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

Quantitative Verification of Programs with Lists

Marius Bozga and Radu Iosif

Report n^o TR-2005-2

January 26, 2005

Quantitative Verification of Programs with Lists

Marius Bozga and Radu Iosif

January 26, 2005

Abstract

In this paper we study imperative programs that handle list objects, performing unrestricted selector updating operations. For this class of programs, we address the problem of *quantitative verification*, by reasoning about arithmetic relations between the lengths of certain lists. We first show undecidability of the general problem, by reduction from the reachability problem for two-counter machines. Second, we introduce a decidable property specification logic and we study the complexity of the decision problem. Finally, we propose a semi-algorithmic method for the verification of Hoare-Floyd style properties of programs with lists. Using this method, we can reduce the invariant computation problem for a program with lists to an acceleration problem for integer counter automata.

Keywords: Hoare logic, first-order arithmetic, quantitative shape analysis, counter automata.

Reviewers:

Notes:

How to cite this report:

@techreport { ,
title = { Quantitative Verification of Programs with Lists},
authors = { Marius Bozga and Radu Iosif},
institution = { Verimag Technical Report },
number = {TR-2005-2},
year = { },
note = { }
}

1 Introduction

The last decade has witnessed unprecedented advances in the design of safety critical embedded systems, that demand more and more complex implementation solutions. As a consequence, dynamic memory allocation and pointers are being used even in mission critical software, to improve the system's memory usage and timing constraints. However, programming with pointers is a great source of errors, in fact maybe the greatest danger of imperative programming.

In this paper we address the problem of verification for safety properties expressed in a logic that allows reasoning about *quantitative shape properties* i.e., properties that combine shape with size information. We consider imperative programs that manipulate simply-linked dynamic lists implemented using only one selector field (next). These systems are inherently infinite state, since the sizes of the lists held by a program can grow arbitrarily. Moreover, list handing programs usually deal with most typical problems of shape analysis e.g. *aliasing* (lists can share a segment) or *circularity* (lists with an end loop). However the lack of branching in a shape graph allows to handle the undecidable problem of verification in an elegant semi-algorithmic fashion. We adopt the Floyd-Hoare verification style [Hoa69, Flo67], by annotating the safety properties within the program code, then use a weakest precondition calculus and invariant generation to infer a finite number of logical entailments, whose validity can be checked automatically.

Our method is based on an equivalence relation, of finite index, between shape graphs. Namely, we encode the length of each non-trivial path with no incoming edges, by a parameter variable. The number of structures obtained in this way, referred to as a *parametric list graphs*, is bounded by an exponential in the number of program variables. Also, the *parametric model checking problem* i.e., checking whether there exists an instance of a parametric list graph satisfying the formula, is decidable. Therefore, deciding validity of entailments is possible, by exhaustive enumeration of parametric models. The invariant generation technique takes also advantage from the fact that parametric list graphs occur in finite number for a given program. In fact, each strongest invariant can be expressed as a disjunction of formulas characterizing parametric graphs (up to isomorphism and renaming of variables) and second-order arithmetic constraints. In general, the latter constraints are not decidable, therefore we suggest using approximative techniques such as widening [CH78] and semi-algorithmic acceleration [CJ98].

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 [MYRS05, BPZ05, BFN04, BCO04]. Interestingly, even though the aforementioned techniques are independent e.g. static analysis [MYRS05], predicate abstraction [BPZ05], acceleration [BFN04] and proof search [BCO04], the idea of abstracting away all the list segments with no incoming edges is common to all of them. In fact, the main reason these approaches work in theory is that the abstraction range is finite (yet bounded by an exponential in the number of program variables). The same idea is used in this paper, this time in the setting of *quantitative verification* i.e. reasoning about the lengths of paths in the program heap. Work on quantitative program analysis is also fairly recent. For instance, [Rug04] proposes a static analysis able to establish that an AVL tree is balanced, whereas in [ZSM04], the decidability of a logic for reasoning about the number of cells in a recursive data structure is studied. In addition, our work makes an attempt towards a complete verification method involving a logical representation of states, a decision procedure, a weakest precondition calculus and an invariant generation technique.

Roadmap The paper is organized as follows. Section 2 proves the undecidability of the quantitative verification problem in general. Section 3 introduces the property specification logic that is the core of our method, and Section 4 discusses decidability of this logic on list models. Section 5 introduces the semi-algorithmic approach based on weakest precondition inference and invariant discovery. Finally, Section 6 concludes.

2 Undecidability of the Quantitative Verification Problem

We consider a simple imperative language whose syntax is given in Figure 2. Note that the statements of most object-oriented languages can be precompiled in this form, possibly by introducing fresh temporary variables. Here $u, v \in V$ denote program variables, and $\rightsquigarrow$ is our

$$
\begin{array}{rcl}
l & \in & \mathbb{N} \\
u, v & \in & V \\
Prog & := & \{l : Stmnt;\}^* \\
Stmnt & := & IfStmnt \mid AsgnStmnt \mid GotoStmnt \\
IfStmnt & := & \text{if } Guard \text{ then } \{Stmnt;\}^* \ \left[(\text{else } \{Stmnt;\}^*)\right] \\
Guard & := & u \rightsquigarrow^* \diamond v \rightsquigarrow^* \ \mid \neg u \rightsquigarrow^* \diamond v \rightsquigarrow^* \\
AsgnStmnt & := & u \ = \text{new} \mid u \rightsquigarrow = v \mid u = v \rightsquigarrow^* \\
GotoStmnt & := & \text{goto } l
\end{array}
$$

Figure 1: A Simple Pointer Language

notation for the "next" selector field. The diamond operator $\diamond$ is yet another particular notation for pointer aliasing i.e., equality of addresses held by the two pointers. The first statement allocates a fresh object for further uses. The second statement updates the "next" the object pointed to by the right-hand side variable. Finally, the last statement assigns the variable u to the object pointed to by $v \rightsquigarrow \ldots \rightsquigarrow$.

2.1 Reduction from the Halting Problem for Minsky Machines

A 2-counter machine [Min67] with non-negative counters c_1, c_2 is a sequential program:

$$1 : \ \texttt{ins}_1; 2 : \ \texttt{ins}_2; \cdots ; n : \ \texttt{ins}_n;$$

where $\texttt{ins}_n$ is a $\texttt{halt}$ instruction and $\texttt{ins}_i$ with $i = 1, 2, \cdots, n-1$ are instructions of the following two types (assuming $1 \leq k, k_1, k_2 \leq n, 1 \leq j \leq 2$):

1. $c_j = c_j + 1;$ `goto` $k;$
2. `if` $c_j = 0$ then goto k_1 else $(c_j = c_j - 1;$ goto $k_2);$

The machine C starts executing at label 0 with values $c_1 = c_2 = 0$. When it reaches the control location k, it executes the instruction $\mathtt{ins}_k$ i.e, it modifies the values of the counter and jumps to the next label according to the instruction. The machine halts if it reaches the `halt` instruction at label n. We use the following well-known theorem:

Proposition 1 *It is undecidable if a given 2-counter machine halts on the zero input.*

Minsky machines are easily modeled using programs with lists. Each counter c_i $(i = 1, 2)$ can be directly encoded by a pointer variable p_i pointing to a non-circular list such that the value of c_i is equal to the length of the list. The two instructions of the machine are modeled as follows:

1. $u = new;\ u \rightsquigarrow = p_j;\ p_j = u$; goto k;
2. if $\neg p_j \Diamond p_j$ then goto k_1 else $(p_j = p_j \rightsquigarrow;$ goto $k_2;)$

The reduction gives the undecidability of the halting (reachability) problem for programs that handle only simply-linked lists. In particular, any alias property of the program i.e., any condition of the form $u \Diamond v$ is undecidable, for we can always append the assignment $u = v$ at the end of the program and ask whether this statement is reachable. The latter result is stronger than the general undecidability of aliasing [Lan92], [Ram94], for the following reason. The existing proofs use reductions from the halting problem for Turing machines [Lan92] and Post Correspondence Problem [Ram94] and need at least two selectors in the reduction proof. Our result holds for only one selector, denoted by $\rightsquigarrow$.

3 A Property Specification Logic

In order to express shape and quantitative properties of the dynamic memory of programs performing selector updating operations, we have defined a specification logic called *alias logic with counters* [BIL04]. Formulas in this logic are interpreted over finite directed graphs. Since we are only concerned about programs that generate and manipulate list structures, we consider that each node in a graph has at most one successor. Formally such a *list graph* is a triple $G = \langle N, V, s \rangle$, where:

- N is a finite set of nodes,
- $s : N \rightharpoonup N$ is a partial successor function, and
- $V \subseteq N$ is a designated set of nodes called *variables*. The requirement is that for no $n \in N$ we have $s(n) \in V$. In other words, the graph is *rooted* on V.

A *path* is a finite sequence $\pi = v \rightsquigarrow \ldots \rightsquigarrow \in V \times \{\rightsquigarrow\}^*$, also denoted by $v \overset{k}{\rightsquigarrow}$, where $k \in \mathbb{Z}$ is the number of occurrences of the $\rightsquigarrow$ symbol in π. The semantics of a path in a graph $G = \langle N, V, s \rangle$ is the node it leads to. Since s is required to be a partial mapping, there can be at most one such node for each path. Formally, we denote by $[\![\pi]\!]$ the semantics of $\pi = v \overset{k}{\rightsquigarrow}$ in G (where G is always clear from the context) as follows: $[\![v \overset{k}{\rightsquigarrow}]\!]$ is undefined if $k < 0$, $[\![v]\!] = v$ and $[\![v \overset{k}{\rightsquigarrow}]\!] = s([\![v \overset{k-1}{\rightsquigarrow}]\!])$ for $k > 1$. We say that two paths π_1 and π_2 are *aliased* if $[\![\pi_1]\!], [\![\pi_2]\!]$ are defined and $[\![\pi_1]\!] = [\![\pi_2]\!]$. A *quantitative path* is of the form $\pi(\vec{x}) = v \overset{f(\vec{x})}{\rightsquigarrow}$, where $\vec{x}$ is a finite set of variables, interpreted over $\mathbb{N}$, and $f : \mathbb{N}^{|\vec{x}|} \to \mathbb{Z}$ is a linear function with integer coefficients on $\vec{x}$. Given an interpretation of variables $\iota : \vec{x} \to \mathbb{N}$, the interpretation of a quantitative path π, denoted as $\iota(\pi)$, is the path $v \overset{f(\iota(\vec{x}))}{\rightsquigarrow}$.

A *base theory* B is any consistent subtheory of the first order arithmetic of natural numbers, usually denoted by $\langle \mathbb{N}, 0, S, +, \cdot \rangle$ [End72]. A logic of *aliases with counters* over B is the theory $AL_B = \langle V, B, \Diamond \rangle$, where $V = \{u, v, w, \ldots\}$ is a set of *program variable symbols* and $\Diamond$ is an *alias operator*. The formulas of AL_B are defined as follows:

- any formula of B is a formula of AL_B,
- if $\pi_1(\vec{x})$ and $\pi_2(\vec{x})$ are quantitative paths, then $\pi_1 \Diamond \pi_2$ is a formula of AL_B,
- if φ, ψ are formulas of AL_B, then $\varphi \wedge \psi$, $\neg\varphi$ and $\forall x \varphi$ are formulas of AL_B,
- nothing else is a formula of AL_B.

Given an interpretation of variables, an alias proposition $\pi_1 \Diamond \pi_2$ holds in a graph if the interpretations of the quantified paths involved are defined and they "meet" in the same node i.e., $[\![\iota(\pi_1)]\!] = [\![\iota(\pi_2)]\!]$. The satisfaction of a closed formula φ on a graph G, denoted as $G \models \varphi$, is defined recursively on the syntax of φ, as usual. Notice that the atomic proposition $\pi \Diamond \pi$ means that $[\![\pi]\!]$ is a defined node in the graph. We shall further denote this by $\delta(\pi)$.

In the rest of the paper we shall consider two base theories. The first one is the additive theory of natural numbers $\langle \mathbb{N}, 0, S, + \rangle$, also known as Presburger Arithmetic (PA) [Pre29]. The second one is the theory of natural numbers with addition and divisibility $\langle \mathbb{N}, 0, S, +, | \rangle$ (AD) [Rob49]. While the first theory is well known in the literature, the second one needs some explanation. A formula in this language is interpreted over $\mathbb{N}$ in the standard way: $+$ denotes the addition of natural numbers, $|$ is the divisibility relation, and $0, 1$ are the constants zero and one. In particular, we consider that $0|0$, $0 \not| \, n$ and $n|0$, for all $n \in \mathbb{N} \setminus \{0\}$. In the following we will intentionally use the same notation for a mathematical constant symbol and its interpretation, as we believe, no confusion will arise from that. In the following, we shall use AL_{PA} exclusively for the specification of properties and AL_{AD} for the weakest precondition calculus that is developed in Section 5.1.

We also define the $\mathcal{L}_|^{(*)}$ fragment of $\langle \mathbb{N}, 0, S, +, | \rangle$, which will be mentioned later in this paper. Let $\mathcal{L}_|^{(n)}$denotes the class of formulas, where only the first n variables (in prenex form) appear to the left of the divisibility sign in an atomic proposition. Formally, $\mathcal{L}_|^{(n)}$is the set of formulas $Q_1 z_1 \dots Q_n z_n R_1 x_1 \dots R_m x_m \varphi(\vec{x}, z)$ where $Q_i, R_j \in \{\exists, \forall\}$ and the only divisibility propositions are $f(\vec{z}) | g(\vec{x}, \vec{z})$, with f and g linear functions. The set $\mathcal{L}_|^{(*)}$is the union of all $\mathcal{L}_|^{(n)}$, for $n \geq 1$. We denote by $\exists\mathcal{L}_|^{(*)}$ the subset of $\mathcal{L}_|^{(*)}$ obtained by setting, in each $\mathcal{L}_|^{(n)}$, $Q_i \equiv \exists$, for all $1 \leq i \leq n$ and, by $\exists\mathcal{L}_|^{(*)+}$, the set of all formulas of $\exists\mathcal{L}_|^{(*)}$ in which no divisibility proposition occurs within the scope of a logical negation.

Example

The use of alias logic with counters is perhaps better understood by means of an example. Consider the program **REVERSE** in Figure 2. The input of the program is a non-cyclic list pointed to by the variable i, and the output is a non-cyclic list, pointed to by j, containing the cells of the first list in reverse order. In order to prove that the output list is non-cyclic and has the same length as the input, we need to state the following loop invariant:

- both i and j point to disjoint non-cyclic lists, and
- the sum of their lengths equals the length of the input list.

A non-cyclic list of length x and pointed to by a variable v, is specified using the following predicate:

$$NCLIST(v, x) = \delta(v \overset{x}{\rightsquigarrow}) \wedge \neg\delta(v \overset{x+1}{\rightsquigarrow})$$

Note that the condition $\neg\delta(v \overset{x+1}{\rightsquigarrow})$ is sufficient to ensure that the list is non-cyclic and of length less than or equal to x. The fact that two lists have at least an element in common can be stated using the following predicate:

$$SHARE(u, v) = \exists x \exists y \; u \overset{x}{\rightsquigarrow} \diamond v \overset{y}{\rightsquigarrow}$$

In order to check that both input and output lists have the same length L, one needs to assert the following loop invariant, at line [1]:

$$\exists x', x'' \; . \; NCLIST(i, x') \wedge NCLIST(j, x'') \wedge x' + x'' = L \wedge \neg SHARE(i, j)$$

Running the program on few small inputs is enough to convince the reader that the above formula is indeed the loop invariant needed for proving the validity of the Hoare triple:

$$\{NCLIST(i, L)\} \; \mathbf{REVERSE} \; \{NCLIST(j, L)\}$$

The interested reader can also refer to [BOB] for a gentle introduction to the functioning of this program. $\square$

```
1 if ¬ i ◇ i goto 7 (else
2    k = i⇝;
3    i⇝ = j;
4    j = i;
5    i = k;
6    goto 1;)
7 halt.
```

Figure 2: List Reversal Program

4 Decidability of Alias Logic with Counters

To decide whether a closed formula φ in alias logic with counters has a model, we use a notion of *parametric graph* $G(\vec{z})$ over a set of variables $\vec{z}$, which is an abstraction of an infinite class of graphs. A formal definition of a parametric graph is given next. The important point is that, since we consider only list graphs with one selector field, the total number of parametric graphs is finite. In fact, this number depends only on the number of program variables. Hence, the satisfiability problem is reduced to deciding whether there exists $z_1, \ldots, z_n$ such that $G(\vec{z}) \models \varphi$. To solve the latter problem, we shall derive an open formula $\Psi_{G,\varphi}(\vec{z})$ within a decidable fragment of the arithmetic of natural numbers with addition and divisibility [Rob49]. The constructed formula is such that, for all interpretations $\iota : \vec{z} \to \mathbb{N}$, $\Psi_{G,\varphi}(\iota(\vec{z}))$ holds if and only if $G(\iota(\vec{z})) \models \varphi$. The formula φ is then satisfiable, if and only if there exists a parametric graph G such that $\exists z_1, \ldots \exists z_n \Psi_{G,\varphi}$ is satisfiable. The latter problem has been dealt with in [BI04]. However the solution of that paper, due to its generality, is of non-elementary complexity. In [BI05] we present a decision procedure that terminates in triple exponential time for the class of arithmetic formulas generated from alias logic.

4.1 A Parametric Model Checking Problem

A parametric list graph over a set of variables $\vec{z}$ is a graph $G = \langle N, V, s \rangle$, the only difference w.r.t. the previous definition being the introduction of an edge alphabet which is a finite (yet large enough) set of variables $\vec{z} = \{z_1, z_2, \ldots\}$. We have now $s : N \times \vec{z} \to N$, in other words, each edge is of the form $n \overset{z}{\rightsquigarrow} m$, with $n, m \in N$ and $z \in \vec{z}$. We assume moreover that each edge is labeled with a different variable from $\vec{z}$, hence $\|s\| \leq \|\vec{z}\|$. Given an interpretation of variables $\iota : \vec{z} \to \mathbb{N}$, we define the interpretation of an edge to be the sequence of edges $n = n_1 \rightsquigarrow n_2 \rightsquigarrow \ldots n_k = m$ of length $k = \iota(z)$. The interpretation of a graph is the graph obtained by replacing each edge with its interpretation. A parametric graph $G(\vec{z})$ is a model of an alias logic formula φ, denoted as $G(\vec{z}) \models \varphi$ if and only if there exists an interpretation ι of the variables in $\vec{z}$ such that $G(\iota(\vec{z})) \models \varphi$ in the non-parametric sense, defined in the previous.

Definition 1 *A parametric graph $G = \langle N, V, s \rangle$ is said to be in* normal form *if and only if:*

- *there are no two adjacent edges $m \overset{z_1}{\rightsquigarrow} n \overset{z_2}{\rightsquigarrow} p$, for some $n, m, p \in N$ and $z_{1,2} \in \vec{z}$, such that the indegree of n is greater than one.*

- *each node in the graph is reachable from a root node in V.*

Notice that each parametric graph can be put in normal form by replacing any pair of edges violating this condition by a single edge labeled with a fresh variable. This can be done without affecting the satisfiability of formulas in alias logic, as stated by the following:

Lemma 1 *Let φ be any closed formula of alias logic with counters. There exists a parametric graph $G(\vec{z})$ such that $G(\vec{z}) \models \varphi$ if and only if there exists a parametric graph $H(\vec{z})$ in normal form such that $H(\vec{z}) \models \varphi$.*

Proof: H is obtained from G by replacing each maximal sequence of adjacent edges $n_1 \overset{z_1}{\rightsquigarrow} n_2 \overset{z_2}{\rightsquigarrow} \ldots \overset{z_{k-1}}{\rightsquigarrow} n_k$, such that the indegree of $n_{2,\ldots,k-1}$ are equal to one, by an edge $n_1 \overset{z}{\rightsquigarrow} n_k$, where z is a fresh variable. Since $G(\vec{z}) \models \varphi$, there exists an interpretation of $\vec{z}$, namely $\iota : \vec{z} \to \mathbb{N}$, such that $G(\iota(\vec{z})) \models \varphi$. Extend the satisfying interpretation ι to ι' such that $\iota'(z) = \Sigma_{i=1}^{k}\iota(z_i)$ and $\iota'(z') = \iota(z')$ for all $z' \in \vec{z} \setminus \{z\}$. Then $G(\iota(\vec{z})) = H(\iota'(\vec{z}))$ and the conclusion follows. $\Box$

As a convention, we implicitly identify any two parametric graphs that are isomorphic and differ only by a renaming of the label variables. The interested reader may also consult [BFN04] for a notion very similar to the parametric graph.

Parametric list graphs enjoy an important property, which is the key to proving decidability for alias logic with counters. The total number of parametric list graphs in normal form, over a given set of variables, is finite. This fact occurs as consequence of the following lemma:

Lemma 2 *Let $G = \langle N, V, s\rangle$ be a parametric list graph in normal form. Then $\|N\| \leq 2\|V\|$.*

Proof: For a set of nodes $M \subset N$, let $succ(M) = \{n \mid m \to n\}$ denote the set of immediate successors of M, and, $fr_i(M) = succ^i(M) \setminus succ^{i-1}(M)$, where $succ^i(M)$ denotes the i-th application of the $succ$ function to M, for $i > 1$. By convention, we take $fr_0(M) = M$. Since each node is reachable from V, we have $N \subseteq \bigcup_{i\geq 0} succ^i(V) = \bigcup_{i\geq 0} fr_i(V)$, therefore $\|N\| \leq \Sigma_{i\geq 0}\|fr_i(V)\|$. Let $fr_k^{>1}(M)$, $fr_k^{=1}(M)$ be the sets of nodes from $fr_k(M)$ with two or more predecessors, and with one predecessor respectively. Obviously, $\|fr_k(M)\| = \|fr_k^{>1}(M)\| + \|fr_k^{=1}(M)\|$. A node with only one predecessor and one successor clearly violates the first condition of Definition 1, hence each node from $fr_k^{=1}(V)$ has no successors for all $k \geq 0$, so we have $\|fr_k^{>1}(V)\| \leq \frac{\|fr_{k-1}^{>1}(V)\| - \|fr_k^{=1}(V)\|}{2}$ for $k > 1$, and $\|fr_1^{>1}(V)\| \leq \frac{\|fr_0(V)\| - \|fr_1^{=1}(V)\|}{2}$. Summing up, we obtain:

$$\begin{gathered}
\Sigma_{i>1}\|fr_i^{>1}(V)\| \leq \frac{\|fr_0(V)\|}{2} + \Sigma_{i>1}\frac{\|fr_i^{>1}(V)\|}{2} - \Sigma_{i>1}\frac{\|fr_i^{=1}(V)\|}{2} \\
\Sigma_{i>1}\frac{\|fr_i^{>1}(V)\|}{2} + \Sigma_{i>1}\frac{\|fr_i^{=1}(V)\|}{2} \leq \frac{\|fr_0(V)\|}{2} \\
\Sigma_{i\geq 0}\|fr_i(V)\| \leq 2\|fr_0(V)\| \\
\|N\| \leq 2\|V\|
\end{gathered}$$

$\Box$

Given a parametric graph and a closed formula in alias logic, we are interested in finding an open formula $\Psi_{G,\varphi}(\vec{z})$ that encodes $G(\vec{z}) \models \varphi$, for all possible interpretations of $\vec{z}$. We will define $\Psi_{G,\varphi}$ inductively on the structure of φ, by first defining characteristic formulas for the alias literals (alias propositions and negations of alias propositions). Intuitively, $\pi_1 \Diamond \pi_2$ holds on $G(\vec{z}) = \langle N, V, s\rangle$ if and only if the paths π_1 and π_2 meet either in an "explicit" node $n \in N$ or in a node that does not occur in N but is "abstracted" within a parametric edge. For the latter case, we need some notation. Given an interpretation ι of variables $\vec{z} \cup \{y\}$, let $d(n, y)$ denote the node situated at "distance" $\iota(y)$ from n in the (non-parametric) graph $G(\iota(\vec{z}))$. With this notation, Figure 3 defines the characteristic formulas $\Psi_{G,l}$, for alias literals l.

$$G \models \pi_1 \Diamond \pi_2 \quad : \quad \bigvee_{n \in N} [\![\pi_1]\!] = n \wedge [\![\pi_2]\!] = n \vee \exists y \bigvee_{n \overset{z}{\rightsquigarrow} m} [\![\pi_1]\!] = d(n,y) \wedge [\![\pi_2]\!] = d(n,y) \wedge y < z$$

$$G \not\models \pi_1 \Diamond \pi_2 \quad : \quad \exists y_1 \exists y_2 \bigvee_{\substack{n_1 \overset{z_1}{\rightsquigarrow} m_1 \\ n_2 \overset{z_2}{\rightsquigarrow} m_2 \\ n_1 \neq n_2}} [\![\pi_1]\!] = d(n_1,y_1) \wedge [\![\pi_2]\!] = d(n_2,y_2) \wedge y_1 < z_1 \wedge y_2 < z_2$$
$$\vee \bigvee_{n \overset{z}{\rightsquigarrow} m} [\![\pi_1]\!] = d(n,y_1) \wedge [\![\pi_2]\!] = d(n,y_2) \wedge y_1 < z \wedge y_2 < z \wedge y_1 \neq y_2$$

Figure 3:

Since both positive and negative literals can be encoded as positive boolean combinations of equalities of the form $[\![\pi]\!] = d(n, y)$[1], it is sufficient to show how such an equality can be defined as a positive formula of the arithmetic of addition and divisibility with the only variables occurring on the left of divisibility being the ones in $\vec{z}$. Let $\pi = v \overset{f(\vec{x})}{\rightsquigarrow}$ be a quantitative path, for some $v \in V$. There are three possibilities:

1. if there is no path in G from v to n, then $[\![\pi]\!] = d(n, y)$ is false.

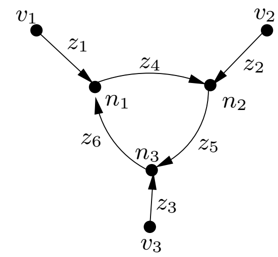
2. if there is an acyclic path $v \overset{z_1}{\rightsquigarrow} n_1 \overset{z_2}{\rightsquigarrow} \ldots n_{k-1} \overset{z_k}{\rightsquigarrow} n$ in G, then $[\![\pi]\!] = d(n, y)$ is equivalent to $f(\vec{x}) = \Sigma_{i=1}^{k} z_i + y$.

3. otherwise, there is a cyclic path $v \overset{z_1}{\rightsquigarrow} \ldots n_{k-1} \overset{z_k}{\rightsquigarrow} n_k = n \overset{z_{k+1}}{\rightsquigarrow} n_{k+1} \ldots n_{l-1} \overset{z_l}{\rightsquigarrow} n_l = n$ in G, and for all $1 \leq i < l$, $i \neq k$ we have $n_i \neq n$. Then $[\![\pi]\!] = d(n, y)$ is equivalent to $f(\vec{x}) \geq \Sigma_{i=1}^{k} z_i + y \wedge \Sigma_{i=k+1}^{l} z_i | f(\vec{x}) - \Sigma_{i=1}^{k} z_i - y$, for the $v \overset{f}{\rightsquigarrow}$ path may iterate through the $n_k, n_{k+1}, \ldots, n_l$ loop multiple times.

Example

The encoding of a query of the form $G(\vec{z}) \models [\![\pi(\vec{x})]\!] = n$ as an arithmetic formula is better understood by means of an example. Figure 4 shows a parametric graph and three sample queries with their equivalent encodings. □

[1] $[\![\pi]\!] = n$ is $[\![\pi]\!] = d(n, 0)$.



$$\widehat{v_1 \overset{x}{\leadsto}} = d(n_1, y) : x \geq z_1 + y \wedge z_4 + z_5 + z_6 | x - z_1 - y$$

$$\widehat{v_1 \overset{x}{\leadsto}} = d(n_2, y) : x \geq z_1 + z_4 + y \wedge z_4 + z_5 + z_6 | x - z_1 - z_4 - y$$

$$\widehat{v_1 \overset{x}{\leadsto}} = d(n_3, y) : x \geq z_1 + z_4 + z_5 + y \wedge z_4 + z_5 + z_6 | x - z_1 - z_4 - z_5 - y$$

Figure 4:

Summarizing the above discussion, we have the following Theorem:

Theorem 1 *The satisfiability problem for the logic of* aliases with counters *on list models is decidable.*

Proof: Let φ be a formula in which all alias propositions are of the form $v_1 \overset{f_1}{\leadsto} \Diamond v_2 \overset{f_2}{\leadsto}$, and let V be the set of all v_i occurring in (a quantitative path of) φ. Using the definitions from Figure 3, we reduce the satisfiability of φ to a positive boolean combination of atomic propositions of the form $[\![v \overset{f}{\leadsto}]\!] = n$. Hence φ has a model if and only if it has a model $\langle N, V, s \rangle$, with the set of roots V, as above. This is true if and only if φ is true over a parametric model with the set of roots V. By Lemma 2, the number of such parametric models is finite, and the parametric model checking problem can be defined in a decidable fragment of the arithmetic with addition and divisibility. The conclusion follows. $\square$

An upper bound for the complexity of the decision procedure for alias logic with counters can be computed as follows. Given a formula φ, the set V of variables that occur in alias propositions is of size $\|V\| \leq 2|\varphi|$. The number of list graphs to be considered is bounded asymptotically by $\|V\|^2 \cdot 2^{\|V\|} \leq 4|\varphi|^2 \cdot 4^{|\varphi|}$, and the size of a characteristic formula $\Psi_{G,\varphi}$ for $G \models \varphi$ is bounded by $|\varphi| \cdot \|G\|^2 \leq 4|\varphi|^3$. According to the result from [BI05], the decision problem for $\Psi_{G,\varphi}$ takes time $O\left(2^{2^{(n^2)^n}}\right)$ in the worst case, where $n = |\Psi_{G,\varphi}|$. So, the entire procedure takes at most $O\left(2^{2^{|\varphi|^{|\varphi|^3}}}\right)$ time.

4.2 Decidability of Arithmetic Formulas (skip on first reading)

The last step towards proving decidability of alias logic with counters on list models is to show that the arithmetic formulas generated from alias formulas are decidable. To start with, we will formally define a decidable fragment of the arithmetic with addition and divisibility, to which the parametric model checking problem previously discussed can be reduced. For a complete presentation of this theory, the interested reader is pointed to [BI04].

A closer look shows that all formulas that are derived by the previously presented method are of the form $\exists \mathcal{L}_{|}^{(*)}$. For technical reasons we shall work with the (equivalent) form (1), where $Q_i \in \{\exists, \forall\}$, $\vec{z} = \{z_1, \ldots, z_n\}$, $\vec{x} = \{x_1, \ldots, x_m\}$, $f_{ij}^k(\vec{z}), g_{ij}(\vec{x}, \vec{z})$ are linear functions, and

$\psi_i(\vec{x}, \vec{z})$ are Presburger formulas. By (A, B) we denote the greatest common divisor of A and B and, since the g.c.d. is associative, we denote $((A, B), C) = (A, (B, C))$ by (A, B, C), which is generalized to $(A_1, \ldots A_n)$.

$$\exists z_1 \ldots \exists z_n Q_1 x_1 \ldots Q_m x_m \bigvee_{i=1}^{N} \Big(\bigwedge_{j=1}^{M_i} \big(\{f_{ij}^k\}_{k=1}^{p_{ij}}\big) | g_{ij} \wedge \psi_i \Big) \tag{1}$$

In particular, the ψ_i formulas can be replaced by Presburger formulas only on $\vec{z}$, by substitutions of variables, using quantifier elimination for the ψ_i (Presburger arithmetic) formulas, as it is done in [BI04]. In the following, we will assume that this has already been done.

Our decision method is based on quantifier elimination. Namely, we eliminate the Q_i quantifiers starting with the innermost one. The result is a formula $\exists z_1 \ldots \exists z_n \exists y_1 \ldots \exists y_p \varphi(\vec{y}, \vec{z})$ of the existential theory of addition and divisibility, which is decidable in double exponential time, according to [Lip76]. In doing so, we consider two cases, based on whether the innermost quantifier is existential or universal.The first case has been developed in [BI04] and is mentioned here for self-containment reasons. The second case is an improvement w.r.t. the method described in [BI04], in the sense that positive formulas can be treated now in worst-case triple exponential time, instead of the non-elementary complexity of the generic method.

The Existential Case

This is a generalization of the Chinese Remainder Theorem (CRT), which states the following:

$$\exists x \bigwedge_{i=1}^{K} m_i | (x - r_i) \iff \bigwedge_{1 \le i,j \le K} (m_i, m_j) | (r_i - r_j)$$

where $m_i \in \mathbb{N}, r_i \in \mathbb{Z}$. The CRT can be generalized as follows:

Corollary 1 *For any integers $m_i \in \mathbb{N}$ and $a_i \in \mathbb{Z} \setminus \{0\}, r_i \in \mathbb{Z}$ with $1 \le i \le K$ we have:* $\exists x \bigwedge_{i=1}^{K} m_i | (a_i x - r_i) \leftrightarrow \bigwedge_{1 \le i,j \le K} (a_i m_j, a_j m_i) | (a_i r_j - a_j r_i) \wedge \bigwedge_{i=1}^{K} (a_i, m_i) | r_i$.

Proof: Let $A = \Pi_{i=1}^{K} a_i$ and $A_i = \frac{A}{a_i}$.

$$\begin{aligned}
\exists x \bigwedge_{i=1}^{K} m_i | (a_i x - r_i) \quad &\longleftrightarrow \quad \exists x \bigwedge_{i=1}^{K} A_i m_i | (Ax - A_i r_i) \longleftrightarrow \exists y \bigwedge_{i=1}^{K} A_i m_i | (y - A_i r_i) \wedge A | y \\
&\overset{CRT}{\longleftrightarrow} \quad \bigwedge_{1 \le i < j \le K} (A_i m_i, A_j m_j) | (A_i r_i - A_j r_j) \wedge \bigwedge_{1 \le i \le K} (A_i m_i, A) | A_i r_i \\
&\longleftrightarrow \quad \bigwedge_{1 \le i,j \le K} (a_i m_j, a_j m_i) | (a_i r_j - a_j r_i) \wedge \bigwedge_{i=1}^{K} (a_i, m_i) | r_i
\end{aligned}$$

$\square$

Let τ_{ij} denote the subterm $\left(\{f_{ij}^k\}_{k=1}^{p_{ij}}\right)$. Since $Q_m \equiv \exists$, the formula (1) becomes:

$$\exists z_{1\ldots n} Q_1 x_1 \ldots Q_{m-1} x_{m-1} \bigvee_{i=1}^{N} \exists x_m \bigwedge_{j=1}^{M_i} \tau_{ij} | g_{ij} \wedge \psi_i \tag{2}$$

W.l.o.g. we can assume that $M_i \neq 0$ for all $1 \leq i \leq N$, and that $g_{ij}(\vec{x}, \vec{z}) = a_{ij} x_m + h_{ij}(\vec{x'}, \vec{z})$, where $\vec{x'} = \vec{x} \setminus \{x_m\}$, and with all coefficients $a_{ij} \neq 0$. Applying Corollary 1 to the i-th disjunct, we obtain (the original i subscript has been omitted throughout):

$$\begin{array}{ll} \exists z_{1\ldots n} Q_1 x_1 \ldots Q_{m-1} x_{m-1} & \bigwedge_{1 \leq k,l \leq M} (a_k \tau_l, a_l \tau_k) | (a_k h_l - a_l h_k) \wedge \\ & \qquad \bigwedge_{1 \leq k \leq M} (a_k, \tau_k) | h_k \wedge \psi \end{array}$$

Now the subterms of the form $(a_k \tau_l, a_l \tau_k)$ and (a_k, τ_k) can be written as $\left(\{f_i\}_{i \in I}\right)$, since the multiplication distributes with the g.c.d, i.e. $C(A, B) = (AC, BC)$. We have thus obtained a formula of the form (1), in which x_m does not occur anymore. The size of the resulting formula is at most quadratic in the size of the input formula, written in disjunctive normal form.

The Universal Case

This case is an application of the pigeonhole principle. Again, we take $g_{ij}(\vec{x}, \vec{z}) = a_{ij} x_m + h_{ij}(\vec{x'}, \vec{z})$, where $\vec{x'} = \vec{x} \setminus \{x_m\}$, with all coefficients $a_{ij} \neq 0$. As before, by τ_{ij} we denote the subterm $\left(\{f_{ij}^k\}_{k=1}^{p_{ij}}\right)$. The formula (1) implies the following (we have just omitted the Presburger conjuncts ψ_i):

$$\exists z_{1\ldots n} Q_1 x_1 \ldots Q_{m-1} x_{m-1} \forall x_m \bigvee_{i=1}^{N} \bigwedge_{j=1}^{M_i} \tau_{ij} | h_{ij} + a_{ij} x_m \tag{3}$$

Since this is true for all interpretations of x_m over $\mathbb{N}$, it must be true for $1, 2, \ldots, N+1$:

$$\begin{array}{llll} \bigwedge_{j=1}^{M_1} \tau_{1j} | h_{1j} + a_{1j} & \vee \ldots \vee & \bigwedge_{j=1}^{M_N} \tau_{Nz} | h_{Nj} + a_{Nj} & \wedge \\ \bigwedge_{j=1}^{M_1} \tau_{1j} | h_{1j} + 2a_{1j} & \vee \ldots \vee & \bigwedge_{j=1}^{M_N} \tau_{Nz} | h_{Nj} + 2a_{Nj} & \wedge \\ & \ldots & & \\ \bigwedge_{j=1}^{M_1} \tau_{1j} | h_{1j} + (N+1)a_{1j} & \vee \ldots \vee & \bigwedge_{j=1}^{M_N} \tau_{Nj} | h_{Nj} + (N+1)a_{Nj} & \end{array}$$

This can be seen as a $(N+1) \times N$ table, where the (k, l)-element is the l-th disjunct of the form $\bigwedge_{j=1}^{M_l} \tau_{lj} | h_{lj} + x_m a_{lj}$ in which x_m is taken to be k. This formula is true only if at least one element of each row is true. Hence there exists at least a column $1 \leq p_0 \leq N$ such that for two different rows $1 \leq q_0 < r_0 \leq N+1$, the (q_0, p_0) and (r_0, p_0) elements are both true. Equivalently, we have:

$$\bigvee_{\substack{1 \leq p_0 \leq N \\ 1 \leq q_0 < r_0 \leq N+1}} \bigwedge_{j=1}^{M_{p_0}} \tau_{p_0 j} | h_{p_0 j} + q a_{p_0 j} \wedge \tau_{p_0 j} | h_{p_0 j} + r_0 a_{p_0 j}$$

$$\Longrightarrow$$

$$\bigvee_{\substack{1 \leq p_0 \leq N \\ 1 \leq q_0 < r_0 \leq N+1}} \bigwedge_{j=1}^{M_{p_0}} \tau_{p_0 j} | (r_0 - q_0) a_{p_0 j}$$

Notice that $(r_0 - q_0)a_{p_0j} \neq 0$, hence the set of possible values for all τ_{p_0j} is included in the following set:

$$\Delta_0 = \bigcup_{\substack{1 \leq p_0 \leq N \\ 1 \leq q_0 < r_0 \leq N+1}} \bigcap_{j=1}^{M_{p_0}} \mathrm{div}((r_0 - q_0)a_{p_0j})$$

where $\mathrm{div}(c)$ is the set of divisors of $c \in \mathbb{N}$. The formula (1) is thus equivalent to:

$$\bigvee_{d_0 \in \Delta_0} \exists z_{1\ldots n} Q_1 x_1 \ldots Q_{m-1} x_{m-1} \quad \forall x_m \bigvee_{1\leq i \leq N}^{i \neq p_0} \bigwedge_{j=1}^{M_i} \tau_{ij} | h_{ij} + a_{ij} x_m \wedge \psi_i \vee$$
$$\bigwedge_{j=1}^{M_{p_0}} d_0 | h_{p_0j} + a_{p_0j} x_m \wedge \tau_{p_0j} = d_0 \wedge \psi_{p_0}$$

Now we are interested by the values of x_m not satisfying $\bigwedge_{j=1}^{M_i} d_0 | h_{p_0j} + a_{p_0j} x_m$, since those values must satisfy $\bigwedge_{j=1}^{M_i} \tau_{ij} | h_{ij} + a_{ij} x_m$. In fact, it is enough to find only one such value, call that x_m^0, in order to repeat the above construction of a $(N-1) \times N$ table, this time. Since $\bigwedge_{j=1}^{M_i} d_0 | h_{p_0j} + a_{p_0j} x_m$ is a system of linear congruences, its solutions form an arithmetic progression with period $\left[\{\frac{d_0}{(d_0, a_{p_0j})}\}_{j=1}^{M_i}\right]$ i.e., the least common multiple of all individual periods. Notice that $\left[\{\frac{d_0}{(d_0, a_{p_0j})}\}_{j=1}^{M_i}\right] = \frac{d_0}{(d_0, a_{p_01}, \ldots, a_{p_0M_i})}$. Call this number π_0. If x_m^0 is not a solution, neither is $x_m^0 + k\pi_0$, for any $k \geq 1$. Moreover, if such an x_m^0 exists, it can always be found, it suffices to test all values $1, 2, \ldots, \pi_0$. Otherwise, if x_m^0 doesn't exist, we can simply delete the p_0 disjunct from the original formula and proceed with the next step.

Now we repeat the construction of the table, taking the rows $1 \leq p \leq N$ with $p \neq p_0$ and the columns $x_m^0, x_m^0 + \pi_0, \ldots, x_m^0 + N\pi_0$. We obtain that:

$$\tau_{p_1j} \in \Delta_1 = \bigcup_{\substack{1 \leq p_1 \leq N \\ p_1 \neq p_0 \\ 1 \leq q_1 < r_1 \leq N}} \bigcap_{j=1}^{M_{p_1}} \mathrm{div}((r_1 - q_1)\pi_0 a_{p_1j})$$

Repeating the above step terminates in at most N iterations, the result being the following equivalent of (1):

$$\bigvee_{d_0 \in \Delta_0} \ldots \bigvee_{d_{N-1} \in \Delta_{N-1}} \exists z_{1\ldots n} Q_1 x_1 \ldots Q_{m-1} x_{m-1} \forall x_m \bigvee_{k=0}^{N-1} \bigwedge_{j=1}^{M_{p_k}} d_k | h_{p_kj} + a_{p_kj} x_m \wedge \tau_{p_kj} = d_k \wedge \psi_{p_k} \tag{4}$$

where, in general, for $1 \leq k \leq N$ we have:

$$\Delta_k = \bigcup_{\substack{1 \leq p_k \leq N \\ p_k \notin \{p_0, p_1, \ldots, p_{k-1}\} \\ 1 \leq q_k < r_k \leq N-k+1}} \bigcap_{j=1}^{M_{p_k}} \mathrm{div}((r_k - q_k)\pi_{k-1} a_{p_kj})$$

Let us now look closer at the resulting formula (4). There are two kinds of subformulas: linear congruences of the form $d|g(\vec{x})$, and formulas of the form $\big(\{f_i(\vec{z})\}_i\big) = d \wedge \psi(\vec{z})$. By performing adequate scoping operations, the formula (4) can be turned in a boolean combination (using only $\wedge$ and $\vee$) of formulas of either one of the two forms:

1. $Q_1 x_1 \ldots Q_m x_m \varphi(\vec{x})$, where φ is a boolean combination of linear congruences of the form $d|g(\vec{x})$, or
2. $\exists z_1 \ldots \exists z_n \psi(\vec{z})$, where ψ is a boolean combination of formulas of the form $\big(\{f_i(\vec{z})\}_i\big) = d \wedge \psi(\vec{z})$

In order to prove decidability, it is sufficient now to express the formulas $\big(\{f_i(\vec{z})\}_i\big) = d$ as formulas in the existential fragment of the arithmetic with addition and divisibility. This is accomplished as follows:

$$
\begin{aligned}
\big(f_1, f_2, \ldots, f_n\big) = d \quad &\iff \quad \exists y_1 \ldots \exists y_n \bigwedge_{i=1}^{n} f_i = dy_i \wedge (y_1, \ldots, y_n) = 1 \\
&\iff \quad \exists y_{1\ldots n} \exists y'_{1\ldots n} \bigwedge_{i=1}^{n} f_i = dy_i \wedge y_i | y'_i \ \wedge \ \Sigma_{i=1}^{n} y'_i = 1
\end{aligned}
$$

The last step is an application of Bézout's Lemma.

Complexity Assessment

The formulas of the first kind are formulas of Presburger arithmetic, hence they can be evaluated in time $2^{2^{O(\nu)}}$, where ν is the size of the formula [FR74]. The formulas of the second kind are formulas in the existential fragment of the arithmetic with addition and divisibility, which is decidable in time $2^{O\left(\mu^{\mu^3}\right)}$, where μ is the maximum between the size of the formula and the maximum absolute value of the coefficients a_{ij} [Lip76]. With the notation from (4), the total number of Presburger formulas of the first kind is bounded by $m^2 2^{\Pi_{k=0}^{N-1} \|\Delta_k\|}$, and the number of formulas of the second kind is bounded by $n^2 2^{\Pi_{k=0}^{N-1} \|\Delta_k\|}$. Moreover, $\|\Delta_k\| \leq N^2\mu^2$, where μ is the maximum between N and absolute values of the coefficients a_{ij}. Hence, the evaluation time for a formula in the form (4) is bounded by $(n^2 + m^2) \cdot 2^{(N^2\mu^2)^N} \cdot 2^{2^{(N^2\mu^2)^N}}$. If $Q_m \equiv \forall$, the transformation of a formula of the form (1) to the form (4) takes at most N steps, each step taking at most $N^4\mu^2$ time. In conclusion, the presence of a single universal quantifier in a positive formula is enough to decide satisfiability, the entire decision procedure being bounded by a triple exponential. Although this complexity is bad in theory, it represents an improvement w.r.t. the general case, with negation, from [BI04], where the worst-case complexity was a tour of exponentials, of height depending on the alternation depth of the quantifiers $Q_1, \ldots Q_m$.

5 Semi-algorithmic Quantitative Verification

Considering the undecidability result from Section 2, we can still ask the question whether a complete and effective semi-algorithmic procedure for verifying quantitative properties of programs with lists does exist. We shall study this problem in the framework of Floyd-Hoare deductive program verification, based on the principle of inferring triples of the form $\{P\}$ **C** $\{Q\}$, where P and Q are the pre- and post-conditions and C is a program to be verified. The main disadvantage of this method is that, in practice, much human assistance is needed guide the proof.

Our attempt to overcome this problem relies on a sound and complete weakest precondition calculus combined with a relatively complete semi-algorithmic invariant generation procedure. The main idea is to annotate properties written in alias logic with counters locally within the program code (only safety properties are considered). The semantics is that, whenever the control reaches an annotated statement, the annotation must hold. To verify the properties, we completely annotate the program with their weakest preconditions and with loop invariants. Then, the properties are checked by checking the validity of a finite number of implications. Since alias logic with counters is decidable, we ask the question whether it is *expressive* enough to (a) encode weakest preconditions for all statements considered in Figure 2 and (b) encode all strongest loop invariants. As we will show, the answer to the second point is negative, therefore we can only hope to find invariants that are expressible in the logic. This amounts to having a *relatively complete* inference system for Hoare triples, which can still be sufficient for verifying many meaningful properties.

5.1 Weakest Precondition Calculus

For the purposes of this section we shall work with AL_{AD}, instead of AL_{PA}, as was the case in Section 3 for the specification of properties. We recall that in AL_{AD}, counter variables are allowed to occur in formulas of $\exists\mathcal{L}_{|}^{(*)+}$, instead of Presburger Arithmetic, as was the case in AL_{PA}. For example, we allow now formulas such as $\exists x \forall y\ u \overset{x}{\leadsto} \wedge v \overset{y+2}{\leadsto} \wedge x|y$ in our logic. This extension does not affect the decidability result of [BI05], as there we have also used reduction of alias propositions to $\exists\mathcal{L}_{|}^{(*)+}$.

Table 1 gives the weakest liberal preconditions $\text{wlp}(S, \varphi)$ for a statement S and an alias proposition $\varphi = u \overset{f}{\leadsto} \diamond v \overset{g}{\leadsto}$, where $f(\vec{x}), g(\vec{x})$ are linear functions over $\vec{x}$. The weakest precondition of an arbitrary formula in alias logic can be computed from the rules in Table 1, recursively on the structure of the formula:

$$\begin{aligned}
\text{wlp}(S, \varphi \wedge \psi) &= \text{wlp}(S, \varphi) \wedge \text{wlp}(S, \psi) \\
\text{wlp}(S, \neg\psi) &= \neg\text{wlp}(S, \varphi) \\
\text{wlp}(S, \forall x\ \varphi) &= \forall x\ \text{wlp}(S, \varphi) \\
\text{wlp}(S, \varphi) &= \varphi, \ \text{ if } \varphi \text{ does not contain alias propositions}
\end{aligned}$$

The rules for conjunction and universal quantification are sound and complete in general. The rule for negation is sound in general and complete only for statements with deterministic outcome, which is the case for us.

The assignment $w \leadsto = t$ needs some explanation. There are five main cases, shown in Figure 5. The five rows of the middle section in Table 1 are the preconditions for the five cases in Figure 5. The weakest precondition is the disjunction of the five different preconditions. The first two cases (a) and (b), require the use of the divisibility operator to relate the initial and final lengths of paths in the precondition. As an example, one can easily verify the validity of the following triple: $\{u \overset{1}{\leadsto} \Diamond v \overset{1}{\leadsto} \wedge\ u \overset{1}{\leadsto} \Diamond t \overset{1}{\leadsto} \wedge\ v \overset{2}{\leadsto} \Diamond w\}$ $\mathbf{w} \leadsto= \mathbf{t}$ $\{u \overset{4}{\leadsto} \Diamond v \overset{7}{\leadsto}\}$. The assignment $w \leadsto= t$ closes a loop of period 3, so the relation satisfied by the counters is $4 - 1 \equiv_3 7 - 1$.

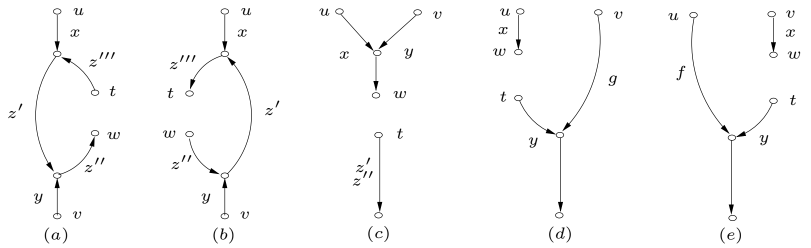


Figure 5:

5.2 Automatic Invariant Generation

The problem of finding suitable loop invariants is the pitfall of program verification. Indeed, there is nothing apparently that can relate the program and the property to prove with the invariant that is needed for the proof to go through. On the other hand, if we are given a candidate formula ψ, it is possible to decide whether this is an invariant for a program loop $B \equiv S_1; S_2; \ldots S_n;$. Just compute $\mathrm{wlp}(B, \psi) = \mathrm{wlp}(S_1, \mathrm{wlp}(S_2, \ldots \mathrm{wlp}(S_n, \psi) \ldots))$ and verify whether $\mathrm{wlp}(B, \psi) \rightarrow \psi$ is a valid alias logic formula. So the main question is how to generate recursively *all* candidate invariants in a way that guarantees termination.

Here the notion of parametric list graph defined in Section 4.1 comes in handy. The main idea is that any invariant must describe a set of states (memory configurations) whose shapes are described by a finite union of parametric list graphs. This is a consequence of Lemma 2, which states that the number of parametric list graphs in normal form is finite. To be more precise, let V be the set of program variables, and let $\{G_k\}_{k \in \mathcal{K}}$ be an enumeration of all possible parametric list graphs over V in normal form. According to Lemma 2, the set $\mathcal{K}$ is finite. Since each edge of a parametric list graph is labeled with a different variable z, we consider the set $\vec{z}$ of all variables needed to label any such graph. By Lemma 2, $|\vec{z}| \leq 2\|V\|$, as each node has at most one successor. Let $\vec{z'}$ and $\vec{z''}$ be the sets $\{z' \;:\; z \in \vec{z}\}$ and $\{z'' \;:\; z \in \vec{z}\}$. For a parametric list graph $G(\vec{z})$, let $\mathcal{G}(\vec{z})$ be an alias logic formula that characterizes G up to isomorphism. That is, for every interpretation $\iota : \vec{z} \rightarrow \mathbb{N}$ we have $G(\iota(\vec{z})) \models \mathcal{G}(\iota(\vec{z}))$, and for no other $G' \neq G$ is this the case. With this notation, the following lemma gives a normal form of alias logic formulas, that is suitable for invariant generation.

Lemma 3 *Any formula $\varphi(\vec{x})$ of AL_{AD} is equivalent to an effectively computable disjunction of the form $\bigvee_{k \in \mathcal{K}} \exists \vec{y}\ \mathcal{G}_k(\vec{y}) \wedge \psi_k(\vec{x}, \vec{y})$, where, for any $k \in \mathcal{K}$, ψ_k is a formula of $\langle \mathbb{N}, 0, S, +, | \rangle$.*

Proof: We denote by g a non-parametric list graph, and by Ψ_g its characteristic alias logic formula. For any interpretation $\vec{v} = \iota(\vec{x})$, we have:

$$\varphi(\vec{v}) \longleftrightarrow \bigvee \{\Psi_g \;:\; g \models \varphi(\vec{v})\}$$

$\{\mathrm{wlp}(S,\varphi)\}$ **S** $\{\varphi\}$	**side conditions**

the cases for $\mathbf{w} = \mathbf{new}$

$\{u\overset{f}{\rightsquigarrow}\Diamond v\overset{g}{\rightsquigarrow}\}$ $\mathbf{w{=}new}$ $\{u\overset{f}{\rightsquigarrow}\Diamond v\overset{g}{\rightsquigarrow}\}$	u, v and w are distinct variables
$\{f{=}1\wedge g{=}1\}$ $\mathbf{u{=}new}$ $\{u\overset{f}{\rightsquigarrow}\Diamond u\overset{g}{\rightsquigarrow}\}$	—

the cases for $\mathbf{w} \rightsquigarrow= \mathbf{t}$

$\left\{\begin{array}{c} \exists x\exists y\exists z'\exists z''\exists z'''\ \xi(x,y,z',z'',z''') \wedge \\ \forall\bar{x}\forall\bar{y}\forall\bar{z'}\forall\bar{z''}\forall\bar{z'''}\ \xi(\bar{x},\bar{y},\bar{z'},\bar{z''},\bar{z'''})\rightarrow \\ x\le\bar{x}\wedge y\le\bar{y}\wedge z'\le\bar{z'}\wedge z''\le\bar{z''}\wedge z'''\le\bar{z'''}\wedge \\ x\le f\wedge y\le g\wedge z'+z''+z'''+1 \mid \lvert f-g-x+y-z'\rvert \end{array}\right\}$ $\mathbf{w{\rightsquigarrow}{=}t}$ $\{u\overset{f}{\rightsquigarrow}\Diamond v\overset{g}{\rightsquigarrow}\}$	(a) $\xi(x,y,z',z'',z''')=u\overset{x}{\rightsquigarrow}\Diamond t\overset{z'''}{\rightsquigarrow}\wedge\ u\overset{x+z'}{\rightsquigarrow}\Diamond v\overset{y}{\rightsquigarrow}\wedge\ v\overset{y+z''}{\rightsquigarrow}\Diamond w$ (b) $\xi(x,y,z',z'',z''')=u\overset{x+z'''}{\rightsquigarrow}\Diamond t\ \wedge\ u\overset{x}{\rightsquigarrow}\Diamond v\overset{y+z'}{\rightsquigarrow}\wedge\ w\overset{z''}{\rightsquigarrow}\Diamond v\overset{y}{\rightsquigarrow}$
$\left\{\begin{array}{c} \exists x\exists y\exists z'\exists z''\ \xi(x,y,z',z'') \wedge \forall\bar{x}\forall\bar{y}\forall\bar{z'}\forall\bar{z''}\ \xi(\bar{x},\bar{y},\bar{z'},\bar{z''})\rightarrow \\ x\le\bar{x}\wedge y\le\bar{y}\wedge z'\le\bar{z'}\wedge z''\le\bar{z''}\wedge \\ x+z'+1=f\wedge y+z''+1=g \end{array}\right\}$ $\mathbf{w{\rightsquigarrow}{=}t}$ $\{u\overset{f}{\rightsquigarrow}\Diamond v\overset{g}{\rightsquigarrow}\}$	(c) $\xi(x,y,z',z'')=u\overset{x}{\rightsquigarrow}\Diamond w\wedge v\overset{y}{\rightsquigarrow}\Diamond w\wedge t\overset{z'}{\rightsquigarrow}\Diamond t\overset{z''}{\rightsquigarrow}$
$\left\{\ \exists x\exists y\ \xi(x,y)\ \wedge\ \forall\bar{x}\forall\bar{y}\ \xi(\bar{x},\bar{y})\rightarrow x\le\bar{x}\wedge y\le\bar{y}\ \right\}$ $\mathbf{w{\rightsquigarrow}{=}t}$ $\{u\overset{f}{\rightsquigarrow}\Diamond v\overset{g}{\rightsquigarrow}\}$	(d) $\xi(x,y)=u\overset{x}{\rightsquigarrow}\Diamond w\wedge\ t\overset{y}{\rightsquigarrow}\Diamond v\overset{g}{\rightsquigarrow}\wedge\ x+y+1=f$ (e) $\xi(x,y)=u\overset{f}{\rightsquigarrow}\Diamond t\overset{y}{\rightsquigarrow}\wedge\ v\overset{x}{\rightsquigarrow}\Diamond w\wedge\ x+y+1=g$

the cases for $\mathbf{w} = \mathbf{t} \overset{\mathbf{k}}{\rightsquigarrow}$

$\{u\overset{f}{\rightsquigarrow}\Diamond v\overset{g}{\rightsquigarrow}\}$ $\mathbf{w{=}t}\overset{\mathbf{k}}{\rightsquigarrow}$ $\{u\overset{f}{\rightsquigarrow}\Diamond v\overset{g}{\rightsquigarrow}\}$	u, v and w are distinct variables
$\{t\overset{k+f}{\rightsquigarrow}\Diamond v\overset{g}{\rightsquigarrow}\}$ $\mathbf{u{=}t}\overset{\mathbf{k}}{\rightsquigarrow}$ $\{u\overset{f}{\rightsquigarrow}\Diamond v\overset{g}{\rightsquigarrow}\}$	—
$\{u\overset{f}{\rightsquigarrow}\Diamond t\overset{k+g}{\rightsquigarrow}\}$ $\mathbf{v{=}t}\overset{\mathbf{k}}{\rightsquigarrow}$ $\{u\overset{f}{\rightsquigarrow}\Diamond v\overset{g}{\rightsquigarrow}\}$	—

Table 1: Weakest Liberal Preconditions for Alias Propositions

The above formula is infinite, since the number of non-parametric graphs g is infinite. However, each g is an instance of a parametric graph G i.e., $g = G(\iota(\vec{y}))$, for a suitable interpretation ι of $\vec{y}$. Moreover, the condition $G(\vec{y}) \models \varphi(\vec{v})$ is equivalent to an arithmetic formula $\Psi_{G,\varphi}(\vec{y}, \vec{v})$. Since the choice of $\vec{v}$ was arbitrary, we have:

$$\varphi(\vec{x}) \longleftrightarrow \bigvee_{k\in\mathcal{K}} \exists\vec{y}\ \mathcal{G}_k(\vec{y}) \wedge \Psi_{G_k,\varphi}(\vec{x}, \vec{y})$$

□

In the light of this lemma, we can express the weakest precondition of a parametric graph characteristic formula w.r.t. a program loop B as:

$$\mathrm{wlp}\big(B, \mathcal{G}_j(\vec{z})\big) = \bigvee_{i\in\mathcal{K}} \exists\vec{z'}\ \mathcal{G}_i(\vec{z'}) \wedge \psi_{ij}(\vec{z}, \vec{z'}) \qquad (5)$$

where some ψ_{ij} are taken to be false. Next we arrange all such ψ_{ij} formulas in a $\mathcal{K} \times \mathcal{K}$ matrix, call this matrix $M_B = [\psi_{ij}]_{(i,j)\in\mathcal{K}\times\mathcal{K}}$. We can interpret M_B as the incidence matrix of a finite state automaton, in which the states are formulas $\mathcal{G}_i(\vec{z})$ and a transition between $\mathcal{G}_i$ and $\mathcal{G}_j$ is labeled with $\psi_{ij}(\vec{z}, \vec{z'})$. Intuitively, $\psi_{ij}(\vec{z}, \vec{z'})$ represents the necessary and sufficient relation between the current and next values of $\vec{z}$, when the program goes from a state characterized by the shape $\mathcal{G}_i$ to a state characterized by the shape $\mathcal{G}_j$.

To compute the necessary invariants, we shall need to effectively compute the transitive closure of this matrix. We can do so by relying on classical automata theory. Let $\mathcal{A} = (A^\infty, \vee, \circ, \star, \top, \bot)$ be a Kleene algebra, where A^∞ is the language of all finite and infinite formulas of arithmetic, and $\circ$ is the composition of relations i.e. $(\varphi \circ \psi)(\vec{x}, \vec{y}) = \exists \vec{z}\, \varphi(\vec{x}, \vec{z}) \wedge \psi(\vec{z}, \vec{y})$. The $\star$ symbol denotes the infinite disjunction $\varphi^\star = \bigvee_{k \leq 0} \varphi^k$, where $\varphi^0 = \varphi$ and $\varphi^k = \varphi^{k-1} \circ \varphi$, for $k \geq 1$. In general, $\varphi^\star$ cannot be expressed by an equivalent finite formula, which is why we have to work with the set A^∞ of finite and infinite formulas, for now. In fact, the results in the rest of this section are given under the proviso that the star-closure of $\mathcal{A}$ is effectively representable and computable[2].

The algebra $\mathcal{A}$ induces an algebra $[\mathcal{A}]$ on matrices, with the $\vee$ and $\circ$ operations defined as usual. The star-closure of the matrix M_B can be computed recursively, using the classical formula [Con71]:

$$\left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]^\star = \left[\begin{array}{c|c} (A \vee BD^\star C)^\star & (A \vee BD^\star C)^\star BD^\star \\ \hline (D \vee CA^\star B)^\star CA^\star & (D \vee CA^\star B)^\star \end{array}\right]$$

The (i, j)-th element of $M_B^\star$ represents the weakest constraint on the variables $\vec{z}$ and $\vec{z'}$ for the loop B to finish in a state satisfying $\mathcal{G}_j(\vec{z'})$, when started in $\mathcal{G}_i(\vec{z})$, after an arbitrary number of iterations i.e.,

$$\text{wlp}^\star\big(B, \mathcal{G}_j(\vec{z})\big) = \bigvee_{i \in \mathcal{K}} \exists \vec{z'}\, \mathcal{G}_i(\vec{z'}) \wedge M_B^\star[i][j](\vec{z}, \vec{z'}) \tag{6}$$

Given the incidence matrix M_B of a loop B, we can derive now a finite number of loop invariants that are incomparable w.r.t. logical implication. We shall argue in the following that these invariants are all we need to prove a quantitative program property. The next lemma gives a sufficient condition for finding invariants of B:

Lemma 4 *Let J be a subset of $\mathcal{K}$ such that, for each $j \in J$ it is the case that wlp$\big(B, \mathcal{G}_j(\vec{z})\big) = \bigvee_{i \in J} \exists \vec{z'}\, \mathcal{G}_i(\vec{z'}) \wedge \psi_{ij}(\vec{z}, \vec{z'})$. Then $\mathcal{I}_J(\vec{z}) = \bigvee_{i,j \in J} \exists \vec{z'}\, \mathcal{G}_i(\vec{z'}) \wedge M_B^\star[i][j](\vec{z}, \vec{z'})$ is a loop invariant for B.*

[2] Both of which are hardly the case in reality, but we are still in an ideal world.

Proof: We calculate:

$$\begin{aligned}
\mathrm{wlp}\big(\mathcal{I}_J(\vec{z})\big) &= \mathrm{wlp}\Big(\bigvee_{i,j\in J} \exists\vec{z'}\ \mathcal{G}_i(\vec{z'}) \wedge M_B^\star[i][j](\vec{z},\vec{z'})\Big) \\
&= \bigvee_{i,j\in J} \exists\vec{z'}\ \mathrm{wlp}\big(\mathcal{G}_i(\vec{z'})\big) \wedge M_B^\star[i][j](\vec{z},\vec{z'}) \\
&= \bigvee_{i,j\in J} \exists\vec{z'} \bigvee_{k\in J} \exists\vec{z''}\mathcal{G}_k(\vec{z''}) \wedge \psi_{ki}(\vec{z'},\vec{z''}) \wedge M_B^\star[i][j](\vec{z},\vec{z'}) \\
&= \bigvee_{j,k\in J} \exists\vec{z''}\ \mathcal{G}_k(\vec{z''}) \wedge \bigvee_{i\in J} \exists\vec{z'}\psi_{ki}(\vec{z'},\vec{z''}) \wedge M_B^\star[i][j](\vec{z},\vec{z'}) \\
&= \bigvee_{j,k\in J} \exists\vec{z''}\ \mathcal{G}_k(\vec{z''}) \wedge \bigvee_{i\in J} \big(\psi_{ki}\circ M_B^\star[i][j]\big)(\vec{z},\vec{z''})
\end{aligned}$$

Notice that $\bigvee_{i\in J}\psi_{ki}\circ M_B^\star[i][j] \leftrightarrow M_B^\star[k][j]$, and therefore $\mathrm{wlp}\big(\mathcal{I}_J(\vec{z})\big) \leftrightarrow \mathcal{I}_J(\vec{z})$. $\square$

We are looking after the minimal subsets J of $\mathcal{K}$ that satisfy the condition from Lemma 4. Finding all these subsets is possible by exhaustive enumeration, since $\mathcal{K}$ is finite. However, one can imagine more effective ways of doing so e.g., by iteratively computing the least fixed points of the function $\{j \mapsto J\ :\ \mathrm{wlp}\big(B,\mathcal{G}_j(\vec{z})\big) = \bigvee_{i\in J}\exists\vec{z'}\ \mathcal{G}_i(\vec{z'}) \wedge \psi_{ij}(\vec{z},\vec{z'})\}$.

The next point concerns the completeness of our method. In fact, we can only talk about relative completeness i.e., completeness w.r.t. the logic of choice, in our case alias logic with counters. That is, our method cannot verify properties that require loop invariants which cannot be expressed in alias logic. However, all properties that can be proved within this logical framework can also be proved by our technique, under the aforementioned proviso.

Theorem 2 *For every invariant $I(\vec{z})$ of a loop B, that is expressible in AL_{AD}, there exists a subset J of $\mathcal{K}$ such that $\mathcal{I}_J$ implies I, where $\mathcal{I}_J(\vec{z}) = \bigvee_{i,j\in J}\exists\vec{z'}\ \mathcal{G}_i(\vec{z'}) \wedge M_B^\star[i][j](\vec{z},\vec{z'})$.*

Proof: By Lemma 3, $I(\vec{z}) = \bigvee_{k\in\mathcal{K}}\exists\vec{z'}\ \mathcal{G}_k(\vec{z'}) \wedge \varphi_k(\vec{z},\vec{z'})$, for some (possibly false) arithmetic formulas φ_k. We calculate:

$$\begin{aligned}
\mathrm{wlp}\big(I(\vec{z})\big) &= \bigvee_{k\in\mathcal{K}} \exists\vec{z'}\ \mathrm{wlp}\big(\mathcal{G}_k(\vec{z'})\big) \wedge \varphi_k(\vec{z},\vec{z'}) \\
&= \bigvee_{k\in\mathcal{K}} \exists\vec{z'} \bigvee_{j\in\mathcal{K}} \exists\vec{z''}\ \mathcal{G}_j(\vec{z''}) \wedge \psi_{kj}(\vec{z'},\vec{z''}) \wedge \varphi_k(\vec{z},\vec{z'}) \\
&= \exists\vec{z''} \bigvee_{j\in\mathcal{K}} \mathcal{G}_j(\vec{z''}) \wedge \bigvee_{k\in\mathcal{K}} (\psi_{kj}\circ\varphi_k)(\vec{z},\vec{z''})
\end{aligned}$$

Now we require that $\forall\vec{z}\ \mathrm{wlp}\big(I(\vec{z})\big) \rightarrow I(\vec{z})$ be a valid alias logic formula. Let g be an arbitrary non-parametric list graph. Notice that there exists exactly one $\gamma\in\mathcal{K}$ such that $g \models \exists\vec{z}\ \mathcal{G}_\gamma(\vec{z})$. Suppose now that $g \models wlp\big(I(\vec{z})\big)$. Then $g \models \exists\vec{z''}\ \mathcal{G}_\gamma(\vec{z''}) \wedge \bigvee_{j\in\mathcal{K}}(\psi_{j\gamma}\circ\varphi_j)(\vec{z},\vec{z''})$ must be the case. But also $g \models I(\vec{z})$, hence $g \models \exists\vec{z'}\ \mathcal{G}_\gamma(\vec{z'}) \wedge \varphi_\gamma(\vec{z},\vec{z'})$. Also, the choice of values for $\vec{z'}$ and $\vec{z''}$ must be the same, since they represent the lengths of the same paths from g. Since the choice of g was arbitrary, also this choice is arbitrary, which implies the validity of the following implication: $\forall \vec{z} \forall \vec{z'} \ \bigvee_{j \in \mathcal{K}} (\psi_{j\gamma} \circ \varphi_j)(\vec{z}, \vec{z''}) \ \rightarrow \ \varphi_\gamma(\vec{z}, \vec{z'})$. But this holds for all $\gamma \in \mathcal{K}$, so the $\{\varphi_i\}_{i \in \mathcal{K}}$ are solutions to the following system of constraints:

$$\left\{ \bigvee_{j \in \mathcal{K}} (\psi_{jk} \circ \varphi_j) \rightarrow \varphi_k \right\}_{k \in \mathcal{K}}$$

However the least solution to this system is given by $\phi_i = \bigvee_{k \in \mathcal{K}} M_B^\star[k][i]$, which leads to $\phi_i \rightarrow \varphi_i$, for all $i \in \mathcal{K}$. The conclusion follows now easily. □

Basically, we have reduced the problem of generating loop invariants for the verification of quantitative properties of programs with lists to the problem of representing and computing star-closures of formulas in the arithmetic of natural numbers with addition and divisibility. The latter problem has been addressed by many different groups and several approaches exist today. The two main directions are the approximative techniques, based on widening [CH78], and the precise techniques, based on acceleration [CJ98].

6 Conclusions and Future Work

We have studied the problem of quantitative verification for imperative programs that handle dynamic list data structures. We show undecidability of this problem and propose an effective semi-algorithmic approach to it. First, we define a decidable logic for property specification. Then we develop a weakest precondition calculus on top of this logic and, finally an invariant generation method. Since invariants can only be expressed using second order arithmetic (star-closure of first-order arithmetic formulas), we propose using already existing widening and acceleration for integer programs. Future work consist in studying the acceleration problem further, and identifying classes of list programs for which a precise acceleration algorithm always terminates.

References

[BCO04] J. Berdine, Cristiano Calcagno, and P. O'Hearn. A decidable fragment of separation logic. In *Proc. Foundations of Software Technology and Theoretical Computer Science*, pages 97 – 109, 2004. 1

[BFN04] S. Bardin, A. Finkel, and D. Nowak. Toward symbolic verification of programs handling pointers. In *Proc. 3rd Int. Workshop on Automated Verification of Infinite-State Systems (AVIS 2004), Barcelona, Spain*. Electronic Notes in Theoretical Computer Science, 2004. 1, 4.1

[BI04] Marius Bozga and Radu Iosif. On decidability with the arithmetic with addition and divisibility. Technical Report TR-2004-18, Verimag, 2004. 4, 4.2, 4.2, 4.2

[BI05] M. Bozga and R. Iosif. Quantitative verification of programs with lists. Technical Report TR-2005-2, Verimag Tech Report, 2005. 4, 4.1, 5.1

[BIL04] Marius Bozga, Radu Iosif, and Yassine Lakhnech. Counting aliases. Technical Report TR-2004-17, Verimag, 2004. 3

[BOB] Richard Bornat, Peter O'Hearn, and Josh Berdine. http://www.cs.mdx.ac.uk/staffpages/r_bornat/movies/circular_list_dance.mpg. 3

[BPZ05] I. Balaban, A. Pnueli, and L. Zuck. Shape analysis by predicate abstraction. In *Proc. 6th Intl. Conference on Verification, Model Checking and Abstract Interpretation*, pages 165 – 180, 2005. 1

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proc. 5th ACM Symposium on Principles of Programming Languages*, 1978. 1, 5.2

[CJ98] Hubert Comon and Yan Jurski. Multiple Counters Automata, Safety Analysis and Presburger Arithmetic. In *Proceedings of the 10th International Conference on Computer Aided Verification*, volume 1427, pages 268 – 279. Lecture Notes in Computer Science, 1998. 1, 5.2

[Con71] J.H. Conway. *Regular algebra and finite machines*. Chapman and Hall, 1971. 5.2

[End72] H. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972. 3

[Flo67] R. W. Floyd. Assigning meanings to programs. *Mathematical Aspects of Computer Science*, 19:19–32, 1967. 1

[FR74] M. J. Fischer and M. O. Rabin. Super-exponential complexity of Presburger arithmetic. *Complexity of Computation (AMS-SIAM Proceedings)*, 7(41), 1974. 4.2

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:576–585, October 1969. 1

[Lan92] William Landi. Undecidability of static analysis. *ACM Letters on Programming Languages and Systems (LOPLAS)*, 1(4):323 – 337, December 1992. 2.1

[Lip76] Leonard Lipshitz. The diophantine problem for addition and divisibility. *Transaction of the American Mathematical Society*, 235:271 – 283, January 1976. 4.2, 4.2

[Min67] M. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967. 2.1

[MYRS05] R. Manevich, E. Yahav, G. Ramalingam, and M. Sagiv. Predicate abstraction and canonical abstraction for singly-linked lists. In *Proc. 6th Intl. Conference on Verification, Model Checking and Abstract Interpretation*, pages 181 – 198, 2005. 1

[Pre29] Mojzesz Presburger. Über die Vollstandigkeit eines gewissen Systems der Arithmetik. *Comptes rendus du I Congrés des Pays Slaves*, Warsaw 1929. 3

[Ram94] G. Ramalingam. The undecidability of aliasing. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1467 – 1471, 1994. 2.1

[Rob49] Julia Robinson. Definability and decision problems in arithmetic. *The Journal of Symbolic Logic*, 14(2):98 – 114, June 1949. 3, 4

[Rug04] R. Rugina. Quantitative shape analysis. In *Proc. 11th Intl. Static Analysis Symposium (SAS04)*, pages 228 – 245, 2004. 1

[ZSM04] T. Zhang, H. Sipma, and Z. Manna. Decision procedures for recursive data structures with integer constraints. In *Proc. 2nd International Joint Conference on Automated Reasoning (IJCAR'04)*, pages 152 – 167, 2004. 1