Decision Procedure for Separation Logic with Inductive Definitions and Presburger Arithmetic

Makoto Tatsuta[1], Quang Loc Le[2], and Wei-Ngan Chin[3]

[1] National Institute of Informatics / Sokendai, Tokyo, tatsuta@nii.ac.jp
[2] Singapore University of Technology and Design, Singapore
[3] National University of Singapore, Singapore

Abstract. This paper considers the satisfiability problem of symbolic heaps in separation logic with Presburger arithmetic and inductive definitions. First the system without any restrictions is proved to be undecidable. Secondly this paper proposes some syntactic restrictions for decidability. These restrictions are identified based on a new decidable subsystem of Presburger arithmetic with inductive definitions. In the subsystem of arithmetic, every inductively defined predicate represents an eventually periodic set and can be eliminated. The proposed system is quite general as it can handle the satisfiability of the arithmetical parts of fairly complex predicates such as sorted lists and AVL trees. Finally, we prove the decidability by presenting a decision procedure for symbolic heaps with the restricted inductive definitions and arithmetic.

1 Introduction

In the last decade, separation logic has provided an appealing paradigm to support memory safety verification [1, 2]. For automated program verification, it is necessary to decide the truth of entailment of symbolic heaps. This paper will examine the decidability of the satisfiability problem for symbolic heaps. Decision procedures for satisfiability are important to support entailment proving [5]. It can directly support entailment proving if complement operation is available. If complementation is unavailable, the unsatisfiability outcome is still important for pruning of infeasible program sub-states during entailment proving itself.

This paper considers the symbolic heaps as the conjunction of equalities and disequalities, and the spatial conjunction of empty heap, points-to predicate, and inductive predicates. Inductive definitions for symbolic-heap systems are important [7, 8, 4] as they can provide a flexible way to express a wide range of recursive data structures. Recently, various extensions of symbolic heaps with arithmetic have been advocated for verifying both quantitative properties and data contents [5, 11, 9, 10]. These extensions aim to handle more complex data structures involving arithmetic as well as shape information, such as length of lists, minimum values of lists, sorted lists, and even height-balanced AVL trees.

Our work extends the satisfiability decision procedure for symbolic heaps with inductive definitions [4] to symbolic heaps with inductive definitions and Presburger arithmetic. First we show that the satisfiability of symbolic heaps in

the system SLA1 which includes (unrestricted) inductive definitions and Presburger arithmetic is undecidable. The undecidability is proved by simulating multiplication and reducing to it Peano arithmetic which is undecidable.

Next, we propose some restrictions on SLA1 to obtain a decidable subsystem, called SLA2. For this purpose, we will use the three ideas: (1) a decidable subsystem DPI of Presburger arithmetic and inductive definitions, (2) projections and unfolding trees, and (3) a periodic structure in the sequence of base pairs.

Our first idea is to propose the decidable system DPI as a subsystem of Presburger arithmetic and inductive definitions with some restrictions. Although the truth for Presburger arithmetic is known to be decidable [6], the decidability of Presburger arithmetic and inductive definitions is challenging; it is undecidable without any restrictions, since some inductive predicate can simulate multiplication and reduce Peano arithmetic to it. We will choose the restrictions so that inductive predicates exactly represent eventually periodic sets. Our choice is reasonable, since Presburger arithmetic is one of the strongest decidable systems and eventually periodic sets are the same as sets characterized by some Presburger arithmetical formulas [6]. Under this restriction, we can show the decidability by eliminating inductive predicates. Our restriction seems complicated, but it is quite general as it can handle non-trivial data structures, such as arithmetical parts of sorted lists and AVL trees.

Our second idea is to decide the satisfiability of a given symbolic heap in separation logic with inductive definitions and arithmetic by deciding the satisfiability of its spatial part and its numeric part. The former satisfiability always implies the latter satisfiability, but the converse does not necessarily hold. In order to synchronize these two parts and guarantee the converse, we will use unfolding trees, which are described, for example, in [7]. An unfolding tree T of an inductive predicate P specifies how P is unfolded. Thus, P unfolded by T is true in separation logic with inductive definitions and arithmetic iff the spatial projection of P unfolded by T is true in separation logic with inductive definitions and the numeric projection of P unfolded by T is true in arithmetic with inductive definitions.

Our third idea is to use base pairs. Brotherston et al [4] showed the satisfiability of symbolic heaps is decidable in the system of separation logic with inductive definitions. They introduced base pairs and an inductive predicate is interpreted by a set of base pairs. In this paper, we will use their ideas and interpret a symbolic heap without inductive predicates by a single base pair. One of the key observations is that we can find some periodic structure in a given sequence of the interpretations of symbolic heaps, since the set of base pairs is finite in our setting.

Then we will define the decidable system SLA2 as the system SLA1 with some restrictions to inductive definitions, so that the arithmetical part of inductive definitions are those of the system DPI. When the unfolding trees are linear, we can find some periodic structure in the sequence of base pairs that interpret the inductive predicate unfolded by those trees, which enables us to decide its satisfiability. For any tree-like data structures, the system SLA2 concurrently

allows size properties, such as the length of lists and the height of trees, and data information such as the minimum and the maximum of data.

To summarize, we make the following technical contributions in this paper: (1) We prove that SLA1 is undecidable. (2) We propose the decidable subsystem DPI of Presburger arithmetic with inductive definitions, and prove its decidability. (3) We present the decidable subsystem SLA2 of symbolic heaps with Presburger arithmetic and inductive definitions, provide its decision procedure, and prove its decidability.

The decidability results of this paper provide theoretical foundations to advance satisfiability decision procedures in verification systems of heap-manipulating programs, like [5, 11, 9, 10]. A system of symbolic heaps with inductive definitions and arithmetic adds significantly to the expressivity of our specification logic. However no decidability results for such a system have been achieved prior to our current proposal. For symbolic-heap systems with inductive definitions and without arithmetic, [4] shows the decidability of the satisfiability of symbolic heaps, and [7, 8] proves the decidability of the truth of the entailments of symbolic heaps under some restrictions such as bounded treewidth. For symbolic-heaps systems with arithmetic and without inductive definitions, entailment decision procedures for hard-coded predicates and entailment of prenex formulas with some quantification were proposed in [11, 9, 10, 3]. For symbolic-heaps systems with inductive definitions and arithmetic, [5] provided a semi-decision procedure for the validity of the entailments for symbolic heaps. Our results thus provide an important step towards state-of-the-art research on the decidability of symbolic heaps with inductive definitions and arithmetic.

Section 2 defines the system SLA1 and its semantics, and shows the undecidability. Section 3 proposes the decidable subsystem DPI of Presburger arithmetic with inductive definitions, and proves its decidability. The decidable system SLA2 is presented in Section 4. This section also defines unfolding trees and base pairs, and proves the decidability of SLA2 by providing its decision procedure. We conclude in Section 5.

2 System SLA1

We start off by defining the system SLA1 of separation logic and Presburger arithmetic with inductive definitions. By combination of separation logic and arithmetic, this system can describe range of complex data structures with pure properties, for example, sorted lists with length information.

2.1 Syntax

We use vector notations $\mathbf{x}$ to denote a sequence $x_1, \ldots, x_k$. $|\mathbf{x}|$ denotes the length of the sequence. For simplicity sometimes we also use a notation of a sequence to denote a set. We also write $\mathbf{x} = \mathbf{y}$ to denote $x_i = y_i$ for all i, and $f(\mathbf{x})$ for the sequence $f(x_1), \ldots, f(x_k)$. We write $\equiv$ for the syntactical equivalence. N denotes the set of natural numbers.

$$
\begin{array}{ll}
\text{Pointer terms } t & ::= x \mid \text{nil} \\
\text{Pure formulas } \Pi & ::= \text{true} \mid \text{false} \mid t = t \mid t \neq t \mid \Pi \wedge \Pi \\
\text{Integer constants } k & ::= \ldots \mid -1 \mid 0 \mid 1 \mid 2 \mid \ldots \\
\text{Arithmetical terms } a & ::= x \mid k \mid k \times a \mid a + a \mid -a \mid \max(a,a) \mid \min(a,a) \\
\text{Arithmetical formulas } \Lambda & ::= \text{true} \mid a = a \mid a \leq a \mid \neg\Lambda \mid \Lambda \wedge \Lambda \mid \exists x.\Lambda \\
\text{Terms } u & ::= a \mid t \\
\text{Spatial formulas } \Sigma & ::= \text{emp} \mid t \mapsto (u_1, \ldots, u_{\mathrm{NC}}) \mid P(\mathbf{t}, \mathbf{a}) \mid \Sigma * \Sigma \\
\text{Symbolic Heaps } \phi & ::= \Pi \wedge \Sigma \wedge \Lambda \\
\text{Definition Clauses } \Phi & ::= \exists \mathbf{x}.\phi \\
\text{Definition Bodies } \Psi & ::= \Phi \mid \Psi \vee \Psi \\
\text{Inductive Definitions} & \quad \texttt{pred} \;\; P(\mathbf{x}) \equiv \Psi
\end{array}
$$

Fig. 1. Syntax of SLA1

The language of SLA1 is defined in Figure 1. We assume first-order variables Vars $::= x, y, v, \ldots$ and inductive predicate symbols $P ::= P_1, P_2, \ldots$. We assume variables are implicitly classified into pointer variables and integer variables. NC is a positive number, which specifies the number of elements in a cell.

We often omit Π or Λ when they are true. SLA1 has an inductive definition system, which is a finite set of inductive definitions given by `pred`. The system SLA1 has symbolic heaps $\Pi \wedge \Sigma$ as well as Presburger arithmetic Λ and inductive predicates P.

We assume $*$ is more tightly bound than $\wedge$. We sometimes write $*_k A_k$ for a sequence of separating conjunctions such as $A_1 * A_2 * A_3$. We often write $a_1 a_2$ for $a_1 \times a_2$. We write $\mathrm{FV}(O)$ for the set of free variables in O where O is some syntactic object.

In the following, we illustrate the expressiveness of SLA1 with two examples and use them as running examples throughout the paper.

Example 1 (Sorted Lists). The following predicate `sortll` for sorted lists can be defined in SLA1.

$$
\begin{array}{l}
\texttt{pred} \;\; \texttt{sortll}(x, y, z) \equiv x \mapsto (z, \text{nil}) \wedge y = 1 \\
\qquad \vee \exists x_1 y_1 z_1 . x \mapsto (z, x_1) * \texttt{sortll}(x_1, y_1, z_1) \wedge y = y_1 + 1 \wedge z \leq z_1 .
\end{array}
$$

y and z represent the length and the minimum value of the list respectively.

Example 2 (AVL Trees). The following predicate `avl` for AVL trees can be defined in SLA1.

$$
\begin{array}{l}
\texttt{pred avl}(x, h) \equiv \text{emp} \wedge x = \text{nil} \wedge h = 0 \vee \\
\qquad \exists x_1 x_2 h_1 h_2 . x \mapsto (x_1, x_2) * \texttt{avl}(x_1, h_1) * \texttt{avl}(x_2, h_2) \\
\qquad \wedge h = \max(h_1, h_2) + 1 \wedge -1 \leq h_1 - h_2 \leq 1.
\end{array}
$$

h is the height of the tree.

We call a definition clause a *base case* when it does not contain any inductive predicates, and we call a definition clause an *induction case* when some inductive predicates appear in it.

We write $\Psi[x := t]$ for ordinary capture-avoiding substitution. $\Phi_1[P := \lambda\mathbf{x}.\Phi_2]$ is defined as the definition clause obtained for Φ_1 by replacing every $P(\mathbf{t})$ by $\Phi_2[\mathbf{x} := \mathbf{t}]$ and moving existential quantifiers to the head. We often write $\Phi[P, \ldots, P]$ to explicitly show occurrences of an inductive predicate P. When we use $\Phi_1[P]$, we write $\Phi_1[\lambda\mathbf{x}.\Phi_2]$ for $\Phi_1[P][P := \lambda\mathbf{x}.\Phi_2]$.

For an induction case $\Phi[P]$ with one occurrence of P and $n \geq 0$, we define

$$\Phi^0[P] \equiv P(\mathbf{x}),$$
$$\Phi^{n+1}[P] \equiv \Phi[\lambda\mathbf{x}.\Phi^n[P]].$$

2.2 Semantics

We write $\mathbf{Z}$ for the set of integers. We assume the set Val of values and the set Loc of addresses such that $\mathrm{Val} = \mathbf{Z} \cup \{\mathrm{null}\}$ and $\mathrm{Val} \cap \mathrm{Loc} = \emptyset$. We use

$$\mathrm{Heaps} = \mathrm{Locs} \rightarrow_{fin} (\mathrm{Loc} \cup \mathrm{Val})^{\mathrm{NC}},$$
$$\mathrm{Stores} = \mathrm{Vars} \rightarrow \mathrm{Loc} \cup \mathrm{Val}.$$

We assume a cell will be interpreted by $(\mathrm{Loc} \cup \mathrm{Val})^{\mathrm{NC}}$ and $s(\mathrm{nil}) = \mathrm{null}$. We use s and h by assuming $s \in \mathrm{Stores}$ and $h \in \mathrm{Heaps}$. We also assume that $s(k) = k$ for an integer constant k, and $\times$,+,−,max,min,$\leq$ are interpreted for integers by a usual semantics, and the interpretation $s \models \Lambda$ for an arithmetic formula Λ is defined using the standard model of integers $\mathbf{Z}$.

The semantics $s, h \models \exists\mathbf{z}.\phi$ of this logic is defined in a usual way as follows.
$s \models t_1 = t_2$ if $s(t_1) = s(t_2)$,
$s \models t_1 \neq t_2$ if $s(t_1) \neq s(t_2)$,
$s \models \Pi_1 \wedge \Pi_2$ if $s \models \Pi_1$ and $s \models \Pi_2$,
$s, h \models \mathrm{emp}$ if $\mathrm{Dom}(h) = \emptyset$,
$s, h \models t \mapsto (t_1, \ldots, t_n)$ if $\mathrm{Dom}(h) = \{s(t)\}$ and $h(s(t)) = (s(t_1), \ldots, s(t_n))$,
$s, h \models \Sigma_1 * \Sigma_2$ if $s, h_1 \models \Sigma_1$ and $s, h_2 \models \Sigma_2$ for some $h_1 + h_2 = h$,
$s, h \models P_i^0(\mathbf{t})$ does not hold,
$s, h \models P_i^{k+1}(\mathbf{t})$ if $s, h \models \Phi[P_i := P_i^k](\mathbf{t})$ for some definition clause Φ of P_i,
$s, h \models P_i(\mathbf{t})$ if $s, h \models P_i^m(\mathbf{t})$ for some m,
$s, h \models \Pi \wedge \Sigma \wedge \Lambda$ if $s \models \Pi$ and $s, h \models \Sigma$ and $s \models \Lambda$, and
$s, h \models \exists z\mathbf{z}\phi$ if $s[z := b], h \models \exists\mathbf{z}\phi$ for some $b \in \mathrm{Loc} \cup \{\mathrm{null}\}$ with a pointer variable z and some $b \in Z$ with an integer variable z.

2.3 Undecidability in SLA1

This section shows that without any restrictions on the shape of inductive definitions, the satisfiability is undecidable in SLA1.

Theorem 2.1 *The satisfiability of symbolic heaps is undecidable in SLA1.*

Proof. For any primitive recursive function $f(\mathbf{x})$, there is an inductive predicate F such that for any numbers $\mathbf{n}, m$, $f(\mathbf{n}) = m$ iff $s_0, h_0 \models F(\mathbf{n}, m)$ where s_0 is the dummy store such that $s_0(x) = \text{null}$ for all x, and h_0 is the empty heap such that $\text{Dom}(h_0) = \emptyset$. In this case, we say the inductive predicate F *represents* the primitive recursive function f. We can show it by induction on the definition of f. We will show only the following cases, since they are only interesting cases.

Case 1. Assume f is the successor function. We define

$$F(x, y) \equiv y = x + 1 \wedge \text{emp}.$$

Then $m = n + 1$ iff $s_0, h_0 \models F(n, m)$.

Case 2. Assume a primitive recursive function $f(x, y)$ is defined by

$$f(0, y) = g(y),$$
$$f(x + 1, y) = h(x, y, f(x, y)).$$

By induction hypothesis for g and h we have inductive predicates G and H that represent g and h respectively. We define the inductive predicate F by

$$F(x, y, z) \equiv x = 0 \wedge G(y, z) \wedge \text{emp} \vee \exists x_1.x = x_1 + 1 \wedge F(x_1, y, z_1) * H(x, y, z_1, z).$$

Then F represents f, namely, $f(n, m) = l$ iff $s_0, h_0 \models F(n, m, l)$.

Let $T(x, y, z)$ be Kleene's T predicate, namely, for any numbers n, m, l, $T(n, m, l)$ is true iff the n-th partial recursive function with input m terminates with the computation history coded by l.

Since T is primitive recursive (namely, its characteristic function is a primitive recursive), there is an inductive predicate T' such that $T(n, m, l)$ is true iff $s_0, h_0 \models T'(n, m, l)$.

Hence the n-th partial recursive function with input m terminates iff $T'(n, m, x)$ is satisfiable in SLA1. Hence the satisfiability in SLA1 would solve the halting problem if the satisfiability in SLA1 were decidable. Consequently the satisfiability in SLA1 is undecidable. □

3 Presburger Arithmetic with Inductive Definitions

In this section, we define the system PI of Presburger arithmetic with positive inductive definitions. The truth in this system is undecidable. We will use this system as our starting point for constructing a decidable subsystem.

3.1 Presburger Arithmetic with Positive Inductive Definitions

Definition 3.1 (System PI) We assume the same first-order variables, the same inductive predicate symbols, the same integer constants, the same arithmetical terms, and the same arithmetical formulas as those of SLA1 presented in Figure 1. For PI, we define the following.

$\phi ::= \Lambda \mid P(\mathbf{a}) \mid \phi \wedge \phi$.

Formulas $\Phi ::= \exists \mathbf{x}.\phi$.

Definition Bodies $\Psi ::= \Phi \mid \Psi \vee \Psi$.

Inductive Definitions $\texttt{pred}\ \ P_i(\mathbf{x}) \equiv \Psi$.

a is interpreted in $\mathbf{Z}$. We define the truth of Λ by the standard model of integers. We interpret an inductive predicate by the least fixed point in a usual way.

The truth of formulas in this system is undecidable for the following reason. We can define multiplication as follows:

$$\texttt{pred}\ \ P(x,y,z) \equiv x = 0 \wedge z = 0 \vee \exists x_1 z_1 . x = x_1 + 1 \wedge P(x_1, y, z_1) \wedge z = z_1 + y.$$

Then $P(x,y,z)$ is true iff $x \times y = z$ is true. Since Presburger arithmetic with multiplication is equivalent to Peano arithmetic, the truth of this system is undecidable.

3.2 Decidable subsystem DPI

We define a subsystem DPI of Presburger arithmetic with inductive definitions. The idea is that we impose some restrictions on the inductive definitions so that every inductive predicate defines some eventually periodic set. Since the decidability proof of Presburger arithmetic relies on the fact that a definable set is exactly an eventually periodic set, this restriction enables us to use the same proof idea for its extension with inductive definitions.

We explain our ideas of restrictions. (1) We assume we have only single induction (namely we do not use mutual induction). Moreover we assume we have at most one induction case. These restrictions enable us to compute the inductive predicates by iteration of the induction case to the base case. (2) When we have more than one arguments of inductive predicates, the i-th argument uses only the i-th arguments of recursive calls. For example, when the induction case of $P(x,y)$ has recursive calls $P(x_1,y_1)$ and $P(x_2,y_2)$, then x is computed by using only x_1 and x_2, and y is computed by using only y_1 and y_2. (3) We assume the induction case has some shape like $\exists x_1(x = x_1 + c \wedge P(x_1))$. In this case, by letting Q be the set represented by the base case, P represents the set $\{x + nc \mid x \in Q, n \in N\}$, which is eventually periodic. We assume this shape of induction case for some argument, for example, the j-th argument. (4) For the other arguments (the i-th argument where $i \neq j$), we assume we reach the fixed point by applying the induction case once. For example, if the induction case for $P(x)$ is $\exists x_1(x \geq x_1 \wedge P(x_1))$, this restriction is satisfied.

Definition 3.2 (System DPI) The language of DPI is the same as that of PI except inductive definitions. The inductive definitions of DPI are defined as those of PI with the following restriction: every inductive definition has the shape

$$\texttt{pred}\ \ P(\mathbf{x}) \equiv \Lambda, \qquad \text{or} \qquad \texttt{pred}\ \ P(\mathbf{x}) \equiv \bigwedge_{1 \leq i \leq m} \Lambda_{0,i} \vee \exists \mathbf{z}. \bigwedge_{1 \leq i \leq m} \Lambda_i \wedge \bigwedge_{1 \leq l \leq L} P(\mathbf{z}^l)$$

where m is the arity of P, $\mathrm{FV}(\Lambda_{0,i}) \subseteq \{x_i\}$, $\mathbf{z} \supseteq \mathbf{z}^l$, there is j such that Λ_i is either of $x_i = f(\mathbf{z}_i)$, $x_i \geq f(\mathbf{z}_i)$, or $x_i \leq f(\mathbf{z}_i)$ for all $i \neq j$, and Λ_j is either of the following:

(1) $x_j = f(\mathbf{z}_j) + c \wedge \Lambda'$,
(2) $x_j \geq f(\mathbf{z}_j) + c \wedge \Lambda'$,

(3) $x_j \leq f(\mathbf{z}_j) + c \wedge \Lambda'$,

(4) a conjunction of the following forms with some integer constant $n > 0$:

$$\Lambda', \qquad nx_j = f(\mathbf{z}_j), \qquad nx_j \geq f(\mathbf{z}_j), \qquad \text{or } nx_j \leq f(\mathbf{z}_j),$$

where c is some integer constant, $\mathbf{z}_j$ is $z_j^1, \ldots, z_j^L$, Λ' is an arithmetical formula such that $\mathrm{FV}(\Lambda') \subseteq \mathbf{z}_j$ and $\Lambda'[\mathbf{z}_j := z]$ is true for any z, $f(\mathbf{z}_j)$ is a combination of $z_j^1, \ldots, z_j^L$ with $\max, \min$, defined by

$$f(\mathbf{z}_j) ::= z_j^l \mid \max(f(\mathbf{z}_j), f(\mathbf{z}_j)) \mid \min(f(\mathbf{z}_j), f(\mathbf{z}_j)),$$

and f's may be different from each other in the conjunction of (4).

Note that in DPI, each inductive definition has at most one induction case, and mutual inductive definitions are not allowed.

Example 3 (Arithmetical Part of Sorted List Predicate). Let $\texttt{sortll}^N$ be an inductive predicate symbol. The arithmetical part $\texttt{sortll}^N$ of the predicate `sortll` is inductively defined by

$$\texttt{pred sortll}^N(y, z) \equiv y = 1 \vee \exists y_1 z_1 . \texttt{sortll}^N(y_1, z_1) \wedge y = y_1 + 1 \wedge z \leq z_1.$$

Example 4 (Arithmetical Part of AVL Tree Predicate). Let $\texttt{avl}^N$ be an inductive predicate symbol. The arithmetical part $\texttt{avl}^N$ of the predicate `avl` is inductively defined by

$$\begin{aligned}\texttt{pred avl}^N(h) \equiv h = 0 \vee \exists h_1 h_2 . \texttt{avl}^N(h_1) \wedge \texttt{avl}^N(h_2) \\ \wedge h = \max(h_1, h_2) + 1 \wedge -1 \leq h_1 - h_2 \leq 1.\end{aligned}$$

Definition 3.3 A set S of integers is defined to be *eventually periodic* if there are some $M \geq 0, p_1, p_2 > 0$ such that $n \in S$ iff $n + p_1 \in S$ for all $n \geq M$, and $n \in S$ iff $n - p_2 \in S$ for all $n \leq -M$. Then we call the set (M, p_1, p_2)-periodic.

Lemma 3.4 *If $S \neq \emptyset$ is (M, p_1, p_2)-periodic, then $\{x \mid nx = y,\ y \in S\}$ is (M, p_1, p_2)-periodic for $n > 0$.*

Proof. Let S' be $\{x \mid nx = y, y \in S\}$. Assume $x \in S'$ and $x \geq M$. There is y such that $nx = y$ and $y \in S$. Since $y = nx \geq x \geq M$ and $n(x + p_1) = nx + np_1 = y + np_1 \in S$, we have $x + p_1 \in S'$.

Assume $x + p_1 \in S'$ and $x \geq M$. There is y such that $n(x + p_1) = y$ and $y \in S$. Since $y - np_1 = nx \geq x \geq M$ and $nx = y - np_1 \in S$, we have $x \in S'$.

Similarly for $x \leq -M$, $x \in S'$ iff $x - p_2 \in S'$.

Hence S' is (M, p_1, p_2)-periodic. □

Theorem 3.5 (Inductive Predicate Elimination) *For every inductive predicate P, there is a formula Λ equivalent to $P(\mathbf{x})$ such that Λ does not contain any inductive predicates.*

Proof. Let

$$\texttt{pred}\ P(\mathbf{x}) \equiv \bigwedge_{1\le i\le m} \Lambda_{0,i} \vee \Phi_1,$$
$$\Phi_1 \equiv \exists \mathbf{z}.\ \bigwedge_{1\le i\le m} \Lambda_i \wedge \bigwedge_l P(\mathbf{z}^l).$$

Let $\mathbf{x}$ be $(x_1,\ldots,x_m)$, S be $\{\mathbf{x} \mid P(\mathbf{x})\}$, Q be $\{\mathbf{x} \mid \bigwedge_i \Lambda_{0,i}\}$, S_i be $\{x_i \mid P(\mathbf{x})\}$, and Q_i be $\{x_i \mid \Lambda_{0,i}\}$. We have $Q = Q_1 \times \ldots \times Q_m$.

Since $\{f(\mathbf{z}_i) \mid \bigwedge_l z_i^l \in X\} = X$, we have the following facts: $\{x_i \mid x_i = f(\mathbf{z}_i) \wedge \bigwedge_l z_i^l \in X\} = X$, $\{x_i \mid x_i \ge f(\mathbf{z}_i) \wedge \bigwedge_l z_i^l \in X\} = X_+$, and $\{x_i \mid x_i \le f(\mathbf{z}_i) \wedge \bigwedge_l z_i^l \in X\} = X_-$, where X_+ is $\emptyset$ if $X = \emptyset$, $\{z \mid z \ge \min X\}$ if $\min X$ exists, $\mathbf{Z}$ otherwise, and X_- is $\emptyset$ if $X = \emptyset$, $\{z \mid z \le \max X\}$ if $\max X$ exists, $\mathbf{Z}$ otherwise.

Define $F : p(\mathbf{Z}^m) \to p(\mathbf{Z}^m)$ by $F(X)=Q \cup \{\mathbf{x} \mid \Phi_1[\lambda\mathbf{x}.(\mathbf{x} \in X)]\}$. Then $S=\bigcup_{n=0}^{\infty} F^n(\emptyset)$. We define $F_j{:}p(\mathbf{Z})\to p(\mathbf{Z})$ by $F_j(X)=Q_j\cup\{x_j \mid \Lambda_j \wedge \bigwedge_l z_j^l \in X\}$.

By the above facts, the i-th element of $F^n(\emptyset)$ is Q_i, Q_{i+}, or Q_{i-} for all $i \ne j$ and $n > 1$. Hence $S = S_1 \times \ldots \times S_m$ where $S_i = Q_i, Q_{i+}$, or Q_{i-} for all $i \ne j$, and $S_j = \bigcup_{n=0}^{\infty} F_j^n(\emptyset)$, since the j-th value x_j depends on only the previous j-th values $\mathbf{z}_j$ in the definition of P.

It is known that a set definable in Presburger arithmetic is exactly an eventually periodic set [6]. Hence each Q_j is eventually periodic. Let Q_j be (M, p_1, p_2)-periodic.

We show S_j is eventually periodic by considering cases by the cases (1) to (4) in the restriction 2 according to the shape of Λ_j.

We have the fact (a) : $\{f(\mathbf{z}_j) \mid \bigwedge_l z_j^l \in X \wedge \Lambda'\} = X$. We can show it as follows: take a in the righthand side. By taking z_j^l to be a, since $\Lambda'[\mathbf{z}_j := a]$ is true and $f(\mathbf{z}_j) = a$, we have a is in the lefthand side.

The case (1). Λ_j is $x_j = f(\mathbf{z}_j)+c\wedge\Lambda'$. We have $F_j(X) = Q_j\cup\{x+c \mid x \in X\}$ and $S_j = \{x + nc \mid x \in Q_j, n \in N\}$. Let R_i be $\{x \in Q_j \mid x \equiv i \pmod c\}$. Assume $c > 0$. Define R'_i as $\emptyset$ if $R_i = \emptyset$, $\{k_i + nc \mid n \in N\}$ if R_i has the minimum k_i, and $\{x \mid x \equiv i \pmod c\}$ otherwise. Then $S_j = \bigcup_{0\le i<c} R'_i$. Then S_j is (M', c, p_2)-periodic where $M' = \max_{0\le i<c}(M, |k_i|)$. Similarly, if $c < 0$ then S_j is $(M', p_1, -c)$-periodic where $M' = \max_{0\le i<c}\{M, |k_i| \mid R_i \text{ has the maximum } k_i\}$. If $c = 0$, then $S_j = Q_j$ and S_j is (M, p_1, p_2)-periodic.

The case (2). Λ_j is $x_j \ge f(\mathbf{z}_j) + c \wedge \Lambda'$. We have $F_j(X) = Q_j \cup \{x \mid x \ge x' + c, x' \in X\}$. If $Q_j = \emptyset$, then $S_j = \emptyset$. Assume $Q_j \ne \emptyset$. S_j is $\mathbf{Z}$ if Q_j does not have any minimum. Assume Q_j has the minimum. If $c < 0$ then $Q_j = \mathbf{Z}$. If $c \ge 0$ then S_j is $Q_j \cup \{x \mid x \ge \min Q_j + c\}$. Hence either is eventually periodic.

The case (3). Λ_j is $x_j \le f(\mathbf{z}_j)+c\wedge\Lambda'$. This case is shown in a similar manner to the case (2).

The case (4). Λ_j is a conjunction of the forms Λ', $nx_j = f_1(\mathbf{z}_j)$, $nx_j \geq f_2(\mathbf{z}_j)$, and $nx_j \leq f_3(\mathbf{z}_j)$. First we show S_j is (M, p_1, p_2)-periodic when Λ_j is either $nx_j = f(\mathbf{z}_j) \wedge \Lambda'$, $nx_j \geq f(\mathbf{z}_j) \wedge \Lambda'$, or $nx_j \leq f(\mathbf{z}_j) \wedge \Lambda'$.

Case (4).1. Λ_j is $nx_j = f(\mathbf{z}_j) \wedge \Lambda'$. If $Q_j = \emptyset$, then $S_j = \emptyset$ and it is (M, p_1, p_2)-periodic. If $Q_j \neq \emptyset$, by Lemma 3.4 and the fact (a), S_j is (M, p_1, p_2)-periodic.

Case (4).2. Λ_j is $nx_j \geq f(\mathbf{z}_j) \wedge \Lambda'$. If $Q_j = \emptyset$, then $S_j = \emptyset$. Assume $Q_j \neq \emptyset$. If Q_j does not have any minimum, $S_j = \mathbf{Z}$. Assume Q_j has the minimum k. If X has the minimum, $F_j(X) = Q_j \cup \{x \mid x \geq \lceil (\min X)/n \rceil\}$. By this, S_j is $\{x \mid x \geq k\}$ if $k > 0$ and $n = 1$, $\{x \mid x > 0\}$ if $k > 0$ and $n > 1$, $\{x \mid x \geq 0\}$ if $k = 0$, $\{x \mid x \geq k\}$ if $k < 0$. Moreover $k \geq -M$. Hence either is (M, p_1, p_2)-periodic.

Case (4).3. Λ_j is $nx_j \leq f(\mathbf{z}_j) \wedge \Lambda'$. In a similar way to the case (4).2, we can show S_j is (M, p_1, p_2)-periodic.

We have shown S_j is (M, p_1, p_2)-periodic when Λ_j is either $nx_j = f(\mathbf{z}_j) \wedge \Lambda'$, $nx_j \geq f(\mathbf{z}_j) \wedge \Lambda'$, or $nx_j \leq f(\mathbf{z}_j) \wedge \Lambda'$.

We show the general case when Λ_j is $\bigwedge_k \Lambda'_k$ where Λ'_k is either $nx_j = f_1(\mathbf{z}_j) \wedge \Lambda'$, $nx_j \geq f_2(\mathbf{z}_j) \wedge \Lambda'$, or $nx_j \leq f_3(\mathbf{z}_j) \wedge \Lambda'$. Let $F'_k(X) = Q_j \cup \{x_j \mid \Lambda'_k \wedge \bigwedge_l z_j^l \in X\}$ and S'_k be the least fixed point of F'_k. Since the least fixed point of $\bigcap_k F'_k(X)$ is the intersection of the least fixed points of $F'_k(X)$ for all k, we have $S_j = \bigcap_k S'_k$. We have the fact: if X_i is (M, p_1, p_2)-periodic for all i, then $\bigcap_i X_i$ is also (M, p_1, p_2)-periodic. By this fact, since S'_k is (M, p_1, p_2)-periodic for all k, S_j is (M, p_1, p_2)-periodic.

We have shown that the truth of $P(x_1, \ldots, x_m)$ is equivalent to $\bigwedge_{1 \leq i \leq m} (x_i \in Q_i) \vee x_j \in (S_j - Q_j) \wedge \bigwedge_{1 \leq i \leq m, j \neq i} (x_i \in S_i)$ and each of Q_i, Q_j, S_i, S_j is eventually periodic. Since an eventually periodic set is definable by a Presburger formula [6], we have a formula that does not contain any inductive predicates and is equivalent to $P(x_1, \ldots, x_m)$. □

The decision procedure for DPI is obtained by computing the above M, p_1, p_2 and S_j according to the decidability proof.

4 Decidable Subsystem SLA2

In this section we define a decidable subsystem SLA2 of the system SLA1.

4.1 Syntax of SLA2

First we define a numeric projection from SLA1 to PI. Next we define a spatial projection from SLA1 to the symbolic-heap system presented in [4] (we call it SL).

We define the system SL. The difference from the system in [4] is that the number of elements in a cell is fixed to be NC in SL and SL has only single induction (namely, it does not have mutual induction).

Definition 4.1 (System SL) Pointer terms $t ::= x \mid \text{nil}$.
Pure formulas $\Pi ::= \text{true} \mid \text{false} \mid t = t \mid t \neq t \mid \Pi \wedge \Pi$.
Spatial formulas $\Sigma ::= \text{emp} \mid t \mapsto (t_1, \ldots, t_{\text{NC}}) \mid P(\mathbf{t}) \mid \Sigma * \Sigma$.
Symbolic Heaps $\phi ::= \Pi \wedge \Sigma$.
Definition Clauses $\Phi ::= \exists \mathbf{x}.\phi$.
Definition Bodies $\Psi ::= \Phi \mid \Psi \vee \Psi$.
Inductive Definitions $\texttt{pred}\ P(\mathbf{x}) \equiv \Psi$.

We assume inductive predicate symbols P^N and P^S for each inductive predicate symbol P. We write $\mathbf{x}^N$ and $\mathbf{x}^S$ for the integer variables and the pointer variables among the variables $\mathbf{x}$ respectively.

Definition 4.2 (Projection) The numeric projection $(\Sigma)^N$ is defined by $(\text{emp})^N \equiv (t \mapsto (\mathbf{u}))^N \equiv \text{true}$, $(P(\mathbf{t}, \mathbf{a}))^N \equiv P^N(\mathbf{a})$, and $(\Sigma_1 * \Sigma_2)^N \equiv (\Sigma_1)^N \wedge (\Sigma_2)^N$.
$(\phi)^N$ is defined by $(\Pi \wedge \Sigma \wedge \Lambda)^N \equiv (\Sigma)^N \wedge \Lambda$.
$(\Phi)^N$ is defined by $(\exists \mathbf{x}.\phi)^N \equiv \exists \mathbf{x}^N.(\phi)^N$.
$(\Psi)^N$ is defined by $(\Psi_1 \vee \Psi_2)^N \equiv (\Psi_1)^N \vee (\Psi_2)^N$.
The spatial projection $(u)^S$ is defined by $(t)^S \equiv t$ and $(a)^S \equiv \text{nil}$.
The spatial projection $(\Sigma)^S$ is defined by $(\text{emp})^S \equiv \text{emp}$, $(t \mapsto (\mathbf{u}))^S \equiv t \mapsto ((\mathbf{u})^S)$, $(P(\mathbf{t}, \mathbf{a}))^S \equiv P^S(\mathbf{t})$, and $(\Sigma_1 * \Sigma_2)^S \equiv (\Sigma_1)^S * (\Sigma_2)^S$.
$(\phi)^S$ is defined by $(\Pi \wedge \Sigma \wedge \Lambda)^S \equiv \Pi \wedge (\Sigma)^S$.
$(\Phi)^S$ is defined by $(\exists \mathbf{x}.\phi)^S \equiv \exists \mathbf{x}^S.(\phi)^S$.
$(\Psi)^S$ is defined by $(\Psi_1 \vee \Psi_2)^S \equiv (\Psi_1)^S \vee (\Psi_2)^S$.

We give the spatial projections of the predicates `sortll` and `avl` in Section 2.1. Their numerical projections are already given in Section 3.

Example 5 (Spatial Part of Sorted Lists).

$$\texttt{pred sortll}^S(x) \equiv x \mapsto (\text{nil}, \text{nil}) \ \vee \ \exists x_1.x \mapsto (\text{nil}, x_1) * \texttt{sortll}^S(x_1).$$

Example 6 (Spatial Part of AVL Trees).

$$\texttt{pred avl}^S(x) \equiv \text{emp} \wedge x = \text{nil} \vee \exists x_1 x_2.x \mapsto (x_1, x_2) * \texttt{avl}^S(x_1) * \texttt{avl}^S(x_2).$$

Definition 4.3 (System SLA2) The language of SLA2 is the same as that of SLA1 except inductive definitions. The inductive definitions of SLA2 are those of SLA1 with the following two restrictions. Let the inductive definition $\texttt{pred}\ P(\mathbf{x}) \equiv \Psi$.

(1) Its numeric projection $\texttt{pred}\ P^N(\mathbf{x}^N) \equiv (\Psi)^N$ is an inductive definition of DPI.

(2) If the induction case has more than one occurrences of P, then the spatial projection of Ψ has the following form

$$(\Psi)^S \equiv \psi_0 \vee \exists \mathbf{z}.\Pi \wedge *_{k \in K} w_k \mapsto (\mathbf{t}^k) * *_l P^S(\mathbf{z}^l),$$

where ψ_0 is a disjunction of the base cases, $\mathbf{z}^l \subseteq \mathbf{z}$, the variables in $(\mathbf{z}^l)_l$ are mutually distinct and do not appear in Π or $\{w_k \mid k \in K\}$.

We explain the condition (2). Let the induction case with more than one occurrences of P be Φ. It says the argument $\mathbf{z}^l$ of P are distinct existential variables and they do not appear in Π or $\{w_k \mid k \in K\}$. Hence for the existential variables, we can choose arbitrary values such that $P^S(\mathbf{x})$ is satisfiable by taking $\mathbf{x}$ to these values. In particular, we can choose some values such that the base case is true. Hence $(\Phi)^{S(T')}$ is satisfiable for some unfolding tree T' of height 1, when $(\Phi)^{S(T)}$ is satisfiable for some unfolding tree T of height ≥ 1. Consequently The restriction (2) guarantees that if an unfolding tree T of P is not linear, then the base pair that interprets P unfolded by T is determined to be two possibilities depending on the height 0 or ≥ 1 of T.

Example 7 (Sorted Lists). The predicate `sortll` in Section 2.1 can be defined in SLA2 as its spatial projection $\texttt{sortll}^S(x)$ satisfies the restriction (2) (the condition trivially holds since it does not apply) and its numeric projection $\texttt{sortll}^N(x)$ is in DPI.

Example 8 (AVL Trees). The predicate `avl` in Section 2.1 can be defined in SLA2 as its spatial projection $\texttt{avl}^S(x)$ satisfies the restriction (2) and its numeric projection $\texttt{avl}^N(x)$ is in DPI.

4.2 Unfolding Tree

This section defines unfolding trees, introduced in [7], in our notation. We use unfolding trees to synchronize the spatial part and the numeric part of a given symbolic heap in the proof of the decidability for SLA2. In general we can define unfolding trees for any logical system with inductive definitions including SLA1, SLA2, DPI, and SL.

Definition 4.4 (Unfolding Tree) Suppose the inductive definition of P

$$\texttt{pred}\ P(\mathbf{x}) \equiv \bigvee_{1 \leq i \leq I} \Phi_i \vee \Phi[P, \ldots, P]$$

where Φ_i is a base case and the induction case $\Phi[P, \ldots, P]$ contains n occurrences of P. An unfolding tree T of P is defined by $T ::= i \mid (T_1, \ldots, T_n)$ where $1 \leq i \leq I$.

An unfolding tree T of P specifies how we unfold the inductive predicate P. It is described as follows.

Definition 4.5 Suppose $\texttt{pred}\ P(\mathbf{x}) \equiv \bigvee_{1 \leq i \leq I} \Phi_i \vee \Phi[P, \ldots, P]$.

For an unfolding tree T of P, $P^{(T)}$ is defined by:

$$P^{(i)} \equiv \lambda \mathbf{x}.\Phi_i,$$
$$P^{((T_1, \ldots, T_n))} \equiv \lambda \mathbf{x}.\Phi[P^{(T_1)}, \ldots, P^{(T_n)}].$$

We write $T(i, k)$ for $(\ldots(i)\ldots)$ where $\ldots$ denotes k parentheses. $T(i, k)$ is the unfolding tree of length k with the leaf i and $n = 1$.

The next proposition guarantees the synchronization of the spatial and numeric projections by an unfolding tree.

Proposition 4.6 *$s, h \models P^{(T)}(\mathbf{t}, \mathbf{a})$ in SLA2 for some h iff $s, h \models P^{S(T)}(\mathbf{t})$ in SL for some h and $s \models P^{N(T)}(\mathbf{a})$ in DPI.*

Proof. By induction on T. □

The next proposition says the truth of P is that of P unfolded by some unfolding tree.

Proposition 4.7 *$s, h \models P^{(T)}(\mathbf{x})$ for some T iff $s, h \models P(\mathbf{x})$.*

Proof. The only if part is proved by $P^k(\mathbf{x})$ where k is the height of T. The if part is shown by the definition of the truth. □

4.3 Base Pairs

In this section, we define base pairs adopted from [4]. We use base pairs to characterize unfolding trees T such that $P^{(T)}(\mathbf{x})$ is satisfiable. For this purpose, we define a base pair (B, Π) for $P^{(T)}(\mathbf{t})$ so that (B, Π) is satisfiable iff $P^{(T)}(\mathbf{t})$ is satisfiable.

Compared with the base pairs in [4], [4] interprets a symbolic heap with inductive predicates by a set of base pairs. On the other hand we will interpret a symbolic heap $\check{\phi}$ without any inductive predicates by a single base pair. A single base pair can work since $\check{\phi}$ does not contain disjunction.

Since we want the set of equivalence classes of base pairs to be finite, we have some notational difference with [4]: While [4] uses a multiset V for a base pair (V, Π), we use a set B for a base pair (B, Π). For free variables, [4] uses $\lambda\mathbf{x}.(V, \Pi)$, but we implicitly use $\mathbf{x}$. (V, Π) is satisfiable when Π is satisfiable in [4], but our (B, Π) is satisfiable when $\Pi \wedge \otimes B$ is satisfiable.

Definition 4.8 (Base Pair) We call (B, Π) a *base pair* when Π is a pure formula, and B is a set of pointer variables. For a pure formula Π, Π is defined to be consistent if $\Pi \not\vdash$ false.

For notation of multisets, we write $\{e[x] \mid_M x \in_M V \wedge \ldots\}$ a multiset counting repetition of $e[x]$ where each x is taken from the multiset V counting repetition.

We define $[t]_\Pi$ as $\{u \mid \Pi \vdash u = t\}$. It is an equivalence class containing t by the equality of Π.

For a multiset V of terms and a pure formula Π, we define V/Π as $\{[t]_\Pi \mid_M t \in_M V\}$. It is a multiset of equivalence classes by the equality of Π. V/Π is called sound when V/Π does not have any duplicates and does not have any equivalence class containing nil.

Definition 4.9 (Satisfiable Base Pair) A base pair (B, Π) is defined to be *satisfiable* if Π is consistent and B/Π is sound.

We define $\bot$ as $(\emptyset, \mathrm{false})$. For a multiset V of terms and a pure formula Π, we define $\overline{(V, \Pi)}$ as (V, Π) if Π is consistent and B/Π is sound. Otherwise we define it as $\bot$.

For a multiset V of terms, we define the multiset $V[\mathbf{x} := \mathbf{t}]_M$ by replacing $\mathbf{x}$ by $\mathbf{t}$ counting repetition. We use $\uplus$ for the multiset union.

We define

$$\Pi_1 \wedge (V, \Pi) = (V, \Pi_1 \wedge \Pi),$$
$$(V, \Pi)[\mathbf{x} := \mathbf{t}]_M = (V[\mathbf{x} := \mathbf{t}]_M, \Pi[\mathbf{x} := \mathbf{t}]),$$
$$(V_1, \Pi_1) * (V_2, \Pi_2) = (V_1 \uplus V_2, \Pi_1 \wedge \Pi_2).$$

Definition 4.10 We define $(V_1, \Pi_1) \simeq (V_2, \Pi_2)$ by $\Pi_1 \leftrightarrow \Pi_2$ and $V_1/\Pi_1 = V_2/\Pi_2$. Then we say (V_1, Π_1) and (V_2, Π_2) are equivalent.

We write $\Pi \leftrightarrow_{\mathbf{x}} \Pi'$ when $\Pi \to \Pi_0$ iff $\Pi' \to \Pi_0$ for every Π_0 such that $\mathrm{FV}(\Pi_0) \subseteq \mathbf{x}$ and Π_0 is either true, false, $t_1 = t_2$, or $t_1 \neq t_2$. We define $\Pi - \mathbf{x}$ as some Π' such that $\mathrm{FV}(\Pi') \subseteq FV(\Pi) - \mathbf{x}$ and $\Pi' \leftrightarrow_{\mathbf{x}} \Pi$.

For a set B of variables, we define $\otimes B$ as

$$\bigwedge\{t \neq u \mid t, u \in B, t \not\equiv u\} \wedge \bigwedge\{t \neq \mathrm{nil} \mid t \in B\}.$$

We define a language that contains $P^{(T)}(\mathbf{t})$. Since $P^{(T)}(\mathbf{t})$ is obtained from $P(\mathbf{t})$ by unfolding inductive predicates, it does not contain any inductive predicates but it may have nested existential quantifiers. We use the name with $\check{}$ for the corresponding syntactical category.

Spatial formulas $\check{\Sigma} ::= \mathrm{emp} \mid t \mapsto (t_1, \ldots, t_{\mathrm{NC}}) \mid \exists \mathbf{x}.\check{\phi} \mid \check{\Sigma} * \check{\Sigma}$.

Symbolic Heaps $\check{\phi} ::= \Pi \wedge \check{\Sigma}$.

We define $[\![\ ]\!]$ for this language. We define $[\![\check{\Sigma}]\!]$ by:

$$[\![\mathrm{emp}]\!] = (\emptyset, \mathrm{true}),$$
$$[\![t \mapsto (t_1, \ldots, t_{\mathrm{NC}})]\!] = (\{t\}, \mathrm{true}),$$
$$[\![\exists \mathbf{x}.\check{\phi}]\!] = \overline{(B - \mathbf{x}, (\Pi \wedge \otimes B) - \mathbf{x})},$$
$$[\![\check{\Sigma}_1 * \check{\Sigma}_2]\!] = \overline{[\![\check{\Sigma}_1]\!] * [\![\check{\Sigma}_2]\!]}.$$

We define $[\![\check{\phi}]\!]$ by $[\![\Pi \wedge \check{\Sigma}]\!] = \overline{\Pi \wedge [\![\check{\Sigma}]\!]}$.

Note that $[\![\check{\phi}]\!]$ is (B, Π) such that B is a set of variables, $\mathrm{FV}(B), \mathrm{FV}(\Pi) \subseteq \mathrm{FV}(\check{\phi})$, and (B, Π) is satisfiable if $(B, \Pi) \neq \bot$.

For a set $\mathbf{x}$ of variables, we write $\beta_{\mathbf{x}}$ for the set of equivalence classes of base pairs with its free variables in $\mathbf{x}$ by $\simeq$. We will often write (B, Π) for the equivalence class containing (B, Π). For example, we will write $(B, \Pi) \in X \subseteq \beta_{\mathbf{x}}$ when the equivalence class containing (B, Π) is in X. Note that $[\![\check{\phi}]\!] \in \beta_{\mathrm{FV}(\check{\phi})}$.

The next lemma is useful to calculate $[\![\ ]\!]$ by substitution. We write $\check{\phi}[\check{\phi}_1]$ to explicitly display an occurrence of $\check{\phi}_1$ in $\check{\phi}$.

Lemma 4.11 *(1)* $[\![\check{\phi}[\mathbf{x} := \mathbf{t}]]\!] \simeq \overline{[\![\check{\phi}]\!][\mathbf{x} := \mathbf{t}]_M}$.

(2) If $[\![\check{\phi}_1]\!] \simeq [\![\check{\phi}_2]\!]$*, then* $[\![\check{\phi}[\check{\phi}_1[\mathbf{x} := \mathbf{t}]]]\!] \simeq [\![\check{\phi}[\check{\phi}_2[\mathbf{x} := \mathbf{t}]]]\!]$.

Proof. (1) By induction on $\check{\phi}$.
(2) By induction on $\check{\phi}$ and (1). □

We have the following lemma similar to Lemmas 3.7 and 3.8 in [4].

Lemma 4.12 *(1) If $[\![\check{\phi}]\!] = (B, \Pi)$ and $s \models \Pi \wedge \otimes B$, then there is h such that $s, h \models \check{\phi}$ and $s(B) \subseteq Dom(h)$, and moreover we can freely choose values in $Dom(h) - s(B)$.*
(2) If $s, h \models \check{\phi}$ and $[\![\check{\phi}]\!] = (B, \Pi)$, then $s \models \Pi \wedge \otimes B$ and $s(B) \subseteq Dom(h)$.

Proof. Each of (1) and (2) is proved by induction on $\check{\phi}$. □

The following proposition is an instance of the theorem 3.9 in [4] in our terms. It says a base pair characterizes the satisfiability.

Proposition 4.13 *$\check{\phi}$ is satisfiable iff $[\![\check{\phi}]\!]$ is satisfiable.*

Proof. By Lemma 4.12 (1)(2). □

4.4 Decidability in SLA2

This section provides the decision procedure of the satisfiability in SLA2 and proves its correctness.

Our ideas of our decision algorithm are as follows. (1) We list up unfolding trees T such that $[\![P^{S(T)}(\mathbf{t})]\!]$ (the spatial part unfolded by T) is satisfiable. (2) The set $\{T \mid [\![P^{S(T)}(\mathbf{t})]\!] \text{ satisfiable}\}$ has a periodic structure for the following reason. In the case where the induction case has only one occurrence of the inductive predicate, since the set of base pairs is finite, we have a periodic structure. In the case where the induction case has more than one occurrences of the inductive predicate, the satisfiability for T of height ≥ 1 is the same as that for T of height 1 by the restriction (2) of SLA2. (3) In the case where the induction case has only one occurrence of the inductive predicate, according to the set $X = \{T \mid [\![P_1^{S(T)}(\mathbf{t})]\!] \text{ satisfiable}\}$, we make new inductive definitions for inductive predicates $P_{1,i}$ so that for any unfolding tree T', $P_{1,i}^{(T')}$ is equivalent to $P_1^{N(T)}$ for some $T \in X$. (4) We decide the numeric part of these inductive predicates $P_{1,i}$ by the decidability of DPI.

We will use the next two lemmas to define our decision procedure for SLA2, which can be straightforwardly shown.

Lemma 4.14 *For an induction case $\Phi[P]$ with one occurrence of P in DPI, $\Phi^n[P]$ is an induction case of P in DPI for $n > 0$.*

We write $\widetilde{T}$ to the set of leaves of an unfolding tree T.

Lemma 4.15 *Assume* `pred` $P_2(x) \equiv \bigvee_{1 \leq i \leq I_2} \Phi_{2,i} \vee \Phi_2[P_2, P_2]$
in SLA2. If $\Phi_{2,i}^S$ is satisfiable for all $i \in \widetilde{(T_1, T_2)} \cup \widetilde{(T_3, T_4)}$, then $[\![P_2^{S((T_1,T_2))}(x)]\!] = [\![P_2^{S((T_3,T_4))}(x)]\!]$.

The next theorem is one of our main results.

Theorem 4.16 *The satisfiability of symbolic heaps is decidable in SLA2.*

Proof. For simplicity, we discuss only the case when only P_1 and P_2 are inductive predicates, the induction case of P_1 has one occurrence of P_1, and the induction case of P_2 has two occurrences of P_2. For simplicity, we also assume P_1 and P_2 take one pointer variable and one integer variable.

The decision procedure for the satisfiability of a given symbolic heap in SLA2 is presented in Algorithm 1, where the input is the following symbolic heap:

$$\phi \equiv \Pi \wedge *_{k\in K} w_k \mapsto (\mathbf{u}^k) * *_{1\leq i\leq I} P_1(t_{1,i}, a_{1,i}) * *_{1\leq j\leq J} P_2(t_{2,j}, a_{2,j}) \wedge \Lambda,$$

with the inductive definitions

$$\begin{aligned}
\texttt{pred}\ P_1(x,y) &\equiv \bigvee_{1\leq i\leq I_1} \Phi_{1,i} \vee \Phi_1[P_1], \\
\texttt{pred}\ P_2(x,y) &\equiv \bigvee_{1\leq i\leq I_2} \Phi_{2,i} \vee \Phi_2[P_2, P_2],
\end{aligned}$$

where x is a pointer variable and y is an integer variable.

Note. In the algorithm, for $1{\leq}i{\leq}I$, we use (l_i, n_i) to represent n_i-times application of the induction case to the l_i-th base case. For $1{\leq}j{\leq}J$, we use $1{\leq}m_j{\leq}I_2$ to represent the m_j-th base case and (r, r) to represent the unfolding tree of height 1 with the r-th base case.

First we will show that there exist $p_i < q_i$ in the step 1. By definition,

$$(\Phi_1^{n+1}[\lambda x.\Phi_{1,i}])^S = (\Phi_1[\lambda x.\Phi_1^n[\lambda x.\Phi_{1,i}]])^S.$$

Hence, by Lemma 4.11 (2), if $[\![\Phi_1^n[\lambda x.\Phi_{1,i}]]\!] \simeq [\![\Phi_1^{n'}[\lambda x.\Phi_{1,i}]]\!]$, then $[\![(\Phi_1^{n+1}[\lambda x.\Phi_{1,i}])^S]\!] \simeq [\![(\Phi_1^{n'+1}[\lambda x.\Phi_{1,i}])^S]\!]$. Since the equivalence class containing $[\![(\Phi_1^n[\lambda x.\Phi_{1,i}])^S]\!]$ is in β_x for $n = 0, 1, 2, \ldots$, and β_x is finite, we have the same occurrences of some equivalence class in the sequence. Hence we have some $n < n'$ such that $[\![(\Phi_1^n[\lambda x.\Phi_{1,i}])^S]\!]$ and $[\![(\Phi_1^{n'}[\lambda x.\Phi_{1,i}])^S]\!]$ are equivalent. We can take p_i, q_i as the least ones among these n, n'.

Next, by the following (1)(2), we will show that the algorithm returns Yes iff ϕ is satisfiable.

(1) We will show that the algorithm returns Yes if ϕ is satisfiable.

Assume ϕ is satisfiable.

Let $\mathbf{x} = x_{1,1} \ldots x_{1,I_1} x_{2,1} \ldots x_{2,I_2}$, $\mathbf{y} = y_{1,1} \ldots y_{1,I_1} y_{2,1} \ldots y_{2,I_2}$, $\mathbf{t} = t_{1,1} \ldots t_{1,I_1} t_{2,1} \ldots t_{2,I_2}$, and $\mathbf{a} = a_{1,1} \ldots a_{1,I_1} a_{2,1} \ldots a_{2,I_2}$. Define the predicate P by

$$P(\mathbf{x}, \mathbf{y}) \equiv \Pi \wedge *_{k\in K} w_k \mapsto (\mathbf{u}^k) * *_{1\leq i\leq I} P_1(x_{1,i}, y_{1,i}) * *_{1\leq j\leq J} P_2(x_{2,j}, y_{2,j}).$$

Since $P(\mathbf{t}, \mathbf{a})$ and ϕ are equivalent, $P(\mathbf{t}, \mathbf{a})$ is satisfiable. By Proposition 4.7 we have some T such that $P^{(T)}(\mathbf{t}, \mathbf{a})$ is satisfiable. By Proposition 4.6, both $P^{S(T)}(\mathbf{t})$ and $P^{N(T)}(\mathbf{a})$ are satisfiable.

Algorithm 1: Decision Procedure for SLA2

input : ϕ

output: Yes or No

Step 1. Compute p_i, q_i for each $1 \leq i \leq I_1$ as follows. Choose i. Compute the sequence $[\![(\Phi_1^n[\lambda x.\Phi_{1,i}])^S]\!]$ for $n = 0, 1, 2, \ldots$. Take the smallest p_i, q_i such that $p_i < q_i$, and the p_i-th occurrence and the q_i-th occurrence are equivalent.
Set I_3 to be $\{i \mid \Phi_{2,i}$ satisfiable$\}$. Set C to be $\{(l_1, n_1, \ldots, l_I, n_I, m_1, \ldots, m_J) \mid 1 \leq l_i \leq I_1, 0 \leq n_i < q_i, 1 \leq m_j \leq I_2 \vee (m_j = (r, r) \wedge r \in I_3)\}$.

Step 2. If C is empty, then return No and stop. Otherwise take some new element in C and go to the next step.

Step 3. Check whether the following formula is satisfiable:

$$[\![\Pi \wedge *_{k\in K} w_k \mapsto ((u^k)^S) * *_{1\leq i\leq I} P_1^{S(T(l_i,n_i))}(t_{1,i}) * *_{1\leq j\leq J} P_2^{S(m_j)}(t_{2,j})]\!].$$

If it is not satisfiable, then go to the step 2 for the next loop.

Step 4. For each $1 \leq i \leq I$, by Lemma 4.14 and Theorem 3.5, define $\Lambda_{1,i}$ as some arithmetical formula equivalent to $P_{1,i}(a_{1,i})$ where $P_{1,i}$ is defined by

$$\texttt{pred}\ \ P_{1,i}(y) \equiv (\Phi_1^{n_i}[\lambda y.\Phi_{1,l_i}])^N \vee (\Phi_1^{q_i-p_i}[P_1])^N[P_1^N := P_{1,i}].$$

For each $1 \leq j \leq J$, by Theorem 3.5, define $\Lambda_{2,j}$ as $(\Phi_{2,m_j})^N[y := a_{2,j}]$ if $1 \leq m_j \leq I_2$, and some arithmetical formula equivalent to $(\Phi_2[P_2, P_2])^N[P_2^N := P_3][y := a_{2,j}]$ where P_3 is defined by

$$\texttt{pred}\ \ P_3(y) \equiv \bigvee_{i\in I_3} \Phi_{2,i}^N \vee (\Phi_2[P_2, P_2])^N$$

if $m_j = (r, r)$.

Step 5. Check if the following formula is satisfiable $\bigwedge_{1\leq i\leq I} \Lambda_{1,i} \wedge \bigwedge_{1\leq j\leq J} \Lambda_{2,j} \wedge \Lambda$
If it is true, then return Yes and stop.

Step 6. Go to the step 2 for the next loop.

Let T be $(T_{1,1}, ..., T_{1,I}, T_{2,1}, ..., T_{2,J})$. Let

$$\phi' \equiv \Pi \wedge *_{k\in K}\ w_k \mapsto (\mathbf{u}^k) * *_{1\leq i\leq I} P_1^{(T_{1,i})}(t_{1,i}, a_{1,i}) * *_{1\leq j\leq J} P_2^{(T_{2,j})}(t_{2,j}, a_{2,j})$$

Then $P^{(T)}(\mathbf{t}, \mathbf{a})$ is ϕ', $P^{S(T)}(\mathbf{t})$ is ϕ'^S, and $P^{N(T)}(\mathbf{a})$ is ϕ'^N. Hence both ϕ'^S and ϕ'^N are satisfiable. By Proposition 4.13, $[\![\phi'^S]\!]$ is satisfiable.

Let $T_{1,i}$ be $T(l_i, n_i')$. Take n_i such that $n_i' = n_i + k(q_i - p_i)$ for some $k \geq 0$ and $n_i < q_i - p_i$. Since $[\![P_1^{S(T_{1,i})}(x)]\!] = [\![(\Phi_1^{n_i'}[\lambda x.\Phi_{1,l_i}])^S]\!]$, and $[\![(\Phi_1^{p_i}[\lambda x.\Phi_{1,l_i}])^S]\!] \simeq [\![(\Phi_1^{q_i}[\lambda x.\Phi_{1,l_i}])^S]\!]$ by the step 1, we have $[\![P_1^{S(T_{1,i})}(x)]\!] \simeq [\![P_1^{S(T(l_i,n_i))}(x)]\!]$.

Define m_j as $T_{2,j}$ if $1 \leq T_{2,j} \leq I_2$ and (r, r) if $T_{2,j}$ is (T_1, T_2) for some T_1, T_2 and r is arbitrarily chosen from $\widetilde{(T_1, T_2)}$. Since $P_2^{S(T_{2,j})}(t_{2,j})$ is satisfiable, $\Phi_{2,i}^S$ is satisfiable for all $i \in \widetilde{(T_1, T_2)}$. Since $[\![P_2^{S((T_1,T_2))}(x)]\!] = [\![P_2^{S((r,r))}(x)]\!]$ by Lemma 4.15, $[\![P_2^{S(T_{2,j})}(x)]\!] = [\![P_2^{S(m_j)}(x)]\!]$.

Since $[\![P_1^{S(T_{1,i})}(x)]\!] \simeq [\![P_1^{S(T(l_i,n_i))}(x)]\!]$, $[\![P_2^{S(T_{2,j})}(x)]\!] = [\![P_2^{S(m_j)}(x)]\!]$, and $[\![\phi'^S]\!]$ is satisfiable,

$$[\![\Pi \wedge *_{k\in K} w_k \mapsto ((u^k)^S) * *_{1\le i\le I} P_1^{S(T(l_i,n_i))}(t_{1,i}) * *_{1\le j\le J} P_2^{S(m_j)}(t_{2,j})]\!]$$

is satisfiable by Lemma 4.11 (2). Hence we go from the step 3 to the step 4.

Since ϕ'^N is satisfiable, we have s such that $s \models \phi'^N$. Hence $s \models P_1^{N(T_{1,i})}(a_{1,i})$, $s \models P_2^{N(T_{2,j})}(a_{2,j})$, and $s \models \Lambda$. Since $P_1^{N(T(l_i,k(q_i-p_i)))}(y) \equiv P_{1,i}^k(y)$, we have $P_1^{N(T_{1,i})}(y) \to P_{1,i}(y)$. Therefore $P_1^{N(T_{1,i})}(a_{1,i}) \to \Lambda_{1,i}$ by the step 4. If $0 \le T_{2,j} \le I_2$, then $P_2^{N(T_{2,j})} \equiv \Lambda_{2,j}$ by the step 4. If $T_{2,j} = (T_1, T_2)$, then $P_2^{N(T_{2,j})}(a_{2,j}) \to (\Phi_2[P_2, P_2])^N[P_2^N := P_3][y := a_{2,j}]$ and $P_2^{N(T_{2,j})} \to \Lambda_{2,j}$ by the step 4. Hence we have $s \models \Lambda_{1,i}$ and $s \models \Lambda_{2,j}$. Hence $s \models \bigwedge_{1\le i\le I} \Lambda_{1,i} \wedge \bigwedge_{1\le j\le J} \Lambda_{2,j} \wedge \Lambda$. Hence the algorithm returns Yes at the step 5.

(2) We will show that ϕ is satisfiable if the algorithm returns Yes.

We have some $(l_1, n_1, \ldots, l_I, n_I, m_1, \ldots, m_J)$ such that

$$[\![\Pi \wedge *_{k\in K} w_k \mapsto ((u^k)^S) * *_{1\le i\le I} P_1^{S(T(l_i,n_i))}(t_{1,i}) * *_{1\le j\le J} P_2^{S(m_j)}(t_{2,j})]\!]$$

is satisfiable by the step 3, and $\bigwedge_{1\le i\le I} \Lambda_{1,i} \wedge \bigwedge_{1\le j\le J} \Lambda_{2,j} \wedge \Lambda$ is satisfiable by the step 5.

Then we have s such that $s \models \bigwedge_{1\le i\le I} \Lambda_{1,i} \wedge \bigwedge_{1\le j\le J} \Lambda_{2,j} \wedge \Lambda$. Then $s \models \Lambda_{1,i}$.

Since $\Lambda_{1,i} \leftrightarrow P_{1,i}(a_{1,i})$, we have some $k_i \ge 0$ such that $s \models P_1^{N(T(l_i,n_i+k_i(q_i-p_i)))}(a_{1,i})$. Define $T_{1,i}$ as $T(l_i, n_i + k(q_i - p_i))$. Then $s \models P_1^{N(T_{1,i})}(a_{1,i})$.

We define $T_{2,j}$ such that $s \models P_2^{N(T_{2,j})}(a_{2,j})$ by cases according to m_j.

Case 1. $1 \le m_j \le I_2$. We define $T_{2,j}$ as m_j. Then $s \models P_2^{N(T_{2,j})}(a_{2,j})$.

Case 2. $m_j = (r, r)$. Since $\Lambda_{2,j} \leftrightarrow (\Phi_2[\widetilde{P_2, P_2}])^N[y := a_{2,j}]$, there are $T_{3,j}, T_{4,j}$ such that $\Phi_{2,i}$ is satisfiable for all $i \in (T_{3,j}, T_{4,j})$, and $s \models (\Phi_2[P, P])^N[P := P_2^{N(T_{3,j})}, P_2^{N(T_{4,j})}][y := a_{2,j}]$, which is $P_2^{N((T_{3,j},T_{4,j}))}(a_{2,j})$. Define $T_{2,j}$ as $(T_{3,j}, T_{4,j})$. Then $s \models P_2^{N(T_{2,j})}(a_{2,j})$.

Let

$$\phi' \equiv \Pi \wedge *_{k\in K}\, w_k \mapsto (\mathbf{u}^k) * *_{1\le i\le I} P_1^{(T_{1,i})}(t_{1,i}, a_{1,i}) * *_{1\le j\le J} P_2^{(T_{2,j})}(t_{2,j}, a_{2,j}) \wedge \Lambda.$$

Then we have $s \models \phi'^N$.

$[\![P_1^{S(T_{1,i})}(x)]\!] \simeq [\![P_1^{S(T(l_i,n_i))}(x)]\!]$ by the step 1.

We can show that $[\![P_2^{S(T_{2,j})}(x)]\!] = [\![P_2^{S(m_j)}(x)]\!]$ as follows. If $1 \le m_j \le I_2$, then $T_{2,j} = m_j$ and the claim holds. Assume $m_j = (r, r)$. Then $T_{2,j}$ is $(T_{3,j}, T_{4,j})$. Since $[\![P_2^{S(T_{2,j})}(x)]\!] = [\![P_2^{S((r,r))}(x)]\!]$ by Lemma 4.15, we have the claim.

Since $[\![P_1^{S(T_{1,i})}(x)]\!] \simeq [\![P_1^{S(T(l_i,n_i))}(x)]\!]$, $[\![P_2^{S(T_{2,j})}(x)]\!] = [\![P_2^{S(m_j)}(x)]\!]$, and

$$[\![\Pi \wedge *_{k\in K} w_k \mapsto ((u^k)^S) * *_{1\le i\le I} P_1^{S(T(l_i,n_i))}(t_{1,i}) * *_{1\le j\le J} P_2^{S(m_j)}(t_{2,j})]\!]$$

is satisfiable, by Lemma 4.11 (2),

$$[\![\Pi \wedge *_{k\in K} w_k \mapsto ((u^k)^S) * *_{1\le i\le I} P_1^{S(T_{1,i})}(t_{1,i}) * *_{1\le j\le J} P_2^{S(T_{2,j})}(t_{2,j})]\!]$$

is satisfiable. Namely $[\![\phi'^S]\!]$ is satisfiable. By Proposition 4.13, ϕ'^S is satisfiable. Hence we have s', h such that $s', h \models \phi'^S$.

We define s'' by $s''(x) = s'(x)$ for a pointer variable x and $s''(y) = s(y)$ for an integer variables y. We have $s'', h \models \phi'^S$.

Let $\mathbf{x} = x_{1,1} \dots x_{1,I_1} x_{2,1} \dots x_{2,I_2}$, $\mathbf{y} = y_{1,1} \dots y_{1,I_1} y_{2,1} \dots y_{2,I_2}$, $\mathbf{t} = t_{1,1} \dots t_{1,I_1} t_{2,1} \dots t_{2,I_2}$, and $\mathbf{a} = a_{1,1} \dots a_{1,I_1} a_{2,1} \dots a_{2,I_2}$. Define the predicate P by

$$P(\mathbf{x}, \mathbf{y}) \equiv \Pi \wedge *_{k\in K} w_k \mapsto (\mathbf{u}^k) * *_{1\le i\le I} P_1(x_{1,i}, y_{1,i}) * *_{1\le j\le J} P_2(x_{2,j}, y_{2,j}).$$

Define T as $(T_{1,1}, ..., T_{1,I}, T_{2,1}, ..., T_{2,J})$. Then $P^{(T)}(\mathbf{t}, \mathbf{a})$ is ϕ', $P^{S(T)}(\mathbf{t})$ is ϕ'^S, and $P^{N(T)}(\mathbf{a})$ is ϕ'^N.

Since

$$s'', h \models P^{S(T)}(\mathbf{t}),$$
$$s'' \models P^{N(T)}(\mathbf{a}),$$

by Proposition 4.6 we have some h' such that

$$s'', h' \models P^{(T)}(\mathbf{t}, \mathbf{a}).$$

Namely $s'', h' \models \phi'$. Hence $s'', h' \models \phi$. □

5 Conclusion

We have proved that the satisfiability of symbolic heaps in SLA1 system with inductive definitions and Presburger arithmetic without any restrictions is undecidable. We have proposed a decidable symbolic-heap subsystem SLA2 with inductive definitions and Presburger arithmetic with some restrictions, and provided its decision algorithm as well as its correctness proof. To support this result, we have also defined a related decidable subsystem DPI of Presburger arithmetic and inductive definitions with some restrictions.

We have imposed a significant restriction on SLA2 for the case when the unfolding trees becomes non-linear. SLA2 supports AVL trees, but does not support sorted AVL trees because the minimum values and the maximum values interact. Future work could relax the restrictions by using semilinear sets so that it supports a wider class of data structures. We have not investigated pointer arithmetic. An extension of our results to pointer arithmetic could be another future work.

Acknowledgments

This work is partially supported by MoE Tier-2 grant MOE2013-T2-2-146.

References

1. J. Berdine, C. Calcagno, P. W. O'Hearn, A Decidable Fragment of Separation Logic, In: Proceedings of FSTTCS 2004, *LNCS* 3328 (2004) 97–109.
2. J. Berdine, C. Calcagno, and P. W. O'Hearn, Symbolic Execution with Separation Logic, In: Proceedings of APLAS 2005, *LNCS* 3780 (2005) 52–68.
3. M. Bozga, R. Iosif, and S. Perarnau, Quantitative Separation Logic and Programs with Lists, *J. Autom. Reasoning* 45 (2) (2010) 131–156.
4. J. Brotherston, C. Fuhs, N. Gorogiannis, and J.N. Perez, A Decision Procedure for Satisfiability in Separation Logic with Inductive Predicates, *In: Proceedings of CSL-LICS'14* (2014) Article 25.
5. W.N. Chin, C. David, H.H. Nguyen, and S. Qin, Automated verification of shape, size and bag properties via user-defined predicates in separation logic, *Sci. Comput. Program.* 77(9) (2012) 1006–1036.
6. H. B. Enderton, *A Mathematical Introduction to Logic, Second Edition*, Academic Press, 2000.
7. R. Iosif, A. Rogalewicz, and J. Simácek, The tree width of separation logic with recursive definitions, In: Proceedings of CADE-24, *LNCS* 7898 (2013) 21–38.
8. R. Iosif, A. Rogalewicz, and T. Vojnar, Deciding Entailments in Inductive Separation Logic with Tree Automata, In: Proceedings of ATVA 2014, *LNCS* 8837, 201–218.
9. R. Piskac, T. Wies, and D. Zufferey, Automating Separation Logic Using SMT, In: Proceedings of CAV 2013, *LNCS 8044* (2013) 773-789.
10. R. Piskac, T. Wies, and D. Zufferey, Automating separation logic with trees and data, In: Proceedings of CAV 2014, *LNCS 8559* (2014) 711–728.
11. J. A. Navarro Préze and A. Rybalchenko, Separation logic modulo theories, In: *Proceedings of APLAS 2013, LNCS 8301 (2013) 90-106.*
12. *J.C. Reynolds, Separation Logic: A Logic for Shared Mutable Data Structures, In: Proceedings of Seventeenth Annual IEEE Symposium on Logic in Computer Science (LICS2002) (2002) 55–74.*
13. *James Brotherston and Nikos Gorogiannis and Max Kanovich and Reuben Rowe, Model Checking for Symbolic-Heap Separation Logic with Inductive Predicates, In: Proceedings of POPL-43 (2016) 84–96.*