Compositional Satisfiability Solving in Separation Logic

Quang Loc Le

University College London, United Kingdom - quang.le@ucl.ac.uk

Abstract. We introduce a novel decision procedure to the satisfiability problem in array separation logic combined with general inductively defined predicates and arithmetic. Our proposal differentiates itself from existing works by solving satisfiability through compositional reasoning. First, following Fermat's method of infinite descent, it infers for every inductive definition a "base" that precisely characterises the satisfiability. It then utilises the base to derive such a base for any formula where these inductive predicates reside in. Especially, we identify an expressive decidable fragment for the compositionality. We have implemented the proposal in a tool and evaluated it over challenging problems. The experimental results show that the compositional satisfiability solving is efficient and our tool is effective and efficient when compared with existing solvers.

Keywords: Separation logic · Satisfiability · Regular proofs.

1 Introduction

Satisfiability solvers are essential to symbolic analysis in checking code correctness. Such an analysis executes programs symbolically and constructs path conditions, those constraints on values stored in program variables and expressed in terms of input parameters, that reach a program point. A satisfiability solver is then utilized for discharging the path conditions to either detect and prune infeasible paths or to generate inputs that exercise a buggy path. So far, while techniques to satisfiability solving for heap-independent domains e.g., satisfiability modulo theories (SMT) [1], have been well developed, there have been a few works targeting heap-oriented logics. Heap-manipulating programs are often building-blocks of real-world applications (e.g., data structures like arrays and trees). Especially, heap-related bugs are the sources of the security vulnerabilities; for instance, memory leak, double free and use-after-free caused the vulnerabilities CVE-2020-12768, CVE-2019-3896 and CVE-2020-8649, respectively, found in the Linux kernel recently. Therefore, satisfiability solver that supports the symbolic execution over heap-manipulating programs is important.

Separation logic [16,36] formalism has been increasingly applied for reasoning about heap-based programs. Combining with general inductive definitions and arithmetic, it can concisely and precisely represent constraints over unbounded and complex data structures (e.g., nested lists, AVL trees) [5,10,19,23,29,30,31,32]. The strength of the logic is the support for compositional reasoning through the separating conjunction operator, which allows reasoning about disjoint portions of heaps locally and independently. The compositional reasoning has been applied through the frame rule and automated using the bi-abduction technique [9,19]. Implemented in Facebook's Infer [8], the compositional reasoning helps the verification scale up to millions of lines of code.

Our research question is whether the compositional reasoning introduced in [9] could be applied to a satisfiability solver or not. The satisfiability problem in separation logic was studied in [5] for inductive definitions with heap-only constraints and in [23] for those combining both heap and arithmetic (heap-independent) constraints. Given a formula, these works essentially compute a *base* that characterises its satisfiability precisely. In these works, a base generated for every formula is used to check the satisfiability of the formula itself. Yet another challenge is to develop a satisfiability solver that can derive a base for formula in a modular way: The base of a formula is defined by terms of the bases of its parts and a means of combining them.

In this paper, we present a satisfiability solver with the capability of the compositionality in *array separation logic* combined with general definitions of inductive predicates and arithmetic properties. We study a decidable fragment with *small heap model* property: The base of the formula is satisfied by those models, the interpretations of variables, where heaps are minimal and finite. Especially, the base is a separation logic formula without any inductive predicates. A base of a symbolic heap, a conjunctive formula, is inductively computed from the bases of its conjuncts. In this endeavour, the difficulty we face is to find a base for each inductive predicate defined with recursive definitions. To overcome this challenge, we develop an algorithm as an application of Fermat's method of infinite descent. The method of infinite descent is a standard approach to Diophantine equations. This method is typically applied in two ways:

1. To show that an equation P has no solution. Let n be a positive integer, suppose that whenever $P(n)$ holds, there exists a positive integer m such that $m < n$ and $P(m)$ holds. Then, $P(x)$ is false for all positive integers x.
2. To show that an equation P has a set of solutions. First, we need to hypothesize a simpler equation Q which satisfies the following two conditions:
   - $Q(a)$ and $P(a)$ hold for some natural number constant a,
   - and whenever $Q(n)$ and $P(n)$ hold, there exists a positive integer m such that $m < n$ and both $Q(m)$ and $P(m)$ hold.

   Then, P has the same set of solutions with Q.

The work in [21,23] used the first application to show the unsatisfiability of a formula. Here, we apply the second approach to derive the base (like Q in the method of infinite descent) through a so-called regular unfolding tree that represents a set of all equisatisfiable solutions of a formula. The bases of inductive predicates are computed once and are independent of the contexts where they are used. Through compositionality, we hope that our proposal could help to improve the performance of those symbolic analyses (e.g., test case generation [29,31]) through the reuse of the bases to discharge hundreds of satisfiability problems over the same set of predicate definitions.

Contributions Our primary contributions are summarised as follows.

- We propose a novel decision algorithm that can compositionally compute a base for checking the satisfiability of a separation logic formula.
- We show that our solver is more expressive than all existing works (without the separating implication). Alongside this, we describe a novel decision procedure for Presburger arithmetic that includes nested inductive definitions.

– We have implemented the proposal in a prototype solver, called `S2S`. Our experimental results on those benchmarks taken from SL-COMP 2019, a competition of separation logic solvers [37], and generated during the program verification show that `S2S` with compositional reasoning is effective and efficient.

Organization The remainder of the paper is organized as follows. Sect. 2 illustrates our ideas and presents motivating examples. Sect. 3 describes the fragment of separation logic. Sect. 4 introduces regular unfolding trees, the intermediate structure constructed to represent all solutions of a formula. In Sect. 5, we present our solver and identify semantic conditions for decidability. We refer to the fragment satisfies these conditions as $\texttt{SLIDLIA}_{\texttt{sem}}$. The existing fragments presented in [3,5,14,18,39,21,23,41] are subsumed by $\texttt{SLIDLIA}_{\texttt{sem}}$ straightforwardly. Sect. 6 shows `SLIDLIA`, a novel syntactic decidable fragment of $\texttt{SLIDLIA}_{\texttt{sem}}$. Sect. 7 presents the implementation and evaluation. Sect. 8 discusses related work. Finally, Sect. 9 concludes.

2 Basic Concepts and Motivating Examples

In this section, we first informally explain how to apply Fermat's method of infinite descent to compute the bases for inductive predicates. Next, we elaborate on our ideas through two examples that are beyond the capability of existing works.

Generating bases via regular unfolding trees We show how to find a base, the "another simpler related equation" in the second application of the method of infinite descent, for an inductive predicate. We infer those bases through regular unfolding trees. Such a tree is a (possibly infinite) tree of formulas: the leaves are either base formulas - those without any inductive predicates - or nodes (called buds) which are linked back to inner nodes (called companions), the root of the tree is the formula being unfolded, and nodes are connected to one or more children through predicate unfolding i.e., replacing an inductive predicate in the parents by its definition. Equivalently, an unfolding tree is a *regular* tree that is generated by a finite directed (cyclic) graph. Fig. 1 shows an example of the regular unfolding tree, a tree with cycles through back-links. In the back-link between B and C, B is a bud, C is a companion and σ is a substitution between variables. B is linked back to C only when:

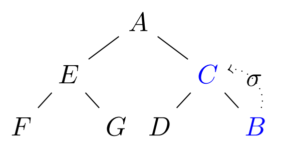


Fig. 1: A regular unfolding tree

– There exist $\texttt{P}(\bar{x}) \in C$ and $\texttt{P}(\bar{y}) \in B$ s.t. $\texttt{P}(\bar{y})$ is in a formula unfolded from $\texttt{P}(\bar{x})$;
– And there exists B' such that B' is *equisatisfiable* with B, $B \Rightarrow B'$ and $B'\sigma \equiv C$.

The base of an inductive predicate is generated as the disjunction of the leaf bases of its regular unfolding tree. The most challenging task we have to solve is to compute bases for subtrees involving cycles. For such a subtree, we need to find a base formula that exactly characterises the set of solutions of the companion. In the following, we illustrate how to compute such a base for the subtree rooted by C in Fig. 1.

1. If the base leaf D is unsatisfiable, then we infer the base as `false`.

2. Otherwise, following the second application of infinite descent, we generate the base as a base formula that exactly characterises the satisfiability of $(D \vee B')$.

After that, the whole subtree rooted by the companion is replaced by the inferred base. The process which computes bases for subtrees involving cycles is applied repeatedly in a bottom-up manner until the tree does not contain any cycles. Then, the base of the root is the disjunctive set of satisfiable leaves of the final tree.

Comparing to the base in [23] Using regular unfolding trees, the work in [23] generates bases to check satisfiability without compositionality. The main difference between this work and ours is in inferring the bases for subtrees involving cycles in the second case above. To make a back-link with a bud B and a companion C, the algorithm in [23] finds a substitution σ and a formula B' for $B \Rightarrow B'$ and $B'\sigma \equiv C$. As it might over-approximate B, it might generate over-approximated bases. Since D is satisfiable, these bases are still complete when utilized to check the satisfiability of companion C and its ancestors. If, however, they are combined with some formula, the combined one might be over-approximated (as the frame rule says: $P(\bar{x}) \Rightarrow base$ implies $P(\bar{x}) * some\_formula \Rightarrow base * some\_formula$). Hence, the solver in [23] may produce false positives if it is for compositional solving. In contrast, ours ensures B' is equisatisfiable with B and the base generated exactly characterises the satisfiability of the predicate provided. Our solver is thus sound and complete for compositionality.

First Example Let us consider the following satisfiability problem $\varDelta_0$:

$$\varDelta_0 = \mathtt{odd}(x,y,m) * \mathtt{odd}(y,\mathtt{null},n) \wedge (\exists k.\ m + n = 2k + 1)$$

where $\mathtt{odd}(x,y,m)$ is an inductive predicate representing singly-linked list segments with head pointer x, ending pointer y, and odd length m. $\varDelta_0$ is a symbolic heap. It is a conjunction of a spatial formula, $\mathtt{odd}(x,y,m) * \mathtt{odd}(y,\mathtt{null},n)$, and a pure (heap-independent) formula, $\exists k.\ m+n = 2k+1$. The spatial one specifies two connected list segments headed by x and y. The two lists are conjoined by the separating conjunction $*$ that tells us they lie on disjoint heap regions. The pure formula specifies that the sum of the lengths of the two lists, m and n, is an odd number.

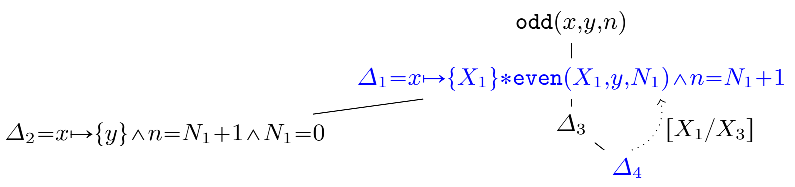
The inductive predicate `odd` is mutually defined as follows.

$$\begin{array}{ll}\mathtt{odd}(x,y,n) & \equiv \exists x_1.\ x{\mapsto}\{x_1\} * \mathtt{even}(x_1,y,n{-}1);\\ \mathtt{even}(x,y,n) & \equiv \mathtt{emp} \wedge x = y \wedge n = 0 \vee \exists x_1. x{\mapsto}\{x_1\} * \mathtt{odd}(x_1,y,n{-}1);\end{array}$$

Here, each inductive definition is a disjunction of symbolic heaps. A symbolic heap may be existentially quantified. A heap formula is a conjunction of atomic predicates: `emp` to specify empty heap, points-to predicates (e.g., $x{\mapsto}\{x_1\}$ above) to assert a singleton heap, and occurrences of inductive predicates (e.g., $\mathtt{even}(x_1,y,n{-}1)$ above).

$\varDelta_0$ is unsatisfiable. This problem is challenging for the existing solvers. It includes the arithmetic constraints which are beyond the fragments presented in Smallfoot [3], `SLSAT` [5], `SPEN` [13], `Asterix` [27] and Harrsh [17]. Due to the mutual recursion, it is beyond the capability of the algorithms presented in [14,18,39,41].

$\varDelta_0$ is in our decidable fragment. The proposed solver, `S2S`, proves its unsatisfiability through the following two phases. First, it computes for predicate `odd` a base that precisely characterises its satisfiability. Here, a base is a (possibly disjunctive) separation


$\mathtt{odd}(x,y,n)$
$\Delta_1 = x \mapsto \{X_1\} * \mathtt{even}(X_1,y,N_1) \wedge n=N_1+1$
$\Delta_2 = x \mapsto \{y\} \wedge n=N_1+1 \wedge N_1=0$
Δ_3
$[X_1/X_3]$
Δ_4

$$\Delta_3 = x \mapsto \{X_1\} * X_1 \mapsto \{X_2\} * \mathtt{odd}(X_2,y,N_2) \wedge n=N_1+1 \wedge N_1=N_2+1$$
$$\Delta_4 = x \mapsto \{X_1\} * X_1 \mapsto \{X_2\} * X_2 \mapsto \{X_3\} * \mathtt{even}(X_3,y,N_3) \wedge n=N_1+1 \wedge N_1=N_2+1 \wedge N_2=N_3+1$$

Fig. 2: Regular unfolding tree of $\mathtt{odd}(x,y,n)$

logic formula without any inductive predicate occurrences. Secondly, it derives a base Δ_0' for Δ_0 by replacing every occurrence of `odd` by the base inferred in the first phase. As Δ_0' does not contain any inductive predicates, satisfiability is decidable [21,28].

In this example, `S2S` infers the base for predicate $\mathtt{odd}(x, y, n)$ as:

$$\{\ x \mapsto \{\_\} \wedge (\exists i.\ n = 2i + 1 \wedge i \geqslant 0)\ \} \text{ where } \_ \text{ denotes existential variables.}$$

After that, it replaces each occurrence of `odd` in Δ_0 with that base to obtain:

$$\Delta_0' \equiv x \mapsto \{\_\} * y \mapsto \{\_\} \wedge \underline{(\exists k.m+n=2k+1) \wedge (\exists i.m=2i+1 \wedge i \geqslant 0) \wedge (\exists i.n=2i+1 \wedge i \geqslant 0)}$$

As Δ_0' is unsatisfiable (the unsatisfiable cores are underlined), so is Δ_0.

We now show how `S2S` infers the base for predicate $\mathtt{odd}(x, y, n)$. It first generates the regular unfolding tree for $\mathtt{odd}(x, y, n)$ in Fig. 2. In this tree, uppercase variables are existentially quantified, and the back-link is constructed based on the spatial (heap-dependent) projection of the formulas. Next, in a bottom-up manner, it finds a base for each subtree rooted by a companion and then replaces the whole subtree with that base. That base is a combination of its spatial part and its numeric part that are derived separately. For the subtree rooted by Δ_1 in Fig. 2, the spatial projection of the base is $x \mapsto \{\_\}$, the spatial formula of Δ_2. We, especially, show that the (infinite) set of all spatial formulas derived from this subtree is equisatisfiable with this base. The numeric projection of the base is equivalent to $\mathtt{even}^N(N_1) \wedge n=N_1+1$ where $\mathtt{even}^N(N_1)$ is defined from the ones of Δ_2 and Δ_4 as: $\mathtt{even}^N(N_1) \equiv N_1=0 \vee \mathtt{even}^N(N_3) \wedge N_1=N_3+2$. Then, it derives for $\mathtt{even}^N(N_1)$ an equivalent closed form, a Presburger formula, as : $\mathtt{even}^N(N_1) \equiv \exists i.\ N_1=2i \wedge i \geqslant 0$. Finally, the base of $\mathtt{odd}(x,y,n)$ is the base of Δ_1, the conjunction of its spatial and numeric projections, as $x \mapsto \{\_\} \wedge (\exists i.\ n=2i+1 \wedge i \geqslant 0)$.

Second Example Let us consider the satisfiability problem in a fragment including the following nested lists whose data values are increasingly sorted.

$$\begin{array}{l}
\mathtt{sllss}(x,y,mi,ma,n,n_0) \equiv \mathtt{emp} \wedge x=y \wedge mi=ma \wedge n=n_0 \\
\quad \vee\ \exists u, mi_1, n_1.x \mapsto \{mi,u\} * \mathtt{sllss}(u,y,mi_1,ma,n_1,n_0) \wedge x \neq y \wedge mi < mi_1 \wedge n=n_1+1 \\
\mathtt{nllss}(x,y,b,mi,ma,n,n_0) \equiv \mathtt{emp} \wedge x=y \wedge mi=ma \wedge n=n_0 \\
\quad \vee\ \exists\ u,Z,m_1,m_2,n_1,n_2.x \mapsto \{mi,u,Z\} * \mathtt{sllss}(Z,b,m_1,m_2,n_1,0) * \\
\qquad \mathtt{nllss}(u,y,b,m_2,ma,n_2,n_0) \wedge x \neq y \wedge x \neq b \wedge mi < m_1 \wedge n=n_1+n_2+1
\end{array}$$

Here, n and n_0 are parameters to capture the size (i.e., the number of heap cells) of the lists. `nllss` predicate contains nested length constraints that are beyond all existing decidable fragments. We show that `nllss` is in the decidable fragment, and can support compositional satisfiability solving. We compute the length constraints in arithmetic with addition and divisibility prior to transforming it into Presburger arithmetic. We show how to derive the bases for these predicates throughout the rest of the paper.

3 Array Separation Logic with Inductive Definitions

In this section, we first present the syntax and semantics of formulas in our work. After that, we show how to obtain spatial and numeric projections of formulas.

Definition 1 (Symbolic heap). *Terms t, (Presburger) pure formulas π, spatial formulas κ, symbolic heaps Δ and disjunctions Φ are given by the following grammar.*

$$\Phi ::= \Delta \mid \Phi \vee \Phi \qquad \Delta ::= \kappa{\wedge}\pi \mid \exists v.\ \Delta$$
$$\kappa ::= \mathtt{emp} \mid v{\mapsto}\{t_1,..,t_N\} \mid \mathtt{P}(\bar{v}) \mid array(t,t) \mid \kappa{*}\kappa \qquad \alpha ::= t{=}t \mid t{=}\mathtt{null} \mid t{\leqslant}t$$
$$\pi ::= \mathtt{true} \mid \alpha \mid \neg\pi \mid \pi{\wedge}\pi \mid \pi{\vee}\pi \qquad t ::= c \mid v \mid t{+}t \mid -t$$

where v ranges over an infinite set Var of variables, $\bar{t}$ over sequences of terms (either variables or `null`*) ($\bar{t}_i$ for its i^{th} element), c over $\mathbb{Z}$, P over a finite set $\mathcal{P}$.*

The array predicate only records the bound of contiguous memory blocks, not their contents. Note that $t_1{\neq}t_2$ is the short form for $\neg(t_1{=}t_2)$. *FV*(Φ) returns the free variables of Φ. We write $\Phi(\bar{v})$ to denote that $\bar{v}$ = *FV*(Φ). $\Delta[t_1/v_2]$ denotes the formula obtained by substituting each term t_1 in Δ for the variable v_2. $\exists \bar{w}, v.\ \Delta \wedge v{=}x$ is normalised into $\exists \bar{w}.\ \Delta[x/v]$ and $\pi \wedge \pi$ is normalised into π.

Given a formula $\exists \bar{w}.\ \mathtt{P}(\bar{v}) * v{\mapsto}\{\bar{t}\} * array(v_1, v_2) * \kappa \wedge x{\diamond}y \wedge \pi$ (where $\diamond{\in}\{=, \neq\}$), inductive predicate $\mathtt{P}(\bar{v})$ (resp. $array(v_1, v_2)$) is called (heap) observable if there exists at least one variable in $\bar{v}$ (resp. $\{v_1, v_2\}$) that is quantifier-free (i.e., $x{\diamond}y \wedge \pi$ implies that $\bar{v}\backslash\bar{w} \neq \emptyset$); $v{\mapsto}\{\bar{t}\}$ is called (heap) observable if v is quantifier-free (i.e., $x{\diamond}y \wedge \pi$ implies that $v \notin \bar{w}$). Finally, $x{\diamond}y$ is observable if both x and y are quantifier-free (i.e., $x{\diamond}y{\wedge}\pi$ implies that $\{x; y\}{\cap}\bar{w} = \emptyset$). If a predicate is not observable, it is unobservable. $\overline{\Delta}$ is a formula obtained by replacing every $v{\mapsto}\{\bar{t}\}{\in}\Delta$ by $v{\mapsto}\{\_\}$.

Φ is a *base* formula if it does not contain any occurrence of inductive predicates. Otherwise, it is an inductive formula. We use B to denote a conjunctive base formula.

A definition of an inductive predicate is a disjunction as $\mathtt{P}(\bar{v}) \equiv \Phi$. In each disjunct of Φ (called a definition rule), all variables which are not formal parameters are existentially quantified. We use $\mathtt{base}^{\mathcal{P}}(\mathtt{P}(\bar{v}))$ to denote the base of $\mathtt{P}(\bar{v})$.

Semantics Concrete heap models assume a set *Loc* of locations (heap addresses), such that *Loc* $\subseteq \mathbb{Z}$ and `null` $\notin \mathbb{Z}$. The semantics is given by a satisfaction relation: $s{,}h \models \Phi$ is valid if the stack $s \in$ *Stacks* and heap $h \in$ *Heaps* satisfies the formula Φ. Stack and heap abstractions are defined (assume that every points-to predicate has at most N fields):

$$\mathit{Heaps} \overset{\text{def}}{=} \mathit{Loc} {\rightharpoonup_{fin}} \mathbb{Z}^N \qquad\qquad \mathit{Stacks} \overset{\text{def}}{=} \mathit{Var} \rightarrow \mathbb{Z}$$

Suppose that $dom(f)$ is the domain of function f, $h_1\#h_2$ denotes disjoint heaps h_1 and h_2, and $h_1{\cdot}h_2$ denotes the union of two disjoint heaps. If s is a stack, $v{\in}$*Var* and

$\alpha{\in}\mathbb{Z}$, we write $s[v{\mapsto}\alpha]{=}s$ if $v{\in}dom(s)$ and $s[v{\mapsto}\alpha]{=}s{\cup}\{(v,\alpha)\}$ if $v{\notin}dom(s)$. The interpretation of an inductive predicate $\mathtt{P}(\bar{v})$, denoted by $[\![\mathtt{P}(\bar{v})]\!]$, is based on the least fixed point semantics (cf. [23]). Then, the semantics is shown below.

$s,h \models \mathtt{emp}$	$\mathtt{iff}$	$dom(h){=}\emptyset$	$s,h \models \mathtt{true}$	$\mathtt{iff}$	always
$s,h \models v{\mapsto}\{v_1,..,v_N\}$	$\mathtt{iff}$	$\mathrm{dom}(h){=}\{s(v)\}$ and $h(s(v)){=}(s(v_1),..,s(v_N))$			
$s,h \models array(t_1,t_2)$	$\mathtt{iff}$	$s(t_1){\leqslant}s(t_2)$ and $\mathrm{dom}(h){=}\{s(t_1),...,s(t_2)\}$			
$s,h \models \mathtt{P}(\bar{v})$	$\mathtt{iff}$	$(h,s(\bar{v}_1),..,s(\bar{v}_k)) \in [\![\mathtt{P}(\bar{v})]\!]$			
$s,h \models \kappa_1 * \kappa_2$	$\mathtt{iff}$	$\exists h_1,h_2.\ h_1\#h_2, h{=}h_1.\ h_2$ s.t. $s,h_1 \models \kappa_1$ and $s,h_2 \models \kappa_2$			
$s,h \models \kappa{\wedge}\pi$	$\mathtt{iff}$	$s,h \models \kappa$ and $s \models \pi$			
$s,h \models \exists v.\ \Delta$	$\mathtt{iff}$	$\exists\alpha.\ s[v{\mapsto}\alpha],h \models \Delta$			
$s,h \models \Phi_1 \vee \Phi_2$	$\mathtt{iff}$	$s,h \models \Phi_1$ or $s,h \models \Phi_2$			

Semantics of pure formulas is omitted, for simplicity.

Projections [39,23] For every variable $v{\in}Var$, if it appears in a spatial formula then it is a *spatial* variable. Otherwise, it is a *numeric* variable. $\bar{x}^S$ (resp. $\bar{x}^N$) is a sequence of variables similar to $\bar{x}$ excluding numeric (resp. spatial) variables. $|\bar{x}^S|$ is a sequence of variables obtained by replacing every spatial variable in $\bar{x}$ with a fresh existential one.

For each inductive predicate $\mathtt{P}(\bar{t}) \equiv \Phi$, we assume the inductive symbol $\mathtt{P^S}$ and predicate $\mathtt{P^S}(\bar{t}^S)$ for its spatial projection that satisfy $\mathtt{P^S}(\bar{t}^S) \equiv \Phi^S$. Similarly, we presume the inductive symbol $\mathtt{P^N}$ and predicate $\mathtt{P^N}(\bar{t}^N)$ for its numeric projection that satisfy $\mathtt{P^N}(\bar{t}^N) \equiv \Phi^N$. Given pure conjunction π, we can rewrite it as $\pi \equiv \alpha{\wedge}\beta{\wedge}\gamma$ where $FV(\alpha) \subseteq FV(\pi)^S$ and there does not exist another $\alpha'{\in}\pi$ such that $\alpha{\in}\alpha'$, $FV(\beta) \subseteq FV(\pi)^N$ and there does not exist another $\beta'{\in}\pi$ such that $\beta \in \beta'$, and γ is the conjunction of the remaining constraints. In the following, we define the two projections.

Definition 2. *The spatial projection $(\Phi)^S$ is defined inductively as follows.*

$$\begin{array}{llll}
(\Delta_1 \vee \Delta_2)^S & \equiv (\Delta_1)^S \vee (\Delta_2)^S & (\mathtt{P}(\bar{v}))^S & \equiv \mathtt{P^S}(\bar{v}^S) \\
(\exists\bar{v}.\Delta)^S & \equiv \exists\bar{v}^S.(\Delta)^S & (x{\mapsto}\{\bar{v}\})^S & \equiv x{\mapsto}\{|\bar{v}^S|\} \\
(\kappa{\wedge}\alpha{\wedge}\beta{\wedge}\gamma)^S & \equiv (\kappa)^S{\wedge}\alpha & (array(v_1,v_2))^S & \equiv array(v_1,v_2) \\
(\kappa_1{*}\kappa_2)^S & \equiv (\kappa_1)^S{*}(\kappa_2)^S & (\mathtt{emp})^S & \equiv \mathtt{emp}
\end{array}$$

Similarly, the numeric projection $(\Phi)^N$ is defined inductively as follows.

$$\begin{array}{llll}
(\Delta_1{\vee}\Delta_2)^N & \equiv (\Delta_1)^N{\vee}(\Delta_2)^N & (\kappa_1{*}\kappa_2)^N & \equiv (\kappa_1)^N{\wedge}(\kappa_2)^N \\
(\exists\bar{v}\cdot\Delta)^N & \equiv \exists\bar{v}^N\cdot(\Delta)^N & (\mathtt{P}(\bar{v}))^N & \equiv \mathtt{P^N}(\bar{v}^N) \\
(\kappa{\wedge}\alpha{\wedge}\beta{\wedge}\gamma)^N & \equiv (\kappa)^N{\wedge}\beta & \multicolumn{2}{l}{(x{\mapsto}\{\bar{v}\})^N{\equiv}(array(v_1,v_2))^N{\equiv}(\mathtt{emp})^N{\equiv}\mathtt{true}}
\end{array}$$

Definition 3 (Closed form). *Any numeric project $\mathtt{P^N}(\bar{v}^N)$ of an definition is called Presburger-definable if there exists a Presburger formula π such that for any stack s, we have: $s \models \mathtt{P^N}(\bar{v}^N)$ iff $s \models \pi$. We call π is the closed formula of the projection.*

We use function $Pres$ to map every Presburger-definable projection into its closed form.

Example 1. The numeric projection of predicate *sllss* in Sect. 2 is:

$$\begin{array}{l}\mathtt{sllss^N}(mi,ma,n,n_0) \equiv mi{=}ma{\wedge}n{=}n_0 \\ \quad \vee\ \exists mi_1,n_1.\mathtt{sllss^N}(mi_1,ma,n_1,n_0){\wedge}mi{<}mi_1{\wedge}n{=}n_1{+}1\end{array}$$

This numeric predicate is in the decidable fragment DPI [39] and its closed form is $Pres(\mathtt{sllss^N}(mi,ma,n,n_0)) \equiv mi{\leqslant}ma{\wedge}n{\geqslant}n_0$. □

4 Regular Unfolding Trees

Given an inductive predicate with spatial and pure constraints, its regular unfolding tree is generated based on the spatial projection and the base is collected for the spatial and arithmetic projections, separately. In this section, we first introduce regular unfolding trees (subsection 4.1). After that, we present an algorithm to construct the trees where back-links are generated based on the spatial projection of formulas (subsection 4.2). We also discuss the properties of the trees that are foundations for correctness.

4.1 Data Structure

A regular unfolding tree $\mathcal{T}$ is a tuple $(V, E, \mathcal{C})$ where

- V is a finite set of nodes each of which is a symbolic heap Δ.
- E is a set of labeled and directed edges $(\Delta, L, \Delta') \in E$ where Δ' is derived from unfolding an inductive predicate in Δ and L is a label to record which disjunct rule of the definition has been used. Given $\mathtt{P}(\bar{v}){\equiv}\bigvee_{i=1}^{n} \exists \bar{w}_i.\Delta_i$ and a node $\mathtt{e} \equiv \Delta{*}\mathtt{P}(\bar{t})$ where $\mathtt{P}(\bar{t})$ is chosen for unfolding, then new n nodes $\mathtt{e}_i \equiv \Delta * (\exists \bar{w}_i.\Delta_i)[\bar{t}/\bar{v}]$ and new n edges $(\mathtt{e}, (\mathtt{P}(\bar{t}), \exists \bar{w}_i.\Delta_i)[\bar{t}/\bar{v}], \mathtt{e}_i)$ are created.
- $\mathcal{C}$ is a back-link (partial) function. In a back-link $\mathcal{C}(\Delta_c{\rightarrow}\Delta_b, \sigma)$, the leaf node Δ_b is linked back to an ancestor Δ_c when the following two conditions hold. First, there exist $\mathtt{P}(\bar{x}) \in \Delta_c$ and $\mathtt{P}(\bar{y}) \in \Delta_b$ such that $\mathtt{P}(\bar{y})$ is in a subformula unfolded from $\mathtt{P}(\bar{x})$. Secondly, there exists Δ_b' s.t. $\Delta_b^S \Rightarrow \Delta_b'^S$, $\Delta_b'^S$ is equisatisfiable with Δ_b^S, and $\Delta_b'^S\sigma \equiv \Delta_c^S$. In such a back-link, Δ_b is a *bud*, and Δ_c is a *companion*.

A leaf node is marked as *open* or *closed*. It is marked as closed when it is either a base formula, unsatisfiable or a bud in a back-link. Otherwise, it is marked as open and may be chosen to reduce into multiple open nodes through predicate unfolding. $\mathtt{base}^{\mathcal{P}}(\Delta)$ denotes the set of satisfiable base formulas of the subtree rooted by node Δ.

4.2 Generating Regular Unfolding Trees

Regular unfolding trees are generated via procedure ω-$\mathtt{SAT}$, described in Algorithm 1. Given a formula Δ_0, ω-$\mathtt{SAT}$ creates an initial tree with one open node Δ_0. Then, it iteratively applies the following procedures until all leaf nodes are marked as closed.

1. Leaf Node Evaluation via procedure $base\_eval$ (line 3). It checks satisfiability for every *base* leaf node and marks them *closed* accordingly.
2. Back-link Construction via procedure $link\_back$ (line 4). It attempts to link an open leaf node with an ancestor via some equisatisfiability and substitution principles.
3. Reduction. It chooses an occurrence of inductive predicates in an open leaf node (line 5) to unfold (a.k.a. instantiate - line 9) in a breadth-first manner.

$base\_eval$ makes use of the following procedure $\mathtt{eXPure}$ to discharge a base formula. $\mathtt{eXPure}$ transforms a separation logic formula to a formula in first-order logic. Given a base formula $B \equiv \exists \bar{w}.\ \ast_{i=1}^{n}\ array(v_i, t_i) * \ast_{i=1}^{m} x_i{\mapsto}\{\bar{y}_i\} \wedge \pi$, $\mathtt{eXPure}$

Algorithm 1: Procedure ω-SAT

```
    input  : $\Delta_0$
    output: $\mathcal{T}$
1   $\mathcal{T}\leftarrow\{\Delta_0\}$ ;                                   /* initialize */
2   while true do
3       $\mathcal{T}\leftarrow base\_eval(\mathcal{T})$ ;                           /* eval bases */
4       $\mathcal{T}\leftarrow link\_back(\mathcal{T})$ ;                  /* generate back-link */
5       (is_exists, $\Delta_i$)$\leftarrow$choose_bfs($\mathcal{T}$) ;  /* open leaf for unfolding */
6       if not is_exists then
7           return $\mathcal{T}$;
8       else
9           $\mathcal{T}\leftarrow$unfold($\Delta_i$);
10  end
```

works as follows. If $\pi \Rightarrow \bigvee_{1\leqslant i\leqslant n} v_i{=}\texttt{null} \vee t_i{=}\texttt{null} \vee \bigvee_{1\leqslant i\leqslant m} x_i{=}\texttt{null}$, then $\pi_B = \textbf{eXPure}(B) \overset{\text{def}}{=} \texttt{false}$. Otherwise,

$$\pi_B = \textbf{eXPure}(B) \overset{\text{def}}{=} \exists\ \bar{w}.\ \bigwedge_{1\leqslant i\leqslant n} v_i{\leqslant}t_i \wedge \bigwedge_{1\leqslant i<j\leqslant n}(t_i{<}v_j) \vee (t_j{<}v_i) \wedge \\ \bigwedge_{1\leqslant i\leqslant n,1\leqslant j\leqslant m}(x_j{<}v_j) \vee (t_i{<}x_j) \wedge \bigwedge\{x_i{\neq}x_j \mid i,j{\in}\{1...m\} \text{ and } i{\neq}j\} \wedge \pi$$

Lemma 1. *For any stack* s *and base formula* B, $s \models \textbf{eXPure}(B)$ *iff* $\exists h.\ s,h \models B$.

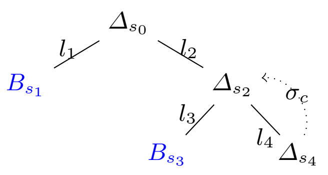
The procedure $link\_back$ was designed based on the spatial part of the formulas. As we show that satisfiability in (dis)equalities relies only on quantifier-free variables, existentially quantified heaps could be discarded. Particularly, a leaf node Δ_b, say $\Delta_b \equiv \exists\bar{w}.\ \Delta_{b_1} * \Delta_{b_2} * \kappa_d$, is linked back to an internal node Δ_c only when:

1. every heap predicates in $(\exists\bar{w}.\ \Delta_{b_2})^S$ are unobservable; and
2. κ_d contains duplicate inductive predicate occurrences. Given $\Delta_{b_1} \equiv \kappa_{b_1} \wedge \pi_{b_1}$, for every inductive predicate $\texttt{P}(\bar{v})$ in κ_d, there exists a substitution σ over a subset of existentially quantified variables of $\bar{v}$ such that $(\texttt{P}(\bar{v}))^S\sigma$ is in $(\kappa_{b_1})^S$; and
3. there exists a substitution $\sigma_c \equiv \underline{[t_1/v_1,..,t_n/v_n]}$ where v_i, t_i ($i \in \{1...n\}$) are existentially quantified such that $\overline{(\Delta_c)^S} \equiv (\exists\bar{w}.\ \overline{(\Delta_{b_1})^S})\sigma_c$.

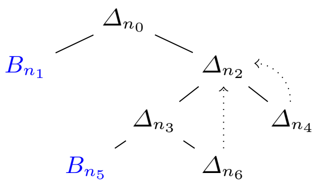
Properties of $\boldsymbol{\omega}$-SAT over Spatial Projection We now show some properties of ω-SAT over fragment SHID, a fragment of array separation logic with spatial-only definitions of inductive predicates. These properties are fundamental for compositionality.

Definition 4 (Fragment SHID). *Every inductive symbol* $\texttt{P}_\texttt{i} \in \mathcal{P}$ *in* SHID *is defined as:* $\texttt{P}_\texttt{i}(\bar{v}_i) \equiv \bigvee_{j=1}^{m}(\exists\bar{w}_{i_j}.\ \kappa_{i_j} \wedge \pi_{i_j})$ *where* π_{i_j} *(*$1{\leqslant}j{\leqslant}m$*) are (dis)equalities.*

For every back-link with a companion Δ_c and a bud Δ_b, if Δ_b is *satisfiable* then every formula derived from unfolding Δ_b is of the form $B{*}B_r$ and there exists a substitution σ such that $B\sigma$ is a leaf node of the subtree rooted by Δ_c and B_r is unobservable.



(a) Unfolding Tree for `sll`



(b) Unfolding Tree for `nll`

Fig. 3: Regular Unfolding Trees

Proposition 1 (Completeness). *For any s, h and a back-link with a companion Δ_c and a bud Δ_b, if $s, h \models \Delta_b$, then $\exists B \in \mathtt{base}^{\mathcal{P}}(\Delta_c)$ and $\exists s' \subseteq s, h' \subseteq h$ s.t. $s', h' \models B$.*

We show the small model property of the bases generated.

Lemma 2 (Small Model). *For any s, h and base satisfiable formula $B{*}B_r$ where B_r is unobservable, if $s, h \models B$, then $\exists s' \supseteq s, h' \supseteq h$. $s', h' \models B * B_r$.*

ω-SAT with `link_back` always terminates over `SHID` as the numbers of both definitions in the system and quantifier-free variables in a formula are finite.

Proposition 2 (Termination). *ω-SAT terminates in* `SHID`.

An unfolding tree is a cyclic proof only when every leaf nodes are either unsatisfiable or linked back. A cyclic proof is generated as a witness for unsatisfiability.

Proposition 3 (Soundness). *If Δ has a cyclic proof, Δ is unsatisfiable.*

Example 2. We illustrate ω-SAT over shape-only singly-linked list `sll` and nested lists `nll`, the spatial projections of the ones in Sect. 2, without arithmetic properties.

$$\mathtt{sll}(x{,}y) \equiv \mathtt{emp}{\wedge}x{=}y \;\vee\; \exists u.x{\mapsto}\{\_{,}u\}{*}\mathtt{sll}(u{,}y){\wedge}x{\neq}y$$
$$\mathtt{nll}(x{,}y{,}b) \equiv \mathtt{emp}{\wedge}x{=}y \vee \exists\, u{,}Z.x{\mapsto}\{\_{,}u{,}Z\} * \mathtt{sll}(Z{,}b) * \mathtt{nll}(u{,}y{,}b){\wedge}x{\neq}y{\wedge}x{\neq}b$$

Figure 3a shows unfolding tree for `sll`. This tree is constructed as follows. Starting from $\Delta_{s_0} \equiv \mathtt{sll}(x{,}y)$, ω-SAT unfolds the inductive symbol to obtain B_{s_1} and Δ_{s_2}.

$$B_{s_1} \equiv \mathtt{emp} \wedge x{=}y \qquad\qquad \Delta_{s_2} \equiv \exists u_1.x{\mapsto}\{\_{,}u_1\}{*}\mathtt{sll}(u_1{,}y){\wedge}x{\neq}y$$

with two new edges whose the labels are as follows.

$$l_1 \equiv (\mathtt{sll}(x{,}y), \mathtt{emp} \wedge x{=}y) \qquad l_2 \equiv (\mathtt{sll}(x{,}y), \exists u_1.x{\mapsto}\{\_{,}u_1\}{*}\mathtt{sll}(u_1{,}y){\wedge}x{\neq}y)$$

ω-SAT evaluates B_{s_1} as satisfiability and marks it as closed. For Δ_{s_2}, ω-SAT unfolds the inductive predicate to obtain $B_{s_3} \equiv x{\mapsto}\{\_{,}y\}{\wedge}x{\neq}y$ and the following Δ_{s_4}.

$$\Delta_{s_4} \equiv \exists u_1{,}u_2.x{\mapsto}\{\_{,}u_1\}{*}u_1{\mapsto}\{\_{,}u_2\}{*}\mathtt{sll}(u_2{,}y){\wedge}x{\neq}y{\wedge}u_1{\neq}y$$

The two new edges have the following labels.

$$l_3 \equiv (\mathtt{sll}(u_1{,}y), \mathtt{emp} \wedge x{=}y) \quad l_4 \equiv (\mathtt{sll}(u_1{,}y), \exists u_2.u_1{\mapsto}\{\_{,}u_2\}{*}\mathtt{sll}(u_2{,}y){\wedge}u_1{\neq}y)$$

Δ_{s_4} is linked back to Δ_{s_2} and marked as closed since Δ_{s_4} could be rearranged as: $\Delta_{s_4} \equiv \exists u_1,u_2.\Delta_{b_1} * \Delta_{b_2}$ where $\Delta_{b_1} \equiv x{\mapsto}\{\_,u_1\}{*}\texttt{sll}(u_2,y){\wedge}x{\neq}y$ and $\Delta_{b_2} \equiv u_1{\mapsto}\{\_,u_2\}{\wedge}u_1{\neq}y$ s.t. (i) $\exists u_1,u_2.\Delta_{b_2}$ is unobservable and (ii) $\Delta_{s_2} \equiv (\exists u_1,u_2.\Delta_{b_1})\sigma_c$ where $\sigma_c \equiv [u_1/u_2]$. That means, if Δ_{s_4} had been kept unfolding, its sub-tree would have included an infinite set of base formulas each of which has the same observable heap with B_{s_3} i.e., of the form $x{\mapsto}\{\_,\_\}{\wedge}x{\neq}y * B_r$ where B_r is unobservable. Obviously, models satisfying B_{s_3} are the smallest and have finite heap domains. Since all leave nodes are marked as closed, ω-$\texttt{SAT}$ terminates.

Similarly, Fig. 3b shows the unfolding tree for `nll` whose details are as follows. $\Delta_{n_0} \equiv \texttt{nll}(x,y,b)$, $B_{n_1} \equiv \texttt{emp} \wedge x{=}y$, $B_{n_5} \equiv x{\mapsto}\{\_,y,b\}{\wedge}x{\neq}y{\wedge}x{\neq}b$

$\Delta_{n_2}{\equiv}\exists\ u_1,Z_1.x{\mapsto}\{\_,u_1,Z_1\} * \texttt{sll}(Z_1,b) * \texttt{nll}(u_1,y,b){\wedge}x{\neq}y{\wedge}x{\neq}b$

$\Delta_{n_3} \equiv \exists\ u_1.x{\mapsto}\{\_,u_1,b\} * \texttt{nll}(u_1,y,b){\wedge}x{\neq}y{\wedge}x{\neq}b$
$\Delta_{n_4} \equiv \exists\ u_1,Z_1,Z_2.x{\mapsto}\{\_,u_1,Z_1\} * Z1{\mapsto}\{\_,Z_2\} * \texttt{sll}(Z_2,b) * \texttt{nll}(u_1,y,b)$
$\quad{\wedge}x{\neq}y{\wedge}x{\neq}b{\wedge}Z_1{\neq}b$
$\Delta_{n_6} \equiv \exists\ u_1,\ u_2,Z_2.x{\mapsto}\{\_,u_1,b\} * u_1{\mapsto}\{\_,u_2,Z_2\} * \texttt{sll}(Z_2,b){*}\texttt{nll}(u_2,y,b)$
$\quad{\wedge}x{\neq}y{\wedge}x{\neq}b{\wedge}u_1{\neq}y{\wedge}u_1{\neq}b$

5 Compositional Satisfiability Solver

`S2S` compositionally discharges a formula as follows. First, it computes for every inductive predicate $\texttt{P}(\bar{t})$ a base, denoted as $\texttt{base}^{\mathcal{P}}(\texttt{P}(\bar{t}))$ - a set of *satisfiable* base formulae, that precisely charaterises its satisfiability. If this set is empty, then $\texttt{P}(\bar{t}) \equiv \texttt{false}$. After that, to discharge formula Δ it replaces every occurrence of inductive predicates with the corresponding base to obtain a disjunctive base formula, denoted by $\texttt{base}^{\mathcal{P}}(\Delta)$, before using procedure `eXPure` to transform this base formula into π_B in first-order logic. Finally, π_B could be discharged efficiently by using an SMT solver.

In the rest of this section, we first present the algorithm to collect the bases (subsection 5.1). After that, we identify five semantic conditions for decidability and compositionality (subsection 5.2). Finally, we show a syntactic decidable fragment, an extension of `SHID`, where the satisfiability solving can be compositional (subsection 5.3).

5.1 Computing Bases

Algorithm 2 describes how to infer bases for inductive predicates. In intuition, for each inductive predicate, it first computes a regular unfolding tree where cycles are generated based on the spatial projection of buds and companions. After that, for every cycle, it infers bases for the spatial projection and closed form for the numeric projection separately. Finally, it conjoins the two bases. In particular, it first generates an unfolding tree at line 2. (We assume that all subtrees whose leaf nodes are either unsatisfiable or linked back are eliminated afterward.) At line 4, `out−most cycle` in each path is the one which has the farthest companion from the root. At line 6, it collects numeric parts of all buds (each cycle has one companion and one or more buds). Next, for each cycle, it collects the spatial projection and numeric projection of all *satisfiable* leaf nodes (lines 7-9). If spatial projection of all base leaf nodes is unsatisfiable, it returns unsatisfiable (line 12). Every base formula generated by the cycle is equisatisfiable

Algorithm 2: Deriving Bases.

```
input  : $\mathcal{P}$
output: $\texttt{base}^{\mathcal{P}}$
1  foreach $\texttt{P}_\texttt{i}(\bar{t}_i) \in \mathcal{P}$ do
2      $(V, E, \mathcal{C}) \leftarrow \omega\texttt{-SAT}(\texttt{P}_\texttt{i}(\bar{t}_i))$ ;                /* reduction tree */
3      repeat
4          $\mathcal{C}(\Delta_c \rightarrow \Delta_b, \sigma) \leftarrow$ out-most cycle of $(V, E, \mathcal{C})$ ;
5          $\Psi^S \leftarrow \emptyset$ ;
6          $\alpha \leftarrow \bigvee\{\Delta^N_{b_i}\}$ ;                /* arithmetic of sat buds */
7          foreach $\Delta^S_{\texttt{sat}} \in \mathcal{C}(\Delta_c \rightarrow \Delta_b, \sigma)$ do
8              $\Psi^S \leftarrow \Psi^S \cup \{\Delta^S_{\texttt{sat}}\}$ ;        /* spatial of sat leaf bases */
9              $\alpha \leftarrow \alpha \vee \Delta^N_{\texttt{sat}}$ ;        /* arithmetic of sat leaf bases */
10         end
11         if $\Psi^S = \emptyset$ then
12             $\Delta_c \leftarrow \texttt{false}$ ;                /* unsat - cyclic proofs */
13         if $\Delta^N_c$ contains one inductive predicate then
14             $\beta \leftarrow Pres(\Delta^N_c)$ ;                /* $\Delta^N_c \equiv \alpha$ */
15         else
16             $\beta \leftarrow Pres(\alpha)$ ;
17         $\Delta_c \leftarrow \bigvee\{B \wedge \beta \mid B \in \Psi^S\}$ ;
18     until no more cycles;
19     $\texttt{base}^{\mathcal{P}}((\texttt{P}_\texttt{i}(\bar{t}_i))) \leftarrow \{\Delta^{noncyc}_{\texttt{sat}} \mid \Delta^{noncyc}_{\texttt{sat}}$ is $\texttt{sat}\}$ ;   /* sat leaf nodes */
20 end
21 return $\texttt{base}^{\mathcal{P}}$;
```

with one of those in the set of all spatial bases collected. For the numeric, it computes the closed form of satisfiable instances (lines 13-16). (Recall that $Pres$ is the function that maps numeric projection of each inductive definition in the decidable fragment to a Presburger formula.) Note that if the numeric companion Δ^N_c is an occurrence of an inductive predicate, it computes the closed form using a more precise on-the-fly definition ($\Delta^N_c \equiv \alpha$ at line 14). Finally, at line 17, it replaces the companion with the combined base. This process of computing bases for cycles is repeated in such a bottom-up manner until the tree does not contain any cycles. Finally, it collects all the satisfiable leaf nodes of the tree.

Example 3. For the inductive predicates in Example 3, the base computed for the cycle of `sll` is $(B_{s_3})^S = \exists Y.\ x{\mapsto}\{\_,Y\} \wedge x{\neq}y$ and the base of generated for `sll` is a disjunctive set of the two satisfiable base leaf nodes: $\texttt{base}^{\mathcal{P}}(\texttt{sll}(x,y)) \equiv \{B_{s_1}; (B_{s_3})^S\}$. Similarly, the base computed for `nll` is: $\texttt{base}^{\mathcal{P}}(\texttt{nll}(x,y,b)) \equiv \{B_{n_1}; (B_{n_5})^S\}$. □

5.2 Decidability and Compositionality

We state the five conditions for a fragment of inductive predicates such that Algorithm 2 is terminating, and the generated bases are both sound and complete. First, the following condition ensures the separation of the spatial and numeric projections such that there

is no over-approximation of the two projections. Suppose that the pure formula π of a definition rule in $\mathtt{SLIDLIA}_{\mathtt{sem}}$ is $\pi \equiv \alpha \wedge \beta \wedge \gamma$ where α is the spatial constraint, β is the arithmetic constraint and γ is the mixing constraint between the two domains.
C1. γ *is* `true`. We note that if all inductive definitions of $\mathtt{P}(\bar{t})$ in a fragment satisfy **C1**, then such γ of any formula derived from unfolding $\mathtt{P}(\bar{v})$ is also `true`.

For the termination of ω-`SAT` at line 2, we need the following condition.
C2. α *in every definition rule in* $\mathtt{SLIDLIA}_{\mathtt{sem}}$ *is a conjunction of (dis)equalities.*

The completeness further requires the three following conditions.

– Every back-link generated for the spatial projection is (sound and) complete when combined with pure bases. Recall that a bud is of the form $\Delta_b \equiv \exists\bar{v}.(\kappa_{b_1} \wedge \alpha_{b_1} \wedge \beta_{b_1}) * (\kappa_{b_2} \wedge \alpha_{b_2} \wedge \beta_{b_2}) * \kappa_d$ where $\kappa_{b_2} \wedge \alpha_{b_2} \wedge \beta_{b_2}$ is unobservable and κ_d is the duplicate conjunction. As ω-`SAT` always returns the observable part, unlike in `SHID`, discarding $(\kappa_{b_2} \wedge \alpha_{b_2} \wedge \beta_{b_2} * \kappa_d)^S$ may make the combined bases incomplete; that is $(\Delta_b)^S$ is unsatisfiable while $(\Delta_b)^N$ is satisfiable. The completeness is retained only when: For any s, we have if $s \models (\Delta_b)^N$, then $\exists\ s', h'.\ s',h' \models (\Delta_b)^S$. As ω-`SAT` always returns the observable part, the following condition is equivalent.
  C3. For any s, h, we have: if $s, h \models (\Delta_{b_1})^S$, then $\exists\ s', h'.\ s',h' \models (\Delta_b)^S$.
– **C4.** *If the system of inductive definitions contains arithmetic constraints, each cycle in the regular unfolding tree derived for an inductive definition contains at most one* satisfiable *spatial projection leaf node* (line 8). This condition forbids the over-approximation of the combination at line 17.
– **C5.** *Numeric projection of every inductive predicate (at line 16) or on-the-fly numeric predicate (at line 14) is Presburger-definable* i.e., the numeric predicate is in a decidable fragment like DPI [39]. This ensures that the numeric base computed is equivalent to the numeric constraints of the whole subtree.

Suppose Algorithm 2 infers a base $B^S \wedge \pi$ for subtree involving a cycle and $B * B_r$ is a base leaf of the subtree. By Lemma 2, for every v where $s(v) \in dom(h' \backslash h)$, v is existentially quantified. Hence, the heaps in $h' \backslash h$ could not be accessed by the outer scope of B_r. As so, for any formula Δ, $B * B_r * \Delta$ is equisatisfiable with $(B^S \wedge \pi) * \Delta$. Therefore, satisfiability could be performed modularly via the inferred base $B^S \wedge \pi$.

Theorem 1 (Composition). *For any stack s, heap h and Δ, we have:*

– *(if) if $s, h \models \mathtt{base}^{\mathcal{P}}(\Delta)$, then $\exists s', h'.\ s', h' \models \Delta$.*
– *(only if) if $s, h \models \Delta$, then $\exists s', h'.\ s', h' \models \mathtt{base}^{\mathcal{P}}(\Delta)$.*

Motivating Example Revisited We show how to compute the bases of predicate $\mathtt{sllss}(x,y,mi,ma,n,n_0)$ in the second motivating example. In the definition of this predicate, x, y are spatial variables and mi, ma, n and n_0 are numeric variables. **C1** and **C2** hold straightforwardly for this definition.

At line 2, ω-`SAT` constructs a reduction tree whose shape is similar to the tree in Fig. 3a. Its respective nodes are as follows. $\Delta_{ss_0} \equiv \mathtt{sllss}(x,y,mi,ma,n,n_0)$

$$
\begin{aligned}
&B_{ss_1} \equiv \mathtt{emp} \wedge x{=}y \wedge mi{=}ma \wedge n{=}n_0\\
&\Delta_{ss_2} \equiv \exists u_1,m_1,n_1.x{\mapsto}\{mi,u_1\}{*}\mathtt{sllss}(u_1,y,m_1,ma,n_1,n_0) \wedge x{\neq}y \wedge mi{<}m_1 \wedge n{=}n_1{+}1\\
&B_{ss_3} \equiv x{\mapsto}\{mi,y\} \wedge x{\neq}y \wedge mi{<}ma \wedge n{=}n_0{+}1\\
&\Delta_{ss_4} \equiv \exists u_1,u_2,m_1,m_2,n_1,n_2.x{\mapsto}\{mi,u_1\}{*}u_1{\mapsto}\{m_1,u_2\}{*}\mathtt{sllss}(u_2,y,m_2,ma,n_2,n_0)\\
&\qquad \wedge x{\neq}y \wedge u_1{\neq}y \wedge mi{<}m_1 \wedge m_1{<}m_2 \wedge n{=}n_1{+}1 \wedge n_1{=}n_2{+}1
\end{aligned}
$$

In Δ_{ss_4}, $\Delta^S_{b_1} \equiv x{\mapsto}\{mi,u_1\}{*}\mathtt{sllss}^{\mathtt{S}}(u_2,y)$, $\kappa_d \equiv \mathtt{emp}$ and the spatial projection of the unobservable is $\Delta^S_{b_2} \equiv u_1{\mapsto}\{m_1,u_2\} \wedge u_1{\neq}y$. As $\overline{\Delta^S_{b_2}}$ is separate from $\overline{\Delta^S_{b_1}}$, **C3** holds. And as the cycle in the reduction tree has only one base B_{ss_3}, **C4** holds. As shown in the preceding subsection, its numeric projection is in DPI and is thus Presburger-definable: $Pres(\mathtt{sllss}^{\mathtt{N}}(mi,ma,n,n_0)) \equiv mi{\leqslant}ma \wedge n{\geqslant}n_0$. Thus, **C5** holds. Moreover, the base for the cycle of the tree is computed as $B_{22} \equiv (B_{ss_3})^S \wedge Pres((\Delta_{ss_3})^N \vee (\Delta_{ss_4})^N)$,

$$
\begin{aligned}
&B_{22} \equiv (B_{ss_3})^S \wedge Pres((\Delta_{ss_2})^N)\\
&\equiv \exists Y.x{\mapsto}\{mi,Y\} \wedge x{\neq}y \wedge Pres(\exists m_1,n_1.\mathtt{sllss}^{\mathtt{N}}(m_1,ma,n_1,n_0) \wedge mi{<}m_1 \wedge n{=}n_1{+}1)\\
&\equiv \exists Y,m_1,n_1.x{\mapsto}\{mi,Y\} \wedge x{\neq}y \wedge (m_1{\leqslant}ma \wedge n_1{\geqslant}n_0 \wedge mi{<}m_1 \wedge n{=}n_1{+}1)
\end{aligned}
$$

Finally, the base computed for `sllss` is: $\mathtt{base}^{\mathcal{P}}(\mathtt{sllss}(x,y,mi,ma,n,n_0)) \equiv \{B_{ss_1}; B_{22}\}$.

The remaining question is syntactic decidable fragments where the satisfiability is compositional. `SHID` satisfies the five conditions for compositionality straightforwardly. As definitions of inductive predicates in [3,14,18,39,21,23,41] satisfy these five conditions, satisfiability is compositional in these fragments. The next subsection shows a new decidable fragment.

5.3 Compositionality with Small-Model Arithmetic Properties

We study fragment `SHIDe` that is an extension of `SHID` with small-model arithmetic pure properties (e.g., sortedness) where every inductive predicate also has small models w.r.t. satisfiability. Given a predicate $\mathtt{P_i}(\bar{v}_i)$, if two parameters $v,t \in \bar{v}_i$ define a small-model pure property then in every instantiation unfolded from $\mathtt{P_i}(\bar{v}_i)$, the constraints over v,t is: $\exists w_1,...,w_n.v \diamond w_1 \wedge w_1 \diamond w_2 \wedge ... \wedge w_n \diamond t$ (where $\diamond \in \{=,\geqslant,\leqslant\}$).

Definition 5 (`SHIDe`)**.** *Given every definition* $\mathtt{P_i}(\bar{v}_i){\equiv}\bigvee_{j=1}^{m}(\exists \bar{w}_{i_j}.\ \kappa_{i_j} \wedge \pi_{i_j})$ *in* `SHID`*,* π_{i_j} *(*$1{\leqslant}j{\leqslant}m$*) may contain* $\diamond$ *operators over parameters of inductive predicate* $\mathtt{P_i}$ *such that for any* s,h *if* $s,h \models \mathtt{P_i}(\bar{v}_i)$*,* $\forall l_1{\in}dom(h).\exists l_2{\in}dom(h)$ *s.t.* $h(l_1){=}(..,l_2,..,l_{j_1},..)$*,* $h(l_2){=}(..,l_{j_2},..)$ *and* $l_{j_1} \diamond l_{j_2}$ *holds where* l_{j_1} *and* l_{j_2} *are the* j^{th} *components.*

Example 4. We define linked lists being sorted as follows.

$$
\begin{aligned}
&\mathtt{sllso}(x,y,mi,ma) \equiv \mathtt{emp} \wedge x{=}y \wedge ma{=}mi\\
&\quad \vee\ \exists u,mi_1.x{\mapsto}\{mi,u\} * \mathtt{sllso}(u,y,mi_1,ma) \wedge x{\neq}y\ \wedge\ mi{<}mi_1
\end{aligned}
$$

For any formula B unfolded from $\mathtt{sllso}(x,y,mi,ma)$, in case B has an empty heap, $mi{=}ma$. Otherwise, $mi{<}ma$. Hence, the base that includes one with the empty heap and another with *one* singleton heap is sufficient to characterise the satisfiability of $\mathtt{sllso}(x,y,mi,ma)$. Particularly, to compute base for $\mathtt{sllso}(x,y,mi,ma)$, ω-`SAT` constructs for it a cyclic reduction tree that has the same structure as the tree of $\mathtt{sll}(x,y)$ (in Example 3). □

Obviously, ω-`SAT` terminates to compute bases for a definition in `SHIDe` that is a combination of a definition in `SHID` with the small-model pure properties.

6 Decidable Fragment `SLIDLIA`

We define a syntactic fragment, called `SLIDLIA` (subsection 6.1). The decidability and compositionality of `SLIDLIA` rely on the decidability of its numeric inductive predicates. In subsection 6.2, we show that `AID` - Arithmetic with Inductive Definitions, the fragment including these numeric projections, is indeed decidable.

6.1 Predicate Definition

A predicate in `SLIDLIA` with one pair of numeric parameters is defined as:

$$\begin{aligned} \mathtt{P}(r,F,\bar{B},v_s,v_t) \equiv & \mathtt{emp} \wedge r{=}F \wedge v_s{=}v_t \\ & \vee\ \exists X_{tl}, \bar{Z}, v_s', \bar{Z}_s. r{\mapsto}\{\bar{p}\} * \kappa' * \mathtt{P}(X_{tl},F,\bar{B},v_s',v_t) \wedge \alpha \wedge \beta \end{aligned}$$

where r is called the root parameter, F the target parameter, $\bar{B}$ the border parameters, v_s, v_t is a pair of parameters to capture a pure property, $r{\mapsto}\{\bar{p}\} * \kappa'$ the matrix of the heaps. r, F and $\bar{B}$ are spatial variables and v_s, v_t are numeric variables. Moreover, this definition is constrained by the following five conditions.
Y1. $\{X_{tl}\} {\cup} \bar{Z} \subseteq \bar{p} \subseteq \{X_{tl}\} {\cup} \bar{Z} {\cup} \bar{B}$.
Y2. $\kappa'{:=}\mathtt{Q}(R,U,\bar{Y},S,T) \mid \kappa'{*}\kappa' \mid \mathtt{emp}$ where $\mathtt{Q} \not\equiv \mathtt{P}$, $R \in \bar{Z}$ and for any $\mathtt{Q}_1(R_1,...) \in \kappa'$ and $\mathtt{Q}_2(R_2,...) \in \kappa'$ then $R_1 \not\equiv R_2$, $U \in \bar{Z} {\cup} \bar{B} {\cup} \{r, X_{tl}, \mathtt{null}\}$, $\bar{Y} \subseteq \bar{B} {\cup} \{r, X_{tl}, \mathtt{null}\}$, and $S \in \bar{Z}_s$ and $T \in \bar{Z}_s \cup \{v_s'\}$.
Y3. α is a conjunction of (dis)equalities and $FV(\alpha) \subseteq \{r, F, \mathtt{null}\} \cup \bar{B}$.
Y4. β is of the one of the following forms:

- $\beta \equiv \beta' \wedge v_s{=}v_s'{+}c_1 z{+}c_2$ where $c_1, c_2 \in \mathbb{Z}$, $FV(\beta') \subseteq \bar{Z}_s$, and $z \in \bar{Z}_s$.
- $\beta \equiv \beta' {\wedge} v_s {\diamond} v_s'{+}c_1$ where $c_1 \in \mathbb{Z}$, $\diamond \in \{=, \geqslant, \leqslant\}$, $FV(\beta') \subseteq \bar{Z}_s$.

Y5. There is no mutual recursion.

The extension to multiple pairs of numeric parameters is straightforward. `SLIDLIA` is subsumes the decidable fragments presented in [3,13,14,18,21,23]. $\mathtt{SLIDLIA} \subset \mathtt{SLIDLIA}_{\mathtt{sem}}$ because **Y3** and **Y4** imply **C1**; **Y3** ensures **C2**; **Y2** implies **C3**; **Y1** and **Y2** imply **C4**. We show **C5** in the next subsection. `SLIDLIA` includes `sllss`, `nllss` (shown in Sect. 2), skip-lists, nested lists.

6.2 Decidability of Fragment `AID`

We show a procedure to compute the closed form of the numeric projections of definitions in `SLIDLIA`. Recap that the numeric projection of a definition is of the form:

$$\mathtt{P}^{\mathtt{N}}(v_s,v_t) \equiv v_s{=}v_t\ \vee\ \exists v_s', \bar{z}. \bigwedge_{i=0}^{n} \mathtt{P}^{\mathtt{N}}_{\mathtt{i}}(S_i,T_i) \wedge \mathtt{P}^{\mathtt{N}}(v_s',v_t) {\wedge} \beta' \wedge \beta_0$$

where $\beta_0 \equiv v_s {\diamond} v_s'{+}c$ or $\beta_0 \equiv v_s{=}v_s'{+}c_1 m{+}c_2$, $\diamond \in \{\geqslant, \leqslant\}$, $\bigwedge_{i=0}^{n} \mathtt{P}^{\mathtt{N}}_{\mathtt{i}}(S_i,T_i) \equiv \mathtt{true}$ when $n{=}0$, $S_i, T_i \in \bar{z}$, and $FV(\beta') \subseteq \bar{z}$.

As definitions in `SLIDLIA` do not allow mutual recursion (condition **Y5**), the computation of the closed form of these numeric definitions can be performed in a bottom-up manner: the closed forms of all `pred` $\mathtt{P}^{\mathtt{N}}_{\mathtt{i}}(S_i,T_i)$ are computed before the computation of the closed formula of `pred` $\mathtt{P}^{\mathtt{N}}(v_s,v_t)$. The computation of the closed formula is based on the two forms of β_0 above. First, we show the computation for the first form.

Lemma 3. *Given any numeric projection in the following form*

$$\mathtt{P}^{\mathtt{N}}(v_s,v_t) \equiv v_s{=}v_t \;\vee\; \exists v_s',\bar{z}.\mathtt{P}^{\mathtt{N}}(v_s',v_t) \wedge \beta'(\bar{z}) \wedge v_s \diamond v_s'{+}c$$

where $\diamond \in \{=, \geqslant, \leqslant\}$*, and* $c \in \mathbb{Z}$*. Then, we have:*

$$\mathtt{P}^{\mathtt{N}}(v_s,v_t) \equiv v_s{-}v_t \vee \exists \bar{z}, k.\beta'(\bar{z}) \wedge v_s{-}v_t \diamond ck \wedge k{\geqslant}1$$

The computation for the second form of β_0 is based on the arithmetic of addition and divisibility [2,4,24]. Authors in [2,24] show that the formulas of the form $\exists \bar{v}. \bigwedge_{i=1}^{K} f_i(\bar{v}) \mid g_i(\bar{v})$ are decidable, where f_i, g_i are linear functions over $\bar{v} \equiv \{v_1, .., v_n\}$ and the symbol $\mid$ means that each f_i is an integer divisor of g_i when both are interpreted over $\mathbb{N}^n$. Recently, authors in [4] presented a decision procedure for an extension with universally quantified formulas. They proposed to eliminate the quantifiers and transform the formulas in the language $\langle +, |, 0, 1 \rangle$ into Presburger arithmetic.

Proposition 4 ([4]). *The following formula is Presburger-definable:*

$$QzQ_1x_1...Q_nx_n. \bigvee_{i=1}^{N}(\bigwedge_{j=1}^{M_i} h_{ij}(z) \mid f_{ij}(\bar{x}, z) \wedge \bigwedge_{j=1}^{M_i} h'_{ij}(z) \nmid g_{ij}(\bar{x}, z) \wedge \pi(\bar{x}, z))$$

where Q, Q_1*,..,*$Q_n \in \{\exists, \forall\}$*,* π *is quantifier-free, f, f′, h, g are linear functions.*

Secondly, we show that the closed form for the second form of β_0 is in the arithmetic of addition and divisibility.

Lemma 4 (Nested Quantitative Property). *Given a numeric projection of the form:*

$$\mathtt{P}^{\mathtt{N}}(v_s,v_t) \equiv v_s{=}v_t \;\vee\; \exists v_s',\bar{z}.\mathtt{P}^{\mathtt{N}}(v_s',v_t) \wedge \beta' \wedge v_s{=}v_s'{+}c_1z{+}c_0$$

where FV(β*)*$\subseteq\bar{z}$ *and* $z{\in}\bar{z}$*,* $c_0, c_1{\in}\mathbb{Z}$*. Then,* $\mathtt{P}^{\mathtt{N}}(v_s,v_t) \equiv \exists \bar{z}.\beta'(\bar{z}) \wedge c_1z{+}c_0 \mid v_s{-}v_t$*.*

By Proposition 4, $\exists \bar{z}.\beta(\bar{z}) \wedge c_1z{+}c_0 \mid v_s{-}v_t$ is Presburger-definable. Hence, Lemma 3, Lemma 4 and Theorem 4 imply that $\mathtt{P}^{\mathtt{N}}(v_s,v_t)$ is Presburger-definable.

Theorem 2. *Numeric projections of definitions in* `SLIDLIA` *is Presburger-definable.*

7 Implementation and Evaluation

We have implemented a prototype tool, called `S2S`, using OCaml for the satisfiability problems. We made use of Z3 [11] as a back-end SMT solver for the arithmetic.

`S2S` gives a precise answer to those problems in a fragment that satisfies the five conditions in Sect. 5.2. For those that are beyond these conditions, `S2S` infers over-approximated bases to check their unsatisfiability. In particular, if **C1.** or **C2.** is violated, i.e., formulas with pointer arithmetic constraints, `S2S` discards these constraints. For efficiency, checking the satisfiability of buds to comply with **C3.** can be ignored. If **C4.** or **C5.** is violated, i.e., an arithmetically inductive definition is beyond the decidable fragment `AID`, `S2S` computes for it an over-approximated closed form using the

Table 1: Satisfiability Solving with/without Compositional Reasoning

Data Structure (pure properties)	#query	#unsat	#sat	*without*		*with*	
				#Z3	Time	#Z3	Time
Singly llist (size)	666	75	591	3,173	1.01	762	0.40
Sorted llist (size, sorted)	217	21	196	796	0.55	336	0.36
Doubly llist (size)	452	50	402	1,803	0.79	552	0.46
Heap trees (size, maxelem)	386	38	348	3,732	6.03	865	2.61
AVL (height, size)	881	64	817	9,051	23.06	2,026	10.85
RBT (size, blackheight, color)	1,741	217	1,524	3,491,730	74,158	1,767	2.81
rose-tree (size)	25	8	17	300	0.34	153	0.25
	4,368	473	3,895	3,510,585	74,189.78	6,461	17.74

technique described in [21]. In intuition, for each definition, it first generates a set of Horn clauses to capture the least fixed point set of its values. After that, it uses the fixed point analysis in [34,40] to solve these clauses.

To demonstrate the efficiency, we have evaluated `S2S` on two sets of satisfiability benchmarks. The first one includes those generated by the program verifier S2 [19,22] (subsection 7.1), and the second one consists of those taken from the recent competition for separation logic solvers SL-COMP 2019 [37,38] (subsection 7.2). All experiments were performed on a machine with Intel Core i7-6700 3.4GHz and 8GB RAM. If a solver runs longer than 600 seconds, we terminate it and mark the result as unknown.

7.1 Efficiency of Compositional Solving

In this section, we present experiments over satisfiability problems with and without compositional reasoning. These problems come from the symbolic execution of the heap-based verification tool [19]. Each test suit consists of a high number of test problems over the same set of inductive predicates. Then we run `S2S` over the suite in two settings. For the first one, `S2S` generates bases for each input without reusing the bases inferred for inductive definitions. For the second one, `S2S` generates bases for each test by reusing the bases of inductive definitions.

The experimental results are shown in Table 1. The first column shows the names of inductive predicates and pure properties that includes cyclic linked-list, sorted singly-linked list, doubly-linked list, AVL trees, red-black tree, and rose trees. Pure properties in each data structure include size properties (number of allocated objects), sortedness, the maximal element of heaps, the height of trees, and color (red or black). The next column captures the number of satisfiability problems sent to the solver for the verification of each program. The next two columns describe the number of `unsat` and `sat` queries, respectively. The remaining four columns are divided into two groups corresponding to the runs without and with composition. For each group, we report the number of Z3 invocations in the first column and the time taken in seconds in the second column. In the last row, we sum the values of all the measurements of all data structures.

Table 2: Experimental Results on SL-COMP benchmarks

Tool	*qf_shls_sat*			*qf_shid_sat*			*qf_shidlia_sat*		
	`sat` (55)	`unsat` (55)	Time (110)	`sat` (81)	`unsat` (18)	Time (99)	`sat` (15)	`unsat` (18)	Time (33)
`Asterix` [28]	55	55	0.56s	-	-	-	-	-	-
`SPEN` [13]	55	55	16.60s	-	-	-	-	-	-
`S2SAT`$_{\texttt{SL}}$ [23]	55	55	36.61s	50	12	382m	13	8	120m6s
Harrsh [17]	55	55	11m7.41s	55	18	274m56s	-	-	-
`SLSAT` [5]	54	54	36m22s	57	18	218m51s	-	-	-
`S2S`	55	55	1.18s	56	18	237m55s	15	18	10.07s

The results show that `S2S` with the compositional reasoning is much more efficient. In all experiments, the compositional solving helps to discharge the queries quickly with small numbers of Z3 invocations. To sum up, `S2S` with the compositional reasoning took as 0.024% (17.74s/74,189.78s) in time and 0.184% (6,461/3,510,585) in the numbers of Z3 invocations as without the compositional reasoning. The experimental results of red-black trees, AVL with height and size properties, especially, confirm the great advantage of the compositional reasoning. On average, `S2S` with the compositional reasoning took 0.0028 seconds to discharge one satisfiability problem.

7.2 Experiments on SL-COMP 2019 benchmarks

We have compared `S2S` against the state-of-the-art solvers like `Asterix` [28], `SLSAT` [5], `SPEN` [13], `S2SAT`$_{\texttt{SL}}$ [23], and Harrsh [17]. We have conducted the comparisons over *all* three satisfiability divisions of the competition: *qf_shls_sat*, *qf_shid_sat*, and *qf_shidlia_sat*. All test problems are in our decidable fragments. For each division, we report the number of correct outputs and time (in minutes and seconds) taken by each tool. We note that as `Asterix` supported hardwired singly-linked lists only, it is unable to handle the problems in *qf_shid_sat*, and *qf_shidlia_sat*. Similarly, as `SPEN`, `SLSAT` and Harrsh have not supported arithmetic, *qf_shidlia_sat* is beyond their interests.

We report the results in Table 2. In this table, the first column presents the name of the tools. The remaining columns show the results of three divisions each of which includes three columns: the number of correct satisfiability results (`sat`), the number of correct unsatisfiability results (`unsat`), and the time (*m* is for minutes and *s* for seconds) taken, respectively, by each tool. - means the solver has not supported these benchmarks in the corresponding division yet. In the third row, the number between "(..)" reports the total number of tests in a column.

The first group shows the results of division *qf_shls_sat*. Each test problem, generated randomly by `Asterix`'s authors [27], contains 10-20 pointer-typed variables pointing to singly-linked lists. In this division, all tools performed pretty well. In particular, `Asterix` performed the best, `S2S` was the second, and `SPEN` was the third. `Asterix` and `S2S` decided all tests correctly in a short time. `SLSAT` was timeout in 3 test problems. We note that as the definition of the singly-linked list was hardwired syntactically in

`Asterix`, in contrast to `S2S`, `Asterix` would save the parsing time for this definition in these 110 tests.

The second group reports the results for division *qf_shid_sat* targeting general inductive definitions. It includes 40 challenging tests (succ-circuit[01..20] and succ-rec[01..20]), generated by `SLSAT`'s authors [5], each of which requires a brute-force search of 2^n values. `SLSAT`, `S2S`, and Harrsh performed pretty well in this division. `SLSAT` performed the best; it was either timeout or stack overflow at only 24 problems: succ-circuit[07..20] and succ-rec[11..20]. Note that, in SL-COMP 2019 [37], `S2S` was implemented together with an under-approximate technique and outperformed other tools; it discharged all problems in this division correctly in a super short time.

The third group, whose tests were contributed by the authors of [14,41,23], describes the results of division *qf_shidlia_sat* targeting the combination of linearly compositional inductive predicates and pure properties. While $\texttt{S2SAT}_{\texttt{SL}}$ could handle 21/33 tests in 120 minutes, `S2S` reported correctly for all tests within 10.07 seconds.

$\texttt{S2SAT}_{\texttt{SL}}$ also base on cyclic proofs. It did not support compositionality. Instead of reusing the bases, it constructed cyclic proofs for every input. Its termination thus relies on not only those definitions of inductive predicates but also the arithmetical constraints in the input. For instance, $\texttt{S2SAT}_{\texttt{SL}}$ could not handle $\Delta_{01}{\equiv}\texttt{els}(x,n){\wedge}n{=}320001$ and $\Delta_{02}{\equiv}\texttt{els}(x,n){\wedge}n{=}320000$ in which `els` predicate represents lists with an even number of elements and n captures the length. If we increased the timeout to a large enough number, $\texttt{S2SAT}_{\texttt{SL}}$ would manage the second test successfully but would not terminate for the first one. In contrast, `S2S` first inferred for $\texttt{els}(x,n)$ the base as $\texttt{base}^{\mathcal{P}}(\texttt{els}(x,n)) \equiv \{\texttt{emp}{\wedge}x{=}\texttt{null}{\wedge}n{=}0, \exists k.\ x{\mapsto}\{\_\}{\wedge}n{=}2k{\wedge}k{>}0\}$. After that, to decide Δ_{01}, it replaced this base into Δ_{01} and found that both disjuncts are unsatisfiable. It thus decided Δ_{01} as unsatisfiable. For Δ_{02}, after replacing the base into Δ_{02}, it found that the second disjunct was satisfiable. Hence, it concluded Δ_{02} is satisfiable.

8 Related Work

The “base” was first introduced for shape-only predicates by Brotherston *et al.* [5] and then extended for the combination of shape and arithmetic properties by Le *et al.* [23]. While the former computes the base based on a fixed point algorithm, the latter makes use of cyclic proofs. However, these works did not discuss compositionality. Unlike [5] and [23], this work presents a compositional reasoning as well as the most expressive syntactic decidable fragments. Our proposal is a generation of the work presented in [23]: It returns all equisatisfiable solutions of a formula. The proposed decidable fragment, `SHIDe`, is slightly extended from the one on general inductive definitions introduced in [5] and [23]. A crucial contribution of our work is to show that we can apply compositional reasoning into this fragment for efficiency.

The authors in [21] propose a cyclic proof system for the combination of heap structures and universally pure constraints. In another direction, work in [39] presents a decision procedure for a fragment where every predicate has two spatial base pairs, and their pure projections are Presburger-definable. Recently, authors in [23] extended the cyclic proof system in [21] for decidable fragments that subsume the ones presented in both [21] and [39]. They also identify *semantic* conditions for decidable fragments with arithmetic constraints. Our focus complements [23] as we target compositional satisfiability solving. The authors in [6,7] studied satisfiability for array separation logic. However, these works did not consider inductive predicates like ours. Nevertheless, developing a cyclic proof system to reason about the contents of array predicates might be future work; for example, by reducing array theory into string constraints [20].

The idea of small models of heap structures and data has been discussed in the literature: in separation logic [3,15,18] and other logics [25,26]. However, unlike ours, the works in separation logic mainly focused on the entailment problem. Berdine *et al.* present pioneering results for lists and binary trees [3]. They show that a singly-linked list has the small model property; every singly-linked list predicate can be precisely characterised by those heap models of size zero or two. Recently, this fragment was extended with some small-model pure properties in [18]. Our proposal infers small models for compositionality in a fragment, far beyond the lists and trees, including array separation logic with general inductive predicates and small-model pure properties.

Other related works include those satisfiability solvers presented in [33,32,17,14,41]. The authors in [17] present a decision procedure based on heap automata. [14,41] present a graph-based technique with predicates that are beyond the singly-linked lists. These works support compositional predicates and one-hole trees with sortedness, size, and balancedness. However, they have supported neither mutually recursive definitions nor nested lists like ours. Our work closely relates to the satisfiability solver for STRAND logic [25]. In [25], the authors discussed satisfiability-preserving embedding that helps to enumerate a finite number of minimal models. Similar to this work, given a formula, our procedure derives for it a base with the minimal model property that precisely characterises its satisfiability. Unlike this work, while our solver works compositionally, STRAND did not support the compositionality.

Those works in [12,35] complement our work. While these works supported the separating implication, they did not consider inductive definitions like ours. Finally, the work in [39] discusses a solver for arithmetic with inductive definitions. This work proposes to infer for each numerically inductive predicate a closed-form, an equivalent formula in Presburger arithmetic. We here extend the decidable fragment in [39] with nested list predicates.

9 Conclusion

We have presented a novel satisfiability solver in a fragment of array separation logic combined with inductive definitions and arithmetic properties. Our proposal differentiates itself from existing works on the compositional reasoning via the base inference. Furthermore, we have shown that satisfiability solving can be compositional in the current fragments. We have implemented the proposal into `S2S` and evaluated it over the two sets of non-trivial benchmarks: taken from the SL-COMP 2019 and generated from the verification of complex-pointer programs. The experimental results show that `S2S` is effective and efficient. and is promising for being used in a verification system.

Acknowledgments The author was partially supported by the UK's Engineering and Physical Sciences Research Council (EPSRC): Grant number EP/R006865/1.

References

1. Clark Barrett, Daniel Kroening, and Thomas Melham. *Problem solving for the 21st century: Efficient solver for satisfiability modulo theories*. Knowledge Transfer Report, Technical Report 3. London Mathematical Society and Smith Institute for Industrial Mathematics and System Engineering, 6 2014.
2. A. P. Bel'tyukov. Decidability of the universal theory of natural numbers with addition and divisibility. *Journal of Soviet Mathematics*, 14(5):1436–1444, Nov 1980.
3. Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. A decidable fragment of separation logic. In *Proceedings of the 24th International Conference on Foundations of Software Technology and Theoretical Computer Science*, FSTTCS'04, page 97–109, Berlin, Heidelberg, 2004. Springer-Verlag.
4. Marius Bozga and Radu Iosif. On decidability within the arithmetic of addition and divisibility. In Vladimiro Sassone, editor, *Foundations of Software Science and Computational Structures*, pages 425–439, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
5. James Brotherston, Carsten Fuhs, Nikos Gorogiannis, and Juan Navarro Pérez. A decision procedure for satisfiability in separation logic with inductive predicates. In *Proceedings of CSL-LICS*. ACM, 2014.
6. James Brotherston, Nikos Gorogiannis, and Max Kanovich. *Biabduction (and Related Problems) in Array Separation Logic*, pages 472–490. Springer International Publishing, 2017.
7. James Brotherston and Max Kanovich. On the complexity of pointer arithmetic in separation logic. In *Proceedings of APLAS-16*, LNCS. Springer, 2018.
8. Cristiano Calcagno, Dino Distefano, Jeremy Dubreil, Dominik Gabi, Pieter Hooimeijer, Martino Luca, Peter O'Hearn, Irene Papakonstantinou, Jim Purbrick, and Dulma Rodriguez. Moving fast with software verification. In Klaus Havelund, Gerard Holzmann, and Rajeev Joshi, editors, *NASA Formal Methods*, pages 3–11. Springer International Publishing, 2015.
9. Cristiano Calcagno, Dino Distefano, Peter W. O'Hearn, and Hongseok Yang. Compositional shape analysis by means of bi-abduction. *J. ACM*, 58(6):26, 2011.
10. Wei-Ngan Chin, Cristian Gherghina, Răzvan Voicu, Quang Loc Le, Florin Craciun, and Shengchao Qin. A specialization calculus for pruning disjunctive predicates to support verification. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification*, pages 293–309, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
11. Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
12. Mnacho Echenim, Radu Iosif, and Nicolas Peltier. The bernays-schönfinkel-ramsey class of separation logic on arbitrary domains. In Mikołaj Bojańczyk and Alex Simpson, editors, *Foundations of Software Science and Computation Structures*, pages 242–259, Cham, 2019. Springer International Publishing.
13. Constantin Enea, Ondrej Lengál, Mihaela Sighireanu, and Tomás Vojnar. Compositional entailment checking for a fragment of separation logic. *Formal Methods in System Design*, 51(3):575–607, 2017.
14. Xincai Gu, Taolue Chen, and Zhilin Wu. A complete decision procedure for linearly compositional separation logic with data constraints. In *Proceedings of the 8th International Joint Conference on Automated Reasoning - Volume 9706*, page 532–549, Berlin, Heidelberg, 2016. Springer-Verlag.
15. Radu Iosif, Adam Rogalewicz, and Tomáš Vojnar. Deciding entailments in inductive separation logic with tree automata. In Franck Cassez and Jean-François Raskin, editors, *Automated Technology for Verification and Analysis*, pages 201–218, Cham, 2014. Springer International Publishing.

16. Samin S. Ishtiaq and Peter W. O'Hearn. Bi as an assertion language for mutable data structures. *SIGPLAN Not.*, 36(3):14–26, January 2001.
17. Christina Jansen, Jens Katelaan, Christoph Matheja, Thomas Noll, and Florian Zuleger. Unified reasoning about robustness properties of symbolic-heap separation logic. In Hongseok Yang, editor, *Programming Languages and Systems: 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22–29, 2017, Proceedings*, pages 611–638, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.
18. Jens Katelaan, Dejan Jovanović, and Georg Weissenbacher. A separation logic with data: Small models and automation. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Automated Reasoning*, pages 455–471, Cham, 2018. Springer International Publishing.
19. Quang Loc Le, Cristian Gherghina, Shengchao Qin, and Wei-Ngan Chin. Shape analysis via second-order bi-abduction. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification*, pages 52–68, Cham, 2014. Springer International Publishing.
20. Quang Loc Le and Mengda He. A decision procedure for string logic with quadratic equations, regular expressions and length constraints. In Sukyoung Ryu, editor, *Programming Languages and Systems*, pages 350–372, Cham, 2018. Springer International Publishing.
21. Quang Loc Le, Jun Sun, and Wei-Ngan Chin. Satisfiability modulo heap-based programs. In Swarat Chaudhuri and Azadeh Farzan, editors, *Computer Aided Verification*, pages 382–404, Cham, 2016. Springer International Publishing.
22. Quang Loc Le, Jun Sun, and Shengchao Qin. Frame inference for inductive entailment proofs in separation logic. In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 41–60, Cham, 2018. Springer International Publishing.
23. Quang Loc Le, Makoto Tatsuta, Jun Sun, and Wei-Ngan Chin. A decidable fragment in separation logic with inductive predicates and arithmetic. In Rupak Majumdar and Viktor Kunčak, editors, *Computer Aided Verification*, pages 495–517, Cham, 2017. Springer International Publishing.
24. L. Lipshitz. The diophantine problem for addition and divisibility. *Transactions of the American Mathematical Society*, 235:271–283, 1978.
25. P. Madhusudan, Gennaro Parlato, and Xiaokang Qiu. Decidable logics combining heap structures and data. In *Proceedings of the 38th Annual Symposium on Principles of Programming Languages*, POPL '11, pages 611–622, New York, NY, USA, 2011. ACM.
26. Scott McPeak and George C. Necula. Data structure specifications via local equality axioms. In Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification*, pages 476–490, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
27. Juan Antonio Navarro Pérez and Andrey Rybalchenko. Separation logic + superposition calculus = heap theorem prover. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, page 556–566, New York, NY, USA, 2011. Association for Computing Machinery.
28. Juan Antonio Navarro Pérez and Andrey Rybalchenko. Separation logic modulo theories. In Chung-chieh Shan, editor, *Programming Languages and Systems*, pages 90–106, Cham, 2013. Springer International Publishing.
29. Long H. Pham, Quang Loc Le, Quoc-Sang Phan, and Jun Sun. Concolic testing heap-manipulating programs. In Maurice H. ter Beek, Annabelle McIver, and José N. Oliveira, editors, *Formal Methods – The Next 30 Years*, pages 442–461, Cham, 2019. Springer International Publishing.
30. Long H. Pham, Quang Loc Le, Quoc-Sang Phan, Jun Sun, and Shengchao Qin. Testing heap-based programs with java starfinder. In *Proceedings of the 40th International Conference on*

Software Engineering: Companion Proceeedings, ICSE '18, pages 268–269, New York, NY, USA, 2018. ACM.

31. Long H. Pham, Quang Loc Le, Quoc-Sang Phan, Jun Sun, and Shengchao Qin. Enhancing Symbolic Execution of Heap-based Programs with Separation Logic for Test Input Generation. *In Proceeding of ATVA*, 2019.
32. Ruzica Piskac, Thomas Wies, and Damien Zufferey. Automating separation logic with trees and data. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification*, pages 711–728, Cham, 2014. Springer International Publishing.
33. Ruzica Piskac, Thomas Wies, and Damien Zufferey. Grasshopper - complete heap verification with mixed specifications. In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 124–139, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
34. C. Popeea and W.-N. Chin. Inferring disjunctive postconditions. In *ASIAN*, pages 331–345, 2006.
35. Andrew Reynolds, Radu Iosif, and Cristina Serban. Reasoning in the bernays-schönfinkel-ramsey fragment of separation logic. In Ahmed Bouajjani and David Monniaux, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 462–482, Cham, 2017. Springer International Publishing.
36. J. C. Reynolds. Separation logic: a logic for shared mutable data structures. In *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science*, pages 55–74, 2002.
37. Mihaela Sighireanu, Nikos Gorogiannis, and Radu Iosif. SL-COMP 2019. https://www.irif.fr/ sighirea/sl-comp/19/index.html, 2019. [accessed 15-Nov-2020].
38. Mihaela Sighireanu, Juan Antonio Navarro Pérez, Andrey Rybalchenko, Nikos Gorogiannis, Radu Iosif, Andrew Reynolds, Cristina Serban, Jens Katelaan, Christoph Matheja, Thomas Noll, Florian Zuleger, Wei-Ngan Chin, Quang Loc Le, Quang-Trung Ta, Ton-Chanh Le, Thanh-Toan Nguyen, Siau-Cheng Khoo, Michal Cyprian, Adam Rogalewicz, Tomás Vojnar, Constantin Enea, Ondrej Lengál, Chong Gao, and Zhilin Wu. SL-COMP: competition of solvers for separation logic. In *Tools and Algorithms for the Construction and Analysis of Systems - 25 Years of TACAS: TOOLympics, Held as Part of ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part III*, pages 116–132, 2019.
39. Makoto Tatsuta, Quang Loc Le, and Wei-Ngan Chin. Decision procedure for separation logic with inductive definitions and presburger arithmetic. In Atsushi Igarashi, editor, *Programming Languages and Systems*, pages 423–443, Cham, 2016. Springer International Publishing.
40. Minh-Thai Trinh, Quang Loc Le, Cristina David, and Wei-Ngan Chin. Bi-abduction with pure properties for specification inference. In Chung-chieh Shan, editor, *Programming Languages and Systems*, pages 107–123, Cham, 2013. Springer International Publishing.
41. Zhaowei Xu, Taolue Chen, and Zhilin Wu. Satisfiability of compositional separation logic with tree predicates and data constraints. In Leonardo de Moura, editor, *Automated Deduction – CADE 26*, pages 509–527, Cham, 2017. Springer International Publishing.