Unité Mixte de Recherche 5104 CNRS - INPG - UJF
Centre Equation
2, avenue de VIGNATE
F-38610 GIERES
tel : +33 456 52 03 40
fax : +33 456 52 03 50
http://www-verimag.imag.fr

Deciding Safety and Termination Properties for Flat Programs with Lists

Marius Bozga and Radu Iosif

Report n^o TR-2006-5

September 28, 2006

Reports are downloadable at the following address
http://www-verimag.imag.fr

Deciding Safety and Termination Properties for Flat Programs with Lists

Marius Bozga and Radu Iosif

September 28, 2006

Abstract

In this paper we analyze the complexity of checking safety and termination properties, for a very simple, yet non-trivial, class of programs with singly-linked list data structures. Since, in general, programs with lists are known to have the power of Turing machines, we restrict the control structure, by forbidding nested loops and destructive updates. Surprisingly, even with these simplifying conditions, programs working on heaps with more than one cycle are undecidable, whereas decidability can be established when the input heap may have at most one loop. The proofs for both the undecidability and the decidability results rely on non-trivial number-theoretic results.

Keywords:

Reviewers:

Notes:

How to cite this report:

@techreport { ,
title = { Deciding Safety and Termination Properties for Flat Programs with Lists},
authors = { Marius Bozga and Radu Iosif},
institution = { Verimag Technical Report },
number = {TR-2006-5},
year = { },
note = { }
}

1 Introduction

The design of automatic verification methods for programs manipulating dynamic linked data structures is a challenging problem. Indeed, the analysis of the behavior of such programs requires reasoning about complex data structures that have general graph-like representations. There are several approaches for tackling this problem addressing different subclasses of programs and using different kinds of formalisms for representing and reasoning about infinite sets of heap structures, e.g., [21, 17, 24, 10].

We consider in this paper the class of programs manipulating linked data structures with a single data-field selector. It corresponds to programs manipulating linked lists with the possibility of sharing and circularities. It is well-known that programs handling lists can simulate, for instance, 2-counter machines, when the control structure is unrestricted. A customary approach to finding decidable classes of counter automata is to consider flat control structures that is, no nested loops are allowed [14, 15, 7]. The decidability of the reachability and termination problems for counter automata is usually established by reduction to the validity problem of Presburger arithmetic [22].

We analyze the problems of deciding safety and termination for programs with lists, assuming the flatness condition on the control structure. Since this restriction is generally not enough, we assume moreover that the program does not perform assignments to selector fields (destructive updates). That is, a program can only traverse the input data structure, but not modify it. We found out that, surprisingly, even this restricted class of programs is undecidable. By further restricting the input heap to at most one cycle, we can establish the decidability of checking both safety and termination properties. The proof relies on the encoding of the set of configurations reachable by the program, as a formula in a decidable fragment of the theory of addition and divisibility [23], that is described in [11].

Let us present in more detail the results. We start with the observation that flat programs with lists can be used to encode the solutions of general Diophantine systems. The existence of such solutions is a well-known undecidable problem, a.k.a *Hilbert's Tenth Problem* [20]. Our reduction uses simple flat programs to encode the $z = x + y$ and $z = [x, y]$ (least common multiple) relations, relying on the fact that multiplication can be defined using only addition and least common multiple.

The source of undecidability lies exactly in the complexity of the input data structure. We noticed that the least common multiple relation can only be encoded by programs running on input structures with at least two (separate) cycles. This observation leads to a decidability result, by imposing that the input heap has at most one cycle. We obtain decidability by first representing the program with lists as a counter automaton. The idea of modeling general programs with singly-linked lists as counter automata, originates in [8, 3]. However, due to the restricted form of our programs, we define a different encoding than the one described in [8, 3], that uses deterministic actions on counters, and preserves the flatness of the control structure. In consequence, we reduce the safety and termination problems from programs with lists to flat counter automata. Finally, we show that, for the latter we can effectively compute the exact loop invariants, using the decidable theory of [11]. In this way, we reduce the original problems of checking safety and termination to verifying validity of formulae in a known decidable logic.

1.1 Related Work

Programs manipulating singly-linked lists have gained a lot of attention within the past two years, as shown by the fairly large number of recent publications on the subject [2, 5, 19, 1, 10]. Interestingly, the idea of abstracting away all the list segments with no incoming edges is common to many of these works, even though they are independent and use different approaches and frameworks (e.g. static analysis [19], predicate abstraction [1], symbolic reachability analysis [2] and proof search [5]). The fact that the number of sharing points in abstract heap structures is bounded by the number of variables in the program is also behind the techniques proposed in [19, 10].

The work that is probably closest to ours has been reported in [8] and [3]. However, the authors' concerns there were rather to develop a general framework for the analysis of programs with lists, than to assess the complexity of the verification problems. Their translation of programs into counter automata uses a generic scheme, which works in the presence of destructive updates. Our translation method is pointed towards programs without destructive updates, the main reason for this being that of establishing decidability. Other closely related work is the one of Chakaravarthy [12], reporting on the undecidability of the points-to analysis in non-flat programs with scalar variables, for which the generated memory configurations are of the same type as in the case of singly-linked lists. Moreover, a reduction from Hilbert's Tenth Problem is also used to prove undecidability, however this result does not hold against the flatness condition on the control structure.

2 Preliminaries

2.1 Programs with Lists

$$\begin{array}{rcl}
l \in Lab; & & u, v, i, j \in PVar \\
Program & := & \{l : Stmnt;\}^* \\
Stmnt & := & WhileStmnt \mid IfStmnt \mid AsgnStmnt \mid Assert \\
WhileStmnt & := & \text{while } Guard \text{ do } \{AsgnStmnt;\}^* \text{ od} \\
IfStmnt & := & \text{if } Guard \text{ then } \{Stmnt;\}^* \left[\text{else } \{Stmnt;\}^*\right] \text{ fi} \\
AsgnStmnt & := & u := \text{null} \mid u := \text{new} \mid u := v \mid u := v.next \mid u.next := \text{null} \mid u.next := v \\
Assert & := & \text{assert}(Guard) \\
Guard & := & u = v \mid u = \text{null} \mid \neg Guard \mid Guard \wedge Guard \mid Guard \vee Guard \mid \text{true}
\end{array}$$

Figure 1: Abstract Syntax of Flat Programs with Lists

We consider imperative programs working with a set of pointer variables *PVar*. The pointer variables refer to list cells. Pointers can be used in assignments such as `u := null`, `u:= v` and `u := v.next`, `u.next := v` and `u.next := null`, and new cell creation `u:= new`. The control structure is composed of iteration (`while`) statements and conditionals (`if-then-else`), and is supposed to be *flat*, meaning that there are no further conditionals or iterations inside a `while` loop. This syntactic restriction is sufficient to ensure that the control flow graph of the program has no nested loops. The guards of the control constructs are pointer equality `u = v`, undefinedness `u = null` tests, and boolean combinations of the above. The `assert` statement has no effect if the condition is true, otherwise the program is sent to an error state.

An assignment statement is said to be a *destructive update* if it is of the form `u := new`, `u.next := v` or `u.next := null`. These are the only statements that can modify a heap structure. Programs without destructive updates can only traverse the heap, but not modify it.

The semantics of programs with lists is defined in terms of heap updates. For a detailed presentation, the reader is referred to [9]. Formally, a heap is a rooted graph in which each node has at most one successor. In the rest of the paper, for a set A we denote by $A_\perp$ the set $A \cup \{\perp\}$. The element $\perp$ is used to denote that a (partial) function is undefined at a given point, e.g. $f(x) = \perp$.

Definition 1 *Let PVar be a set of pointer variables. A* heap *is a tuple $H = \langle N, S, V, Roots \rangle$, where N is a finite set of nodes, $S : N \to N_\perp$ is a* successor function*, $V : PVar \to N_\perp$ is a function associating nodes to variables, and $Roots \subseteq PVar$ is a set of* root variables.

Intuitively, the nodes represent heap-allocated cells, the successor function describes the position of the `next` selectors, for each node, and the variable mapping keeps track of which nodes are directly pointed to by program variables. The set of roots denotes special *entry points* in the heap, which will be used mainly in Section 4 for technical purposes. For now, we consider the following conditions, that must be satisfied by any program P, operating on a heap $\langle N, S, V, Roots \rangle$:

- for all $r \in Roots$, $V(r)$ is defined,
- P does not change the values of the variables in $Roots$, and
- all nodes in N are reachable via S, from a node pointed to by a variable in $Roots$.

Technically, the conditions above are not real limitations, since any program with lists can be transformed into a program that meets the requirements concerning $Roots$. In particular, the third point can be ensured by keeping a free list pointed to by a root variable, and linking all nodes that become unreachable from the other program variables (garbage nodes) into it.

A heap is said to be n-cyclic if it contains exactly n different cycles. Notice that, since each node in the heap can have at most one selector, each cycle must reside in a separate part of the heap. A node in the heap is said to be a *cut point* if it either has two distinct predecessors, or it is pointed to by a root variable. A *list segment* is a sequence of nodes $n_1, n_2, \ldots, n_k$ related by the successor function ($S(n_i) = n_{i+1}, 1 \leq i < k$), such that either (i) n_1 and n_k are the only cut points in the sequence, or (ii) n_1 is the only cut point and $S(n_k) = \perp$. It has been proved in e.g. [8] that the number of cut points is bounded by twice the number of root variables. As a consequence, the number of list segments is bounded by the number of roots. In the following, we will denote by $ls_H(n,m)$ the list segment that lies between the cut points n and m in H, or $ls_H(n,\perp)$, if the last node of the list segment has a null successor. The subscript may be omitted when it is not needed or obvious from the context. If the two cut points are distinct not directly connected (either they are disconnected or there are other cut points in between) we consider that $ls(n,m) = \emptyset$. If $V(u) = n$ and $V(v) = m$, for some $u, v \in PVar$, we may also denote $ls(n,m)$ by $ls(u,v)$. The length of a list segment $ls(n,m)$, i.e. the number of nodes it contains, is denoted by $|ls(n,m)|$.

2.2 Arithmetic of Integers

The undecidability of first-order arithmetic of natural numbers occurs as a consequence of Gödel's Incompleteness Theorem [16], discovered by A. Church [13]. Consequences of this result are the undecidability of the theory of natural numbers with *multiplication and successor function* and with *divisibility and successor function*, both discovered by J. Robinson in [23]. To complete the picture, the existential fragment of the full arithmetic i.e., *Hilbert's Tenth Problem* was proved undecidable by Y. Matiyasevich [20]. The interested reader is further pointed to [6] for an excellent survey of the (un)decidability results in arithmetic.

On the positive side, the decidability of the arithmetic of natural numbers with addition and successor function $\langle \mathbb{N}, +, 0, 1 \rangle$ has been shown by M. Presburger [22], result which has found many applications in modern computer science, especially in the field of automated reasoning. Another important result is the decidability of the *existential* theory of addition and divisibility, proved independently by A. P. Beltyukov [4] and L. Lipshitz [18]. Namely,it is shown that formulas of the form $\exists x_1, \ldots \exists x_n \bigwedge_{i=1}^{K} f_i(\mathbf{x}) | g_i(\mathbf{x})$ are decidable, where f_i, g_i are linear functions over $x_1, \ldots x_n$ and the symbol $|$ means that each f_i is an integer divisor of g_i when both are interpreted over $\mathbb{N}^n$. The decidability of formulas of the form $\exists x_1, \ldots \exists x_n \varphi(\mathbf{x})$, where φ is an open formula in the language $\langle +, |, 0, 1 \rangle$, is stated as a corollary in [18]. This theory will be denoted further by $\langle \mathbb{N}, +, |, 0, 1 \rangle^{\exists}$.

A generalization of this result has been presented in [11], involving the class of formulae of the form $QzQ_1x_1 \ldots Q_mx_m\varphi(\mathbf{x}, z)$, where $Q, Q_i \in \{\forall, \exists\}$ and φ is a boolean combination of formulae of the form $f(z)|g(\mathbf{x}, z)$, and arbitrary Presburger formulae. In other words, the first variable occurring in the quantifier prefix is the only variable allowed to occur to the left of the divisibility sign. The decidability of this class of formulae, denoted by $\mathcal{L}_{|}^{(1)}$, has been established in [11] using quantifier elimination, by reduction to Presburger arithmetic.

However, the result on $\langle \mathbb{N}, +, |, 0, 1 \rangle^{\exists}$ remains one of the strongest decidability results in integer arithmetic. It can be shown that, even formulae involving one universal quantifier, i.e. of the form $\exists x_1, \ldots \exists x_n \forall y \; \varphi(\mathbf{x}, y)$ are undecidable. This is done using the classical definition of the least common multiple relation $[x, y] = z$ [23]: $\forall t \; x|t \wedge y|t \leftrightarrow z|t$. The undecidability of this fragment is a direct consequence of the following[1]:

Theorem 1 *The satisfiability and validity problems for the quantifier-free fragment of the theory $\langle \mathbb{N}, +, [] \rangle$ of natural numbers with addition and the least common multiple relation are undecidable.*

Proof: By reduction from Hilbert Tenth Problem. Let $S \equiv \{P_i(\mathbf{x}) = 0\}_{i=1}^{n}$ be a Diophantine system, and assume the problem of whether S has a strictly positive solution, $\mathbf{x} = \langle x_1, \ldots, x_k \rangle$, which is undecidable. By introducing a sufficient number of variables, we can write S equivalently, as a system of equations of the form either $x + y = z$ or $xy = z$. For any strictly positive $x, y, z \in \mathbb{N}$, we have $y = x^2$ iff $y + x = [x, x+1] = x(x+1)$, and $z = xy$ iff $4z = (x+y)^2 - (x-y)^2$. Consequently, one can write S using only addition and least common multiple. Since no quantifiers have been used in the reduction, the result follows from the undecidability of Hilbert's Tenth Problem. □

2.3 Counter Automata

A counter automaton with n counters is a tuple $A = \langle \mathbf{x}, Q, \rightarrow \rangle$, where $\mathbf{x} = \{x_1, \ldots, x_n\}$ are the counter variables, Q is a finite set of control states, and $\rightarrow \in Q \times \Phi \times Q$ are the transitions, and Φ is the set of arithmetic formulae with free variables from $\{x_i, x_i' \mid 1 \leq i \leq n\}$. A configuration of a counter automata with n counters is a tuple $\langle q, \nu \rangle$, where ν is a mapping from $\mathbf{x}$ to $\mathbb{N}$. The transition relation is defined by $(q, \nu) \rightarrow (q', \nu')$ iff there exists a transition $q \xrightarrow{\varphi} q'$ such that, if σ is an assignment of the free variables of φ (denoted in the following by $FV(\varphi)$), such that, for all $x \in \mathbf{x}$, $\sigma(x) = \nu(x)$ and $\sigma(x') = \nu'(x)$, we have that $\varphi(FV(\varphi)\sigma)$ holds and $\nu(x) = \nu'(x)$, for all variables x with $x' \notin FV(\varphi)$. A *run* of A is a sequence of configurations $(q_0, \nu_0), (q_1, \nu_1), \ldots$ such that $(q_i, \nu_i) \rightarrow (q_{i+1}, \nu_{i+1})$, for each $i \geq 0$.

The control graph of a counter automaton A is the graph having as vertices the set Q of control states, and, for any two states q and q', there is an edge between q and q' in the control graph if and

---

[1] Theorem 1 gives a simple proof of the undecidability of $\langle \mathbb{N}, +, |, 0, 1 \rangle$ that is different from the one published by J. Robinson in [23]. However, the undecidability of Hilbert's Tenth Problem, which is used here was not known in 1949.

only if there exists a transition $q \xrightarrow{\varphi} q'$ in A. A counter automaton is said to be *flat* if its control graph has no nested loops.

3 Undecidable Flat List Programs

In this section we define the safety and termination properties for various classes of flat list programs with possibly unbounded input, and prove their undecidability. A decidable subclass is defined in the next section. Before proceeding, we need to introduce several notions.

Definition 2 *A tuple of strictly positive natural numbers* $\mathbf{n} \in \mathbb{N}^k$ *is said to be* encoded *by a heap* $H = \langle N, S, V, Roots \rangle$, *denoted as* $H(\mathbf{n})$, *if and only if there exists two mappings* $e : \{1, \ldots, k\} \rightarrow Roots$ *and* $f : \{1, \ldots, k\} \rightarrow Roots_\perp$ *such that, for all* $1 \leq i \leq k$, $n_i = |ls_H(e(i), f(i))|$.

In other words, each number is represented by a list segment in between two root variables, or between a root variable and $\perp$. Notice that the condition $n_i > 0$ for all $1 \leq i \leq k$ implies that $e(i)$ and $f(i)$ actually delineate a non-trivial list segment.

Definition 3 *Two heaps* $H = \langle N, S, V, Roots \rangle$ *and* $H' = \langle N', S', V', Roots \rangle$ *are said to* share the same structure, *denoted by* $H \simeq H'$ *if and only if for all* $r_1, r_2 \in Roots$ $ls_H(r_1, r_2) \neq \emptyset \iff ls_{H'}(r_1, r_2) \neq \emptyset$.

In other words, H and H' differ only by the lenghts of their list segments that are delineated by roots. Notice that $\simeq$ is an equivalence relation on heaps. This leads to a notion of *parametric heap* $H(\mathbf{x})$, defined as the infinite set of heaps that share the same structure, with respect to a set of variables $\mathbf{x} = \{x_1, \ldots, x_k\}$, ranging over natural numbers. Given any interpretation $\mathbf{x} \mapsto \mathbf{n}$, we have that $H(\mathbf{n}) \in H(\mathbf{x})$. In other words, $H(\mathbf{x})$ is the equivalence class of $H(\mathbf{n})$ with respect to $\simeq$. By $ls^{x_i}(u,v)$ we denote the set $\{ls(u,v) \mid n_i = |ls(u,v)| \text{ in some } H(n_1, \ldots, n_i, \ldots, n_k) \in H(x_1, \ldots, x_i, \ldots, x_k)\}$. For instance, in Figure 2 (a), $ls^x(u,v)$ in $H(x,y,z)$ denotes all list segments that encode the values of x, and $ls^y(v,\perp)$, $ls^z(w,\perp)$ encode all possible values of y and z, respectively.

We consider the following definition of *safety properties*:

Definition 4 *Let* P *be a flat list program,* $S = \{l_i \,:\, assert(\varphi_i)\}_{i=1}^k$ *a set of statements occurring in* P, *and* $H(\mathbf{x})$ *a parametric heap.* P *is said to be* safe w.r.t $H(\mathbf{x})$ and S *if and only if for all heaps* $G \in H(\mathbf{x})$, *and* $1 \leq i \leq k$, φ_i *is true whenever* P*, started on input* G*, reaches* l_i.

The above property is vacuously true if the given program never reaches any of the locations in S. In order to cover this case, we consider the following definition of *termination*, with respect to a parametric heap.

Definition 5 *Let* P *be a flat list program, and* $H(\mathbf{x})$ *a parametric heap.* P *is said to* terminate w.r.t $H(\mathbf{x})$ *if and only if for all heaps* $G \in H(\mathbf{x})$, P *started on input* G*, has a finite execution.*

Notice that Definition 4 corresponds to partial correctness, whereas the combination of Definitions 4 and 5 can express total correctness, as understood in the setting of program verification using Hoare logic.

In order to prove undecidability of safety and termination for flat list programs, with respect to parametric heaps, we shall use the undecidability of the *validity* problem for the quantifier-free fragment of the theory of addition and least common multiple $\langle \mathbb{N}, +, [] \rangle$, which is stated by Theorem

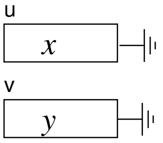
1. The reduction is as follows: given a quantifier-free formula φ of $\langle \mathbb{N}, +, [] \rangle$, we build a flat list program P and a parametric heap $H(\mathbf{x})$, such that φ is valid if and only if P is safe w.r.t $H(\mathbf{x})$. The same reasoning is done for termination. This leads to undecidability results for both the safety and termination problems, as defined in the previous.

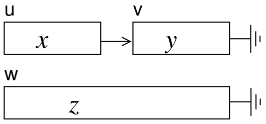
The key of the reduction is to use three basic programs, $P_{x=y}$, $P_{x+y=z}$ and $P_{[x,y]=z}$ (Figure 2), that encode the atomic formulae $x = y$, $x + y = z$ and $[x, y] = z$, respectivelly. Each program works on a heap of a predefined shape, also shown in Figure 2. The program $P_{x=y}$, in Figure 2 (a) is guaranteed to terminate, since both the lists pointed to by u and v are acyclic. Moreover, if i and j are both null at the end, the lists have equal length.

The program in Figure 2 (b) is guaranteed to terminate because both lists pointed to by u and w are acyclic. Moreover, at the end line, both i and j are null if and only if both lists have equal length, which is only the case if and only if $x + y = z$. In this case the variable v plays the only role of splitting the list segment pointed to by u into $ls^x(u, v)$ and $ls^y(v, \perp)$.

The program in Figure 2 (c) terminates because eventually $i = u$ and $j = v$ at the same time. In fact this happens after a number of loop iterations equal to the least common multiple of x and y. Then k is null at the end if and only if the length of the list pointed by w equals this number, i.e. $[x, y] = z$.

φ	P_φ	C_φ	H_φ

$x = y$	i := u;
	j := w;
	while i $\neq$ null $\wedge$ j $\neq$ null do
	i := i.next;
	j := j.next;
	od;
		i = null
		$\wedge$
		j = null
			

$x + y = z$	i := u;
	j := w;
	while i $\neq$ null $\wedge$ j $\neq$ null do
	i := i.next;
	j := j.next;
	od;
		i = null
		$\wedge$
		j = null
			

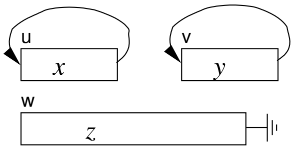
$[x, y] = z$	i := u.next;
	j := v.next;
	k := w.next;
	while (i $\neq$ u $\vee$ j $\neq$ v)
	$\wedge$ k $\neq$ null do
	i := i.next;
	j := j.next;
	k := k.next;
	od;
		k = null
		$\wedge$
		i = u
		$\wedge$
		j = v
			

Figure 2: Basic Programs

Let us consider now a quantifier-free formula $\varphi(\mathbf{x})$ in the language of $\langle \mathbb{N}, +, [] \rangle$. Since we are interested in reducing the validity problem, i.e $\forall \mathbf{x} \, . \, \varphi(\mathbf{x})$, it is sufficient to consider w.l.o.g. that φ is a disjunction of atomic formulae of the forms $x = y$, $x + y = z$ or $[x, y] = z$ and their negations. Let $\varphi = \bigvee_{i=1}^{n} \psi_i$, where ψ_i is either (1) $x_i = y_i$, (2) $x_i + y_i = z_i$, (3) $[x_i, y_i] = z_i$ or their negations, for $x_i, y_i, z_i \in \mathbf{x}$. For each condition of the form (2) or (3) the input heap contains a separate heap as in Figure 2 with roots u_i, v_i and w_i. Then the program encoding the validity of φ has the following structure:

```
$P_{\psi_1}$;
if $C_{\neg\psi_1}$ then $P_{\psi_2}$;
  if $C_{\neg\psi_2}$ then $P_{\psi_3}$;
    ...
    assert(false);
    ...
  fi
fi
```

where, for all $1 \leq p \leq n$ we have:

- if ψ_p is a positive literal, P_{ψ_p} and C_{ψ_p} are as in Figure 2.
- if ψ_p is a negative literal, P_{ψ_p} is $P_{\neg\psi_p}$ and C_{ψ_p} is $\neg C_{\neg\psi_p}$.

Moreover, the program has to test that all list segments encoding occurrences of the same variable are of the same length. This can be done in the beginning, using a sequence of flat programs of the same kind as $P_{x=y}$, and is skipped for brevity reasons.

For any heap that corresponds to the parameterized input, the above program reaches the `assert(false)` statement if and only if the input encodes a tuple of numbers that falsifies all disjuncts of the original formula. Hence the program is safe if and only if for all instance $H(\mathbf{n})$ of the parametric input heap $H(\mathbf{x})$, $\mathbf{n}$ satisfies at least one clause ψ_i, hence φ is valid. This proves the undecidability of the safety problem.

To show undecidability of the termination problem, we use the same reduction, with the only difference that the `assert(false)` statement is replaced by a non-terminating loop `while(true) do ... od`. The program then terminates if and only if φ is valid.

Notice further that the least common multiple relation has been encoded using an input heap with at least two separate cycles. The above considerations lead to the following Theorem:

Theorem 2 *The problems of verifying safety and termination properties, for flat list programs without destructive updates, running on n-cyclic inputs, with $n > 1$, are undecidable.*

3.1 Extensions of the Undecidability Results

The properties of safety and termination for list programs parameterized by the shape of their input are universally quantified properties (see Definition 4 and 5). The following *reachability* property is existential:

Definition 6 *Let P be a flat list program, l a control location of P, and $H(\mathbf{x})$ a parametric heap. l is said to be* reachable *in P w.r.t. $H(\mathbf{x})$ if and only if there exists a heap $G \in H(\mathbf{x})$ such that P, started with input G, eventually reaches l.*

We can show undecidability of the reachability problem by reduction from the satisfiability problem for the quantifier-free fragment of $\langle \mathbb{N}, +, [] \rangle$ (Theorem 1). The reduction is similar to the one in the previous section.

Theorem 3 *The problem of verifying reachability for flat list programs without destructive updates, running on n-cyclic input, with* $n > 1$ *is undecidable.*

Proof: Let $\varphi = \bigwedge_{i=1}^{n} \psi_i$, where ψ_i is either (1) $x_i = y_i$, (2) $x_i + y_i = z_i$, (3) $[x_i, y_i] = z_i$ or their negations, for $x_i, y_i, z_i \in \mathbf{x}$. We use the basic programs in Figure 2 as follows:

```
$P_{\psi_1}$;
if $C_{\psi_1}$ then $P_{\psi_2}$;
  if $C_{\psi_2}$ then $P_{\psi_3}$;
    ...
    l: ... ;
    ...
  fi
fi
```

where P_ψ and C_ψ are defined as in the previous. Notice that l is reachable if and only if there exists a heap in the input (defined also as in Figure 2) such that all ψ_i are verified $1 \leq i \leq k$. Undecidability follows directly from Theorem 1. □

Up to now, we have considered separately the problems of verifying safety and termination properties for programs parameterized by the shape of the input heap, and abstracting away the exact lengths of the list segments. We show now how these results can be extended to verifying properties of programs with either unknown shape, or empty input heap.

Definition 7 *Let P be a flat list program, and* $S = \{l_i \ : \ assert(\varphi_i)\}_{i=1}^{k}$ *a set of statements occurring in P. P is said to be* correct w.r.t *S if and only if for all heaps H, P started on input H, reaches location* l_i*, and* φ_i *is true whenever the control is at* l_i*, for all* $1 \leq i \leq k$*.*

The problem of correctness of a program P with unknown input can be shown undecidable by reducing the safety problem for programs on parameterized heaps to it. Namely, given $P, H(\mathbf{x})$ and $S = \{l_i \ : \ \mathrm{assert}(\varphi_i)\}_{i=1}^{n}$ a set of the statements in P, we can build a program Q such that P is a subset of Q, and P is safe w.r.t. $H(\mathbf{x})$ and S if and only if Q is correct w.r.t. S. In order to obtain Q, we prefix P with a program T, i.e. $Q = T; P$. The role of T is to test that the input heap is an instance of $H(\mathbf{x})$. In case this test succeeds, the control is passed on to P, otherwise, T (and implicitly Q) does not terminate.

In order to build the tester program T, we assume that each cut point in $H(\mathbf{x})$ is labeled by a root variable. This restriction does not prevent us from applying the undecidability result of Theorem 2, since the reduction from the previous section can performed using only heaps fulfilling this condition, according to Figure 2. Thus we assume that each list segment is marked by two root variables. For each $ls(u, v)$, with $u, v \in Roots$, T will test if v is the first root variable reachable starting from u:

```
i := u;
while $\bigwedge_{w \in Roots} i \neq w$ do
      i := i.next;
od
assert(i = v);
```

Note that this program might not terminate, in case when the given input heap is not an instance of $H(\mathbf{x})$, the list pointed to by u is cyclic, and the starting point of the loop is not properly marked by a root variable.

Corollary 1 *The correctness problem for flat list programs is undecidable.*

The other problem for which we show undecidability, based on the previous results, is the reachability problem for non-deterministic flat list programs, started on empty heap. A non-deterministic program uses undefined guards of the conditional statements, i.e. `while * do ... do` or `if * then ... else ... fi`.

Definition 8 *Let P be a non-deterministic flat list program, and l a control location of P. l is said to be* reachable on empty heap *if and only if P, started with the empty heap, has at least one execution path leading to l.*

We show undecidability of the reachability problem on empty heap, by reduction from the reachability problem on parametric input heap (Definition 6). Namely, given a program P and a parametric heap $H(\mathbf{x})$, we build a program Q as sequential composition of a (non-deterministic) constructor program C and P, i.e. $Q = C;P$, such that for a given location l of P, P reaches l w.r.t. $\mathbf{H}(\mathbf{x})$ if and only if l is reachable on empty heap. Intuitively, C is a flat non-deterministic program with dynamic creation and destructive updates, that will create an arbitrary instance of $H(\mathbf{x})$. For each list segment $ls^x(u,v)$ of $H(\mathbf{x})$, C will have a loop of the form:

```
i := u;
i.next := new; i := i.next;
while * do
        i.next := new; i := i.next;
od
if v = null then v := i;
else i.next := v;
fi
```

Note that each distinct path through the loop generates a list segment of a different length. Consequently, each path through C will generate a different instance of $H(\mathbf{x})$. Then there exists an instance $H(\mathbf{n})$ of $H(\mathbf{x})$, such that l is reachable in P started on $H(\mathbf{n})$ if and only if there exists a path through Q that reaches l and viceversa.

Corollary 2 *The problem of reachability on empty heap for non-deterministic flat list programs is undecidable.*

4 Decidability on Acyclic and 1-Cyclic Heaps

As pointed out before, the undecidability of the safety and termination problems for programs parameterized by the shape of the input heap relies on the fact that the input heap has at least two loops. In this section, we prove that, by restricting the input heap to have at most one loop, both problems become decidable. In practice, this result provides a precise and fully automated way of analyzing simple programs with list iterators, i.e. variables that can only traverse a list, but not modify it.

The tool for proving decidability is a sub-fragment of the arithmetic of addition and divisibility $\langle \mathbb{N}, +, |, 0, 1 \rangle$, namely the class of formulae of the form $QzQ_1x_1 \ldots Q_mx_m\varphi(\mathbf{x}, z)$, where φ is a boolean combination of divisibility predicates of the form $f(z)|g(\mathbf{x}, z)$ and Presburger constraints. The restriction here is that z is the only variable occurring to the left of the divisibility sign. This fragment, called $\mathcal{L}_{|}^{(1)}$, has been shown decidable in [11].

4.1 From List Programs to Counter Automata

Let P be a flat list program, and $H(\mathbf{x})$, $\mathbf{x} = \{x_1, \ldots, x_k\}$ be a parametric heap with at most one cycle. Since P is flat, its control structure has a finite number of branches, and each branch is a finite sequence of simple loops, connected via linear paths. Assume that, for each loop, one can describe the relation between the input and output heaps, after any number of iterations. Then the input-output relation of the whole program can be described as a finite composition of relations. In order to compute this relation, we simulate P by a counter automaton A, and reduce both the safety and termination problems for P to safety and termination problems for A.

Let $Roots = \{r_1, \ldots, r_p\}$ be the set of root variables of $H(\mathbf{x})$, and $H = \langle N, S, V, Roots \rangle$ be an instance of $H(\mathbf{x})$. We recall upon the fact that each node $n \in N$ must be reachable from a node pointed to by a variable from $Roots$. Moreover, each variable from $PVar$, that is not a root variable, can be assigned by P. Since the structure of the heap does not change during the execution of P, the current configuration of the program can be represented only by recording the position of the variables from $PVar \setminus Roots$ in the structure. Let u be such a variable. If $V(u)$ is defined, i.e. $V(u) = n \in N$, then n must be reachable from at least one root variable, call it r_i, for some $1 \leq i \leq p$. The number of steps on the path between $V(r_i)$ and $V(u)$ is denoted by δ_i. The pair of integers $\langle i, \delta_i \rangle$ gives the exact position of u in H, see e.g. Figure 3 (right). Obviously, this encoding of the position is not unique, since u may be reachable from more than one root variable, and there might be more than one path from r_i to u, due to the possible presence of a cycle. In the following, let $root(u)$ and $dist(u)$ denote the first and the second elements of the pair encoding the position of u.

The counter automaton corresponding to P is $A = \langle \mathbf{x} \cup \mathbf{y}, Q, \rightarrow \rangle$, where the set of counters consists of a set of parameters $\mathbf{x}$ and a set of working counters $\mathbf{y} = \{y_1, \ldots, y_r\}$, i.e. one working counter for each variable from $PVar \setminus Roots = \{u_1, \ldots, u_r\}$, and the set of control states $Q = Lab \times \{1, \ldots, p\}^r$. A configuration of A is a tuple $\langle q, \delta_1, \ldots, \delta_r \rangle \in Q \times \mathbb{N}^r$, where:

- $q = \langle l, \rho_1, \ldots, \rho_r \rangle$ represents the current program label, and the current roots of the iterator variables of P.
- δ_i is the distance of u_i from its root ρ_i, for each $1 \leq i \leq r$.

In principle, the counter y_i keeps track of $dist(u_i)$, w.r.t $root(u_i)$. A transition between two configurations $c = \langle \langle l, \rho \rangle, \delta \rangle$ and $c' = \langle \langle l', \rho' \rangle, \delta' \rangle$ is triggered by the execution of a program statement $l : s; l'$, and is denoted by $c \xrightarrow{s} c'$. The table in Figure 3 (left) summarizes the transition relation for all non-destructive assignment statements from Figure 2.1.

The most interesting case is $u_i := u_j$.next, which is depicted in Figure 3 (right). Notice that all assignment statements are encoded by deterministic transitions in the counter automaton. Since the program P is supposed to be flat, the resulting counter automaton will also have a flat control structure.

A guard condition of the form $u_i =$ null is encoded by an arithmetic constraint on the position of u. We distinguish between three situations:

- $root(u_i) = 0$, e.g. because of an assignment $u_i :=$ null,

Assignment	Control change	Counter update
$u_i :=$ null	$\rho'_i = 0$	$y'_i = 0$
$u_i := u_j$	$\rho'_i = \rho_j$	$y'_i = y_j$
$u_i := u_j$.next	$\rho'_i = \rho_j$	$y'_i = y_j + 1$

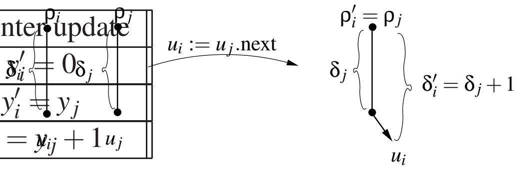


Figure 3: Semantics of Assignments

- $root(u_i) \neq 0$ is the origin of a path that ends in a cycle, in which case $u_i =$ null is false,
- the path starting with $root(u_i) \neq 0$ is finite, and let π_i denote the set of list segments on this path. In this case, we have: $y_i > \sum_{ls^x(n,m) \in \pi_i} x$, meaning that u_i has gone beyond the end of the path.

Due to the fact that the encoding of the variables is not unique, a pointer equality condition of the form $u_i = u_j$ has a more complex encoding, which is going to be detailed next. The fact that the parametric structure of $H(\mathbf{x})$ is known, is playing an important role. Suppose that $u_i = u_j$ is true for some arbitrary instance $H = \langle N, S, V, Roots \rangle$ of $H(\mathbf{x})$, i.e $V(u_i) = V(u_j) = n_0 \in N$. We distinguish two cases, as shown in Figure 4:

- n_0 does not belong to a cycle in H. In this case, there is a unique path from $V(root(u_i))$ to n_0, and a unique path from $V(root(u_j))$ to n_0. Let $ls^{x_0}(m_1, m_2)$ be the list segment on which n_0 resides, π_i be the set of list segments on the path from $V(root(u_i))$ to m_1, and π_j be the set of list segments on the path from $V(root(u_j))$ to m_1. Consequently, we have:

  $$0 \leq y_i - \sum_{ls^x(n,m) \in \pi_i} x = y_j - \sum_{ls^x(n,m) \in \pi_j} x \leq x_0$$

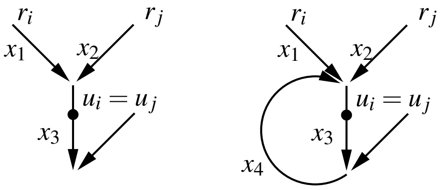
  For instance, the guard corresponding to the configuration in Figure 4 (a) is: $0 \leq y_i - x_1 = y_j - x_2 \leq x_3$.

- n_0 belongs to the only cycle in H. Let γ denote the set of list segments in the cycle, and $\pi_{i,j}$ denote the paths from $V(root(u_{i,j}))$ to the beginning of γ, respectively. Then we have:

  $$\Big(\sum_{ls^x(n,m) \in \gamma} x \Big) | \Big((y_i - \sum_{ls^x(n,m) \in \pi_i} x) - (y_j - \sum_{ls^x(n,m) \in \pi_j} x) \Big)$$

  As an example, the guard corresponding to the configuration in Figure 4 (b) is: $x_3 + x_4 | (y_i - x_1) - (y_j - x_2)$

The semantics of a pointer equality condition in the program with lists can be written as a finite disjunction of all possible configurations, which will fall into one of the cases above. This formula denotes all possible values of $\mathbf{x}$ and $\mathbf{y}$ for which $u_i = u_j$ in $H(\mathbf{x})$. Therefore, a boolean condition on pointers, of the form $\neg Guard$ or $Guard \diamond Guard$, for $\diamond \in \{\wedge, \vee\}$ can be translated to a counter automaton guard, by replacing all atomic propositions with the corresponding formulae on counters. Notice that, since $H(\mathbf{x})$ has at most one cycle, all divisibility predicates will have the same expression on the left-hand side.



(a) (b)

Figure 4: Two Cases of Equality between Pointers

4.2 Reasoning about Counter Automata

Our translation scheme associates one program statement exactly one action on counters, therefore the resulting counter automaton A preserves the control structure of the original program P. In particular, if P was flat, A is also flat. The goal of this section is to compute, for a given control location q of A, the relation between the input values of the counters and the values at q. Since A is flat, it is sufficient to compute, for each loop, the input-output relation after n iterations of the loop, and define global input-output relations by composition. The safety and termination properties are decidable if this relation can be expressed in a decidable logic. We shall use here the $\mathcal{L}_{|}^{(1)}$ fragment of $\langle \mathbb{N}, +, |, 0, 1 \rangle$ [11], explained in Section 2.2.

By construction, all transitions of A are of the form $q \xrightarrow{\varphi(\mathbf{x},\mathbf{y},\mathbf{y}')} q'$ where φ is of the form:

$$\varphi(\mathbf{x},\mathbf{y},\mathbf{y}') \ : \ \psi(\mathbf{x},\mathbf{y}) \ \wedge \bigwedge_{1 \le i,j \le r} y'_i = b_i y_j + c_i \tag{1}$$

with ψ a boolean combination of divisibility predicates of the form $f(\mathbf{x})|g(\mathbf{x},\mathbf{y})$ (the same f occurs everywhere to the left of $|$) and Presburger constraints, $b_i \in \{0,1\}$ and $c_i \in \mathbb{Z}$, for all $1 \le i \le r$.

It can be easily shown that this class of relations is closed under composition, defined as:

$$(\varphi_1 \circ \varphi_2)(\mathbf{x},\mathbf{y},\mathbf{y}') = \exists \mathbf{y}'' \ \varphi_1(\mathbf{x},\mathbf{y},\mathbf{y}'') \ \wedge \ \varphi_2(\mathbf{x},\mathbf{y}'',\mathbf{y}')$$

In other words, the existential quantifiers above can be eliminated, the result being written as another relation of the same form. As a consequence, we can assume without losing generality, that each control path $q_1 \xrightarrow{\varphi_1} q_2 \ldots q_{n-1} \xrightarrow{\varphi_{n-1}} q_n$, with no incoming or outgoing transitions, is equivalent to a single transition $q_1 \xrightarrow{\varphi_1 \circ \ldots \circ \varphi_{n-1}} q_n$.

Without losing generality, we consider that A consists of only two transitions:

$$q \xrightarrow{\psi(\mathbf{x},\mathbf{y}) \ \wedge \ \bigwedge_{1 \le i,j \le r} y'_i = b_i y_j + c_i} q \text{ and } q \xrightarrow{\neg\psi(\mathbf{x},\mathbf{y})} q'$$

Here the variables $\mathbf{x}$ are meant as parameters, while $\mathbf{y}$ are the working counter variables. Let $I(n,\mathbf{x},\mathbf{y},\mathbf{y}')$ denote the relation between the input ($\mathbf{y}$) and the output ($\mathbf{y}'$) values of the counters after exactly n iterations of the loop, where $\mathbf{x}$ are the values of the parameters. For the moment, let us assume that $I(n,\mathbf{x},\mathbf{y},\mathbf{y}')$ is effectively computable and can be expressed in the quantifier-free fragment of $\mathcal{L}_{|}^{(1)}$.

A safety property for a counter automaton can be described by a pair $\langle q, \phi(\mathbf{x},\mathbf{y}) \rangle$, where $\phi(\mathbf{x},\mathbf{y})$ is a formula expressible in the quantifier-free fragment of $\mathcal{L}_{|}^{(1)}$, with the following meaning: for all valuations of the parameters, whenever the control reaches the location q, the values of the counters must satisfy ϕ. Moreover, let us assume that all atomic predicates in I and φ satisfy the condition that only variables from $\mathbf{x}$ may appear to the left of the divisibility sign, and moreover, that only one linear combination $f(\mathbf{x})$ can occur in this position. With the assumptions above, the safety problem reduces to checking the validity of the formula:

$$\sigma \stackrel{\Delta}{=} \forall \mathbf{x} \forall \mathbf{y} \forall \mathbf{y}' \forall n \; . \; I(n, \mathbf{x}, \mathbf{y}, \mathbf{y}') \rightarrow \phi(\mathbf{x}, \mathbf{y}')$$

Termination is the problem whether the counter automaton reaches its final control location, for every valuation of the parameters. In our case, this is equivalent to the validity of:

$$\theta \stackrel{\Delta}{=} \forall \mathbf{x} \exists n \exists \mathbf{y} \exists \mathbf{y}' \; . \; I(n, \mathbf{x}, \mathbf{y}, \mathbf{y}') \wedge \neg \varphi(\mathbf{x}, \mathbf{y}')$$

We can prove the validity of σ and θ by proving that their negations are contradictions. For instance, $\neg\sigma$ is expressible in the decidable fragment of $\mathcal{L}_{|}^{(1)}$ [11], as: $\exists z \; . \; \neg\sigma[z/f(\mathbf{x})] \wedge z = f(\mathbf{x})$. Same is done for θ. In order to prove decidability of safety and termination for counter automata, it is sufficient to show how to express I as a quantifier-free formula of $\mathcal{L}_{|}^{(1)}$. This is achieved in the proof of the following Theorem:

Theorem 4 *The safety and termination problems for flat counter automata with transitions of the form (1) are decidable.*

Proof: The formula describing the updates of the counters can be equivalently written as $\mathbf{y}' = B\mathbf{y} + C$, where $B \in \{0,1\}^r \times \{0,1\}^r$ is a square matrix, $B[i][j] = 1$ if and only if $y'_i = y_j + c_i$ and $C \in \{0,1\}^r$ is a column vector, $C[i] = c_i$, $1 \leq i, j \leq r$. Notice that each line of B has at most one occurrence of 1. Because of this, the set $\mathcal{B} = \{B^i \mid i \geq 0\} = \{B^0, B^1, \ldots, B^{k-1}\}$ is finite, and effectively computable. Moreover the powers of B take values from $\mathcal{B}$ in a periodic fashion: Let p represent the prefix and q the period of the set $\mathcal{B}$, where $p + q = k$. We have:

$$B^i = \begin{cases} B^i & \text{if } i \leq p \\ B^{p+(i-p) \bmod q} & \text{otherwise} \end{cases}$$

$$C_i \stackrel{\Delta}{=} C \sum_{j=0}^{i-1} B^j = C\Big(\sum_{j=0}^{p} B^j + \sum_{j=p+1}^{p+(i-p) \bmod q} B^j + \frac{i-p}{q} \sum_{j=p+1}^{q} B^j \Big)$$

where $\frac{a}{b}$ denotes integer division. As a result of these considerations, the formula describing the values of the counters after exactly n iterations has the following form:

$$I(n, \mathbf{x}, \mathbf{y}) \; : \; \mathbf{y} = B^n \mathbf{y}_0 + C_n \wedge \forall m \; . \; 0 \leq m \leq n \rightarrow \psi(\mathbf{x}, B^m \mathbf{y}_0 + C_m)$$

The role of the second conjunct is to ensure that, at each iteration prior to n the transition guard $\psi(\mathbf{x}, \mathbf{y})$ has been respected by the values of the counters.

It is clear, from this expression, that $I(n, \mathbf{x}, \mathbf{y})$ satisfies the requirement concerning the occurrence of only variables from $\mathbf{x}$ to the left of $|$, within divisibility predicates, because ψ does. Consequently, both safety and termination properties of the counter automata generated from flat list programs without destructive updates, working on heaps with at most one cycle, can be expressed in $\mathcal{L}_{|}^{(1)}$, and hence, are decidable. □

The decidability of safety and termination for programs with lists is consequence of Theorem 4:

Corollary 3 *The problems of verifying safety and termination properties, for flat list programs without destructive updates, running on acyclic and* 1*-cyclic inputs, are decidable.*

5 Conclusions

We addressed the problems of verifying safety and termination properties for programs handling singly-linked lists, without destructive update assignments, and whose control structure is flat. We found out that, despite the strong syntactic restrictions, these programs, parameterized by the size of the input heap, have the expressive power of Turing machines. These undecidability results are a consequence of the complexity of the input data structures, even when the program does not change the structure. By further limiting the input heaps to at most one cycle, we obtain decidability of the safety and termination problems. All our results rely on non-trivial number-theoretic arguments.

References

[1] I. Balaban, A. Pnueli, and L. Zuck. Shape Analysis by Predicate Abstraction. In *VMCAI*, volume 3385 of *LNCS*, 2005. 1.1

[2] S. Bardin, A.Finkel, and D. Nowak. Toward Symbolic Verification of Programs Handling Pointers. In *AVIS*, Barcelona, Spain, 2004. 1.1

[3] Sebastien Bardin, Alain Finkel, Etienne Lozes, and Arnaud Sangnier. From pointer systems to counter systems using shape analysis. In Springer Verlag, editor, *Proc. 5th International Workshop on Automated Verification of Infinite-State Systems (AVIS)*. LNCS, 2006. 1, 1.1

[4] A. P. Beltyukov. Decidability of the universal theory of natural numbers with addition and divisibility. *Zapiski Nauch. Sem. Leningrad Otdeleniya Mathematical Institute*, 60:15 – 28, 1976. 2.2

[5] J. Berdine, C. Calcagno, and P. O'Hearn. A Decidable Fragment of Separation Logic. In *FSTTCS*, volume 3328 of *LNCS*, 2004. 1.1

[6] Alexis Bés. A survey of arithmetical definability. *A Tribute to Maurice Boffa. Bulletin de la Société Mathématique de Belgique*, 1 - 54, 2002. 2.2

[7] B. Boigelot. On iterating linear transformations over recognizable sets of integers. *TCS*, 309(2):413–468, 2003. 1

[8] A. Bouajjani, M. Bozga, P. Habermehl, R. Iosif, P. Moro, and T. Vojnar. Programs with lists are counter automata. In Springer Verlag, editor, *Proc. Computer Aided Verification (CAV)*. LNCS, 2006. 1, 1.1, 2.1

[9] A. Bouajjani, M. Bozga, P. Habermehl, R. Iosif, P. Moro, and T. Vojnar. Programs with lists are counter automata. Technical Report TR-2006-3, VERIMAG, 2006. 2.1

[10] A. Bouajjani, P. Habermehl, P. Moro, and T. Vojnar. Verifying Programs with Dynamic 1-Selector-Linked Structures in Regular Model Checking. In *TACAS*, volume 3440 of *LNCS*, 2005. 1, 1.1

[11] M. Bozga and R. Iosif. On decidability within the arithmetic of addition and divisibility. In Springer Verlag, editor, *Proc. Foundations of Software Science and Computation Structures (FOSSACS)*, volume 3441, pages 425 – 439. LNCS, 2005. 1, 2.2, 4, 4.2, 4.2

[12] Venkatesan T. Chakaravarthy. New results on the computability and complexity of points-to-analysis. In Springer Verlag, editor, *Proc. International Conference on Principles of Programming Languages (POPL)*. LNCS, 2003. 1.1

[13] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345 – 363, 1936. 2.2

[14] Hubert Comon and Yan Jurski. Multiple Counters Automata, Safety Analysis and Presburger Arithmetic. In *Proc. CAV*, volume 1427 of *LNCS*, pages 268 – 279. Springer, 1998. 1

[15] A. Finkel and J. Leroux. How to compose presburger-accelerations: Applications to broadcast protocols. In *Proc. FST&TCS*, volume 2556 of *LNCS*, pages 145–156. Springer, 2002. 1

[16] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38:173 – 198, 1931. 2.2

[17] S. Ishtiaq and P. O'Hearn. BI as an assertion language for mutable data structures. In *POPL*, 2001. 1

[18] Leonard Lipshitz. The diophantine problem for addition and divisibility. *Transaction of the American Mathematical Society*, 235:271 – 283, January 1976. 2.2

[19] R. Manevich, E. Yahav, G. Ramalingam, and M. Sagiv. Predicate Abstraction and Canonical Abstraction for Singly-Linked Lists. In *VMCAI*, volume 3385 of *LNCS*, 2005. 1.1

[20] Yuri Matiyasevich. Enumerable sets are diophantine. *Journal of Sovietic Mathematics*, 11:354 – 358, 1970. 1, 2.2

[21] A. Møller and M.I. Schwartzbach. The Pointer Assertion Logic Engine. In *PLDI*, 2001. 1

[22] Mojzesz Presburger. Über die Vollstandigkeit eines gewissen Systems der Arithmetik. *Comptes rendus du I Congrés des Pays Slaves*, Warsaw 1929. 1, 2.2

[23] Julia Robinson. Definability and decision problems in arithmetic. *The Journal of Symbolic Logic*, 14(2):98 – 114, June 1949. 1, 2.2, 1

[24] S. Sagiv, T.W. Reps, and R. Wilhelm. Parametric Shape Analysis via 3-Valued Logic. *TOPLAS*, 2002. 1

A Proofs