, we have:

$$\begin{aligned}
d_1 &= \operatorname{amgu}(d, v = w) \\
&= \bigl\langle \{vwy, xy, yz\}, \varnothing, \{u, x, z\} \bigr\rangle, \\
d_{1,2} &= \operatorname{amgu}(d_1, x = y) \\
&= \bigl\langle \{vwxy, vwxyz, xy, xyz\}, \varnothing, \{u, z\} \bigr\rangle.
\end{aligned}$$

Using the reverse ordering, we have:

$$\begin{aligned}
d_2 &= \operatorname{amgu}(d, x = y) \\
&= \bigl\langle \{vwxy, vwxyz, vxy, vxyz, wxy, wxyz, xy, xyz\}, \varnothing, \{u, z\} \bigr\rangle, \\
d_{2,1} &= \operatorname{amgu}(d_2, v = w) \\
&= \bigl\langle \{vwxy, vwxyz, xy, xyz\}, \varnothing, \{u\} \bigr\rangle.
\end{aligned}$$

Thus $d_{2,1}$ loses the linearity of z (which, in turn, could cause bigger precision losses later in the analysis).

In principle, optimality can be obtained by adopting the *brute-force* approach: trying all the possible orderings of the non-grounding bindings. However, this is clearly not feasible. While lacking a better alternative, it is reasonable to look for heuristics that can be applied in the context of a *local search* paradigm: at each

[15] Elements of *SH* are written in a simplified notation, omitting the inner braces. For instance, the set $\bigl\{\{x\}, \{x, y\}, \{x, z\}, \{x, y, z\}\bigr\}$ is written as $\{x, xy, xz, xyz\}$.

step, the next binding for the amgu procedure is chosen by evaluating the effect of its abstract execution, considered in isolation, on the precision of the analysis.

Suppose the number of independent pairs is taken as a measure of precision. Then, at each step, for each of the bindings under consideration, the new component sh', as given by Definition 4, must be computed. However, because the computation of sh' is the most costly operation to be performed in the computation of the amgu operator, a direct application of this heuristic does not appear to be feasible. As an alternative, consider a heuristic based on the number of star-unions that have to be computed. Star-unions are likely to cause large losses in the number of independent pairs that are found. As only non-grounding bindings are considered, any binding requiring the computation of a star-union will need the star-union even if it is delayed, although a binding that does not require the star-union may require it if its computation is postponed: its variables may lose their freeness, linearity or independence as a result of evaluating the other bindings. It follows that one potential heuristic is: "delay the bindings requiring star-unions as much as possible". In the next example, by adopting this heuristic, the linearity of variable y is preserved.

Consider the application of the bindings $x = z$ and $v = w$ to the following abstract description:

$$d \stackrel{\mathrm{def}}{=} \bigl\langle \{vw, wx, wy, z\}, \varnothing, \{u, v, x, y\} \bigr\rangle.$$

Since x is linear and independent from z, computing $\mathrm{amgu}(d, x = z)$ requires one star-union, while two star-unions are needed when computing $\mathrm{amgu}(d, v = w)$ because v and w may share. Thus, with the proposed heuristic, $x = z$ is applied before $v = w$, giving:

$$\begin{aligned}
d_1 &= \mathrm{amgu}(d, x = z) \\
&= \bigl\langle \{vw, wxz, wy\}, \varnothing, \{u, v, y\} \bigr\rangle, \\
d_{1,2} &= \mathrm{amgu}(d_1, v = w) \\
&= \bigl\langle \{vw, vwxyz, vwxz, vwy\}, \varnothing, \{u, y\} \bigr\rangle.
\end{aligned}$$

In contrast, if $v = w$ is applied first, we have:

$$\begin{aligned}
d_2 &= \mathrm{amgu}(d, v = w) \\
&= \bigl\langle \{vw, vwx, vwxy, vwy, z\}, \varnothing, \{u, x, y\} \bigr\rangle, \\
d_{2,1} &= \mathrm{amgu}(d_2, x = z) \\
&= \bigl\langle \{vw, vwxyz, vwxz, vwy\}, \varnothing, \{u\} \bigr\rangle.
\end{aligned}$$

Note that the same number of independent pairs is computed in both cases.

It should be noted that this heuristic, considered in isolation, is not a general solution and can actually lead to precision losses. The problem is that, if a binding that needs a star-union is delayed, then, when the star-union is computed, it may be done on a larger sharing-set, forcing more (independent) pairs of variables into the same sharing group.

Consider the application of the bindings $u = x$ and $v = w$ to the abstract description

$$d \stackrel{\text{def}}{=} \bigl\langle \{u, uw, v, w, xy, xz\}, \{u, x\}, \{u, x\} \bigr\rangle.$$

Since x and u are both free variables, no star-union is needed in the computation of $\mathrm{amgu}(d, u = x)$, while two star-unions are needed when computing $\mathrm{amgu}(d, v = w)$.

$$\begin{aligned} d_1 &= \mathrm{amgu}(d, u = x) \\ &= \bigl\langle \{uwxy, uwxz, uxy, uxz, v, w\}, \{u, x\}, \{u, x\} \bigr\rangle, \\ d_{1,2} &= \mathrm{amgu}(d_1, v = w) \\ &= \bigl\langle \{uvwxy, uvwxyz, uvwxz, uxy, uxz, vw\}, \varnothing, \varnothing \bigr\rangle. \end{aligned}$$

Using the other ordering, we have:

$$\begin{aligned} d_2 &= \mathrm{amgu}(d, v = w) \\ &= \bigl\langle \{u, uvw, vw, xy, xz\}, \{x\}, \{x\} \bigr\rangle, \\ d_{2,1} &= \mathrm{amgu}(d_2, u = x) \\ &= \bigl\langle \{uvwxy, uvwxz, uxy, uxz, vw\}, \varnothing, \varnothing \bigr\rangle. \end{aligned}$$

Note that in $d_{2,1}$ variables y and z are independent, whereas they may share in $d_{1,2}$. Thus, in this example, by delaying the only binding that requires the star-unions, $v = w$, the number of known independent pairs is decreased.

Another possibility is to consider a heuristic that uses the numbers of free and linear variables as a measure of precision for local optimization. That is, it chooses first those bindings for which these numbers are maximal. However, the last example shown above is evidence that even such a proposal may also cause precision losses (the binding $u = x$ would be chosen first as it preserves the freeness of variable u).

In order to evaluate the effects of these two heuristics on real programs, we have implemented and compared them with respect to the "straight" abstract computation, which considers the non-grounding bindings using the left-to-right order.[16] The results reported in Tables 4 and 5 can be summarized as follows:

1. the precision on the groundness and freeness components is not affected;
2. the precision on the independent pairs and linearity components is rarely affected, in particular when considering goal-dependent analyses;
3. even for real programs, as was the case for the artificial examples given above, the precision can be increased as well as decreased.

Looking at Tables 4 and 5, it can be seen that the heuristic based on freeness and linearity information is slightly better than the use of the straight order, which, in its turn, is slightly better than the heuristic based on the number of star-unions.

Clearly, since these results could not be generalized to other orderings, our investigation cannot be considered really conclusive. Besides designing "smarter" heuristics, it would be interesting to provide a kind of *responsiveness test* for the underlying domain with respect to the choice of ordering for the non-grounding bindings: a

[16] The base domain is $Pos \times SFL$, both with and without structural information.

Goal Independent	without Struct Info					with Struct Info				
Prec. class	O	I	G	F	L	O	I	G	F	L
$0 < p \leq 2$	0.9	—	—	—	0.9	—	—	—	—	—
same precision	94.6	95.5	96.4	96.4	95.5	91.3	91.3	93.1	93.1	93.1
unknown	3.6	3.6	3.6	3.6	3.6	6.9	6.9	6.9	6.9	6.9
$-2 \leq p < 0$	0.9	0.9	—	—	—	1.8	1.8	—	—	—

Goal Dependent	without Struct Info					with Struct Info				
Prec. class	O	I	G	F	L	O	I	G	F	L
same precision	96.4	96.4	96.4	96.4	96.4	90.5	90.5	90.5	90.5	90.5
unknown	3.6	3.6	3.6	3.6	3.6	9.5	9.5	9.5	9.5	9.5

Time diff. class	Goal Ind.		Goal Dep.	
	w/o SI	with SI	w/o SI	with SI
degradation > 1	4.5	3.0	7.2	4.1
$0.5 <$ degradation ≤ 1	0.6	0.3	—	—
$0.2 <$ degradation ≤ 0.5	2.4	0.9	0.5	0.5
$0.1 <$ degradation ≤ 0.2	1.5	0.6	0.5	0.5
both timed out	3.0	6.3	3.6	9.5
same time	80.7	80.7	85.5	76.9
$0.1 <$ improvement ≤ 0.2	1.5	1.2	0.5	0.5
$0.2 <$ improvement ≤ 0.5	1.8	1.2	1.4	2.3
$0.5 <$ improvement ≤ 1	0.9	0.6	—	0.9
improvement > 1	3.0	5.1	0.9	5.0

Total time class	Goal Independent						Goal Dependent					
	without SI			with SI			without SI			with SI		
	%1	%2	Δ	%1	%2	Δ	%1	%2	Δ	%1	%2	Δ
timed out	3.3	3.3	—	6.6	6.6	—	3.6	3.6	—	9.5	9.5	—
$t > 10$	9.0	8.1	-0.9	8.4	9.0	0.6	7.2	7.7	0.5	8.6	8.1	-0.5
$5 < t \leq 10$	0.3	0.9	0.6	1.5	1.2	-0.3	1.4	0.9	-0.5	1.8	2.7	0.9
$1 < t \leq 5$	7.5	7.5	—	6.6	6.3	-0.3	3.6	3.2	-0.5	5.0	4.1	-0.9
$0.5 < t \leq 1$	2.7	2.4	-0.3	3.3	3.0	-0.3	5.4	5.9	0.5	3.2	3.6	0.5
$0.2 < t \leq 0.5$	8.4	9.3	0.9	10.2	10.5	0.3	13.1	12.7	-0.5	13.6	13.1	-0.5
$t \leq 0.2$	68.7	68.4	-0.3	63.3	63.3	—	65.6	66.1	0.5	58.4	58.8	0.5

Table 4. *The heuristic based on the number of star-unions.*

simple test consists in measuring how much the precision can be affected, in either way, by the application of an almost arbitrary order. This is the motivation for the comparison reported in Table 6, where the order is from right-to-left, the reverse of the usual one. As for the results given in Tables 4 and 5, the number of changes to the precision observed in Table 6 is small and all the observations made above

Goal Independent	without Struct Info					with Struct Info				
Prec. class	O	I	G	F	L	O	I	G	F	L
$5 < p \leq 10$	0.3	—	—	—	0.3	0.3	—	—	—	0.3
$0 < p \leq 2$	0.9	—	—	—	0.9	2.7	2.4	—	—	0.3
same precision	94.3	95.5	96.4	96.4	95.2	89.5	90.1	93.4	93.4	92.8
unknown	3.6	3.6	3.6	3.6	3.6	6.6	6.6	6.6	6.6	6.6
$-2 \leq p < 0$	0.6	0.6	—	—	—	0.9	0.9	—	—	—
$p < -20$	0.3	0.3	—	—	—	—	—	—	—	—

Goal Dependent	without Struct Info					with Struct Info				
Prec. class	O	I	G	F	L	O	I	G	F	L
$0 < p \leq 2$	0.5	—	—	—	0.5	—	—	—	—	—
same precision	94.6	95.0	95.5	95.5	95.0	89.6	89.6	89.6	89.6	89.6
unknown	4.5	4.5	4.5	4.5	4.5	10.4	10.4	10.4	10.4	10.4
$-20 \leq p < -10$	0.5	0.5	—	—	—	—	—	—	—	—

Time diff. class	Goal Ind.		Goal Dep.	
	w/o SI	with SI	w/o SI	with SI
degradation > 1	6.9	4.8	8.1	7.7
$0.5 <$ degradation ≤ 1	2.1	1.5	1.8	0.5
$0.2 <$ degradation ≤ 0.5	2.4	1.8	1.8	2.7
$0.1 <$ degradation ≤ 0.2	1.2	3.3	2.3	3.2
both timed out	2.4	5.7	3.6	9.0
same time	77.4	73.5	78.7	71.9
$0.1 <$ improvement ≤ 0.2	1.2	0.3	—	—
$0.2 <$ improvement ≤ 0.5	0.6	1.8	0.9	0.9
$0.5 <$ improvement ≤ 1	0.9	—	0.5	—
improvement > 1	4.8	7.2	2.3	4.1

Total time class	Goal Independent						Goal Dependent					
	without SI			with SI			without SI			with SI		
	%1	%2	Δ	%1	%2	Δ	%1	%2	Δ	%1	%2	Δ
timed out	3.3	2.7	-0.6	6.6	5.7	-0.9	3.6	4.5	0.9	9.5	10.0	0.5
$t > 10$	9.0	9.6	0.6	8.4	8.7	0.3	7.2	6.8	-0.5	8.6	7.7	-0.9
$5 < t \leq 10$	0.3	2.1	1.8	1.5	1.8	0.3	1.4	1.4	—	1.8	2.7	0.9
$1 < t \leq 5$	7.5	6.0	-1.5	6.6	6.9	0.3	3.6	4.5	0.9	5.0	5.0	—
$0.5 < t \leq 1$	2.7	3.0	0.3	3.3	3.9	0.6	5.4	4.1	-1.4	3.2	3.6	0.5
$0.2 < t \leq 0.5$	8.4	9.9	1.5	10.2	13.3	3.0	13.1	13.1	—	13.6	15.4	1.8
$t \leq 0.2$	68.7	66.6	-2.1	63.3	59.6	-3.6	65.6	65.6	—	58.4	55.7	-2.7

Table 5. *The heuristic based on freeness and linearity.*

still hold. Surprisingly, this reversed ordering provides marginally better precision results than those obtained using the considered heuristics.[17]

7 The Reduced Product between Pos and Sharing

The overlap between the information provided by *Pos* and the information provided by Sharing mentioned in Section 4 means that the Cartesian product $Pos \times SFL$ contains redundancy, that is, there is more than one element that can characterize the same set of concrete computational states.

In (Bagnara et al. 2000), two techniques that are able to remove some of this redundancy were experimentally evaluated. One of these aims at identifying those pairs of variables (x, y) for which the Boolean formula of the *Pos* component implies the *binary disjunction* $x \vee y$. In such a case, it is always safe to assume that the variables x and y are independent.[18] Since the number of independent pairs is one of the quantities explicitly measured, this enhancement has the potential for "immediate" precision gains. The other technique exploits the knowledge of the sets of *ground-equivalent* variables: the variables in $e \subseteq VI$ are ground-equivalent in $\phi \in Pos$ if and only if, for each $x, y \in e$, $\phi \models (x \leftrightarrow y)$. For a description of how these sets can be used to improve sharing analysis, the reader is referred to (Bagnara et al. 2000). The main motivation for experimenting with this specific reduction was the ease of its implementation, since all the needed information can easily be recovered from the already computed E component of the GER implementation of *Pos* (Bagnara and Schachte 1999). The experimental evaluation results given in (Bagnara et al. 2000) for these two techniques show precision improvements with only three of the programs and, also, only with respect to the number of independent pairs that were found. Those results just apply to these limited forms of reduction, so could not be considered a complete account of all the possible precision gains.

The full reduced product (Cousot and Cousot 1979) between *Pos* and Sharing has been elegantly characterized in (Codish et al. 1999), where set-sharing *à la* Jacobs and Langen is expressed in terms of elements of the *Pos* domain itself. Let $[\phi]_{VI}$ denote the set of all the models of the Boolean function ϕ defined over the set of variables VI. Then, the isomorphism maps each set-sharing element $sh \in SH$ into the Boolean formula $\phi \in Pos$ such that

$$[\phi]_{VI} = \{\, VI \setminus S \mid S \in sh \,\} \cup \{\, VI \}.$$

The sharing information encoded by an element $(\phi_g, \phi_{sh}) \in Pos \times Pos$ can be improved by replacing the second component (that is, the Boolean formula describing set-sharing information) with the conjunction $\phi_g \wedge \phi_{sh}$. The reader is referred to (Codish et al. 1999) for a complete account of this composition and a justification of its correctness.

[17] It is worth noting that the only precision improvement reported in Table 6 for the goal-dependent analysis with structural information (caused by the program `semi`) corresponds to the precision decrease reported in Table 2. This confirms that, as informally discussed in Section 5, such a precision decrease was due to the non-commutativity of the amgu operator on $Pos \times SFL$.

[18] Note that this observation dates back, at least, to (Crnogorac et al. 1996).

Goal Independent	without Struct Info					with Struct Info				
Prec. class	O	I	G	F	L	O	I	G	F	L
$5 < p \leq 10$	0.3	—	—	—	0.3	0.3	—	—	—	0.3
$0 < p \leq 2$	0.9	0.3	—	—	0.6	4.2	3.0	—	—	1.2
same precision	94.3	95.2	96.4	96.4	95.5	87.7	89.2	93.4	93.4	91.9
unknown	3.6	3.6	3.6	3.6	3.6	6.6	6.6	6.6	6.6	6.6
$-2 \leq p < 0$	0.6	0.6	—	—	—	1.2	1.2	—	—	—
$p < -20$	0.3	0.3	—	—	—	—	—	—	—	—

Goal Dependent	without Struct Info					with Struct Info				
Prec. class	O	I	G	F	L	O	I	G	F	L
$0 < p \leq 2$	0.5	—	—	—	0.5	0.5	—	—	—	0.5
same precision	95.5	95.9	96.4	96.4	95.9	90.0	90.5	90.5	90.5	90.0
unknown	3.6	3.6	3.6	3.6	3.6	9.5	9.5	9.5	9.5	9.5
$-20 \leq p < -10$	0.5	0.5	—	—	—	—	—	—	—	—

Time diff. class	Goal Ind.		Goal Dep.	
	w/o SI	with SI	w/o SI	with SI
degradation > 1	4.2	6.0	4.5	6.8
$0.5 <$ degradation ≤ 1	0.6	0.6	—	—
$0.2 <$ degradation ≤ 0.5	2.4	1.5	1.4	0.9
$0.1 <$ degradation ≤ 0.2	1.8	0.9	0.5	—
both timed out	2.4	5.7	3.6	9.0
same time	78.3	76.2	82.8	74.2
$0.1 <$ improvement ≤ 0.2	1.5	1.2	1.8	0.9
$0.2 <$ improvement ≤ 0.5	1.8	0.3	1.4	1.8
$0.5 <$ improvement ≤ 1	0.9	0.9	0.5	0.5
improvement > 1	6.0	6.6	3.6	5.9

Total time class	Goal Independent						Goal Dependent					
	without SI			with SI			without SI			with SI		
	%1	%2	Δ	%1	%2	Δ	%1	%2	Δ	%1	%2	Δ
timed out	3.3	2.7	-0.6	6.6	5.7	-0.9	3.6	3.6	—	9.5	9.0	-0.5
$t > 10$	9.0	8.7	-0.3	8.4	9.9	1.5	7.2	7.7	0.5	8.6	8.1	-0.5
$5 < t \leq 10$	0.3	1.8	1.5	1.5	1.5	—	1.4	0.5	-0.9	1.8	2.7	0.9
$1 < t \leq 5$	7.5	6.9	-0.6	6.6	6.0	-0.6	3.6	3.2	-0.5	5.0	4.5	-0.5
$0.5 < t \leq 1$	2.7	2.4	-0.3	3.3	2.7	-0.6	5.4	5.4	—	3.2	3.6	0.5
$0.2 < t \leq 0.5$	8.4	8.7	0.3	10.2	11.1	0.9	13.1	13.1	—	13.6	12.2	-1.4
$t \leq 0.2$	68.7	68.7	—	63.3	63.0	-0.3	65.6	66.5	0.9	58.4	59.7	1.4

Table 6. *Reversing the ordering of the non-grounding bindings.*

Goal Independent	without Struct Info					with Struct Info				
Prec. class	O	I	G	F	L	O	I	G	F	L
$5 < p \leq 10$	—	—	—	—	—	0.3	0.3	—	—	—
$2 < p \leq 5$	0.3	0.3	—	—	—	—	—	—	—	—
$0 < p \leq 2$	2.7	2.7	—	—	0.6	3.9	3.9	—	—	0.6
same precision	86.1	86.1	89.2	89.2	88.6	80.7	80.7	84.9	84.9	84.3
unknown	10.8	10.8	10.8	10.8	10.8	15.1	15.1	15.1	15.1	15.1

Goal Dependent	without Struct Info					with Struct Info				
Prec. class	O	I	G	F	L	O	I	G	F	L
$p > 20$	0.5	0.5	—	—	—	—	—	—	—	—
$10 < p \leq 20$	—	—	—	—	—	0.5	0.5	—	—	—
$5 < p \leq 10$	—	—	—	—	—	0.5	0.5	—	—	—
$0 < p \leq 2$	2.7	2.7	—	—	—	2.7	2.7	—	—	—
same precision	89.1	89.1	92.3	92.3	92.3	77.8	77.8	81.4	81.4	81.4
unknown	7.7	7.7	7.7	7.7	7.7	18.6	18.6	18.6	18.6	18.6

Table 7. $Pos \times SFL_2$ *versus* $Pos \otimes SFL$.

This specification of the reduced product can be reformulated, using the standard set-sharing representation for the second component, to define a reduction procedure reduce: $Pos \times SH \rightarrow SH$ such that, for all $\phi_g \in Pos$, $sh \in SH$,

$$\mathrm{reduce}(\phi_g, sh) = \bigl\{\, S \in sh \bigm| (VI \setminus S) \in [\phi_g]_{VI} \,\bigr\}.$$

The enhanced integration of *Pos* and *SFL*, based on the above reduction operator, is denoted here by $Pos \otimes SFL$. From a formal point of view, this is *not* the reduced product between *Pos* and *SFL*: while there is a complete reduction between *Pos* and *SH*, the same does not necessarily hold for the combination with freeness and linearity information. Also note that the domain $Pos \otimes SFL$ is strictly more precise than the domain $\mathsf{Sh}^{\mathsf{PSh}}$, defined in (Scozzari 2000) for pair-sharing analysis. This is because the domain $\mathsf{Sh}^{\mathsf{PSh}}$ is the reduced product of a strict abstraction of *Pos* and a strict abstraction of *SH*.

When using the domain *PSD* in place of *SH*, the 'reduce' operator specified above can interact in subtle ways with an implementation removing the ρ-redundant sharing groups from the elements of *PSD*. The following is an example where such an interaction provides results that are not correct.

Let $VI = \{x, y, z\}$ and $sh = \{xy, xz, yz, xyz\} \in PSD$ be the current set-sharing description. Suppose that the implementation internally represents *sh* by using the ρ-reduced element $sh_{\mathrm{red}} = \{xy, xz, yz\}$, so that $sh = \rho(sh_{\mathrm{red}})$. Suppose also that the groundness description computed on the domain *Pos* is $\phi_g = (x \leftrightarrow y \leftrightarrow z)$. Note that we have $[\phi_g]_{VI} = \bigl\{\varnothing, \{x, y, z\}\bigr\}$. Then we have

$$\begin{aligned} sh' &= \mathrm{reduce}(sh, \phi_g) = \{xyz\}; \\ sh'_{\mathrm{red}} &= \mathrm{reduce}(sh_{\mathrm{red}}, \phi_g) = \varnothing. \end{aligned}$$

The two *Pos*-reduced elements sh' and sh'_{red} are not equivalent, even modulo ρ.

Note that the above example does not mean that the reduced product between *Pos* and *PSD* yields results that are not correct; neither does it mean that it is less precise than the reduced product between *Pos* and *SH* for the computation of the observables. More simply, the optimizations used in our current implementation of *PSD* are not compatible with the above reduction process. Therefore, in Table 7 we show the precision results obtained when comparing the base domain $Pos \times SFL_2$ with the domain $Pos \otimes SFL$: the implementation of $Pos \otimes SFL$, by avoiding ρ-reductions, is not affected by the correctness problem mentioned above.

The precision comparison provides empirical evidence that $Pos \otimes SFL$ is more effective than the combination considered in (Bagnara et al. 2000). However, as indicated by the number of time-outs reported in Table 7, using $Pos \otimes SFL$ is not feasible due to its intrinsic exponential complexity. We deliberately decided not to include the time comparison, since it would have provided no information at all: the efficiency degradations, which are largely caused by the lack of ρ-reductions, should not be attributed to the enhanced combination with *Pos*. In this respect, the reader looking for more details is referred to (Bagnara et al. 2002).

For the only purpose of investigating how many precision improvements may have been missed in the previous comparison due to the high number of time-outs, we have performed another experimental evaluation where we have compared the base domain $Pos \times SFL_2$ and the domain $Pos \otimes SFL_2$. We stress the fact that, given the observation made previously, such a precision comparison provides an *over-estimation* for the actual improvements that can be obtained by a correct integration of the ρ-reduction and the 'reduce' operators. A detailed investigation of the experimental data, which cannot be reported here for space reasons, has shown that the number of precision improvements shown in Table 7 could at most double. In particular, improvements are more likely to occur for goal-independent analyses.

8 Ground-or-free Variables

Most of the ideas investigated in the present work are based on earlier work by other authors. In this section, we describe one originally proposed in (Bagnara et al. 2000). Consider the analysis of the binding $x = t$ and suppose that, on a set of computation paths, this binding is reached with x ground while, on the remaining computation paths, the binding is reached with x free. In both cases x will be linear and this is all that will be recorded when using the usual combination $Pos \times SFL$. This information is valuable since, in the case that x and t are independent, it allows the star-union operation for the relevant component for t to be dispensed with. However, the information that is lost, that is, x being either ground or free, is equally valuable, since this would allow the avoidance of the star-union of *both* the relevant components for x and t, even when x and t may share. This loss has the disadvantages that CPU time is wasted by performing unnecessary but costly operations and that the precision is potentially degraded: not only are the extra star-unions useless for correctness but may introduce redundant sharing groups to the detriment of accuracy. It is therefore useful to track the additional mode 'ground-or-free'.

The analysis domain *SFL* is extended with the component $GF \stackrel{\mathrm{def}}{=} \wp(VI)$ consisting of the set of variables that are known to be either ground or free. As for freeness and linearity, the approximation ordering on *GF* is given by reverse subset inclusion. When computing the abstract mgu on the new domain

$$SGFL \stackrel{\mathrm{def}}{=} SH \times F \times GF \times L,$$

the property of being ground-or-free is used and propagated in almost the same way as freeness information.

Definition 6
(Improved abstract operations over *SGFL*.) Let $d = \langle sh, f, gf, l\rangle \in SGFL$. We define the predicate $\mathit{gfree}_d \colon \mathit{Terms} \to \mathit{Bool}$ such that, for each first order term t, where $V_t \stackrel{\mathrm{def}}{=} \mathit{vars}(t) \subseteq VI$,

$$\mathit{gfree}_d(t) \stackrel{\mathrm{def}}{=} \bigl(\mathrm{rel}(V_t, sh) = \varnothing\bigr) \lor (\exists x \in VI \mathrel{.} x = t \land x \in gf).$$

Consider the specification of the abstract operations over *SFL* given in Definition 4. The improved operator $\mathrm{amgu} \colon SGFL \times \mathit{Bind} \to SGFL$ is given by

$$\mathrm{amgu}\bigl(d, x = t\bigr) \stackrel{\mathrm{def}}{=} \langle sh', f', gf', l'\rangle,$$

where f' and l'' are defined as in Definition 4 and

$$\begin{aligned}
sh' &= \overline{\mathrm{rel}}(V_{xt}, sh) \cup \mathrm{bin}\bigl(S_x, S_t\bigr);\\
S_x &= \begin{cases} R_x, & \text{if } \mathit{gfree}_d(x) \lor \mathit{gfree}_d(t) \lor \bigl(\mathit{lin}_d(t) \land \mathit{ind}_d(x,t)\bigr);\\ R_x^\star, & \text{otherwise;}\end{cases}\\
S_t &= \begin{cases} R_t, & \text{if } \mathit{gfree}_d(x) \lor \mathit{gfree}_d(t) \lor \bigl(\mathit{lin}_d(x) \land \mathit{ind}_d(x,t)\bigr);\\ R_t^\star, & \text{otherwise;}\end{cases}\\
gf' &= \bigl(VI \setminus \mathit{vars}(sh')\bigr) \cup gf'';\\
gf'' &= \begin{cases} gf, & \text{if } \mathit{gfree}_d(x) \land \mathit{gfree}_d(t);\\ gf \setminus \mathit{vars}(R_x), & \text{if } \mathit{gfree}_d(x);\\ gf \setminus \mathit{vars}(R_t), & \text{if } \mathit{gfree}_d(t);\\ gf \setminus \mathit{vars}(R_x \cup R_t), & \text{otherwise;}\end{cases}\\
l' &= gf' \cup l''.
\end{aligned}$$

The computation of the set gf'' is very similar to the computation of the set f' as given in Definition 4. The new ground-or-free component gf' is obtained by adding to gf'' the set of all the ground variables: in other words, if a variable "loses freeness" then it also loses its ground-or-free status unless it is known to be definitely ground. It can be noted that, in the computation of this improved amgu, the ground-or-free property takes the role previously played by freeness. In particular, when computing sh', all the tests for freeness have been replaced by tests on the newly defined Boolean function gfree_d; similarly, in the computation of the new linearity component l', the set f' has been replaced by gf' (since any ground-or-free variable is also linear). It is also easy to generalize the improvement for definitely cyclic bindings introduced in Definition 5 to the domain *SGFL*: as before, the test $\mathit{free}_d(x)$ needs to be replaced with the new test $\mathit{gfree}_d(x)$.

To summarize, the incorporation of the set of ground-or-free variables is cheap, both in terms of computational complexity and in terms of code to be written. As far as computational complexity is concerned this extension looks particularly promising, since the possibility of avoiding star-unions has the potential of absorbing its overhead if not of giving rise to a speed-up.

Thus the domain $\mathit{Pos} \times \mathit{SGFL}$ was experimentally evaluated on our benchmark suite, with and without the structural information provided by Pattern$(\cdot)$, both in a goal-dependent and in a goal-independent way, and the results compared with those previously obtained for the domain $\mathit{Pos} \times \mathit{SFL}$. Note that the implementation uses the non-redundant version $\mathit{SGFL}_2 \stackrel{\mathrm{def}}{=} \mathit{PSD} \times F \times \mathit{GF} \times L$. In the precision comparisons of Table 8, the new column labeled GF reports precision improvements measured on the ground-or-free property itself.[19]

As far as the timings are concerned, the experimentation fully confirms our qualitative reasoning: efficiency improvements are more frequent than degradations and, even with widening operators switched off, the distributions of the total analysis times show minor changes only. As for precision, disregarding the many improvements in the GF columns, few changes can be observed, and almost all of these concern just the linearity information.[20]

The results in Table 8, show that tracking ground-or-free variables, while being potentially useful for improving the precision of a sharing analysis, rarely reaches such a goal. In contrast, the precision gains on the ground-or-free property itself are remarkable, affecting from 39% to 74% of the programs in the benchmark suite. It is possible to foresee several *direct* applications for this information that, together with the just mentioned negligible computational cost, fully justify the inclusion of this enhancement in a static analyzer. In particular, there are at least two ways in which a knowledge of ground-or-free variables could improve the concrete unification procedure.

The first case applies in the context of occurs-check reduction (Søndergaard 1986; Crnogorac et al. 1996), that is when a program designed for a logic programming system performing the occurs-check is to be run on top of a system omitting this test. In order to ensure correct execution, all the explicit and implicit unifications in the program are treated as if the ISO Prolog built-in `unify_with_occurs_check/2` was used to perform them. In order to minimize the performance overhead, it is important to detect, as precisely as possible and at compile-time, those *NSTO* (short for *Not Subject To the Occurs-check* (Deransart et al. 1991; ISO/IEC 1995)) unifications where the occurs-check will not be needed. For these unifications, `=/2`

[19] For this comparison, in the analysis using $\mathit{Pos} \times \mathit{SFL}$, the number of ground-or-free variables is computed by summing the number of ground variables with the number of free variables.

[20] In fact the sole improvement to the number of independent pairs is due to a synthetic benchmark, named `gof`, that was explicitly written to show that variable independence could be affected.

Goal Ind.	without Struct Info						with Struct Info					
Prec. class	O	I	G	F	GF	L	O	I	G	F	GF	L
$p > 20$	52.7	0.3	—	—	52.7	—	48.5	0.3	—	—	48.5	—
$10 < p \leq 20$	11.7	—	—	—	11.7	—	16.0	—	—	—	16.0	—
$5 < p \leq 10$	5.4	—	—	—	5.4	—	7.5	—	—	—	7.5	—
$2 < p \leq 5$	2.4	—	—	—	2.4	—	1.8	—	—	—	1.8	—
$0 < p \leq 2$	0.3	—	—	—	0.3	1.5	0.6	—	—	—	0.6	1.5
same precision	24.1	96.4	96.7	96.7	24.1	95.2	19.0	93.1	93.4	93.4	19.0	91.9
unknown	3.3	3.3	3.3	3.3	3.3	3.3	6.6	6.6	6.6	6.6	6.6	6.6

Goal Dep.	without Struct Info						with Struct Info					
Prec. class	O	I	G	F	GF	L	O	I	G	F	GF	L
$p > 20$	5.9	—	—	—	5.9	—	5.9	—	—	—	5.9	—
$10 < p \leq 20$	4.5	—	—	—	4.5	—	5.4	—	—	—	5.4	—
$5 < p \leq 10$	7.7	0.5	—	—	7.7	—	5.4	0.5	—	—	5.4	—
$2 < p \leq 5$	13.1	—	—	—	13.1	—	12.2	—	—	—	12.2	—
$0 < p \leq 2$	8.1	—	—	—	8.1	0.5	10.0	—	—	—	10.0	—
same precision	57.0	95.9	96.4	96.4	57.0	95.9	51.6	90.0	90.5	90.5	51.6	90.5
unknown	3.6	3.6	3.6	3.6	3.6	3.6	9.5	9.5	9.5	9.5	9.5	9.5

Time diff. class	Goal Ind.		Goal Dep.	
	w/o SI	with SI	w/o SI	with SI
degradation > 1	—	0.6	—	0.9
$0.5 <$ degradation ≤ 1	0.3	—	0.5	—
$0.2 <$ degradation ≤ 0.5	—	0.6	0.5	1.4
$0.1 <$ degradation ≤ 0.2	0.3	—	—	0.5
both timed out	3.3	6.6	3.6	9.5
same time	88.6	85.2	87.3	82.8
$0.1 <$ improvement ≤ 0.2	1.2	1.2	1.8	1.4
$0.2 <$ improvement ≤ 0.5	2.4	2.4	1.8	0.9
$0.5 <$ improvement ≤ 1	2.1	0.9	2.3	0.9
improvement > 1	1.8	2.4	2.3	1.8

Total time class	Goal Independent						Goal Dependent					
	without SI			with SI			without SI			with SI		
	%1	%2	Δ	%1	%2	Δ	%1	%2	Δ	%1	%2	Δ
timed out	3.3	3.3	—	6.6	6.6	—	3.6	3.6	—	9.5	9.5	—
$t > 10$	9.0	9.0	—	8.4	8.4	—	7.2	7.2	—	8.6	8.6	—
$5 < t \leq 10$	0.3	0.3	—	1.5	1.5	—	1.4	1.4	—	1.8	1.8	—
$1 < t \leq 5$	7.5	7.5	—	6.6	6.6	—	3.6	3.6	—	5.0	5.0	—
$0.5 < t \leq 1$	2.7	2.7	—	3.3	3.6	0.3	5.4	5.9	0.5	3.2	3.2	—
$0.2 < t \leq 0.5$	8.4	8.7	0.3	10.2	10.5	0.3	13.1	12.7	-0.5	13.6	14.0	0.5
$t \leq 0.2$	68.7	68.4	-0.3	63.3	62.7	-0.6	65.6	65.6	—	58.4	57.9	-0.5

Table 8. $Pos \times SFL_2$ *versus* $Pos \times SGFL_2$.

can safely be used; for the remaining ones, the program will have to be transformed so that `unify_with_occurs_check/2` is explicitly called to perform them. Ground-or-freeness can be of help for this application, since a unification between two ground-or-free variables is *NSTO*. Note that this is an improvement with respect to the technique used in (Crnogorac et al. 1996), since it is not required that the two considered variables are independent.

As a second application, ground-or-freeness can be useful to replace the full concrete unification procedure by a simplified version. Since a ground-or-free term is either ground or free, a *single* run-time test for freeness will discriminate between the two cases: if this test succeeds, unification can be implemented by a single assignment; if the test fails, any specialized code for unification with a ground term can be safely invoked. In particular, when unifying two ground-or-free variables that are not free at run-time, the full unification procedure can be replaced by a simpler recursive test for equivalence.

9 More Precise Exploitation of Linearity

In (King 1994), A. King proposes a domain for sharing analysis that performs a quite precise tracking of linearity. Roughly speaking, each sharing group in a sharing-set carries its own linearity information. In contrast, in the approach of (Langen 1990), which is the one usually followed, a set of definitely linear variables is recorded along with each sharing-set. The proposal in (King 1994) gives rise to a domain that is quite different from the ones presented here. Since (King 1994) does not provide an experimental evaluation and we are unaware of any subsequent work on the subject, the question whether this more precise tracking of linearity is actually worthwhile (both in terms of precision and efficiency) seems open.

What interests us here is that part of the theoretical work presented in (King 1994) may be usefully applied even in the more classical treatments of linearity such as the one being used in this paper. As far as we can tell, this fact was first noted in (Bagnara et al. 2000).

In (King 1994), point 3 of Lemma 5 (which is reported to be proven in (King 1993)) states that, if s is a linear term independent from a term t, then in the unifier for $s = t$ any sharing between the variables in s is necessarily caused by those variables that can occur more than once in t.

This result can be exploited even when using the domain *SFL*. Given the abstract element $d = \langle sh, f, l\rangle$, let $x \in (l \setminus f)$ be a non-free but linear variable and let t be a non-linear term such that $ind_d(x, t)$. Let also V_x, V_t, V_{xt}, R_x and R_t be as given in Definition 4. In such a situation, when abstractly evaluating the binding $x = t$, the standard amgu operator gives the set-sharing component

$$sh' = \overline{\mathrm{rel}}(V_{xt}, sh) \cup \mathrm{bin}(R_x^\star, R_t).$$

Suppose the set V_t is partitioned into the two components V_t^{l} and V_t^{nl}, where V_t^{nl} is the set of the "problematic" variables, that is, those variables that potentially

	Goal Independent					Goal Dependent				
Prec. class	O	I	G	F	L	O	I	G	F	L
$p > 20$	0.3	0.3	—	—	—	—	—	—	—	—
$2 < p \leq 5$	—	—	—	—	—	0.5	0.5	—	—	—
same precision	93.1	93.1	93.4	93.4	93.4	90.0	90.0	90.5	90.5	90.5
unknown	6.6	6.6	6.6	6.6	6.6	9.5	9.5	9.5	9.5	9.5

Time difference class	% benchmarks	
	Goal Ind.	Goal Dep.
degradation > 1	0.3	—
$0.5 <$ degradation ≤ 1	—	—
$0.2 <$ degradation ≤ 0.5	—	—
$0.1 <$ degradation ≤ 0.2	0.3	0.5
both timed out	6.6	9.5
same time	85.2	83.7
$0.1 <$ improvement ≤ 0.2	0.9	1.8
$0.2 <$ improvement ≤ 0.5	2.4	0.5
$0.5 <$ improvement ≤ 1	0.6	2.7
improvement > 1	3.6	1.4

Total time class	Goal Ind.			Goal Dep.		
	%1	%2	Δ	%1	%2	Δ
timed out	6.6	6.6	—	9.5	9.5	—
$t > 10$	8.4	8.4	—	8.6	8.6	—
$5 < t \leq 10$	1.5	1.5	—	1.8	1.8	—
$1 < t \leq 5$	6.6	6.6	—	5.0	5.0	—
$0.5 < t \leq 1$	3.3	3.3	—	3.2	3.2	—
$0.2 < t \leq 0.5$	10.2	11.1	0.9	13.6	14.0	0.5
$t \leq 0.2$	63.3	62.3	-0.9	58.4	57.9	-0.5

Table 9. *The effect of enhanced linearity on* $\mathrm{Pattern}(\mathit{Pos} \times \mathit{SFL}_2)$.

make t a non-linear term. Formally,

$$V_t^{\mathrm{l}} \stackrel{\mathrm{def}}{=} \left\{ y \in \mathit{vars}(t) \;\middle|\; \begin{array}{l} y \in l \\ y \in \mathit{mvars}(t) \implies y \notin \mathit{vars}(sh) \\ \forall z \in \mathit{vars}(t) : \bigl(y = z \lor \mathit{ind}_d(y, z)\bigr) \end{array} \right\};$$

$$V_t^{\mathrm{nl}} \stackrel{\mathrm{def}}{=} V_t \setminus V_t^{\mathrm{l}}.$$

Let $R_t^{\mathrm{l}} = \mathrm{rel}(V_t^{\mathrm{l}}, sh)$ and $R_t^{\mathrm{nl}} = \mathrm{rel}(V_t^{\mathrm{nl}}, sh)$. Note that $R_t^{\mathrm{nl}} \neq \varnothing$, because t is a non-linear term. If also $R_t^{\mathrm{l}} \neq \varnothing$ then the standard amgu can be replaced by an improved version (denoted by amgu_k) computing the following set-sharing component:

$$sh'_k = \overline{\mathrm{rel}}(V_{xt}, sh) \cup \mathrm{bin}(R_x, R_t^{\mathrm{l}}) \cup \mathrm{bin}(R_x^{\star}, R_t^{\mathrm{nl}}).$$

As a consequence of King's result (King 1994, Lemma 5), only R_t^{nl} (the relevant component of *sh* with respect to the problematic variables V_t^{nl}) has to be combined with $R_x^\star$ while R_t^{l} can be combined with just R_x (without the star-union).

For a working example, suppose $VI = \{v, w, x, y, z\}$ is the set of variables of interest and consider the *SFL* element

$$d \stackrel{\mathrm{def}}{=} \bigl\langle \{vx, wx, y, z\}, \{v, w, y\}, \{v, w, x, y\} \bigr\rangle$$

with the binding $x = f(y, z)$. Note that all the applicability conditions specified above are met: in particular $t = f(y, z)$ is not linear because $z \notin l$. As $R_x = \{vx, wx\}$ and $R_t = \{y, z\}$, a standard analysis would compute

$$\begin{aligned} d' &= \mathrm{amgu}\bigl(d, x = f(y, z)\bigr) \\ &= \bigl\langle \{vwxy, vwxz, vxy, vxz, wxy, wxz\}, \varnothing, \{y\} \bigr\rangle. \end{aligned}$$

On the other hand, since $V_t^{\mathrm{l}} = \{y\}$ and $V_t^{\mathrm{nl}} = \{z\}$, the enhanced analysis would compute

$$\begin{aligned} d'_k &= \mathrm{amgu}_k\bigl(d, x = f(y, z)\bigr) \\ &= \bigl\langle \{vwxz, vxy, vxz, wxy, wxz\}, \varnothing, \{y\} \bigr\rangle. \end{aligned}$$

Note that d'_k does not include the sharing group $vwxy$. This means that, if in the sequel of the computation variable z is bound to a ground term, then variables v and w will be known to be definitely independent. This independence is not captured when using the standard amgu since d' includes the sharing group $vwxy$, and therefore the variables v and w will potentially share even after grounding z.

The experimental evaluation for this enhancement is reported in Table 9. The comparison of times shows that the efficiency of the analysis, when affected, is more likely to be improved than degraded. As for the precision, improvements are observed for only two programs; moreover, these are synthetic benchmarks such as the above example. Nevertheless, despite its limited practical relevance, this result demonstrates that the standard combination of Sharing with linearity information is *not* optimal, even when all the possible orderings of the non-grounding bindings are tried.

10 Sharing and Freeness

As noted by several authors (Bruynooghe et al. 1994a; Bueno et al. 1994; Cabeza and Hermenegildo 1994), the standard combination of Sharing and *Free* is not optimal. G. Filé (Filé 1994) formally identified the reduced product of these domains and proposed an improved abstract unification operator. This new operator exploits two properties that hold for the most precise abstract description of a *single* concrete substitution:

1. each free variable occurs in exactly one sharing group;
2. two free variables occur in the same sharing group if and only if they are aliases (i.e., they have become the same variable).

When considering the general case, where sets of concrete substitutions come into play, property 1 can be used to (partially) recover disjunctive information. In particular, it is possible to decompose an abstract description into a set of (maximal) descriptions that necessarily come from different computation paths, each one satisfying property 1. The abstract unification procedure can thus be computed separately on each component, and the results of each subcomputation are then joined to give the final description. As such components are more precise than the original description (they possibly contain more ground variables and less sharing pairs), precision gains can be obtained.

Furthermore, by exploiting property 2 on each component, it is possible to correctly infer that for some of them the computation will fail due to a functor clash (or to the occurs-check, if considering a system working on finite trees). Note that a similar improvement is possible even without decomposing the abstract description. As an example, consider an abstract element such as the following:

$$d = \bigl\langle \{xy, u, v\}, \{x, y\}, \{x, y\} \bigr\rangle.$$

Since the sharing group xy is the only one where the free variables x and y occur, property 2 states that x and y are indeed the same variable in all the concrete computation states described by $d \in \mathit{SFL}$. Therefore, when abstractly evaluating the substitution $\bigl\{x = f(u), y = g(v)\bigr\}$, it can be safely concluded that its concrete counterparts will result in failure due to the functor clash. In the same circumstances, it can also be concluded that a concrete substitution corresponding to, say, $\bigl\{x = f(y)\bigr\}$ will cause a failure of the occurs-check, if this is performed.

As was the case for the reduced product between *Pos* and *SH* (see Section 7), the interaction between the enhanced abstract unification operator and the elimination of ρ-redundant elements can lead to results that are not correct.

To see this, let $\mathit{VI} = \{w, x, y, z\}$ and consider the set of concrete substitutions $\Sigma = \wp(\sigma)$, where $\sigma = \{x \mapsto v, y \mapsto v, z \mapsto v\}$ (note that $v \notin \mathit{VI}$). The abstract element describing Σ is $d = \langle sh, f, l \rangle \in \mathit{SFL}$, where $sh = \{w, x, xy, xyz, xz, y, yz, z\}$ and $f = l = \mathit{VI}$. Suppose that the implementation represents d by using the reduced element $d_{\mathrm{red}} = \langle sh_{\mathrm{red}}, f, l \rangle$, where $sh_{\mathrm{red}} = sh \setminus \{xyz\}$, so that $sh = \rho(sh_{\mathrm{red}})$.

According to the specification of the enhanced operator, d_{red} can be decomposed into the following four components:

$$c_1 = \bigl\langle \{w, x, y, z\}, f, l \bigr\rangle, \qquad c_3 = \bigl\langle \{w, xz, y\}, f, l \bigr\rangle,$$
$$c_2 = \bigl\langle \{w, x, yz\}, f, l \bigr\rangle, \qquad c_4 = \bigl\langle \{w, xy, z\}, f, l \bigr\rangle.$$

Consider the binding $x = f(y, w)$ and, for each $i \in \{1, \ldots, 4\}$, the computation of $c_i' = \langle sh_i', f_i', l_i' \rangle = \mathrm{amgu}\bigl(c_i, x = f(y, w)\bigr)$, where we have $l_1' = l_2' = l_3' = \mathit{VI}$ and $l_4' = \{w, z\}$. In all four cases, we have $z \in l_i'$, so that z keeps its linearity even after merging the results of the four subcomputations into a single abstract description.

In contrast, when performing the same computation with the original abstract description d in the decomposition phase, we also obtain a fifth component,

$$c_5 = \bigl\langle \{w, xyz\}, f, l \bigr\rangle.$$

Goal Independent	without Struct Info					with Struct Info				
Prec. class	O	I	G	F	L	O	I	G	F	L
$p > 20$	0.3	0.3	—	—	—	—	—	—	—	—
$5 < p \leq 10$	—	—	—	—	—	0.3	—	—	—	0.3
$0 < p \leq 2$	0.9	0.3	—	—	0.6	3.6	3.0	—	—	0.6
same precision	94.6	95.2	95.8	95.8	95.2	86.1	87.0	90.1	90.1	89.2
unknown	4.2	4.2	4.2	4.2	4.2	9.9	9.9	9.9	9.9	9.9

Goal Dependent	without Struct Info					with Struct Info				
Prec. class	O	I	G	F	L	O	I	G	F	L
same precision	96.4	96.4	96.4	96.4	96.4	89.6	89.6	89.6	89.6	89.6
unknown	3.6	3.6	3.6	3.6	3.6	10.4	10.4	10.4	10.4	10.4

Time diff. class	Goal Ind.		Goal Dep.	
	w/o SI	with SI	w/o SI	with SI
degradation > 1	9.6	13.6	3.2	5.9
$0.5 <$ degradation ≤ 1	0.6	1.8	1.4	1.4
$0.2 <$ degradation ≤ 0.5	3.3	2.4	1.8	3.6
$0.1 <$ degradation ≤ 0.2	0.6	1.5	2.3	1.4
both timed out	3.3	6.6	3.6	9.5
same time	82.2	73.5	87.8	77.8
$0.1 <$ improvement ≤ 0.2	—	—	—	—
$0.2 <$ improvement ≤ 0.5	0.3	—	—	—
$0.5 <$ improvement ≤ 1	—	—	—	—
improvement > 1	—	0.6	—	0.5

Total time class	Goal Independent						Goal Dependent					
	without SI			with SI			without SI			with SI		
	%1	%2	Δ	%1	%2	Δ	%1	%2	Δ	%1	%2	Δ
timed out	3.3	4.2	0.9	6.6	9.9	3.3	3.6	3.6	—	9.5	10.4	0.9
$t > 10$	9.0	9.6	0.6	8.4	8.4	—	7.2	7.2	—	8.6	8.1	-0.5
$5 < t \leq 10$	0.3	0.9	0.6	1.5	1.2	-0.3	1.4	1.4	—	1.8	1.8	—
$1 < t \leq 5$	7.5	6.9	-0.6	6.6	5.7	-0.9	3.6	3.6	—	5.0	4.5	-0.5
$0.5 < t \leq 1$	2.7	2.1	-0.6	3.3	4.5	1.2	5.4	5.9	0.5	3.2	3.2	—
$0.2 < t \leq 0.5$	8.4	8.4	—	10.2	12.0	1.8	13.1	12.7	-0.5	13.6	14.9	1.4
$t \leq 0.2$	68.7	67.8	-0.9	63.3	58.1	-5.1	65.6	65.6	—	58.4	57.0	-1.4

Table 10. *The effect of enhanced freeness on* $Pos \times SFL_2$.

When computing $c'_5 = \langle sh'_5, f'_5, l'_5 \rangle = \mathrm{amgu}\bigl(c_5, x = f(y, w)\bigr)$, we obtain $l'_5 = \{w\}$, so that z loses its linearity when merging the five results into a single abstract description. Note that this is not an avoidable precision loss, since in the concrete computation path corresponding to the substitution σ we would have computed

$$\sigma' = \bigl\{x \mapsto f(x, w), y \mapsto f(y, w), z \mapsto f(z, w)\bigr\},$$

where z is bound to a non-linear term (namely, an infinite rational term with an infinite number of occurrences of variable w). Therefore, the result obtained when using the abstract description d_{red} is not correct.

As already observed in Section 7, the above correctness problem lies not in the SFL_2 domain itself, but rather in our optimized implementation, which removes the ρ-redundant elements from the set-sharing description.

We implemented the first idea by Filé (i.e., the exploitation of property 1) on the usual base domain $Pos \times SFL_2$. As noted above, this implementation may yield results that are not correct: the precision comparison reported in Table 10 provides an over-estimation of the actual improvements that could be obtained by a correct implementation. However, it is not possible to assess the magnitude of this over-estimation, since our implementation of this enhancement on the domain $Pos \times SFL$, where no ρ-redundancy elimination is performed, times-out on a large fraction of the benchmarks. The results in Table 10 show that precision improvements are only observed for goal-independent analysis. When looking at the time comparisons, it should be observed that the analysis of several programs had to be stopped because of the combinatorial explosion in the decomposition, even though we used the domain $Pos \times SFL_2$. Among the proposals experimentally evaluated in this paper, this one shows the worst trade-off between cost and precision.

Note that, in principle, such an approach to the recovery of disjunctive information can be pursued beyond the integration of sharing with freeness. In fact, by exploiting the ground-or-free information as in Section 8, it is possible to obtain decompositions where each component contains *at most one* occurrence (in contrast with the *exactly one* occurrence of Filé's idea) of each ground-or-free variable. In each component, the ground-or-free variable could then be "promoted" as either a ground variable (if it does not occur in the sharing groups of that component) or as a free variable (if it occurs in exactly one sharing group).

It would be interesting to experiment with the second idea of Filé. However, such a goal would require a big implementation effort, since at present there is no easy way to incorporate this enhancement into the modular design of the China analyzer.[21]

11 Tracking Compoundness

In (Bruynooghe et al. 1994a; Bruynooghe et al. 1994b), Bruynooghe and colleagues considered the combination of the standard set-sharing, freeness, and linearity domains with compoundness information. As for freeness and linearity, compoundness

[21] Roughly speaking, the *SFL* component should be able to produce some new (implicit) structural information and notify it to the enclosing Pattern($\cdot$) component, which would then need to combine this information with the (explicit) structural information already available. However, in order to be able to receive notifications from its parameter, the Pattern($\cdot$) component, which is implemented as a C++ template, would have to be heavily modified.

was represented by the set of variables that definitely have the corresponding property.

As discussed in (Bruynooghe et al. 1994a; Bruynooghe et al. 1994b), compoundness information is useful in its own right for clause indexing. Here though, the focus is on improving sharing information, so that the question to be answered is: can the tracking of compoundness improve the sharing analysis itself? This question is also considered in (Bruynooghe et al. 1994a; Bruynooghe et al. 1994b) where a technique is proposed that exploits the combination of sharing, freeness and compoundness. This technique relies on the presence of the occurs-check.

Informally, consider the binding $x = t$ together with an abstract description where x is a free variable, t is a compound term and x *definitely* shares with t. Since x is free, x is aliased to one of the variables occurring in t. As a consequence, the execution of the binding $x = t$ will fail due to the occurs-check. In a more general case, when only *possible* sharing information is available, the precision of the abstract description can be safely improved by removing, just before computing the abstract binding, all the sharing groups containing both x and a variable in t. In addition, if this reduction step removes all the sharing groups containing a free variable, then it can be safely concluded that the computation will fail.

To see how this works in practice, consider the binding $x = f(y, z)$ and the description $d_1 \stackrel{\text{def}}{=} \langle sh_1, f_1, l_1 \rangle \in \mathit{SFL}$ such that

$$\begin{aligned} sh_1 &\stackrel{\text{def}}{=} \{wx, xy, xz, y, z\}, \\ f_1 &\stackrel{\text{def}}{=} \{x\}, \\ l_1 &\stackrel{\text{def}}{=} \{w, x, y, z\}. \end{aligned}$$

Since x is free and $f(y, z)$ is compound, the sharing-groups xy and xz can be removed so that the amgu computation will give the set-sharing and linearity components

$$\begin{aligned} sh_1' &\stackrel{\text{def}}{=} \{wxy, wxz\}, \\ l_1' &\stackrel{\text{def}}{=} \{w, x, y, z\} \end{aligned}$$

instead of the less precise

$$\begin{aligned} sh_1' &\stackrel{\text{def}}{=} \{wxy, wxz, xy, xyz, xz\}, \\ l_1' &\stackrel{\text{def}}{=} \{w\}. \end{aligned}$$

Note that the precision improvement of this particular example could also be obtained by applying, in its full generality, the second technique proposed by Filé and sketched in the previous section. This is because the term with which x is unified is "explicitly" compound. However, if the term t was "implicitly" compound (i.e., if it was an abstract variable known to represent compound terms) then the technique by Filé would not be applicable. For example, consider the binding $x = y$ and the description $d_2 \stackrel{\mathrm{def}}{=} \langle sh_2, f_2, l_2 \rangle \in \mathit{SFL}$ such that

$$\begin{aligned} sh_2 &\stackrel{\mathrm{def}}{=} \{wx, xyz, y\}, \\ f_2 &\stackrel{\mathrm{def}}{=} \{x\}, \\ l_2 &\stackrel{\mathrm{def}}{=} \{w, x, y, z\} \end{aligned}$$

supplemented by a compoundness component ensuring that y is compound. Then the sharing-group xyz can be removed so that the amgu will compute

$$\begin{aligned} sh_2' &\stackrel{\mathrm{def}}{=} \{wxy\}, \\ l_2' &\stackrel{\mathrm{def}}{=} \{w, x, y, z\} \end{aligned}$$

instead of

$$\begin{aligned} sh_2' &\stackrel{\mathrm{def}}{=} \{wxy, wxyz, xyz\}, \\ l_2' &\stackrel{\mathrm{def}}{=} \{w\}. \end{aligned}$$

To see how a knowledge of the compoundness can be used to identify definite failure, consider the unification $x = f(y, z)$ and the description $d_3 \stackrel{\mathrm{def}}{=} \langle sh_3, f_3, l_3 \rangle \in \mathit{SFL}$ such that

$$\begin{aligned} sh_3 &\stackrel{\mathrm{def}}{=} \{wxy, wxz, x, y, z\}, \\ f_3 &\stackrel{\mathrm{def}}{=} \{w, x\}, \\ l_3 &\stackrel{\mathrm{def}}{=} \{w, x, y, z\}. \end{aligned}$$

As in the examples above, variable x is free and term $t \stackrel{\mathrm{def}}{=} f(y, z)$ is compound so that, by applying the reduction step, we can remove the sharing groups wxy and wxz. However, this has removed all the sharing groups containing the free variable w, resulting in an inconsistent computation state.

We did not implement this technique, since it is only sound for the analysis of systems performing the occurs-check, whereas we are targeting at the analysis of systems possibly omitting it. Nonetheless, an experimental evaluation would be interesting for assessing how much this precision improvement can affect the accuracy of applications such as occurs-check reduction.

12 Conclusion

In this paper we have investigated eight enhanced sharing analysis techniques that, at least in principle, have the potential for improving the precision of the sharing information over and above that obtainable using the classical combination of set-sharing with freeness and linearity information. These techniques either make a better use of the already available sharing information, by defining more powerful abstract semantic operators, or combine this sharing information with that captured by other domains. Our work has been systematic since, to the best of our knowledge, we have considered all the proposals that have appeared in the literature: that is, better exploitation of groundness, freeness, linearity, compoundness, and structural information.

Using the CHINA analyzer, seven of the eight enhancements have been experimentally evaluated. Because of the availability of a very large benchmark suite, including several programs of respectable size, the precision results are as conclusive as possible and provide an almost complete account of what is to be expected when analyzing any real program using these domains.

The results demonstrate that good precision improvements can be obtained with the inclusion of explicit structural information. For the groundness domain *Pos*, several good reasons have been given as to why it should be combined with set-sharing. As for the remaining proposals, it is hard to justify them as far as the precision of the analysis is concerned.

Regarding the efficiency of the analysis, it has been explained why the reported time comparisons can be considered as upper bounds to the additional cost required by the inclusion of each technique. Moreover, it has been argued that, from this point of view, the addition of a 'ground-or-free' mode and the more precise exploitation of linearity are both interesting: they are not likely to affect the cost of the analysis and, when this is the case, they usually give rise to speed-ups.

No further positive indications can be derived from the precision and time comparisons of the remaining techniques. In particular, it has not been possible to identify a good heuristic for the reordering of the non-grounding bindings. The experimentation suggests that sensible precision improvements cannot be expected from this technique. When considering these negative results, the reader should be aware that the precision gains are measured with respect to an analysis tool built on the base domain $Pos \times SFL$ which, to our knowledge, is the most accurate sharing analysis tool ever implemented.

The experimentation reported in this paper resulted in both positive and negative indications. We believe that all of these will provide the right focus in the design and development of useful tools for sharing analysis.

Acknowledgments

This paper is dedicated to all those who take a visible stance in favor of scientific integrity. In particular, it is dedicated to David Goodstein, for *"Conduct and Misconduct in Science"*; to John Koza, for *"A Peer Review of the Peer Reviewing Process of the International Machine Learning Conference"*; to Krzsystof Apt, Veronica Dahl and Catuscia Palamidessi for the Association for Logic Programming's *"Code of Conduct for Referees"*; and to the large number of honest and thorough referees who do so much to help maintain and improve the quality of all publications.

References

ARMSTRONG, T., MARRIOTT, K., SCHACHTE, P., AND SØNDERGAARD, H. 1998. Two classes of Boolean functions for dependency analysis. *Science of Computer Programming 31,* 1, 3–45.

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

Bagnara, R. 1997b. Structural information analysis for CLP languages. In *Proceedings of the "1997 Joint Conference on Declarative Programming (APPIA-GULP-PRODE'97)"*, M. Falaschi, M. Navarro, and A. Policriti, Eds. Grado, Italy, 81–92.

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

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

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. and Schachte, P. 1999. Factorizing equivalent variable pairs in ROBDD-based implementations of *Pos*. In *Proceedings of the "Seventh International Conference on Algebraic Methodology and Software Technology (AMAST'98)"*, A. M. Haeberer, Ed. Lecture Notes in Computer Science, vol. 1548. Springer-Verlag, Berlin, Amazonia, Brazil, 471–485.

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

Blockeel, H., Demoen, B., Janssens, G., Vandencasteele, H., and Van Laer, W. 2000. Two advanced transformations for improving the efficiency of an ILP system. In *Work-in-Progress Reports, Tenth International Conference on Inductive Logic Programming*, J. Cussens and A. Frisch, Eds. London, UK, 43–59.

Bourdoncle, F. 1993a. Efficient chaotic iteration strategies with widenings. In *Proceedings of the International Conference on "Formal Methods in Programming and Their Applications"*, D. Bjørner, M. Broy, and I. V. Pottosin, Eds. Lecture Notes in Computer Science, vol. 735. Springer-Verlag, Berlin, Academgorodok, Novosibirsk, Russia, 128–141.

Bourdoncle, F. 1993b. Sémantiques des langages impératifs d'ordre supérieur et interprétation abstraite. PRL Research Report 22, DEC Paris Research Laboratory.

Bruynooghe, M. and Codish, M. 1993. Freeness, sharing, linearity and correctness — All at once. In *Static Analysis, Proceedings of the Third International Workshop*, P. Cousot, M. Falaschi, G. Filé, and A. Rauzy, Eds. Lecture Notes in Computer Science, vol. 724. Springer-Verlag, Berlin, Padova, Italy, 153–164. 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 *Verification and Analysis of Logic Languages, Proceedings of the W2 Post-Conference Workshop, International Conference on Logic Programming*, F. S. de Boer and M. Gabbrielli, Eds. Santa Margherita Ligure, Italy, 213–230.

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. July.

Bueno, F., de la Banda, M. G., and Hermenegildo, M. V. 1994. Effectiveness of global analysis in strict independence-based automatic program parallelization. In *Logic Programming: Proceedings of the 1994 International Symposium*, M. Bruynooghe, Ed. MIT Press Series in Logic Programming. The MIT Press, Ithaca, NY, USA, 253–268.

Bueno, F., de la Banda, M. G., and Hermenegildo, M. V. 1999. Effectivness of abstract interpretation in automatic parallelization: a case study in logic programming. *ACM Transactions on Programming Languages and Systems 21,* 2, 189–239.

Cabeza, D. and Hermenegildo, M. V. 1994. Extracting non-strict independent and-parallelism using sharing and freeness information. In *Static Analysis: Proceedings of the 1st International Symposium*, B. Le Charlier, Ed. Lecture Notes in Computer Science, vol. 864. Springer-Verlag, Berlin, Namur, Belgium, 297–313.

Chang, J.-H., Despain, A. M., and DeGroot, D. 1985. AND-parallelism of logic programs based on a static data dependency analysis. In *Digest of Papers of COMPCON Spring'85*. IEEE Computer Society Press, San Francisco, California, 218–225.

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

Codish, M., Dams, D., and Yardeni, E. 1991. Derivation and safety of an abstract unification algorithm for groundness and aliasing analysis. See Furukawa (1991), 79–93.

Codish, M., Søndergaard, H., and Stuckey, P. J. 1999. Sharing and groundness dependencies in logic programs. *ACM Transactions on Programming Languages and Systems 21,* 5, 948–976.

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

Cortesi, A., Filé, G., and Winsborough, W. 1992. Comparison of abstract interpretations. In *Proceedings of the 19th International Colloquium on Automata, Languages and Programming (ICALP'92)*, M. Kuich, Ed. Lecture Notes in Computer Science, vol. 623. Springer-Verlag, Berlin, Wien, Austria, 521–532.

Cortesi, A., Le Charlier, B., and Van Hentenryck, P. 1994. Combinations of abstract domains for logic programming. In *Conference Record of POPL'94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.* ACM Press, Portland, Oregon, 227–239.

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

Crnogorac, L., Kelly, A. D., and Søndergaard, H. 1996. A comparison of three occur-check analysers. In *Static Analysis: Proceedings of the 3rd International Symposium*, R. Cousot and D. A. Schmidt, Eds. Lecture Notes in Computer Science, vol. 1145. Springer-Verlag, Berlin, Aachen, Germany, 159–173.

Deransart, P., Ferrand, G., and Téguia:, M. 1991. NSTO programs (Not Subject to Occur-Check). In *Logic Programming: Proceedings of the 1991 International Symposium*, V. A. Saraswat and K. Ueda, Eds. MIT Press Series in Logic Programming. The MIT Press, San Diego, USA, 533–547.

Filé, G. 1994. Share $\times$ Free: Simple and correct. Tech. Rep. 15, Dipartimento di Matematica, Università di Padova. Dec.

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

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

Hermenegildo, M. V. and Greene, K. J. 1990. &-Prolog and its performance: Exploiting independent And-Parallelism. In *Logic Programming: Proceedings of the Seventh International Conference on Logic Programming*, D. H. D. Warren and P. Szeredi, Eds. MIT Press Series in Logic Programming. The MIT Press, Jerusalem, Israel, 253–268.

Hermenegildo, M. V. and Rossi, F. 1995. Strict and non-strict independent and-parallelism in logic programs: Correctness, efficiency, and compile-time conditions. *Journal of Logic Programming 22,* 1, 1–45.

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

ISO/IEC. 1995. *ISO/IEC 13211-1: 1995 Information technology — Programming languages — Prolog — Part 1: General core.* International Standard Organization.

Jacobs, D. and Langen, A. 1989. Accurate and efficient approximation of variable aliasing in logic programs. In *Logic Programming: Proceedings of the North American Conference*, E. L. Lusk and R. A. Overbeek, Eds. MIT Press Series in Logic Programming. The MIT Press, Cleveland, Ohio, USA, 154–165.

Jacobs, D. and Langen, A. 1992. Static analysis of logic programs for independent AND parallelism. *Journal of Logic Programming 13,* 2&3, 291–314.

Janssens, G. and Bruynooghe, M. 1992. Deriving descriptions of possible values of program variables by means of abstract interpretation. *Journal of Logic Programming 13,* 2&3, 205–258.

King, A. 1993. A new twist on linearity. Tech. Rep. CSTR 93-13, Department of Electronics and Computer Science, Southampton University, Southampton, UK.

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

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

Langen, A. 1990. Advanced techniques for approximating variable aliasing in logic programs. Ph.D. thesis, Computer Science Department, University of Southern California. Printed as Report TR 91-05.

Mulkers, A., Simoens, W., Janssens, G., and Bruynooghe, M. 1994. On the practicality of abstract equation systems. Report CW 198, Department of Computer Science, K. U. Leuven, Leuven, Belgium.

Mulkers, A., Simoens, W., Janssens, G., and Bruynooghe, M. 1995. On the practicality of abstract equation systems. In *Logic Programming: Proceedings of the Twelfth International Conference on Logic Programming*, L. Sterling, Ed. MIT Press Series in Logic Programming. The MIT Press, Kanagawa, Japan, 781–795.

Muthukumar, K. and Hermenegildo, M. V. 1991. Combined determination of sharing and freeness of program variables through abstract interpretation. See Furukawa (1991), 49–63. An extended version appeared in (Muthukumar and Hermenegildo 1992).

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

Santos Costa, V., Srinivasan, A., and Camacho, R. 2000. A note on two simple transformations for improving the efficiency of an ILP system. In *Inductive Logic Programming: Proceedings of the 10th International Conference, ILP 2000*, J. Cussens and A. Frisch, Eds. Lecture Notes in Computer Science, vol. 1866. Springer-Verlag, Berlin, London, UK, 397–412.

Scozzari, F. 2000. Abstract domains for sharing analysis by optimal semantics. In *Static Analysis: 7th International Symposium, SAS 2000*, J. Palsberg, Ed. Lecture Notes in Computer Science, vol. 1824. Springer-Verlag, Berlin, Santa Barbara, CA, USA, 397–412.

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

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

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