Sharing, Freeness, Linearity, Redundancy, Widenings, and Cliques

Francisco Bueno, Jorge Navas, Manuel Hermenegildo

TR CLIP5/2005.0
April 19, 2005

We discuss here different variants of the Sharing abstract domain, including the base domain that captures set-sharing, a variant to capture pair-sharing, in which redundant sharing groups (w.r.t. the pair-sharing property) can be eliminated, and an alternative representation based on cliques. The original proposal for using cliques in the non-redundant version of the domain is reviewed, then extended to the base domain. Variants of all the domains including freeness alone, and freeness together with linearity are also studied.

1 Preliminaries

Let V be a set of variables of interest (e.g., the variables of a program). Let $\wp^0(S)$ denote the *proper powerset* of set S, i.e., $\wp^0(S) = \wp(S) \setminus \{\emptyset\}$. A *sharing group* is a set of variables of interest representing their possible sharing. Let $SG = \wp^0(V)$ be the set of all sharing groups. A *sharing set* is a set of sharing groups. The Sharing domain is $SH = \wp(SG)$, the set of all sharing sets.

Let F be a set of ranked[1] functors of interest (e.g., the functors and predicates of a program). Let $Term$ be the set of terms constructed from F and V. Let $\hat{t}$ denote the set of variables of $t \in Term$. Let $[t]_x$ denote the number of occurrences of $x \in V$ within $t \in Term$. Let $solve(t_1 = t_2)$ denote the solved form of unification equation $t_1 = t_2$, $t_1 \in Term$, $t_2 \in Term$.

[1] That is, with a given arity.

1.1 Abstract Functions for Top-down Analysis

For abstract interpretation-based analysis in a top down framework there are three domain-dependent abstract functions which are essential: *call*2*entry*, *exit*2*succ*, and *extend*. The first two can be defined from the abstract unification operation *amgu*. The third one, however, has to be defined specifically for a given abstract domain.

Abstract unification Let an operation $amgu(x = t, ASub)$ of abstract unification for equation $x = t$, $x \in V$, $t \in Term$, and $ASub$ an abstract substitution (the domain of which contains variables $\hat{t} \cup \{x\}$). Abstract unification for equation $t_1 = t_2$, $t_1 \in Term$, $t_2 \in Term$, in the context of abstract substitution $ASub$ (the domain of which contains variables $\hat{t_1} \cup \hat{t_2}$) is given by $Amgu(solve(t_1 = t_2), ASub)$, where:

$$Amgu(Eq, ASub) = \begin{cases} ASub & \text{if } Eq = \emptyset \\ Amgu(Eq', amgu(x = t, ASub)) & \text{if } Eq = Eq' \cup \{x = t\} \end{cases}$$

Abstract functions Functions *call*2*entry* and *exit*2*succ* can be defined as follows.[2] Note that, in *call*2*entry*, $ASub$ is an abstract substitution with domain the variables of *Goal*; the result is an abstract substitution with domain the variables of $Head$. In *exit*2*succ* the domain of $ASub$ is the variables of $Head$, and that of the result the variables of *Goal*.

$$\begin{array}{lcl} call2entry(ASub, Goal, Head) & = & unify(ASub, Head, Goal) \\ exit2succ(ASub, Goal, Head) & = & unify(ASub, Goal, Head) \end{array}$$

where:

$$\begin{array}{l} unify(ASub, t_1, t_2) \quad = \\ \quad project(t_1, Amgu(solve(t_1 = t_2), augment(t_1, ASub))) \end{array}$$

whereas $extend(ASub_1, Goal, ASub_2)$ has to be defined in a way such that it yields a substitution for the success of *Goal* when it is called in a context represented by substitution $ASub_1$ on a set of variables which contains the variables of *Goal*, given that in such context the success of *Goal* is already represented by substitution $ASub_2$ on the variables of *Goal*. The domain of the resulting substitution is the same as the domain of $ASub_1$.

Thus, the functions that need be defined to complete an analysis are *amgu*, *extend*, *project*, and *augment*.

[2] But such definitions imply a possible loss of precision: see Section 2.3.

2 Sharing Domains

The Sharing domain was first presented in [**?**]. A complete set of abstract functions was defined in [**?**] (see Appendix A). The presentation here follows that of [**?**], since the notation used and the abstract unification operation obtained are rather intuitive.

Related sharing Let $t \in Term$ and $sh \in SH$, we denote by sh_t the sharing in sh related to t, defined as:

$$sh_t = rel(t, sh) = \{s \mid s \in sh, s \cap \hat{t} \neq \emptyset\}$$

i.e., the set of sets in sh which have non-empty intersection with the set of variables of t. By extension, in sh_{st} st acts as a single term. Also, $\overline{sh_t}$ is the complement of sh_t, i.e., $sh \setminus sh_t$.

Binary union Let $sh_1 \in SH$, $sh_2 \in SH$,

$$sh_1 \bowtie sh_2 = \{s_1 \cup s_2 \mid s_1 \in sh_1, s_2 \in sh_2\}$$

i.e., the result of applying union to each pair in their cartesian product.

Star union Let $sh \in SH$,

$$sh^* = \{s_1 \cup s_2 \cup \ldots \cup s_n \mid s_i \in sh, i = 1, \ldots, n\}$$

i.e., its closure under union.

Abstract unification Abstract unification for equation $x = t$, $x \in V$, $t \in Term$, and $sh \in SH$, is defined as:[3]

$$amgu(x = t, sh) = \overline{sh_{xt}} \cup (sh^*_x \bowtie sh^*_t)$$

Abstract functions Let $sh_1 \in SH$, $sh_2 \in SH$, and $g \in Term$ (a goal),

$$extend(sh_1, g, sh_2) = \overline{sh_{1g}} \cup \{\ s \mid s \in sh_{1g}^*,\ (s \cap \hat{g}) \in sh_2\ \}$$

$$project(g, sh_1) = \{s \cap \hat{g} \mid s \in sh_1\} \setminus \{\emptyset\}$$

$$augment(g, sh_2) = sh_2 \cup \{\{x\} \mid x \in \hat{g}\}$$

These functions were defined in [**?**] (although not all of them with that name, and maybe some were already in Langen's thesis). For all the domains discussed below, functions *project* and *augment* are the natural extension of the ones defined above.

[3] Note that $sh^*_t = (sh_t)^*$.

2.1 Sharing+Freeness

The presentation of this domain here follows that of [**?**]. The Sharing domain is augmented with a new component which tracks the variables which are free. The Sharing+Freeness domain is thus $SHF = SH \times V$.

Abstract unification Abstract unification for equation $x = t$, $x \in V$, $t \in Term$, and $(sh, f) \in SHF$, is given by $amgu^f(x = t, (sh, f)) = (sh', f')$, where:[4]

$$sh' = \begin{cases} \overline{sh_{xt}} \cup (sh_x \bowtie sh_t) & \text{if } x \in f \text{ or } t \in f \\ \overline{sh_{xt}} \cup (sh_x \bowtie sh_t^*) & \text{if } x \notin f,\ t \notin f, \text{ but } \hat{t} \subseteq f \text{ and } lin(t) \\ amgu(x = t, sh) & \text{otherwise} \end{cases}$$

and $lin(t)$ holds iff for all $y \in \hat{t}$: $[t]_y = 1$ and for all $z \in \hat{t}$ such that $y \neq z$, $sh_y \cap sh_z = \emptyset$;

$$f' = \begin{cases} f & \text{if } x \in f, t \in f \\ f \setminus (\cup sh_x) & \text{if } x \in f, t \notin f \\ f \setminus (\cup sh_t) & \text{if } x \notin f, t \in f \\ f \setminus (\cup (sh_x \cup sh_t)) & \text{if } x \notin f, t \notin f \end{cases}$$

Note that, for implementation, the direct definition of $lin(t)$ might be rather expensive: sh_y has to be calculated for every $y \in \hat{t}$ to check that each pairwise intersection is empty. Instead, an equivalent condition, which is more efficient, can be checked: for all $s \in sh_t$ $|s \cap \hat{t}| = 1$.

Abstract functions Function $extend^f((sh_1, f_1), g, (sh_2, f_2))$ for this domain was defined in [**?**] as given by (sh', f'), where:

$$sh' = extend(sh_1, g, sh_2)$$

$$f' = f_2 \cup \{x \mid x \in (f_1 \setminus \hat{g}), ((\cup sh'_x) \cap \hat{g}) \subseteq f_2\}$$

Functions $project^f$ and $augment^f$ are defined as follows:

$$project^f(g, (sh, f)) = (project(g, sh), f \cap \hat{g})$$

$$augment^f(g, (sh, f)) = (augment(g, sh), f \cup \hat{g})$$

[4]Note that t is not necessarily a variable: $t \in f$ means "t *is* a variable and is known to be free".

2.2 Sharing+Freeness+Linearity

The presentation of this domain here follows that of [**?**]. The Sharing+Freeness domain is augmented with a new component which tracks the variables which are linear. The Sharing+Freeness+Linearity domain is thus $SHL = SH \times V \times V$.

Abstract unification Abstract unification for equation $x = t$, $x \in V$, $t \in Term$, and $(sh, f, l) \in SHL$, is given by $amgu^l(x = t, (sh, f, l)) = (sh', f', l')$. Let $alin(t)$ iff $\hat{t} \subseteq l$ and $lin(t)$. Then:

$$sh' = \begin{cases} \overline{sh_{xt}} \cup sh'' & \text{if } x \in f \text{ or } t \in f \text{ or } alin(x) \text{ or } alin(t) \\ amgu(x = t, sh) & \text{otherwise} \end{cases}$$

$$sh'' = \begin{cases} sh_x \bowtie sh_t & \text{if } x \in f \text{ or } t \in f \\ (sh_x \cup (sh_x \bowtie sh^*_{xt})) \bowtie & \\ (sh_t \cup (sh_t \bowtie sh^*_{xt})) & \text{if } alin(x), alin(t) \\ sh^*_x \bowtie sh_t & \text{if } alin(x), \neg alin(t) \\ sh_x \bowtie sh^*_t & \text{if } \neg alin(x), alin(t) \end{cases}$$

$$l' = f' \cup \begin{cases} l \setminus (\cup sh_x \cap \cup sh_t) & \text{if } alin(x), alin(t) \\ l \setminus (\cup sh_x) & \text{if } alin(x), \neg alin(t) \\ l \setminus (\cup sh_t) & \text{if } \neg alin(x), alin(t) \\ l \setminus (\cup (sh_x \cup sh_t)) & \text{otherwise} \end{cases}$$

and f' is as in $amgu^f$.

Under construction

Abstract functions The idea here is that, having linearity, the *extend* function can be made more precise than without linearity. I think this function has not ever been defined...

Let $extend^l((sh_1, f_1, l_1), g, (sh_2, f_2, l_2)) = (sh', f', l')$, where:

$$\begin{array}{rcll} sh' & = & \overline{sh_{1g}} & \cup \ \cup\{\ \{s \mid s \in sh_{1g},\ s' \subseteq s\ \}^* \mid s' \in sh_2,\ s' \not\subseteq l_2\ \} \\ & & \cup & \{\ s \mid s \in sh_{1g},\ (s \cap \hat{g}) \in sh_2,\ (s \cap \hat{g}) \subseteq l_2\ \} \end{array}$$

Note that the *extend* function for the original Sharing domain is equivalent to the above expression *without* the subexpression $s \not\subseteq l_2$. This is precisely the gain in precision that having linearity information provides.

Comment: Check what is the role of the variables in $s \setminus \hat{g}$: is it relevant whether they are linear or not?

Functions $project^l$ and $augment^l$ are defined as follows:

$$project^l(g,(sh,f,l)) = (project(g,sh), f \cap \hat{g}, l \cap \hat{g})$$

$$augment^l(g,(sh,f,l)) = (augment(g,sh), f \cup \hat{g}, l \cup \hat{g})$$

2.3 Specific Abstract Functions for Unification

The abstract functions $call2entry$ and $exit2succ$ can be defined easily from $amgu$, as we have seen. However, by defining such functions specifically for each domain, precision can be improved in some cases. Function $amgu$ has the drawback that it is symmetric: all variables in the unification equations are treated uniformly. Contrary to this, the equation $Goal = Head$ which is used during analysis is not symmetric. When using $call2entry$, the variables of $Head$ are known to be new, i.e., free and unaliased; when using $exit2succ$, it is the variables of $Goal$ that can be considered new (since they are not in the domain of the exit abstract substitution). One can take advantage of this fact and improve precision w.r.t. the definition of these two functions based on $amgu$.

For the domains which already include freeness information, this is not an issue, since the new variables in each case are taken care by function $unify$ (the variables correspond to t_1 in the definition of $unify$). This function calls $augment$, and both $augment^f$ and $augment^l$ include such variables as free and unaliased. Thus, this information is already present during $Amgu$. This is not the case, however, for the base Sharing domain. One way to take advantage of the extra information is to define $unify$ in such a way that it uses $amgu^f$ or $amgu^l$ (which do exploit such information) and provide them with the necessary information on the new variables. For example, for the Sharing+Freeness domain, let $Amgu^f$ the version of $Amgu$ which uses $amgu^f$ (a similar construction could be done with $amgu^l$ for the Sharing+Freeness+Linearity domain); function $unify$ can be defined specifically for the Sharing domain, as follows:

$$unify(ASub, t_1, t_2) = project(t_1, ASub')$$

where:

$$(ASub', Free) = Amgu^f(solve(t_1 = t_2), augment^f(t_1, (ASub, \emptyset)))$$

so that abstract functions $augment^f$ and $amgu^f$ for the Sharing+Freeness domain are used, but the result "projected" onto the sharing component.

However, the use of $amgu^f$ (or $amgu^l$) implies the overhead of carrying around freeness information during abstract unification. This could be alleviated with the following alternative definition, specific for Sharing, to take advantage of new variables in abstract unification:

$$unify(ASub, t_1, t_2) \;=\; project(t_1, Aunify(solve(t_1 = t_2), \hat{t_1}, augment(t_1, ASub)))$$

$$Aunify(Eq, Lin, ASub) = \begin{cases} ASub & \text{if } Eq = \emptyset \\ Aunify(\; Eq', & \text{if } Eq = Eq' \cup \{x = t\} \\ \quad Lin \setminus (\{x\} \cup \hat{t}), & \\ \quad lamgu(x = t, Lin, ASub)\;) & \end{cases}$$

$$lamgu(x = t, Lin, sh) = \begin{cases} \overline{sh_{xt}} \cup (sh_x \bowtie sh_t) & \text{if } x \in Lin \text{ or } t \in Lin \\ \overline{sh_{xt}} \cup (sh_x \bowtie sh_t^*) & \text{if } x \notin Lin,\ t \notin Lin, \\ & \text{but } \hat{t} \subseteq Lin \text{ and } lin(t) \\ amgu(x = t, sh) & \text{otherwise} \end{cases}$$

Example. Consider goal $p(u, v, w)$ called with $\{uv, uw\}$, and unification with clause head $p(x, y, z)$. Analysis with $Amgu$ will yield $\{xy, xyz, xz\}$, whereas analysis with $Aunify$ will yield $\{xy, xz\}$.

Comment: Check applicability of this to bottom-up analyses.

The trade-offs involved in using this specific abstract unification function or the one based on $amgu^f$ (or even $amgu^l$) have never been investigated.

3 Non-Redundant Sharing Domains

The Sharing domains capture set-sharing: "whether there can be one or more run-time variables shared between a set of program variables". If instead the property of interest is pair-sharing: "whether there can be one or more run-time variables shared between two program variables", then the abstract operations can be simplified.

Note that any given sharing group with more than two variables, such as, e.g., xyz, conveys the same information as the set of all pairs of its variables, e.g., $\{xy, xz, yz\}$. Regarding pair-sharing, any of the two representations have the same information: there can be shared run-time variables between any pair of the program variables involved. Alternatively, one can say that if

$\{xy, xz, yz\}$ is a subset of the abstract substitution, then the sharing group xyz is *redundant* for pair-sharing information [**?**].

The Non-redundant version of Sharing is defined in [**?**]. The idea is to eliminate or avoid the occurrence within a sharing set of sharing groups which are redundant w.r.t. the pair-sharing that the sharing set represents.

Self binary union Let $sh \in SH$, its self-binary union is $s_1^{\times} = s_1 \bowtie s_1$.

Abstract unification Abstract unification for equation $x = t$, $x \in V$, $t \in Term$, and $sh \in SH$, is defined as:[5] v

$$amgu^{\rho}(x = t, sh) = \overline{sh_{xt}} \cup (sh_x^{\times} \bowtie sh_t^{\times})$$

i.e., substituting self-binary union for the star union.

Abstract functions Functions *project* and *augment* for the Sharing domain are also correct for this domain. An abstract function $extend^{\rho}$ for the non-redundant version of the Sharing domain has never been defined. However, the function for the original Sharing domain should serve. (Although it can probably be improved on efficiency by using self-binary union instead of star union).

3.1 Non-Redundant Sharing+Freeness

The inclusion of freeness into the Non-redundant Sharing domain is mentioned in [**?**]. However, no abstract functions seem to have been defined. We present them here by modifying those of Sharing along the lines suggested by the "non-redundancy" idea of [**?**]. Basically, star union is replaced everywhere by self-binary union.

Abstract unification Abstract unification for equation $x = t$, $x \in V$, $t \in Term$, and $(sh, f) \in SHF$, is given by (sh', f'), where:

$$sh' = \begin{cases} \overline{sh_{xt}} \cup (sh_x \bowtie sh_t) & \text{if } x \in f \text{ or } t \in f \\ \overline{sh_{xt}} \cup (sh_x \bowtie sh_t^{\times}) & \text{if } x \notin f,\ t \notin f, \text{ but } \hat{t} \subseteq f \text{ and } lin(t) \\ amgu^{\rho}(x = t, sh) & \text{otherwise} \end{cases}$$

and f' is as in $amgu^{f}$.

[5] Note that $sh_t^{\times} = (sh_t)^{\times}$.

Abstract functions Functions $project^f$ and $augment^f$ are correct for this domain. An *extend* abstract function for the non-redundant version of the Sharing+Freeness domain has never been defined. However, the function $extend^f$ for the original Sharing+Freeness domain should serve.

3.2 Non-Redundant Sharing+Freeness+Linearity

Under construction

Under construction

4 Non-Redundant Clique-Sharing Domains

This domain is defined in [**?**]. The idea is to eliminate or avoid the occurrence within sharing sets of sharing groups which are the powerset of some set of variables. Such sets of variables are called *cliques* and are carried along within the sharing representation in a separate component.

The Clique-Sharing domain is $SH^W = \{(cl, sh) \mid cl \in SH, sh \in SH\}$, i.e., the set of pairs of a clique set (a set of cliques) and a sharing set. Note that clique sets are sharing sets, although they represent sharing in a different manner than sharing sets. To distinguish them we will write $cl \in CL$ and $sh \in SH$ for any pair $(cl, sh) \in SH^W$.

Self binary union Let $(cl, sh) \in SH^W$,

$$(cl, sh)^\times = cl^\times \cup (cl \bowtie sh)$$

is the extension of self-binary union to SH^W.

Non-related sharing Let $t \in Term$ and $cl \in CL$,

$$\overline{rel}(t, cl) = \{\ c \setminus \hat{t} \mid c \in cl\ \} \setminus \{\emptyset\}$$

For two terms s and t we will write $\overline{rel}(st, cl)$, using st as a single term.

Abstract unification Abstract unification for equation $x = t$, $x \in V$, $t \in Term$, and $(cl, sh) \in SH^W$ is given by:

$$\begin{array}{ll} amgu^W(x = t, (cl, sh)) & = \\ (\ & \overline{rel}(xt, cl) \cup \\ & ((cl_x, sh_x)^\times \bowtie (cl_t, sh_t)^\times) \cup ((cl_x, sh_x)^\times \bowtie sh_t^\times) \cup (sh_x^\times \bowtie (cl_t, sh_t)^\times) \\ , & \overline{sh_{xt}} \cup (sh_x^\times \bowtie sh_t^\times) \quad) \end{array}$$

This abstract unification operation is defined after the one presented in Definition 11 of [**?**], and is equivalent.

Abstract functions Abstract functions for this domain have never been defined. However, the functions for the Clique-Sharing domain below should serve.

4.1 Non-Redundant Clique-Sharing+Freeness

Under construction

Under construction

4.2 Non-Redundant Clique-Sharing+Freeness+Linearity

Under construction

Under construction

5 Clique-Sharing Domains

Abstract unification $amgu^W$ is correct for the version of Sharing which is non-redundant w.r.t. pair-sharing [**?**]. For the original Sharing domain, the natural counterpart of $amgu^W$ obtained by replacing self-binary union by star union should serve. Correctness results for the operations on this domain are included in Appendix D.

Star union Let $(cl, sh) \in SH^W$,

$$(cl, sh)^* = cl^* \cup (cl^* \bowtie sh^*)$$

Note that $^* : SH^W \longrightarrow CL$ is not an operator (its image is not SH^W): it operates on a pair of a clique set and a sharing set but returns a clique set, not another pair.

Abstract unification Abstract unification for equation $x = t$, $x \in V$, $t \in Term$, and $(cl, sh) \in SH^W$ is given by:

$$\begin{array}{ll} amgu^s(x = t, (cl, sh)) & = \\ (\quad \overline{rel}(xt, cl) \cup & \\ \quad \underline{((cl_x, sh_x)^* \bowtie (cl_t, sh_t)^*) \cup ((cl_x, sh_x)^* \bowtie sh_t^*) \cup (sh_x^* \bowtie (cl_t, sh_t)^*)} & \\ , \quad \overline{sh_{xt}} \cup (sh_x^* \bowtie sh_t^*) \quad) & \end{array}$$

This abstract unification operation is defined after the one in the previous section, replacing self-binary union by star union.

Abstract functions Let $g \in Term$, $(cl, sh) \in SH^W$. Functions $project^s$ and $augment^s$ are defined as follows:

$$project^s(g, (cl, sh)) = (project(g, cl), project(g, sh))$$

$$augment^s(g, (cl, sh)) = (cl, augment(g, sh))$$

Function $extend^s(Call, g, Prime)$ is defined as follows. Let $Call = (cl_1, sh_1)$ and $Prime = (cl_2, sh_2)$. Let *normalize* be a function which normalizes a pair (cl, sh) so that no powersets occur in sh (all are "transferred" to cliques in cl). Let $Prime$ be already normalized, and:

$$(cl', sh') = normalize(((cl_{1g}, sh_{1g})^*, sh_{1g}^*))$$

The following two functions lift the classical extend to the case of clique-sets and sharing-sets occurring in the pairs of $Call$ and $Prime$:

$$extsh(sh_1, g, sh_2) = \overline{sh_{1g}} \cup \{\ s \mid s \in sh',\ (s \cap g) \in sh_2\ \}$$

$$extcl(cl_1, g, cl_2) = \overline{rel}(\hat{g}, cl_1) \cup \{\ (s' \cap s) \cup (s' \setminus \hat{g}) \mid s' \in cl',\ s \in cl_2\ \}$$

The following two functions account for the cases of the clique-set of $Call$ and the sharing-set of $Prime$, and the other way around:

$$clsh(cl', g, sh_2) = \{\ s \mid s \subseteq c \in cl',\ (s \cap g) \in sh_2\ \}$$

$$shcl(sh', g, cl_2) = \{\ s \mid s \in sh',\ (s \cap g) \subseteq c \in cl_2\ \}$$

The function extend for Sharing-clique is thus:

$$\begin{array}{ll} extend^s((cl_1, sh_1), g, (cl_2, sh_2)) & = \\ (\quad extcl(cl_1, g, cl_2) & \\ ,\quad extsh(sh_1, g, sh_2) \cup clsh(cl', g, sh_2) \cup shcl(sh', g, cl_2) &) \end{array}$$

5.1 Clique-Sharing+Freeness

This domain has not been presented previously. The Clique-Sharing domain is augmented with a new component which tracks the variables which are free. The Clique-Sharing+Freeness domain is thus $SHF^W = SH^W \times V$. Correctness results for the operations on this domain are included in Appendix E.

Abstract unification Abstract unification for equation $x = t$, $x \in V$, $t \in Term$, and $(clsh, f) \in SHF^W$, $clsh = (cl, sh)$, is given by $amgu^{sf}(x = t, (clsh, f)) = (clsh', f')$, where:

$$clsh' = \begin{cases} amgu^{sff}(x = t, clsh) & \text{if } x \in f \text{ or } t \in f \\ amgu^{sfl}(x = t, clsh) & \text{if } x \notin f,\ t \notin f \text{ but } \hat{t} \subseteq f \text{ and } lin^s(t) \\ amgu^{s}(x = t, clsh) & \text{otherwise} \end{cases}$$

and $lin^s(t)$ holds iff for all $y \in \hat{t}$: $[t]_y = 1$ and for all $z \in \hat{t}$ such that $y \neq z$, $sh_y \cap sh_z = \emptyset$ and $cl_y \cap cl_z = \emptyset$;

$$\begin{aligned} amgu^{sff}(x = t, (cl, sh)) = \quad & (\quad \overline{rel}(xt, cl) \cup \\ & \quad\ \ ((cl_x \cup sh_x) \bowtie cl_t) \cup (cl_x \bowtie sh_t) \\ & \quad , \quad \overline{sh_{xt}} \cup (sh_x \bowtie sh_t) \qquad\qquad) \end{aligned}$$

$$\begin{aligned} amgu^{sfl}(x = t, (cl, sh)) = \quad & (\quad \overline{rel}(xt, cl) \cup \\ & \quad\ \ ((cl_x \cup sh_x) \bowtie (cl_t, sh_t)^*) \cup (cl_x \bowtie sh_t^*) \\ & \quad , \quad \overline{sh_{xt}} \cup (sh_x \bowtie sh_t^*) \qquad\qquad) \end{aligned}$$

$$f' = \begin{cases} f & \text{if } x \in f, t \in f \\ f \setminus (\cup(sh_x \cup cl_x)) & \text{if } x \in f, t \notin f \\ f \setminus (\cup(sh_t \cup cl_t)) & \text{if } x \notin f, t \in f \\ f \setminus (\cup(sh_x \cup cl_x \cup sh_t \cup cl_t) & \text{if } x \notin f, t \notin f \end{cases}$$

Note again that checking emptyness of each pairwise intersection in the definition of $lin^s(t)$ (as in $lin(t)$) can be reduced to a more efficient equivalent condition: for all $s \in sh_t$ and all $s \in cl_t$, $|s \cap \hat{t}| = 1$.

Abstract functions Function $extend^{sf}$ for this domain is given by $extend^{sf}((clsh_1, f_1), g, (clsh_2, f_2)) = ((cl', sh'), f')$, where:

$$(cl', sh') = extend^s(clsh_1, g, clsh_2)$$

$$f' = f_2 \cup \{x \mid x \in (f_1 \setminus \hat{g}),\ ((\cup(sh'_x \cup cl'_x)) \cap \hat{g}) \subseteq f_2\}$$

Functions $project^{sf}$ and $augment^{sf}$ are defined as follows:

$$project^{sf}(g, (clsh, f)) = (project^s(g, clsh), f \cap \hat{g})$$

$$augment^{sf}(g, (clsh, f)) = (augment^s(g, clsh), f \cup \hat{g})$$

5.2 Clique-Sharing+Freeness+Linearity

Under construction

Under construction

6 Detecting cliques

Obviously, to minimize the representation in SH^W, it pays off to replace any set S of sharing groups which is the proper powerset of some set of variables C by including C as a clique. Once this is done, the set S can be eliminated from the sharing set, since the presence of C in the clique set makes S redundant. This is the normalization mentioned in Section 5 when defining *extend* for the Clique-Sharing domain, and denoted there by a function *normalize*. In this section we present an algorithm for such a normalization.

Given an element $(cl, sh) \in SH^W$, sharing groups might occur in sh which are already implicit in cl. Such groups are redundant with respect to the sharing represented by the pair. We say that an element $(cl, sh) \in SH^W$ is *minimal* if $\Downarrow cl \cap sh = \emptyset$. An algorithm for minimization is straightforward: it should delete from sh all sharing groups which are a subset of an existing clique in cl. But normalization goes a step further by "moving sharing" from the sharing set of a pair to the clique set, thus forcing redundancy of some sharing groups.

While normalizing, it turns out that powersets may exist which can be obtained from sharing groups in the sharing set plus sharing groups implied by existing cliques in the clique set. The representation can be minimized further if such sharing groups are also "tranferred" to the clique set by adding the adequate clique. We say that an element $(cl, sh) \in SH^W$ is *normalized* if whenever there is an $s \subseteq (\Downarrow cl \cup sh)$ such that $s = \downarrow c$ for some set c then $s \cap sh = \emptyset$.

Our normalization algorithm is presented in Figure 1. It starts with an element $(cl, sh) \in SH^W$, which is already minimal, and obtains an equivalent element (w.r.t. the sharing represented) which is normalized. First, the number m is computed, which is the length of the longest possible clique. Then the sharing set sh is traversed and candidate cliques of that length obtained. Existing subsets of a candidate clique S are extracted from sh. If there are $2^i - 1 - [S]$ subsets of S in sh then S is a clique: it is added to cl and its subsets deleted from sh. Note that the test is performed on the number of existing subsets, and requires the computation of a number $[S]$, which is crucial for the correctness of the test.

The number $[S]$ corresponds to the number of subsets of S which may

1. Let $n = |sh|$; if $n < 3$, stop.
2. Compute the maximum m such that $n \geq 2^m - 1$.
3. Let $i = m$.
4. If $i = 1$, stop.
5. Let $C = \{s \mid s \in sh, |s| = i\}$.
6. If $C = \emptyset$ then decrement i and go to 4.
7. Take $S \in C$ and delete it from C.
8. Let $SS = \{s \mid s \in sh, s \subseteq S\}$.
9. Compute $[S]$.
10. If $|SS| = 2^i - 1 - [S]$ then:
    (a) Add S to cl (regularize cl).
    (b) Subtract SS from sh.
11. Go to 6.

Figure 1: Algorithm for detecting cliques

not appear in sh because they are already represented in cl (i.e., they are already subsets of an existing clique). In order to correctly compute this number it is essential that the input to the algorithm is already minimal; otherwise, redundant sharing groups might bias the calculation: the formula below may count as not present in sh a (redundant) group which is in fact present. The computation of $[S]$ is as follows. Let $I = \{S \cap C \mid C \in cl\} \setminus \{\emptyset\}$ and $A_i = \{\cap A \mid A \subseteq I,\ |A| = i\}$. Then:

$$[S] = \sum_{1 \leq i \leq |I|} (-1)^{i-1} \sum_{A \in A_i} (2^{|A|} - 1)$$

Note that the representation can be minimized further by eliminating cliques which are redundant with other cliques. This is the regularization mentioned in step 10 of the algorithm. We say that a clique set cl is *regular* if there are no two cliques $c_1 \in cl$, $c_2 \in cl$, such that $c_1 \subset c_2$. This can be tested while adding cliques in step 10 above.

Finally, there is a chance for further minimization by considering as cliques candidate sets of variables such that not all of their subsets exist in the given element of SH^W. This opens up the possibility of using the above

algorithm as a widening. Note that the algorithm preserves precision, since the sharing represented by the element of SH^W input to the algorithm is the same as that represented by the element which is output. However, we can set up a threshold for the number of subsets of the candidate clique that need be detected, and in this case the output element may in general represent more sharing. This might, nonetheless, be worth for practical purposes. We have experimented which such widenings by imposing thresholds which are a percentage of the number of subsets to be detected. Given such a percentage p, the test in step 10 above would check for $|SS| + [S] \geq (2^i - 1)(p/100)$, instead. The results of our experiments are reported in Appendix G.

A Abstract functions for Sharing

Function *call2entry* for the Sharing domain was defined in [**?**] as follows.

Let $S \subseteq V$, $R \subseteq V \times V$, and R^+ the symmetric and transitive closure of relation R.[6]

$$partition(S, R) = \{B \mid B \subseteq S, (x \in B \ \& \ y \in B) \Leftrightarrow (x, y) \in R^+\}$$

Let $t \in Term$ be of the form $p(t_1, \ldots, t_n)$, $S \subseteq V$, and $sh \in SH$,

$$pos(t, S) = \{i \mid S \cap \hat{t_i} \neq \emptyset\}$$

$$\mathcal{P}(t, sh) = \{pos(t, S) \mid S \in sh\}$$

Let $g \in Term$ and $h \in Term$ be a goal and the head of a clause for the same predicate, such that they are unifiable, and $sh \in SH$ an abstract substitution for $\hat{g}$. The result of $call2entry(sh, g, h)$ is an abstract substitution for $\hat{h}$ given by:

$$call2entry(sh, g, h) = \{S \mid S \in \beta, pos(h, S) \in (\mathcal{P}(g, sh))^*\}$$

where:

$$\beta = \bigcup_{S \in P} \wp^{\emptyset}(S)$$

$$P = partition(\hat{h} \setminus G, DG)$$

$$DG = \{(x_i, x_j) \mid x_i \in S, x_j \in S, S \in sh, i \neq j\} \cup \{(x, y) \mid (x \rightarrow S) \in Eq, y \in S\}$$

$$G = \{x \mid (x \rightarrow \emptyset) \in Eq\}$$

$$Eq = propagate(normalize(solve(g = h)) \cup \{x \rightarrow \emptyset \mid x \in (\hat{g} \setminus \cup sh)\})$$

and:

$$normalize(E) = \{x \rightarrow \hat{t} \mid (x = t) \in E, x \in V, t \in Term\}$$

$$propagate(E) = \begin{cases} \{x \rightarrow \emptyset\} \ \cup \ propagate(ground(E', x)) \\ \qquad \text{if } E = E' \cup \{x \rightarrow \emptyset\} \\ E \qquad \text{otherwise} \end{cases}$$

$$ground(E, x) = \{y \rightarrow S \setminus \{x\} \mid (y \rightarrow S) \in E, y \neq x\} \ \cup \\ \{y \rightarrow \emptyset \mid (x \rightarrow S) \in E, y \in S\}$$

[6] If R is understood as an undirected graph, then $(x, y) \in R^+$ and $(y, x) \in R^+$ iff there is a path between x and y in R.

B Correctness Results for Basic Operations

For any set s, we will denote $\downarrow s = \wp^0(s)$. For a set of sets ss, we define $\Downarrow ss = \cup\{\downarrow s \mid s \in ss\}$.

Let $c \subseteq V$ be a clique. Note that $\downarrow c$ denotes all the sharing that is implicitely represented in c. The sharing represented by a clique set $cl \in CL$ is thus $\Downarrow cl$. The sharing represented by an element $(cl, sh) \in SH^W$ is then $\Downarrow cl \cup sh$.

Lemma 1 *Let s_1 and s_2 be sets:*

$$(\downarrow s_1)^* = \downarrow s_1 \tag{1}$$

Powerset and union do not commute:

$$\downarrow(s_1 \cup s_2) \supseteq \downarrow s_1 \cup \downarrow s_2 \tag{2}$$

but not the other way around.

Lemma 2 *For every $cl \in CL$:*

$$\downarrow \cup cl \supseteq cl^* \tag{3}$$

$$\downarrow \cup cl \supseteq \Downarrow cl \tag{4}$$

but not the other way around.

> **Proof** (4) is immediate from (2).
>
> For (3): for every $s \in cl^*$, there is $\{s_1, \ldots, s_n\} \subseteq cl$, $n \geq 1$, so that $s = (\cup_{i=1}^n s_i) \neq \emptyset$. But $(\cup_{i=1}^n s_i) \subseteq \cup cl$, so that $s \in \downarrow \cup cl$. However, cl might not have singleton sets, which are thus not in cl^*, but they are indeed subsets of $\cup cl$.

Star-union is correct and precise for clique sets (and can be avoided, replacing it by set union):

Lemma 3 *For every $cl \in CL$:*[7] *vv*

$$\Downarrow cl^* = \downarrow \cup cl \tag{5}$$

$$\Downarrow cl^* = (\Downarrow cl)^* \tag{6}$$

[7] Note that $\Downarrow cl^* = \Downarrow (cl^*)$.

Proof First note that:

$$\cup \Downarrow cl = \cup cl^* = \cup cl \tag{7}$$

and, if $cl \neq \emptyset$:

$$\cup cl \in cl^* \tag{8}$$

Thus, $\Downarrow cl^* \subseteq \downarrow \cup cl$, since, using (4) and (7), we have that: $\Downarrow cl^* \subseteq \downarrow \cup cl^* = \downarrow \cup cl$.

Also, $\downarrow \cup cl \subseteq \Downarrow cl^*$. To see this, take $s \in \downarrow \cup cl$, we also have that (8) $\cup cl \in cl^*$ (if $cl \neq \emptyset$), so that, from the definition, $s \in \Downarrow cl^*$. If $cl = \emptyset$, the result follows directly.

Now, $\Downarrow cl^* \subseteq (\Downarrow cl)^*$. Take $s \in \Downarrow cl^*$, so that $s \in \downarrow C$, $C \in cl^*$. Then $s \subseteq C$ and there are $\{s_1, \ldots, s_n\} \subseteq cl$, $n \geq 1$, such that $C = \cup_{i=1}^{n} s_i$. Then there is an $I \subseteq \{1, \ldots, n\}$, $I \neq \emptyset$, such that there are, for all $j \in I$, $c_j \subseteq s_j$, and $s = \overset{i \in I}{\cup} c_i$. Therefore, for all $i \in I$, $c_i \in \downarrow s_i$ and $s_i \in cl$, so that $c_i \in \Downarrow cl$. Thus, $(\overset{i \in I}{\cup} c_i) \in (\Downarrow cl)^*$. Since $s = \overset{i \in I}{\cup} c_i$, then $s \in (\Downarrow cl)^*$.

Finally, using (4) with (18), then (1) and (5), we also have $(\Downarrow cl)^* \subseteq (\downarrow \cup cl)^* = \downarrow \cup cl = \Downarrow cl^*$.

Binary union is correct (but not precise) for clique sets:

Proposition 1 *For every $cl_1 \in CL$, $cl_2 \in CL$:*

$$\Downarrow (cl_1 \bowtie cl_2) \supseteq \Downarrow cl_1 \bowtie \Downarrow cl_2 \tag{9}$$

but not in general the other way around.

Proof If either $cl_1 = \emptyset$ or $cl_2 = \emptyset$ ther result is straightforward. In other case, it is a direct corollary of Lemma 4.

Lemma 4 *For every $cl_1 \in CL$, $cl_2 \in CL$, such that $cl_1 \neq \emptyset$ and $cl_2 \neq \emptyset$:*

$$\Downarrow (cl_1 \bowtie cl_2) = \Downarrow cl_1 \cup (\Downarrow cl_1 \bowtie \Downarrow cl_2) \cup \Downarrow cl_2 \tag{10}$$

Proof Note that for every set of sets ss, $s \in \Downarrow ss$ iff there is a $c \in ss$ such that $s \in \downarrow c$. In other words, $s \subseteq c \in ss$ and $s \neq \emptyset$.

We first prove that $\Downarrow (cl_1 \bowtie cl_2) \subseteq \Downarrow cl_1 \cup (\Downarrow cl_1 \bowtie \Downarrow cl_2) \cup \Downarrow cl_2$. Take $s \in \Downarrow (cl_1 \bowtie cl_2)$. Then $s \neq \emptyset$ and $s \subseteq c \in (cl_1 \bowtie cl_2)$. That

is, $c = c_1 \cup c_2$, with $c_i \in cl_i$, $i = 1, 2$. Therefore, either $s \subseteq c_i$ for $i = 1$ or for $i = 2$ (or both), or $s \cap c_i \neq \emptyset$ for both $i = 1$ and $i = 2$.

Consider first $s \subseteq c_i$ for some $i = 1, 2$. Then we have that $s \subseteq c_i \in cl_i$ and $s \neq \emptyset$, so that $s \in \Downarrow cl_i$. Consider now $s \cap c_i \neq \emptyset$ for both $i = 1, 2$. Then $s = s_1 \cup s_2$, $s_i \subseteq c_i$, $s \cap s_i \neq \emptyset$. Thus, $s_i \subseteq c_i \in cl_i$ and $s_i \neq \emptyset$, so that $s_i \in \Downarrow cl_i$. Therefore, $s \in (\Downarrow cl_1 \otimes \Downarrow cl_2)$. In any case, $s \in \Downarrow cl_1 \cup (\Downarrow cl_1 \otimes \Downarrow cl_2) \cup \Downarrow cl_2$.

We now prove that $\Downarrow cl_1 \cup (\Downarrow cl_1 \otimes \Downarrow cl_2) \cup \Downarrow cl_2 \subseteq \Downarrow (cl_1 \otimes cl_2)$. Take $s \in \Downarrow cl_1 \cup (\Downarrow cl_1 \otimes \Downarrow cl_2) \cup \Downarrow cl_2$. Then, either $s \in (\Downarrow cl_1 \otimes \Downarrow cl_2)$ or $s \in \Downarrow cl_i$ for $i = 1$ or $i = 2$ (or both).

Consider first $s \in (\Downarrow cl_1 \otimes \Downarrow cl_2)$. Then $s = s_1 \cup s_2$, $s_i \subseteq \Downarrow cl_i$, so that $s_i \subseteq c_i \in cl_i$, $s_i \neq \emptyset$. Thus, $s_1 \cup s_2 = s \subseteq (c_1 \cup c_2) \in (cl_1 \otimes cl_2)$ and $s \neq \emptyset$, so that $s \in \Downarrow (cl_1 \otimes cl_2)$. Consider now $s \in \Downarrow cl_i$ for $i = 1$ or $i = 2$. Then $s \subseteq c_i \in cl_i$ and $s \neq \emptyset$. Therefore, $s \subseteq (c_1 \cup c_2) \in (cl_1 \otimes cl_2)$, so that $s \in \Downarrow (cl_1 \otimes cl_2)$.

Projection is correct for clique sets, but imprecise:

Lemma 5 *For every $cl \in CL$ and term t:*

$$\Downarrow cl_t \supseteq (\Downarrow cl)_t \tag{11}$$

but not the other way around.

Proof Let $s \in (\Downarrow cl)_t$. Then $s \cap \hat{t} \neq \emptyset$ and there is $C \in cl$ such that $s \in \downarrow C$. Thus, $s \subseteq C$, so that $C \cap \hat{t} \neq \emptyset$. Therefore, $C \in cl_t$, but also $s \in \downarrow C$, so that $s \in \Downarrow cl_t$.

To see that the other direction does not hold in general, take $cl = \{xy\}$ and $t = x$. We have $\Downarrow cl = \{x, xy, y\}$ and $cl_t = cl = \{xy\}$, so that $(\Downarrow cl)_t = \{x, xy\}$ but $\Downarrow cl_t = \{x, xy, y\}$.

C Optimized Unification for Clique-Sharing

Some basic results which are proved somewhere else:

Lemma 6 *Let ss_1, ss_2, and ss_3 be sets of sets:*

$$(ss_1 \otimes ss_2)^* = ss_1^* \otimes ss_2^* \tag{12}$$

$$(ss_1 \cup ss_2)^* \supseteq ss_1^* \cup ss_2^* \tag{13}$$

$$(ss_1 \cup \{\emptyset\})^* = ss_1^* \cup \{\emptyset\} \tag{14}$$

$$ss_1 \otimes (ss_2 \cup ss_3) = (ss_1 \otimes ss_2) \cup (ss_1 \otimes ss_3) \tag{15}$$

If both $ss_1 \neq \emptyset$ and $ss_2 \neq \emptyset$ then:

$$\cup(ss_1 \otimes ss_2) = \cup(ss_1 \cup ss_2) \tag{16}$$

Computing $(cl, sh)^*$ can be reduced to simple union of sets:

Lemma 7 *Given $(cl, sh) \in SH^W$, we have that, if $cl \neq \emptyset$:*

$$\Downarrow(cl, sh)^* = \downarrow\cup(cl \cup sh)$$

Proof We first prove the following auxiliary result:

$$(cl, sh)^* = (cl \otimes (sh \cup \{\emptyset\}))^* \tag{17}$$

$$\begin{array}{rcccl} (cl, sh)^* & = & cl^* \cup (cl^* \otimes sh^*) & \overset{(15)}{=} & cl^* \otimes (sh^* \cup \{\emptyset\}) \\ & \overset{(14)}{=} & cl^* \otimes (sh \cup \{\emptyset\})^* & \overset{(12)}{=} & (cl \otimes (sh \cup \{\emptyset\}))^* \end{array}$$

Now, since $cl \neq \emptyset$ and also $sh \cup \{\emptyset\} \neq \emptyset$, we can apply (16), so that:

$$\begin{array}{rcccl} \Downarrow(cl, sh)^* & \overset{(17)}{=} & \Downarrow(cl \otimes (sh \cup \{\emptyset\}))^* & \overset{(5)}{=} & \downarrow\cup(cl \otimes (sh \cup \{\emptyset\})) \\ & \overset{(16)}{=} & \downarrow\cup(cl \cup sh \cup \{\emptyset\}) & = & \downarrow\cup(cl \cup sh) \end{array}$$

Also, the basic expression for unification in sharing can be reduced, for clique sets, to union of sets:

Lemma 8 *Given $cl_1 \in CL$, $cl_2 \in CL$, we have that, if $cl_1 \neq \emptyset$ and $cl_2 \neq \emptyset$:*

$$\Downarrow(cl_1^* \otimes cl_2^*) = \downarrow\cup(cl_1 \cup cl_2)$$

Proof Note that (16) can be applied below because $cl_1 \neq \emptyset$ and $cl_2 \neq \emptyset$. Thus:

$$\Downarrow(cl_1^* \bowtie cl_2^*) \overset{(12)}{=} \Downarrow(cl_1 \bowtie cl_2)^* \overset{(5)}{=} \downarrow\cup(cl_1 \bowtie cl_2) \overset{(16)}{=} \downarrow\cup(cl_1 \cup cl_2)$$

With this, we can redefine abstract unification without loss of precision (and correctness) into an optimized operation $amgu^o$, which can be used in place of $amgu^s$.

$$amgu^o(x = t, (cl, sh)) = \begin{cases} (\ cl\ ,\ \overline{sh_{xt}} \cup (sh_x^* \bowtie sh_t^*)\ ) & \text{if } cl_x = cl_t = \emptyset \\ (\ \overline{rel}(xt, cl)\ ,\ \overline{sh_{xt}}\ ) & \text{if } cl_x = sh_x = \emptyset \\ & \text{or } cl_t = sh_t = \emptyset \\ (\ \overline{rel}(xt, cl) \cup \{\cup(cl_x \cup cl_t \cup sh_x \cup sh_t)\}\ ,\ \overline{sh_{xt}}\ ) & \\ & \text{otherwise} \end{cases}$$

Theorem 1 *Given $x = t$, $x \in V$, $t \in Term$, and $(cl, sh) \in SH^W$. Let $amgu^o(x = t, (cl, sh)) = (cl', sh')$ and $amgu^s(x = t, (cl, sh)) = (cl'', sh'')$. Then:*

$$\Downarrow cl' \cup sh' = \Downarrow cl'' \cup sh''$$

Proof Let $clsh = ((cl_x, sh_x)^* \bowtie (cl_t, sh_t)^*) \cup ((cl_x, sh_x)^* \bowtie sh_t^*) \cup (sh_x^* \bowtie (cl_t, sh_t)^*)$. Recall that:

$$cl'' = \overline{rel}(xt, cl) \cup clsh \quad \text{and} \quad sh'' = \overline{sh_{xt}} \cup (sh_x^* \bowtie sh_t^*)$$

For cl' and sh' we have three cases:

- $cl_x = cl_t = \emptyset$
  Since $cl_x = cl_t = \emptyset$ then $\overline{rel}(xt, cl) = cl$ and $(cl_x, sh_x)^* = (cl_t, sh_t)^* = \emptyset$, so that $(cl_x, sh_x)^* \bowtie (cl_t, sh_t)^* = (cl_x, sh_x)^* \bowtie sh_t^* = sh_x^* \bowtie (cl_t, sh_t)^* = \emptyset$. Then $clsh = \emptyset$, so that $cl'' = cl$. Thus, what we have to prove is:

  $$\Downarrow cl' \cup sh' = \Downarrow cl \cup sh''$$

  vvwhich follows because, in this case, $cl' = cl$ and $sh' = \overline{sh_{xt}} \cup (sh_x^* \bowtie sh_t^*) = sh''$.
- $cl_x = sh_x = \emptyset$ or $cl_t = sh_t = \emptyset$
  Take $cl_x = sh_x = \emptyset$. Then also $(cl_x, sh_x)^* = sh_x{}^* = \emptyset$. Thus, we have $(cl_x, sh_x)^* \bowtie (cl_t, sh_t)^* = (cl_x, sh_x)^* \bowtie sh_t^* =$

$sh_x^* \bowtie (cl_t, sh_t)^* = \emptyset$, so that $clsh = \emptyset$. Also, $sh_x^* \bowtie sh_t^* = \emptyset$, so that $sh'' = \overline{sh_{xt}}$. The same reasoning applies to the case $cl_t = sh_t = \emptyset$. Thus, what we have to prove is:

$$\Downarrow cl' \cup sh' = \Downarrow \overline{rel}(xt, cl) \cup \overline{sh_{xt}}$$

which follows because, in this case, $cl' = \overline{rel}(xt, cl)$ and $sh' = \overline{sh_{xt}}$.

- any other case

  Let $W = \cup(cl_x \cup cl_t \cup sh_x \cup sh_t)$. We now have:

  $$\Downarrow cl'' = \Downarrow(\ \overline{rel}(xt, cl) \cup clsh\ ) = \Downarrow \overline{rel}(xt, cl) \cup \Downarrow clsh$$

  $$\begin{aligned} \Downarrow cl' &= \Downarrow(\overline{rel}(xt, cl) \cup \{W\}) = \Downarrow \overline{rel}(xt, cl) \cup \Downarrow \{W\} \\ &= \Downarrow \overline{rel}(xt, cl) \cup \downarrow W \end{aligned}$$

  and $sh' = \overline{sh_{xt}}$, so that what we have to prove is:

  $$\Downarrow \overline{rel}(xt, cl) \cup \downarrow W \cup \overline{sh_{xt}} = \Downarrow \overline{rel}(xt, cl) \cup \Downarrow clsh \cup \overline{sh_{xt}} \cup (sh_x^* \bowtie sh_t^*)$$

  or, equivalently:

  $$\downarrow W = \Downarrow clsh \cup (sh_x^* \bowtie sh_t^*)$$

  which follows because $\downarrow W = \Downarrow clsh$ and $(sh_x^* \bowtie sh_t^*) \subseteq \downarrow W$, which we proceed to prove.

  First, note that $sh_x^* \bowtie sh_t^*$ is a set of sets of variables from $\cup(sh_x \cup sh_t)$, and $\cup(sh_x \cup sh_t) \subseteq W$. Therefore, $sh_x^* \bowtie sh_t^*$ is a subset of $\downarrow W$.

  Second, we show that $\downarrow W = \Downarrow clsh$. We proceed by cases. The third case of $amgu^o$ excludes the other two, so that now we only have the following three possible cases:

  - $cl_x \neq \emptyset$ and $cl_t \neq \emptyset$
  - $cl_x = \emptyset$ (but $sh_x \neq \emptyset$) and $cl_t \neq \emptyset$
  - $cl_x \neq \emptyset$ and $cl_t = \emptyset$ (but $sh_t \neq \emptyset$)

- $cl_x \neq \emptyset$, $cl_t = \emptyset$, $sh_t \neq \emptyset$

  Since $cl_t = \emptyset$ also $(cl_t, sh_t)^* = \emptyset$, so $clsh = (cl_x, sh_x)^* \bowtie sh_t^*$. Then, from (17), $clsh = (cl_x \bowtie (sh_x \cup \{\emptyset\}))^* \bowtie sh_t^*$. Since $cl_x \neq \emptyset$, $sh_t \neq \emptyset$, and also $sh_x \cup \{\emptyset\} \neq \emptyset$, we can apply Lemma 8 and equation (16), so that:

$$\begin{array}{rcl} \Downarrow clsh & \stackrel{\text{L.8}}{=} & \downarrow\cup((cl_x \bowtie (sh_x \cup \{\emptyset\})) \cup sh_t) \\ & = & \downarrow(\cup(cl_x \bowtie (sh_x \cup \{\emptyset\})) \cup \cup sh_t) \\ & \stackrel{(16)}{=} & \downarrow(\cup cl_x \cup \cup(sh_x \cup \{\emptyset\}) \cup \cup sh_t) \\ & = & \downarrow\cup(cl_x \cup sh_x \cup sh_t) \end{array}$$

But, since $cl_t = \emptyset$, we have that $W = \cup(cl_x \cup sh_x \cup sh_t)$, so that $\Downarrow clsh = \downarrow W$.

- $cl_x = \emptyset$, $cl_t \neq \emptyset$, $sh_x \neq \emptyset$
  This case is symmetric to the previous one.
- $cl_x \neq \emptyset$ and $cl_t \neq \emptyset$
  Now we have to prove that:

$$\begin{array}{rcl} \downarrow W & = & \Downarrow clsh \\ & = & \Downarrow((cl_x, sh_x)^* \bowtie (cl_t, sh_t)^*) \\ & & \cup \Downarrow((cl_x, sh_x)^* \bowtie sh_t^*) \cup \Downarrow(sh_x^* \bowtie (cl_t, sh_t)^*) \end{array}$$

But, since $\Downarrow((cl_x, sh_x)^* \bowtie sh_t^*)$ is a set of sets of variables from $\cup(cl_x \cup sh_x \cup sh_t)$, and $\cup(cl_x \cup sh_x \cup sh_t) \subseteq W$, then $\Downarrow((cl_x, sh_x)^* \bowtie sh_t^*)$ is a subset of $\downarrow W$. The same happens for $\Downarrow(sh_x^* \bowtie (cl_t, sh_t)^*)$. Therefore, it suffices to prove that $\downarrow W = \Downarrow((cl_x, sh_x)^* \bowtie (cl_t, sh_t)^*)$.
Since $cl_x \neq \emptyset$, $cl_t \neq \emptyset$, and also $sh_x \cup \{\emptyset\} \neq \emptyset$, $sh_t \cup \{\emptyset\} \neq \emptyset$, Lemma 8 and equation (16) can be applied as follows:

$$\begin{array}{rcl} & & \Downarrow((cl_x, sh_x)^* \bowtie (cl_t, sh_t)^*) \\ & \stackrel{(17)}{=} & \Downarrow((cl_x \bowtie (sh_x \cup \{\emptyset\}))^* \bowtie (cl_t \bowtie (sh_t \cup \{\emptyset\}))^*) \\ & \stackrel{\text{L.8}}{=} & \downarrow\cup((cl_x \bowtie (sh_x \cup \{\emptyset\})) \cup (cl_t \bowtie (sh_t \cup \{\emptyset\}))) \\ & = & \downarrow(\cup(cl_x \bowtie (sh_x \cup \{\emptyset\})) \cup \cup(cl_t \bowtie (sh_t \cup \{\emptyset\}))) \\ & \stackrel{(16)}{=} & \downarrow(\cup(cl_x \cup (sh_x \cup \{\emptyset\})) \cup \cup(cl_t \cup (sh_t \cup \{\emptyset\}))) \\ & = & \downarrow\cup(cl_x \cup sh_x \cup cl_t \cup sh_t) \\ & = & \downarrow W \end{array}$$

D Correctness Results for Clique-Sharing

Some basic results which are proved somewhere else:

Lemma 9 *Let ss_1, ss_2, ss_3, and ss_4 be sets of sets. If $ss_1 \subseteq ss_3$ and $ss_2 \subseteq ss_4$ then:*

$$ss_1^* \subseteq ss_3^* \tag{18}$$

$$ss_1 \bowtie ss_2 \subseteq ss_3 \bowtie ss_4 \tag{19}$$

$$ss_1^* \bowtie ss_2^* \subseteq ss_3^* \bowtie ss_4^* \tag{20}$$

The following result is supposedly (?) proved in [**?**]. The operation of non-related sharing for clique sets is correct:

Lemma 10 *Given $cl \in CL$, we have that:*

$$\Downarrow\overline{rel}(xt, cl) = \overline{(\Downarrow cl)_{xt}}$$

v The extension of star-union to SH^W is correct but imprecise:

Lemma 11 *Given $(cl, sh) \in SH^W$, we have that:*

$$\Downarrow(cl, sh)^* \cup sh^* \supseteq (\Downarrow cl \cup sh)^*$$

but not the other way around.

> **Proof** First, we consider $cl = \emptyset$. In this case, $\Downarrow(cl, sh)^* = \emptyset$ and $\Downarrow cl = \emptyset$, so that $\Downarrow(cl, sh)^* \cup sh^* = sh^* = (\Downarrow cl \cup sh)^*$.
>
> If $cl \neq \emptyset$ then, by Lemma 7, $\Downarrow(cl, sh)^* = \downarrow\cup(cl \cup sh)$. This is the proper powerset of the set of variables $\cup(cl \cup sh)$, and therefore it is a superset of any other set of sets of variables from $\cup(cl \cup sh)$, such as, for example, $(\Downarrow cl \cup sh)^*$.
>
> To see that the other direction does not hold in general, take $cl = \{xy\}$ and $sh = \{yz\}$. We have that $\Downarrow(cl, sh)^* = \downarrow(xyz)$ and $\Downarrow cl = \{x, xy, y\}$. Thus, $(\Downarrow cl \cup sh)^* = \{x, xy, y, yz\}^* = \{x, xy, xyz, y, yz\}$, which is a proper subset of $\downarrow(xyz)$.

Note Although imprecise in general, $(cl, sh)^*$ is in fact precise when $cl = \emptyset$ or $sh = \emptyset$. When $cl = \emptyset$ we have $(cl, sh)^* = \emptyset$, what makes this operation unnecessary (which is precisely the observation behind the first and second cases of $amgu^o$). When $sh = \emptyset$ we have $\Downarrow(cl, sh)^* = \Downarrow cl^*$ and $(\Downarrow cl \cup sh)^* = (\Downarrow cl)^*$, but $\Downarrow cl^* = (\Downarrow cl)^*$, from (6).

Abstract unification for Clique-Sharing is correct (but not precise):

Theorem 2 *Let* $(cl, ss) \in SH^W$, $sh \in SH$, *equation* $x = t$, $x \in V$ *and* $t \in Term$, *and* $amgu^s(x = t, (cl, ss)) = (cl^o, ss^o)$. *If* $\Downarrow cl \cup ss \supseteq sh$ *then:*

$$\Downarrow cl^o \cup ss^o \supseteq amgu(x = t, sh)$$

Proof We first prove the following instrumental results:

$$\Downarrow cl_t \cup ss_t \supseteq sh_t \tag{21}$$

since

$$\Downarrow cl \cup ss \supseteq sh \Rightarrow (\Downarrow cl \cup ss)_t \supseteq sh_t \Rightarrow (\Downarrow cl)_t \cup ss_t \supseteq sh_t$$
$$\stackrel{(11)}{\Rightarrow} \Downarrow cl_t \cup ss_t \supseteq sh_t$$

Also:

$$\overline{(\Downarrow cl)_{xt}} \cup \overline{ss_{xt}} \supseteq \overline{sh_{xt}} \tag{22}$$

since

$$\Downarrow cl \cup ss \supseteq sh \Rightarrow \overline{(\Downarrow cl \cup ss)_{xt}} \supseteq \overline{sh_{xt}} \Rightarrow \overline{(\Downarrow cl)_{xt}} \cup \overline{ss_{xt}} \supseteq \overline{sh_{xt}}$$

From (21) we have $\Downarrow cl_t \cup ss_t \supseteq sh_t$ and the same also for x: $\Downarrow cl_x \cup ss_x \supseteq sh_x$. Thus, by (20),

$$(\Downarrow cl_x \cup ss_x)^* \bowtie (\Downarrow cl_t \cup ss_t)^* \supseteq sh_x^* \bowtie sh_t^* \tag{23}$$

By Lemma 10, $\Downarrow\overline{rel}(xt, cl) = \overline{(\Downarrow cl)_{xt}}$, and from (22),

$$\Downarrow\overline{rel}(xt, cl) \cup \overline{ss_{xt}} \supseteq \overline{sh_{xt}} \tag{24}$$

Now, recall that $amgu(x = t, sh) = \overline{sh_{xt}} \cup (sh_x^* \bowtie sh_t^*)$. We will use $amgu^o$ instead of $amgu^s$, since by Theorem 1 they are equivalent. So, we have three cases:

- $cl_x = cl_t = \emptyset$
  In this case, $cl^o = cl$ and $ss^o = \overline{ss_{xt}} \cup (ss_x^* \bowtie ss_t^*)$. So, what we have to prove is:

  $$\Downarrow cl \cup \overline{ss_{xt}} \cup (ss_x^* \bowtie ss_t^*) \supseteq \overline{sh_{xt}} \cup (sh_x^* \bowtie sh_t^*)$$

  We first show that $ss_x^* \bowtie ss_t^* \supseteq sh_x^* \bowtie sh_t^*$. This follows from (23), since $cl_x = cl_t = \emptyset$.
  We now show that $\Downarrow cl \cup \overline{ss_{xt}} \supseteq \overline{sh_{xt}}$. This follows from (22), since $\overline{(\Downarrow cl)_{xt}} \subseteq \Downarrow cl$.

- $cl_x = ss_x = \emptyset$ or $cl_t = ss_t = \emptyset$
  In this case, $cl^o = \overline{rel}(xt, cl)$ and $ss^o = \overline{ss_{xt}}$. Also, we have that either $\Downarrow cl_x \cup ss_x = \emptyset$ or $\Downarrow cl_t \cup ss_t = \emptyset$. Thus, from (21), either $sh_x = \emptyset$ or $sh_t = \emptyset$. In any case, $sh_x^* \bowtie sh_t^* = \emptyset$. So, what we have to prove is:

  $$\Downarrow \overline{rel}(xt, cl) \cup \overline{ss_{xt}} \supseteq \overline{sh_{xt}}$$

  but this is precisely (24), proved above.
- any other case
  Now we have $cl^o = \overline{rel}(xt, cl) \cup \{\cup(cl_x \cup cl_t \cup ss_x \cup ss_t)\}$ and $ss^o = \overline{ss_{xt}}$. Let $W = \cup(cl_x \cup cl_t \cup ss_x \cup ss_t)$. We have that $\Downarrow(\overline{rel}(xt, cl) \cup \{W\}) = \Downarrow \overline{rel}(xt, cl) \cup \Downarrow\{W\} = \Downarrow \overline{rel}(xt, cl) \cup \downarrow W$. So, what we have to prove is:

  $$\Downarrow \overline{rel}(xt, cl) \cup \downarrow W \cup \overline{ss_{xt}} \supseteq \overline{sh_{xt}} \cup (sh_x^* \bowtie sh_t^*)$$

  First, we have that $\Downarrow \overline{rel}(xt, cl) \cup \overline{ss_{xt}} \supseteq \overline{sh_{xt}}$ (24).
  Second, we show that $\downarrow W \supseteq sh_x^* \bowtie sh_t^*$. Let $S = (\Downarrow cl_x \cup ss_x)^* \bowtie (\Downarrow cl_t \cup ss_t)^*$; note that S is a set of sets of variables from W, thus it is a subset of the proper powerset $\downarrow W$. From (23) we have that $S \supseteq sh_x^* \bowtie sh_t^*$, so that $\downarrow W \supseteq sh_x^* \bowtie sh_t^*$.

The previous result holds even for the case in which $\Downarrow cl \cup ss = sh$. That is, $amgu^s$ is neccessarily imprecise.

Proposition 2 *Let $(cl, ss) \in SH^W$, $sh \in SH$, equation $x = t$, $x \in V$ and $t \in Term$, and $amgu^s(x = t, (cl, ss)) = (cl^o, ss^o)$. If $\Downarrow cl \cup ss = sh$ then:*

$$\Downarrow cl^o \cup ss^o \supseteq amgu(x = t, sh)$$

but not in general $\Downarrow cl^o \cup ss^o = amgu(x = t, sh)$.

Proof The general statement is a direct corollary of Theorem 2. To see that equality does not hold in general, take $(cl, ss) = (\{xy\}, \emptyset)$ and $sh = \{x, xy, y\}$. We have $\Downarrow cl \cup ss = sh$. Take also $t = y$. Then $(cl^o, ss^o) = (\{xy\}, \emptyset)$, so that $\Downarrow cl^o \cup ss^o = \{x, xy, y\}$. But $amgu(x = t, sh) = \overline{sh_{xt}} \cup (sh_x^* \bowtie sh_t^*) = \{xy\}$, which is a proper subset of $\Downarrow cl^o \cup ss^o$.

Note Loss of precision occurs only in the third case of $amgu^o$. If $\Downarrow cl \cup ss = sh$ then equations (22) and (24) can be shown to be equalities, so that equality can also be shown to hold for the second case of $amgu^o$. In the first case, equations (21) and (23) turn also into equalities, so that equality also holds for the first case of $amgu^o$. Only the third case is imprecise.

Function *extend* for Clique-Sharing is correct (but not precise):

Theorem 3 *Let* $Call = (cl_1, ss_1) \in SH^W$ *and* $Prime = (cl_2, ss_2) \in SH^W$, *such that the conditions for the extend function hold,* $g \in Term$, *and* $extend^s(Call, g, Prime) = (cl', ss')$. *If* $\Downarrow cl_1 \cup ss_1 \supseteq sh_1$ *and* $\Downarrow cl_2 \cup ss_2 \supseteq sh_2$ *then:*

$$\Downarrow cl' \cup ss' \supseteq extend(sh_1, g, sh_2)$$

Proof Pending proof

E Correctness Results for Clique-Sharing+Freeness

Some basic results which are proved somewhere else:

Lemma 12 *Let ss_1, ss_2, ss_3, and ss_4 be sets of sets. If $ss_1 \supseteq ss_3$ and $ss_2 \supseteq ss_4$ then:*

$$\cup ss_1 \supseteq \cup ss_3 \tag{25}$$

$$ss_1 \cup ss_2 \supseteq ss_3 \cup ss_4 \tag{26}$$

Abstract unification for Clique-Sharing+Freeness is correct (but not precise):

Theorem 4 *Let $((cl, ss), f) \in SHF^W$, $(sh, e) \in SHF$, and equation $x = t$, $x \in V$, $t \in Term$. Let also $amgu^{sf}(x = t, ((cl, ss), f)) = ((cl^o, ss^o), f^o)$ and $amgu^f(x = t, (sh, e)) = (sh', f')$. If $\Downarrow cl \cup ss \supseteq sh$ and $f \subseteq e$ then:*

$$\Downarrow cl^o \cup ss^o \supseteq sh' \quad \textit{and} \quad f^o \subseteq f'$$

Proof First, we prove that $\Downarrow cl^o \cup ss^o \supseteq sh'$. From the definition of $amgu^{sf}$ we have three cases (plus two subcases of one of them):

- $x \in f$ or $t \in f$
  In this case, since $f \subseteq e$, we have $x \in e$ or $t \in e$, so that $sh' = \overline{sh_{xt}} \cup (sh_x \bowtie sh_t)$. Also, $ss^o = \overline{ss_{xt}} \cup (ss_x \bowtie ss_t)$ and $cl^o = \overline{rel}(xt, cl) \cup ((cl_x \cup ss_x) \bowtie cl_t) \cup (cl_x \bowtie ss_t)$. Thus, what we have to prove is:

  $$\begin{array}{c} \Downarrow(\ \overline{rel}(xt, cl) \cup ((cl_x \cup ss_x) \bowtie cl_t) \cup (cl_x \bowtie ss_t)\ ) \\ \cup \overline{ss_{xt}} \cup (ss_x \bowtie ss_t) \quad \supseteq \quad \overline{sh_{xt}} \cup (sh_x \bowtie sh_t) \end{array}$$

  that is:

  $$\begin{array}{c} \Downarrow\overline{rel}(xt, cl) \cup\ \Downarrow((cl_x \cup ss_x) \bowtie cl_t) \cup\ \Downarrow(cl_x \bowtie ss_t) \\ \cup \overline{ss_{xt}} \cup (ss_x \bowtie ss_t) \quad \supseteq \quad \overline{sh_{xt}} \cup (sh_x \bowtie sh_t) \end{array}$$

  or, equivalently, since $\Downarrow\overline{rel}(xt, cl) \cup \overline{ss_{xt}} \supseteq \overline{sh_{xt}}$ (24):

  $$\Downarrow((cl_x \cup ss_x) \bowtie cl_t) \cup\ \Downarrow(cl_x \bowtie ss_t) \cup (ss_x \bowtie ss_t) \supseteq sh_x \bowtie sh_t$$

  This is proved in Lemma 13 below.

- $x \notin f$, $t \notin f$, but $\hat{t} \subseteq f$ and $lin^s(t)$
  In this case, $ss^o = \overline{ss_{xt}} \cup (ss_x \otimes ss_t^*)$ and $cl^o = \overline{rel}(xt, cl) \cup ((cl_x \cup ss_x) \otimes (cl_t, ss_t)^*) \cup (cl_x \otimes ss_t^*)$, so that:

  $$\Downarrow cl^o = \Downarrow\overline{rel}(xt, cl) \cup \Downarrow((cl_x \cup ss_x) \otimes (cl_t, ss_t)^*) \cup \Downarrow(cl_x \otimes ss_t^*)$$

  Also, we may have that $x \in e$ or $t \in e$ or none. However, $\hat{t} \subseteq e$, since $\hat{t} \subseteq f$ and $f \subseteq e$. We also have that

  $$lin^s(t) \Rightarrow lin(t)$$

  To see this... Proof pending
  Thus, we have now two cases: either (1) $x \in e$ or $t \in e$, or (2) $x \notin e$, $t \notin e$, but $\hat{t} \subseteq e$ and $lin(t)$. We proceed with them.
- $x \notin f$, $t \notin f$, $\hat{t} \subseteq f$, $lin^s(t)$, $x \notin e$, $t \notin e$, $\hat{t} \subseteq e$, $lin(t)$
  Now, $sh' = \overline{sh_{xt}} \cup (sh_x \otimes sh_t^*)$, so that what we have to prove is:

  $$\begin{array}{l} \Downarrow\overline{rel}(xt, cl) \cup \Downarrow((cl_x \cup ss_x) \otimes (cl_t, ss_t)^*) \cup \Downarrow(cl_x \otimes ss_t^*) \\ \quad \cup\overline{ss_{xt}} \cup (ss_x \otimes ss_t^*) \quad \supseteq \quad \overline{sh_{xt}} \cup (sh_x \otimes sh_t^*) \end{array}$$

  or, equivalently, since $\Downarrow\overline{rel}(xt, cl) \cup \overline{ss_{xt}} \supseteq \overline{sh_{xt}}$ (24):

  $$\begin{array}{l} \Downarrow((cl_x \cup ss_x) \otimes (cl_t, ss_t)^*) \cup \Downarrow(cl_x \otimes ss_t^*) \cup (ss_x \otimes ss_t^*) \\ \quad \supseteq \quad sh_x \otimes sh_t^* \end{array}$$

  This is proved in Lemma 14 below.
- $x \notin f$, $t \notin f$, $\hat{t} \subseteq f$, $lin^s(t)$, $x \in e$ or $t \in e$
  Now, $sh' = \overline{sh_{xt}} \cup (sh_x \otimes sh_t)$, so that what we have to prove is:

  $$\begin{array}{l} \Downarrow\overline{rel}(xt, cl) \cup \Downarrow((cl_x \cup ss_x) \otimes (cl_t, ss_t)^*) \cup \Downarrow(cl_x \otimes ss_t^*) \\ \quad \cup\overline{ss_{xt}} \cup (ss_x \otimes ss_t^*) \quad \supseteq \quad \overline{sh_{xt}} \cup (sh_x \otimes sh_t) \end{array}$$

  or, equivalently, since $\Downarrow\overline{rel}(xt, cl) \cup \overline{ss_{xt}} \supseteq \overline{sh_{xt}}$ (24):

  $$\begin{array}{l} \Downarrow((cl_x \cup ss_x) \otimes (cl_t, ss_t)^*) \cup \Downarrow(cl_x \otimes ss_t^*) \cup (ss_x \otimes ss_t^*) \\ \quad \supseteq \quad sh_x \otimes sh_t \end{array}$$

  But this follows from the previous case, since $sh_t^* \supseteq sh_t$ and thus, from (19), $sh_x \otimes sh_t^* \supseteq sh_x \otimes sh_t$.

- any other case
  We now have that $sh' = amgu(x = t, sh)$ and that $(cl^o, ss^o) = amgu^s(x = t, (cl, ss))$. Thus, the result follows directly from Theorem 2.

Now we prove that $f^o \subseteq f'$. From the definition of amg^{sf} we have four cases. Note that in every case $f^o \subseteq f$. Also:

$$\cup(cl_t \cup ss_t) \supseteq \cup sh_t \tag{27}$$

To see this, note that $\cup(cl_t \cup ss_t) = \cup(\Downarrow cl_t \cup ss_t)$, since both expressions are made of the same set of variables. But, from (21), $\Downarrow cl_t \cup ss_t \supseteq sh_t$, so that from (25) the result follows.

- $x \in f$ and $t \in f$
  In this case, since $f \subseteq e$, we have $x \in e$ and $t \in e$, so that $f' = e$. Also, $f^o = f$. Thus, the result is straightforward.
- $x \notin f$ and $t \in f$
  Now, we have $t \in e$, but either $x \in e$ or $x \notin e$. If $x \in e$, we have $f' = e$. Thus, the result is straightforward, since $f^o \subseteq f$ and $f \subseteq e$.
  If $x \notin e$, we have $f' = e \setminus \cup sh_t$. Also, $f^o = f \setminus \cup(ss_t \cup cl_t)$, so that what we have to prove is:

  $$f \setminus \cup(ss_t \cup cl_t) \subseteq e \setminus \cup sh_t$$

  which holds because $f \subseteq e$, and $\cup(ss_t \cup cl_t) \supseteq \cup sh_t$ (27).
- $x \in f$ and $t \notin f$
  This case is symmetric to the previous one, with x for t and vice versa.
- $x \notin f$ and $t \notin f$
  In this case, $f^o = f \setminus \cup(cl_x \cup cl_t \cup ss_x \cup cl_x)$, but we may or may not have $x \in e$ and $t \in e$, so we have four more cases.
- $x \notin f$, $t \notin f$, $x \notin e$, and $t \notin e$
  We now have $f' = f \setminus \cup(sh_x \cup sh_t)$. Thus what we have to prove is:

  $$f \setminus \cup(cl_x \cup cl_t \cup ss_x \cup ss_t) \subseteq e \setminus \cup(sh_x \cup sh_t)$$

which holds because $f \subseteq e$ and also, from (26):

$$\cup(cl_x \cup cl_t \cup ss_x \cup ss_t) \supseteq \cup(sh_x \cup sh_t)$$

since we have $\cup(cl_t \cup ss_t) \supseteq \cup sh_t$ (27) and the same for x: $\cup(cl_x \cup ss_t) \supseteq \cup sh_x$.

- $x \notin f$, $t \notin f$, $x \in e$, and $t \notin e$
  In this case, $f' = f \setminus \cup sh_x$. The result then follows from the previous case, since $\cup sh_x \subseteq \cup(sh_x \cup sh_t)$.
- $x \notin f$, $t \notin f$, $x \notin e$, and $t \in e$
  In this case, $f' = f \setminus \cup sh_t$. As before, the result follows because $\cup sh_t \subseteq \cup(sh_x \cup sh_t)$.
- $x \notin f$, $t \notin f$, $x \in e$, and $t \in e$
  Now, $f' = e$, and the result follows because $f^o \subseteq f$ and $f \subseteq e = f'$.

Lemma 13 *Under the same conditions of Theorem 4:*

$$\Downarrow((cl_x \cup ss_x) \otimes cl_t) \cup \Downarrow(cl_x \otimes ss_t) \cup (ss_x \otimes ss_t) \supseteq sh_x \otimes sh_t$$

Proof We first prove the following instrumental result:

$$(\Downarrow cl_x \cup ss_x) \otimes (\Downarrow cl_t \cup ss_t) \supseteq sh_x \otimes sh_t \tag{28}$$

From (21) we have $\Downarrow cl_t \cup ss_t \supseteq sh_t$ and the same holds also for x: $\Downarrow cl_x \cup ss_x \supseteq sh_x$. Thus the result follows from (19).

To see that the main statement of the lemma holds, consider that $\Downarrow ss \supseteq ss$. Then:

$$\begin{array}{ll}
 & \Downarrow((cl_x \cup ss_x) \otimes cl_t) \cup \Downarrow(cl_x \otimes ss_t) \cup (ss_x \otimes ss_t) \\
? & (\Downarrow(cl_x \cup ss_x) \otimes \Downarrow cl_t) \cup (\Downarrow cl_x \otimes \Downarrow ss_t) \cup (ss_x \otimes ss_t) \\
= & ((\Downarrow cl_x \cup \Downarrow ss_x) \otimes \Downarrow cl_t) \cup (\Downarrow cl_x \otimes \Downarrow ss_t) \cup (ss_x \otimes ss_t) \\
\overset{(19)}{\supseteq} & ((\Downarrow cl_x \cup ss_x) \otimes \Downarrow cl_t) \cup (\Downarrow cl_x \otimes ss_t) \cup (ss_x \otimes ss_t) \\
\overset{(15)}{=} & ((\Downarrow cl_x \cup ss_x) \otimes \Downarrow cl_t) \cup ((\Downarrow cl_x \cup ss_x) \otimes ss_t) \\
\overset{(15)}{=} & (\Downarrow cl_x \cup ss_x) \otimes (\Downarrow cl_t \cup ss_t) \\
\overset{(28)}{\supseteq} & sh_x \otimes sh_t
\end{array}$$

Lemma 14 *Under the same conditions of Theorem 4:*

$$\Downarrow((cl_x \cup ss_x) \otimes (cl_t, ss_t)^*) \cup \Downarrow(cl_x \otimes ss_t^*) \cup (ss_x \otimes ss_t^*) \supseteq sh_x \otimes sh_t^*$$

Proof Proof pending

$$
\begin{array}{ll}
& \Downarrow((cl_x \cup ss_x) \bowtie (cl_t, ss_t)^*) \cup \Downarrow(cl_x \bowtie ss_t^*) \cup (ss_x \bowtie ss_t^*) \\
= & \Downarrow((cl_x \cup ss_x) \bowtie (cl_t^* \cup (cl_t^* \bowtie ss_t^*))) \cup \Downarrow(cl_x \bowtie ss_t^*) \cup (ss_x \bowtie ss_t^*) \\
? & (\Downarrow(cl_x \cup ss_x) \bowtie \Downarrow(cl_t^* \cup (cl_t^* \bowtie ss_t^*))) \cup (\Downarrow cl_x \bowtie \Downarrow ss_t^*) \cup (ss_x \bowtie ss_t^*) \\
= & ((\Downarrow cl_x \cup \Downarrow ss_x) \bowtie (\Downarrow cl_t^* \cup \Downarrow(cl_t^* \bowtie ss_t^*))) \cup (\Downarrow cl_x \bowtie \Downarrow ss_t^*) \cup (ss_x \bowtie ss_t^*) \\
? & ((\Downarrow cl_x \cup \Downarrow ss_x) \bowtie (\Downarrow cl_t^* \cup (\Downarrow cl_t^* \bowtie \Downarrow ss_t^*))) \cup (\Downarrow cl_x \bowtie \Downarrow ss_t^*) \cup (ss_x \bowtie ss_t^*) \\
\overset{(19)}{\supseteq} & ((\Downarrow cl_x \cup ss_x) \bowtie (\Downarrow cl_t^* \cup (\Downarrow cl_t^* \bowtie ss_t^*))) \cup (\Downarrow cl_x \bowtie ss_t^*) \cup (ss_x \bowtie ss_t^*) \\
\overset{(15)}{=} & ((\Downarrow cl_x \cup ss_x) \bowtie (\Downarrow cl_t^* \cup (\Downarrow cl_t^* \bowtie ss_t^*))) \cup ((\Downarrow cl_x \cup ss_x) \bowtie ss_t^*) \\
\overset{(15)}{=} & (\Downarrow cl_x \cup ss_x) \bowtie (\Downarrow cl_t^* \cup (\Downarrow cl_t^* \bowtie ss_t^*) \cup ss_t^*) \\
\overset{(19)}{\supseteq} & (\Downarrow cl_x \cup ss_x) \bowtie (\Downarrow cl_t^* \cup ss_t^*) \\
\overset{(21)+(19)}{\supseteq} & sh_x \bowtie (\Downarrow cl_t^* \cup ss_t^*) \\
\supseteq ? & sh_x \bowtie sh_t
\end{array}
$$

Function *extend* for Clique-Sharing+Freeness is correct (but not precise):

Theorem 5 *Let* $Call = ((cl_1, ss_1), e_1) \in SHF^W$ *and* $Prime = ((cl_2, ss_2), e_2) \in SHF^W$*, such that the conditions for the extend function hold,* $g \in Term$*,* $extend^{sf}(Call, g, Prime) = ((cl', ss'), e')$*, and* $extend^f((sh_1, f_1), g, (sh_2, f_2)) = (sh', f')$*. If* $\Downarrow cl_1 \cup ss_1 \supseteq sh_1$*,* $e_1 \subseteq f_1$*,* $\Downarrow cl_2 \cup ss_2 \supseteq sh_2$*, and* $e_2 \subseteq f_2$ *then:*

$$\Downarrow cl' \cup ss' \supseteq sh' \quad \textit{and} \quad e' \subseteq f'$$

Proof Pending proof

F Precision and Efficiency Results for the Clique-Sharing Domains

We have measured experimentally the relative efficiency and precision obtained with the inclusion of cliques in the Sharing and Sharing+Freeness domains. We measure absolute precision of a sharing set by the number of its sharing groups relative to the number of sharing groups in the worst-case for the set of variables in its domain. The number of sharing groups in the worst-case sharing for n variables is given by $2^n - 1$. Thus, precision of $sh \in SH$ is given by $|sh|/(2^n - 1)$. This is a number in $[0, 1]$ such that sh is more precise the closer its precision is to 0. For $(cl, sh) \in SH^W$ precision is $|\Downarrow cl \cup sh|/(2^n - 1)$.

Our results are shown in tables 1 for Sharing and 2 for Sharing+Freeness. Columns **time** show analysis times in milliseconds. on a medium-loaded Pentium IV Xeon 2.0Ghz with two processors, 4Gb of RAM memory, running Fedora Core 2.0, and averaging several runs after eliminating the best and worst values. Ciao version 1.11#326 and CiaoPP 1.0#2292 were used. Columns labeled **precision** show the number of sharing groups in the information inferred and, between parenthesis, the number of sharing groups for the worst-case sharing. Since our analyses infer information at all program points (before and after calling each clause body atom), and several variants for each program point, we show the accumulated number of sharing groups in all variants for all program points, instead of the absolute precision.

In both tables, first the numbers for the original domain are shown, then the numbers for the clique-domain. The columns $\Delta\%$ and $\Delta\#$ show the relative comparison of the clique-domain to the original domain for time and for precision, respectively. Given TP the (total) number of sharing groups for the clique-domain, R the (total) number of sharing groups for the original domain and $W1$ and $W2$ the (total) number of sharing groups in the worst-case sharing in each case, respectively, the precision is computed as $100 * (R/W2 - TP/W1)$. This number ($\Delta\#$) shows the variation in units of precision measured in percentage, so that we can talk of "points" of precision gained (if positive) or lost (if negative) by the clique-domain. For efficiency, given TP the number for the clique-domain and R the number for the original domain, the relation ($\Delta\%$) is computed as $100 * (1 - TP/R)$, showing the percentage of improvement for the clique-domain, if positive, or of how worse it goes, if negative, over the original domain.

Benchmarks are divided into three groups. The first group, append through serialize, is a set of simple programs, used as a testbed for an anal-

	Sharing		Clique-Sharing			
	time	**precision**	**time**	**Δ%**	**precision**	**Δ#**
append	11.99	29 (60)	12.49	-4.17	44 (60)	-25
deriv	33.99	27 (546)	36.99	-8.82	27 (546)	0
mmatrix	11.99	14 (694)	13.99	-16.68	14 (694)	0
qsort	31.49	30 (1716)	34.99	-11.11	30 (1716)	0
query	17.49	35 (501)	18.99	-8.57	35 (501)	0
serialize	547.51	1734 (10531)	132.18	75.85	2443 (10531)	-6.72
aiakl	54.49	145 (13238)	61.99	-13.76	145 (13238)	0
boyer	937.52	1688 (4631)	463.92	50.51	2005 (4631)	-6.83
browse	36.49	69 (776)	41.49	-13.70	69 (776)	0
prolog_read	566.71	1080 (408755)	628.90	-10.97	1080 (408755)	0
rdtok	450.59	1350 (11513)	482.67	-7.11	1370 (11513)	-0.16
warplan	3457.27	4202 (26306)	1566.56	54.68	5989 (19006)	-15.53
zebra	44.66	280 (671088746)	56.49	-26.48	280 (671088746)	0
ann	-	-	1220.21	-	19658 (314825)	-
peephole	1702.07	2210 (12148)	748.88	56	3329 (12845)	-7.71
qplan	-	-	2175.91	-	420519 (3827610)	-
witt	556.41	858 (4545564)	603.90	-8.53	858 (4545564)	0

Table 1: Precision and Time-efficiency for Sharing

ysis: they have only direct recursion and make a straightforward use of unification (basically, for input/output of arguments). The second group, aiakl through zebra, are more involved: they make use of mutual recursion and of elaborated aliasing between arguments to some extent; some of them are parts of "real" programs (aiakl is part of an analyzer of the AKL language; prolog_read and rdtok are parsers of Prolog). The benchmarks in the third group are all (parts of) "real" programs: ann is the &-prolog parallelizer, peephole is the peep-hole optimizer of the SB-Prolog compiler, qplan is the core of the Chat-80 application, and witt is a conceptual clustering application. Of each group we only show a reduced number of the benchmarks actually used: those which are more representative.

In order to understand the results shown in the above tables it is important to note an existing synergy between normalization, efficiency, and precision. If normalization causes no change in the sharing representation (i.e., sharing groups are not moved to cliques), usually because powersets do not really occur during analysis, then the clique part is empty. Analysis is the same as without cliques, but with the extra overhead due to the use of the normalization process. Then precision is the same but the time spent in analyzing the program is a little longer. This also occurs often if the use of normalization is kept to a minimum: only for correctness (in our implementation, normalization is required for correctness at least for the

	Sharing+Freeness		Clique-Sharing+Freeness			
	time	**precision**	**time**	**Δ%**	**precision**	**Δ#**
append	7.5	7 (30)	8.24	-9.8	7 (30)	0
deriv	21	21 (546)	23.32	-11	21 (546)	0
mmatrix	8	12 (694)	9	-12.5	12 (694)	0
qsort	20	30 (1716)	31	-3.33	30 (1716)	0
query	11	22 (501)	12	-9.1	22 (501)	0
serialize	57.32	545 (5264)	46.5	18.87	736 (5264)	-3.63
aiakl	34	145 (13238)	38.6	-13.52	145 (13238)	0
boyer	380.74	1739 (5036)	259.56	31.82	2082 (5036)	-6.81
browse	24	69 (776)	26	-8.33	69 (776)	0
prolog_read	351.94	1050 (408634)	453.43	-28.83	1050 (408634)	0
rdtok	360.44	1047 (11513)	315.95	12.34	1061 (11513)	-0.12
warplan	2001	2436 (19644)	1601.35	20	6959 (23274)	-17.5
zebra	25.66	280 (671088746)	30.32	-18.16	280 (671088746)	0
ann	1703.5	7811 (401220)	1907.9	-12	14439 (394830)	- 1.71
peephole	957.65	1475 (9941)	791.28	17.37	2861 (12788)	-7.44
qplan	-	-	-	-		-
witt	722.14	813 (4545594)	780.68	-8.1	813 (4545594)	0

Table 2: Precision and Time-efficiency for Sharing+Freeness

extend function and other functions used for comparing abstract substitutions). This should not be surprising, since the fact that powersets occur during analysis at a given time does not necessarily mean that they keep on occurring afterwards: they can disappear because of groundness or other precision improvements during subsequent analysis (of, e.g., builtins).

When the normalization process is used more often (like for example at every call to *call*2*entry* and *extend*, as we have done), then more often sharing groups are moved to cliques. Thus, the use of the operations that compute on clique sets produces efficiency gains, and also precision losses, as it was expected. However, precision losses are not high. Finally, if normalization is used too often, then the analysis process suffers from a heavy overhead, causing such a penalty in efficiency that it makes the analysis intractable. Therefore it is very clear that a thorough tuning of the use of the normalization process is crucial to lead analysis to good results in terms of both precision and efficiency.

However, there are always programs the analysis of which does not produce cliques. This shows in some of the benchmarks (like all of the first group but serialize and some of the second one such as aikl, browse, prolog_read, and zebra). In this case, as it was expected, precision is maintained but there is a small loss of efficiency (around 10% or little higher) due to the commented extra overhead.

On the other hand, for those benchmarks which do generate cliques (like serialize, boyer, warplan, and peephole) the gain in efficiency is very high (around 50% or higher), at the cost of a small precision loss (of around 10 precision points). As usual, efficiency and precision correlate inversely: if precision increases then efficiency decreases and vice versa. An special case is that of append (and, to some extent, rdtok), since precision losses are not coupled with efficiency gains. The reason is that for these benchmarks there are extra success substitutions (which, in fact, do not convey extra precision, but the other way around) that make the analysis to run longer. The effects are maintained with the addition of freeness, although the efficiency gains are lower. The reason is that the function $amgu^{sf}$ is less efficient than $amgu^{s}$ (but more precise). Overall, however, the trade between precision and efficiency is beneficial. Moreover, the more compact representation of the clique-domains allows to analyze benchmarks (ann and qplan) which ran out of memory with the standard representation.

talk about use of memory is pending

G Precision and Efficiency Results when using Normalization as a Widening