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

Let $F$ be a set of ranked 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 $\ell$ 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 $\text{solve}(t_1 = t_2)$ denote the solved form of unification equation $t_1 = t_2$, $t_1 \in Term$, $t_2 \in Term$.

\footnote{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: call2entry, exit2succ, 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 \( x = t, x \in V, t \in \text{Term}, \) and \( ASub \) an abstract substitution (the domain of which contains variables \( t \cup \{ x \} \)). Abstract unification for equation \( t_1 = t_2, t_1 \in \text{Term}, t_2 \in \text{Term}, \) in the context of abstract substitution \( ASub \) (the domain of which contains variables \( t_1 \cup t_2 \)) 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 call2entry and exit2succ can be defined as follows.\(^2\) Note that, in call2entry, \( ASub \) is an abstract substitution with domain the variables of Goal; the result is an abstract substitution with domain the variables of Head. In exit2succ the domain of \( ASub \) is the variables of Head, and that of the result the variables of Goal.

\[
call2entry(ASub, Goal, Head) = unify(ASub, Head, Goal) \\
exit2succ(ASub, Goal, Head) = unify(ASub, Goal, Head)
\]

where:

\[
unify(ASub, t_1, t_2) = project(t_1, Amgu(solve(t_1 = t_2), augment(t_1, ASub)) \})
\]

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 \( t \in \text{Term} \) and \( sh \in SH \), we denote by \( sh_t \) the sharing in \( sh \) related to \( t \), defined as:

\[
sh_t = \text{rel}(t, sh) = \{s \mid s \in sh, s \cap \hat{t} \neq \emptyset \}
\]
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 \( sh_1 \in SH, sh_2 \in SH \),

\[
sh_1 \& sh_2 = \{s_1 \cup s_2 \mid s_1 \in sh_1, s_2 \in sh_2 \}
\]
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 \( x = t, x \in V, t \in \text{Term} \), and \( sh \in SH \), is defined as:

\[
amgu(x = t, sh) = \overline{sh_t} \cup (sh^*_x \& sh^*_x)
\]

**Abstract functions** Let \( sh_1 \in SH, sh_2 \in SH \), and \( g \in \text{Term} \) (a goal),

\[
\text{extend}(sh_1, g, sh_2) = \overline{sh_1} \cup \{s \mid s \in sh_1^*, (s \cap \hat{g}) \in sh_2 \}
\]

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

\[
\text{augment}(g, sh_2) = sh_2 \cup \{\{x \mid x \in \hat{g}\}
\]

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 \( \text{project} \) and \( \text{augment} \) are the natural extension of the ones defined above.

\(^3\)Note that \( sh^*_x = (sh_t)^* \).
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 $\text{SHF} = \text{SH} \times V$.

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

$$sh' = \begin{cases} sh_{xt} \cup (sh_x \otimes sh_t) & \text{if } x \in f \text{ or } t \in f \\ sh_{xt} \cup (sh_x \otimes sh_t^j) & \text{if } x \not\in f, t \not\in f, \text{ but } i \subseteq f \text{ and } \text{lin}(t) \\ \text{amgu}(x = t, sh) & \text{otherwise} \end{cases}$$

and $\text{lin}(t)$ holds iff for all $y \in i$: $|t|_y = 1$ and for all $z \in i$ such that $y \not\in z$, $sh_y \cap sh_z = \emptyset$;

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

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

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

$$sh' = \text{extend}(sh_1, g, sh_2)$$

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

Functions $\text{project}^f$ and $\text{augment}^f$ are defined as follows:

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

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

---

4Note 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 \( x = t, x \in V, t \in \text{Term} \), and \((sh, f, l) \in SHL\), is given by
\[
\text{amgu}^l(x = t, (sh, f, l)) = (sh', f', l').
\]
Let \( \text{alin}(t) \) iff \( t \subseteq l \) and \( \text{lin}(t) \). Then:
\[
\begin{align*}
sh' &= \begin{cases} 
\overline{sh_x} \cup \overline{sh_t} & \text{if } x \in f \text{ or } t \in f \text{ or } \text{alin}(x) \text{ or } \text{alin}(t) \\
\text{amgu}(x = t, sh) & \text{otherwise}
\end{cases} \\
sh'' &= \begin{cases} 
sh_x \otimes sh_t & \text{if } x \in f \text{ or } t \in f \\
(\overline{sh_x} \cup (sh_x \otimes sh_{x}^{*})) \otimes & \text{if } \text{alin}(x), \text{alin}(t) \\
(\overline{sh_t} \cup (sh_t \otimes sh_t^{*})) & \text{if } \text{alin}(x), -\text{alin}(t) \\
sh_x \otimes sh_t^{*} & \text{if } -\text{alin}(x), \text{alin}(t)
\end{cases}
\end{align*}
\]
\[
l' = f' \cup \begin{cases} 
l \setminus (\overline{sh_x} \cap \overline{sh_t}) & \text{if } \text{alin}(x), \text{alin}(t) \\
l \setminus (\overline{sh_x}) & \text{if } \text{alin}(x), -\text{alin}(t) \\
l \setminus (\overline{sh_t}) & \text{if } -\text{alin}(x), \text{alin}(t) \\
l \setminus (\overline{sh_x} \cup \overline{sh_t}) & \text{otherwise}
\end{cases}
\]
and \( f' \) is as in \( \text{amgu}^l \).

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 \( \text{extend}^l((sh_1, f_1, l_1), g, (sh_2, f_2, l_2)) = (sh', f', l') \), where:
\[
\begin{align*}
sh' &= \overline{sh_{1g}} \cup \cup \{ s \mid s \in sh_{1g}, s' \subseteq s \} \cup \{ s' \mid s' \in sh_2, s' \not\subseteq l_2 \} \\
&\quad \cup \{ s \mid s \in sh_{1g}, (s \cap \hat{g}) \in sh_2, (s \cap \hat{g}) \subseteq l_2 \}
\end{align*}
\]
Note that the \( \text{extend} \) function for the original Sharing domain is equivalent to the above expression \textbf{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 $\text{project}^t$ and $\text{augment}^t$ are defined as follows:

$$\text{project}^t(g, (sh, f, l)) = (\text{project}(g, sh), f \cap \check g, l \cap \check g)$$

$$\text{augment}^t(g, (sh, f, l)) = (\text{augment}(g, sh), f \cup \check g, l \cup \check g)$$

### 2.3 Specific Abstract Functions for Unification

The abstract functions $\text{call2entry}$ and $\text{exit2succ}$ can be defined easily from $\text{amgu}$, as we have seen. However, by defining such functions specifically for each domain, precision can be improved in some cases. Function $\text{amgu}$ has the drawback that it is symmetric: all variables in the unification equations are treated uniformly. Contrary to this, the equation $\text{Goal} = \text{Head}$ which is used during analysis is not symmetric. When using $\text{call2entry}$, the variables of $\text{Head}$ are known to be new, i.e., free and unaliased; when using $\text{exit2succ}$, it is the variables of $\text{Goal}$ 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 $\text{amgu}$.

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 $\text{unify}$ (the variables correspond to $t_1$ in the definition of $\text{unify}$). This function calls $\text{augment}$, and both $\text{augment}^t$ and $\text{augment}^1$ include such variables as free and unaliased. Thus, this information is already present during $\text{Amgu}$. This is not the case, however, for the base Sharing domain. One way to take advantage of the extra information is to define $\text{unify}$ in such a way that it uses $\text{amgu}^t$ or $\text{amgu}^1$ (which do exploit such information) and provide them with the necessary information on the new variables. For example, for the Sharing+Freeness domain, let $\text{Amgu}^t$ the version of $\text{Amgu}$ which uses $\text{amgu}^t$ (a similar construction could be done with $\text{amgu}^1$ for the Sharing+Freeness+Linearity domain); function $\text{unify}$ can be defined specifically for the Sharing domain, as follows:

$$\text{unify}(\text{ASub}, t_1, t_2) = \text{project}(t_1, \text{ASub}')$$

where:

$$(\text{ASub}', \text{Free}) = \text{Amgu}^t(\text{solve}(t_1 = t_2), \text{augment}^t(t_1, (\text{ASub}, \emptyset)))$$

so that abstract functions $\text{augment}^t$ and $\text{amgu}^t$ for the Sharing+Freeness domain are used, but the result “projected” onto the sharing component.
However, the use of $amgu^l$ (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:

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

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

$$\text{amu}(x = t, \text{Lin}, \text{sh}) = \begin{cases} \text{sh}_{xt} \cup (\text{sh}_x \otimes \text{sh}_t) & \text{if } x \in \text{Lin} \text{ or } t \in \text{Lin} \\ \text{sh}_{xt} \cup (\text{sh}_x \otimes \text{sh}_t^*) & \text{if } x \notin \text{Lin}, t \notin \text{Lin}, \text{ but } t \subseteq \text{Lin} \text{ and } \text{lin}(t) \end{cases}$$

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

The Non-redundant version of Sharing is defined in \[\text{[?]}\]. 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 \(s^x_1 = s_1 \Join s_1\).

**Abstract unification** Abstract unification for equation \(x = t\), \(x \in V\), \(t \in \text{Term}\), and \(sh \in SH\), is defined as:\[^5\] \(\forall\) \(amgu^\theta(x = t, sh) = \overline{sh_{xt}} \cup (sh^x_1 \Join sh^x_1)\)
i.e., substituting self-binary union for the star union.

**Abstract functions** Functions \textit{project} and \textit{augment} for the Sharing domain are also correct for this domain. An abstract function \textit{extend} 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 \[\text{[?]}\]. 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 \[\text{[?]}\]. Basically, star union is replaced everywhere by self-binary union.

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

\[
sh' = \begin{cases} 
\overline{sh_{xt}} \cup (sh_x \Join sh_t) & \text{if } x \notin f \text{ or } t \notin f \\
\overline{sh_{xt}} \cup (sh_x \Join sh^x_t) & \text{if } x \notin f, \ t \notin f, \text{ but } \bar{t} \subseteq f \text{ and } \text{lin}(t) \\
amgu^\theta(x = t, sh) & \text{otherwise}
\end{cases}
\]

and \(f'\) is as in \(amgu^\theta\).

\[^5\]Note that \(sh^x_1 = (sh_t)^x\).
Abstract functions  Functions project\textsuperscript{\textdagger} and augment\textsuperscript{\textdagger} 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\textsuperscript{\textdagger} for the original Sharing + Freeness domain should serve.

3.2 Non-Redundant Sharing + Freeness + Linearity

Under construction

4 Non-Redundant Clique-Sharing Domains

This domain is defined in \cite{[2]}. 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 \((cl, sh) \in SH^W\),
\[
(cl, sh)^\times = cl^\times \cup (cl \otimes sh)
\]
is the extension of self-binary union to \( SH^W \).

Non-related sharing  Let \( t \in Term \) and \( cl \in CL \),
\[
\text{rel}(t, cl) = \{ c \mid t \in cl \} \setminus \{0\}
\]
For two terms \( s \) and \( t \) we will write \( \text{rel}(st, cl) \), 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:
\[
amgu^W(x = t, (cl, sh)) = \\
( \text{rel}(xt, cl) \cup \\
((cl_x, sh_x)^\times \otimes (cl_t, sh_t)^\times) \cup ((cl_x, sh_x)^\times \otimes sh_t^\times) \cup (sh_x^\times \otimes (cl_t, sh_t)^\times) \\
sh_{xt} \cup (sh_x^\times \otimes sh_t^\times) )
\]
This abstract unification operation is defined after the one presented in Definition 11 of \cite{[2]}, 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

4.2 Non-Redundant Clique-Sharing+Freeness+Linearity

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 $(cl, sh) \in SH^W$, $(cl, sh)^* = cl^* \cup (cl^* \& sh^*)$

Note that $*: SH^W \rightarrow CL$ 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:

$amgu^w(x = t, (cl, sh)) =$

$\{ \text{rel}(xt, cl) \cup ((cl_x, sh_x)^* \& (cl_t, sh_t)^*) \cup ((cl_x, sh_x)^* \& sh^*_t) \cup (sh^*_x \& (cl_t, sh_t)^*) \} \cup (sh_{xt} \& (sh^*_x \& sh^*_t))$  

This abstract unification operation is defined after the one in the previous section, replacing self-binary union by star union.
Abstract functions Let \( g \in \text{Term} \), \((cl, sh) \in SH^W\). Functions \( \text{project}^* \) and \( \text{augment}^* \) are defined as follows:

\[
\text{project}^*(g, (cl, sh)) = (\text{project}(g, cl), \text{project}(g, sh))
\]

\[
\text{augment}^*(g, (cl, sh)) = (cl, \text{augment}(g, sh))
\]

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

\[
(cl', sh') = \text{normalize}(((cl_1^g, sh_1^g), (cl_2^g, sh_2^g)))
\]

The following two functions lift the classical extend to the case of clique-sets and sharing-sets occurring in the pairs of \( \text{Call} \) and \( \text{Prime} \):

\[
\text{extsh}(sh_1, g, sh_2) = \overline{sh_1^g} \cup \{ s \mid s \in sh', (s \cap g) \in \overline{sh_2^g} \}
\]

\[
\text{extcl}(cl_1, g, cl_2) = \overline{cl_1^g} \cup \{ (s' \cap s) \cup (s' \setminus \hat{g}) \mid s' \in cl', s \in cl_2 \}
\]

The following two functions account for the cases of the clique-set of \( \text{Call} \) and the sharing-set of \( \text{Prime} \), and the other way around:

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

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

The function extend for Sharing-clique is thus:

\[
\text{extend}^*((cl_1, sh_1), g, (cl_2, sh_2)) = \\
\text{extcl}(cl_1, g, cl_2) \cup \text{extsh}(sh_1, g, sh_2) \cup \text{clsh}(cl', g, sh_2) \cup \text{shcl}(sh', g, cl_2)
\]

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 \( x = t, x \in V, t \in \text{Term} \), and \((clsh, f) \in \text{SHF}^W\), \(clsh = (cl, sh)\), is given by \( amgu^f(x = t, (clsh, f)) = (clsh', f') \), where:

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

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

\[
amgu^f(x = t, (cl, sh)) = \begin{cases} 
    \overline{\text{rel}}(xt, cl) \cup ((cl_x \cup sh_x) \otimes cl_t) \cup (cl_x \otimes sh_t) \\
    sh_x \cup (sh_x \otimes sh_t)
\end{cases}
\]

\[
amgu^f(x = t, (cl, sh)) = \begin{cases} 
    \overline{\text{rel}}(xt, cl) \cup ((cl_x \cup sh_x) \otimes (cl_t, sh_t) \uplus) \cup (cl_x \otimes sh_t) \\
    sh_x \cup (sh_x \otimes sh_t)
\end{cases}
\]

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

Note again that checking emptiness of each pairwise intersection in the definition of \( \text{lin}^*(t) \) (as in \( \text{lin}(t) \)) 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 \( \text{extend}^sf \) for this domain is given by \( \text{extend}^sf((clsh_1, f_1), g, (clsh_2, f_2)) = ((cl', sh'), f') \), where:

\[
(cl', sh') = \text{extend}^s(clsh_1, g, clsh_2)
\]

\[
f' = f_2 \cup \{ x \mid x \in (f_1 \setminus \hat{g}), ((\cup (sh_x \cup cl_x)) \cap \hat{g}) \subseteq f_2 \}
\]

Functions \( \text{project}^sf \) and \( \text{augment}^sf \) are defined as follows:

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

\[
\text{augment}^sf(g, (clsh, f)) = (\text{augment}^s(g, clsh), f \cup \hat{g})
\]
5.2 Clique-Sharing+Freeness+Linearity

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 $\forall cl \cap sh = \emptyset$. 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 “transferred” 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 $s \subseteq (\cup cl \cup sh)$ such that $s = \{c\}$ 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 $2^l - 1 - |S|$ 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 $SS = \{s \mid s \in sh, s \subseteq S\}$.
10. If $|SS| = 2^i - 1 - |S|$ then:
   (a) Add $S$ to $cl$ (regularize $cl$).
   (b) Subtract $SS$ from $sh$.

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 $c_1 \subseteq c_2$. 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 $R$.

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

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

\[
\text{pos}(t, S) = \{i \mid S \cap t_i \neq \emptyset\}
\]

\[
\mathcal{P}(t, sh) = \{\text{pos}(t, S) \mid S \in sh\}
\]

Let $g \in \text{Term}$ and $h \in \text{Term}$ 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 $\text{call2entry}(sh, g, h)$ is an abstract substitution for $h$ given by:

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

where:

\[
\beta = \bigcup_{S \in \mathcal{P}} \phi^S(S)
\]

\[
P = \text{partition}(h \setminus G, DG)
\]

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

\[
G = \{x \mid (x \to \emptyset) \in Eq\}
\]

\[
Eq = \text{propagate}(\text{normalize}(\text{solve}(g = h)) \cup \{x \to \emptyset \mid x \in (\hat{g} \setminus sh)\})
\]

and:

\[
\text{normalize}(E) = \{x \to t \mid (x = t) \in E, x \in V, t \in \text{Term}\}
\]

\[
\text{propagate}(E) = \begin{cases} 
\{x \to \emptyset\} \cup \text{propagate}(\text{ground}(E', x)) & \text{if } E = E' \cup \{x \to \emptyset\} \\
E & \text{otherwise}
\end{cases}
\]

\[
\text{ground}(E, x) = \{y \to S \setminus \{x\} \mid (y \to S) \in E, y \neq x\} \cup \{y \to \emptyset \mid (x \to S) \in E, y \in S\}
\]

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 \( \downarrow s = \varphi^0(s) \). For a set of sets \( ss \), we define \( \downarrow ss = \cup \{ s \mid s \in ss \} \).

Let \( c \subseteq V \) be a clique. Note that \( \downarrow c \) denotes all the sharing that is implicitly 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:

\[
(\downarrow s_1)^* = \downarrow s_1
\]

Powerset and union do not commute:

\[
(\downarrow(s_1 \cup s_2))^* \supseteq \downarrow s_1 \cup \downarrow s_2
\]

but not the other way around.

**Lemma 2** For every \( cl \in CL \):

\[
\downarrow cl \supseteq cl^*
\]

\[
\downarrow cl \supseteq \downarrow cl
\]

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 \( (\cup_{i=1}^n s_i) \subseteq \downarrow cl \), so that \( s \in \downarrow \downarrow cl \).

However, \( cl \) might not have singleton sets, which are thus not in \( cl^* \), but they are indeed subsets of \( \downarrow cl \).

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

**Lemma 3** For every \( cl \in CL \):

\[
\downarrow cl^* = \downarrow cl
\]

\[
\downarrow cl^* = (\downarrow cl)^*
\]

\(^7\)Note that \( \downarrow cl^* = \varphi(cl^*) \).
Proof First note that:

\[ \cup \psi d = \psi d^* = \cup \psi d \]  \hspace{1cm} (7)

and, if \( cl \neq \emptyset \):

\[ \cup \psi d \in cl^* \]  \hspace{1cm} (8)

Thus, \( \psi cl^* \subseteq \psi cl^* \), since, using (4) and (7), we have that:

\[ \psi cl^* \subseteq \psi cl^* = \psi cl. \]

Also, \( \cup \psi cl \subseteq \psi cl^* \). To see this, take \( s \in \cup \psi cl \), we also have that

\( \psi cl \subseteq \psi cl^* \). If \( cl = \emptyset \), the result follows directly.

Now, \( \psi cl^* \subseteq (\psi cl)^* \). Take \( s \in (\psi cl)^* \), so that \( s \subseteq C, C \in cl^* \).

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 \( s = \cup c_i \). Therefore, for all \( i \in I, c_i \subseteq s_i \), and \( s_i \subseteq cl \), so that \( c_i \in \psi cl \). Thus,

\( \cup c_i \in (\psi cl)^* \). Since \( s = \cup c_i \), then \( s \in (\psi cl)^* \).

Finally, using (4) with (18), then (1) and (5), we also have

\( (\psi cl)^* \subseteq (\psi cl)^* = \psi cl = \psi cl^* \).

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

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

\[ \psi (cl_1 \circ cl_2) \supseteq \psi cl_1 \circ \psi cl_2 \]  \hspace{1cm} (9)

but not in general the other way around.

**Proof** If either \( cl_1 = \emptyset \) or \( cl_2 = \emptyset \) the 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 \( cl_2 \neq \emptyset \):

\[ \psi (cl_1 \circ cl_2) = \psi cl_1 \cup (\psi cl_1 \circ \psi cl_2) \cup \psi cl_2 \]  \hspace{1cm} (10)

**Proof** Note that for every set of sets \( ss, s \in \psi ss \) iff there is a \( c \in ss \) such that \( s \cup c \in \psi cl \). In other words, \( s \subseteq c \in ss \) and \( s \neq \emptyset \).

We first prove that \( \psi (cl_1 \circ cl_2) \subseteq \psi cl_1 \cup (\psi cl_1 \circ \psi cl_2) \cup \psi cl_2 \).

Take \( s \in \psi (cl_1 \circ cl_2) \). Then \( s \neq \emptyset \) and \( s \subseteq c \in (cl_1 \circ cl_2) \). That
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 $s \in \psi cl_i$. 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 $s_i \in \psi cl_i$. Therefore, $s \in (\psi cl_1 \cap \psi cl_2)$. In any case, $s \in \psi cl_1 \cup (\psi cl_1 \cap \psi cl_2) \cup \psi cl_2$.

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

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

Projection is correct for clique sets, but imprecise:

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

$$\psi cl_t \supseteq (\psi cl)_t$$

but not the other way around.

**Proof** Let $s \in (\psi cl)_t$. Then $s \cap t \neq \emptyset$ and there is $C \in cl$ such that $s \in [C]$. Thus, $s \subseteq C$, so that $C \cap t \neq \emptyset$. Therefore, $C \in cl_t$, but also $s \subseteq [C]$, so that $s \in \psi cl_t$.

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


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:

\begin{align*}
(s_{1} \uplus s_{2})^{*} &= s_{1}^{*} \uplus s_{2}^{*} \tag{12} \\
(s_{1} \cup s_{2})^{*} &\supseteq s_{1}^{*} \uplus s_{2}^{*} \tag{13} \\
(s_{1} \cup \{\emptyset\})^{*} &= s_{1}^{*} \uplus \{\emptyset\} \tag{14} \\
ss_{1} \uplus (ss_{2} \cup ss_{3}) &= (ss_{1} \uplus ss_{2}) \cup (ss_{1} \uplus ss_{3}) \tag{15}
\end{align*}

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

\[
\cup(ss_1 \uplus ss_2) = \cup(ss_1 \cup ss_2)
\tag{16}
\]

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

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

\[
\psi(cl, sh)^{*} = \cup(cl \cup sh)
\]

**Proof** We first prove the following auxiliary result:

\[
(cl, sh)^{*} = (cl \uplus (sh \cup \{\emptyset\}))^{*} \tag{17}
\]

\[
(cl, sh)^{*} = cl^{*} \cup (cl^{*} \uplus sh^{*}) \tag{15} \]

\[
= cl^{*} \uplus (sh^{*} \cup \{\emptyset\}) \tag{14} \]

\[
= cl^{*} \uplus (sh^{*} \uplus \{\emptyset\}) \tag{12} \]

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

\[
\psi(cl, sh)^{*} = \psi(cl \uplus (sh \cup \{\emptyset\}))^{*} \tag{15} \]

\[
= \cup(cl \cup (sh \cup \{\emptyset\})) \tag{16} \]

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 $cl_{2} \neq \emptyset$:

\[
\psi(cl_{1}^{*} \uplus cl_{2}^{*}) = \cup(cl_{1} \cup cl_{2})
\]
Proof  Note that (16) can be applied below because $c_{1} \neq \emptyset$ and $c_{2} \neq \emptyset$. Thus:

$$\bigcup(c_{1} \uplus c_{2})^{\ast} \overset{(12)}{=} \bigcup(c_{1} \uplus c_{2}) \overset{(16)}{=} \bigcup(c_{1} \cup c_{2})$$

With this, we can redefine abstract unification without loss of precision (and correctness) into an optimized operation $amgu^{\circ}$, which can be used in place of $amgu$. \[amgu^{\circ}(x = t, (cl, sh)) = \begin{cases} (cl \uplus sh_{xt} \uplus (sh_{x}^{*} \uplus sh_{t}^{*})) & \text{if } cl_{x} = cl_{t} = \emptyset \\ (rel(xt, cl), sh_{xt}) & \text{if } cl_{x} = sh_{x} = \emptyset \\ \text{or } cl_{t} = sh_{t} = \emptyset \\ \text{otherwise} \end{cases} \]

Theorem 1 Given $x = t$, $x \in V$, $t \in Term$, and $(cl, sh) \in SH^{W}$. Let $amgu^{\circ}(x = t, (cl, sh)) = (cl', sh')$ and $amgu^{\ast}(x = t, (cl, sh)) = (cl'', sh'')$. Then:

$$\bigcup cl' \cup sh' = \bigcup cl'' \cup sh''$$

Proof Let $clsh = ((cl_{x}, sh_{x})^{*} \uplus (cl_{t}, sh_{t})^{*}) \cup ((cl_{x}, sh_{x})^{*} \uplus sh_{t}^{*}) \cup (sh_{x}^{*} \uplus (cl_{t}, sh_{t})^{*})$. Recall that:

$$cl'' = rel(xt, cl) \cup clsh \quad \text{and} \quad sh'' = sh_{xt} \cup (sh_{x}^{*} \uplus sh_{t}^{*})$$

For $cl'$ and $sh'$ we have three cases:

- $cl_{x} = cl_{t} = \emptyset$
  
  Since $cl_{x} = cl_{t} = \emptyset$ then $rel(xt, cl) = cl$ and $(cl_{x}, sh_{x})^{*} = (cl_{t}, sh_{t})^{*} = \emptyset$, so that $(cl_{x}, sh_{x})^{*} \uplus (cl_{t}, sh_{t})^{*} = (cl_{x}, sh_{x})^{*} \uplus sh_{t}^{*} = sh_{x}^{*} \uplus (cl_{t}, sh_{t})^{*} = \emptyset$. Then $clsh = \emptyset$, so that $cl'' = cl$. Thus, what we have to prove is:\n
$$\bigcup cl' \cup sh' = \bigcup cl \cup sh''$$

which follows because, in this case, $cl' = cl$ and $sh' = sh_{xt} \cup (sh_{x}^{*} \uplus sh_{t}^{*}) = sh''$.

- $cl_{x} = sh_{x} = \emptyset$ or $cl_{t} = sh_{t} = \emptyset$
  
  Take $cl_{x} = sh_{x} = \emptyset$. Then also $(cl_{x}, sh_{x})^{*} = sh_{x}^{*} = \emptyset$. Thus, we have $(cl_{x}, sh_{x})^{*} \uplus (cl_{t}, sh_{t})^{*} = (cl_{x}, sh_{x})^{*} \uplus sh_{t}^{*} = \emptyset$.
\[ sh^*_x \uplus (cl_t, sh_t)^* = \emptyset, \text{ so that } clsh = \emptyset. \] Also, \( sh^*_x \uplus sh^*_t = \emptyset \), so that \( sh^* = sh_{xt} \). The same reasoning applies to the case \( cl_t = sh_t = \emptyset \). Thus, what we have to prove is:

\[ \psi cl' \cup sh' = \psi \overline{rel}(xt, cl) \cup \overline{sh_{xt}} \]

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:

\[ \psi cl'' = \psi( \overline{rel}(xt, cl) \cup clsh ) = \psi \overline{rel}(xt, cl) \cup \psi clsh \]

\[ \psi cl' = \psi( \overline{rel}(xt, cl) \cup \{W\} ) = \psi \overline{rel}(xt, cl) \cup \psi \{W\} \]

\[ = \psi \overline{rel}(xt, cl) \cup \{W\} \]

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

\[ \psi \overline{rel}(xt, cl) \cup \{W\} \cup \overline{sh_{xt}} = \psi \overline{rel}(xt, cl) \cup \psi clsh \cup \overline{sh_{xt}} \cup (sh^*_x \uplus sh^*_t) \]

or, equivalently:

\[ \psi clsh \cup (sh^*_x \uplus sh^*_t) \]

which follows because \( \psi clsh \cup (sh^*_x \uplus sh^*_t) \subseteq W \), which we proceed to prove.

First, note that \( sh^*_x \uplus sh^*_t \) is a set of sets of variables from \( \cup(sh_x \cup sh_t) \), and \( \cup(sh_x \cup sh_t) \subseteq W \). Therefore, \( sh^*_x \uplus sh^*_t \) is a subset of \( \psi clsh \).

Second, we show that \( \psi clsh \). We proceed by cases. The third case of \( amgu^c \) 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 \( clsh = (cl_x, sh_x)^* \uplus sh^*_t \).

Then, from (17), \( clsh = (cl_x \uplus (sh_x \cup \{\emptyset\}))^* \uplus sh^*_t \). 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:
\[ \psi_{\text{clsh}} \overset{\text{L8}}{=} \{ (cl_x \otimes (sh_x \cup \{\emptyset\}) \cup sh_t) \} \]
\[ = \{ (\cup (cl_x \otimes (sh_x \cup \{\emptyset\})) \cup sh_t) \} \]
\[ = \{ (\cup cl_x \cup (sh_x \cup \{\emptyset\}) \cup sh_t) \} \]
\[ = \{ U(cl_x \cup sh_x \cup sh_t) \} \]

But, since \( cl_t = \emptyset \), we have that \( W = U(cl_x \cup sh_x \cup sh_t) \), so that \( \psi_{\text{clsh}} = \{ W \} \).

- \( cl_x = \emptyset \), \( cl_t \neq \emptyset \), \( sh_x \neq \emptyset \)
  This case is symmetric to the previous one.
- \( cl_x \neq \emptyset \) and \( cl_t \neq \emptyset \)

Now we have to prove that:

\[ \{ W \} = \psi_{\text{clsh}} \]
\[ = \psi((cl_x \otimes (sh_x \cup \{\emptyset\}) \cup sh_t)^*) \]
\[ \cup \psi((cl_x \otimes (sh_x \cup \{\emptyset\}) \cup sh_t)^*) \]
\[ \cup \psi((cl_x \otimes (sh_x \cup \{\emptyset\}) \cup sh_t)^*) \]

But, since \( \psi((cl_x, sh_x)^* \otimes (cl_t, sh_t)^*) \) is a set of sets of variables from \( U(cl_x \cup sh_x \cup sh_t) \), and \( U(cl_x \cup sh_x \cup sh_t) \subseteq W \), then \( \psi((cl_x, sh_x)^* \otimes (cl_t, sh_t)^*) \) is a subset of \( \{ W \} \). The same happens for \( \psi((sh_x \otimes (cl_t, sh_t)^*) \). Therefore, it suffices to prove that

\[ \{ W \} = \psi((cl_x, sh_x)^* \otimes (cl_t, sh_t)^*) \]

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:

\[ \psi((cl_x, sh_x)^* \otimes (cl_t, sh_t)^*) \]
\[ \overset{\text{L8}}{=} \psi((cl_x \otimes (sh_x \cup \{\emptyset\}) \cup sh_t^*) \cup (cl_t \otimes (sh_t \cup \{\emptyset\}) \]
\[ = \{ U(cl_x \cup (sh_x \cup \{\emptyset\}) \cup (cl_t \otimes (sh_t \cup \{\emptyset\}))) \}
\[ = \{ U(cl_x \cup sh_x \cup cl_t \cup sh_t) \}
\[ = \{ W \} \]
D Correctness Results for Clique-Sharing

Some basic results which are proved somewhere else:

**Lemma 9** Let $s_1$, $s_2$, $s_3$, and $s_4$ be sets of sets. If $s_1 \subseteq s_3$ and $s_2 \subseteq s_4$ then:

$$s_1 \cup s_2 \subseteq s_3 \cup s_4$$  \hspace{1cm} (18)

$$s_1 \cap s_2 \subseteq s_3 \cap s_4$$  \hspace{1cm} (19)

$$s_1 \cap s_2 \supseteq s_3 \cap s_4$$  \hspace{1cm} (20)

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

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

$$\nabla_{rel}(x_t, c_l) = (\nabla c_l)_{x_t}$$

The extension of star-union to $SH_W$ is correct but imprecise:

**Lemma 11** Given $(c_l, s_h) \in SH_W$, we have that:

$$\nabla (c_l, s_h) \cup s_h \supseteq (\nabla c_l \cup s_h)$$

but not the other way around.

**Proof** First, we consider $c_l = \emptyset$. In this case, $\nabla (c_l, s_h) = \emptyset$ and $\nabla c_l = \emptyset$, so that $\nabla (c_l, s_h) \cup s_h = s_h = (\nabla c_l \cup s_h)$.

If $c_l \neq \emptyset$ then, by Lemma 7, $\nabla (c_l, s_h) = \nabla (c_l \cup s_h)$. This is the proper powerset of the set of variables $\nabla (c_l \cup s_h)$, and therefore it is a superset of any other set of sets of variables from $\nabla (c_l \cup s_h)$, such as, for example, $(\nabla c_l \cup s_h)$.

To see that the other direction does not hold in general, take $c_l = \{xy\}$ and $s_h = \{yz\}$. We have that $\nabla (c_l, s_h) = \{xyz\}$ and $\nabla c_l = \{x, xy, y\}$. Thus, $(\nabla c_l \cup s_h) = \{x, xy, y, yz\}$, which is a proper subset of $\{xyz\}$.

**Note** Although imprecise in general, $(c_l, s_h)$ is in fact precise when $c_l = \emptyset$ or $s_h = \emptyset$. When $c_l = \emptyset$ we have $(c_l, s_h) = \emptyset$, what makes this operation unnecessary (which is precisely the observation behind the first and second cases of amgu (6)). When $s_h = \emptyset$ we have $\nabla (c_l, s_h) = \nabla c_l$ and $(\nabla c_l \cup s_h) = (\nabla c_l)$, but $\nabla c_l = (\nabla c_l)$, 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 \(t \in \text{Term}\), and \(amgu^0(x = t, (cl, ss)) = (cl^0, ss^0)\). If \(\psi cl \cup ss \supseteq sh\) then:

\[
\psi cl^0 \cup ss^0 \supseteq amgu(x = t, sh)
\]

**Proof** We first prove the following instrumental results:

\[
\psi cl \cup ss \supseteq sh \Rightarrow (\psi cl \cup ss)t \supseteq sh_t \Rightarrow (\psi cl)_t \cup ss_t \supseteq sh_t
\]

(11) \(\psi cl \cup ss \supseteq sh_t\)

Also:

\[
(\psi cl)_t \cup \overline{ss}_t \supseteq \overline{sh}_t
\]

(22) since

\[
\psi cl \cup ss \supseteq sh \Rightarrow (\psi cl \cup ss)_t \supseteq sh_t \Rightarrow (\psi cl)_t \cup \overline{ss}_t \supseteq \overline{sh}_t
\]

From (21) we have \(\psi cl \cup ss \supseteq sh_t\) and the same also for \(x\):

\[
\psi cl_x \cup ss_x \supseteq sh_x.
\]

Thus, by (20),

\[
(\psi cl_x \cup ss_x)^* \supseteq \overline{sh_x} \supseteq \overline{sh}_t
\]

(23) By Lemma 10, \(\overline{\text{rel}}(xt, cl) = (\psi cl)_xt\), and from (22),

\[
\overline{\text{rel}}(xt, cl) \cup \overline{ss}_xt \supseteq \overline{sh}_xt
\]

(24) Now, recall that \(amgu(x = t, sh) = \overline{sh}_xt \cup (sh_x^* \uplus sh_t^*)\). We will use \(amgu^0\) instead of \(amgu^a\), since by Theorem 1 they are equivalent. So, we have three cases:

- \(cl_x = cl = \emptyset\)
  
  In this case, \(cl^0 = cl\) and \(ss^0 = \overline{ss}_xt \cup (ss_x^* \uplus ss_t^*)\). So, what we have to prove is:

\[
\psi cl \cup \overline{ss}_xt \cup (ss_x^* \uplus ss_t^*) \supseteq \overline{sh}_xt \cup (sh_x^* \uplus sh_t^*)
\]

We first show that \(ss_x^* \uplus ss_t^* \supseteq sh_x^* \uplus sh_t^*\). This follows from (23), since \(cl_x = cl = \emptyset\).

We now show that \(\psi cl \cup \overline{ss}_xt \supseteq \overline{sh}_xt\). This follows from (22), since \((\psi cl)_xt \subseteq \psi cl\).
• \( cl_x = ss_x = \emptyset \) 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 \( |cl_x \cup ss_x| = 0 \) or \( |cl_t \cup ss_t| = 0 \). Thus, from (21), either \( sh_x = \emptyset \) or \( sh_t = \emptyset \). In any case, \( sh^*_x \preceq sh^*_t \). So, what we have to prove is:

\[
\overline{rel}(xt, cl) \cup \overline{ss}_{xt} \supseteq \overline{sh}_{xt}
\]

but this is precisely (24), proved above.

• any other case
Now we have \( cl^o = \overline{rel}(xt, cl) \cup \{cl_x \cup cl_t \cup ss_x \cup ss_t\} \) and \( ss^o = \overline{ss}_{xt} \). Let \( W = (cl_x \cup cl_t \cup ss_x \cup ss_t) \). We have that \( \overline{rel}(xt, cl) \cup \{W\}) = \overline{rel}(xt, cl) \cup \{W\} = \overline{rel}(xt, cl) \cup \{W\} \). So, what we have to prove is:

\[
\overline{rel}(xt, cl) \cup \{W\} \cup \overline{ss}_{xt} \supseteq \overline{sh}_{xt} \cup (sh^*_x \preceq sh^*_t)
\]

First, we have that \( \overline{rel}(xt, cl) \cup \overline{ss}_{xt} \supseteq \overline{sh}_{xt} \) (24).
Second, we show that \( \{W\} \supseteq sh^*_x \preceq sh^*_t \). Let \( S = (\{cl_x \cup ss_x\}^* \preceq (\{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 \( S \supseteq sh^*_x \preceq sh^*_t \), so that \( \{W\} \supseteq sh^*_x \preceq sh^*_t \).

The previous result holds even for the case in which \( \cup cl \cup ss = sh \). That is, \( amgu^* \) is necessarily imprecise.

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

\[
\cup cl^o \cup ss^o \supseteq amgu(x = t, sh)
\]

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

**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 \( \cup cl \cup ss = sh \). Take also \( t = y \). Then \((cl^o, ss^o) = (\{xy\}, \emptyset)\), so that \( \cup cl^o \cup ss^o = \{x, xy, y\} \).
But \( amgu(x = t, sh) = \overline{sh}_{xt} \cup (sh^*_x \preceq sh^*_t) = \{xy\} \), which is a proper subset of \( \cup cl^o \cup ss^o \)
Note

Loss of precision occurs only in the third case of $\text{amgu}^o$. If $\downarrow_1 \cup \downarrow_2 = sh$ 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 $\text{amgu}^o$. In the first case, equations (21) and (23) turn also into equalities, so that equality also holds for the first case of $\text{amgu}^o$. Only the third case is imprecise.

Function $\text{extend}$ for Clique-Sharing is correct (but not precise):

**Theorem 3** Let $\text{Call} = (c_1, s_1) \in SH^W$ and $\text{Prime} = (c_2, s_2) \in SH^W$, such that the conditions for the $\text{extend}$ function hold, $g \in \text{Term}$, and $\text{extend}^*(\text{Call}, g, \text{Prime}) = (c', s')$. If $\downarrow_1 \cup s_1 \supseteq sh_1$ and $\downarrow_2 \cup s_2 \supseteq sh_2$ then:

$$\downarrow_1 \cup s' \supseteq \text{extend}(sh_1, g, sh_2)$$

**Proof** Pending proof
E  Correctness Results for Clique-Sharing+Freeness

Some basic results which are proved somewhere else:

Lemma 12 Let ss₁, ss₂, ss₃, and ss₄ be sets of sets. If ss₁ ⊇ ss₃ and ss₂ ⊇ ss₄ then:

\[
\begin{align*}
\cup ss₁ & \supseteq \cup ss₃ \\
ss₁ \cup ss₂ & \supseteq ss₃ \cup ss₄
\end{align*}
\]

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

Theorem 4 Let \(((cl, ss), f) \in SHFW, (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^{sf}(x = t, (sh, e)) = (sh', f'). If \cup cl \cup ss \supseteq sh and f \subseteq e then:

\[
\cup cl^o \cup ss^o \supseteq sh' \quad \text{and} \quad f^o \subseteq f'
\]

Proof First, we prove that \cup 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 \( sh' = sh_{xt} \cup (sh_x \& sh_t) \). Also, \( ss^o = ss_{xt} \cup (ss_x \& ss_t) \) and \( cl^o = rel(xt, cl) \cup ((cl_x \cup ss_x) \& cl_t) \cup (cl_x \& ss_t) \). Thus, what we have to prove is:

\[
\cup (rel(xt, cl) \cup ((cl_x \cup ss_x) \& cl_t) \cup (cl_x \& ss_t))
\]

that is:

\[
\cup rel(xt, cl) \cup (ss_{xt} \cup (ss_x \& ss_t)) \supseteq sh_{xt} \cup (sh_x \& sh_t)
\]

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

\[
\cup (cl_x \cup ss_x) \& cl_t \cup (cl_x \& ss_t) \cup (ss_x \& ss_t) \supseteq sh_x \& sh_t
\]

This is proved in Lemma 13 below.
• $x \not\in f$, $t \not\in f$, but $i \subseteq f$ and $\text{lin}^*(t)$

In this case, $ss^0 = ss_{xt} \cup (ss_x \otimes ss_t^*)$ and $cl^0 = rel(xt, cl) \cup ((cl_x \cup ss_x) \otimes (cl_t, ss_t)^*) \cup (cl_x \otimes ss_t^*)$, so that:

$$\nabla cl^0 = \nabla \text{rel}(xt, cl) \cup \nabla((cl_x \cup ss_x) \otimes (cl_t, ss_t)^*) \cup \nabla(cl_x \otimes ss_t^*)$$

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

$$\text{lin}^*(t) \Rightarrow \text{lin}(t)$$

To see this...

Thus, we have now two cases: either (1) $x \in e$ or $t \in e$, or (2) $x \not\in e$, $t \not\in e$, but $i \subseteq e$ and $\text{lin}(t)$. We proceed with them.

• $x \notin f$, $t \notin f$, $i \subseteq f$, $\text{lin}^*(t)$, $x \not\in e$, $t \not\in e$, $i \subseteq e$, $\text{lin}(t)$

Now, $sh' = sh_{xt} \cup (sh_x \otimes sh_t^*)$, so that what we have to prove is:

$$\nabla \text{rel}(xt, cl) \cup \nabla((cl_x \cup ss_x) \otimes (cl_t, ss_t)^*) \cup \nabla(cl_x \otimes ss_t^*)$$

or, equivalently, since \(\nabla \text{rel}(xt, cl) \cup ss_{xt} \supseteq sh_{xt}\) (24):

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

This is proved in Lemma 14 below.

• $x \notin f$, $t \notin f$, $i \subseteq f$, $\text{lin}^*(t)$, $x \in e$ or $t \in e$

Now, $sh' = sh_{xt} \cup (sh_x \otimes sh_t)$, so that what we have to prove is:

$$\nabla \text{rel}(xt, cl) \cup \nabla((cl_x \cup ss_x) \otimes (cl_t, ss_t)^*) \cup \nabla(cl_x \otimes ss_t^*)$$

or, equivalently, since \(\nabla \text{rel}(xt, cl) \cup ss_{xt} \supseteq sh_{xt}\) (24):

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

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

We now have that \( sh' = amgu(x = t, sh) \) and that \((cl^o, ss^o) = amgu^*(x = t, (cl, ss))\). Thus, the result follows directly from Theorem 2.

Now we prove that \( f^o \subseteq f' \). From the definition of \( amg * f \) 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 \( \cup (cl_t \cup ss_t) = \cup (\cup cl_t \cup ss_t) \), since both expressions are made of the same set of variables. But, from (21), \( \cup cl_t \cup ss_t \supseteq sh_t \), 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, \( f^o = f \setminus (ss_t \cup cl_t) \), so that what we have to prove is:

\[
f \setminus (ss_t \cup cl_t) \subseteq e \setminus \cup sh_t \tag{25}
\]

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 \( f' = f \setminus (sh_x \cup sh_t) \). Thus what we have to prove is:

\[
f \setminus (cl_x \cup cl_t \cup ss_x \cup ss_t) \subseteq e \setminus (sh_x \cup sh_t)
\]
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 $x$: $\cup(cl_x \cup ss_x) \supseteq \cup sh_x$.

- $x \notin f$, $t \notin f$, $x \in e$, 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:

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

**Proof** We first prove the following instrumental result:

$$(\bigcup(cl_x \cup ss_x) \otimes cl_t) \cup (\bigcup(cl_x \otimes ss_t) \cup (ss_x \otimes ss_t)) \supseteq sh_x \otimes sh_t \quad (28)$$

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

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

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

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

$$\bigcup((cl_x \cup ss_x) \otimes (cl_t, ss_t)^*) \cup (\bigcup cl_x \otimes ss_t^*) \cup (ss_x \otimes ss_t^*) \supseteq sh_x \otimes sh_t^*$$
Function \( \text{extend} \) for Clique-Sharing+Freeness is correct (but not precise):

**Theorem 5** Let \( \text{Call} = ((c_1, s_1), e_1) \in \text{SHF}_W \) and \( \text{Prime} = ((c_2, s_2), e_2) \in \text{SHF}_W \), such that the conditions for the \( \text{extend} \) function hold, \( g \in \text{Term}, \) \( \text{extend}_{\text{SHF}} \) \((\text{Call}, g, \text{Prime}) = ((c', s'), e')\), and \( \text{extend}_{\text{SHF}} \) \((\text{sh}_1, f_1), g, (\text{sh}_2, f_2) = (\text{sh}', f')\). If \( \text{call}_1 \cup s_1 \supseteq \text{sh}_1, e_1 \subseteq f_1, \) \( \text{call}_2 \cup s_2 \supseteq \text{sh}_2, \) and \( e_2 \subseteq f_2 \) then:

\[
\text{call}' \cup s' \supseteq \text{sh}' \quad \text{and} \quad e' \subseteq f'
\]

**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 $|\{cl \cup sh\}|/(2^n - 1)$.

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 $100 \times (R/W2 - TP/W1)$. 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 $100 \times (1 - TP/R)$, 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-
Table 1: Precision and Time-efficiency for Sharing

<table>
<thead>
<tr>
<th></th>
<th>time</th>
<th>precision</th>
<th>time</th>
<th>Δ%</th>
<th>precision</th>
<th>Δ#</th>
</tr>
</thead>
<tbody>
<tr>
<td>append</td>
<td>11.99</td>
<td>29 (60)</td>
<td>12.49</td>
<td>-4.17</td>
<td>44 (60)</td>
<td>-25</td>
</tr>
<tr>
<td>deriv</td>
<td>33.99</td>
<td>27 (546)</td>
<td>36.99</td>
<td>-8.82</td>
<td>27 (546)</td>
<td>0</td>
</tr>
<tr>
<td>mmatrix</td>
<td>11.99</td>
<td>14 (694)</td>
<td>13.99</td>
<td>-16.68</td>
<td>14 (694)</td>
<td>0</td>
</tr>
<tr>
<td>qsort</td>
<td>31.49</td>
<td>30 (1716)</td>
<td>34.99</td>
<td>-11.11</td>
<td>30 (1716)</td>
<td>0</td>
</tr>
<tr>
<td>query</td>
<td>17.49</td>
<td>35 (501)</td>
<td>18.99</td>
<td>-8.57</td>
<td>35 (501)</td>
<td>0</td>
</tr>
<tr>
<td>serialize</td>
<td>547.51</td>
<td>1734 (10531)</td>
<td>132.18</td>
<td>75.85</td>
<td>2443 (10531)</td>
<td>-6.72</td>
</tr>
<tr>
<td>aiakl</td>
<td>54.49</td>
<td>145 (13238)</td>
<td>61.99</td>
<td>-13.76</td>
<td>145 (13238)</td>
<td>0</td>
</tr>
<tr>
<td>boyer</td>
<td>937.52</td>
<td>1688 (4631)</td>
<td>463.92</td>
<td>50.51</td>
<td>2005 (4631)</td>
<td>-8.83</td>
</tr>
<tr>
<td>browse</td>
<td>36.49</td>
<td>69 (776)</td>
<td>41.49</td>
<td>-13.70</td>
<td>69 (776)</td>
<td>0</td>
</tr>
<tr>
<td>prolog_read</td>
<td>566.71</td>
<td>1080 (408755)</td>
<td>628.90</td>
<td>-10.97</td>
<td>1080 (408755)</td>
<td>0</td>
</tr>
<tr>
<td>rdtok</td>
<td>450.59</td>
<td>1350 (11513)</td>
<td>482.67</td>
<td>-7.11</td>
<td>1370 (11513)</td>
<td>-0.16</td>
</tr>
<tr>
<td>warplan</td>
<td>3457.27</td>
<td>4292 (26306)</td>
<td>1566.56</td>
<td>54.68</td>
<td>5989 (19006)</td>
<td>-15.53</td>
</tr>
<tr>
<td>zebra</td>
<td>44.66</td>
<td>280 (671088746)</td>
<td>56.49</td>
<td>-26.48</td>
<td>280 (671088746)</td>
<td>0</td>
</tr>
<tr>
<td>ann</td>
<td>-</td>
<td>-</td>
<td>1220.21</td>
<td>-</td>
<td>19658 (314825)</td>
<td>-</td>
</tr>
<tr>
<td>peephole</td>
<td>1702.07</td>
<td>2210 (12148)</td>
<td>748.88</td>
<td>56</td>
<td>3329 (12845)</td>
<td>-7.71</td>
</tr>
<tr>
<td>qplan</td>
<td>-</td>
<td>-</td>
<td>2175.91</td>
<td>-</td>
<td>426519 (3827610)</td>
<td>-</td>
</tr>
<tr>
<td>witt</td>
<td>556.41</td>
<td>858 (4545564)</td>
<td>603.90</td>
<td>-8.53</td>
<td>858 (4545564)</td>
<td>0</td>
</tr>
</tbody>
</table>

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
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 call2entry 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.

<table>
<thead>
<tr>
<th></th>
<th>Sharing+Freeness</th>
<th>Clique-Sharing+Freeness</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time</td>
<td>precision</td>
</tr>
<tr>
<td>append</td>
<td>7.5</td>
<td>7 (30)</td>
</tr>
<tr>
<td>deriv</td>
<td>21</td>
<td>21 (546)</td>
</tr>
<tr>
<td>minmatrix</td>
<td>8</td>
<td>12 (694)</td>
</tr>
<tr>
<td>qsort</td>
<td>20</td>
<td>30 (1716)</td>
</tr>
<tr>
<td>query</td>
<td>11</td>
<td>22 (501)</td>
</tr>
<tr>
<td>serialize</td>
<td>57.32</td>
<td>545 (5264)</td>
</tr>
<tr>
<td>aikl</td>
<td>34</td>
<td>145 (13238)</td>
</tr>
<tr>
<td>boyer</td>
<td>380.74</td>
<td>1739 (5036)</td>
</tr>
<tr>
<td>browse</td>
<td>24</td>
<td>69 (776)</td>
</tr>
<tr>
<td>prolog_read</td>
<td>351.94</td>
<td>1050 (408634)</td>
</tr>
<tr>
<td>rdtok</td>
<td>360.44</td>
<td>1047 (11513)</td>
</tr>
<tr>
<td>warplan</td>
<td>2001</td>
<td>2436 (19644)</td>
</tr>
<tr>
<td>zebra</td>
<td>25.66</td>
<td>280 (671088746)</td>
</tr>
<tr>
<td>ann</td>
<td>1703.5</td>
<td>7811 (401220)</td>
</tr>
<tr>
<td>peephole</td>
<td>957.65</td>
<td>1475 (9941)</td>
</tr>
<tr>
<td>qplan</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>witt</td>
<td>722.14</td>
<td>813 (4545594)</td>
</tr>
</tbody>
</table>

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 call2entry 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* is less efficient than amgu+ (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.
G Precision and Efficiency Results when using Normalization as a Widening