Widening Sharing

Enea Zaffanella[1], Roberto Bagnara[2], and Patricia M. Hill[3]*

[1] Servizio IX Automazione, Università degli Studi di Modena, Italy.
zaffanella.enea@unimo.it
[2] Dipartimento di Matematica, Università degli Studi di Parma, Italy.
bagnara@cs.unipr.it
[3] School of Computer Studies, University of Leeds, Leeds, LS2 9JT, U. K.
hill@scs.leeds.ac.uk

Abstract. We study the problem of an efficient *and* precise sharing analysis of (constraint) logic programs. After recognizing that neither plain Sharing nor its non-redundant (but equivalent) abstraction scale well to real programs, we consider the domain proposed by C. Fecht [12, 13]. This domain consists of a combination of *Pos* with a quite weak abstraction of Sharing. While verifying that this domain is truly remarkable, in terms of both precision and efficiency, we have revealed significant precision losses for several real programs. This loss concerns groundness, pair-sharing, linearity, but not freeness. (Indeed, we have proved that a wide family of abstractions of Sharing do not incur precision loss on freeness.) We define a simple domain for sharing analysis that supports the implementation of several widening techniques. In particular, with this domain it is straightforward to turn Fecht's idea into a proper widening. More precise widenings are also considered. However, in spite of thorough experimentation we found that the first widening we propose is hard to improve on, provided *Pos* is included in the domain. We show that when *Pos* is not included, a widening based on cliques of sharing pairs is preferred.
Keywords: Mode Analysis, Sharing Analysis, Widening.

1 Introduction

For (constraint) logic programs, the main purpose of sharing analysis is to detect pair-sharing; that is, which *pairs* of variables are definitely independent. In a previous work [3] we observed that the Sharing domain of Jacobs and Langen [15] is redundant for pair-sharing. This achievement has important theoretical consequences (some of which will be exploited in the present work) and also a practical interest. In fact, it allows to keep sharing-sets as small as possible without any precision loss and to replace the *star-union* operation, whose complexity is exponential, by *self-bin-union*, which is quadratic. Even though significant speed-ups have been observed in practice (up to three orders of magnitude on the analysis of real programs), the problem of scalability of the analysis, both in terms

* This research was partly supported by EPSRC, grant GR/M05645.

of precision (that is, the number of pairs that are detected as being definitely independent) and of resource usage, was still to be solved.

In this paper we address this problem. However, in order to give the right focus to the present work, we need to explain in detail what we aim at.

1.1 Analyses and Analyzers

The experimental part of our work is devoted to the construction of *practical*, *precise* and *efficient* data-flow analyzers for constraint logic-based languages. Some issues connected with the emphasized words deserve clarification.

A "practical analyzer" is one that has a chance to be turned into a useful tool. On one hand this means that compromising assumptions about the languages and the programs to be analyzed must be avoided as far as possible. Researchers in our area (including the present authors) have often made assumptions that are falsified by the implemented languages and their programs. This state of affairs can be justified in a relatively immature field, but this is no longer the case for the data-flow analysis of logic programs. Therefore we believe that we should now rid ourselves of most, if not all, limiting assumptions. We must take into account, for instance, that implemented languages perform unification that omits the *occur-check*; that programmers do exploit "nasty constructs" such as `assert/1` and `call/1`; that real programs make use of all kinds of built-ins provided by the language; as well as libraries, foreign language interfaces etc.

Many applications of data-flow analysis, such as semantics-based programming environments, need very precise information about a program's behavior in order, say, to assist the programmer during development, debugging, and certification. In the literature there are several papers reporting on the experimental evaluation of data-flow analyzers. In some of them one can find analysis' times well under the second for non-trivial, lengthy programs. What can one conclude from the fact that a program of several thousands lines can be analyzed in a couple of seconds on a desktop computer? If one excludes the possibility outlined above that special assumptions have been exploited so that the results cannot be generalized, the answer is probably that more precision is attainable. One of the important applications of data-flow analysis is in computer-assisted program verification or certification. In this field, what is not done by the computer must be done by hand. Who will spend hours to complete proofs by hand when the computer can do them in the same or even double the time? Similar remarks hold also for optimized compilation, if one takes into account that (1) only *production versions* deserve to be compiled with the optimization passes turned on, (2) a production version is compiled once and used thousands, perhaps millions of times, and (3) computers do work overnight.

So we do not participate in the race for the fastest ever analysis, especially when done (as is often the case) at the expense of precision. The real problem is how to increase precision yet avoid the concrete effects of exponential complexity. Consider groundness analysis, for instance. The cruder domains do not pose any efficiency problem. In contrast, the more refined domains for groundness, such as *Pos*, work perfectly until you bump into a "nasty" program clause (i.e., with more than, say, fifty variables for which the analyzer knows too little at that point of the analysis). When this happens, *Pos* will exhaust your computer's memory. One would like to have a more *linear*, or *stable* behavior. The right solution, as indicated by Cousot and Cousot [11], is not to revert to the simpler domains. We should use instead complex domains together with widening/narrowing operators. With such techniques we can try to limit precision losses to those cases where the cost of the complexity implied by these refined domains exceeds the available resources.

Ideally, it should be possible to endow data-flow analyzers with a *knob*. The user could then "rotate the knob" in order to control the complexity/precision ratio of the system. The widening/narrowing approach can make this possibility a reality. Unfortunately, the design of widening operators tends somewhat to escape the realm of theoretical analysis, and thus, in the authors' opinion, it has not been studied enough. Indeed, the development of successful widening operators requires, perhaps more than other things, extensive experimentation.

1.2 Fecht's Work

C. Fecht [12, 13] proposed a domain $\downarrow SH$ for sharing analysis based on an abstraction of the usual Jacobs and Langen domain SH [15]. This domain is the same as SH but the concretization of a set of variables in $\downarrow SH$ is equivalent to the concretization of its powerset in SH. The advantage of $\downarrow SH$ is not just that an element can be normalized by removing all but the maximal sets, thereby reducing its size, but because it enables more efficient (but less precise) abstract operations than those used for SH and its non-redundant version SH^{ρ} [3]. Moreover, for computing the abstract unification in $\downarrow SH$, Fecht describes two useful optimizations that improve efficiency without losing any further precision.

One of the problems with the domain $\downarrow SH$ is that it does not capture ground dependencies. These are important for tracking sharing dependencies and, hence, sharing. Fecht solved this by deriving the ground dependencies through the *Pos* component of the combined domain $Pos + \downarrow SH$ and also $Pos + \downarrow SH + Lin$. Fecht tested both these domains and showed that, with his benchmarks, they compared favorably with equivalent ones using SH for the sharing and ground dependencies. He reported a negligible loss of precision and demonstrated that large programs could be analyzed using both $Pos + \downarrow SH$ and $Pos + \downarrow SH + Lin$ in a reasonable time scale. The results, although inconclusive, demonstrated real promise for an analyzer based on the $\downarrow SH$ approach. We say the results were inconclusive. The reason for this is that only a few non-trivial programs were tested and, for most of these, precision was not compared. (Fecht's SH analyzer could not cope with large programs possibly due to the problem that there was no redundancy elimination.) We note that Fecht did not present the domain $\downarrow SH$ as a widening[1] and did not discuss how a widening based on his domain might be achieved.

[1] Indeed the approach of Fecht falls under the category "use a simpler domain" which, as clearly explained in [11], is both contrary and inferior to the approach "use a complex domain with widening" that is advocated in this paper.

1.3 Combining Domains

In Fecht's work, and also in the work presented here, the combination of a sharing domain with *Pos* is the simplest possible. For any operation of the analysis, abstract mgu in particular, the *Pos* component is evaluated first. All sharing groups containing at least one variable that is definitely ground according to the resulting *Pos* formula are removed from the sharing component. This combination is made particularly efficient by the ready availability of definite groundness information allowed by the GER representation introduced in [5], where obtaining the set of definitely ground variables (and also the classes of groundness-equivalent variables) is a constant-time operation. Note that, theoretically speaking, more sophisticated combinations of *Pos* with $\mathsf{Sharing}$ are possible [8].

Following several other authors, we observed in [3], that, from a practical point of view, sharing analysis without freeness or linearity does not make sense. Both these properties allow, in a significant proportion of cases, to dispense with costly operations (such as star-union or, better, self-bin-union [4]) increasing the precision of sharing information at the same time, and this with very little overhead. Moreover, freeness is a useful property in itself. For details on how the combination with freeness is realized, we refer the reader to [17, 19]. See [7] for the combination of both freeness and linearity information.

1.4 Experimental Results

We have compared the domain of Fecht *enhanced with freeness information*, that is $Pos{+}{\downarrow}SH+Free{+}Lin$, with the same domain where ${\downarrow}SH$ is substituted by the non-redundant sharing domain SH^{ρ} [3]. The precision of the analysis is measured by summing results over the success-patterns, for goal-independent (GI) analysis, and in both the call- and success-patterns, for goal-dependent (GD) analysis, for each procedure. For the domains tested, that is, $Pos{+}{\downarrow}SH+Free{+}Lin$, abbreviated as P+DSH+F+L, and $Pos{+}SH^{\rho}{+}Free{+}Lin$, abbreviated as P+NSH+F+L, the precision results consist of: the total number of definitely non-sharing pairs of program variables, NSP, the total number of definitely ground variables, GV, and the total number of definitely linear variables that are possibly not ground, LV. The freeness results are not compared because, as we have shown in [19], freeness is not affected, neither by abstracting SH to ${\downarrow}SH$, nor by redundancy elimination.

The comparison involved all the 92 Prolog programs in our current test-suite. On 73 of them there was no difference in precision. This is really remarkable considering that the ${\downarrow}SH$ approximation is rather crude.

The combined domain $Pos + {\downarrow}SH + Lin$ is isomorphic to $\mathsf{ASub} + Pos$ (where ASub is the pair-sharing domain of Søndergaard [18]), and the domain $Pos{+}{\downarrow}SH$ is exactly the domain ASub^{+} defined by Cortesi and Filé in [9]. However, they considered this domain only *en passant* and only from a theoretical point of view. In other words, Fecht has the whole merit for having trusted on this domain from a precision/efficiency perspective.

The results for the remaining 19 programs are summarized in Table 1. The

	Goal-Independent						Goal-Dependent					
	P+DSH+F+L			P+NSH+F+L			P+DSH+F+L			P+NSH+F+L		
Program	NSP	GV	LV	NSP	GV	LV	NSP	GV	LV	NSP	GV	LV
aqua_c	10749	⋆406	2753	?	?	?	16306	⋆1186	2028	?	?	?
bmtp	1451	136	972	1461	136	976						
bryant	784	10	146	1088	10	223	1033	141	58	1781	141	58
caslog	6456	⋆466	1588	7027	⋆474	1615	11073	⋆1625	1079	?	?	?
cg_parser	136	31	159	138	31	160						
dpos_an	92	40	76	95	40	76	183	76	53	183	76	53
lg_sys	7274	645	2260	7334	645	2261						
nand	473	23	182	475	23	182	1341	481	70	1341	481	70
nbody	261	52	104	262	52	104	477	151	41	478	151	41
oldchina	2185	285	1163	2193	285	1166	3985	802	760	?	?	?
quot_an	288	37	160	288	37	160	639	159	122	646	159	122
reg	774	42	272	796	42	284	207	67	52	207	67	52
rubik	70	⋆55	110	73	⋆76	93	174	⋆110	124	201	⋆200	103
scc	63	0	37	63	0	37	503	174	46	506	174	46
sfecht	28	0	14	85	0	31	221	0	47	278	0	64
simple_an	370	27	139	373	27	139	572	82	76	639	82	76
slice	427	126	453	428	126	453						
spsys	788	81	386	800	81	394						
trs	32	6	22	53	6	22	73	⋆12	20	104	⋆12	20

Table 1. $Pos + \downarrow SH + Free + Lin$ vs $Pos + SH^{\rho} + Free + Lin$: precision.

blank entries in the goal-dependent columns are for those program whose goal-dependent analysis is pointless. This usually happens because the program contains a procedure call to an unknown procedure (e.g., by means of `call/1`). The China analyzer (i.e., our system [1]) promptly recognizes these cases and reverts to a goal-independent analysis. This is one of the reasons why focusing only on goal-dependent analyses is, in our opinion, a mistake. The other reason being that the ability of analyzing libraries once and for all is desirable and, more generally, so is the separate analysis of different program modules, especially in very large projects. Focusing only on goal-independent analyses is the opposite mistake: GD analyses, when possible, are more precise than GI ones. For these reasons, we insist in presenting experimental results for both.

A star symbol ($\star$) in the GV column signifies that one of the widenings we employ on the GER representation of *Pos* fired. This is a widening imposing a limit on the number of ROBDD nodes simultaneously allocated. It makes approximations of the R (ROBDD) component when this limit is reached[2], while retaining full precision on the G (definitely ground variables) and the E (classes of equivalent variables) components [2, 5]. The scarcity of stars in this and the following tables, shows how seldom this widening is actually required.[3]

[2] That is, by approximating $x \wedge y$ with x or with y, $x \vee y$ with *true* and so forth.

[3] Indeed, the newest version of China avoids also the widening for the `caslog` program.

Apart from `sfecht`, which is a synthetic benchmark designed in order to show that arbitrary precision losses are possible with $\downarrow SH$, Table 1 illustrates how heavy precision penalties can be incurred by $\downarrow SH$ even on real programs. Most notably, for `bryant` we see a precision loss as high as 28% on goal-independent analysis (GI) and 42% on goal-dependent analysis. In addition, `simple_an` loses 10% (GD), while `trs` loses 40% (GI) and 30% (GD). Note that, for these programs, the *Pos* widening fires only on the GD analysis of `trs`. The `rubik` program shows an interesting phenomenon: here the *Pos* widening fires incurring a precision loss of exactly 1 ground variable (a critical one indeed), but SH^{ρ} saves the day by recovering the lost groundness information. A similar thing happens for `caslog`. Thus, the widely held opinion (now proved in [8]) that Sharing does not help *Pos* on groundness does not carry through when widenings are considered.

While space limitations do not allow to report full timing information, we can easily confirm Fecht's claim: the speedup is dramatic. Just a few examples: the fixpoint computation time in seconds for `bmtp`, `caslog`, `lg_sys`, and `spsys` drops from 15.6, 614.7, 735.9, and 2.2, to 0.8, 2.0, 3.3, and 0.6, respectively. All the experiments described in this paper were performed on a PC equipped with an AMD K6@400MHz, 128MB of main memory, and running Linux 2.2.1.

1.5 The Present Work

The objective of this work, after having recognized that Fecht's approach incurs significant precision loss on several real programs, is to improve the state of the art in mode analysis, in general, and sharing analysis in particular.

We moved from the observation that, when the sharing-sets become large, then they are at the same time heavy to manipulate and, at least for a subset of the variables involved, light as far as information content is concerned. We thus introduce a new representation for set-sharing made of two components. They are both sharing-sets. However, while the second one is interpreted in the usual way, the first component records worst-case sharing assumptions of sets of variables.

We define the operations required for the analysis with this representation, and we prove them correct. We also introduce two safe optimizations that turn out to be very effective in practice.

We then show how the proposed representation supports a variety of widenings. One of those is a simple adaptation of Fecht's idea. Others are much more sophisticated and involve only a limited precision loss. However, in spite of thorough experimentation (of which only a tiny fraction can be reported here) we found that the first widening we propose is hard to improve on, provided *Pos* is included in the domain. This suggests that what is lost by this widening is mostly constituted by ground dependencies, and these can be recovered (and improved) by the *Pos* component. We show that when *Pos* is not included, a widening based on cliques of sharing pairs is preferred. Since some authors advocate the use of Sharing without coupling it with *Pos* (we do not share this view), this is an important message for them.

Among the contributions of this paper we would like to stress the following: we present a data-flow analysis for groundness, freeness, pair-sharing, and linearity, with unprecedented levels of precision *and* efficiency. With the implementation described in this paper, the China analyzer is able to honor one of its most important design goals: never crash (e.g., by exhausting all the available memory), always terminate with a correct result and in reasonable time.

The paper is structured as follows: In Section 2 we briefly recall the required notions and notations, even though we assume general acquaintance with the topics of abstract interpretation, sharing analysis and groundness analysis. Section 3 introduces a simple domain for sharing analysis that supports the implementation of several widening techniques. In particular, with this domain it is straightforward to turn Fecht's idea into a proper widening. This is done in Section 4, after the introduction of an infinite family of widenings and the proof of their safety. More precise widenings are also considered. The experimental evaluation of the proposed approach is presented in Section 5. Section 6 concludes with some final remark. The reader is referred to [19] for full proofs of all the results presented in this paper, and for more material on this subject.

2 Preliminaries

For any set S, $\wp(S)$ denotes the power set of S and $\# S$ is the cardinality of S. A monotone and idempotent self-map $\rho \colon P \rightarrow P$ over a poset $\langle P, \preceq \rangle$ is called a *closure operator* (or *upper closure operator*) if it is also *extensive*, namely $\forall x \in P : x \preceq \rho(x)$. In this paper, we assume there is a fixed and finite set of variables of interest denoted by VI. If t is a first-order term over VI, then $vars(t)$ denotes the set of variables in t. *Bind* denotes the set of equations of the form $x = t$ where $x \in VI$ and t 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 [14] that this is not required to ensure correctness of the operations of $\mathsf{Sharing}$ and its derivatives. The following definition is a simplification of the standard definition for the $\mathsf{Sharing}$ domain [10, 14, 15] where the set of variables of interest is fixed and finite.

Definition 1. (The *set-sharing* domain SH.) *The set SH is defined as $SH \stackrel{\mathrm{def}}{=} \wp(SG)$, where $SG \stackrel{\mathrm{def}}{=} \{\, S \in \wp(VI) \mid S \neq \varnothing \,\}$.*

We now introduce the required abstract operations over SH.

Definition 2. (Some abstract operations over SH.) *The binary function* $\mathrm{proj} \colon SH \times \wp(VI) \rightarrow SH$ projects *an element of SH onto a subset of VI: if $sh \in SH$ and $V \in \wp(VI)$, then* $\mathrm{proj}(sh, V) \stackrel{\mathrm{def}}{=} \{\, S \cap V \mid S \in sh, S \cap V \neq \varnothing \,\}$.

For each $sh \in SH$ and each $V \in \wp(VI)$, the extraction of the relevant component of sh with respect to V *is encoded by the function* $\mathrm{rel} \colon \wp(VI) \times SH \rightarrow SH$ *defined as* $\mathrm{rel}(V, sh) \stackrel{\mathrm{def}}{=} \{\, S \in sh \mid S \cap V \neq \varnothing \,\}$.

For $sh \in SH$ *and* $V \in \wp(VI)$*, the exclusion of the* irrelevant component of sh with respect to V *is encoded by the function* $\overline{\mathrm{rel}} \colon \wp(VI) \times SH \to SH$ *defined as* $\overline{\mathrm{rel}}(V, sh) \stackrel{\mathrm{def}}{=} sh \setminus \mathrm{rel}(V, sh)$.

The star-union *function* $(\cdot)^\star \colon SH \to SH$*, is given, for each* $sh \in SH$*, by* $sh^\star \stackrel{\mathrm{def}}{=} \{\, S \in SG \mid \exists n \geq 1 \,.\, \exists T_1, \ldots, T_n \in sh \,.\, S = T_1 \cup \cdots \cup T_n \,\}$.

For each $sh_1, sh_2 \in SH$*, the* binary union *function* $\mathrm{bin} \colon SH \times SH \to SH$ *is given by* $\mathrm{bin}(sh_1, sh_2) \stackrel{\mathrm{def}}{=} \{\, S_1 \cup S_2 \mid S_1 \in sh_1, S_2 \in sh_2 \,\}$.

We also use the self-bin-union *function* $\mathrm{sbin} \colon SH \to SH$ *which is given by* $\mathrm{sbin}(sh) \stackrel{\mathrm{def}}{=} \mathrm{bin}(sh, sh)$

The function amgu *captures the effects of a binding on an SH element. Let* $(x = t) \in Bind$*,* $sh \in SH$*,* $V_x = \{x\}$*,* $V_t = vars(t)$*, and* $V_{xt} = V_x \cup V_t$*. Then*

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

The domain SH captures *set-sharing*. However, the property we wish to detect is *pair-sharing* and, for this, it has been shown that SH includes unwanted redundancy [3].

Definition 3. (Redundancy.) *Let* $sh \in SH$ *and* $S \in SG$*.* S *is* redundant for sh *if and only if* $\# S > 2$ *and* $\mathrm{pairs}(S) = \bigcup \{\, \mathrm{pairs}(T) \mid T \in sh, T \subset S \,\}$ *where* $\mathrm{pairs}(S) \stackrel{\mathrm{def}}{=} \{\, P \in \wp(S) \mid \# P = 2 \,\}$.

Definition 4. (The domain SH^ρ.) *The function* $\rho \colon SH \to SH$ *is given, for each* $sh \in SH$*, by* $\rho(sh) \stackrel{\mathrm{def}}{=} sh \cup \{\, S \in SG \mid S \text{ is redundant for } sh \,\}$*. Then* $SH^\rho = \rho(SH) = \{\, \rho(sh) \mid sh \in SH \,\}$.

We use the notation $sh_1 =_\rho sh_2$ and $sh_1 \subseteq_\rho sh_2$ to denote $\rho(sh_1) = \rho(sh_2)$ and $\rho(sh_1) \subseteq \rho(sh_2)$, respectively. The advantage of SH^ρ is that we can replace the star-union operation in the definition of the amgu by self-bin-union without loss of precision [3]. In particular, it is shown that

$$\mathrm{amgu}(sh, x = t) =_\rho \overline{\mathrm{rel}}(V_{xt}, sh) \cup \mathrm{bin}\Bigl(\mathrm{sbin}\bigl(\mathrm{rel}(V_x, sh)\bigr), \mathrm{sbin}\bigl(\mathrm{rel}(V_t, sh)\bigr)\Bigr). \quad (1)$$

3 A New Representation for Set-Sharing

We introduce here a new representation for set-sharing. It is made up of two components: one is the original set-sharing domain while the other represents all possible subsets of each of its elements and, for this reason, is called a *clique-set*.

Definition 5. (Clique-set.) *A* clique-set *is an element of* CL *and* $CL \stackrel{\mathrm{def}}{=} SH$.

An element of a clique-set is called a *clique*.

Definition 6. (Sharing-sets representation for clique-sets.) *The (overloaded) functions* $\downarrow \colon SG \to SH$ *and* $\downarrow \colon CL \to SH$ *are given, for each* $C \in SG$ *and each* $cl \in CL$*, by* $\downarrow C \stackrel{\mathrm{def}}{=} \wp(C) \setminus \{\varnothing\}$ *and* $\downarrow cl \stackrel{\mathrm{def}}{=} \bigcup_{C \in cl} \downarrow C$*. Observe that* $\downarrow$ *is an upper closure operator over SH. If* $cl \in CL$ *and* $C \in SG$ *then we say that* C *is* down-redundant *in* cl *if there exists* $C' \in cl$ *such that* $C \subset C'$.

The addition or removal of down-redundant elements to or from a clique-set makes no difference to the sharing-sets that it represents. So, a clique represents a *worst case*[4] pair-sharing condition on the set of variables it contains.

In an implementation, as we need to keep the clique-sets as small as possible, down-redundant cliques are removed via a *normalization* function.

Definition 7. (Normalization of clique-sets.) *The normalization function* $|\cdot|\colon CL \to CL$ *is given, for each* $cl \in CL$, *by*

$$|cl| \stackrel{\text{def}}{=} cl \setminus \{\, C \in cl \mid C \textit{ is down-redundant for } cl \,\}.$$

We now define abstract unification over clique-sets and state its soundness.

Definition 8. (Abstract unification over cliques.) *For each* $V \in \wp(VI)$ *and each* $cl \in CL$, *the function* $\overline{\mathrm{rel}}^{\mathsf{CL}}\colon \wp(VI) \times CL \to CL$ *is given by*

$$\overline{\mathrm{rel}}^{\mathsf{CL}}(V, cl) \stackrel{\text{def}}{=} \{\, C \setminus V \mid C \in cl \,\} \setminus \{\varnothing\}.$$

The function $\mathrm{amgu}^{\mathsf{CL}}\colon CL \times Bind \to CL$ *is given, for each* $cl \in CL$ *and each* $(x = t) \in Bind$, *by*

$$\mathrm{amgu}^{\mathsf{CL}}(cl, x = t) \stackrel{\text{def}}{=} \overline{\mathrm{rel}}^{\mathsf{CL}}(V_{xt}, cl) \cup \mathrm{bin}\bigl(\mathrm{sbin}(cl_x), \mathrm{sbin}(cl_t)\bigr),$$

where $cl_x = \mathrm{rel}(V_x, cl)$, $cl_t = \mathrm{rel}(V_t, cl)$, $V_x = \{x\}$, $V_t = vars(t)$, *and, finally,* $V_{xt} = V_x \cup V_t$.

Theorem 1. *For each* $cl \in CL$ *and each* $(x = t) \in Bind$,

$$\mathrm{amgu}\bigl(\downarrow cl, x = t\bigr) \subseteq_\rho \downarrow \mathrm{amgu}^{\mathsf{CL}}(cl, x = t).$$

Because cliques represent their downward closure, $\mathrm{amgu}^{\mathsf{CL}}$ introduces down-redundant cliques when both the relevant components are non-empty. As already observed (without proof) and implemented by Fecht, there are two optimizations for computing the $\mathrm{amgu}^{\mathsf{CL}}$ that enable useful efficiency improvements. These are reformulated in Section 3.1 for the domains defined here.

We next define our new sharing domain for widening.

Definition 9. (The SH^{W} representation.) *The set* SH^{W} *is given by*

$$SH^{\mathsf{W}} \stackrel{\text{def}}{=} \bigl\{\, (cl, sh) \bigm| cl \in CL, sh \in SH \,\bigr\}$$

and is ordered by $\sqsubseteq$ *defined as follows, for each* $shw, (cl_1, sh_1), (cl_2, sh_2) \in SH^{\mathsf{W}}$:

$$(cl_1, sh_1) \sqsubseteq (cl_2, sh_2) \iff (cl_1 \subseteq cl_2) \land (sh_1 \subseteq sh_2).$$

It can be seen that SH^{W} *is a complete lattice.*

The sharing-set represented by an element of SH^{W} *is given by the function* $\mathcal{I}(\cdot)\colon SH^{\mathsf{W}} \to SH$ *defined, for each* $(cl, sh) \in SH^{\mathsf{W}}$, *by* $\mathcal{I}\bigl((cl, sh)\bigr) \stackrel{\text{def}}{=} \downarrow cl \cup sh$. *The normalization of an element of* SH^{W} *is given by* $|\cdot|\colon SH^{\mathsf{W}} \to SH^{\mathsf{W}}$ *defined, for each* $(cl, sh) \in SH^{\mathsf{W}}$, *by* $|(cl, sh)| \stackrel{\text{def}}{=} \bigl(|cl|, sh \setminus \downarrow cl\bigr)$.

[4] While this terminology is due to Langen [16], our definition differs from the one he used.

The normalization removes unnecessary elements from a description in SH^{W}. We now define an upper closure operator ϱ inducing an equivalence relation on the elements of SH^{W}.

Definition 10. (The $\varrho(SH^{\mathsf{W}})$ domain.) *The function $\varrho\colon SH^{\mathsf{W}} \to SH^{\mathsf{W}}$ is given, for each $shw \in SH^{\mathsf{W}}$ with $shw \stackrel{\mathrm{def}}{=} (cl, sh)$, by $\varrho(shw) \stackrel{\mathrm{def}}{=} \bigl(\rho(\downarrow cl), \rho\bigl(\mathcal{I}(shw)\bigr)\bigr)$. Then ϱ is an upper closure operator for SH^{W} [19]. We will use the notation $shw_1 =_\varrho shw_2$ to denote $\varrho(shw_1) = \varrho(shw_2)$ and and $shw_1 \sqsubseteq_\varrho shw_2$ to denote $\varrho(shw_1) \sqsubseteq \varrho(shw_2)$.*

As we will see, when $shw_1 =_\varrho shw_2$, shw_1 and shw_2 behave the same way as far as representing and propagating pair-sharing and groundness is concerned. In other words, ϱ is for SH^{W} what ρ is for SH.

Proposition 1. *If $shw \in SH^{\mathsf{W}}$, then $\mathcal{I}(shw) =_\rho \mathcal{I}\bigl(\varrho(shw)\bigr)$ and $shw =_\varrho |shw|$.*

Definition 11. (Operations over SH^{W}.) *For each $(cl, sh), (cl_i, sh_i) \in SH^{\mathsf{W}}$, $i = 1$, 2, and each $V \in \wp(VI)$, the functions $\mathrm{rel}^{\mathsf{W}}, \overline{\mathrm{rel}}^{\mathsf{W}}\colon \wp(VI) \times SH^{\mathsf{W}} \to SH^{\mathsf{W}}$ and $\cup^{\mathsf{W}}, \mathrm{bin}^{\mathsf{W}}\colon SH^{\mathsf{W}} \times SH^{\mathsf{W}} \to SH^{\mathsf{W}}$, the functions $\mathrm{sbin}^{\mathsf{W}}\colon SH^{\mathsf{W}} \to SH^{\mathsf{W}}$ and $\mathrm{amgu}^{\mathsf{W}}\colon SH^{\mathsf{W}} \times Bind \to SH^{\mathsf{W}}$ are defined as follows:*

$$
\begin{aligned}
\mathrm{rel}^{\mathsf{W}}\bigl(V, (cl, sh)\bigr) &\stackrel{\mathrm{def}}{=} \bigl(\mathrm{rel}(V, cl), \mathrm{rel}(V, sh)\bigr),\\
\overline{\mathrm{rel}}^{\mathsf{W}}\bigl(V, (cl, sh)\bigr) &\stackrel{\mathrm{def}}{=} \bigl(\overline{\mathrm{rel}}^{\mathsf{CL}}(V, cl), \overline{\mathrm{rel}}(V, sh)\bigr),\\
(cl_1, sh_1) \cup^{\mathsf{W}} (cl_2, sh_2) &\stackrel{\mathrm{def}}{=} \bigl(cl_1 \cup cl_2, sh_1 \cup sh_2\bigr),\\
\mathrm{bin}^{\mathsf{W}}\bigl((cl_1, sh_1), (cl_2, sh_2)\bigr) &\stackrel{\mathrm{def}}{=} \bigl(\mathrm{bin}(cl_1, cl_2) \cup \mathrm{bin}(cl_1, sh_2) \cup \mathrm{bin}(sh_1, cl_2),\\
&\qquad \mathrm{bin}(sh_1, sh_2)\bigr),\\
\mathrm{sbin}^{\mathsf{W}}\bigl((cl, sh)\bigr) &\stackrel{\mathrm{def}}{=} \mathrm{bin}^{\mathsf{W}}\bigl((cl, sh), (cl, sh)\bigr)\\
&= \bigl(\mathrm{sbin}(cl) \cup \mathrm{bin}(cl, sh), \mathrm{sbin}(sh)\bigr),\\
\mathrm{amgu}^{\mathsf{W}}(shw, x = t) &\stackrel{\mathrm{def}}{=} \overline{\mathrm{rel}}^{\mathsf{W}}(V_{xt}, shw)\\
&\qquad \cup^{\mathsf{W}} \mathrm{bin}^{\mathsf{W}}\Bigl(\mathrm{sbin}^{\mathsf{W}}\bigl(\mathrm{rel}^{\mathsf{W}}(V_x, shw)\bigr),\\
&\qquad\qquad \mathrm{sbin}^{\mathsf{W}}\bigl(\mathrm{rel}^{\mathsf{W}}(V_t, shw)\bigr)\Bigr),
\end{aligned}
$$

where $V_x = \{x\}$, $V_t = vars(t)$, and $V_{xt} = V_x \cup V_t$.

The next two theorems, proven in [19], state the correctness of $\mathrm{amgu}^{\mathsf{W}}$ and that ϱ is a congruence for it.

Theorem 2. *For each $shw \in SH^{\mathsf{W}}$ and each $(x = t) \in Bind$,*

$$\mathrm{amgu}\bigl(\mathcal{I}(shw), x = t\bigr) \subseteq_\rho \mathcal{I}\bigl(\mathrm{amgu}^{\mathsf{W}}(shw, x = t)\bigr).$$

Theorem 3. *For each $shw_1, shw_2 \in SH^{\mathsf{W}}$ and each $(x = t) \in Bind$, such that $shw_1 =_\varrho shw_2$,*

$$\mathrm{amgu}^{\mathsf{W}}(shw_1, x = t) =_\varrho \mathrm{amgu}^{\mathsf{W}}(shw_2, x = t).$$

Combining Proposition 1 with Theorem 3 we obtain

$$\mathrm{amgu}^{\mathsf{W}}(shw, x = t) =_{\varrho} \mathrm{amgu}^{\mathsf{W}}\bigl(|shw|, x = t\bigr).$$

In Eq. (1), the basic amgu operation is defined using the SH^{ρ} domain. However, when we have freeness and linearity information it has been proven that we can avoid one or both of the self-bin-unions occurring as components of the binary union operation. The question arises as to whether this optimization can be applied when we have the $\mathrm{amgu}^{\mathsf{W}}$ operation for the SH^{W} domain. That is, can we avoid the corresponding $\mathrm{sbin}^{\mathsf{W}}$ operations under the same linearity and freeness conditions? The answer is *yes*, we can generalize Theorem 2 and show that such an optimization is sound. However, the completeness of Theorem 3 does not hold in all cases and further work on this aspect is ongoing.

3.1 Optimizations

We can optimize the computation of $\mathrm{amgu}^{\mathsf{W}}$ in two ways. To explain these, we need some extra notation. Let: $shw = (cl, sh) \in SH^{\mathsf{W}}$ and $x = t \in \mathit{Bind}$; $V_x = \{x\}$, $V_t = \mathit{vars}(t)$, and $V_{xt} = V_x \cup V_t$; $shw_x = (cl_x, sh_x) = \mathrm{rel}^{\mathsf{W}}(V_x, shw)$ and $shw_t = (cl_t, sh_t) = \mathrm{rel}^{\mathsf{W}}(V_t, shw)$; and, finally,

$$shw_{\mathrm{rel}} = (cl_{\mathrm{rel}}, sh_{\mathrm{rel}}) = \mathrm{bin}^{\mathsf{W}}\bigl(\mathrm{sbin}^{\mathsf{W}}(shw_x), \mathrm{sbin}^{\mathsf{W}}(shw_t)\bigr).$$

Theorem 4. *If neither $shw_x = (\varnothing, \varnothing)$ nor $shw_t = (\varnothing, \varnothing)$, then*

$$\overline{\mathrm{rel}}^{\mathsf{W}}(V_{xt}, shw) = \bigl(\overline{\mathrm{rel}}(V_{xt}, cl) \cup cl', \overline{\mathrm{rel}}(V_{xt}, sh)\bigr),$$

where $cl' \subseteq \downarrow cl_{\mathrm{rel}}$.

Theorem 5. *Suppose that $C_x = \bigcup cl_x$, $C_t = \bigcup cl_t$, $S_x = \bigcup sh_x$, $S_t = \bigcup sh_t$, $A_{xt} = \mathrm{bin}\bigl(\mathrm{sbin}(sh_x), \mathrm{sbin}(sh_t)\bigr)$, $B_{xt} = \mathrm{bin}(sh_x, sh_t)$, $B_x = \mathrm{bin}(cl_x, sh_x)$, and $B_t = \mathrm{bin}(cl_t, sh_t)$. Let also*

$$shw^{\mathrm{opt}}_{\mathrm{rel}} \stackrel{\mathrm{def}}{=} \begin{cases} \bigl(\{C_x \cup C_t \cup S_x \cup S_t\}, \varnothing\bigr), & \text{if } cl_x \neq \varnothing, cl_t \neq \varnothing; \\ \bigl(\{C_x \cup S_t\} \cup B_x \cup B_{xt}, A_{xt}\bigr), & \text{if } cl_x \neq \varnothing, cl_t = \varnothing; \\ \bigl(\{C_t \cup S_x\} \cup B_t \cup B_{xt}, A_{xt}\bigr), & \text{if } cl_x = \varnothing, cl_t \neq \varnothing; \\ (\varnothing, A_{xt}), & \text{if } cl_x = \varnothing, cl_t = \varnothing. \end{cases}$$

Then,

$$shw_{\mathrm{rel}} =_{\varrho} \begin{cases} shw^{\mathrm{opt}}_{\mathrm{rel}}, & \text{if } shw_x \neq (\varnothing, \varnothing), shw_t \neq (\varnothing, \varnothing); \\ (\varnothing, \varnothing), & \text{otherwise.} \end{cases}$$

Both these results are proven in [19]. These can then be combined to provide the following optimization (also proven in [19]) for the computation of $\mathrm{amgu}^{\mathsf{W}}$.

Corollary 1. *Assuming the notation used in* Theorem 5*, then*

$$\operatorname{amgu}^{\mathsf{w}}(shw, x = t)$$
$$=_{\varrho} \begin{cases} \bigl(\overline{\operatorname{rel}}(V_{xt}, cl), \overline{\operatorname{rel}}(V_{xt}, sh)\bigr) \cup^{\mathsf{w}} shw^{\mathrm{opt}}_{\mathrm{rel}}, & \textit{if } shw_x \neq (\varnothing, \varnothing), shw_t \neq (\varnothing, \varnothing); \\ \overline{\operatorname{rel}}^{\mathsf{w}}(V_{xt}, shw), & \textit{otherwise.} \end{cases}$$

Observe that this result applies to the basic $\operatorname{amgu}^{\mathsf{w}}$ operation as given in Definition 11. When one or both of the self-bin-unions here is omitted due to available freeness and linearity information, then $=_{\varrho}$ in the corollary becomes $\sqsubseteq_{\varrho}$ and we may lose precision. Further work on this subject is ongoing.

4 Widening Set-Sharing

We can now define a family of *unary* widenings over SH^{w}.

Definition 12. (Widening for SH^{w}.) *The function $\nabla \colon SH^{\mathsf{w}} \to SH^{\mathsf{w}}$ is a* widening for SH^{w} *if, for each $shw \in SH^{\mathsf{w}}$, we have $shw \sqsubseteq_{\varrho} \nabla\, shw$.*

The following result establishes the safety of such widening operators.

Theorem 6. *For each $shw \in SH^{\mathsf{w}}$ and each $(x = t) \in Bind$ we have*

$$\operatorname{amgu}^{\mathsf{w}}(shw, x = t) \sqsubseteq_{\varrho} \operatorname{amgu}^{\mathsf{w}}\bigl(\nabla\, shw, x = t\bigr).$$

The obvious corollary is that any analysis using these widenings, possibly a different widening at each step of the analysis, is correct. After widening we always normalize the resulting description to provide a smaller representation. Moreover, it is also shown in [19] that similar results hold for each of the component operators, such as $\operatorname{bin}^{\mathsf{w}}$, for $\operatorname{amgu}^{\mathsf{w}}$. Thus we can (and do) safely widen and normalize within the actual computation of $\operatorname{amgu}^{\mathsf{w}}$. The analyzer has the freedom of using whichever widening suits its current needs. Those needs can be dictated by a number of heuristics. Of course, really useful widenings are *guarded* by some applicability condition. The simplest conditions are those based on the cardinality of the sets in the SH^{w} description. For example, for each widening ∇ and for suitable choices of $f \colon \mathbb{N}^2 \to \mathbb{N}$ and $n \in \mathbb{N}$, one can define

$$\nabla_{f,n}(cl, sh) \stackrel{\mathrm{def}}{=} \begin{cases} \nabla(cl, sh), & \text{if } f(\#\, cl, \#\, sh) > n, \\ (cl, sh), & \text{otherwise.} \end{cases}$$

We order the widenings in the obvious way. If ∇_1 and ∇_2 are two widenings and for all shw, $\nabla_1(shw) \sqsubseteq_{\varrho} \nabla_2(shw)$, then let $\nabla_1 \sqsubseteq_{\varrho} \nabla_2$, meaning that ∇_1 is more precise than ∇_2.

At the top end of the scale of widenings we have two *panic widenings*. They are defined by

$$\nabla^{p}(cl, sh) \stackrel{\mathrm{def}}{=} \Bigl(cl \cup \bigl\{\textstyle\bigcup sh\bigr\}, \varnothing\Bigr),$$
$$\nabla^{P}(cl, sh) \stackrel{\mathrm{def}}{=} \Bigl(\bigl\{\textstyle\bigcup cl \cup \bigcup sh\bigr\}, \varnothing\Bigr).$$

The panic widenings are present in the CHINA implementation, with very strict guards, only to obey the "never crash" motto: no real program we have access to makes them fire.

At the other extreme we have very soft widenings.

Definition 13. (Cautious widening.) *A widening* $\nabla \colon SH^{\mathsf{W}} \to SH^{\mathsf{W}}$ *is called a* cautious widening *if, for each* $shw \in SH^{\mathsf{W}}$*,*

$$\mathcal{I}(\nabla\, shw) =_\rho \mathcal{I}(shw).$$

Thus, a widening is cautious if it is invariant with respect to the set-sharing representation. In particular, it never introduces new pair-sharings nor new singletons in the description. However, information is lost as soon as the operations for the analysis given by Definition 11 are considered. For example, consider two elements of SH^{W}: $shw_1 \stackrel{\mathrm{def}}{=} \bigl(\varnothing, \{x, y, z, xy, xz, yz\}\bigr)$ and $shw_2 \stackrel{\mathrm{def}}{=} \bigl(\{xyz\}, \varnothing\bigr)$ so that we have $\mathcal{I}(shw_1) =_\rho \mathcal{I}(shw_2)$ but $\varrho(shw_1) \neq \varrho(shw_2)$. While sharing between y and z is not contemplated in $\mathrm{rel}^{\mathsf{W}}\bigl(\{x\}, shw_1\bigr) = \bigl(\varnothing, \{x, xy, xz\}\bigr)$, the same does not hold for $\mathrm{rel}^{\mathsf{W}}\bigl(\{x\}, shw_2\bigr) = shw_2$.

A useful cautious widening is the *gentle widening*, defined as follows. Consider $shw \in SH^{\mathsf{W}}$, and let us define the undirected graph $G \stackrel{\mathrm{def}}{=} (N, E)$ such that $N \stackrel{\mathrm{def}}{=} \{\, x \mid \{x\} \in \mathcal{I}(shw) \,\}$ and $E \stackrel{\mathrm{def}}{=} \bigl\{\, (x, y) \bigm| \{x, y\} \in \mathcal{I}(shw), x, y \in N, x \neq y \,\bigr\}$. Then

$$\nabla^G\, shw \stackrel{\mathrm{def}}{=} \bigl(\{C_1, \ldots, C_k\}, sh\bigr),$$

where $C_1, \ldots, C_k$ are all the maximal cliques of G. Note that, although the problem of enumerating all the maximal cliques of an undirected graph is NP-complete, this does not seem to be a problem for the graphs arising during the analysis of even the biggest real programs. For the experimentation we used the algorithm by Bron and Kerbosch [6], which is *Algorithm 457* in the ACM collection, even though more efficient algorithms are present in the literature.

Of intermediate precision is the widening based on Fecht's idea, which we will call *Fecht's widening*. It is simply given by

$$\nabla^F(cl, sh) \stackrel{\mathrm{def}}{=} (cl \cup sh, \varnothing).$$

This widening is not cautious. However, it does not introduce new pairs. As it can introduce new singletons, it may destroy ground dependencies, and this is why this kind of widening is better coupled with *Pos*.

5 Experimental Evaluation

For the experimental evaluation of the Fecht's widening ∇^F, precision is compared with respect to the non-redundant sharing domain SH^ρ. In fact, this approach is almost always as precise as the optimal one using SH^ρ.

	Goal-Independent						Goal-Dependent					
	P+WSH+F+L			P+NSH+F+L			P+WSH+F+L			P+NSH+F+L		
Program	NSP	GV	LV	NSP	GV	LV	NSP	GV	LV	NSP	GV	LV
`aqua_c`	11147	⋆406	2757	?	?	?	16364	⋆118 8	2028	?	?	?
`caslog`	6553	⋆474	1615	7027	⋆474	1615	11338	⋆1739	1062	?	?	?
`oldchina`	2193	285	1166	2193	285	1166	3985	802	760	?	?	?
`quot_an`	288	37	160	288	37	160	639	159	122	646	159	122

Table 2. $Pos + SH^{\mathsf{W}} + Free + Lin$ vs $Pos + SH^{\rho} + Free + Lin$ using ∇^F_{100}: precision.

For this and the following experiments, the widening was guarded by a size threshold of 100 on the second component (i.e., the normal sharing part). In other words, immediately before each abstract mgu operation the analyzer operated redundancy elimination, as usual. If after this the operand (cl, sh) was such that $\# sh > 100$, then (cl, sh) was substituted by $\nabla^F(cl, sh)$. Let us call this guarded widening ∇^F_{100}. The results are reported in Table 2. Note that only the programs where the analysis with SH^{W} gives different results from the analysis with SH^{ρ} are reported in the table. Thus, for all the programs in the test-suite, the analysis with SH^{W} using the (rather drastic) widening ∇^F_{100} gives the same results obtainable (at a much higher cost) with SH^{ρ}, apart from those in Table 2. For `aqua_c` we obtain termination in reasonable time, as with Fecht's technique but with higher precision. The same holds for the GD analysis of `caslog` and `oldchina`. However, while the GI analysis of `oldchina` is "optimal" (meaning "as precise as SH^{ρ}"), this is not the case for `caslog`. Non-optimality happens also for the GD analysis of `quot_an`.

Obviously, ∇^F_{100} is never less precise than Fecht's domain. What is surprising, however, is that it is almost as efficient. The timings and the number of applications of the widening are reported in Table 3 for all the programs such that at least one timing was above 0.4 seconds. The first observation to be made is that the widening comes into play only a few times on the test-suite. On average, it is safe to say that on 99.9% of cases the sharing-sets remain of reasonable size (100 groups or less in this experiment). Table 3 says that this definition of "reasonable" makes sense: for those programs where widening does not take place the difference in performance between Fecht's domain and our SH^{W} with the ∇^F_{100} widening is very limited. Analysis of `aqua_c` shows that limiting precision may cost (less precision means more self-bin-unions to perform, thus even less precision, ...).

The results on the precision of ∇^F_{100} are so good that we are left with a ridiculous test-suite for checking how much we can improve by using a more cautious widening. Our experimentation showed that the gentle widening ∇^G_{100} improves over ∇^F_{100} only on `quot_an`. The same does, but at a lower price, a bigger widening ∇^g that is defined as ∇^G apart from the fact that singletons are disregarded. In other words, the undirected graph considered for ∇^g, given

	Goal-Independent			Goal-Dependent		
	P+DSH+F+L	P+WSH+F+L		P+DSH+F+L	P+WSH+F+L	
Program	T	T	#W	T	T	#W
`action`	0.1	0.1	1	1.1	1.4	1
`aircraft`	0.2	0.2	0	0.7	0.7	0
`aqua_c`	10.4	10.9	56	48.6	40.7	3
`bmtp`	0.8	0.9	6			
`bryant`	0.1	0.1	0	0.6	1.4	1
`caslog`	2.0	2.5	17	17.7	19.2	22
`chat80`	0.9	1.0	2	4.3	4.9	6
`chat_parser`	0.4	0.4	1	1.7	1.8	1
`dpos_an`	0.2	0.2	0	0.5	0.8	1
`eliza`	0.1	0.1	0	0.2	0.4	1
`4lg_sys`	3.3	3.9	23			
`log_interp`	0.2	0.4	2	0.7	0.9	1
`mixtus`	0.9	0.9	4			
`oldchina`	1.2	1.4	11	7.7	8.3	4
`parser_dcg`	0.2	0.1	0	0.7	0.6	0
`peephole1`	0.1	0.1	0	0.4	0.7	1
`pets_an`	0.8	0.9	4	4.5	4.5	1
`peval`	0.2	0.3	3	0.4	0.5	1
`plaiclp`	0.7	0.7	3			
`press`	0.1	0.1	0	0.4	0.7	0
`quot_an`	0.3	0.4	0	1.3	1.7	1
`read`	0.1	0.1	0	0.3	0.6	1
`reg`	0.4	0.4	4	0.4	0.4	1
`sdda`	0.1	0.1	1	0.2	0.4	2
`sim`	0.2	0.3	2	0.7	0.8	2
`simple_an`	0.1	0.2	0	0.6	0.9	2
`slice`	0.6	0.7	2			
`spsys`	0.6	0.7	5			
`trs`	0.2	0.3	2	0.5	0.6	1
`unify`	0.1	0.1	0	0.5	0.7	0

Table 3. $\mathit{Pos} + {\downarrow}\mathit{SH} + \mathit{Free} + \mathit{Lin}$ vs $\mathit{Pos} + \mathit{SH}^{\mathrm{W}} + \mathit{Free} + \mathit{Lin}$ using ∇^{F}_{100}: timings (T) and number of (sharing) widenings (#W).

$\mathit{shw} \in \mathit{SH}^{\mathrm{W}}$, is $G \stackrel{\mathrm{def}}{=} (N, E)$ such that $E \stackrel{\mathrm{def}}{=} \bigl\{\, (x,y) \bigm| \{x,y\} \in \mathcal{I}(\mathit{shw}), x \neq y \,\bigr\}$ and $N \stackrel{\mathrm{def}}{=} \bigl\{\, x \bigm| (x,y) \in E \text{ or } (y,x) \in E \,\bigr\}$.

Now, suppose we perform sharing analysis *without* combining the sharing domain with *Pos*. Then using a more or less precise widening makes a difference. In Table 4 are reported the results (fixpoint time and number of definitely not-sharing pairs) for SH^{W} with ∇^{F}_{100}, SH^{W} with ∇^{g}_{100}, and plain SH^{ρ}. The GD analysis of `bryant` is particularly eloquent example of the superiority of more cautious widenings when *Pos* is not used.

	Goal-Independent						Goal-Dependent					
	$SH^{\mathsf{W}}, \nabla^F_{100}$		$SH^{\mathsf{W}}, \nabla^g_{100}$		SH^{ρ}		$SH^{\mathsf{W}}, \nabla^F_{100}$		$SH^{\mathsf{W}}, \nabla^g_{100}$		SH^{ρ}	
Program	T	NSP	T	NSP	T	NSP	T	NSP	T	NSP	T	NSP
`aqua_c`	4.1	10703	15.0	10899	?	?	27.6	15754	37.8	15754	?	?
`bryant`	0.2	1066	0.2	1066	0.2	1066	1.0	1033	1.0	1781	0.9	1781
`caslog`	2.1	6506	4.5	6539	744.4	7027	17.9	11054	21.8	11054	?	?
`chat80`	0.6	2536	1.1	2536	9.1	2536	4.4	3923	7.7	3926	285.7	5111
`eliza`	0.1	49	0.1	49	0.1	49	0.3	109	0.5	113	0.5	113
`lg_sys`	2.7	7328	7.9	7334	725.1	7334						
`oldchina`	0.9	2187	2.6	2189	5.0	2193	5.9	3936	9.5	3936	?	?
`pets_an`	0.6	2525	1.3	2563	19.8	2569	3.7	4664	5.3	4664	1006.5	4710
`quot_an`	0.3	288	0.4	288	0.3	288	1.4	639	3.2	646	3.1	646
`simple_an`	0.1	373	0.1	373	0.1	373	0.8	572	1.3	639	17.6	639
`slice`	0.6	426	0.9	428	0.8	428						

Table 4. SH^{W} with ∇^F_{100} vs SH^{W} with ∇^g_{100} vs plain SH^{ρ}: timings and precision.

6 Conclusion

We believe we have made a significant step forward towards the solution of the problem of practical, precise, and efficient sharing analysis of (constraint) logic programs. We have studied a new representation for set-sharing that allows for the incorporation of a variety of widenings. Extensive experimentation has shown that one of these widenings, which is based on an idea of C. Fecht, provides seemingly hard to beat precision and performance, when combined with *Pos*. When this combination is not performed, we have also shown that "more cautious" widenings offer more precision at an acceptable extra-cost.

We are now studying how to increase precision of the analysis beyond the limits of set-sharing. This includes more precise tracking of freeness and linearity, and the efficient incorporation of structural information into the analysis domain.

References

1. R. Bagnara. *Data-Flow Analysis for Constraint Logic-Based Languages.* PhD thesis, Dipartimento di Informatica, Università di Pisa, Corso Italia 40, I-56125 Pisa, Italy, March 1997. Printed as Report TD-1/97.
2. R. Bagnara. Widening *Pos*: Simple, effective, and rarely needed. Unpublished short note, 1998.
3. 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.
4. R. Bagnara, P. M. Hill, and E. Zaffanella. Set-sharing is redundant for pair-sharing. *Theoretical Computer Science*, 1999. To appear.
5. R. Bagnara and P. Schachte. Factorizing equivalent variable pairs in ROBDD-based implementations of *Pos*. In A. M. Haeberer, editor, *Proceedings of the "Seventh

International Conference on Algebraic Methodology and Software Technology (AMAST'98)", volume 1548 of *Lecture Notes in Computer Science*, pages 471–485, Amazonia, Brazil, 1999. Springer-Verlag, Berlin.
6. C. Bron and J. Kerbosch. Finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.
7. 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.
8. M. Codish, H. Søndergaard, and P. J. Stuckey. Sharing and groundness dependencies in logic programs. Submitted for publication.
9. A. Cortesi and G. Filé. Comparison and design of abstract domains for sharing analysis. In D. Saccà, editor, *Proceedings of the "Eighth Italian Conference on Logic Programming (GULP'93)"*, pages 251–265, Gizzeria, Italy, 1993. Mediterranean Press.
10. A. Cortesi and G. Filé. Sharing is optimal. *Journal of Logic Programming*, 38(3):371–386, 1999.
11. 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.
12. C. Fecht. An efficient and precise sharing domain for logic programs. In H. Kuchen and S. D. Swierstra, editors, *Programming Languages: Implementations, Logics and Programs, Proceedings of the Eighth International Symposium*, volume 1140 of *Lecture Notes in Computer Science*, pages 469–470, Aachen, Germany, 1996. Springer-Verlag, Berlin. Poster.
13. C. Fecht. Efficient and precise sharing domains for logic programs. Technical Report A/04/96, Universität des Saarlandes, Fachbereich 14 Informatik, Saarbrücken, Germany, 1996.
14. 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.
15. D. Jacobs and A. Langen. Static analysis of logic programs for independent AND parallelism. *Journal of Logic Programming*, 13(2&3):291–314, 1992.
16. A. Langen. *Static Analysis for Independent And-Parallelism in Logic Programs*. PhD thesis, Computer Science Department, University of Southern California, 1990. Printed as Report TR 91-05.
17. K. Muthukumar and M. Hermenegildo. Compile-time derivation of variable dependency using abstract interpretation. *Journal of Logic Programming*, 13(2&3):315–347, 1992.
18. 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.
19. E. Zaffanella, R. Bagnara, and P. M. Hill. Widening Sharing. Submitted for publication. Available at `http://www.cs.unipr.it/~bagnara/`, 1999.