Enhanced Sharing Analysis Techniques: A Comprehensive Evaluation*

Roberto Bagnara
Department of Mathematics
University of Parma
Parma, Italy
bagnara@cs.unipr.it

Enea Zaffanella
Department of Mathematics
University of Parma
Parma, Italy
zaffanella@cs.unipr.it

Patricia M. Hill
School of Computer Studies
University of Leeds
Leeds, U.K.
hill@scs.leeds.ac.uk

ABSTRACT

Sharing, a domain due to 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 standard integration of Sharing with freeness and linearity information, can significantly improve the precision of Sharing. 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 discuss and/or experimentally evaluate: helping Sharing with definitely ground variables computed with *Pos*; the incorporation of explicit structural information into the domain of analysis; more sophisticated ways of integrating Sharing and *Pos*; the issue of reordering the bindings in the computation of the abstract mgu; an original proposal concerning the addition of a domain recording the set of variables that are deemed to be ground or free; a more refined way of using linearity to improve the analysis; the issue of whether tracking compoundness allows to compute more precise sharing information; and, finally, the recovery of hidden information in the combination of Sharing with the usual domain for freeness.

Categories and Subject Descriptors

F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*Program Analysis*

General Terms

Experimentation, Measurement, Performance

*The work of the first two authors has been partly supported by MURST project "Certificazione automatica di programmi mediante interpretazione astratta". The work of the third author has been partly supported by EPSRC under grant M05645.

Keywords

Mode Analysis, Sharing Analysis, Abstract Interpretation

1. INTRODUCTION

In the execution of a logic program, two variables are *aliased* at some program point if they are bound to terms that share a common variable. In logic programming, a knowledge of the possible aliasing between variables has some important applications.

Information about variable aliasing is essential for the efficient exploitation of AND-parallelism [10, 21, 24, 32]. 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, that is, it will never happen that the processes associated to the two atoms try to bind the same variable.

Another significant application is *occur-check reduction* [18, 33]. It is well-known that many implemented logic programming languages (e.g., almost all Prolog systems) omit the *occur-check* from the unification procedure. Occur-check reduction amounts to identifying the unifications where such 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 on the precision with which aliasing can be tracked [8, 11, 19, 27, 28, 31].

Notice that, sometimes, the property under investigation is called "definite independence". Two variables are *independent* if they are bound to terms that have no variables in common. That is, if two variables are not possibly aliased then they are definitely independent and vice versa. Thus analyzers for possible aliasing and definite independence are effectively equivalent.

Before continuing, a brief note on terminology: a variable is *compound* if it is bound to a non-variable term, it is *ground* if it is bound to a term containing no variables, 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.

In logic programming the expression "sharing information" often refers to a mixture of groundness, aliasing, freeness and linearity information, since groundness, freeness and linearity are properties that allow a more precise characterization of the sharing of program variables. Thus, what is called "a domain for sharing" usually captures groundness, aliasing, and quite often also freeness and linearity. A "sharing analysis" is an analysis based on a sharing domain. Notice that 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".

Sharing, a domain due to D. Jacobs and A. Langen [23, 24, 28], is based on the concept of *sharing-set*. A sharing-set, which is a set of sets of variables, represents, in the context of a set of variables of interest, groundness, groundness dependencies, possible aliasing between variables, and more complex *sharing-dependencies* among the variables that are involved in the execution of a logic program [4].

Even though Sharing is, in a sense, remarkably precise, it is well-known that more precision is attainable by combining Sharing with other domains. In the PhD thesis of Langen [28], it is shown that linearity can greatly improve the accuracy of sharing analysis. The synergy attainable from the integration between aliasing and freeness information was pointed out, for the first time, by K. Muthukumar and M. Hermenegildo [32]. These two enhancements have been combined into a single proposal by W. Hans and S. Winkler in [20] and are now widely accepted (see also [8]), so that nobody would seriously think to perform sharing analysis without them.

In this paper, we investigate eight sharing analysis techniques and/or enhancements which have potential for improving the precision of the sharing information over and above that obtainable using the classical combination of Sharing with the usual domains for freeness and linearity (denoted by *Free* and *Lin*, respectively). These include the combination of Sharing with other domains and using more powerful abstract semantic operators. Although many of these enhancements have been circulating for years, they do not seem to have undergone a thorough experimental evaluation.

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 and, only if this is established, should better implementations be researched. One of the enhancements is the integration of structural information in Sharing and an important contribution of this paper is that it shows both the feasibility and the positive impact of this combination. Note, however, that precise sharing analysis techniques are valuable even if rather inefficient, provided they are feasible. While inefficiency may prevent their adoption in production analyzers, they can help in assessing the precision of the more competitive techniques. Consider a program P and suppose that, at the program points of interest in P, there are p pairs of variables that, in principle, may be aliased. Using a certain sharing analysis technique T we prove that, indeed, n of these p pairs cannot be aliased. Since we know nothing about the $p - n$ remaining pairs, it is difficult to say something on the precision of T. Let us call i the number of independent variable pairs that T could not discover, and a the number of pairs that are aliased in some computation path of P. We have $p - n = i + a$ and the precision of T is high when i is small. A complex analysis technique T' may be useful, regardless of its efficiency, in order to provide a lower bound for i, which is otherwise only limited from above by $p - n$. In other words, if technique T' cannot supersede technique T in production analyzers, it can still be useful for assessing the quality of T.

The experimental part of this work has been conducted with the China analyzer [2]. China is a data-flow analyzer for $\mathrm{CLP}(\mathcal{H}_\mathcal{N})$ languages (i.e., Prolog, $\mathrm{CLP}(\mathcal{R})$, `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.

The comparison involved a test-suite of 160 programs. This includes several real programs of respectable size and is the biggest one ever reported in the literature on data-flow analysis of (constraint) logic programs.

Because of the exponential complexity of Sharing, a practical data-flow analysis using this domain can only be achieved by resorting to widening operators [17, 34]. However, this would affect the precision of the analysis and add unwanted noise to our experimental results. Thus, for an unbiased assessment of the different domain combination enhancements, we disabled all the widenings available to China. The consequence of this is that the analysis of a few programs did not terminate in reasonable time or absorbed memory beyond acceptable limits. In fact, this prevented any comparisons for only 8 of the benchmark programs, and in all experimental evaluations we have compared results on at least 140 programs. Thus, when a program does not appear in a comparison table, this can mean one of two things: one or both the analyses required excessive time or exhausted the available memory and had to be stopped, or both completed but with identical results.

For space reasons, not all the details of the experimental work (such as a description of the constantly growing benchmark suite) could be included here. The interested reader can find much more information on the subject at the URI `http://www.cs.unipr.it/China`.

This paper represents the latest step in our systematic investigation of many diverse issues concerning the Sharing domain. We first examined the adequacy of Sharing with respect to the property of interest, that is, *pair-sharing* [6]. Prior to this work, Sharing had been accepted and implemented as it was, but in [4] we proved that Sharing is redundant for pair-sharing and we identified the weakest abstraction of Sharing that can capture pair-sharing and groundness with the same degree of precision. One notable advantage of this abstraction is that the costly star-union operator is no longer necessary. For this reason, all the experimental work for this paper was conducted using this abstraction instead of the original Sharing domain of Jacobs and Langen [23]. In [22] we have proved the soundness, idempotence, and commutativity of Sharing. Most importantly, these results have been established, for the first time, without assuming that the analyzed language performs the *occur-check* in the unification procedure. This closed a long-standing gap, as all the works on the use of Sharing for the analysis of Prolog programs had always disregarded this problem. The problem of scalability of Sharing, still retaining as much precision as possible, was tackled in [34], where a family of widenings is presented that allows to achieve the desired goal. Finally, in [35] we have studied the decomposition of Sharing and its non-redundant counterpart via complementation. This work has shed new light on the relation between these domains and PS (the usual domain for pair-sharing) and Def (the domain of definite Boolean functions), and on the use of complementation to obtain (minimal) decompositions.

The present paper is structured as follows. In Section 2, we define some notation and briefly recall the definitions associated with Sharing. In each of the next eight sections, we discuss different enhancements and precision optimizations for Sharing. Section 3 considers the combination of Pos with Sharing; Section 4, investigates the effect of including explicit structural information by means of the Pattern($\cdot$) construction; Section 5 discusses possible heuristics for reordering the bindings so as to maximize the precision of Sharing with $Free$ and Lin; Section 6 studies further optimizations with respect to the combination of Sharing and Pos; Section 7 describes a new mode 'ground or free' for propagation with Sharing, $Free$ and Lin, and discusses the precision improvements obtained; Section 8 researches a simple idea for improving the efficiency and precision of Sharing with Lin; Section 9 looks at the question of whether compoundness information would be useful for precision gains; and Section 10 studies the possible exploitation of hidden information available in the Sharing plus $Free$ domain. Section 11 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 $a \Subset M$. $Bind$ denotes the set of equations of the form $x = t$ where $x \in VI$ and $t \neq x$ is a first-order term over VI. Note that we do not impose the *occur-check* condition $x \notin vars(t)$, since we have proved in [22] that this is not required to ensure correctness of the operations of Sharing and its derivatives. The following definitions are a simplification of the standard definitions for the Sharing domain [13, 22, 24] and assume that the set of variables of interest is VI.[1]

Definition 1. **(The *set-sharing* domain SH.)** The set SH is defined by $SH \stackrel{\text{def}}{=} \wp(SG)$, where $SG \stackrel{\text{def}}{=} \wp(VI) \setminus \{\varnothing\}$.

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.)** *Projection* of an element of SH onto a subset of VI is encoded by the binary function proj: $SH \times \wp(VI) \to SH$: if $sh \in SH$ and $V \in \wp(VI)$, then

$$\operatorname{proj}(sh, V) \stackrel{\text{def}}{=} \{\, S \cap V \mid S \in sh, S \cap V \neq \varnothing \,\} \cup \{\, \{x\} \mid x \in VI \setminus V \,\}.$$

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

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

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

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

The domain SH captures *set-sharing*. However, the property we wish to detect is *pair-sharing* and, for this, it has been shown in [6] that SH includes unwanted redundancy. The same paper introduces an upper-closure operator ρ on SH and the domain $PSD \stackrel{\text{def}}{=} \rho(SH)$, which is the weakest abstraction of SH that is *as precise as* SH as far as tracking groundness and pair-sharing is concerned. 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 [6] it is shown that

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

where the notation $sh_1 =_\rho sh_2$ means $\rho(sh_1) = \rho(sh_2)$.

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) has polynomial complexity. Practically speaking, very often this makes the difference between thrashing and termination of the analysis in reasonable time. That is why all the experimental work described in this paper was conducted using the PSD domain. Nonetheless, in order not to confuse the reader unfamiliar with PSD, in the sequel we only refer to the original SH domain and its amgu operator as defined in (1). The results proved in [6] ensure that, provided pair-sharing is the actual property of interest, all that follows applies equally well for the PSD domain, where star-union is systematically replaced by self-bin-union.

Now, in a similar way as done in [20], we briefly recall the standard integration of set-sharing with freeness [32] and linearity [28]. Freeness and linearity are each represented

[1]Note that, during the analysis process, the set of variables of interest is continuously expanded (when solving clause's bodies) and restricted (when abstract descriptions are projected onto the variables occurring in clause's heads). 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.

by a set of variables. Note that these are ordered by *reverse* subset inclusion, so that the lub and glb operators are given by set intersection and union, respectively.

Definition 3. **(The domain** *SHFL***.)** The set *SHFL* is defined by $SHFL \stackrel{\mathrm{def}}{=} SH \times F \times L$, where $F \stackrel{\mathrm{def}}{=} L \stackrel{\mathrm{def}}{=} \wp(VI)$.

The Cartesian product *SHFL* is ordered, as usual, by the component-wise extension of the orderings defined on the three sub-domains. Note that a complete definition, besides explicitly dealing with the set of relevant variables VI, would require the addition of a bottom element $\perp$ representing the semantics of those program fragments having failed computations only. We could also define an equivalence relation identifying this bottom element with all the elements in *SHFL* corresponding to an impossible concrete computation state. E.g., elements $\langle sh, f, l\rangle \in SHFL$ such that $f \not\subseteq vars(sh)$ (because free variables cannot be ground) or $VI \setminus vars(sh) \not\subseteq l$ (because ground variables are also linear). Note however that these spurious elements are never generated during the analysis process.

Definition 4. **(Abstract operations over** *SHFL***.)** *Projection* of an element of *SHFL* onto a subset of VI is encoded by the binary function $\mathrm{proj}\colon SHFL \times \wp(VI) \to SHFL$: if $d = \langle sh, f, l\rangle \in SHFL$ and $V \in \wp(VI)$, then

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

For each $d = \langle sh, f, l\rangle \in SHFL$ and each pair of first order terms s, t having variables in VI, we define the predicates

$$\begin{aligned}
ind_d(s,t) &\stackrel{\mathrm{def}}{=} \Bigl(\mathrm{rel}\bigl(vars(s), sh\bigr) \cap \mathrm{rel}\bigl(vars(t), sh\bigr) = \varnothing\Bigr),\\
free_d(t) &\stackrel{\mathrm{def}}{=} \bigl(\exists x \in VI \,.\, x = t \land x \in f\bigr),\\
lin_d(t) &\stackrel{\mathrm{def}}{=} \bigl(vars(t) \subseteq l\bigr)\\
&\qquad \land \bigl(\forall x, y \in vars(t) : x = y \lor ind_d(x,y)\bigr)\\
&\qquad \land \Bigl(\forall x \in vars(t) : x \Subset mvars(t) \Rightarrow x \notin vars(sh)\Bigr).
\end{aligned}$$

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

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

where

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

Note that, when analyzing languages omitting the occur-check, further improvements of this abstract mgu operator allow to avoid some precision losses when dealing with cyclic bindings. These special cases, already implemented in the CHINA analyzer, have been omitted from the definition above for ease of presentation.

3. COMBINING WITH *POS*

It is well known that Sharing keeps track of ground dependencies. More precisely, Sharing contains *Def*, the domain of definite Boolean functions [1], as a proper subdomain [14, 35]. However, there are several good reasons to couple Sharing with *Pos*:

1. this combination is essential for a powerful widening technique on Sharing to be applied [34]. This is very important, since analysis based on Sharing without a widening is not practical.

2. *Def* is not expressive enough to capture all the ground dependencies of Prolog programs [1]. Moreover, *Def* cannot even capture the dependencies induced by the primitive constraints of some CLP languages, and we target the analysis at Prolog *and* CLP programs.

3. 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 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 Sharing alone either will lose precision on pair-sharing (if numerical variables are allowed to "share" in order to compute their groundness) or will not be able to compute the groundness of numerical variables (if numerical variables are excluded from the sharing-sets). In the first alternative, as we have already noted, the precision with which groundness of numerical variables can be tracked will also be limited. Since groundness of numerical 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 Sharing at the same time.

4. Detecting definitely ground variables through *Pos* and exploiting them to simplify the operations on Sharing is very worthwhile as far as efficiency is concerned if the set of ground variables is readily available. This is the case, for instance, with the GER implementation of *Pos* [7], the fastest *Pos* implementation known to date. This technique alone allows to obtain speedups of up to two orders of magnitude.

5. Knowing the set of ground variables in advance, not only reduces the complexity of Sharing operations. It also improves precision when the domain keeps track of freeness and linearity information by incorporating

Free and *Lin*. 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 [22], Sharing with *Free* and *Lin* does not enjoy this property. In particular, for the combination of Sharing with *Lin* it is known since [28, pp. 66-67] that better results are obtained if the *grounding bindings* are considered before the others.[2] As a simple example, consider the sequences of unifications $\bigl(f(X,X,Y) = A,\ X = a\bigr)$ and $\bigl(X = a,\ f(X,X,Y) = A\bigr)$ [28, p. 66]. Again, the combination with *Pos*, since it allows the analyzer to know the set of all definitely ground variables in advance, is clearly advantageous in this respect.

We have thus compared the combination of Sharing with *Free* and *Lin* in isolation and with the addition of *Pos*. The combination with *Pos* considered here is the simplest one:[3] definitely ground variables are propagated from the *Pos* component to the sharing domain and used (1) to eliminate the sharing groups containing at least one ground variable, and (2) to reorder the bindings so as to handle the grounding ones first. These techniques are systematically applied in all the (combinations of) domains described in this paper.

The results are reported in Table 1. For all the tables in this paper, P is the number of possibly sharing pairs, V is the number of variables, that is, the number of argument positions of the predicates, I is the number of pairs of *independent* variables, G, L, and F, are the number of ground, linear, and free variables, respectively.

While in the case of goal-independent analysis the only improvements are for linearity, for goal dependent analysis better results are also observed for ground variables and independent pairs. Because all of the above reasons, for the remaining comparisons of the different enhancements and precision optimizations, the *Pos* domain is always included unless otherwise stated.

4. TRACKING EXPLICIT STRUCTURAL INFORMATION

A way of increasing the precision of almost any analysis domain is by incrementing it with structural information. This technique was proposed by A. Cortesi et al. in [15], where the generic structural domain $\mathtt{Pat}(\Re)$ was introduced. A similar proposal, tailored to sharing analysis, is due to [8], where *abstract equation systems* are considered. In our experimental evaluation we use the Pattern$(\cdot)$ construction [2, 3, 5], which is similar to $\mathtt{Pat}(\Re)$ and correctly supports the analysis of languages omitting the occur-check in the unification procedure as well as those that do not. The construction 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. Of course, there is a price to be paid: in the analysis based on Pattern$(\mathcal{D})$, the elements of $\mathcal{D}$ that are to be manipulated are often bigger (i.e., they consider more variables) than those that arise in analyses that are simply based on $\mathcal{D}$. There are also many occasions where retaining structural information gives rise to consistent speedups. The reason for this is twofold. On the one hand, structural information has the potential of pruning some computation paths on the grounds that they cannot be followed by the program being analyzed. On the other hand, maintaining a tuple of terms with many variables, each with its own description, can be cheaper than computing a description for the whole tuple [5].

Let us call *Modes*, the combination of *Pos*, Sharing, *Free*, and *Lin*. We have compared the precision for *Modes* with that obtained using Pattern(*Modes*). The results are reported in Tables 2 and 3.

The approach we have chosen for measuring the precision gain is simple though unsatisfactory: throw away all the structural information at the end of the analysis and compare the number of independent pairs, as well as the numbers of ground, linear and free variables. However, the analysis with Pattern(*Modes*) yields much more information. Consider a simple but not trivial Prolog program: `mastermind`.[4] 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 *Modes* domain cannot say anything on the successes of `extend_code/1`.

If we perform the analysis with Pattern(*Modes*) the situation changes radically. Here is what such a domain allows CHINA to derive:[5]

```
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(*Modes*) only specifies that `extend_code/1` may succeed. Thus, our approach to the comparison pretends that explicit structural information gives no improvements in the analysis of `extend_code/1` in `mastermind`. In other words, we have only measured how the explicit structural information present in Pattern(*Modes*) improves the precision on *Modes* itself, which is only a tiny part of the real gain in accuracy. Of course, structural information is very valuable in itself. When exploited for optimized compilation it allows for enhanced clause indexing and simplified unification. Moreover, several program verification techniques are highly dependent on this kind of information. However, the

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

[3] More precise combinations will be considered in Section 6.

[4] A program implementing the game "Mastermind", rewritten by H. Koenig and T. Hoppe after code by M. H. van Emden. Available at `http://www.cs.unipr.it/China/Benchmarks/Prolog/mastermind.pl`.

[5] 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.

Goal-dependent analysis			Without *Pos* / With *Pos*			
Program	P	V	I	G	L	F
chat_parser	4070	1484	3321/3332	505/505	906/908	357/357
dpos_an	324	366	187/188	78/79	131/132	44/44
knight	117	92	102/103	44/45	62/63	16/16
sim_v5-2	459	535	456/457	415/417	535/535	106/106
tsp	502	220	488/488	122/122	206/220	38/38

Goal-independent analysis			Without *Pos* / With *Pos*			
Program	P	V	I	G	L	F
bmtp	3091	1681	1759/1759	146/146	1148/1151	295/295
bp0-6	264	115	215/215	31/31	88/90	21/21
bryant	1252	330	1112/1112	32/32	124/210	4/4
cg_parser	257	274	138/138	31/31	192/193	58/58
km-all	28898	14046	18379/18379	1929/1929	9887/9900	2943/2943
knight	58	45	37/37	14/14	37/38	3/3
oldchina	3584	2178	2266/2266	309/309	1451/1457	281/281
sax	3284	1993	1697/1697	269/269	970/974	202/202
tsp	251	110	219/219	26/26	95/98	19/19

Table 1: The effect of integrating *Pos*.

			Without s. i. / With s. i.			
Program	P	V	I	G	L	F
action	160	180	15/17	4/5	10/11	6/6
ann	832	479	563/575	110/117	192/202	78/91
astar	59	66	56/57	50/51	63/66	11/12
chasen	158	185	71/72	55/58	89/92	33/35
dpos_an	324	366	188/212	79/105	132/170	44/50
eliza	224	208	115/116	69/71	106/109	33/36
ftfsg2	223	238	144/156	45/57	105/117	42/42
ftfsg	134	122	91/95	22/24	60/62	26/26
grammar	32	34	28/28	7/9	34/34	16/16
jugs	71	68	33/34	8/8	22/24	13/13
knight	117	92	103/110	45/45	63/92	16/17
lc	106	112	32/105	11/91	28/112	17/18
ljt	513	285	513/513	270/270	285/285	13/17
llprover	616	670	434/435	180/190	284/294	101/111
log_interp	455	477	178/182	43/43	118/118	72/72
loops	66	86	63/63	66/66	82/86	13/15
nbody	600	347	478/478	155/155	196/200	40/40
parser	436	365	336/344	60/60	278/365	202/202
press	294	266	174/178	44/44	77/79	30/30
quot_an	1132	817	639/664	144/167	266/289	95/95
read	437	281	359/359	118/118	198/201	64/64
reg	334	387	208/208	69/78	121/130	49/49
sdda	195	172	69/79	24/28	49/54	25/25
sim_v5-2	459	535	457/457	417/417	535/535	106/111
tictactoe	274	130	270/270	88/88	104/108	11/13
tsp	502	220	488/489	122/126	220/220	38/38
yasmm	78	60	49/51	21/21	27/41	6/6

Table 2: The effect of explicit structural information: goal-dependent analysis.

			Without s. i. / With s. i.			
Program	P	V	I	G	L	F
8puzzle	20	18	10/10	7/7	14/16	2/2
action	80	90	36/38	1/2	48/49	22/22
aircraft	391	588	344/346	208/234	570/582	58/58
ann	416	239	216/225	15/18	129/132	41/45
arch1	437	285	167/182	9/9	113/120	33/33
bmtp	3091	1681	1759/1761	146/148	1151/1152	295/297
bup-all	177	168	68/70	16/16	95/99	34/34
cg_parser	257	274	138/138	31/31	193/194	58/58
chat80	3722	1646	2628/2660	342/342	1296/1303	308/308
cobweb	782	361	444/482	30/33	115/130	33/35
cs2	166	94	115/119	31/35	66/71	4/4
cugini_ut	372	407	153/166	71/74	219/226	40/40
difflists	33	40	16/16	8/9	22/32	2/2
dpos_an	164	192	98/100	42/45	118/120	21/23
files	98	131	57/59	59/59	112/113	20/20
ftfsg2	384	404	255/260	94/95	282/286	73/73
ftfsg	254	263	124/129	17/18	149/153	57/57
ga	433	147	364/366	55/60	123/128	15/15
grammar	16	17	11/11	4/5	17/17	4/4
km-all	28898	14046	18379/18535	1929/1986	9900/9979	2943/2962
knight	58	45	37/43	14/14	38/44	3/3
ljt	256	140	29/35	5/9	42/46	19/23
llprover	307	333	191/191	84/86	256/253	24/24
log_interp	261	254	66/88	14/14	97/98	28/28
metutor	534	494	324/326	138/139	326/327	43/47
mixtus-all	3874	2186	2344/2364	161/163	1308/1319	419/423
nbody	300	173	265/265	64/64	160/164	8/8
oldchina	3584	2178	2266/2284	309/309	1457/1462	281/281
parser	218	182	117/117	28/28	148/182	61/61
petsan	3838	1461	2603/2603	278/281	934/934	217/219
plaiclp	2453	1296	1760/1760	158/165	947/957	230/240
quot_an	563	400	289/292	38/41	199/203	46/46
reg	1600	693	814/1212	49/61	336/419	63/63
sax	3284	1993	1697/1772	269/327	974/1034	202/232
sdda	96	80	27/27	4/5	31/35	13/13
sim_v5-2	242	281	104/104	53/54	190/190	25/25
sim	1502	459	911/1100	76/80	254/273	25/25
slice-all	833	800	438/444	135/137	582/619	119/119
spsys	1582	1093	805/988	88/123	483/551	103/104
tictactoe	101	56	93/93	13/13	46/54	4/5
trees1	71	62	50/62	29/40	50/61	4/12
trs	109	73	53/56	6/8	28/34	4/4

Table 3: The effect of explicit structural information: goal-independent analysis.

value of this extra precision can only be measured from the point of view of the target application of the analysis.

Tables 2 and 3 indicate that enhancing Sharing with structural information can make useful improvements to precision. Moreover, occasionally (such as for `lc` in Table 2), this improvement is considerable. This demonstrates that there is a relevant amount of sharing information that is not detected when using the classical set-sharing domains. Therefore, we conducted the remaining experiments both with and without explicit structural information. A similar experimental evaluation, but based on the abstract equation systems of [8], was reported by Mulkers et al. in [29, 30]. Here a depth-k abstraction (replacing all subterms occurring at a depth greater 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 to the analysis of real programs and, in 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).

Another previous attempt to evaluate the impact of structural information on sharing failed because of combinatorial explosion of the analysis [A. Cortesi, personal communication, 1996]. What makes a realistic experimentation now possible is the adoption of the non-redundant domain *PSD*, where the exponential star-union operation is replaced by quadratic self-bin-union, and the integration of this domain with the GER implementation of *Pos* [7]. Indeed, as demonstrated by the results reported in [5], an analyzer that incorporates a carefully designed structural information component, besides being more precise, can also be very efficient.

5. REORDERING THE NON-GROUNDING BINDINGS

In Section 3, we have already explained why we always consider the grounding bindings first. All the examples we could find in the literature use a grounding binding to show that abstract unification on the combination of Sharing with *Free* and *Lin* is non-commutative. However, the problem is more general than that.

As an example, consider $\{u, v, w, x, y, z\}$ as the set of relevant variables, and the *SHFL* element[6]

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

where u, x, and z are linear variables. We now apply the bindings $v = w$ and $x = y$. Using the binding $v = w$ first, we have:

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

Using the binding $x = y$ first, we have:

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

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

The *brute-force* approach that tries all the orderings of the non-grounding bindings is clearly not feasible. Of course, it would be highly desirable to find a heuristic based on the *local search* paradigm: at each step, the next binding for the amgu procedure should be chosen by evaluating just the effect of its abstract execution, considered in isolation, on the precision of the analysis.

One such heuristic would be to consider saying "delay the bindings requiring star-unions". The logic underlying this is that it is likely to obtain intermediate abstract descriptions having less sharing groups. This, in turn, should reduce precision losses in the freeness and linearity components. 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:

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

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

Since x is linear and it does not share with z, computing $\operatorname{amgu}(d, x = z)$ requires a single star-union, while both star-unions are needed for $\operatorname{amgu}(d, v = w)$ because v and w may share. Thus, using the proposed heuristic, we apply $x = z$ first and obtain:

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

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

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

It should be noted that this heuristic, considered in isolation, is not a general solution to the problem. Indeed in our first example the star-unions have to be computed in both cases. Moreover, a further example shows that such a heuristic can even cause a precision loss. Consider the bindings $u = x$ and $v = w$ and the abstract description

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

Since x and u are free variables, star-unions are not needed for computing $\operatorname{amgu}(d, u = x)$, while they are needed for $\operatorname{amgu}(d, v = w)$.

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

Using the other ordering we have:

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

Note that in $d_{2,1}$ variables y and z are independent, whereas they may share in $d_{1,2}$.

Another possibility would be to consider a heuristic that, maybe disregarding the number of star-unions required, directly tries to improve the precision on the freeness and linearity components by choosing first those bindings that maximally preserve freeness and linearity information. However, the last example shown above witnesses that in some cases even such a proposal causes precision losses (the binding $u = x$ would be preferred and chosen first as it preserves the freeness of variable u).

Even if the behavior of other heuristics could be investigated, up to now all our attempts to find a *good* heuristic — one that rarely decreases precision — have failed. In conclusion, on this subject there seems to be room for more research.

6. MORE PRECISE COMBINATIONS WITH *POS*

Since there is an overlap between the information provided by *Pos* and the information provided by Sharing, it is clear that the Cartesian product $Pos \times$ Sharing contains redundant elements, i.e., different pairs that characterize the same set of concrete computational states. The reduced product [16] between *Pos* and Sharing has been elegantly characterized in [12], where set-sharing *à la* Jacobs and Langen is expressed in *Pos* itself. Without aiming at the full power of the reduced product we illustrate here two techniques (besides the propagation of definitely ground variables) in which the information contained in the *Pos* component can be used to improve the description provided by the Sharing component.

Suppose the *Pos* component (a Boolean formula) implies a *binary* disjunction $x \vee y$. This means that either x is ground or y is ground or both are so. In any case, x and y cannot share a common variable. It is consequently safe to remove from the Sharing component all the sharing groups containing both x and y.

Now suppose the *Pos* component implies $\bigwedge_{x,y \in X}(x \leftrightarrow y)$ for some set of variables X. In this case, the groundness of any variable in X implies the groundness of all the variables in X. Since all the variables in X share the same (possibly empty) set of variables, *after the same set of abstract bindings has been performed on both the Pos and* Sharing *components*, we can remove from the Sharing component each sharing group S such that $S \cap X \neq \varnothing$ and $X \nsubseteq S$.

We implemented both these techniques together. The few improvements we observed in the experiments concerned only the number of independent variable pairs, even though, in theory, also the accuracy of both freeness and linearity can be affected. In fact, only three programs showed any changes to the number of such pairs. For goal-independent analysis, the `bmtp` program had one more pair when analyzed with and without explicit structural information, and the `sim` program had two more pairs when analyzed without explicit structural information. For goal-dependent analysis, the program `knight` had one more pair only when analyzed without explicit structural information.

7. 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 that is new to this paper. 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 what will be recorded in the usual combination of Sharing with *Free* and *Lin*. This information is valuable, since, in case x and t are independent, it allows to dispense with the star-union of the relevant component for t. However, the information that is lost, i.e., x being 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, independently of whether or not x and t may share. This loss has two disadvantages: CPU time is wasted by performing a costly operation and precision is degraded. In fact, in these cases the extra star-unions are useless to ensure correctness and, moreover, they may introduce unneeded sharing groups to the detriment of accuracy.

It is therefore natural to extend the analysis domain with another component $gf \in \wp(VI)$ consisting of the set of variables that are ground or free, thus adding an additional mode to the picture. As for freeness and linearity, the approximation ordering is given by reverse subset inclusion. The 'ground or free' property is then propagated in the abstract mgu operation almost the same way as freeness:

$$gf' = \big(VI \setminus vars(sh')\big) \cup \big(gf \setminus (f \setminus f')\big).$$

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. In synthesis, the incorporation of the set of 'ground or free' variables can be done cheaply, both in terms of computational complexity and in terms of code to be written. As far as computational complexity is concerned this extension is particularly promising, since the possibility of avoiding star-unions has the potential of absorbing its overhead if not of giving rise to a speedup.

We have thus implemented the combination of *Pos* (in order to maximize the number of definitely ground variables detected), Sharing, *Free*, *Lin*, and 'ground or free' variables and tried it on our benchmark suite. The experimentation has been performed both with and without added structural information, and both in a goal-dependent and goal-independent way. The only difference we have observed is for the goal-independent analysis of `knight` without structural information. In this case, the 'ground or free' extension is worth 6 more definitely independent pairs of variables and 6 more variables detected as definitely linear.

8. MORE PRECISE EXPLOITATION OF LINEARITY

In [26] 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 [28], which is the one usually followed, a set of definitely linear variables is recorded along with each sharing-set. The proposal in [26] gives rise to a domain that is quite different from the ones presented here. Since [26] 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 [26] can be usefully applied even in the more classical treatment of linearity presented in [28]. As far as we can tell, this fact has gone unnoticed up to now.

In [26], point 3 of Lemma 5 (which is reported to be proven in [25]) states formally that, if s is a linear term and t is a (possibly) non-linear term, then, after computing the unification $s = t$, a variable occurring only once in t can only be aliased to one variable in s. This result can be exploited even when using the standard domain for linearity-enhanced sharing analysis.

Let x be a linear variable and t be a non-linear term. Let $V_x = \{x\}$, $V_t = vars(t)$ and $V_{xt} = V_x \cup V_t$. Let V_t^{l} be the set of variables that can occur only once in term t. These are exactly the variables $y \in V_t$ such that: (1) y is linear, (2) y occurs once in t, and (3) y does not share with other variables in t. Let $V_t^{\mathrm{nl}} = V_t \setminus V_t^{\mathrm{l}}$. Note that $V_t^{\mathrm{nl}} \neq \varnothing$, because t is a non-linear term. If also $V_t^{\mathrm{l}} \neq \varnothing$ (and, obviously, if x

and t do not share) then we can use an improved version of the amgu operator computing the following set-sharing component:

$$sh' = \overline{\text{rel}}(V_{xt}, sh) \cup \text{bin}\big(\text{rel}(V_x, sh), \text{rel}(V_t^{\text{l}}, sh)\big) \\ \cup \text{bin}\big(\text{rel}(V_x, sh)^\star, \text{rel}(V_t^{\text{nl}}, sh)\big).$$

Note that precision can be improved because, thanks to the Lemma, the star-union of the relevant component of sh with respect to V_x is only combined with the relevant component of sh with respect to the non-linear variables in V_t^{nl}. However, the only programs for which we observed an improvement in the accuracy of the analysis were the synthetic benchmarks we wrote in order to show that a precision gain is indeed possible. Despite its apparently limited practical relevance, this result has an important theoretical consequence in that it demonstrates that the standard combination of Sharing with *Lin* (even when all orderings of the non-grounding bindings are tried) is *not* optimal. Also note that the scope of this observation is not limited to the domain defined in [28], since the same enhancement can be equally applied to any stronger abstract domain (e.g., those including also freeness [8, 20]).

9. TRACKING COMPOUNDNESS

In [8, 9], M. Bruynooghe et al. considered the combination of the standard set-sharing, freeness, and linearity domains with compoundness information. As for freeness and linearity, compoundness was represented by the set of variables that definitely have the respective property.

As discussed in [8, 9], compoundness information is useful in its own right for clause indexing. We are interested here though in the question: can the tracking of compoundness improve the sharing analysis itself? This question is also considered in [8, 9] where two techniques are proposed that exploit the combination of freeness and compoundness.

The first one relies on the presence of the occur-check (and thus it cannot be applied when analyzing systems that omit it). Informally, if x is free and t is compound then, just before computing the binding $x = t$, we can safely say that x cannot share with any variable of t. Thus we can improve our sharing description by removing all the sharing groups containing both x and a variable in t. In particular, if in this case there also exists another free variable definitely sharing with both x and t, then the computation is guaranteed to fail. We decided that the implementation effort needed to evaluate this idea was not warranted as the technique is only sound for the analysis of systems performing the occur-check. Moreover, only one program in our benchmark suite was written for such a system.[7]

The second idea is proposed inside the specification of the function *Reduce*, which is intended to remove from an abstract description some spurious information that is redundant. In particular, it is shown that compoundness can be safely inferred for all the variables that definitely share with a pair of independent free variables.[8] It should be noted that such a *Reduce* function is not part of the abstract unification algorithm presented in [8] and it is unclear where and when it should be applied. The authors suggest that the algorithm should start with reduced abstract descriptions. However, there is no proof (or even claim) that the algorithm actually preserves this reduction property. Their only assertion is that precision losses may be avoided if the algorithm starts with a reduced description. It is our opinion that the second improvement proposed in [9] may well never apply because

1. the initial descriptions, computed by the abstraction function, are always reduced;
2. it seems very unlikely that a well-designed compoundness analysis can lose this kind of information, which stems from bindings of the form $x = f(y, z)$. In particular, this holds when tracking compoundness information by using another instance of the domain *Pos*, as currently done in the CHINA analyzer.

For these reasons, and also because we were interested in helping Sharing with compoundness and not the other way round, we did not implement this idea.

10. SHARING AND FREENESS

As noted by several authors (see, e.g., [8]) the standard combination of Sharing and *Free* is not optimal. G. Filé [19] 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, we can use property 1 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 occur-check. Note that this is possible even without decomposing the abstract description: for example, consider the substitutions $\sigma_1 = \{x = f(u), y = g(v)\}$ or $\sigma_2 = \{x = f(y)\}$ together with an abstract description saying that x and y are both free and the only sharing group allowed is $\{xy\}$.

The experimental results we obtained for the first of the two ideas by Filé presented above do not depend on whether or not structural information or *Pos* is included in the domain (we have tried all possible combinations). While no difference was observed for goal-dependent analysis, the goal-independent analyses of `petsan` and `cobweb` gave rise to 3

[7] This is `ileanTAP`, an intuitionistic theorem prover written by J. Otten. The program begins with the directive `:- set_flag(occur_check,on).`

[8] The function *Reduce* also deals with some hidden interactions between sharing and freeness information: as these improvements are subsumed by the work of G. Filé [19], we discuss them in Section 10.

and 2 more pairs of independent variables exposed, respectively. It must be observed that the analysis of several programs had to be stopped because of the combinatorial explosion in the decomposition. Indeed, among the proposals experimentally evaluated in this paper, this one is the most expensive in computational terms.

We note in passing that such an approach to the recovery of disjunctive information can be pursued beyond the integration of sharing with freeness. Indeed, by exploiting 'ground or free' information as in Section 7, 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.

It would be interesting to experiment with the second idea of Filé (exploitation of the *concrete* structural information contained in a sharing description with freeness). At present, the problem is how to incorporate this into the China analyzer without destroying its modular design.[9]

11. 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 Sharing with *Free* and *Lin*. To do this, we have considered including other domains and using more powerful abstract semantic operators. Our work has been systematic since, to the best of our knowledge, we have evaluated all the techniques that have been proposed in the literature regarding how the usual domains of analysis may contribute to Sharing; that is, better exploitation of linearity, freeness, compoundness, groundness, and structural information.

We have experimentally evaluated, using the China analyzer, seven of the eight enhancements. We have demonstrated the impact of structural information on sharing analysis on the analysis of real programs, showing that good improvements can be obtained with the inclusion of explicit structural information. We have given several reasons why *Pos* should be combined with Sharing. It is hard to justify the remaining proposals as far as precision is concerned. For the addition of a 'ground or free' mode and for the more precise exploitation of linearity, at least the computational cost is negligible. Unfortunately, this is no longer true for the enhanced combination with *Pos* and for the exploitation of the interaction of Sharing and *Free*. When considering these negative results, the reader should be aware that for such experimental evaluations we measured precision gains with respect to a base domain (the combination of Sharing plus *Free* plus *Lin* plus *Pos* plus the anticipation of the grounding bindings) that to our knowledge is the most accurate sharing analysis tool ever implemented.

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

[9] Roughly speaking, the generic structural information domain Pattern(·) (a C++ template at the implementation level) would have to be heavily modified so as to be able to receive notifications from its parameter.

12. REFERENCES

[1] T. Armstrong, K. Marriott, P. Schachte, and H. Søndergaard. Two classes of Boolean functions for dependency analysis. *Science of Computer Programming*, 31(1):3–45, 1998.

[2] R. Bagnara. *Data-Flow Analysis for Constraint Logic-Based Languages*. PhD thesis, Dipartimento di Informatica, Università di Pisa, Italy, 1997. Printed as Report TD-1/97.

[3] R. Bagnara. Structural information analysis for CLP languages. In M. Falaschi, M. Navarro, and A. Policriti, editors, *Proceedings of the "1997 Joint Conference on Declarative Programming"*, pages 81–92, Grado, Italy, 1997.

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

[5] R. Bagnara, P. M. Hill, and E. Zaffanella. Efficient structural information analysis for real CLP languages. Quaderno 229, Dipartimento di Matematica, Università di Parma, 2000. Available at `http://www.cs.unipr.it/~bagnara`.

[6] R. Bagnara, P. M. Hill, and E. Zaffanella. Set-sharing is redundant for pair-sharing. *Theoretical Computer Science*, 2000. To appear.

[7] R. Bagnara and P. Schachte. Factorizing equivalent variable pairs in ROBDD-based implementations of *Pos*. In A. M. Haeberer, editor, *Proceedings of the 7-th International Conference on Algebraic Methodology and Software Technology*, volume 1548 of *Lecture Notes in Computer Science*, pages 471–485, Amazonia, Brazil, 1999. Springer-Verlag, Berlin.

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

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

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

[11] M. Codish, D. Dams, G. Filé, and M. Bruynooghe. Freeness analysis for logic programs — and correctness? In D. S. Warren, editor, *Logic Programming: Proceedings of the 10-th International Conference on Logic Programming*, pages 116–131, Budapest, Hungary, 1993. The MIT Press. An extended version is available as Technical Report CW 161, Department of Computer Science, K.U. Leuven, Belgium, 1992.

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

[13] A. Cortesi and G. Filé. Sharing is optimal. *Journal of Logic Programming*, 38(3):371–386, 1999.

[14] A. Cortesi, G. Filé, and W. Winsborough. Comparison of abstract interpretations. In M. Kuich, editor, *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*, volume 623 of *Lecture Notes in Computer Science*, pages 521–532, Wien, Austria, 1992. Springer-Verlag, Berlin.

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

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

[17] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In M. Bruynooghe and M. Wirsing, editors, *Proceedings of the 4th International Symposium on Programming Language Implementation and Logic Programming*, volume 631 of *Lecture Notes in Computer Science*, pages 269–295, Leuven, Belgium, 1992. Springer-Verlag, Berlin.

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

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

[20] W. Hans and S. Winkler. Aliasing and groundness analysis of logic programs through abstract interpretation and its safety. Technical Report 92–27, Technical University of Aachen, 1992.

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

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

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

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

[25] A. King. A new twist on linearity. Technical Report CSTR 93-13, Department of Electronics and Computer Science, Southampton University, Southampton, UK, 1993.

[26] A. King. A synergistic analysis for sharing and groundness which traces linearity. In D. Sannella, editor, *Proceedings of the 5-th European Symposium on Programming*, volume 788 of *Lecture Notes in Computer Science*, pages 363–378, Edinburgh, UK, 1994. Springer-Verlag, Berlin.

[27] A. King and P. Soper. Depth-k sharing and freeness. In P. Van Hentenryck, editor, *Logic Programming: Proceedings of the 11-th International Conference on Logic Programming*, pages 553–568, Santa Margherita Ligure, Italy, 1994. The MIT Press.

[28] A. Langen. *Advanced Techniques for Approximating Variable Aliasing in Logic Programs*. PhD thesis, Computer Science Department, University of Southern California, 1990. Printed as Report TR 91-05.

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

[30] A. Mulkers, W. Simoens, G. Janssens, and M. Bruynooghe. On the practicality of abstract equation systems. In L. Sterling, editor, *Logic Programming: Proceedings of the 12-th International Conference on Logic Programming*, pages 781–795, Kanagawa, Japan, 1995. The MIT Press.

[31] K. Muthukumar and M. Hermenegildo. Combined determination of sharing and freeness of program variables through abstract interpretation. In K. Furukawa, editor, *Logic Programming: Proceedings of the 8-th International Conference on Logic Programming*, pages 49–63, Paris, France, 1991. The MIT Press. An extended version appeared in [32].

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

[33] H. Søndergaard. An application of abstract interpretation of logic programs: Occur check reduction. In *Proceedings of the 1986 European Symposium on Programming*, volume 213 of *Lecture Notes in Computer Science*, pages 327–338. Springer-Verlag, Berlin, 1986.

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

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