Collapsing Closures

Xuan Li[1], Andy King[2] and Lunjin Lu[1]

[1] Oakland University, Rochester, MI 48309, USA
[2] University of Kent, Canterbury, CT2 7NF, UK

Abstract. A description in the Jacobs and Langen domain is a set of sharing groups where each sharing group is a set of program variables. The presence of a sharing group in a description indicates that all the variables in the group can be bound to terms that contain a common variable. The expressiveness of the domain, alas, is compromised by its intractability. Not only are descriptions potentially exponential in size, but abstract unification is formulated in terms of an operation, called closure under union, that is also exponential. This paper shows how abstract unification can be reformulated so that closures can be collapsed in two senses. Firstly, one closure operation can be folded into another so as to reduce the total number of closures that need to be computed. Secondly, the remaining closures can be applied to smaller descriptions. Therefore, although the operation remains exponential, the overhead of closure calculation is reduced. Experimental evaluation suggests that the cost of analysis can be substantially reduced by collapsing closures.

1 Introduction

The philosophy of abstract interpretation is to simulate the behaviour of a program without actually running it. This is accomplished by replacing each operation in the program with an abstract analogue that operates, not on the concrete data, but a description of the data. The methodology applied in abstract interpretation is first to focus on the data, that is, pin down the relationship between the concrete data and a description, and then devise abstract operations that preserve this relationship. This amounts to showing that if the input to the abstract operation describes the input to the concrete operation, then the output of the abstract operation faithfully describes the output of the concrete operation. When this methodology is applied in logic programming, the focus is usually on the operation of abstract unification since this is arguably the most complicated domain operation. The projection operation, that merely removes information from a description, is rarely a major concern.

In this paper, we revisit the projection operation of the classic set-sharing domain and we argue that the complexity of the abstract unification (*amgu*) operation can be curbed by the careful application of a reformulated projection operation. The computational problem at the heart of *amgu* is the closure under union operation [14] that operates on sharing abstractions which are constructed from sets of sharing groups. Each sharing group is, in turn, a set of program variables. Closure under union operation repeatedly unions together sets of sharing groups, drawn from a given sharing abstraction, until no new sharing group can be obtained. This operation is inherently exponential, hence the interest in different, and possibly more tractable, encodings of set-sharing [8, 10]. However, even the most creative and beautiful set-sharing encoding proposed thus far [8], does not entirely finesse the complexity of closure under union; closure under union simply manifests itself in the form of a different (and equally non-trivial) closure operator [18].

Our work was motivated by the observation that often a set-sharing analysis will calculate a series of closures that involve variables that appear in the body of a clause, only for these variables to be later eliminated when the resulting set-sharing description is restricted to those variables that occur in the head. It seems somewhat unsatisfactory that information — which is often expensive to derive — is simply thrown away. Ideally, closure operations should only be applied to variables that appear within the head of a clause. This paper shows that this ideal can be realised by reformulating abstract unification so that it can be applied in a two stage process: a phase that precedes projection (of quadratic complexity) and a phase that is applied after projection (of exponential complexity). This tactic collapses closure calculations in two important respects. Firstly, it reduces the number of variables that participate in closure calculations since there are typically fewer variables in the head of the clause than the whole of the clause. This is important because the cost of closure is related to the number of sharing groups that it operates over and this, in turn, is exponential in the number of variables in scope. Secondly, it turns out that when closure calculation is applied after projection, then the closures that arise from different unifications in the body of a clause, frequently collapse to a single closure operation. Thus, not only is the complexity of each closure operation lessened, but the total number of closure operations is also reduced.

The paper is structured as follows: Section 2 introduces the key ideas with a familiar example. Section 3 reports the main correctness results (the proofs themselves are given in the technical report [15]). Section 4 details the experimental evaluation. Section 5 reviews the related work and finally Section 6 concludes.

2 Motivating example

This section illustrates the basic ideas behind the analysis in relation to a familiar example — the append program that is listed below:

```
append(Xs,Ys,Zs) :- Xs = [], Ys = Zs.
append(Xs,Ys,Zs) :- Xs = [X|Vs], Zs = [X|Ws], append(Vs,Ys,Ws).
```

The behaviour of the program can be captured with a T-operator that is sensitive to aliasing between the arguments of atoms [4, 11]. Such an operator can be iteratively applied to obtain the following series of interpretations:

$$
\begin{array}{ll}
I_0 = \emptyset & \\
I_1 = \{\texttt{append}(Xs, Ys, Zs) \texttt{:-} \theta_1\} & \text{where } \theta_1 = \{Xs \mapsto [], Ys \mapsto Zs\} \\
I_2 = \{\texttt{append}(Xs, Ys, Zs) \texttt{:-} \theta_2\} \cup I_1 & \text{where } \theta_2 = \{Xs \mapsto [X], Zs \mapsto [X|Ys]\} \\
I_3 = \{\texttt{append}(Xs, Ys, Zs) \texttt{:-} \theta_3\} \cup I_2 & \text{where} \\
& \theta_3 = \{Xs \mapsto [X, Y], Zs \mapsto [X, Y|Ys]\} \\
\vdots & \\
I_i = \{\texttt{append}(Xs, Ys, Zs) \texttt{:-} \theta_i\} \cup I_{i-1} \text{ where} & \\
\qquad \theta_i = \{Xs \mapsto [X_1, \ldots, X_{i-1}], Zs \mapsto [X_1, \ldots, X_{i-1}|Ys]\} &
\end{array}
$$

Each interpretation I_i is a set of atoms each of which is constrained by a substitution. The limit of the sequence (and the least fixpoint of the T operator) is the interpretation $I = \{\texttt{append}(Xs, Ys, Zs) \texttt{:-} \theta_i \mid i \in \mathbb{N}\}$ which is an infinite set. It therefore follows that I cannot be finitely computed by applying iteration.

2.1 Set-sharing abstract domain

The analysis problem is to finitely compute a set-sharing abstraction of the limit I. To apply abstract interpretation to this problem, it is necessary to detail how a substitution, and more generally a set of substitutions, can be described by a set-sharing abstraction. A set-sharing abstraction for a substitution θ is constructed from a set of sharing groups: one sharing group $occ(\theta, y)$ for each variable $y \in \mathcal{V}$ drawn from the universe of variables $\mathcal{V}$. The sharing group $occ(\theta, y)$ is defined by $occ(\theta, y) = \{x \in \mathcal{V} \mid y \in var(\theta(x))\}$ and therefore contains exactly those variables which are bound by θ to terms that contain the variable y. In the particular case of θ_3 it follows that:

$$
\begin{array}{ll}
occ(\theta_3, Xs) = \emptyset & occ(\theta_3, X) = \{X, Xs, Zs\} \\
occ(\theta_3, Ys) = \{Ys, Zs\} & occ(\theta_3, Y) = \{Y, Xs, Zs\} \\
occ(\theta_3, Zs) = \emptyset & occ(\theta_3, y) = \{y\} \text{ where } y \notin \{Xs, Ys, Zs, X, Y\}
\end{array}
$$

Since the number of sharing groups for any θ is itself infinite, the abstraction map $\alpha_{\mathcal{X}}(\theta)$ is parameterised by a set of program variables $\mathcal{X}$ and defined so that $\alpha_{\mathcal{X}}(\theta) = \{occ(\theta, y) \cap \mathcal{X} \mid y \in \mathcal{V}\}$. If $\mathcal{X}$ is finite, it follows that $\alpha_{\mathcal{X}}(\theta)$ is finite. For example, if $\mathcal{X} = \{Xs, Ys, Zs\}$ then $\alpha_{\mathcal{X}}(\theta_3) = \{\emptyset, \{Xs, Zs\}, \{Ys, Zs\}\}$. The abstraction map $\alpha_{\mathcal{X}}(\theta_3)$ still records useful information: it shows that Xs and Zs can share, and similarly that Ys and Zs can share.

The domain construction is completed by lifting $\alpha_{\mathcal{X}}$ to subsets of Sub where Sub is the computational domain of substitutions. This is achieved by defining $\alpha_{\mathcal{X}} : \wp(Sub) \to Sharing_{\mathcal{X}}$ where $Sharing_{\mathcal{X}} = \wp(\wp(\mathcal{X}))$ and $\alpha_{\mathcal{X}}(\Theta) = \cup_{\theta \in \Theta} \alpha_{\mathcal{X}}(\theta)$. The concretisation map $\gamma_{\mathcal{X}} : Sharing_{\mathcal{X}} \to \wp(Sub)$ specifies which substitutions are represented by a set-sharing abstraction and is defined thus $\gamma_{\mathcal{X}}(S) = \{\theta \in Sub \mid \alpha_{\mathcal{X}}(\theta) \subseteq S\}$. (Note that an alternative definition for this domain is $Sharing_{\mathcal{X}} = \{S \mid \emptyset \in S \wedge S \subseteq \wp(\mathcal{X})\}$ since for any $\theta \in Sub$ there always exists $y \in \mathcal{V}$ such that $occ(\theta, y) \cap \mathcal{X} = \emptyset$, whence $\emptyset \in \alpha_{\mathcal{X}}(\theta)$.)

2.2 Set-sharing domain operations

The concretisation mapping $\gamma_{\mathcal{X}}$ pins down the meaning of a set-sharing abstraction and thereby provides a criteria for constructing and then judging the correctness of an abstract version of the T operator. Successive interpretations J_i generated by this operator are deemed to be correct iff for each constrained atom $\texttt{append}(Xs, Ys, Zs) \texttt{:-} \theta \in I_i$ there exists $\texttt{append}(Xs, Ys, Zs) \texttt{:-} S \in J_i$ such that $\theta \in \gamma_{\mathcal{X}}(S)$. To illustrate the problems of tractability in this operator (that stem from closure under union), the discussion focusses on the computation of the interpretation J_3; the preceding iterates are listed below:

$$
\begin{array}{ll}
J_0 = \emptyset & \\
J_1 = \{\texttt{append}(Xs, Ys, Zs) \texttt{:-} S_{J_1}\} & \text{where } S_{J_1} = \{\emptyset, \{Ys, Zs\}\} \\
J_2 = \{\texttt{append}(Xs, Ys, Zs) \texttt{:-} S_{J_2}\} \cup J_1 \text{ where} & \\
\qquad S_{J_2} = \{\emptyset, \{Xs, Zs\}, \{Xs, Ys, Zs\}, \{Ys, Zs\}\} & \\
J_3 = J_2 &
\end{array}
$$

Note that $\{Xs, Ys, Zs\} \in S_{J_2}$ but $\{Xs, Ys, Zs\} \notin \alpha_{\mathcal{X}}(\theta_i)$ for any $i \in \mathbb{N}$. This is symptomatic of the imprecision incurred by working in an abstract rather than the concrete setting. Notice too that the absence of the sharing group $\{Xs, Ys\}$ from S_{J_2} asserts that Xs and Ys can only share if there is sharing between Xs and Zs and likewise sharing between Ys and Zs.

A single application of the abstract T operator takes, as input, an interpretation J_i and produces, as output, an interpretation J_{i+1}. J_{i+1} is obtained as the union of the two interpretations: one interpretation generated by each clause in the program acting on J_i. Applying the first and second clauses to J_2 yield $\{\texttt{append}(Xs, Ys, Zs) \texttt{:-} S_{J_1}\}$ and $\{\texttt{append}(Xs, Ys, Zs) \texttt{:-} S_{J_2}\}$ respectively which, when combined, give $J_3 = J_2$. To illustrate how these interpretations are computed, consider the application of the second clause.

Computation is initiated with a set-sharing abstraction for the identity substitution ε with $\mathcal{X}$ assigned to the variables of the clause $\mathcal{X} = \{Vs, Ws, X, Xs, Ys, Zs\}$. This initial description is $S_0 = \alpha_{\mathcal{X}}(\varepsilon) = \{\emptyset, \{Vs\}, \{Ws\}, \{X\}, \{Xs\}, \{Ys\}, \{Zs\}\}$. Next, S_0 is progressively instantiated by firstly, simulating the unification $Xs = [X|Vs]$ with input S_0 to obtain output S_1; then secondly, solving $Zs = [X|Ws]$ in the presence of S_1 to give S_2; then thirdly, adding the bindings imposed by the body atom $\texttt{append}(Vs, Ys, Ws)$ to S_2 to obtain a description that characterises the whole clause. Each of these steps is outlined below:

- The abstract unification operation $S_1 = amgu(Xs, [X|Vs], S_0)$ of Jacobs and Langen [14] provides a way of simulating concrete unification with set-sharing abstractions. The algorithm satisfies the correctness criteria that if $\theta_0 \in \gamma_{\mathcal{X}}(S_0)$ and $\delta \in mgu(\theta_0(X), \theta_0([X|Vs]))$ then $\theta_1 = \delta \circ \theta_0 \in \gamma_{\mathcal{X}}(S_1)$ where $mgu(t_1, t_2)$ denotes the set of most general unifiers for the terms t_1 and t_2. The algorithm is formulated in terms of three auxiliary operations: relevance operation $rel(o, S)$ where o is any syntactic object, the cross union $T_1 \uplus T_2$ of two descriptions T_1 and T_2, and closure under union $cl(S)$. The relevance mapping is defined by $rel(o, S) = \{G \in S \mid var(o) \cap G \neq \emptyset\}$

where $var(o)$ is the set of variables contained in the syntactic object o. The mapping $rel(o, S)$ thus returns those sharing groups G of S which share a variable with o. Cross union is defined by $T_1 \uplus T_2 = \{G \cup H \mid G \in T_1 \wedge H \in T_2\}$ and thus computes the union of all the pairs of sharing groups in the cross-product $T_1 \times T_2$. The closure $cl(S)$ is defined as the least superset of S satisfies the closure property that if $G \in cl(S)$ and $H \in cl(S)$ then $G \cup H \in cl(S)$. With these operations in place, abstract unification can be defined thus $amgu(t_1, t_2, S) = (S \setminus (T_1 \cup T_2)) \cup cl(T_1 \uplus T_2)$ where $T_i = rel(t_i, S)$. (This definition is actually a reformulation [10] of the classic definition [14] that is better suited to illustrate our purposes). In the particular case of $S_1 = amgu(Xs, [X|Vs], S_0)$ it follows that:

$$\begin{aligned} T_1 = rel(Xs, S_0) &= \{\{Xs\}\} \\ T_2 = rel([X|Vs], S_0) &= \{\{Vs\}, \{X\}\} \\ T_1 \uplus T_2 &= \{\{Vs, Xs\}, \{X, Xs\}\} \\ cl(T_1 \uplus T_2) &= \{\{Vs, Xs\}, \{Vs, X, Xs\}, \{X, Xs\}\} \end{aligned}$$

and hence $S_1 = \{\emptyset, \{Vs, Xs\}, \{Vs, X, Xs\}, \{Ws\}, \{X, Xs\}, \{Ys\}, \{Zs\}\}$.

- Repeating this process for $S_2 = amgu(Zs, [X|Ws], S_1)$ yields $S_2 = \{\emptyset, \{Vs, Ws, X, Xs, Zs\}, \{Vs, X, Xs, Zs\}, \{Vs, Xs\}, \{Ws, X, Xs, Zs\}, \{Ws, Zs\}, \{X, Xs, Zs\}, \{Ys\}\}$.
- Next, the bindings imposed by the body atom $\texttt{append}(Vs, Ys, Ws)$ need to be added to S_2. The technical problem is that these bindings are recorded in J_2, not in terms of $\texttt{append}(Vs, Ys, Ws)$, but in terms of a renaming of the atom, that is, $\texttt{append}(Xs, Ys, Zs)$. (This problem manifests itself because, in theory, interpretations are defined as sets of constrained atoms where each constrained atom represents a set of constrained atoms that are equivalent under variable renaming [4, 11].) This problem is resolved, in practise, by extending S_2 to give $S_3 = S_2 \cup \{\{\underline{Xs}, \underline{Zs}\}, \{\underline{Ys}, \underline{Zs}\}, \{\underline{Xs}, \underline{Ys}, \underline{Zs}\}\}$ where $\underline{Xs}$, $\underline{Ys}$ and $\underline{Zs}$ are fresh variables. Then a series of abstract unifications are applied which are interleaved with projection operations to incrementally remove the freshly introduced variables. This strategy proceeds thus: $S_4 = amgu(Vs, \underline{Xs}, S_3)$, $S_5 = S_4 \restriction (\mathcal{X} \setminus \{\underline{Xs}\})$, $S_6 = amgu(Ys, \underline{Ys}, S_5)$, $S_7 = S_6 \restriction (\mathcal{X} \setminus \{\underline{Ys}\})$, $S_8 = amgu(Ws, \underline{Zs}, S_7)$ and $S_9 = S_8 \restriction (\mathcal{X} \setminus \{\underline{Zs}\})$. The projection operation $\restriction$ is defined $S \restriction Y = \{G \cap Y \mid G \in S\}$ and eliminates all variables from S other than those drawn from Y. Projection preserves correctness since $\gamma_{\mathcal{X}}(S) \subseteq \gamma_{\mathcal{X}}(S \restriction Y)$. This strategy computes the following descriptions for $S_4, \ldots, S_9$:

$$\begin{aligned} S_4 = \{&\emptyset, \{\underline{Xs}, \underline{Ys}, \underline{Zs}, Vs, Ws, X, Xs, Zs\}, \{\underline{Xs}, \underline{Ys}, \underline{Zs}, Vs, X, Xs, Zs\}, \\ &\{\underline{Xs}, \underline{Ys}, \underline{Zs}, Vs, Xs\}, \{\underline{Xs}, \underline{Zs}, Vs, Ws, X, Xs, Zs\}, \\ &\{\underline{Xs}, \underline{Zs}, Vs, X, Xs, Zs\}, \{\underline{Xs}, \underline{Zs}, Vs, Xs\}, \{\underline{Ys}, \underline{Zs}\}, \\ &\{Ws, X, Xs, Zs\}, \{Ws, Zs\}, \{X, Xs, Zs\}, \{Ys\}\} \\ S_5 = \{&\emptyset, \{\underline{Ys}, \underline{Zs}, Vs, Ws, X, Xs, Zs\}, \{\underline{Ys}, \underline{Zs}, Vs, X, Xs, Zs\}, \\ &\{\underline{Ys}, \underline{Zs}, Vs, xs\}, \{\underline{Zs}, Vs, Ws, X, Xs, Zs\} \\ &\{\underline{Zs}, Vs, X, Xs, Zs\}, \{\underline{Zs}, Vs, Xs\}, \{\underline{Ys}, \underline{Zs}\}, \\ &\{Ws, X, Xs, Zs\}, \{Ws, Zs\}, \{X, Xs, Zs\}, \{Ys\}\} \end{aligned}$$

$$
\begin{aligned}
&\vdots\\
S_8 = \{&\emptyset, \{\underline{Zs}, Vs, Ws, X, Xs, Ys, Zs\}, \{\underline{Zs}, Vs, Ws, Xs, Ys, Zs\},\\
&\{\underline{Zs}, Vs, Ws, X, Xs, Zs\}, \{\underline{Zs}, Vs, Ws, Xs, Zs\},\\
&\{\underline{Zs}, Ws, X, Xs, Ys, Zs\}, \{\underline{Zs}, Ws, Ys, Zs\}, \{X, Xs, Zs\}\}\\
S_9 = \{&\emptyset, \{Vs, Ws, X, Xs, Ys, Zs\}, \{Vs, Ws, Xs, Ys, Zs\},\\
&\{Vs, Ws, X, Xs, Zs\}, \{Vs, Ws, Xs, Zs\},\\
&\{Ws, X, Xs, Ys, Zs\}, \{Ws, Ys, Zs\}, \{X, Xs, Zs\}\}
\end{aligned}
$$

It should be noted that in these steps, abstract matching can be substituted for abstract unification. This can improve both the precision and the efficiency [13] but does not reduce the overall number of closures.

The description S_9 expresses the bindings imposed on the variables of the whole clause as a result of the unification and the body atom. The restriction $S_{10} = S_9 \restriction \{Xs, Ys, Zs\} = \{\emptyset,\ \{Xs, Zs\},\ \{Xs, Ys, Zs\},\ \{Ys, Zs\}\}$ then describes these bindings solely in term of the variables in the head. Since S_{10} coincides with S_{J_2} it follows that a fixpoint has been reached and therefore J_2 faithfully describes the limit interpretation I. The observation that motivated this work is that this application of the abstract T operator alone, requires 5 closure calculations to compute S_1, S_2, S_4, S_6 and S_8 each of which are non-trivial descriptions that are defined over at least 6 variables. Yet the objective is merely to compute S_{10} which is necessarily defined over just 3 variables.

2.3 Reformulating set-sharing domain operations

One solution to the problem of calculating closures is to not compute them immediately, but defer evaluation until a more propitious moment, that is, when the descriptions contain less variables. Consider again the definition $amgu(t_1, t_2, S) = (S \setminus (T_1 \cup T_2)) \cup cl(T_1 \uplus T_2)$. Instead of computing $cl(T_1 \uplus T_2)$, the strategy is to tag all the groups within $T_1 \uplus T_2$ with an identifier — a unique number — that identifies those groups that participate in a particular closure. The tags are retained until head projection whereupon they are used to activate closure calculation. Then the tags are discarded. This idea leads to an abstract unification operator that is defined $amgu'(t_1, t_2, n, S) = (S \setminus (T_1 \cup T_2)) \cup tag(T_1 \uplus' T_2, n)$ where the descriptions S, T_1, T_2 are enriched with tagging information and n is a new tag that distinguishes those groups generated from $T_1 \uplus' T_2$. Formally, descriptions are drawn from a domain $Sharing'_{\mathcal{X}} = \wp(\wp(\mathcal{X}) \times \wp(\mathbb{N}))$ since, in general, a sharing group can own several tags. (Elements of this domain are only used for intermediate calculations and the infinite nature of $Sharing'_{\mathcal{X}}$ does not compromise termination.) The tagging operation $tag(S, n)$ inserts a tag n into each group in S and thus $tag(S, n) = \{\langle G, N \cup \{n\}\rangle \mid \langle G, N\rangle \in S\}$. Over this new domain, cross union is redefined $T_1 \uplus' T_2 = \{\langle G \cup H, N \cup M\rangle \mid \langle G, N\rangle \in T_1 \wedge \langle H, M\rangle \in T_2\}$. Note that rel can be used without adaption.

Now reconsider the computation of J_3 using the second clause. The initial description is again $S'_0 = \{\emptyset,\ \{Vs\},\ \{Ws\},\ \{X\},\ \{Xs\},\ \{Ys\},\ \{Zs\}\}$ but with the interpretation that an untagged group G is actually syntactic sugar for a pair $\langle G, \emptyset \rangle$ that is equipped with an empty set of tags. Then $S'_0 \in \mathit{Sharing}'_{\mathcal{X}}$. Each application of abstract unification is required to introduce a fresh identifier and these are chosen to be 1 and 2 when computing S'_1 and S'_2. Computation unfolds as follows:

– Applying $S'_1 = \mathit{amgu}'(Xs, [X|Vs], 1, S'_0)$ it follows that:

$$\begin{aligned} T_1 = \mathit{rel}(Xs, S'_0) &= \{\{Xs\}\} \\ T_2 = \mathit{rel}([X|Vs], S'_0) &= \{\{Vs\}, \{X\}\} \\ T_1 \uplus' T_2 &= \{\{Vs, Xs\}, \{X, Xs\}\} \\ \mathit{tag}(T_1 \uplus' T_2, 1) &= \{\langle \{Vs, Xs\}, \{1\} \rangle, \langle \{X, Xs\}, \{1\} \rangle\} \end{aligned}$$

hence $S'_1 = \{\emptyset, \langle \{Vs, Xs\}, \{1\} \rangle, \{Ws\}, \langle \{X, Xs\}, \{1\} \rangle, \{Ys\}, \{Zs\}\}$.

– Repeating this strategy for $S'_2 = \mathit{amgu}'(Zs, [X|Ws], 2, S'_1)$ yields:

$$\begin{aligned} T_1 = \mathit{rel}(Zs, S'_1) &= \{\{Zs\}\} \\ T_2 = \mathit{rel}([X|Ws], S'_1) &= \{\{Ws\}, \langle \{X, Xs\}, \{1\} \rangle\} \\ T_1 \uplus' T_2 &= \{\{Ws, Zs\}, \langle \{X, Xs, Zs\}, \{1\} \rangle\} \\ \mathit{tag}(T_1 \uplus' T_2, 2) &= \{\langle \{Ws, Zs\}, \{2\} \rangle, \langle \{X, Xs, Zs\}, \{1, 2\} \rangle\} \end{aligned}$$

thus $S'_2 = \{\emptyset, \langle \{Vs, Xs\}, \{1\} \rangle, \langle \{Ws, Zs\}, \{2\} \rangle, \langle \{X, Xs, Zs\}, \{1, 2\} \rangle, \{Ys\}\}$. The identifiers 1 and 2 indicate which groups participate in which closures. The group $\{X, Xs, Zs\}$ is tagged with $\{1, 2\}$ since it is involved in both closures. Note that, unlike before, $|S'_2| < |S'_1| < |S'_0|$.

– The bindings from the body atom are added by again extending S'_2 to $S'_3 = S'_2 \cup \{\{\underline{Xs}, \underline{Zs}\}, \{\underline{Ys}, \underline{Zs}\}, \{\underline{Xs}, \underline{Ys}, \underline{Zs}\}\}$. The interwoven unification and projection steps are modified by introducing fresh identifers and by redefining projection so that $S \restriction' Y = \{\langle G \cap Y, N \rangle \mid \langle G, N \rangle \in S\}$. Hence $S'_4 = \mathit{amgu}'(Vs, \underline{Xs}, 3, S'_3)$, $S'_5 = S'_4 \restriction' (\mathcal{X} \setminus \underline{Xs})$, $S'_6 = \mathit{amgu}(Ys, \underline{Ys}, 4, S'_5)$, $S'_7 = S'_6 \restriction' (\mathcal{X} \setminus \underline{Ys})$, $S'_8 = \mathit{amgu}'(Ws, \underline{Zs}, 5, S'_7)$ and $S'_9 = S'_8 \restriction' (\mathcal{X} \setminus \underline{Zs})$ which generates the following sequence of descriptions:

$$\begin{aligned} S'_4 = \{&\emptyset, \langle \{\underline{Xs}, \underline{Zs}, Vs, Xs\}, \{1, 3\} \rangle, \langle \{\underline{Xs}, \underline{Ys}, \underline{Zs}, Vs, Xs\}, \{1, 3\} \rangle, \\ &\{\underline{Ys}, \underline{Zs}\}, \langle \{Ws, Zs\}, \{2\} \rangle, \langle \{X, Xs, Zs\}, \{1, 2\} \rangle, \{Ys\}\} \\ S'_5 = \{&\emptyset, \langle \{\underline{Zs}, Vs, Xs\}, \{1, 3\} \rangle, \langle \{\underline{Ys}, \underline{Zs}, Vs, Xs\}, \{1, 3\} \rangle, \\ &\{\underline{Ys}, \underline{Zs}\}, \langle \{Ws, Zs\}, \{2\} \rangle, \langle \{X, Xs, Zs\}, \{1, 2\} \rangle, \{Ys\}\} \\ S'_6 = \{&\emptyset, \langle \{\underline{Zs}, Vs, Xs\}, \{1, 3\} \rangle, \langle \{\underline{Ys}, \underline{Zs}, Vs, Xs, Ys\}, \{1, 3, 4\} \rangle, \\ &\langle \{\underline{Ys}, \underline{Zs}, Ys\}, \{4\} \rangle, \langle \{Ws, Zs\}, \{2\} \rangle, \langle \{X, Xs, Zs\}, \{1, 2\} \rangle\} \\ S'_7 = \{&\emptyset, \langle \{\underline{Zs}, Vs, Xs\}, \{1, 3\} \rangle, \langle \{\underline{Zs}, Vs, Xs, Ys\}, \{1, 3, 4\} \rangle, \\ &\langle \{\underline{Zs}, Ys\}, \{4\} \rangle, \langle \{Ws, Zs\}, \{2\} \rangle, \langle \{X, Xs, Zs\}, \{1, 2\} \rangle\} \\ S'_8 = \{&\emptyset, \langle \{\underline{Zs}, Vs, Xs, Ws, Zs\}, \{1, 2, 3, 5\} \rangle, \langle \{\underline{Zs}, Ys, Ws, Zs\}, \{2, 4, 5\} \rangle, \\ &\langle \{\underline{Zs}, Vs, Xs, Ys, Ws, Zs\}, \{1, 2, 3, 4, 5\} \rangle, \langle \{X, Xs, Zs\}, \{1, 2\} \rangle\} \\ S'_9 = \{&\emptyset, \langle \{Vs, Xs, Ws, Zs\}, \{1, 2, 3, 5\} \rangle, \langle \{Ys, Ws, Zs\}, \{2, 4, 5\} \rangle, \\ &\langle \{Vs, Xs, Ys, Ws, Zs\}, \{1, 2, 3, 4, 5\} \rangle, \langle \{X, Xs, Zs\}, \{1, 2\} \rangle\} \end{aligned}$$

Computing $S'_{10} = S'_9 \restriction' \mathcal{X} \setminus \{Xs, Ys, Zs\}$ restricts S'_9 to those variables in the head of the clause which yields the description $S'_{10} = \{\emptyset, \langle \{Xs, Zs\}, \{1, 2, 3, 5\} \rangle, \langle \{Xs, Zs\}, \{1, 2, 3, 4, 5\} \rangle, \langle \{Ys, Zs\}, \{2, 4, 5\} \rangle, \langle \{Xs, Zs\}, \{1, 2\} \rangle\}$. Now, only the pending closures remain to be evaluated.

2.4 Evaluating pending closures

The pending closures can be activated by applying a closure operation $cl(S,i)$ for each identifier i contained within S. The operation $cl(S,i)$ is a form of closure under union that is sensitive to i in the sense that it only merges two pairs $\langle G, I\rangle$ and $\langle H, J\rangle$ when both I and J contain the identifier i. The merge of these pairs is defined as $\langle G \cup H, I \cup J\rangle$ so that merge combines both the sharing groups and the tagging sets. Like classic closure, $cl(S,i)$ performs repeatedly merging until no new element can be generated. To express this process, $cl(S,i)$ is formulated in terms of $cl_K(S) = S \cup \{\langle G \cup H, I \cup J\rangle \mid \{\langle G, I\rangle, \langle H, J\rangle\} \subseteq S \wedge I \cap J \cap K \neq \emptyset\}$ where $K \subseteq \mathbb{N}$. Then $cl(S,i)$ can be defined as the limit of a sequence $cl(S,i) = \cup_{j=0}^{\infty} S_j$ where $S_0 = S$, $S_j = cl_{\{i\}}(S_{j-1})$. For instance, continuing with the example $S'_{10} = \{\emptyset, \langle\{Xs, Zs\}, \{1,2,3,5\}\rangle, \langle\{Xs, Zs\}, \{1,2,3,4,5\}\rangle, \langle\{Ys, Zs\}, \{2,4,5\}\rangle, \langle\{Xs, Zs\}, \{1,2\}\rangle\}$, then $cl_{\{4\}}(S'_{10}) = S'_{10} \cup \{\langle\{Xs, Ys, Zs\}, \{1,2,3,4,5\}\rangle\}$. In fact, in this case, no further applications of $cl_{\{4\}}$ are required for before convergence is obtained and $cl(S'_{10}, 4) = cl_{\{4\}}(S'_{10})$. In general, $cl(S,i)$ will only need to be applied a finite number of times before convergence is reached.

Applying the closure operator $cl(S,i)$ is sufficient to evaluate the closure that is delimited by i; a single application of $cl(S,i)$ is not sufficient to active all the pending closures. Therefore $cl(S,i)$ is itself iteratively applied by computing the sequence of descriptions $T_0 = S$ and $T_i = cl(T_{i-1}, i)$ that culminates in T_n where n is understood to be the maximal identifier of S. Henceforth, let $cl'(S) = T_n$. Returning to the running example, $cl'(S'_{10}) = \{\emptyset, \langle\{Xs, Ys, Zs\}, \{1,2,4,5\}\rangle, \{Xs, Ys, Zs\}, \{1,2,3,4,5\}\rangle, \langle\{Xs, Zs\}, \{1,2,3,5\}\rangle, \langle\{Xs, Zs\}, \{1,2,3,4,5\}\rangle, \langle\{Ys, Zs\}, \{2,4,5\}\rangle, \langle\{Xs, Zs\}, \{1,2\}\rangle\}$.

2.5 Collapsing closures: the duplicated group rule

The remaining tags can be eliminated with $untag(S) = \{G \mid \langle G, N\rangle \in S\}$. Composing and then applying these two operations to S'_{10} gives $untag(cl'(S'_{10})) = \{\{Xs, Ys, Zs\}, \{Xs, Zs\}, \{Ys, Zs\}\}$ as desired. Although this is an advance — closure calculations have been collapsed to range over the variables of the head rather than the clause — it does not exploit the fact that the closure calculations for different identifiers can be collapsed into a single computation.

To see this, observe that S'_{10} contains three pairs $\langle G, N_1\rangle$, $\langle G, N_2\rangle$ and $\langle G, N_3\rangle$ where $G = \{Xs, Zs\}$, $N_1 = \{1,2\}$, $N_2 = \{1,2,3,5\}$ and $N_2 = \{1,2,3,4,5\}$. The pairs $\langle G, N_1\rangle$ and $\langle G, N_2\rangle$ are redundant since $N_1 \subseteq N_3$ and $N_2 \subseteq N_3$. Removing these pairs from S'_{10} yields the description $S'_{11} = \{\emptyset, \langle\{Xs, Zs\}, \{1,2,3,4,5\}\rangle, \langle\{Ys, Zs\}, \{2,4,5\}\rangle\}$ which compromises neither correctness nor precision since $untag(cl'(S'_{10})) = untag(cl'(S'_{11}))$. Actually, the underlying principle is not that of eliminating a pair that shares a common group with another pair whose identifiers subsume it, but rather that all pairs which share a common group can merged into single pair. In this particular case of S'_{10}, the three pairs can be combined into the single pair $\langle G, N_1 \cup N_2 \cup N_3\rangle$ that subsumes them all; S'_{10} merely illustrates the special case of when $N_1 \cup N_2 \cup N_3 = N_3$. In general, if $\{\langle G, N\rangle, \langle G, M\rangle\} \subseteq S$, $N \cap M \neq \emptyset$ and $S' = (S \setminus \{\langle G, N\rangle, \langle G, M\rangle\}) \cup \{\langle G, N \cup M\rangle\}$

then $untag(cl'(S)) = untag(cl'(S'))$. Henceforth this equivalence will be referred to as the duplicated group rule.

2.6 Collapsing closures: the uniqueness rule

Further reductions can be applied. Since the identifiers 1 and 3 occur in just one pair, it follows that these identifiers can be immediately removed from S'_{11} to obtain $S'_{12} = \{\emptyset, \langle\{Xs, Zs\}, \{2, 4, 5\}\rangle, \langle\{Ys, Zs\}, \{2, 4, 5\}\rangle\}$ whilst preserving the relationship $untag(cl'(S'_{10})) = untag(cl'(S'_{12}))$. This strategy of removing those identifiers that occur singly will henceforth be called the uniqueness rule.

2.7 Collapsing closures: the covering rule

Moreover, identifier 2 always occurs within a set of identifiers that also contains 4. In this sense 4 is said to cover 2. The value of this concept is that if one identifier is covered by another, then the first identifier is redundant. Since 5 covers both 2 and 4, then both 2 and 4 are redundant and can be removed from S'_{12} to obtain $S'_{13} = \{\emptyset, \langle\{Xs, Zs\}, \{5\}\rangle, \langle\{Ys, Zs\}, \{5\}\rangle\}$ whilst again preserving $untag(cl'(S'_{10})) = untag(cl'(S'_{13}))$. This form of reduction will be called the covering rule. The key point is that by applying these three rules S'_{10} can be simplified to S'_{13} which only requires one application $cl(S'_{13}, 5)$ followed by *untag* to evaluate the remaining closure. This results in $untag(cl(S'_{13}, 5)) = \{\emptyset, \{Xs, Zs\}, \{Ys, Zs\}, \{Xs, Ys, Zs\}\}$ as required; the same result as classic set-sharing is derived but with a significant reduction in closure calculation.

3 Equivalence results

This section reports some new equivalence results which show that neither closure collapsing nor delaying closure evaluation incur a precision loss over classic set-sharing. The results are summarised in sections 3.1 and 3.2 respectively.

3.1 Equivalence rules for collapsing closures

The closure operator $cl'(S)$ applies $cl(S, i)$ for each identifier i to evaluate each pending closure in turn. This offers a sequential model for computing $cl'(S)$. The following result provides an alternative parallel model for closure evaluation.

Proposition 1. $cl'(S) = \cup_{i=0}^{\infty} S_i$ where $S_0 = S$ and $S_{i+1} = cl_{\mathbb{N}}(S_i)$.

The force of this result is twofold. Firstly, it can save passing over S multiply, once for each identifier. Secondly, it provides a way for arguing correctness of the three collapsing rules. For pedagogical reasons, these rules were introduced in terms of the sequential model of $cl'(S)$ evaluation, yet they are still applicable in the parallel setting. This is because the cost of closure calculation is dominated by the cost of the underlying set operations and therefore any reduction in the number or size of these sets is useful. For completeness, the rules are formally stated below, complete with a counter-example which illustrates the need for the $M \cap N \neq \emptyset$ condition in the duplicated group rule.

Proposition 2 (duplicated group rule). Suppose $\langle G, M\rangle \in S$, $\langle G, N\rangle \in S$ and $M \cap N \neq \emptyset$. Then $untag(cl'(S)) = untag(cl'(S'))$ where:

$$S' = (S \setminus \{\langle G, M\rangle, \langle G, N\rangle\}) \cup \{\langle G, M \cup N\rangle\}$$

Example 1. The following values of S and S' illustrate the necessity of the $N \cap M \neq \emptyset$ condition in the duplicated group rule. This condition bars the pairs $\langle\{y\}, \{1\}\rangle$ and $\langle\{y\}, \{2\}\rangle$ within S from being merged to obtain S'. Merging looses equivalence since $\{x, y, z\} \in untag(cl'(S'_3))$ but $\{x, y, z\} \notin untag(cl'(S_3))$.

$$\begin{aligned}
S &= \{\langle\{x\},\{1\}\rangle, \langle\{y\},\{1\}\rangle, \langle\{y\},\{2\}\rangle, \langle\{z\},\{2\}\rangle\}\\
S_1 = cl(S,1) &= \{\langle\{x\},\{1\}\rangle, \langle\{x,y\},\{1\}\rangle, \langle\{y\},\{1\}\rangle, \langle\{y\},\{2\}\rangle, \langle\{z\},\{2\}\rangle\}\\
S_2 = cl(S_1,2) &= \{\langle\{x\},\{1\}\rangle, \langle\{x,y\},\{1\}\rangle, \langle\{y\},\{1\}\rangle, \langle\{y\},\{2\}\rangle, \langle\{y,z\},\{2\}\rangle,\\
&\quad\ \langle\{z\},\{2\}\rangle\}\\
S_3 = untag(S_2) &= \{\{x\}, \{x,y\}, \{y\}, \{y,z\}, \{z\}\}
\end{aligned}$$

$$\begin{aligned}
S' &= \{\langle\{x\},\{1\}\rangle, \langle\{y\},\{1,2\}\rangle, \langle\{z\},\{2\}\rangle\}\\
S'_1 = cl(S',1) &= \{\langle\{x\},\{1\}\rangle, \langle\{x,y\},\{1,2\}\rangle, \langle\{y\},\{1,2\}\rangle, \langle\{z\},\{2\}\rangle\}\\
S'_2 = cl(S'_2,2) &= \{\langle\{x\},\{1\}\rangle, \langle\{x,y\},\{1,2\}\rangle, \langle\{x,y,z\},\{1,2\}\rangle, \langle\{y\},\{1,2\}\rangle,\\
&\quad\ \langle\{y,z\},\{1,2\}\rangle, \langle\{z\},\{2\}\rangle\}\\
S'_3 = untag(S'_2) &= \{\{x\}, \{x,y\}, \{x,y,z\}, \{y\}, \{y,z\}, \{z\}\}
\end{aligned}$$

Proposition 3 (uniqueness rule). Suppose $\langle G, N\rangle$ is the only element of S for which $n \in N$. Then $untag(cl'(S)) = untag(cl'(S'))$ where:

$$S' = (S \setminus \{\langle G, N\rangle\}) \cup \{\langle G, N \setminus \{n\}\rangle\}$$

Proposition 4 (covering rule). Suppose that $n \neq m$ and that if $n \in N$ and $\langle G, N\rangle \in S$ then $m \in N$. Then $untag(cl'(S)) = untag(cl'(S'))$ where:

$$S' = \{\langle G, N \setminus \{n\}\rangle \mid \langle G, N\rangle \in S\}$$

3.2 Equivalence of pending closures

This paper proposes the general strategy of delaying closure evaluation until a time when the pending closures can be evaluated over fewer variables. This technique of procrastination is founded on lemma 1. The first result stated in the lemma explains how $amgu'$ is basically a reformulation of $amgu$ that postpones the closure calculation (providing its input is closed); in the $amgu$ closure arises within the operator whereas in the $amgu'$ closure is applied after the operator. The second result states a circumstance in which a closure can be avoided; that it is not necessary to apply $amgu'$ to an S' that is closed, ie. it is not necessary to compute $cl'(S')$, providing that the result of the $amgu'$ is then closed. The strength of these two results is that they can be composed to show that only one single closure need be applied at the end of a sequence of $amgu'$ applications. This leads to the main result — theorem 1 — which is stated immediately after the lemma. The condition in the lemma on i asserts that i is a fresh tag.

Lemma 1. Suppose that $i \notin I$ for all $\langle G, I\rangle \in S'$. Then

- if $cl'(S') = S'$ then $untag(cl(amgu'(s,t,i,S'),i)) = amgu(s,t,untag(S'))$
- $cl'(amgu'(s,t,i,S')) = cl'(amgu'(s,t,i,cl'(S')))$

Example 2. Let $t = f(y,z)$ and $S' = \{\langle\{v\},\{1\}\rangle, \langle\{x\},\emptyset\rangle, \langle\{y\},\{1\}\rangle, \langle\{z\},\emptyset\rangle\}$. Then $cl'(S') = \{\langle\{v\},\{1\}\rangle, \langle\{v,y\},\{1\}\rangle, \langle\{x\},\emptyset\rangle, \langle\{y\},\{1\}\rangle, \langle\{z\},\emptyset\rangle\}$ and

$$\begin{aligned}
amgu(x,t,untag(cl'(S'))) &= \{\{v\},\{v,x,y\},\{v,x,y,z\},\{x,y\},\{x,y,z\},\{x,z\}\} \\
amgu'(x,t,2,cl'(S')) &= \{\langle\{v\},\{1\}\rangle, \langle\{v,x,y\},\{1,2\}\rangle, \\
&\quad\ \langle\{x,y\},\{1,2\}\rangle, \langle\{x,z\},\{2\}\rangle\} \\
cl(amgu'(x,t,2,cl'(S')),2) &= \{\langle\{v\},\{1\}\rangle, \langle\{v,x,y\},\{1,2\}\rangle, \langle\{v,x,y,z\},\{1,2\}\rangle, \\
&\quad\ \langle\{x,y\},\{1,2\}\rangle, \langle\{x,y,z\},\{1,2\}\rangle, \langle\{x,z\},\{2\}\rangle\} \\
cl'(amgu'(x,t,2,cl'(S'))) &= \{\langle\{v\},\{1\}\rangle, \langle\{v,x,y\},\{1,2\}\rangle, \langle\{v,x,y,z\},\{1,2\}\rangle, \\
&\quad\ \langle\{x,y\},\{1,2\}\rangle, \langle\{x,y,z\},\{1,2\}\rangle, \langle\{x,z\},\{2\}\rangle\} \\
amgu'(x,t,2,S') &= \{\langle\{v\},\{1\}\rangle, \langle\{x,y\},\{1,2\}\rangle, \langle\{x,z\},\{2\}\rangle\} \\
cl'(amgu'(x,t,2,S')) &= \{\langle\{v\},\{1\}\rangle, \langle\{v,x,y\},\{1,2\}\rangle, \langle\{v,x,y,z\},\{1,2\}\rangle, \\
&\quad\ \langle\{x,y\},\{1,2\}\rangle, \langle\{x,y,z\},\{1,2\}\rangle, \langle\{x,z\},\{2\}\rangle\}
\end{aligned}$$

Theorem 1. Let $s_1 = t_1, \ldots, s_n = t_n$ be a sequence of syntactic equations and

$$\begin{aligned}
S_0 &= S & S_i &= amgu(s_i, t_i, S_{i-1}) \\
S'_0 &= \{\langle G, \emptyset\rangle \mid G \in S\} & S'_i &= amgu'(s_i, t_i, i, S'_{i-1})
\end{aligned}$$

Then $untag(cl'(S'_n)) = S_n$.

Example 3. Consider the equations $s_1 = t_1$ and $s_2 = t_2$ where $(s_1 = t_1) = (w = f(x,y))$ and $(s_2 = t_2) = (x = z)$, $S_0 = \alpha_{\mathcal{X}}(\varepsilon)$ and $\mathcal{X} = \{w,x,y,z\}$. Then

$$\begin{aligned}
S_0 &= \{\{w\},\{x\},\{y\},\{z\}\} & S'_0 &= \{\langle\{w\},\emptyset\rangle, \langle\{x\},\emptyset\rangle, \langle\{y\},\emptyset\rangle, \langle\{z\},\emptyset\rangle\} \\
S_1 &= \{\{w,x\},\{w,y\},\{w,x,y\},\{z\}\} & S'_1 &= \{\langle\{w,x\},\{1\}\rangle, \langle\{w,y\},\{1\}\rangle, \langle\{z\},\emptyset\rangle\} \\
S_2 &= \{\{w,x,y,z\},\{w,x,z\},\{w,y\}\} & S'_2 &= \{\langle\{w,x,z\},\{1,2\}\rangle, \langle\{w,y\},\{1\}\rangle\}
\end{aligned}$$

Therefore $cl'(S'_2) = \{\langle\{w,x,y,z\},\{1,2\}\rangle, \langle\{w,x,z\},\{1,2\}\rangle, \langle\{w,y\},\{1\}\rangle\}$ and $untag(cl'(S'_2)) = \{\{w,x,y,z\},\{w,x,z\},\{w,y\}\} = S_2$ as theorem 1 predicts.

Theorem 1 can be taken further to obtain corollary 1 by exploiting the property that projection $\restriction'$ distributes over closure cl'. The force of this result — that is formally stated in proposition 5 — is that if the set of variables Y is smaller than $var(S')$ then $cl'(S' \restriction' Y)$ is cheaper to compute than $cl'(S') \restriction Y$.

Proposition 5. $cl'(S') \restriction' Y = cl'(S' \restriction' Y)$ where $Y \subseteq \mathcal{V}$

Corollary 1. Let $s_1 = t_1, \ldots, s_n = t_n$ be a sequence of syntactic equations and

$$\begin{aligned}
S_0 &= S & S_i &= amgu(s_i, t_i, S_{i-1}) \\
S'_0 &= \{\langle G, \emptyset\rangle \mid G \in S\} & S'_i &= amgu'(s_i, t_i, i, S'_{i-1})
\end{aligned}$$

Then $untag(cl'(S'_n \restriction' Y')) = S_n \restriction Y$.

Example 4. Continuing with example 3, suppose that the objective is to compute $S_2 \restriction Y$ where $Y = \{x,y\}$. The corollary asserts $S_2 \restriction Y = untag(cl'(S'_2 \restriction' Y))$ and $cl'(\{\langle\{x\},\{1,2\}\rangle, \langle\{y\},\{1\}\rangle\}) = \{\langle\{x\},\{1,2\}\rangle, \langle\{x,y\},\{1,2\}\rangle, \langle\{y\},\{1\}\rangle\}$ whence $untag(cl'(S'_2 \restriction' Y)) = \{\{x\},\{x,y\},\{y\}\}$. Indeed, from example 3 it can be seen that $S_2 \restriction Y = \{\{x\},\{x,y\},\{y\}\}$.

4 Implementation

In order to assess the usefulness of collapsing closures, both classic set-sharing [14] and set-sharing with closure collapsing have been implemented in Sicstus 3.8.6. To obtain a credible comparison, both techniques were implemented, wherever possible, with the same data-structures and code. Both forms of set-sharing were integrated into a goal-independent (bottom-up) fixpoint engine and a goal-dependent (bottom-up) analyser that applied a magic transform [12] to collect call and answer patterns. Both frameworks track set-sharing alone, ie., they do not trace sharing as one component of a product domain [9]. This was partly to isolate the impact of collapsing on set-sharing from the effect of other domains (quantifying these interactions even for more conventional forms of sharing is a long study within itself [3]) and partly as an experiment in worst-case sharing. The rationale was that if closure collapsing had little impact in this scenario, then it would not warrant investigating how the technique can be composed with other domains. Both frameworks also computed strongly connected components (SCCs) of the call-graph so as to stratify the fixpoint into a series of fixpoints that stabilise on one SCC before moving onto another [12].

Table 1 summaries the four analysis combinations — goal-dependent versus goal-independent and classic set-sharing versus set-sharing with collapsing — for the series of common benchmark programs. As a sanity check of the theory, success patterns derived by two forms of goal-independent analysis were verified to be equivalent; likewise the call and answer patterns computed with collapsing coincided exactly with those generated by classic set-sharing. The column labeled T indicates the time in milliseconds required to compute the fixpoint on 2.4 GHz PC equipped with 512 MBytes running Windows XP. The variance in timings between different runs of all analyses was found to be negligible. For set-sharing with collapsing, the timings were generated using sequential closure evaluation. The dashed columns indicate that a timeout was exceeded. The column labeled N records the total number of closure operations that were required (closures over one group are counted as zero). The column labeled S reports the average number of sharing groups that participate in a closure calculation and the column labeled M gives the maximal number of groups that participated in a closure.

The T columns suggest that collapsing closures is potentially useful, though actual speedup will depend on the actual implementation, the overarching fixpoint framework and the underlying machine. The N, S and M columns present a more abstract view of closure collapsing and suggest that the speedup stems from reduced sharing groups manipulation; both the number of closures are reduced (due to the three simplification rules) and the complexity of each closure is reduced (closures are applied to fewer and smaller sharing groups). Interestingly, collapsing is not uniformly faster as is witnessed, by browse and conman, for goal-independent and goal-dependent analysis. This is because S is very small for these programs (2 or 3) even without collapsing. Therefore the overhead of manipulating data-structures that are sets of pairs of sets (rather than merely sets of sets) is not repaid by a commensurate reduction in closure calculation.

	goal-independent								goal-dependent							
	collapsed				classic				collapsed				classic			
file	T	N	S	M	T	N	S	M	T	N	S	M	T	N	S	M
8puzzle	47	31	7	12	2281	77	83	255	78	1	2	2	78	3	2	2
ann	2734	200	6	69	11661	806	14	336	5564	615	3	12	5812	2916	7	33
asm	172	247	2	9	140	563	4	42	937	500	5	148	20701	2299	10	484
boyer	31	110	2	7	47	233	3	64	218	251	4	27	453	740	5	112
browse	32	43	2	4	16	132	2	7	62	52	3	7	31	206	3	8
conman	1187	32	2	3	1235	136	3	8	1906	93	2	4	1813	326	2	16
crip	438	132	8	113	6766	946	15	216	5093	560	8	258	25483	3917	14	304
cry_mult	5907	39	4	13	6219	201	6	32	12500	48	2	7	13016	460	9	32
cs_r	2687	149	25	274	–	–	–	–	516	32	2	8	3250	204	37	240
disj_r	110	45	9	25	8500	321	24	240	219	16	2	3	94	105	2	6
dnf	2	0	0	0	2	28	2	3	31	6	2	2	16	44	2	3
draw	78	28	2	5	281	192	2	6	172	9	2	2	78	25	2	3
ga	203	47	11	50	672	141	12	187	1422	105	12	212	11966	348	15	580
gauss	16	13	3	7	15	81	3	26	47	27	5	17	94	190	5	88
kalah	78	62	3	11	250	204	8	84	141	7	2	6	109	25	7	63
life	516	33	3	7	547	114	6	32	1532	18	3	9	1500	77	3	14
lookup	2	7	2	4	2	34	4	8	15	8	3	4	2	52	3	8
matrix	8	12	2	5	3	54	2	12	63	67	3	21	63	265	4	48
math	31	45	3	6	31	216	3	12	46	43	3	10	31	210	3	24
maze	10	9	2	3	8	14	2	3	31	1	2	2	15	3	2	2
nbody	938	93	3	18	7344	267	18	1022	5641	489	6	94	–	–	–	–
peep	125	333	3	15	329	619	5	108	5077	1576	6	44	17174	3752	10	704
peg	16	7	3	5	63	62	10	75	15	2	4	5	31	49	8	24
plan	62	73	4	26	300	193	8	108	249	178	4	18	499	769	6	112
press	266	293	5	28	641	801	7	70	1937	1184	5	45	6156	4137	8	168
qsort	15	2	3	4	16	12	2	6	16	1	2	2	16	42	2	5
queens	2	2	3	4	2	18	2	6	16	0	0	0	16	0	0	0
robot	63	49	2	10	63	179	4	24	265	203	4	19	281	669	5	32
ronp	47	35	7	31	640	185	13	87	297	137	10	32	2438	660	15	96
rotate	2	4	3	4	2	18	2	6	16	19	5	8	16	82	4	16
shape	31	8	9	14	328	77	28	64	63	32	6	14	672	220	21	64
tictactoe	63	54	6	9	12796	96	100	510	94	1	2	2	62	2	2	2
treeorder	32	28	6	17	469	102	11	120	734	154	8	87	14157	628	17	360
tsp	219	76	3	68	1156	239	10	88	407	253	2	18	187	628	3	28
yasmm	4614	9	3	8	–	–	–	–	19716	40	9	24	–	–	–	–

Table 1. Classic set-sharing versus set-sharing with collapsing

However, if S is large — which happens more often in goal-independent analysis — then the speedup from collapsing closures can be considerable. For example, collapsing requires only 63 milliseconds for tictactoe for goal-independent analysis. Although these results are promising, they are not conclusive and future work will quantify the impact of parallel closure evaluation and investigate collapsing in the context of combined domains [9] and other frameworks [5, 7].

5 Related work

Independence information has many applications in logic programming that include occurs-check reduction [19], automatic parallelisation [17] and finite-tree analysis [1]. Set-sharing analysis [14], as opposed to pair-sharing analysis [19], has particularly attracted much attention. This is because researchers have been repeatedly drawn to the algebraic properties of the domain since these offer tantalising opportunities for implementation. For example, Bagnara et al. [2] observe that set-sharing is redundant when the objective is to detect pairs of independent variables. A sharing group G is redundant with respect to a set-sharing abstraction S iff S contains the pair sharing information that G encodes. The value of redundancy is that it reduces the complexity of abstract unification to a quadratic operation. As a response to this work, Bueno et al. [6] argue that the redundancy assumption is not always valid. Our work adds to this debate by showing the closures are not always as expensive as one might expect.

Another thread of promising work is in representing set-sharing with Boolean functions [10]. In this approach, closure under union is reformulated as a closure operator that maps one function to the smallest function that contains it which can be represented as conjunction of propositional Horn clauses. This operator, though elegant, is non-trivial [18]. Nevertheless, this encoding is advantageous since ROBDDs [18] provide a canonical representation for Boolean functions which enables memoisation to be applied to avoid repeated closure calculation.

The most commonly deployed tactic for reducing the number of closure calculations is to combine set-sharing with other abstract domains [3, 9] that can be used to determine whether closure calculation is actually unnecessary. This is not merely a computational tactic, but also a way to improve precision. Although it seems straightforward to integrate groundness with closure collapsing, future work will address the issue of how to fuse linearity [3, 9] with this tactic.

The work reported in this paper is hinted at by a recent paper of the authors [16] that endeavors to compute closures in an entirely lazy fashion. Each unevaluated closure is represented by a clique — a set of program variables — that augments a standard set-sharing abstractions. Alas, the advantage of this approach is compromised by the way cliques interact with projection operation to loose precision. By way of contrast, this paper shows that projection is a catalyst for collapsing closures which leads to a simpler tactic for avoiding work.

6 Conclusions

Two issues govern the efficiency of set-sharing analysis: the number of sharing groups that participate in each closure operation and the total number of closures that are applied. This paper proposes a new tactic for reducing the overhead of closure calculation based on postponing closures until a propitious moment when they can be applied on less variables. This collapses the size of closures whilst collapsing one closure calculation into another. The resulting analysis is as precise as classic set-sharing analysis.

Acknowledgments This work was funded by NSF grants CCR-0131862 and INT-0327760, EPSRC grant EP/C015517 and the Royal Society grant 2005/R4-JP.

References

1. R. Bagnara, R. Gori, P. Hill, and E. Zaffanella. Finite-Tree Analysis for Constraint Logic-Based Languages. *Information and Computation*, 193(2):84–116, 2004.
2. R. Bagnara, P. Hill, and E. Zaffanella. Set-Sharing is Redundant for Pair-Sharing. *Theoretical Computer Science*, 277(1-2):3–46, 2002.
3. R. Bagnara, E. Zaffanella, and P. Hill. Enhanced Sharing Analysis Techniques: A Comprehensive Evaluation. *Theory and Practice of Logic Programming*, 5, 2005.
4. A. Bossi, M. Gabbrielli, G. Levi, and M. Martelli. The *s*-Semantics Approach: Theory and Applications. *The Journal of Logic Programming*, 19:149–197, 1994.
5. M. Bruynooghe. A Practical Framework for the Abstract Interpretation of Logic Programs. *The Journal of Logic Programming*, 10(2):91–124, 1991.
6. F. Bueno and M. García de la Banda. Set-Sharing Is Not Always Redundant for Pair-Sharing. In *Symposium on Functional and Logic Programming*, volume 2998 of *LNCS*, pages 117–131. Springer-Verlag, 2004.
7. M. Codish. Efficient Goal Directed Bottom-Up Evaluation of Logic Programs. *The Journal of Logic Programming*, 38(3):355–370, 1999.
8. M. Codish, V. Lagoon, and F. Bueno. An Algebraic Approach to Sharing Analysis of Logic Programs. *The Journal of Logic Programming*, 42(2):111–149, 2000.
9. M. Codish, A. Mulkers, M. Bruynooghe, M. García de la Banda, and M. Hermenegildo. Improving Abstract Interpretations by Combining Domains. *ACM TOPLAS*, 17(1):28–44, 1995.
10. M. Codish, H. Søndergaard, and P. Stuckey. Sharing and Groundness Dependencies in Logic Programs. *ACM TOPLAS*, 21(5):948–976, 1999.
11. M. Falaschi, G. Levi, C. Palamidessi, and M. Martelli. Declarative Modeling of the Operational Behavior of Logic Languages. *Theoretical Computer Science*, 69(3):289–318, 1989.
12. J. Gallagher. A Bottom-Up Analysis Toolkit. Technical Report 95-016, Department of Computer Science, University of Bristol, 1995. (Invited paper at WAILL).
13. W. Hans and S. Winkler. Aliasing and Groundness Analysis of Logic Programs through Abstract Interpretation and its Safety. Technical Report Nr. 92-27, RWTH Aachen, Lehrstuhl für Informatik II Ahornstraße 55, W-5100 Aachen, 1992.
14. D. Jacobs and A. Langen. Static Analysis of Logic Programs for Independent And-Parallelism. *The Journal of Logic Programming*, 13(2&3):291–314, 1992.
15. X. Li, A. King, and L. Lu. Correctness of Closure Collapsing. Technical Report 2-06, University of Kent, 2006. http://www.cs.kent.ac.uk/pubs/2006/2370.
16. X. Li, A. King, and L. Lu. Lazy Set-Sharing. In *Symposium on Functional and Logic Programming*, volume 3945 of *LNCS*, pages 177–191. Springer-Verlag, 2006.
17. K. Muthukumar and M. Hermenegildo. Compile-Time Derivation of Variable Dependency Using Abstract Interpretation. *The Journal of Logic Programming*, 13(2-3):315–347, 1992.
18. P. Schachte and H. Søndergaard. Closure Operators for ROBDDs. In *Verification, Model Checking and Abstract Interpretation*, volume 3855 of *LNCS*, pages 1–16. Springer-Verlag, 2006.
19. H. Søndergaard. An Application of Abstract Interpretation of Logic Programs: Occur Check Reduction. In *European Symposium on Programming*, volume 213 of *LNCS*, pages 327–338. Springer-Verlag, 1986.