Quantitative Separation Logic and Programs with Lists

Marius Bozga, Radu Iosif, Swann Perarnau

Quantitative Separation Logic and Programs with Lists

Marius Bozga, Radu Iosif, and Swann Perarnau

VERIMAG, 2 Avenue de Vignate, F-38610 Gières
{iosif,bozga,perarnau}@imag.fr

Abstract. This paper presents an extension of a decidable fragment of Separation Logic for singly-linked lists, defined by Berdine, Calcagno and O'Hearn [8]. Our main extension consists in introducing atomic formulae of the form $ls^k(x,y)$ describing a list segment of length k, stretching from x to y, where k is a logical variable interpreted over positive natural numbers, that may occur further inside Presburger constraints.
We study the decidability of the full first-order logic combining unrestricted quantification of arithmetic and location variables. Although the full logic is found to be undecidable, validity of entailments between formulae with the quantifier prefix in the language $\exists^*\{\exists_{\mathbb{N}},\forall_{\mathbb{N}}\}^*$ is decidable. We provide here a model theoretic method, based on a parametric notion of shape graphs.
We have implemented our decision technique, providing a fully automated framework for the verification of quantitative properties expressed as pre- and post-conditions on programs working on lists and integer counters.

1 Introduction

Separation Logic [15, 21] has recently become a widespread formalism for the specification of programs with dynamic data structures. Due to the intrinsic complexity of the heap structures allocated and manipulated by such programs, any attempt to formalize their correctness has to be aware of the inherent bounds of undecidability. Indeed, even programs working on simple acyclic lists have the power of Turing machines, and it is expected that a general logic describing sets of configurations reached in such programs has an undecidable satisfiability (or validity) problem. An interesting problem is to define decidable logics that are either specialized for a certain kind of recursive data structures (e.g. lists, trees), or that are restricted by the quantifier prefix.

This paper presents an extension of a decidable fragment of Separation Logic for singly-linked lists, defined by Berdine, Calcagno and O'Hearn [8] and used as an internal representation for sets of states in the Smallfoot tool [4]. Our main extension consists in introducing atomic formulae of the form $ls^k(x,y)$ describing a list segment of length k, stretching from x to y, where k is a logical variable interpreted over positive natural numbers, that may occur further inside Presburger constraints. This is motivated by the need to reason about programs that work on both singly-linked list structures and integer variables (counters). We denote the extended logic as Quantitative Separation Logic (**QSL**).

In reality, many programs would traverse a list structure, while performing some iterative computation on the integer variables. The result of this computation usually

depends on the number of steps, which, in turn, depends of the length of the list. A specification of the correct behavior for such a program needs to take into account both the lengths of the lists and the values of the counters.

We study the decidability properties of the full first-order logic combining unrestricted quantification of arithmetic and location variables. Although the full logic is found to be undecidable, validity of entailments between formulae with the quantifier prefix in the language $\exists^*\{\exists_{\mathbb{N}}, \forall_{\mathbb{N}}\}^*$ is decidable. We provide here a model theoretic method for decidability, based on a parametric notion of shape graphs. As a byproduct, we obtain a decision procedure for the fragment of Separation Logic considered in [8].

The decision procedure for a fragment of **QSL** is currently implemented in the L2CA tool [3], a tool for translating programs with singly-linked lists into bisimilar counter automata, according to the method of [9], which opens the possibility of using well-known counter automata techniques and tools, e.g. [6, 22, 5], in order to verify pre- and post- conditions expressed in **QSL**, on programs working on both singly-linked lists and integer variables.

1.1 Related Work

The saga of logics for describing heap structures has its roots in the early work of Burstall [11]. Later on, work by Benedikt, Reps and Sagiv [7], Reynolds [21] and Ishtiaq and O'Hearn [15], has brought the subject into focus, whereas recent advances have been made in tackling the decidability problem [14, 8, 23]. The idea of combining shape and arithmetic specifications arises in the work of Zhang, Sipma and Manna [24], where a combination of free term algebras with Presburger constraints is studied for decidability. The work that is closest to ours is the one of Berdine, Calcagno and O'Hearn [8], which defines a decidable subset of Separation Logic [21] interpreted over singly-linked heap models. The work in this paper is in fact an extension of the logic in [8] with integer variables representing list lengths. One of the main challenges in the present paper was to adapt the model of parametric shape graphs in order to cope with the notion of disjunctive heaps, which is the essence of the semantic model for Separation Logic.

Regarding program analysis, the use of abstract domains (including integers and memory addresses) with quantifiers of the form $\exists^*\forall^*$ has been considered in the work of Gulwani et al. [13, 12]. Unlike our approach, their work is based on using abstractions that prove to be sufficient, in general, for checking correctness of a large body of programs. Some of our examples, such as InsertSort, are also verified using the method of [12]. Recently, Magill et al. [17] report on a program analysis technique that uses Separation Logic [21] extended with first-order arithmetic. However, the main emphasis of [17] is a program analysis based on counterexample-driven abstraction refinement, whereas our work focuses on distinguishing decidable from undecidable when combining Separation Logic with first-order arithmetic. As a matter of fact, [17] claims that validity of entailments in the purely existential fragment of Separation Logic with the $ls^k(x,y)$ predicate and linear constraints is decidable, without giving the proof, by analogy to the proof-theoretic method from [8]. We extend their result by showing decidability of the validity of entailments in the $\exists^*\{\exists_{\mathbb{N}}, \forall_{\mathbb{N}}\}^*$ fragment, versus undecid-

ability of satisfiability in the $\exists^*\exists^*_{\mathbb{N}}(\forall \mid \forall_{\mathbb{N}})\exists^*\exists^*_{\mathbb{N}}$ fragment (or equivalently, validity in the $\forall^*\forall^*_{\mathbb{N}}(\exists \mid \exists_{\mathbb{N}})\forall^*\forall^*_{\mathbb{N}}$ fragment).

Roadmap The paper is organized as follows. Section 2 defines the syntax and semantics of the logic **QSL**. Section 3 proves the undecidability of the logic, while Section 4 proves the decidability of entailments in the $\exists^*\{\exists_{\mathbb{N}}, \forall_{\mathbb{N}}\}^*$ fragment. Section 5 gives some examples of programs verified using **QSL**, and Section 6 concludes. For space reasons, all proofs are given in [10].

2 Definitions

In the rest of the paper, for a set A we denote by $A_\perp$ the set $A \cup \{\perp\}$. For a function $f : A \to B$, we denote by $dom(f) = \{x \in A \mid f(x) \neq \perp\}$ its domain and by $img(f) = \{y \in B \mid \exists x \in A \,.\, f(x) = y\}$ we denote its image. The element $\perp$ is used to denote that a (partial) function is undefined at a given point, e.g. $f(x) = \perp$. Sometimes we shall use the graph notation for functions, i.e. $f = \{\langle a, b\rangle, \ldots\}$ if $f(a) = b, \ldots$, etc. The notation $\lambda x : A.y$ stands for the function $\{\langle x, y\rangle \mid x \in A\}$, and $\lambda x : A.\perp$ is the empty function $\emptyset$, by convention. Let $Part(S)$ denote the set of all partitions of the set S.

By $\mathcal{T}(X)$ we denote the set of all terms build using variables $x \in X$. For a term (formula) $\tau(X)$ and a mapping $\mu : X \to \mathcal{T}(X)$, we denote by $\tau[\mu]$ the term (formula) in which each occurrence of x is replaced with $\mu(x)$. For a formula φ, we denote as $FV(\varphi)$ the set of its free variables. If φ is a formula of the first-order arithmetic of integers, and $\nu : FV(\varphi) \to \mathbb{Z}$ is an interpretation of its free variables, we denote by $\nu \models \varphi$ the fact that $\varphi[\nu]$ is a valid formula.

Presburger arithmetic $\langle \mathbb{N}, +, 0, 1\rangle$ is the theory of first-order logic of addition and successor function ($S(n) = n + 1$) [20]. The interpretation of logical variables is the set of natural numbers $\mathbb{N}$, and the meaning of the function symbols $0, 1, +$ is the natural one. It is well-known that the satisfiability problem for Presburger arithmetic is decidable [20].

$u, v, \ldots \in$	$PVar$	program variables
$x, y, \ldots \in$	$LVar$	location variables
$k, l, \ldots \in$	$IVar$	integer variables
L	$:= \text{nil} \mid u \mid x$	location expressions
I	$:= n \in \mathbb{N} \mid k \mid I + I$	integer expressions
A	$:= I = I \mid L = L \mid \text{emp} \mid L \mapsto L \mid ls^I(L, L)$	atomic propositions
F	$:= \mathbb{T} \mid A \mid \neg F \mid F \wedge F \mid F * F \mid \exists x \,.\, F \mid \exists_{\mathbb{N}} k \,.\, F$	formulae

Fig. 1. Separation Logic with Presburger Arithmetic

The syntax of **QSL** is given in Figure 1. Notice the difference between program variables $PVar$ and location variables $LVar$, the former being logical constants, whereas the latter may occur within the scope of a quantifier. Let $FV_L(\varphi) = FV(\varphi) \cap LVar$ and $FV_I(\varphi) = FV(\varphi) \cap IVar$ denote the sets of location and integer free variables of φ, respectively.

As usual, we define $\varphi \vee \psi \stackrel{\Delta}{=} \neg(\neg\varphi \wedge \neg\psi)$, $\varphi \Rightarrow \psi \stackrel{\Delta}{=} \neg\varphi \vee \psi$, $\forall x \,.\, \varphi \stackrel{\Delta}{=} \neg\exists x \,.\, \neg\varphi$ and $\forall_{\mathbb{N}} k \,.\, \varphi \stackrel{\Delta}{=} \neg\exists_{\mathbb{N}} k \,.\, \neg\varphi$. Moreover, we write $k \leq l$ and $ls(x,y)$ as shorthands for $\exists_{\mathbb{N}} k' \,.\, k + k' = l$ and $\exists_{\mathbb{N}} k \,.\, ls^k(x,y)$, respectively. $\mathbb{F}$ is a shorthand for $\neg\mathbb{T}$. The bounded quantifiers $\exists_{\mathbb{N}} m \leq n \,.\, \varphi(m)$ and $\forall_{\mathbb{N}} m \leq n \,.\, \varphi(m)$ are used instead of $\exists_{\mathbb{N}} m \,.\, m \leq n \wedge \varphi(m)$ and $\forall_{\mathbb{N}} m \,.\, m \leq n \Rightarrow \varphi(m)$, respectively. We shall also deploy some of the classical shorthands in Separation Logic: $x \mapsto \_ \stackrel{\Delta}{=} \exists y \,.\, x \mapsto y$, and $x \hookrightarrow y \stackrel{\Delta}{=} x \mapsto y * \mathbb{T}$, where y is either a location variable or nil. For list segment formulae we define $\widetilde{ls}^k(x,y) \stackrel{\Delta}{=} ls^k(x,y) * \mathbb{T}$ and $\widetilde{ls}(x,y) \stackrel{\Delta}{=} ls(x,y) * \mathbb{T}$.

The semantics of **QSL** formulae is given in terms of heaps. A *heap* is a rooted graph in which each node has at most one successor. Let *Loc* denote the set of *locations*. We assume henceforth that *Loc* is an infinite, countable set, with a designated element $nil \in Loc$. In what follows, we identify heaps that differ only by a renaming of their locations.

Definition 1. *A* heap *is a pair $H = \langle s, h \rangle$, where $s : PVar \cup LVar \to Loc_\perp$ associates variables with locations, and $h : Loc \to Loc_\perp$ is the partial successor mapping. In particular, we have $h(nil) = \perp$. We denote by $\mathcal{H}$ the set of all heaps with variables from $PVar \cup LVar$ and locations from Loc.*

The interpretation of a formula is defined by a forcing relation $\models$ between tuples $\langle H, \nu, \iota \rangle \in \mathcal{H} \times (LVar \mapsto Loc_\perp) \times (IVar \mapsto \mathbb{N}_\perp)$ and formulae. Here $\nu : LVar \to Loc_\perp$ is a partial valuation of location variables, and $\iota : IVar \to \mathbb{N}_\perp$ is a partial valuation of integer variables. The semantics of **QSL** formulae is given below, for a given heap $H = \langle s, h \rangle$:

$$[\![u]\!]_{\langle H,\nu\rangle} = s(u),\ [\![x]\!]_{\langle H,\nu\rangle} = \nu(x),\ [\![\text{nil}]\!]_{\langle H,\nu\rangle} = nil$$

$\langle H, \nu, \iota \rangle \models \mathbb{T}$		always
$\langle H, \nu, \iota \rangle \models L_1 = L_2$	iff	$[\![L_1]\!]_{\langle H,\nu\rangle} = [\![L_2]\!]_{\langle H,\nu\rangle}$
$\langle H, \nu, \iota \rangle \models \text{emp}$	iff	$h = \emptyset$
$\langle H, \nu, \iota \rangle \models L_1 \mapsto L_2$	iff	$h = \{\langle [\![L_1]\!]_{\langle H,\nu\rangle}, [\![L_2]\!]_{\langle H,\nu\rangle} \rangle\}$
$\langle H, \nu, \iota \rangle \models \neg\varphi$	iff	$\langle H, \nu, \iota \rangle \not\models \varphi$
$\langle H, \nu, \iota \rangle \models \varphi \wedge \psi$	iff	$\langle H, \nu, \iota \rangle \models \varphi$ and $\langle H, \nu, \iota \rangle \models \psi$
$\langle H, \nu, \iota \rangle \models \varphi * \psi$	iff	there exist H_1, H_2 such that $H = H_1 \bullet H_2$
		and $\langle H_1, \nu, \iota \rangle \models \varphi, \langle H_2, \nu, \iota \rangle \models \psi$
$\langle H, \nu, \iota \rangle \models \exists x \,.\, \varphi$	iff	$\langle H, \nu[x \leftarrow l], \iota \rangle \models \varphi$ for some $l \in Loc \setminus \{nil\}$

Here $H_1 \bullet H_2$ denotes the disjoint union of $H_1 = \langle s, h_1 \rangle$ and $H_2 = \langle s, h_2 \rangle$, i.e, $dom(h_1) \cap dom(h_2) = \emptyset$, $h = h_1 \cup h_2$. The above definitions are standard in Separation Logic [21]. The rules below are specific to our extension:

$$[\![I]\!]_\iota = I[\iota]$$

$$
\begin{array}{rcl}
\langle H,\nu,\iota\rangle \models I_1 = I_2 & \text{iff} & [\![I_1]\!]_\iota = [\![I_2]\!]_\iota \\
\langle H,\nu,\iota\rangle \models ls^0(L_1,L_2) & \text{iff} & \langle H,\nu,\iota\rangle \models L_1 = L_2 \ \wedge\ \text{emp} \\
\langle H,\nu,\iota\rangle \models ls^{n+1}(L_1,L_2) & \text{iff} & \langle H,\nu,\iota\rangle \models \exists x\,.\, ls^n(L_1,x) * x \mapsto L_2 \\
\langle H,\nu,\iota\rangle \models ls^I(x,y) & \text{iff} & \langle H,\nu,\iota\rangle \models ls^{[\![I]\!]_\iota}(x,y) \\
\langle H,\nu,\iota\rangle \models \exists_{\mathbb{N}} k\,.\,\varphi & \text{iff} & \langle H,\nu,\iota[k \leftarrow n]\rangle \models \varphi, \text{ for some } n \in \mathbb{N}
\end{array}
$$

There are two types of quantifiers, $\exists$ ranges over locations Loc, and $\exists_{\mathbb{N}}$ over natural numbers $\mathbb{N}$. A tuple $\langle H,\nu,\iota\rangle$ is said to be a *model* of φ iff $\langle H,\nu,\iota\rangle \models \varphi$. If $FV(\varphi) = \emptyset$, we denote the fact that H is a model of φ directly as $H \models \varphi$. An *entailment* is a formula of type $\varphi \Rightarrow \psi$. Given such an entailment, the *validity problem* asks if it holds for any tuple $\langle H,\nu,\iota\rangle$, i.e. if any model of φ is also a model of ψ.

Note that we use the "classical" (non-intuitionistic) semantics of Separation Logic [15], in which a points-to relation $L_1 \mapsto L_2$ is true iff the heap is defined on only one cell whose address is the value of L_1. As a result, $ls^k(L_1,L_2)$ is true iff the heap is defined only on the set of addresses that form the list from L_1 up to (but not including) L_2. Another consequence of using this semantics is that $ls^0(L_1,L_2)$ is true only on the empty heap. One can however recover the intuitionistic semantics of [21] by using the shorthands $L_1 \hookrightarrow L_2$ and $\widetilde{ls}^k(L_1,L_2)$ instead.

The following notion of *dangling location* is essential for the semantics of Separation Logic on heaps [15, 21]. To understand this point, consider the formula $\varphi : (u \mapsto v) * (v \mapsto nil)$, describing a heap $H = \langle s,h\rangle$, in which u and v are allocated to two different cells, i.e. $s(u) = l_1$, $s(v) = l_2$, and nothing else is in the domain of the heap, i.e. $h = \{\langle l_1,l_2\rangle, \langle l_2,nil\rangle\}$. The reason for which $H \models \varphi$, is that there exists two disjoint heaps, namely $H_1 = \langle s, \{\langle l_1,l_2\rangle\}\rangle$ and $H_2 = \langle s, \{\langle l_2,nil\rangle\}\rangle$, such that $H_1 \models u \mapsto v$ and $H_2 \models v \mapsto nil$. Notice the role of the location l_2, pointed to by the variable v, which is referenced by the first heap, but allocated in the second one. This location ensures that the disjoint union of H_1 and H_2 is defined, and that $H_1 \bullet H_2 \models u \mapsto v * v \mapsto \text{nil}$.

Definition 2. *A location $l \in Loc \setminus \{nil\}$ is said to be* dangling *in a heap $H = \langle s,h\rangle$ iff $l \in (img(s) \cup img(h)) \setminus dom(h)$.*

In the following, we denote by $dng(H)$ the set of all dangling nodes of H, and by $loc(H) = img(s) \cup dom(h) \cup img(h)$ the set of all locations, either defined or dangling in H.

2.1 Motivating Example

Let us consider the program in Figure 2. The loop on the left hand side inserts elements into the list pointed to by u, while incrementing the c counter, and the loop on the right removes the elements in reversed order, while decrementing c. The pre- and post-condition of the program are inserted as Hoare-style annotations. Both initially and finally, the value of c is zero and the heap is empty.

In order to prove that the program terminates without a null pointer dereferencing, and moreover ensuring that the post-condition holds, one needs to relate the value of c to the length of the list pointed to by u, as it is done in the invariants of the left and right

$\{c = 0 \wedge \mathbf{emp} \wedge u = \text{nil}\}$

1: while ... do	7: while $c \neq 0$ do
$\{c \geq 0 \wedge c = k \wedge \mathbf{ls}^k(u, nil)\}$	$\{c > 0 \wedge c = k \wedge \mathbf{ls}^k(u, nil)\}$
2: t := new;	8: u := u.next;
3: t.next := u;	9: c := c - 1;
4: u := t;	10: od
5: c := c + 1;	$\{c = 0 \wedge \mathbf{emp}\}$
6: od	

Fig. 2. Program verification using **QSL**

hand side : $c = k \wedge \mathbf{ls}^k(u, nil)$. This example could not be handled using standard Separation Logic, since we explicitly need the ability of reasoning about both list lengths and integer variables.

3 Undecidability of QSL

In this section we prove the undecidability of the **QSL** logic. Namely the class of formulae with quantifier prefix in the language $\exists^* \exists^*_{\mathbb{N}} (\forall \mid \forall_{\mathbb{N}}) \exists^* \exists^*_{\mathbb{N}}$ are shown to have an undecidable satisfiability problem. It is to be noticed that undecidability of **QSL** is not a direct consequence of the undecidability of Separation Logic [19], since the proof in [19] uses multiple selector heaps, while in this case we consider only heaps composed of singly-linked lists. Our result is non-trivial since it is well-known also that, even simple logics, e.g. FOL, MSOL are decidable when interpreted over singly-linked lists, and become quickly undecidable when interpreted over grid-like, and more general graph structures.

Theorem 1. *The set of* **QSL** *formulae which, written in prenex normal form, have the quantifier prefix in the language* $\exists^* \exists^*_{\mathbb{N}} (\forall \mid \forall_{\mathbb{N}}) \exists^* \exists^*_{\mathbb{N}}$*, is undecidable.*

The idea of the proof is that one can encode all terminating runs of an arbitrary 2-counter machine [18] by a formula of **QSL** in the $\exists^* \exists^*_{\mathbb{N}} (\forall \mid \forall_{\mathbb{N}}) \exists^* \exists^*_{\mathbb{N}}$ quantifier fragment. Since the halting problem is undecidable for 2-counter machines, the satisfiability of formulae in the above mentioned fragment is also undecidable.

In the following developments, we shall prove that logical entailment in the $\exists^* \{\exists_{\mathbb{N}}, \forall_{\mathbb{N}}\}^*$ fragment of **QSL** is decidable. We have found no argument for (un)decidability concerning the quantifier prefix fragment $\{\exists, \forall\}^* \{\exists_{\mathbb{N}}, \forall_{\mathbb{N}}\}^*$. In particular, all attempts to reduce (from) to known fragments of MSO with cardinality constraints [16] have failed.

4 Model Theoretic Method

The validity of an entailment $\varphi \Rightarrow \psi$ is equivalent to the non-satisfiability of the formula $\varphi \wedge \neg\psi$, i.e. there should be no tuples $\langle H, \nu, \iota \rangle$ such that $\langle H, \nu, \iota \rangle \models \varphi$ and $\langle H, \nu, \iota \rangle \not\models \psi$. Our main result, leading immediately to decidability of entailments, is that, if φ is

of the form $\exists x_1 \ldots \exists x_n Q_1 l_1 \ldots Q_m l_m \ . \ \theta(\mathbf{x}, \mathbf{l})$, with $Q_i \in \{\exists_{\mathbb{N}}, \forall_{\mathbb{N}}\}$ and θ is a boolean combination of predicates with $\neg$, $\wedge$ and $*$, all models $\langle H, \nu, \iota \rangle$ of φ can be represented using a finite number of (finite) structures called symbolic graph representations (SGR).

The decision procedure for the validity of a QSL entailment $\varphi \Rightarrow \psi$ is based on the following idea. We first define operators on sets of SGRs that are the counterparts of the logical connectives $\vee$, $\wedge$, $*$ and the existential quantifiers $\exists x$, $\exists_{\mathbb{N}} x$. Second, for each existential QSL formula φ, we compute a set $[\![\varphi]\!]$ of SGRs that represent all models of φ. The construction of this set is recursive, on the structure of φ. The entailment $\varphi \Rightarrow \psi$ is valid iff the set $[\![\varphi]\!] \ominus [\![\psi]\!]$ is empty, where $\ominus$ is an operator defined on SGRs, that computes the representation of the difference between the set of concrete models of the φ and the one of ψ. Least, the emptiness problem for sets of SGRs is shown to be decidable, by reduction to the satisfiability problem for the Presburger arithmetic.

4.1 Symbolic Shape Graphs

In this section we define a finite representation of (possibly infinite) sets of heaps, called symbolic shape graphs (SSG), which is the essence of our decision method. The next section defines the SGR representation for sets of heaps, which is based on SSGs and arithmetic constraints.

Definition 3. *Given a heap $H = \langle s, h \rangle \in \mathcal{H}$, a location $l \in Loc$ is said to be a* cut point *in H if either $l \in img(s) \cup dng(H) \cup \{nil\}$, or there exists two distinct locations $l_1, l_2 \in Loc$ such that $h(l_1) = h(l_2) = l$.*

A location l is a cut point in a heap if either (1) l is pointed to directly by a program variable, i.e. $l \in img(s)$, (2) l is dangling or *nil*, or (3) l has more than one predecessor in the heap. We denote by $l_1 \rhd_H l_2$ the fact that $h(l_1) = l_2$ and $l_2 \neq \bot$ is not a cut point in H. Let $\sim_H$ denote the reflexive, symmetric and transitive closure of the $\rhd_H$ relation, i.e. the smallest equivalence relation that includes $\rhd_H$, and $[l]_\sim$ be the equivalence class of $l \in Loc$ w.r.t. $\sim_H$. We also refer to these equivalence classes as to *list segments*. By convention, we have $[\bot]_\sim = \bot$. Let $H_{/\sim} = \langle s_{/\sim}, h_{/\sim} \rangle$ be the *quotient heap*, where:

- $s_{/\sim} : PVar \cup LVar \to Loc_{/\sim_\bot}$ and $s_{/\sim}(u) = [s(u)]_\sim$, for all $u \in PVar$,
- $h_{/\sim} : Loc_{/\sim} \to Loc_{/\sim_\bot}$ and for all $l \in dom(h)$, if $h(l) = l'$ and l' is either $\bot$ or a cut point in $\langle s, h \rangle$, then $h_{/\sim}([l]) = [l']_\sim$. In particular, $h_{/\sim}([l]) = \bot$, for all $l \notin dom(h)$.

Note that $s_{/\sim}$ and $h_{/\sim}$ are well-defined functions. We extend the rest of notations to quotient heaps, i.e. $dng(H_{/\sim}) = \{[l]_\sim \mid l \in dng(H)\}$ and $loc(H/\sim) = \{[l]_\sim \mid l \in loc(H)\}$.

For example, in the heap from Figure 3 (a), the cut points are marked by hollow nodes and the $\sim$-equivalence classes are enclosed in solid boxes. The quotient heap is the heap in which these boxes are taken as nodes, instead of the individual locations.

Definition 4. *Given a set PVar of program variables, a set LVar of location variables, and a set of* counters $\mathcal{Z} = \{z_1, \ldots, z_n\}$*, a* symbolic shape graph *(SSG) is a tuple $G = \langle N, D, R, Z, S, V \rangle$, where:*

- *N is a finite set of symbolic nodes, with a designated node $Nil \in N$,*
- *$D \subseteq N$ is a set of symbolic dangling nodes,*

– $R \subseteq N$ *is a set of symbolic root nodes,*
– $Z : N \setminus D \to \mathcal{Z}$ *is an injective function assigning each non-dangling node to a counter,*
– $S : N \to N_\perp$ *is the successor function, where:*
  - $S(Nil) = \perp$ *and* $S(d) = \perp$*, for all* $d \in D$*,*
  - $S(n) \notin R$*, for all* $n \in N$*,*
  - $S(n) \in N$*, for all* $n \in N \setminus (D \cup \{Nil\})$*.*
– $V : PVar \cup LVar \to N$ *assigns program and location variables with nodes.*

Intuitively, each node of a SSG represents a list segment of a concrete heap. The node *Nil* stands for the concrete *nil* location, and each symbolic dangling node represents one dangling location.

Definition 5. *An SSG* $G = \langle N, D, R, Z, S, V \rangle$ *is said to be in* normal form *if:*

– *each node in* $n \in N \setminus \{Nil\}$ *is reachable either from* $V(u)$*, for some* $u \in PVar \cup LVar$*, or from some symbolic root* $r \in R$*, and*
– *either* $n \in img(V) \cup D \cup \{Nil\}$*, or there exist two distinct nodes* $n_1, n_2 \in N$ *such that* $S(n_1) = S(n_2) = n$*.*

$\mathcal{S}_k$ *denotes the set of SSGs in normal form, with* $|R| \leq k$ *and* $img(V) \subseteq PVar \cup LVar$*.*

Sometimes we denote by $\mathcal{S}$ the union $\bigcup_{k \in \mathbb{N}} \mathcal{S}_k$. We identify SSGs which are equivalent under renaming of nodes and counters. The following was proved in [9]:

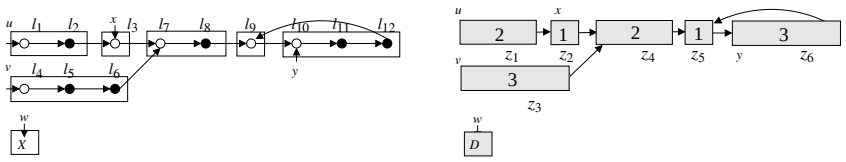
Lemma 1. *Let* $G = \langle N, D, R, Z, S, V \rangle \in \mathcal{S}_k$ *be a normal form SSG. Then,* $|N| \leq 2(|dom(V)| + |R|)$*. As a consequence, the number of such SSGs is bounded asymptotically by* $2(|PVar| + |LVar| + k)^{2(|PVar| + |LVar| + k)}$*, and the bound is tight.*

The following definition relates the notions of heap and SSG.

Definition 6. *Let* $G = \langle N, D, R, Z, S, V \rangle \in \mathcal{S}$ *be a SSG,* $\nu : dom(V) \cap LVar \to dom(h)$ *a valuation of the location variables of G, and* $\iota : img(Z) \to \mathbb{N}^+$ *a valuation of the counters in G. Let* $H = \langle s, h \rangle \in \mathcal{H}$ *be a heap such that* $dom(s) = dom(V) \cap PVar$*, and* $H_{/\sim} = \langle s_{/\sim}, h_{/\sim} \rangle$ *be the quotient of H with respect to* $\sim_H$*. We say that H is* the $\langle \nu, \iota \rangle$-concretization *of G iff there exists a bijective mapping* $\eta : N_\perp \to (loc(h_{/\sim}) \cup \{nil, \perp\})$ *such that:*

– $\eta(Nil) = \{nil\}$ *and* $\eta(\perp) = \perp$*,*
– $\eta(V(u)) = s_{/\sim}(u)$*, for all* $u \in PVar$*,*
– $\eta(S(n)) = h_{/\sim}(\eta(n))$*, for all* $n \in N \setminus D$*,*
– $\eta(n) \in dng(H_{/\sim})$*, for all* $n \in D$*,*
– $\iota(Z(n)) = |\eta(n)|$*, for all* $n \in N \setminus D$*.*

We recall upon the fact that heaps are identical, up to isomorphism, which implies that a $\langle \nu, \iota \rangle$ is uniquely defined. We say that H is a concretization of G if there exist ν, ι such that H is the $\langle \nu, \iota \rangle$-concretization of G. Roughly speaking, the $\langle \nu, \iota \rangle$-concretization of a SSG G is the heap obtained by replacing each node n of G with a list segment whose length equals the value of the counter $Z(n)$. Moreover, if G has a $\langle \nu, \iota \rangle$-concretization,



(a) (b)

Fig. 3. SSG and Concretization

we must have $\iota(Z(n)) > 0$, for all non-dangling symbolic nodes $n \in N$. Notice also that dangling locations are represented by symbolic dangling nodes. We denote by $\gamma_{\nu,\iota}(G)$ the $\langle \nu, \iota \rangle$-concretization of G and by $\Gamma(G)$ the set of all concretizations of G.

For example, the SSG in Figure 3 (b) has as $\langle \nu, \iota \rangle$-concretization the heap in Figure 3 (a), for the valuations:

- $\nu(x) = l_3$, $\nu(y) = l_{10}$, and
- $\iota(z_1) = 2$, $\iota(z_2) = 1$, $\iota(z_3) = 3$, $\iota(z_4) = 2$, $\iota(z_5) = 1$, $\iota(z_6) = 3$.

Notice that the symbolic dangling node pointed to by w corresponds to a dangling location pointed to by w in Figure 3 (a).

The following result expresses the fact that one heap may not be the concretization of two different (non-isomorphic) SSGs:

Lemma 2. *For two non-isomorphic SSGs $G_1, G_2 \in \mathcal{S}$, we have $\Gamma(G_1) \cap \Gamma(G_2) = \emptyset$.*

4.2 Symbolic Graph Representations

In this section we introduce the notion of symbolic graph representation (SGR) together with a number of operators on these structures. In the next section, we shall provide a stepwise translation of a **QSL** formula with quantifier prefix $\exists^* \{\exists_{\mathbb{N}}, \forall_{\mathbb{N}}\}^*$ into a set of symbolic graph representations.

A *symbolic graph representation* is a pair $\langle G, \varphi \rangle$, where $G = \langle N, D, R, Z, S, V \rangle$ is a SSG in normal form and φ an open formula over the counters of G, i.e. $FV(\varphi) \subseteq img(Z)$. By $\mathcal{G}$ we denote the set of all SGRs $R = \langle G, \varphi \rangle$, where $G \in \mathcal{S}$ and the set of counters in each G is a subset of $\mathcal{Z}$.

A heap $H = \langle s, h \rangle$ is the $\langle \nu, \iota \rangle$-concretization of $\langle G, \varphi \rangle$ iff $\nu : dom(V) \cap LVar \to dom(h)$ is a valuation of the location variables of G, and $\iota : img(Z) \to \mathbb{N}^+$ is a valuation of the counters in G that satisfies φ, i.e. $\iota \models \varphi$. This is denoted in the following as $H = \gamma_{\nu,\iota}(\langle G, \varphi \rangle)$. $\Gamma(\langle G, \varphi \rangle)$ denotes the set of all $\langle \nu, \iota \rangle$-concretizations of $\langle G, \varphi \rangle$. The notation is lifted to finite sets of SGRs in the obvious way: $\Gamma(\{R_1, \ldots, R_n\}) = \bigcup_{i=1}^{n} \Gamma(R_i)$.

We introduce now three operators on finite sets of SGRs, that correspond to the boolean operators of union, intersection and set difference. Let $S_1, S_2 \subseteq \mathcal{G}$ be two finite sets of SGRs.

$$S_1 \sqcup S_2 = \{\langle G, \varphi_1 \vee \varphi_2\rangle \mid \langle G,\varphi_1\rangle \in S_1 \text{ and } \langle G,\varphi_2\rangle \in S_2\} \cup \{\langle G,\varphi\rangle \in S_1 \mid \langle G,\_\rangle \notin S_2\} \cup \{\langle G,\varphi\rangle \in S_2 \mid \langle G,\_\rangle \notin S_1\}$$

$$S_1 \sqcap S_2 = \{\langle G, \varphi_1 \wedge \varphi_2\rangle \mid \langle G,\varphi_1\rangle \in S_1 \text{ and } \langle G,\varphi_2\rangle \in S_2\}$$

$$S_1 \ominus S_2 = \{\langle G, \varphi_1 \wedge \neg\varphi_2\rangle \mid \langle G,\varphi_1\rangle \in S_1 \text{ and } \langle G,\varphi_2\rangle \in S_2\} \cup \{\langle G,\varphi\rangle \in S_1 \mid \langle G,\_\rangle \notin S_2\}$$

Here the notation $\langle G,\_\rangle$ stands for any SGR pair having G as its first component. Let $G = \langle N,D,R,Z,S,V\rangle$ and notice that, since $FV(\varphi_1) \subseteq img(Z)$ and $FV(\varphi_2) \subseteq img(Z)$, then $FV(\varphi_1 \vee \varphi_2)$, $FV(\varphi_1 \wedge \varphi_2)$ and $FV(\varphi_1 \wedge \neg\varphi_2)$ are also subsets of $img(Z)$.

Lemma 3. *SGRs are effectively closed under union, intersection and difference. In particular, we have* $\Gamma(S_1 \sqcup S_2) = \Gamma(S_1) \cup \Gamma(S_2)$, $\Gamma(S_1 \sqcap S_2) = \Gamma(S_1) \cap \Gamma(S_2)$ *and* $\Gamma(S_1 \ominus S_2) = \Gamma(S_1) \setminus \Gamma(S_2)$.

The $\circledast$ operator is defined on SGRs with the following meaning : for two SGRs R_1 and R_2, we have $\Gamma(R_1 \circledast R_2) = \{H_1 \bullet H_2 \mid H_1 \in \Gamma(R_1) \text{ and } H_2 \in \Gamma(R_2)\}$. In other words, $\circledast$ is the SGR counterpart of the disjoint union operator on heaps. However, $\circledast$ is not a total operator, i.e. it is not defined for any pair of SGRs, but only for the ones complying with the following definition :

Definition 7. *Two SSGs* $G_i = \langle N_i, D_i, Z_i, S_i, V_i\rangle$, $i = 1,2$ *are said to* match *iff there exists a mapping* $\mu : D_1 \cup D_2 \to (N_1 \cup N_2)_\perp$ *such that, for all* $u \in dom(V_1) \cap dom(V_2)$, *either:*

- $V_1(u) \in D_1$ *and* $\mu(V_1(u)) = V_2(u)$, *or*
- $V_2(u) \in D_2$ *and* $\mu(V_2(u)) = V_1(u)$.

and $\mu(d) = \perp$, *for all* $d \in (D_1 \cup D_2) \setminus (dom(V_1) \cap dom(V_2))$.

Intuitively, two SSGs match if it is possible to relate any dangling node pointed to by program variable in one SSG to a node pointed to by the same variable in the other SSG. Note that two SSGs do not match if the same variable points to some non-dangling node in both. Figure 4 gives an example of two matching SSGs (a) and (b) together with the mapping μ between their nodes (in dotted lines). According to Definition (7), the choice of μ is not unique.

Given two SGRs $R_1 = \langle G_1, \varphi_1\rangle$ and $R_2 = \langle G_2, \varphi_2\rangle$, with *matching* underlying SSGs $G_i = \langle N_i, D_i, Z_i, S_i, V_i\rangle$, (for the purposes of this definition, we can assume w.l.o.g. that $N_1 \cap N_2 = \{Nil\}$ and $img(Z_1) \cap img(Z_2) = \emptyset$), we define $R_1 \circledast R_2 = \langle G, \varphi_1 \wedge \varphi_2\rangle$, $G = \langle N,D,Z,S,V\rangle$, where:

- $N = (N_1 \cup N_2) \setminus dom(\mu)$,
- $D = (D_1 \cup D_2) \setminus dom(\mu)$,
- $R = (R_1 \cup R_2) \setminus dom(\mu)$,
- $Z = Z_1 \cup Z_2$,
- for all $n \in N$:

$$S(n) = \begin{cases} S_i(n) & \text{if } n \in N_i \text{ and } S_i(n) \notin dom(\mu) \\ \mu(S_i(n)) & \text{if } n \in N_i \text{ and } S_i(n) \in dom(\mu) \end{cases} \quad i = 1,2$$

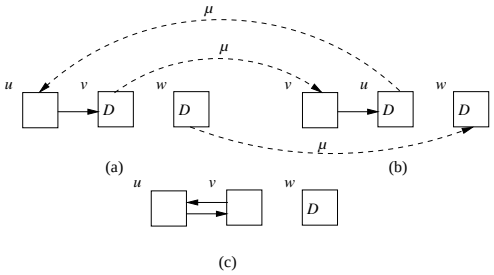


Fig. 4. Matching SSGs

– for all $u \in dom(V_1) \cup dom(V_2)$:

$$V(u) = \begin{cases} V_i(u) & \text{if } V_i(u) \notin dom(\mu) \\ \mu(V_i(u)) & \text{if } V_i(u) \in dom(\mu) \end{cases} \quad i = 1,2$$

For example, the SSG in Figure 4 (c) is the result of the $\circledast$-composition of the SSGs in Figure 4 (a) and (b).

The $\circledast$ operator is undefined, if G_1 and G_2 do not match. Notice that if $G_1 \in \mathcal{S}_{k_1}$, $G_2 \in \mathcal{S}_{k_2}$ and $\langle G, \varphi \rangle = \langle G_1, \varphi_1 \rangle \circledast \langle G_2, \varphi_2 \rangle$, then $G \in \mathcal{S}_{k_1+k_2}$. The correctness of the definition is captured by the following Lemma:

Lemma 4. *Given two SGRs $R_1 = \langle G_1, \varphi_1 \rangle$ and $R_2 = \langle G_2, \varphi_2 \rangle$, such that G_1 and G_2 match, we have $\Gamma(R_1 \circledast R_2) = \{H_1 \bullet H_2 \mid H_1 \in \Gamma(R_1),\ H_2 \in \Gamma(R_2)\}$.*

The following *projection* operator captures the effect of dropping one location variable out of the heap. Let $R = \langle G, \varphi \rangle$ be an SGR, where $G = \langle N, D, R, Z, S, V \rangle$ is the underlying SSG, and $x \in img(V) \cap LVar$ be a location variable occurring in G. For an arbitrary symbolic node $n \in N$, let $prec_G(n) = \{m \in N \mid m \neq n,\ S(m) = n\}$ be the set of predecessors of n, different from itself, in G.

We define $R\downarrow_x$ to be the SGR, having a normal-form underlying SSG (cf. Definition 4), from which x is missing. Formally, let $R\downarrow_x = \langle G', \varphi' \rangle$, where:

1. if $x \notin dom(V)$ then $G' = G$ and $\varphi' = \varphi$.
2. else, if $x \in dom(V)$ and either:
   (a) there exists $u \in dom(V) \setminus \{x\}$ such that $V(u) = V(x)$, or
   (b) there exist $m_1, m_2 \in dom(S)$ s.t. $m_1 \neq m_2$ and $S(m_1) = S(m_2) = V(x)$
   then $G' = \langle N, D, R, Z, S, V[x \leftarrow \bot] \rangle$ and $\varphi' = \varphi$.
3. else, if $x \in dom(V)$, $V(x) = n$, and for all $u \in dom(V) \setminus \{x\}$, we have $V(u) \neq n$, and either:
   (a) $prec_G(n) = \emptyset$, then $G' = \langle N, D, R \cup \{n\}, Z, S, V[x \leftarrow \bot] \rangle$ and $\varphi' = \varphi$, or
   (b) $n \in D$ and $prec_G(n) \neq \emptyset$, then $G' = \langle N, D, R, Z, S, V[x \leftarrow \bot] \rangle$ and $\varphi' = \varphi$,
   (c) $n \notin D$ and $m \in prec_G(n)$, where $Z(m) = k_1$ and $Z(n) = k_2$, then
   $G' = \langle N \setminus \{n\}, D, R, Z[m \leftarrow k_3][n \leftarrow \bot], S[m \leftarrow S(n)][n \leftarrow \bot], V[x \leftarrow \bot] \rangle$ and
   $\varphi' = \exists k_1 \exists k_2 \,.\, \varphi \wedge k_3 = k_1 + k_2$, where $k_3 \notin img(Z)$ is a fresh counter name.

The correctness of this definition is captured in the following Lemma:

Lemma 5. *Let $R = \langle G, \varphi \rangle$ be a SGR, $G = \langle N, D, Z, S, V \rangle$ be its underlying SSG, and $x \in LVar$ be a location variable. Then* $\Gamma(R\downarrow_x) = \{\langle s[x \leftarrow \bot], h\rangle \mid \langle s, h\rangle \in \Gamma(R)\}$.

Given a set S of SGRs, the emptiness problem $\Gamma(S) = \emptyset$ is effectively decidable if all constraints φ occurring within elements $\langle G, \varphi\rangle \in \mathcal{G}$ are written in a logic decidable for satisfiability. In our case, this logic is the Presburger arithmetic, for which the satisfiability problem is known to be decidable [20].

4.3 From Formulae to Sets of SGR

We are now ready to describe the construction of a set of SGRs for a given formula :

$$\varphi \ : \ \exists x_1 \ldots \exists x_n Q_1 l_1 \ldots Q_m l_m \ . \ \theta(\mathbf{x}, \mathbf{l})$$

where $Q_i \in \{\exists_{\mathbb{N}}, \forall_{\mathbb{N}}\}$ and θ is a quantifier-free QSL formula. The construction is performed incrementally, following the structure of the abstract syntax tree of θ. The set $\mathbf{x} = \{x_1, \ldots, x_n\}$ is called from now on the *support set* of θ. Without losing generality, we consider that the leaves of this tree are atomic propositions of one of the forms : $\mathbb{T}$, emp, $x = y$, $x \mapsto y$ and $ls^l(x, y)$, where $x \in \mathbf{x} \cup PVar$, $y \in \mathbf{x} \cup PVar \cup \{nil\}$ and $l \in \{l_1, \ldots, l_m\}$.

From now on, let $\mathcal{S}_k(\mathbf{x})$ be the set of all SSGs with at most k root nodes, support variables from $PVar \cup \mathbf{x}$, and counters from a fixed given set $\mathcal{Z}$. Given a formula φ, we denote by $[\![\varphi]\!]_{\mathbf{x}}(k)$ the set of SGRs with at most k root nodes, over the support set $\mathbf{x}$, defining the models of φ, in the following sense. The set of concrete heaps corresponding to $[\![\varphi]\!]_{\mathbf{x}}(k)$ is exactly the set of models of φ.

For atomic spatial propositions, $[\![\varphi]\!]_{\mathbf{x}}(k)$ is computed according to the definitions from Table 1. In the definition of $[\![\text{emp}]\!]_{\mathbf{x}}(k)$ we consider as parameter the partition $\langle Y_1, \ldots, Y_p\rangle \in Part(\mathbf{x})$. That is $[\![\text{emp}]\!]_{\mathbf{x}}(k) = \bigcup_{\langle Y_1, \ldots, Y_p\rangle \in Part(\mathbf{x})} [\![\text{emp}]\!]^{Y_1, \ldots, Y_p}$, where $[\![\text{emp}]\!]^{Y_1, \ldots, Y_p}$ is defined in Table 1. Intuitively, Y_i, $1 \leq i \leq p$ is the set of variables that are aliased, pointing to the same dangling node d_i, in the empty heap. In Table 1, let $D = \{d_1, \ldots, d_{p-1}\}$, $R = \emptyset$ and $\Delta = \bigcup_{i=1}^{p-1} \lambda x : Y_i.d_i \cup \lambda x : Y_p.Nil$.

Since emp denotes all heaps with empty domain, in the SGR representation there are no symbolic nodes, which are not dangling or nil. Moreover, there are no counters, and therefore, no arithmetic constraints.

In the definition of $[\![\varphi]\!]_{\mathbf{x}}(k)$ for $x \mapsto nil$ and $x \mapsto y$, we consider two parameters: (1) a set $Z \subseteq \mathbf{x} \cup \{x\}$, such that $x \in Z$, and (2) a partition $\langle Y_1, \ldots, Y_p\rangle \in Part(\mathbf{x} \setminus Z)$. In other words, we have $[\![\varphi]\!]_{\mathbf{x}}(k) = \bigcup\{[\![\varphi]\!]_Z^{Y_1, \ldots, Y_p} \mid Z \subseteq \mathbf{x} \cup \{x\},\ x \in Z,\ \langle Y_1, \ldots, Y_p\rangle \in Part(\mathbf{x} \setminus Z)\}$, where $[\![\varphi]\!]_Z^{Y_1, \ldots, Y_p}$ is defined in Table 1. Intuitively, Z corresponds to the set of support variables that are aliased with x in some concrete model, and $\langle Y_1, \ldots, Y_p\rangle$ is used with the same meaning as in the previous definition of $[\![\text{emp}]\!]_{\mathbf{x}}(k)$.

We recall upon the fact that the models of the atomic propositions $x \mapsto nil$ and $x \mapsto y$ are all heaps whose domains consists of only one address, which is the value of x. Therefore the SGR representation uses one non-dangling node n to which one counter z_1 is mapped. The associated constraint sets $z_1 = 1$, according to the semantics.

In the definition of $[\![ls^l(x, nil)]\!]_{\mathbf{x}}(k)$ we consider an ordered sequence of disjoint subsets of $\mathbf{x}$, namely $Z_1, \ldots, Z_k$, where $Z_i \subseteq \mathbf{x} \cup \{x\}$, $1 \leq k \leq n$, such that $x \in Z_1$, and $Z_i \cap$

$Z_j = \emptyset$, for all $1 \leq i < j \leq k$. Similarly, in the definition of $[\![ls^l(x,y)]\!]_{\mathbf{x}}(k)$ we consider sets $Z_1, \ldots, Z_k$, where $Z_i \subseteq \mathbf{x} \cup \{x,y\}$, $1 \leq k \leq n$, such that $x \in Z_1$, $y \in Z_k$, and $Z_i \cap Z_j = \emptyset$, for all $1 \leq i < j \leq k$. In both cases, we consider also a partition $\langle Y_1, \ldots, Y_p \rangle \in Part(\mathbf{x} \setminus (\bigcup_{i=1}^{k} Z_i))$. Intuitively, $Z_1, \ldots, Z_k$ correspond to the sets of support variables that are aliased while pointing to the same node in the list, in some concrete model, and $\langle Y_1, \ldots, Y_p \rangle$ is used with the same meaning as in the previous definition of $[\![\text{emp}]\!]_{\mathbf{x}}(k)$.

Since the models of $ls^k(x, nil)$ and $ls^l(x,y)$ are all heaps defined only on the addresses of the nodes in the list pointed to by x, we represent them by a list of symbolic non-dangling nodes $n_1, \ldots, n_k$, where all variables from Z_i point to n_i, $1 \leq i \leq k$. Each node n_i has an associated counter z_i, and the sum of the values of z_i must equal l. Moreover, if x points to the end (nil or y) of the list, the length l must be zero.

	$[\![\text{emp}]\!]^{Y_1,\ldots,Y_p}$	$[\![x \mapsto \text{nil}]\!]_Z^{Y_1,\ldots,Y_p}$	$[\![x \mapsto y]\!]_Z^{Y_1,\ldots,Y_p}$	$[\![ls^l(x,\text{nil})]\!]_{Z_1,\ldots,Z_k}^{Y_1,\ldots,Y_p}$	$[\![ls^l(x,y)]\!]_{Z_1,\ldots,Z_k}^{Y_1,\ldots,Y_p}$
N	$D \cup \{Nil\}$	$D \cup \{n, Nil\}$	$D \cup \{n, Nil\}$	$D \cup \{n_1, \ldots, n_k, Nil\}$	$D \cup \{n_1, \ldots, n_k, Nil\}$
Z	$\emptyset$	$\{\langle n, z_1 \rangle\}$	$\{\langle n, z_1 \rangle\}$	$\{\langle n_i, z_i \rangle\}_{i=1}^{k}$	$\{\langle n_i, z_i \rangle\}_{i=1}^{k}$
S	$\emptyset$	$\{\langle n, Nil \rangle\}$	$\{\langle n, d_k \rangle\}$, if $y \in Y_k$	$\{\langle n_i, n_{i+1} \rangle\}_{i=1}^{k-1} \cup \{\langle n_k, Nil \rangle\}$	$\{\langle n_i, n_{i+1} \rangle\}_{i=1}^{k-1}$
V	Δ	$\lambda x : Z.n \cup \Delta$	$\lambda x : Z.n \cup \Delta$	$\bigcup_{i=1}^{k} \lambda x : Z_i.n_i \cup \Delta$	$\bigcup_{i=1}^{k} \lambda x : Z_i.n_i \cup \Delta$
φ	$\top$	$z_1 = 1$	$z_1 = 1$	$\sum_{i=1}^{k} z_k = l \wedge$ $(x \in Z_k \rightarrow l = 0)$	$\sum_{i=1}^{k} z_k = l \wedge$ $(x, y \in Z_k \rightarrow l = 0)$

Table 1. SGR for atomic spatial propositions

The pure formulae $x = \text{nil}$ ($x = y$) correspond to sets of SGRs are the ones in which x points to nil (y), and the counters occur unconstrained. Their SGR semantics is defined as follows:

$$[\![x = \text{nil}]\!]_{\mathbf{x}}(k) = \{\langle G, \top \rangle \mid G = \langle N, D, R, Z, S, V \rangle \in \mathcal{S}_k(\mathbf{x}),\ V(x) = Nil\}$$
$$[\![x = y]\!]_{\mathbf{x}}(k) = \{\langle G, \top \rangle \mid G = \langle N, D, R, Z, S, V \rangle \in \mathcal{S}_k(\mathbf{x}),\ V(x) = V(y)\}$$

The SGR semantics for the QSL connectives is defined as follows:

$$\begin{array}{rcl} [\![\mathbb{T}]\!]_{\mathbf{x}}(k) & = & \{\langle G, \top \rangle \mid G \in \mathcal{S}_k(\mathbf{x})\} \\ [\![\psi_1 \wedge \psi_2]\!]_{\mathbf{x}}(k) & = & [\![\psi_1]\!]_{\mathbf{x}}(k) \sqcap [\![\psi_2]\!]_{\mathbf{x}}(k) \\ [\![\neg\psi]\!]_{\mathbf{x}}(k) & = & \{\langle G, \top \rangle \mid G \in \mathcal{S}_k(\mathbf{x})\} \ominus [\![\psi]\!]_{\mathbf{x}}(k) \\ [\![\psi_1 * \psi_2]\!]_{\mathbf{x}}(k) & = & [\![\psi_1]\!]_{\mathbf{x}}(k) \circledast [\![\psi_2]\!]_{\mathbf{x}}(k) \end{array}$$

If π is a purely arithmetic formula, then we have:

$$[\![\psi \wedge \pi]\!]_{\mathbf{x}}(k) = \{\langle G, \theta \wedge \pi \rangle \mid \langle G, \theta \rangle \in [\![\psi]\!]_{\mathbf{x}}(k)\}$$

The semantics for the existential quantifiers is as follows:

$$\begin{array}{rcl} [\![\exists x\ .\ \psi]\!]_{\mathbf{x}}(k) & = & \{R{\downarrow_x} \mid R \in [\![\psi]\!]_{\mathbf{x} \cup \{x\}}(k-1)\},\ k \geq 1 \\ [\![\exists_{\mathbb{N}} l\ .\ \psi]\!]_{\mathbf{x}}(k) & = & \{\langle G, \exists l\ .\ \theta \rangle \mid \langle G, \theta \rangle \in [\![\psi]\!]_{\mathbf{x}}(k)\} \end{array}$$

The following lemma formalizes the correctness of our construction.

Lemma 6. *Given* φ *a* **QSL** *formula containing only numeric quantifiers* ($\exists_{\mathbb{N}}$, $\forall_{\mathbb{N}}$)*, and* $\{x_1,\ldots,x_n\} \subseteq FV_L(\varphi)$*, we have, for all valuations* $\nu : FV_L(\varphi) \setminus \{x_1,\ldots,x_n\} \to Loc$*, and* $\iota : FV_I(\varphi) \to \mathbb{N}^+$*:* $\Gamma_{\nu,\iota}\big([\![\exists x_1 \ldots \exists x_n \,.\, \varphi]\!]_{FV_L(\varphi)\setminus\{x_1,\ldots,x_n\}}(n)\big) = \{H \mid \langle H,\nu,\iota\rangle \models \exists x_1 \ldots \exists x_n \,.\, \varphi\}$

Theorem 2. *The validity of entailments between formulae in the* $\exists^*\{\exists_{\mathbb{N}},\forall_{\mathbb{N}}\}^*$ *quantifier fragment of* **QSL** *is a decidable problem.*

The proof of Theorem 2 uses the fact that the set of models for each formula in the $\exists^*\{\exists_{\mathbb{N}},\forall_{\mathbb{N}}\}^*$ quantifier fragment of **QSL** can be finitely represented using a set of SGRs (cf. Lemma 6). An entailment $\varphi \Rightarrow \psi$ is valid if and only if the set of models of the formula $\varphi \wedge \neg\psi$ is empty. The latter is given by the set of SGRs encoding the set-theoretic difference between the models of φ and the models of ψ. Since the emptiness of this set is decidable, by reduction to Presburger arithmetic, the validity of the entailment is also decidable.

According to Lemma 1, the number of SSGs that can be generated using N variables is of the order of $O(N^N)$. However, the number of SSGs encountered in practice is relatively small, since in principle, the explosion occurs only due to the unrestricted use of negation and the $\mathbb{T}$ proposition, which can be easily avoided.

5 Application of the Model Theoretic Method for QSL

The translation between **QSL** formulae with quantifier prefix of the form $\exists^*\{\exists_{\mathbb{N}},\forall_{\mathbb{N}}\}^*$ and sets of SGRs gives a method for deciding the validity of entailments in this logic. Moreover, there is another, more practical advantage to this approach, that gives us a effective method for the verification of both shape and numeric properties of programs with lists.

The L2CA tool [3] is a tool for verifying safety and termination properties of programs with singly-linked lists, based on the translation of programs into counter automata [9]. A counter automaton generated by L2CA has control states of the form $\langle l, G\rangle$, where l is a control label of the original program, and G is a SSG over the set *PVar* of pointer variables of the input program. By Lemma 1, the set of control states of a counter automaton generated by L2CA is finite, which guarantees that each program with lists will be translated into a finite-control counter automaton. The semantics (set of runs) of the counter automaton generated by L2CA is in a bisimulation relation with the semantics of the original program, therefore all results of the analysis of the counter automaton (e.g. safety properties, termination) carry over to the original program.

The fact that any $\exists^*\{\exists_{\mathbb{N}},\forall_{\mathbb{N}}\}^*$ **QSL** formula φ corresponds to a set $[\![\varphi]\!]$ of pairs $\langle G,\psi\rangle$, where G is an SSG and ψ is a Presburger constraint, allows us to extend the L2CA tool to check total correctness of Hoare triples in which the pre- and post-conditions are expressed as $\exists^*\{\exists_{\mathbb{N}},\forall_{\mathbb{N}}\}^*$ **QSL** formulae. Suppose that $\{\varphi\}$ **P** $\{\psi\}$ is such a triple. Then for each SGR $\langle G_k,\phi_k\rangle \in [\![\varphi]\!]$ the L2CA tool will generate a counter automaton A_k with initial state $\langle l_0, G_k\rangle$, where l_0 is the initial control label of the program **P**. This automaton corresponds to the semantics of **P** when started in an initial control state $\langle l_0, H_0\rangle$, where $H_0 \in \Gamma(\langle G_k,\phi_k\rangle)$. Let A be the union of all such A_k.

By using a combination of existing tools for the analysis of counter automata, e.g. [6, 2, 1] we can verify whether A, started in each control state $\langle l_0, G_k \rangle$ with values of counters satisfying the Presburger constraint ϕ_k, reaches a final control state $\langle l_f, G_f \rangle$ with the counters satisfying some Presburger constraint ϕ[1] such that $\models \phi \rightarrow \phi'$, for some $\langle G_f, \phi' \rangle \in [\![\psi]\!]$. This suffices for checking partial correctness. On what concerns total correctness, we use a termination analysis tool for counter automata, e.g. [1], to check whether **P**, started with any heap H_0 such that $H_0 \models \varphi$, terminates.

5.1 Experimental Results

Table 2 presents some experimental results of verifying Hoare triples of the form $\{\varphi\}\ \mathbf{P}\ \{\psi\}$, where φ and ψ are **QSL** formulae, and **P** is a program handling lists. The ListReversal example receives in input a non-circular list pointed to by u of length l and returns a non-circular list pointed to by v containing the cells of the first list in reversed order. The BubbleSort and InsertSort programs are classical sorting algorithms for which we verified that the length of the input list stays the same. The ListCounter example is a simple loop traversing a list pointed to by u, while incrementing an integer counter c. InsertDelete is the example from Figure 2.

$\{\varphi\}$	**P**	$\{\psi\}$	Size	Gen (s)	Verif (s)	Tool
$\{ls^l(u,nil)\}$	ListReversal	$\{ls^l(v,nil)\}$	4	0.4	0.3	Fast
$\{ls^l(u,nil)\}$	BubbleSort	$\{ls^l(u,nil)\}$	25	0.4	0.4	Aspic
$\{ls^l(u,nil)\}$	InsertSort	$\{ls^l(u,nil)\}$	58	0.6	0.6	Aspic
$\{ls^l(u,nil) \wedge c = 0\}$	ListCounter	$\{ls^l(u,nil) \wedge c = l\}$	16	0.2	0.1	Aspic
$\{c = 0 \wedge emp \wedge u = nil\}$	InsertDelete	$\{c = 0 \wedge emp\}$	6	1.5	0.5	Fast

Table 2. Experimental Results using the L2CA and ASPIC tools

For all examples, the size (number of control locations) of the automata generated by L2CA is given in the second (Size) column, the time needed for generation in the third (Gen) column, and the time needed to verify partial correctness of the model is given in the fourth (Verif) column. The tool used (either Aspic [2] or Fast [6]) is given in the fifth column. All programs were found to be correct.

6 Conclusions

We have developed an extension of Separation Logic interpreted over singly-linked heaps, that allows to specify properties related to the sizes of the lists. This logic is especially useful for reasoning about programs that combine dynamically allocated data with variables ranging over integer domains.

The decidability of the extended logic is studied, the full quantifier fragment being shown to be undecidable, by a reduction from the halting problem for 2 counter machines. However the validity of entailments in the $\exists^*\{\exists_{\mathbb{N}}, \forall_{\mathbb{N}}\}^*$ fragment of the logic

[1] In general this is an over-approximation of the set of reachable configurations, obtained using a combination of precise (acceleration) and abstract (widening) methods.

is decidable, which allows the use this fragment to specify Hoare triples for programs with lists. The verification of total correctness properties specified in this way was made possible by an extension of the L2CA tool.

References

1. ARMC. http://www.mpi-sb.mpg.de/~rybal/armc/.
2. ASPIC. http://www-verimag.imag.fr/~gonnord/aspic/aspic.html.
3. L2CA. http://www-verimag.imag.fr/~async/L2CA/l2ca.html.
4. Smallfoot. http://www.dcs.qmul.ac.uk/research/logic/theory/projects/smallfoot/index.html.
5. A. Annichini, A. Bouajjani, and M.Sighireanu. Trex: A tool for reachability analysis of complex systems. In *Proc.CAV*, volume 2102 of *LNCS*, pages 368 – 372. Springer, 2001.
6. S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. Fast: Fast accelereation of symbolic transition systems. In *Proc. TACAS*, volume 2725 of *LNCS*. Springer, 2004.
7. M. Benedikt, T. Reps, and M. Sagiv. A decidable logic for describing linked data structures. In Springer Verlag, editor, *Proc. European Symposium On Programming*. LNCS, 1999.
8. J. Berdine, C. Calcagno, and P. O'Hearn. A Decidable Fragment of Separation Logic. In *FSTTCS*, volume 3328 of *LNCS*, 2004.
9. 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.
10. M. Bozga, R. Iosif, and S. Perarnau. Quantitative separation logic and programs with lists. Technical Report TR 2007-9, VERIMAG, 2007.
11. R. M. Burstall. Some techniques for proving correctness of programs which alter data structures. *Machine Intelligence*, 7:23–50, 1972.
12. S. Gulwani, B. McCloskey, and A. Tiwari. Lifting abstract interpreters to quantified logical domains. In *Proc. 35th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM, 2008.
13. S. Gulwani and A. Tiwari. An abstract domain for analyzing heap-manipulating low-level software. In *Proc. Intl. Conference on Computer Aided Verification*, 2007.
14. N. Immerman, A. Rabinovich, T. Reps, M. Sagiv, and G. Yorsh. Verification via Structure Simulation. In *CAV*, volume 3114 of *LNCS*, 2004.
15. S. Ishtiaq and P. O'Hearn. BI as an assertion language for mutable data structures. In *POPL*, 2001.
16. F. Klaedtke and H. Ruess. Monadic second-order logics with cardinalities. In *Proc.30th International Colloquium on Automata, Languages and Programming*. LNCS, 2003.
17. S. Magill, J. Berdine, E. Clarke, and B. Cook. Arithmetic Strengthening for Shape Analysis. In *SAS*, volume 4634 of *LNCS*, 2007.
18. M. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.
19. P. O'Hearn, C. Calcagno, and H. Yang. Computability and Complexity Results for a Spatial Assertion Language for Data Structures. In *FSTTCS*, volume 2245 of *LNCS*, 2001.
20. M. Presburger. Über die Vollstandigkeit eines gewissen Systems der Arithmetik. *Comptes rendus du I Congrés des Pays Slaves*, Warsaw 1929.
21. J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In Springer Verlag, editor, *Proc. 17th IEEE Symposium on Logic in Computer Science*. LNCS, 2002.
22. P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *Proc. CAV*, volume 1427 of *LNCS*, pages 88–97. Springer, 1998.
23. G. Yorsh, A. Rabinovich, M. Sagiv, A. Meyer, and A. Bouajjani. A logic of reachable patterns in linked data-structures. In *Proc. Foundations of Software Science and Computation Structures*. LNCS, 2006.
24. T. Zhang, H. Sipma, and Z. Manna. Decision procedures for recursive data structures with integer constraints. In *Proc. Intl. Joint Conference of Automated Reasoning*, 2004.