Enhancing Sharing for Precision

Roberto Bagnara, Enea Zaffanella, Patricia M. Hill

Abstract

Regarding the precision of combined domains including Jacobs and Langen's Sharing there is a core of techniques, such as the standard integration with freeness and linearity information, that are widely used and well accepted. However, a number of other proposals for refined domain combinations have been circulating more or less clandestinely for years. One feature that is common to these proposals is that they do not seem to have undergone experimental evaluation. We question whether significantly more precision is obtainable thanks to these techniques. In particular, 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.

Keywords: Mode Analysis, Sharing Analysis, Abstract Interpretation.

1 Introduction

In this paper, we present one of the final steps in our revamp of the set-sharing domain, Sharing, of Jacobs and Langen [17]. We have first questioned the adequacy of Sharing with respect to the property of interest, that is, *pair-sharing*. In [4] we have proved that Sharing is redundant for pair-sharing and we have identified the weakest abstraction of Sharing that can capture pair-sharing with the same degree of precision. One notable advantage of this abstraction is that the costly star-union operator is no longer necessary. In [16] 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 perform the *occur-check* in the unification procedure. This closed a long-standing

Roberto Bagnara is with the Dipartimento di Matematica, Università degli Studi di Parma, Italy; Email: `bagnara@cs.unipr.it`. Enea Zaffanella is with the Servizio IX Automazione, Università degli Studi di Modena, Italy; Email: `zaffanella.enea@unimo.it`. Patricia M. Hill is with the School of Computer Studies, University of Leeds, U.K.; E-mail: `hill@scs.leeds.ac.uk`.

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 [23], where a family of widenings is presented that allows to achieve the desired goal. Finally, in [24] we have studied the decomposition of Sharing and its 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.

Here we try to answer the following question: how much more precision is attainable by combining Sharing with other domains? A first positive answer was already given in the PhD thesis of A. Langen [21]: *linearity* (the property of all variables that can only be bound to terms without multiple occurrences of variables) can greatly improve the accuracy of sharing analysis. The synergy attainable from the integration between aliasing and freeness information has been pointed out, for the first time, by Muthukumar and Hermenegildo [22]. These standard combinations (see [6] for details) are now widely accepted and nobody would seriously think to perform sharing analysis without them.

However, a number of other proposals for refined domain combinations have been circulating more or less clandestinely for years. One feature that is common to these proposals is that they do not seem to have undergone experimental evaluation. Thus our curiosity is justified, since nobody seems to know whether these techniques enable more precision to be obtained or not.

In this paper we analyze the problem from the point of view of precision only. Although reasonable efficiency is also clearly of interest, this has to be secondary to the question as to whether precision is significantly improved. Only when this is established, should better implementations be researched.

The experimental part of this work has been conducted with the China analyzer [2]. China is a data-flow analyzer for CLP($\mathcal{H}_\mathcal{N}$) languages (i.e., Prolog, CLP($\mathcal{R}$), `clp(FD)` and so forth), $\mathcal{H}_\mathcal{N}$ being an extended Herbrand system where the values of a numeric domain $\mathcal{N}$ can occur as leaves of the terms. China, which is written in C++, performs bottom-up analysis deriving information on both call-patterns and success-patterns by means of program transformations and optimized fixpoint computation techniques.

Because of the exponential complexity of Sharing, stable behavior can only be achieved by means of widening operators [14]. However, widenings also affect the precision of the results and would add unwanted noise to the results reported here. Thus, for an unbiased assessment of the different domain combination enhancements we disabled all the widenings available to China. Unfortunately, the consequence of this is that the analysis of some programs did not terminate in reasonable time or absorbed memory beyond acceptable limits. 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.

The comparison involved all the 160 programs in our current test-suite. This includes several real programs of respectable size. While this test-suite is probably the biggest one reported in the literature on data-flow analysis of (constraint) logic programs, we cannot fail to mention, in good conscience, that we believe a ten times bigger suite would be highly desirable for the obtained results to be conclusive or nearly so. Nonetheless, we believe that from the experimentation presented in this paper some qualitative consideration can safely follow.

The 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 the ordering of bindings so as to maximize the precision of Sharing + *Lin*;[1] 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 + *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 and $\# S$ is the cardinality of S. 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 $\mathit{vars}(t)$ 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 $x \notin \mathit{vars}(t)$, since we have proved in [16] 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 [10, 16, 18] and assume that the set of variables of interest is fixed and finite.

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

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

Definition 2 (Abstract operations over *SH*.) Projection *an element of SH onto a subset of VI is encoded by the binary function* $\mathrm{proj}\colon SH \times \wp(VI) \to SH$*: 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* $\mathrm{rel}\colon \wp(VI) \times SH \to SH$ *defined as* $\mathrm{rel}(V, sh) \stackrel{\mathrm{def}}{=} \{\, S \in sh \mid S \cap V \neq \varnothing \,\}$.

[1]We denote by *Lin* and *Free* the usual domains for linearity and freeness.

For each $sh \in SH$ *and each* $V \in \wp(VI)$*, 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 function $(\cdot)^\star \colon SH \to SH$*, also called* star-union*, is given, for each* $sh \in SH$*, by* $sh^\star \stackrel{\mathrm{def}}{=} \bigl\{\, S \in SG \bigm| \exists n \geq 1 \mathrel{.} \exists T_1, \ldots, T_n \in sh \mathrel{.} S = T_1 \cup \cdots \cup T_n \,\bigr\}$.

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, for each* $sh \in SH$*, 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* $(x = t) \in Bind$*,* $sh \in SH$*,* $V_x = \{x\}$*,* $V_t = vars(t)$*,* $V_{xt} = V_x \cup V_t$*. 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 in [4] that SH includes unwanted redundancy. The same paper introduces an operator ρ on SH and the domain $SH^\rho \stackrel{\mathrm{def}}{=} \rho(SH)$, which is the weakest non-redundant abstraction of SH that is as precise as SH on tracking groundness and pair-sharing. A notable 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. In particular, in [4] it is shown that

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

where we use the notation $sh_1 =_\rho sh_2$ to denote $\rho(sh_1) = \rho(sh_2)$.

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 [11, 24]. 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 [23]. 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 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* [5], 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 linearity information by incorporating *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 [16], Sharing plus *Lin* does not enjoy this property. In particular, it is known since [21, pp. 66-67] that best results are obtained if the grounding unifications are considered before the others.[2] 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: definitely ground variables are propagated from the *Pos* component to the sharing domain. (More precise combinations will be considered in Section 6.) The results are reported in Table 1.

For the tables reported 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 for goal-independent analysis the only differences we have observed concern linearity, for goal dependent analysis there are differences also as far as the numbers of ground variables and of independent pairs are concerned. It is important to notice that our implementation of Sharing with *Free* and *Lin* (whether combined with *Pos* or not) always reorders the bindings so as to handle the grounding ones first. For the remaining comparisons of the different enhancements and precision optimizations, the *Pos* domain is always included unless otherwise stated.

4 Explicit Structural Information

A way of increasing the precision of almost any analysis domain is by incrementing it with structural information. This technique was introduced by A. Cortesi et al. in [12], where the generic structural domain $\mathtt{Pat}(\Re)$ was introduced. Instead of $\mathtt{Pat}(\Re)$ we use the Pattern$(\cdot)$ construction [2, 3], 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 $\mathcal{D}$

[2]As an example, consider the sequences of unifications `f(X, X, Y) = A, X = a` and `X = a, f(X, X, Y) = A` [21, p. 66].

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
Goal-dependent analysis						
`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
`sim`	3004	923	1561/2540	222/366	331/508	101/116
`tsp`	502	220	488/488	122/122	206/220	38/38

Table 1: The effect of integrating *Pos*.

(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 then $\mathcal{D}$ 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 rare occasions where retaining structural information gives rise to a speedup. The reason for this is that maintaining a tuple of terms with many variables, each with its own description, can be cheaper than computing a description for the whole tuple.

We have compared the precision gain made possible by Pattern($\cdot$) applied to the combination of *Pos*, Sharing, *Lin*, and *Free*. The results are reported in Tables 2 and 3. As far as we know, this is the first time that such a comparison is performed on a realistic benchmark suite. Previous attempts failed because of combinatorial explosion in the analysis [9]. What makes this possible now is the adoption of the non-redundant domain in [4], where the exponential star-union operation is replaced by quadratic binary-union, and the integration of this domain with the GER implementation of *Pos* [5].

Tables 2 and 3 indicate that, in many cases, enhancing Sharing with structural information can make useful improvements to precision. Moreover, occasionally (such as for `lc` in Table 3), this improvement is considerable. The integration of structural information is so effective that it seemed worthwhile to conduct all the following experiments both with and without it.

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

5 Reordering the Bindings

While the non-commutativity of Sharing plus *Lin* is well-known, all the examples we found in the literature use a grounding binding to show the existence of the problem. However, we have seen in Section 3 that for grounding bindings the solution is easy:

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

perform them first. Unfortunately, the problem is more general than that.

Let us consider $\{v, w, x, y, z\}$ as the set of relevant variables, and the $\mathsf{Sharing} \times \mathit{Lin}$ element $sh = \langle \{vy, wy, xy, yz\}, \{x, z\} \rangle$, i.e., x and z are the only linear variables and no variable is ground. We now apply the bindings $v = w$ and $x = y$. Using the first ordering we have:

$$\begin{aligned} sh_1 &= \mathrm{amgu}(v = w, sh) = \bigl\langle \{vwy, xy, yz\}, \{x, z\} \bigr\rangle, \\ sh_{1,2} &= \mathrm{amgu}(x = y, sh_1) = \bigl\langle \{vwxy, vwxyz, xyz, xy\}, \{z\} \bigr\rangle. \end{aligned}$$

Using the other ordering we have:

$$\begin{aligned} sh_2 &= \mathrm{amgu}(x = y, sh) = \bigl\langle \{vwxy, vwxyz, vxy, wxy, wxyz, xyz, xy\}, \{z\} \bigr\rangle, \\ sh_{2,1} &= \mathrm{amgu}(v = w, sh_2) = \bigl\langle \{vwxy, vwxyz, xyz, xy\}, \varnothing \bigr\rangle, \end{aligned}$$

therefore losing the linearity of z (which in turn could cause bigger precision losses later in the analysis).

Note that in both cases we need to compute the star-closures (even if in the first case the star-closure does not introduce new sharings). This means that even enhancing the known heuristics to "compute first the grounding binding, then the non star-closing bindings and only then the star-closing ones" would not be enough.

Work on this problem is still in progress. We are currently investigating the behavior of an even more enhanced heuristics that says "if the star-union has to be done, choose a binding $x = t$ that minimizes the number of linear variables involved". More formally, if sh is the current sharing-set and L is the set of linear variables, then, given a set of bindings, the quantity to be minimized is the cardinality of $\bigl(\bigcup \mathrm{rel}\bigl(\{x\} \cup \mathit{vars}(t), sh\bigr)\bigr) \cap L$ for each binding $x = t$ in this set.

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 $\mathit{Pos} \times \mathsf{Sharing}$ contains redundant elements, i.e., different pairs that characterize the same set of concrete computational states. The reduced product [13] between *Pos* and Sharing has been elegantly characterized in [8], 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 ways (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.

Suppose now that 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. Stated differently, all the variables in X share the same (possibly empty) set of variables. Thus, *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 $X \nsubseteq S$.

The few differences we observed in the experiments concerned only the number of independent variable pairs, even though, in principle, also the accuracy of both freeness and linearity can be affected. More precisely, for goal-independent analysis, one more independent pair of variables was discovered in the program `bmtp` both without and with explicit structural information. In the latter case, two more independent pairs were also obtained for the `sim` program. For goal-dependent analysis one more independent pair is detected in the program `knight` if explicit structural information is not part of the analysis domain, while no difference was observed in the analysis with 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 contrast, in this section, we describe one that is, to our knowledge, 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 self-bin-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 to avoid the self-bin-union of *both* the relevant components for x and t, and this independently from whether x and t may share or not. The disadvantage caused by this loss is twofold: CPU time is wasted by performing a costly operation and precision is degraded. In fact, in these cases the extra self-bin-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 a component consisting of the set of variables that are ground or free, thus adding an additional mode to the picture. These sets are populated by the join operation of the domain: if G_i, F_i, and S_i, for $i = 1, 2$, are the sets of ground, free, and 'ground or free' variables of two abstract descriptions to be joined, the set S of 'ground or free' variables in the join is given by $S = (G_1 \cup F_1 \cup S_1) \cap (G_2 \cup F_2 \cup S_2)$. The 'ground or free' property is then propagated in the abstract mgu operation the same way as freeness. 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 computationally complexity is concerned this extension is particularly promising, since the possibility of avoiding self-bin-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. Unfortunately, the results are rather discouraging. 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. In all the other cases no difference was observed on any benchmark program.

8 More Precise Exploitation of Linearity

In [20] 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 [6], which is the one usually followed, a set of definitely linear variables is recorded along with each sharing-set. The proposal in [20] gives rise to a domain that is quite different from the ones presented here. Since [20] 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 a piece of theoretical work presented in [20] can be exploited even in the more classical treatment of linearity. As far as we can tell, this fact has gone unnoticed up to now.

In [20], point 3 of Lemma 5 (which is reported to be proven in [19]) 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 applied even when using the standard domain for linearity-enhanced sharing analysis, i.e., Sharing plus *Lin*.

Let x be a linear variable and t be a non-linear term. Let $V_x = \{x\}$ and $V_t = \mathit{vars}(t)$. 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: y is linear, y occurs once in t, and 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 the following improved version of the amgu operator:[3]

$$\begin{aligned}\mathrm{amgu}(sh, x = t) = \overline{\mathrm{rel}}(V_{xt}, sh) & \\ \cup \operatorname{bin}\bigl(\mathrm{rel}(V_x, sh), \mathrm{rel}(V_t^{\mathrm{l}}, sh)\bigr) & \\ \cup \operatorname{bin}\bigl(\mathrm{rel}(V_x, sh)^\star, \mathrm{rel}(V_t^{\mathrm{nl}}, sh)\bigr). &\end{aligned}$$

Note that precision is improved because, thanks to the Lemma, the star-closure of rel_x is only combined with the non-linear part of rel_t.

The experimental evaluation of this new technique is quite disappointing: 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.

9 Tracking Compoundness

In [6, 7], Bruynooghe et al. considered the combination of sharing, freeness, and linearity with compoundness. Compoundness here means ‘non-variable’; that is, a variable is compound if it is bound to a term that is definitely not free. The authors represent sharing with the standard set-sharing domain, while freeness, linearity, and compoundness are each represented by the set of variables that definitely have the respective property.

As discussed in [6, 7], 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 [6, 7] where two improvements 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

[3]Even in this case, when using the non-redundant sharing domain SH^ρ star-union can be safely replaced by self-bin-union.

safely say that x and t cannot share. 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. The second improvement 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.[4] It should be noted that such a *Reduce* function is not part of the abstract unification algorithm presented in [6] and it is unclear whether and when it should be applied. The authors suggest that their algorithm should start from reduced abstract descriptions. One could then imagine that reduction is preserved by the algorithm, but theoretical results (or, for that matter, even simple claims) are not presented. It is our opinion that the second improvement proposed in [7] may well never apply because (1) the initial descriptions, computed by the abstraction function, are always reduced, and (2) it seems very unlikely that a well-designed compoundness analysis can lose this kind of information, which stems from bindings of the form $x = f(y, z)$.

10 Recovering Hidden Information

As noted by several authors (see, e.g., [6]) the standard combination of Sharing and *Free* is not optimal. G. Filé [15] formally identified the reduced product of these domains and proposed an improved abstract unification operator. This new operator exploits two properties holding for the abstract description of a *single* concrete substitution: (1) each free variable occurs in exactly one sharing group; and (2) two free variables occur in the same sharing group if and only if they are aliased (i.e., they have become the same variable). When considering the general case, where *sets* of concrete substitutions come into play, the first of the above observations allows to (partially) recover disjunctive information. An abstract description can thus be decomposed into a set of (maximal) descriptions that necessarily come from different computation paths, each one satisfying point (1) above. 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), some precision gains can be obtained. Also, by exploiting point (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.[5]

The experimental results we obtained for the first of the two ideas by G. Filé presented above are independent from the fact that structural information and/or

[4]The function *Reduce* also deals with some hidden interactions between sharing and freeness information: as these improvements are subsumed by the work of Filé [15], we discuss them in Section 10.

[5]This is possible even without decomposing the abstract description: as examples, consider the substitutions $\sigma_1 = \{x = f(u), y = g(v)\}$ and $\sigma_2 = \{x = f(y)\}$ together with an abstract description saying that x and y are both free and the only sharing group allowed is $\{xy\}$.

Pos is included or not in the combined domain used (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 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 described in this paper, this one is the most expensive in computational terms.

We note on 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 could be 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.

We plan to experiment with the 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.

11 Conclusion

In this paper, we have investigated enhanced sharing analysis techniques which have the potential for improving the precision of the sharing information over and above that obtainable using the classical combination of Sharing with *Lin* and *Free*. To do this, we have considered including other domains and using more powerful abstract semantic operators. We have evaluated, using the China analyzer, most of the proposals that have appeared in the literature together with a few ideas that, to the best of our knowledge, are new to this paper. For the combinations with *Pos* (for groundness) and Pattern($\cdot$) (for structural information), the results were positive, while, for the other cases, few improvements were observed.

The key issue is whether it is worthwhile including any particular enhancement in a sharing analyzer. The problem is that although the enhancement may add to the precision or even provide useful information in its own right, it may increase the cost, and possibly the complexity, of the computation. Thus, we believe, it is best to spend time developing the implementation of those enhancements that have real potential for achieving non-trivial improvements to the generated information together with manageable increases in the computational cost.

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, Corso Italia 40, I-56125 Pisa, Italy, March 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 (APPIA-GULP-PRODE'97)"*, pages 81–92, Grado, Italy, 1997.

[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] 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.

[7] 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, July 1994.

[8] M. Codish, H. Søndergaard, and P. J. Stuckey. Sharing and groundness dependencies in logic programs. Submitted for publication.

[9] A. Cortesi. On Pat(Sharing). Personal communication, February 1996.

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

[11] 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 (ICALP'92)*, volume 623 of *Lecture Notes in Computer Science*, pages 521–532, Wien, Austria, 1992. Springer-Verlag, Berlin.

[12] 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.

[13] 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 Fourth Annual ACM Symposium on Principles of Programming Languages*, pages 238–252, 1977.

[14] 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.

[15] G. Filé. Share $\times$ Free: Simple and correct. Technical Report 15, Dipartimento di Matematica, Università di Padova, December 1994.

[16] 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.

[17] 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*, MIT Press Series in Logic Programming, pages 154–165, Cleveland, Ohio, USA, 1989.

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

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

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

[21] 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.

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

[23] E. Zaffanella, R. Bagnara, and P. M. Hill. Widening set-sharing. In *Principles and Practice of Declarative Programming, Proceedings of the First International Conference*, Lecture Notes in Computer Science, Paris, France, September 1999. Springer-Verlag, Berlin. To appear.

[24] E. Zaffanella, P. M. Hill, and R. Bagnara. Decomposing non-redundant sharing by complementation. In *Static Analysis: Proceedings of the 6th International Symposium*, Lecture Notes in Computer Science, Venice, Italy, September 1999. Springer-Verlag, Berlin. To appear.