FixBag* : A Fixpoint Calculator for Quantified Bag Constraints

Tuan-Hung Pham[1], Minh-Thai Trinh[2], Anh-Hoang Truong[2], Wei-Ngan Chin[3]

[1] University of Minnesota, Twin Cities, hung@cs.umn.edu
[2] Vietnam National University, Hanoi, {thaitm_52, hoangta}@vnu.edu.vn
[3] National University of Singapore, chinwn@comp.nus.edu.sg

Abstract. Abstract interpretation techniques have played a major role in advancing the state-of-the-art in program analysis. Traditionally, stand-alone tools for these techniques have been developed for the numerical domain which may be sufficient for lower levels of program correctness. To help us analyze a wider range of programs, we have developed a tool to compute symbolic fixpoints for *quantified bag domain*. This domain is useful for programs that deal with collections of values. Our tool is able to derive both *loop invariants* and *method pre/post conditions* via fixpoint analysis of recursive bag constraints. To support better precision, we have allowed disjunctive formula to be inferred, where appropriate. As a stand-alone tool, we have tested it on a range of small but challenging examples with acceptable precision and performance.

1 Introduction

Abstract interpretation [2] is a technique to infer program's properties. It requires the least fixed point of a monotone function over an abstract domain of the program's semantics to be computed. Let $\langle L, \prec \rangle$ be a complete lattice and $\bot$ be its least element. A function $f : L \to L$ is monotone if $f(u) \prec f(v)$ when $u \prec v$ with all u, v in L. One classical method to find the fixed point is Kleene iteration, which computes the ascending chain $f_0 = \bot$, $f_{i+1} = f(f_i)$ with $i > 0$ until we find i^* satisfying $f_{i^*+1} = f_{i^*}$. Widening operator [3] is used to guarantee that the ascending chain is finite.

Traditionally, stand-alone abstract interpretation (AI) tools have been developed for the numerical domains (such as octagon [8] and polyhedral [4]). Little attention has been paid to building such tools for richer pure domains, such as bags, maps and sequences. Stand-alone AI tools focus primarily on the logics of the abstract domains and use sound mechanisms for approximating recursion via fixed point computation. They have been widely adopted by program analysis systems that are customized to analyze properties from programs (e.g. [1, 9]).

Some recent works [10, 11] have proposed methods to automatically infer disjunctive numerical invariants for added precision. However, numerical invariants are often insufficient for higher levels of program correctness. For example, many

* The tool is available at http://loris-7.ddns.comp.nus.edu.sg/~project/fixbag/

programs are constructed to compute a collection of values whose correctness cannot be captured using only numerical properties. Instead, we require a *quantified bag domain* to provide more precise program analyzers for such programs. To the best of our knowledge, there are no current published tool that can discover quantified bag invariants. We present FixBag, a stand-alone fixpoint calculator for quantified bag constraints. The tool has the following characteristics:

- FixBag can infer disjunctive fixed points of formulae with bag constraints. The maximum number of disjuncts is provided by end-users. Supported bag's operators are union ($\cup$), intersection ($\cap$), and subset ($S_1 \subseteq S_2$), where S_1 and S_2 denote bags.
- FixBag can find fixed points with quantified constraints. Specifically, the system supports universal quantifier of the form $\forall x \in S : P(x)$ and existential quantifier of the form $\exists x \in S : P(x)$ where x, S, and $P(x)$ are a variable, a bag, and a predicate concerning x, respectively.
- FixBag partially supports arithmetic constraints on size properties over bags. Currently, FixBag allows the following types of properties to be inferred : $|S_1| = m \times |S_2|$ and $|S| \leq m$.

Section 2 gives an overview via examples. Section 3 introduces the algorithm to infer fixed points, as used in our tool. Section 4 summarizes our experimental results. Section 5 concludes with a short discussion on related works.

2 Motivating Examples

Our tool is able to compute disjunctive fixpoints for constraint abstractions over the bag domain. To illustrate its capability, we shall analyze two list functions that are commonly used in functional languages, by initially showing their respective constraint abstractions prior to fixpoint analysis. We stress that our tool is language-independent, as its inputs are logical formulae (with bag and size constraints) that may be applied to similar abstractions for programs from other programming languages too.

Our first example is a well-known `filter` function to select elements from a list that satisfy a predicate, `p`, as given below in Caml syntax.

```
filter p xs = match xs with
    | [] → []
    | x : xs → if (p x) then x : (filter p xs) else (filter p xs)
```

The corresponding constraint abstraction, named *filterB*, for this function has three parameters: p to denote the predicate `p` of `filter`, S to capture the elements of input list `xs`, and R to capture the elements of the method's output.

$$\begin{array}{rcl} \mathit{filterB}(p,S,R) & \equiv & S{=}\{\} \wedge R{=}\{\} \ \vee \ \exists x, S_1, R_1 \cdot S{=}\{x\}{\cup}S_1 \wedge \\ & & (p(x) \wedge R{=}\{x\}{\cup}R_1 \wedge \mathit{filterB}(p,S_1,R_1)) \ \vee \\ & & \neg p(x) \wedge R{=}R_1 \wedge \mathit{filterB}(p,S_1,R_1)) \end{array}$$

When this constraint abstraction is passed to our tool, we could infer the following fairly precise closed-form formula using universally quantified bags.

$$\mathit{filterB}(p,S,R) \ \equiv \ (\forall x \in R : p(x)) \ \wedge \ (\forall x \in S - R : \neg p(x)) \ \wedge \ R \subseteq S.$$

Our next example is a membership function to determine if an element exists within an input list or not. Its Caml code is given below:

```
mem v xs = match xs with | [] → false
                         | x : xs → if x = v then true else (mem v xs)
```

The corresponding constraint abstraction has three parameters, as shown:
$memB(v, S, r) \equiv S{=}\{\}\wedge\neg r \ \vee \ \exists x, S_1 \cdot S{=}\{x\}\cup S_1 \wedge (x{=}v\wedge r \ \vee \ x{\neq}v\wedge memB(v, S_1, r))$

A precise closed-form formula for this function requires both disjunction and quantified bag formula, as shown below, which our tool can derive.

$$memB(v, S, r) \ \equiv \ (\forall x \in S : x{\neq}v) \wedge \neg r \ \vee \ (\exists x \in S : x{=}v) \wedge r$$

These two examples show that a good treatment of quantified formula and disjunction is needed to support more precise analysis.

3 Algorithm

This section presents the algorithm behind FixBag. It starts by defining some operators used in the bag domain and then comes up with a general algorithm to find quantified, disjunctive fixpoints. When we mention that ϕ is a formula, it means that the formula is *normalized*. The definition of the normalizing process is given below.

Definition 1 (Normalizing). *A formula ϕ can be normalized in a twofold process: First, ϕ is equivalently converted into ϕ_{DNF} in Disjunctive Normal Form (DNF). Second, we remove all redundant parts that can be safely eliminated without changing the logical value of ϕ_{DNF}. They may be duplicate conjuncts, duplicate disjuncts,* `true` *value in a conjunctive formula with more than one conjunct, or* `false` *value in a disjunctive formula with more than one disjunct.*

One of the necessary operators used in fixpoint analysis is hulling which is well-developed for numerical domains. However, to the best of our knowledge, there is no work attempting to calculate hulling operations on bag/set domain to date. To realize this, we propose to use a rule-based approach that uses propagation and simplification rules to attain hulling of formulae for the bag domain. Similar to CHR [5], our propagation rules $\mathcal{R}_p$ add new redundant constraints to a formula while simplification rules $\mathcal{R}_s$ aim to reduce the size of a formula by removing redundant constraints. Although these rules themselves preserve the logical equivalences of the original formula, they help create intermediate results that can play important roles in other operations. Let $\phi_1 \Cap \phi_2$ be $\bigwedge_{i=1}^{k} d_i$ where $d_1, ..., d_k$ are common conjuncts of two conjunctive formulae ϕ_1 and ϕ_2. Definition 2 shows our approach to find the hulling result of two conjunctive formulae.

Definition 2 (Hulling). *Given two conjunctive formulae ϕ_1 and ϕ_2, we divide each of them into two parts: the first one (Γ) contains all conjuncts of the form $m_1 \leq |S| \leq m_2$ and the other (Δ) has the remaining conjuncts. The two original formulae are represented as $\phi_1 = \Gamma_1 \wedge \Delta_1$ and $\phi_2 = \Gamma_2 \wedge \Delta_2$. The hulling operation is defined as $\phi_1 \boxtimes \phi_2 = \Gamma_1 \boxtimes \Gamma_2 \wedge \Delta_1 \boxtimes \Delta_2$ where*

– $\Gamma_1 \boxtimes \Gamma_2 =$
$\underbrace{(\bigwedge m_{i1} \leq |S_i| \leq m_{i2})}_{(1)} \wedge \underbrace{(\bigwedge m_{j1} \leq |S_j| \leq m_{j2})}_{(2)} \wedge \underbrace{(\bigwedge min(m_{i'1}, m_{j'1}) \leq |S_{i'j'}| \leq max(m_{i'2}, m_{j'2}))}_{(3)}$
where (1) contains all $(m_{i1} \leq |S_i| \leq m_{i2}) \in \Gamma_1$ *that* S_i *is not in in* Γ_2*, (2) consists of all* $(m_{j1} \leq |S_j| \leq m_{j2}) \in \Gamma_2$ *that* S_j *is not in* Γ_1*, and (3) has all* $S_{i'j'}$ *that has* $(m_{i'1} \leq |S_{i'j'}| \leq m_{i'2}) \in \Gamma_1$ *and* $(m_{j'1} \leq |S_{i'j'}| \leq m_{j'2}) \in \Gamma_2$.
– $\Delta_1 \boxtimes \Delta_2 = \underset{\mathcal{R}_s}{simplify}(\Delta_1 \widehat{\boxtimes} \Delta_2)$ *where* $\Delta_1 \widehat{\boxtimes} \Delta_2 = \underset{\mathcal{R}_p}{propagate} \Delta_1 \Cap \underset{\mathcal{R}_p}{propagate} \Delta_2$.

We also define a version of the hulling operation without simplification as $\phi_1 \widehat{\boxtimes} \phi_2 = \Gamma_1 \boxtimes \Gamma_2 \wedge \Delta_1 \widehat{\boxtimes} \Delta_2$. It is needed when we want to check whether a particular conjunct contributes to the hulling result or not. This notion is used to measure the closeness of two conjunctive formulae in Definition 3. Given two conjunctive formula ϕ_1 and ϕ_2, $\phi_1 \oslash \phi_2$ quantifies their closeness as an integer in the range of 1..99. The larger the number is, the closer they are to each other. We denote $\lceil \phi \rceil$ ($\lfloor \phi \rfloor$) the number of conjuncts (disjuncts) in a conjunctive (disjunctive) formula ϕ.

Definition 3 (Affinity Measure). *Given two conjunctive formulae* ϕ_1 *and* ϕ_2*, the affinity measure* $\oslash$ *is defined as follows:* $\phi_1 \oslash \phi_2 = \frac{\lceil (\phi_1 \wedge \phi_2) \Cup (\phi_1 \widehat{\boxtimes} \phi_2) \rceil}{\lceil \phi_1 \wedge \phi_2 \rceil} \times 98 + 1$

While our hulling only works with conjuncts, selective hulling [10] can deal with disjunctive formulae. Our tool can increase the precision of the output fixpoints by allowing up to μ disjuncts to be present during the analysis process. The main idea is to repeatedly call hulling with a closest pair of disjuncts taken from both the two formulae until there are at most μ disjuncts remaining.

Definition 4 (Selective Hulling). *Given a disjunctive formula* $\phi = \bigvee_{i=1}^{k} d_i$ *and a maximum number of disjuncts* μ*, the selective hulling operation* $\oplus_\mu$ *is defined as* $\oplus_\mu \phi = normalize(\widehat{\oplus}_\mu \phi)$ *where* $\widehat{\oplus}_\mu \phi$ *is recursively defined as follows:*

$$\widehat{\oplus}_\mu \phi = \begin{cases} \phi \text{ if } k \leq \mu \\ \widehat{\oplus}_\mu \big((d_{i'} \boxtimes d_{i''}) \vee \bigvee_{i \in \{1..k\} \setminus \{i', i''\}} d_i \big) \text{ if } k > \mu \\ \qquad \text{where } (i', i'') = argmax(d_{j'} \oslash d_{j''}) \text{ for } 1 \leq j', j'' \leq k \end{cases}$$

Widening [3] is used to ensure that the fixpoint analysis terminates. To maintain disjunctions in the widening process, selective widening [10] is required.

Definition 5 (Selective Widening). *Given two disjunctive formulae* $\phi_1 = \bigvee_{i=1}^{k} d_i$ *and* $\phi_2 = \bigvee_{j=1}^{k} e_j$*, the selective widening operation* $\phi_1 \triangledown \phi_2$ *is defined as* $\phi_1 \triangledown \phi_2 = normalize(\phi_1 \widehat{\triangledown} \phi_2)$ *where* $\phi_1 \widehat{\triangledown} \phi_2$ *is recursively defined as follows:*

$$\phi_1 \widehat{\triangledown} \phi_2 = \begin{cases} \phi_1 \boxtimes \phi_2 \text{ if } k = 1 \\ (d_{i'} \boxtimes e_{j'}) \vee (\bigvee_{i \in \{1..k\} \setminus \{i'\}} d_i \widehat{\triangledown} \bigvee_{j \in \{1..k\} \setminus \{j'\}} e_j) \text{ if } k > 1 \\ \qquad \text{where } (i', j') = argmax(d_{i''} \oslash e_{j''}) \text{ with } 1 \leq i'', j'' \leq k \end{cases}$$

The algorithm to find the fixpoint for formulae with bag constraints shares the same ideas with the one used in [10] to infer disjunctive postconditions. The main challenges here are how we support planar affinity, selective widening, and selective hulling to work with bag domain. Given a recursive function f, we start with $f_0 = \bot$ (which is `false` in the bag domain) and then compute an ascending chain $f_1, f_2, ...$, until we find two equally consecutive elements $f_{i^*+1} = f_{i^*}$. If the current value in the chain is f_i, the next item f_{i+1} will be calculated as $bottomup(f, f_i, \mu)$, which is either $f(f_i)$ or its approximation with the help of selective hulling and widening operations. The first kind of operations helps us achieve disjunctive fixpoint while the second kind ensures that the chain will converge. Algorithm 1 shows how we can obtain f_{i+1} from f, f_i, and μ.

Algorithm 1: Calculating $f_{i+1} = bottomup(f, f_i, \mu)$

```
1 f_{f_i} ← normalize(f(f_i));
2 if ⌊f_{f_i}⌋ < μ then
3     return f_{f_i};
4 else if f_i = ⊥ or ⌊⊕_μ f_{f_i}⌋ < μ or ⌊f_i⌋ < μ then
5     return ⊕_μ f_{f_i};
6 else
7     return f_i ▽ (⊕_μ f_{f_i});
```

First, we normalize the result of $f(f_i)$ to achieve f_{f_i}, which is a candidate for f_{i+1}. If $\lfloor f_{f_i} \rfloor < \mu$, we return f_{f_i}, otherwise we need to find a suitable approximation of f_{f_i} that has no more than μ disjuncts. If $f_i = \bot$ or $\lfloor \oplus_\mu f_{f_i} \rfloor < \mu$, the approximation is $\oplus_\mu f_{f_i}$. If the two previous conditions fail, we have $f_i \neq \bot$ and $\lfloor \oplus_\mu f_{f_i} \rfloor = \mu$. At this point, the approximation will depend on $\lfloor f_i \rfloor$ because selective widening only works with two formulae that have the same number of disjuncts. Therefore, if $\lfloor f_i \rfloor < \mu$, we still return $\oplus_\mu f_{f_i}$. Finally, when $\lfloor \oplus_\mu f_{f_i} \rfloor = \lfloor f_i \rfloor = \mu$ holds, the best approximation is $f_i \triangledown (\oplus_\mu f_{f_i})$, which not only maintains up to μ disjuncts but also contributes to the convergence of the chain.

4 Experimental results

We tested our tool on a set of methods from the Ocaml's List library. The tool takes the abstraction of each (possibly recursive) function and a number μ, denoting the maximum number of disjuncts allowed during the fixpoint inference, as its arguments and returns a corresponding fixpoint. We also measured the running-time of the analysis process. These results can be found in Appendix B. The grammar form of quantified bag formulae and inference rules used (by our tool) are detailed in `http://loris-7.ddns.comp.nus.edu.sg/~project/fixbag/`.

We have encountered several examples where disjunctive analysis can obtain more precise fixpoints than conjunctive analysis. Conjunctive analysis can be simulated using $\mu = 1$. In general, each analyzed function has an upper bound of μ; increasing μ over this bound does not help achieve more precise fixpoints and does not affect the analysis time.

5 Related Works and Conclusion

Libraries to support abstract interpretation are popular for program analysis systems, but they are focused mostly on the numeric domains [8, 1]. In the non-numeric domains, abstract interpretation tools have been developed for shape analysis [7] and for constraint-based analysis [6]. The former is for discovering data shapes of heap-manipulating programs rather than their pure properties; and are thus focused on program codes rather than logical formula. The latter is meant as a scalable tool for flow-based constraints, rather than for analyzing collections. Both systems do not automatically handle quantified formula and have restricted use of disjunctive formulae.

We have built a stand-alone abstract interpretation tool for quantified bag domain. Our use of simplification and propagation techniques is inspired from CHR [5], while the use of affinity-based hulling and widening is targeted at more precise disjunctive fixpoints. Our experiments on a code library have shown that our tool is capable of efficiently analyzing the collection properties for non-trivial functions.

References

1. Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *SCP*, 72:3–21, June 2008.
2. Patrick Cousot and Radhia Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *POPL'77*, pages 238–252, 1977.
3. Patrick Cousot and Radhia Cousot. Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation. In *PLILP*, pages 269–295, 1992.
4. Patrick Cousot and Nicolas Halbwachs. Automatic Discovery of Linear Restraints Among Variables of a Program. In *POPL'78*, pages 84–96, 1978.
5. Thom W. Frühwirth. Theory and Practice of Constraint Handling Rules. *Journal of Logic Programming*, 37(1-3):95–138, 1998.
6. John Kodumal and Alexander Aiken. Banshee: A Scalable Constraint-Based Analysis Toolkit. In *SAS'05*, pages 218–234, 2005.
7. Tal Lev-Ami and Shmuel Sagiv. TVLA: A System for Implementing Static Analyses. In *SAS'00*, pages 280–301, 2000.
8. Antoine Miné. The Octagon Abstract Domain. *Higher-Order and Symbolic Computation*, 19:31–100, March 2006.
9. Bertrand net and Antoine Miné. Apron: A Library of Numerical Abstract Domains for Static Analysis. In *CAV'09*, pages 661–667, 2009.
10. Corneliu Popeea and Wei-Ngan Chin. Inferring Disjunctive Postconditions. In *ASIAN'06*, pages 331–345, 2007.
11. Sriram Sankaranarayanan, Franjo Ivancic, Ilya Shlyakhter, and Aarti Gupta. Static Analysis in Disjunctive Numerical Domains. In *SAS'06*, pages 3–17, 2006.

A Inference rules

This section describes a list of reference rules used in FixBag. Let R, S, T, U, V be general bags, $\{v_1, ..., v_n\}$ be an explicit bag of n elements, and $U[S \leftarrow T]$ be a bag obtained by substituting T for S in U. In expression $m_1 \leq |S| \leq m_2$, m_1 may be 0 and m_2 may be $+\infty$; in that cases it is displayed as $|S| \leq m_2$ and $m_1 \leq |S|$, respectively.

Table 1. List of propagation rules

RULE		LHS		RHS
EMPTY	@	$S = \varnothing$	$\Longrightarrow$	$S \subseteq \top,\ S \cap \top = \varnothing,\ S - \top = \varnothing,\ \|S\| = 0,\ \forall x \in S : \top$
INTER-1	@	$R = S \cap T$	$\Longrightarrow$	$R \subseteq S,\ R \subseteq T$
INTER-2	@	$R \subseteq S \cap T$	$\Longrightarrow$	$R \subseteq S,\ R \subseteq T$
MINUS-1	@	$R = S - T$	$\Longrightarrow$	$R \subseteq S$
MINUS-2	@	$R \subseteq S - T$	$\Longrightarrow$	$R \subseteq S$
UNION-1	@	$R = S \cup T$	$\Longrightarrow$	$S \subseteq R,\ T \subseteq R$
UNION-2	@	$R \supseteq S \cup T$	$\Longrightarrow$	$S \subseteq R,\ T \subseteq R$
TRANS	@	$R \subseteq S,\ S \subseteq T$	$\Longrightarrow$	$R \subseteq T$
SUBS-1	@	$S = T,\ U = V$	$\Longrightarrow$	$U[S \leftarrow T] = V,\ U[S \leftarrow T] = V[S \leftarrow T],$ $U[S \leftarrow T] = V[T \leftarrow S]$
SUBS-2	@	$S = T,\ U \subseteq V$	$\Longrightarrow$	$U[S \leftarrow T] \subseteq V,\ U \subseteq V[S \leftarrow T],$ $U[S \leftarrow T] \subseteq V[S \leftarrow T],\ U[T \leftarrow S] \subseteq V[S \leftarrow T],$ $U[S \leftarrow T] \subseteq V[T \leftarrow S],\ S[V \leftarrow U] \subseteq T,$ $S \subseteq T[U \leftarrow V],\ S[V \leftarrow U] \subseteq T[U \leftarrow V]$
IN	@	$S = U \cup \{v\},\ T = V \cup \{v\}$	$\Longrightarrow$	$S \cap T = U \cap V \cup \{v\},\ S - T = U - V$
NOTIN	@	$S = U \cup \{v\},\ T = V$	$\Longrightarrow$	$\{v\} \cap T = \varnothing \mid S \cap T = U \cap V,\ S - T = U - V \cup \{v\}$
QUAN-1	@	$S = \{x\},\ P(x)$	$\Longrightarrow$	$\forall x \in S : P(x)$
QUAN-2	@	$S = \{x\},\ P(x)$	$\Longrightarrow$	$\exists x \in S : P(x)$
QUAN-3	@	$S \subseteq T,\ \forall x \in T : P(x)$	$\Longrightarrow$	$\forall x \in S : P(x)$
QUAN-4	@	$S \subseteq T,\ \exists x \in S : P(x)$	$\Longrightarrow$	$\exists x \in T : P(x)$
QUAN-5	@	$\forall x \in S : P(x), \forall x \in T : P(x), R = S \cup T$	$\Longrightarrow$	$\forall x \in R : P(x)$
BASE	@	$S = \{v_1, \ldots, v_n\}$	$\Longrightarrow$	$\|S\| = n$
INF-1	@	$S = T$	$\Longrightarrow$	$\|S\| = \|T\|$
INF-2	@	$S \subseteq T$	$\Longrightarrow$	$\|S\| \leq \|T\|$
INF-3	@	$S = R \cup T$	$\Longrightarrow$	$\|S\| = \|R\| + \|T\|$
MUL	@	$\|S\| = m,\ \|T\| = m \times n$	$\Longrightarrow$	$\|T\| = n \times \|S\|$

Table 2. List of simplification rules

RULE		LHS		RHS
EMP-1	@	$S \subseteq \top$	$\Longleftrightarrow$	$true$
EMP-2	@	$\varnothing \subseteq R$	$\Longleftrightarrow$	$true$
EMP-3	@	$S = \varnothing$	$\Longleftrightarrow$	$\|S\| = 0$
SUB-1	@	$S \cup \top$	$\Longleftrightarrow$	$\top$
SUB-2	@	$S \cap \top$	$\Longleftrightarrow$	S
SUB-3	@	$S - \top$	$\Longleftrightarrow$	$\varnothing$
SUB-4	@	$S - \varnothing$	$\Longleftrightarrow$	S
SUB-5	@	$\varnothing - S$	$\Longleftrightarrow$	$\varnothing$
REF	@	$S = S$	$\Longleftrightarrow$	$true$
ANT	@	$S \subseteq R,\ R \subseteq S$	$\Longleftrightarrow$	$S = R$
SIZE	@	$\|S\| \geq 0$	$\Longleftrightarrow$	$true$

B Experimental data

This Appendix contains our experimental data set. We provide them here to illustrate the capability and performance of our tool. Table 3 shows the abstractions for a list of small but challenging functions successfully analyzed by our tool. Functions 1-28 make use of a variety of different formulae over the bag domain, while functions 29-35 and 36-39 are, respectively, cases where quantified bag formulae and size properties are automatically inferred by our tool.

Table 3. List of input functions

No.	Function	
1	$inter_1(S_1, S_2, S_3)$	$:= (S_1 = \{\} \wedge S_3 = \{\}) \vee$ $\exists v : S_1 = \{v\} \cup S_{1a} \wedge S_2 = \{v\} \cup S_{2a} \wedge inter_1(S_{1a}, S_{2a}, S_{3a}) \wedge S_3 = \{v\} \cup S_{3a} \vee$ $\exists v : S_1 = \{v\} \cup S_{1a} \wedge S_2 \cap \{v\} = \{\} \wedge inter_1(S_{1a}, S_2, S_3)$
2	$inter_2(S_1, S_2, S_3)$	$:= (S_1 = \{\} \wedge S_3 = \{\}) \vee$ $(\exists v : S_1 = \{v\} \cup S_{1a} \wedge S_2 = \{v\} \cup S_{2a} \wedge inter_2(S_{1a}, S_{2a}, S_{3a}) \wedge S_3 = \{v\} \cup S_{3a}) \vee$ $(\exists v : S_1 = \{v\} \cup S_{1a} \wedge S_2 = S_{2a} \wedge inter_2(S_{1a}, S_{2a}, S_3))$
3	$decomp_1(S_1, S_2, S_3)$	$:= (S_1 = \{\} \wedge S_2 = \{\} \wedge S_3 = \{\}) \vee$ $(\exists v : S_1 = \{v\} \cup S_{1a} \wedge S_3 = \{v\} \cup S_{3a} \wedge decomp_1(S_{1a}, S_2, S_{3a})) \vee$ $(\exists v : S_2 = \{v\} \cup S_{2a} \wedge S_3 = \{v\} \cup S_{3a} \wedge decomp_1(S_1, S_{2a}, S_{3a}))$
4	$flatten_1(S_1, S_2)$	$:= (S_1 = \{\} \wedge S_2 = \{\}) \vee (S_1 = S \cup S_{1a} \wedge S_2 = S \cup S_{2a} \wedge flatten_1(S_{1a}, S_{2a}))$
5	$flatten_2(S_1, S_2)$	$:= (S_1 = \{\} \wedge S_2 = \{\}) \vee \exists v : S_1 = S_{1a} \cup \{v\} \wedge S_2 = S_{2a} \cup \{v\} \wedge flatten_2(S_{1a}, S_{2a})$
6	$flatten_3(S_1, S_2)$	$:= (S_1 = \{\} \wedge S_2 = \{\}) \vee \exists S : S_1 = S \cup S_{1a} \wedge S_2 = S \cup S_{2a} \wedge flatten_3(S_{1a}, S_{2a})$
7	$append(S_1, S_2, S_3)$	$:= (S_1 = \{\} \wedge S_3 = S_2) \vee \exists v : S_1 = \{v\} \cup S_{1a} \wedge append(S_{1a}, S_2, S_{3a}) \wedge S_3 = \{v\} \cup S_{3a}$
8	$fil_1(S_1, S_2)$	$:= S_2 = \{\} \vee \exists v : S_1 = \{v\} \cup S_{1a} \wedge fil_1(S_{1a}, S_{2a}) \wedge S_2 = \{v\} \cup S_{2a} \vee$ $\exists v : S_1 = \{v\} \cup S_{1a} \wedge fil_1(S_{1a}, S_{2a}) \wedge S_2 = S_{2a}$
9	$fil_2(S_1, S_2)$	$:= S_2 = \{\} \vee \exists v : S_1 = \{v\} \cup S_{1a} \wedge fil_2(S_{1a}, S_{2a}) \wedge (S_2 = \{v\} \cup S_{2a} \vee S_2 = S_{2a})$
10	$id(S_1, S_2)$	$:= (S_1 = \{\} \wedge S_2 = S_1) \vee \exists v : S_1 = \{v\} \cup S_{1a} \wedge id(S_{1a}, S_{2a}) \wedge S_2 = \{v\} \cup S_{2a}$
11	$part_1(S_1, S_2, S_3)$	$:= (S_1 = \{\} \wedge S_2 = \{\} \wedge S_3 = \{\}) \vee (\exists v : S_1 = \{v\} \cup S_{1a} \wedge part_1(S_{1a}, S_{2a}, S_{3a}) \wedge$ $((S_2 = S_{2a} \wedge S_3 = \{v\} \cup S_{3a}) \vee (S_2 = \{v\} \cup S_{2a} \wedge S_3 = S_{3a})))$
12	$revit(S_1, S_2, R)$	$:= (S_1 = \{\} \wedge S_2 = R) \vee (\exists v : revit(S_{1a}, S_{2a}, R) \wedge S_1 = S_{1a} \cup \{v\} \wedge S_{2a} = S_2 \cup \{v\})$
13	$head_1(S_1, S_3)$	$:= (\exists S_2 : (\exists v : S_1 = \{v\} \cup S_2 \wedge S_3 = \{v\}))$
14	$head_2(S_1, v)$	$:= S_1 = \{v\} \cup S_2$
15	$head_3(S_1, S_3)$	$:= \exists v : S_1 = \{v\} \cup S_2 \wedge S_3 = \{v\}$
16	$head_4(S_1, S_2, S_3)$	$:= \exists v : S_1 = \{v\} \cup S_2 \wedge S_3 = \{v\}$
17	$tailtwice(S_1, S_3)$	$:= (\exists w : (\exists v : S_1 = \{v\} \cup S_2 \wedge S_2 = \{w\} \cup S_3))$
18	$tail(S_1, S_2)$	$:= \exists v : S_1 = \{v\} \cup S_2$
19	$disj(S0, S_1, S_2, S_3)$	$:= (S_2 = \{\} \wedge S_3 = \{\} \wedge S_1 = \{\}) \vee$ $\exists v : S0 = \{\} \wedge S_3 = S_{3a} \cup \{v\} \wedge disj(S0, S_{1a}, S_2, S_{3a}) \wedge S_1 = S_{1a} \cup \{v\} \vee$ $\exists v : (!(S0 = \{\})) \wedge S_2 = S_{2a} \cup \{v\} \wedge disj(S0, S_{1a}, S_{2a}, S_3) \wedge S_1 = S_{1a} \cup \{v\}$
20	$inters_1(S_1, S_2, S_3)$	$:= (S_1 = \{\} \wedge S_3 = \{\}) \vee$ $(\exists v : S_1 = \{v\} \cup S_{1a} \wedge S_2 = \{v\} \cup S_{2a} \wedge inters_1(S_{1a}, S_{2a}, S_{3a}) \wedge S_3 = \{v\} \cup S_{3a}) \vee$ $(\exists v : S_1 = \{v\} \cup S_{1a} \wedge inters_1(S_{1a}, S_2, S_3))$
21	$inters_2(S_1, S_2, S_3)$	$:= (S_1 = \{\} \wedge S_3 = \{\}) \vee \exists v : S_1 = \{v\} \cup S_{1a} \wedge inters_2(S_{1a}, S_2, S_3)$
22	$inters_3(S_1, S_2, S_3)$	$:= (S_1 = \{\} \wedge S_3 = \{\}) \vee$ $\exists v : S_1 = \{v\} \cup S_{1a} \wedge S_2 = \{v\} \cup S_{2a} \wedge inters_3(S_{1a}, S_{2a}, S_{3a}) \wedge S_3 = \{v\} \cup S_{3a}$
23	$decomps_1(S_1, S_2, S_3)$	$:= S_1 = \{\} \wedge S_2 = \{\} \wedge S_3 = \{\} \vee$ $(\exists v : S_2 = \{v\} \cup S_{2a} \wedge S_3 = \{v\} \cup S_{3a} \wedge decomps_1(S_1, S_{2a}, S_{3a}))$
24	$decomps_2(S_1, S_2, S_3)$	$:= S_1 = \{\} \wedge S_2 = \{\} \wedge S_3 = \{\} \vee$ $(\exists v : S_1 = \{v\} \cup S_{1a} \wedge S_3 = \{v\} \cup S_{3a} \wedge decomps_2(S_{1a}, S_2, S_{3a}))$
25	$fils_1(S_1, S_2)$	$:= S_2 = \{\} \vee \exists v : S_1 = \{v\} \cup S_{1a} \wedge fils_1(S_{1a}, S_{2a}) \wedge S_2 = S_{2a}$
26	$fils_2(S_1, S_2)$	$:= S_2 = \{\} \vee \exists v : S_1 = \{v\} \cup S_{1a} \wedge fils_2(S_{1a}, S_{2a}) \wedge S_2 = \{v\} \cup S_{2a}$
27	$parts_1(S_1, S_2, S_3)$	$:= (S_1 = \{\} \wedge S_2 = \{\} \wedge S_3 = \{\}) \vee$ $(\exists v : S_1 = \{v\} \cup S_{1a} \wedge parts_1(S_{1a}, S_{2a}, S_{3a}) \wedge S_2 = S_{2a} \wedge S_3 = \{v\} \cup S_{3a})$
28	$parts_2(S_1, S_2, S_3)$	$:= (S_1 = \{\} \wedge S_2 = \{\} \wedge S_3 = \{\}) \vee$ $(\exists v : S_1 = \{v\} \cup S_{1a} \wedge parts_2(S_{1a}, S_{2a}, S_{3a}) \wedge S_2 = \{v\} \cup S_{2a} \wedge S_3 = S_{3a})$
29	$part(S_1, S_2, S_3, p)$	$:= (S_1 = \{\} \wedge S_2 = \{\} \wedge S_3 = \{\}) \vee (\exists v : S_1 = \{v\} \cup S_{1a} \wedge part(S_{1a}, S_{2a}, S_{3a}, p) \wedge$ $((S_2 = S_{2a} \wedge S_3 = \{v\} \cup S_{3a} \wedge p > v) \vee (S_2 = \{v\} \cup S_{2a} \wedge S_3 = S_{3a} \wedge v >= p)))$
30	$member(S, p, res)$	$:= (S = \{\} \wedge !\$res) \vee (\exists v : S = \{v\} \cup S_1 \wedge ((v = p \wedge \$res) \vee (v \neq$ $p \wedge member(S_1, p, res))))$
31	$filter(S_1, f, S_2)$	$:= (S_1 = \{\} \wedge S_2 = \{\}) \vee (\exists v : S_1 = \{v\} \cup S_{1a} \wedge filter(S_{1a}, f, S_{2a}) \wedge ((f(v) \wedge S_2 =$ $\{v\} \cup S_{2a}) \vee (!f(v) \wedge S_2 = S_{2a})))$
32	$forall_1(f, S, res)$	$:= (S = \{\} \wedge \$res) \vee \exists v : S = \{v\} \cup S_1 \wedge ((forall_1(f, S_1, res) \wedge f(v)) \vee (!f(v) \wedge !\$res))$
33	$exists_1(f, S, res)$	$:= (S = \{\} \wedge !\$res) \vee \exists v : S = \{v\} \cup S_1 \wedge ((f(v) \wedge \$res) \vee (exists_1(f, S_1, res) \wedge !f(v)))$
34	$mem(a, S, res)$	$:= (S = \{\} \wedge !\$res) \vee \exists v : S = \{v\} \cup S_1 \wedge ((mem(a, S_1, res) \wedge a \neq v) \vee (a = v \wedge \$res))$
35	$memAssoc(a, S, res)$	$:= (S = \{\} \wedge !\$res) \vee \exists v : S = \{v\} \cup S_1 \wedge v = a \wedge \$res \vee$ $\exists v : S = \{v\} \cup S_1 \wedge v \neq a \wedge memAssoc(a, S_1, res)$
36	$tran_1(S, R, v_2)$	$:= (R = \{\} \wedge S = \{\}) \vee (\exists v : (tran_1(S_1, R_1, v_2) \wedge R = R_1 \cup \{v_2\} \cup \{v_2\} \wedge S = S_1 \cup \{v\}))$
37	$tran_2(S, R, v_2)$	$:= (R = \{\} \wedge S = \{\}) \vee (\exists v : ((tran_2(S_1, R_1, v_2) \wedge R = R_1 \cup \{v_2\}) \wedge S = S_1 \cup \{v\}))$
38	$tran_3(S, R, v_2)$	$:= (R = \{\} \wedge S = \{\}) \vee (\exists v : ((tran_3(S_1, R_1, v_2) \wedge R = R_1) \wedge S = S_1 \cup \{v\}))$
39	$removeFst(a, S, R)$	$:= (S = \{\} \wedge R = \{\}) \vee \exists v : S = S_1 \cup \{v\} \wedge a = v \wedge R = S1 \vee$ $\exists v : S = S_1 \cup \{v\} \wedge removeFst(a, S_1, R_1) \wedge a \neq v \wedge R = \{v\} \cup R_1$

Our test platform was a Pentium Dual Core 2.5 GHz system with 1GBytes main memory, running Ubuntu 10.04. Table 4 summarizes the statistics obtained for each function. Our tool uses μ to limit the number of disjuncts in the analyzing process. For each value of μ, we calculated the fixpoint result and recorded the time taken by our tool for inference.

Table 4. Statistics for fixpoint inference. Timings exclude parsing time and "–" means a time or a fixpoint is similar to those from the immediate lower value of μ.

No.	Maximum number of disjuncts					
	$\mu = 1$		$\mu = 2$		$\mu = 3$	
	fixpoint	(secs)	fixpoint	(secs)	fixpoint	(secs)
1	$S_3 = S_1 \cap S_2$	0.35	–	1.06	–	–
2	$S_3 = S_1 \cap S_2$	0.38	–	1.09	–	–
3	$S_3 = S_1 \cup S_2$	0.82	–	2.85	–	–
4	$S_1 = S_2$	0.01	–	0.06	–	–
5	$S_1 = S_2$	0.01	–	0.06	–	–
6	$S_1 = S_2$	0.01	–	0.06	–	–
7	$S_3 = S_1 \cup S_2$	0.14	–	0.31	–	–
8	$S_2 \subseteq S_1$	0.05	–	0.15	–	–
9	$S_2 \subseteq S_1$	0.05	–	0.15	–	–
10	$S_2 = S_1$	0.06	–	0.15	–	–
11	$S_1 = S_2 \cup S_3$	2.45	–	5.43	–	–
12	$R = S_1 \cup S_2$	0.11	–	0.19	–	–
13	$S_3 \subseteq S_1 \wedge \lvert S_3\rvert = 1$	0.00	–	0.01	–	–
14	$\{v\} \subseteq S_1$	0.00	–	0.01	–	–
15	$S_3 \subseteq S_1 \wedge \lvert S_3\rvert = 1$	0.00	–	0.01	–	–
16	$S_1 = S_2 \cup S_3 \wedge \lvert S_3\rvert = 1$	0.00	–	0.01	–	–
17	$S_3 \subseteq S_1 \wedge \lvert S_1\rvert \geq 2$	0.02	–	0.02	–	–
18	$S_2 \subseteq S_1 \wedge \lvert S_1\rvert \geq 1$	0.00	–	0.01	–	–
19	$S_1 = S_2 \cup S_3$	0.53	$(S_1 = S_2 \wedge S_3 = \emptyset \wedge \lvert S_0\rvert \geq 1) \vee (S_0 = \emptyset \wedge S_1 = S_3 \wedge S_2 = \emptyset)$	2.13	–	–
20	$S_3 \subseteq S_1 \wedge S_3 \subseteq S_2$	0.20	–	0.71	–	–
21	$S_3 = \emptyset$	0.01	–	0.02	–	–
22	$S_1 \subseteq S_2 \wedge S_1 = S_3$	0.22	–	0.79	–	–
23	$S_1 = \emptyset \wedge S_3 = S_2$	0.43	–	1.25	–	–
24	$S_1 = S_3 \wedge S_2 = \emptyset$	0.41	–	1.22	–	–
25	$S_2 = \emptyset$	0.02	–	0.02	–	–
26	$S_2 \subseteq S_1$	0.03	–	0.12	–	–
27	$S_1 = S_3 \wedge S_2 = \emptyset$	0.43	–	1.01	–	–
28	$S_1 = S_2 \wedge S_3 = \emptyset$	0.45	–	0.94	–	–
29	$S_1 = S_2 \cup S_3 \wedge \forall x \in S_2 : p \leq x \wedge \forall x \in S_3 : x < p$	2.64	–	5.56	–	–
30	$\top$	0.00	$(\exists x \in S : x = p \wedge res) \vee (\forall x \in S : x \neq p \wedge \neg res)$	0.01	–	–
31	$S_2 \subseteq S_1 \wedge \forall x \in S_2 : f(x) \wedge \forall x \in S_1 - S_2 : \neg f(x)$	0.13	–	0.36	–	–
32	$\top$	0.01	$(\forall x \in S : f(x) \wedge res) \vee (\exists x \in S : \neg f(x) \wedge \neg res)$	–	–	–
33	$\top$	0.00	$(\exists x \in S : f(x) \wedge res) \vee (\forall x \in S : \neg f(x) \wedge \neg res)$	0.01	–	–
34	$\top$	0.01	$(\exists x \in S : a = x \wedge res) \vee (\forall x \in S : a \neq x \wedge \neg res)$	0.02	–	–
35	$\top$	0.01	$(\exists x \in S : x = a \wedge res) \vee (\forall x \in S : x \neq a \wedge \neg res)$	0.02	–	–
36	$\forall x \in R : x = v_2 \wedge \lvert R\rvert = 2 \times \lvert S\rvert$	0.03	–	0.14	–	–
37	$\forall x \in R : x = v_2 \wedge \lvert S\rvert = \lvert R\rvert$	0.04	–	0.13	–	–
38	$R = \emptyset$	0.01	–	0.02	–	–
39	$R \subseteq S$	0.01	$(S = R \wedge \forall x \in S : a \neq x) \vee (R \subseteq S \wedge \exists x \in S : a = x \wedge \lvert S\rvert = 1 + \lvert R\rvert)$	0.15	–	–