Automating Program Proofs Based on Separation Logic with Inductive Definitions

Constantin Enea[1], Mihaela Sighireanu[1] and Zhilin Wu[2,1]

[1] LIAFA, Université Paris Diderot, France
[2] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China.

Abstract. This paper investigates the use of Separation Logic with inductive definitions in reasoning about programs that manipulate dynamic data structures. We propose a novel approach for exploiting the inductive definitions in automating program proofs based on inductive invariants. We focus on iterative programs, although our techniques apply to recursive programs as well, and specifications that describe not only the shape of the data structures, but also their content or their size. This approach is based on a careful inspection of the typical lemmas needed in such program proofs and efficiently checkable criteria for recognizing inductive definitions that satisfy these lemmas. Empirically, we find that our approach is powerful enough to deal with sophisticated benchmarks, e.g., iterative procedures for searching, inserting, or deleting elements in binary search tress, red-black trees, and AVL trees, in a very efficient way.

1 Introduction

Program verification requires reasoning about complex, unbounded size data structures that may carry data ranging over infinite domains. Examples of such structures are multi-linked lists, nested lists, trees, etc. Programs manipulating such structures perform operations that may modify their shape (due to dynamic creation and destructive updates) as well as the data attached to their elements. An important issue is the design of logic-based frameworks allowing to express assertions about program configurations (at given control points), and then to check automatically the validity of these assertions, for all computations. This leads to the challenging problem of finding relevant compromises between expressiveness, automation, and scalability.

An established approach for scalability is the use of *Separation logic* (SL) [19, 25]. Indeed, its support for local reasoning based on the Frame Rule leads to compact proofs, that can be dealt with in an efficient way. However, finding expressive fragments of SL for writing program assertions, that enable efficient automated validation of the verification conditions, remains a major issue. Typically, SL is used in combination with *inductive definitions*, which provide a natural description of the data structures manipulated by a program. Moreover, since program proofs themselves are based on induction, using inductive definitions instead of universal quantifiers (like in approaches based on first-order logic) enables scalable automation, especially for recursive programs which traverse the data structure according to their inductive definition, e.g., [23]. Nevertheless, automating the validation of the verification conditions generated for **iterative programs**, that traverse the data structures using while loops, remains a challenge. The loop invariants use inductive definitions for *fragments of data structures*, traversed during a partial execution of the loop, and proving the inductiveness of these invariants requires non-trivial *lemmas* relating (compositions of) such inductive definitions. Most of the existing works require that these lemmas are provided by the user of the verification system, e.g., [8, 18, 23] or they use translations of SL to first-order logic to avoid this problem. However, the latter approaches work only for rather limited fragments [21, 22]. In general, it is difficult to have lemmas relating complex user-defined inductive predicates that describe not only the shape of the data structures but also their content.

To illustrate this difficulty, consider the simple example of a sorted singly linked list. The following inductive definition describes a sorted list segment from the location E to F, storing a multiset of values M:

$$\begin{aligned}
&\mathit{lseg}(E, M, F) ::= E = F \land \mathtt{emp} \land M = \emptyset \\
&\mathit{lseg}(E, M, F) ::= (\exists X, v, M_1.\ E \mapsto \{(\mathtt{next}, X), (\mathtt{data}, v)\} * \mathit{lseg}(X, M_1, F) \\
&\qquad\qquad\qquad\qquad\qquad\land\ v \leq M_1 \land M = M_1 \cup \{v\}
\end{aligned}$$

where $\mathtt{emp}$ denotes the empty heap, $E \mapsto \{(\mathtt{next}, X), (\mathtt{data}, v)\}$ states that the pointer field `next` of E points to X while its field `data` stores the value v, and $*$ is the separating conjunction. Proving inductive invariants of typical sorting procedures requires such an inductive definition and the following lemma:

$$\exists E_2.\ \mathit{lseg}(E_1, M_1, E_2) * \mathit{lseg}(E_2, M_2, E_3) \land M_1 \leq M_2 \Rightarrow \exists M.\ \mathit{lseg}(E_1, M, E_3).$$

The data constraints in these lemmas, e.g., $M_1 \leq M_2$ (stating that every element of M_1 is less or equal than all the elements of M_2), which become more complex when reasoning for instance about binary search trees, are an important obstacle for trying to synthesize them automatically.

Our work is based on a new class of inductive definitions for describing fragments of data structures that (i) support lemmas **without additional** data constraints like $M_1 \leq M_2$ and (ii) allow to **automatically synthesize** these lemmas using efficiently checkable, almost syntactic, criteria. For instance, we use a different inductive definition for *lseg*, with an additional parameter M':

$$\mathit{lseg}(E, M, F, M') ::= E = F \land \mathtt{emp} \land M = M' \tag{1}$$

$$\begin{aligned}
\mathit{lseg}(E, M, F, M') ::= (&\exists X, v, M_1.\ E \mapsto \{(\mathtt{next}, X), (\mathtt{data}, v)\} * \mathit{lseg}(X, M_1, F, M') \\
&\qquad\qquad\land\ v \leq M_1 \land M = M_1 \cup \{v\}
\end{aligned} \tag{2}$$

The additional multiset parameter M' provides a "port" for appending another sorted list segment, just like F does when we are referring strictly to the shape of the list segment. The new definition satisfies the following lemma, which contains no additional data constraints:

$$\exists E_2, M_2.\ \mathit{lseg}(E_1, M_1, E_2, M_2) * \mathit{lseg}(E_2, M_2, E_3, M_3) \Rightarrow \mathit{lseg}(E_1, M_1, E_3, M_3). \tag{3}$$

Besides such "composition" lemmas (formally defined in Sec. 5), we define (in Sec. 6) other classes of lemmas needed in program proofs and we provide efficient criteria for generating them automatically. Moreover, we propose (in Sec. 4) a proof strategy using such lemmas, based on simple syntactic matchings of spatial atoms (points-to atoms or predicate atoms like *lseg*) and reductions to SMT solvers for dealing with the data constraints. We show experimentally (in Sec. 7) that this proof strategy is powerful enough to deal with sophisticated benchmarks, e.g., the verification conditions generated from the iterative procedures for searching, inserting, or deleting elements in binary search tress, red-black trees, and AVL trees, in a very efficient way.

2 Motivating Example

Fig. 1 lists an iterative implementation of a search procedure for binary search trees (BSTs). The property that E points to a BST storing a multiset of values M can be expressed by the following inductively-defined predicate:

$$bst(E,M) ::= E = \mathsf{nil} \wedge \mathtt{emp} \wedge M = \emptyset \tag{4}$$

$$\begin{aligned} bst(E,M) ::= \exists X,Y,M_1,M_2,v.\, E \mapsto \{(\mathtt{left},X),(\mathtt{right},Y),(\mathtt{data},v)\} \\ *\ bst(X,M_1) * bst(Y,M_2) \\ \wedge\ E \neq \mathsf{nil} \wedge M = \{v\} \cup M_1 \cup M_2 \wedge M_1 < v < M_2 \end{aligned} \tag{5}$$

```
int search(struct Tree*
    root, int key) {
  struct Tree *t = root;
  while (t != NULL) {
    if (t->data == key)
      return 1;
    else if (t->data > key)
      t = t->left;
    else
      t = t->right;
  }
  return 0;
}
```

Fig. 1.

The predicate $bst(E,M)$ is defined by two rules describing respectively empty (eq. (4)) and non-empty trees (eq. (5)). The body (right-hand side) of each rule is a conjunction of a pure formula, formed of (dis)equalities between location variables (e.g. $E = \mathsf{nil}$) and data constraints (e.g. $M = \emptyset$), and a spatial formula describing the structure of the heap. The data constraints of the rule (5) define M to be the multiset of values stored in the tree, and state the sortedness property of BSTs.

The precondition of `search` is $bst(\mathtt{root}, M_0)$, where M_0 is a ghost variable denoting the multiset of values stored in the tree, while its postcondition is $bst(\mathtt{root}, M_0) \wedge (\mathtt{key} \in M_0 \rightarrow ret = 1) \wedge (\mathtt{key} \notin M_0 \rightarrow ret = 0)$, where *ret* denotes the return value.

The while loop traverses the BST in a top-down manner using the pointer variable `t`. This variable decomposes the heap into two domain-disjoint subheaps: the tree rooted at `t`, and the truncated tree rooted at `root` which contains a "hole" `t`. To specify the invariant of this loop, we define another predicate $bsthole(E, M_1, F, M_2)$ describing "truncated" BSTs with one hole F as follows:

$$bsthole(E,M_1,F,M_2) ::= E = F \wedge \mathtt{emp} \wedge M_1 = M_2, \tag{6}$$

$$\begin{aligned} bsthole(E,M_1,F,M_2) ::= \exists X,Y,M_3,M_4,v.\, E \mapsto \{(\mathtt{left},X),(\mathtt{right},Y),(\mathtt{data},v)\} \\ *\ bst(X,M_3) * bsthole(Y,F,M_4,M_2) \\ \wedge\ M_1 = \{v\} \cup M_3 \cup M_4 \wedge M_3 < v < M_4 \end{aligned} \tag{7}$$

$$\begin{aligned} bsthole(E,M_1,F,M_2) ::= \exists X,Y,M_3,M_4,v.\, E \mapsto \{(\mathtt{left},X),(\mathtt{right},Y),(\mathtt{data},v)\} \\ *\ bsthole(X,F,M_3,M_2) * bst(Y,M_4) \\ \wedge\ M_1 = \{v\} \cup M_3 \cup M_4 \wedge M_3 < v < M_4 \end{aligned} \tag{8}$$

Intuitively, the parameter M_2, interpreted as a multiset of values, is used to specify that the structure described by $bsthole(E, M_1, F, M_2)$ could be extended with a BST rooted at F and storing the values in M_2, to obtain a BST rooted at E and storing the values in M_1. Thus, the parameter M_1 of *bsthole* is the union of M_2 with the multiset of values stored in the truncated BST represented by $bsthole(E, M_1, F, M_2)$.

Using *bsthole*, we obtain a succinct specification of the loop invariant:

$$Inv ::= \exists M_1.\ bsthole(\texttt{root}, M_0, \texttt{t}, M_1) * bst(\texttt{t}, M_1) \wedge (\texttt{key} \in M_0 \Leftrightarrow \texttt{key} \in M_1). \quad (9)$$

We illustrate that such inductive definitions are appropriate for automated reasoning, by taking the following branch of the loop: `assume(t != NULL); assume(t->data > key); t' = t->left` (as usual, `if` statements are transformed into `assume` statements and primed variables are introduced in assignments). The postcondition of *Inv* w.r.t. this branch, denoted $post(Inv)$, is computed as usual by unfolding the *bst* predicate:

$$\begin{aligned}\exists M_1, Y, v, M_2, M_3.\ & bsthole(\texttt{root}, M_0, \texttt{t}, M_1) * \texttt{t} \mapsto \{(\texttt{left}, \texttt{t}'), (\texttt{right}, Y), (\texttt{data}, v)\} \\ & * \ bst(\texttt{t}', M_2) * bst(Y, M_3) \wedge M_1 = \{v\} \cup M_2 \cup M_3 \wedge M_2 < v < M_3 \\ & \wedge \ (\texttt{key} \in M_0 \Leftrightarrow \texttt{key} \in M_1) \wedge v > \texttt{key}. \qquad (10)\end{aligned}$$

The inductiveness of *Inv* w.r.t. this branch is expressed by the entailment $post(Inv) \Rightarrow Inv'$, where Inv' is obtained from *Inv* by replacing `t` with `t'`.

One of the contributions of this paper is a proof strategy for proving the validity of entailments of the form $\varphi_1 \Rightarrow \exists \vec{X}.\varphi_2$, where $\vec{X}$ contains only data variables[3]. The strategy is based on two steps: (i) enumerating spatial atoms A from φ_2, and for each of them, carving out a sub-formula φ_A of φ_1 that entails A, and (ii) proving that the data constraints from φ_A imply those from φ_2 (this can be done using SMT solvers). The step (i) may generate constraints on the existentially-quantified variables $\vec{X}$ in φ_2 that are used in the second step. If the step (ii) succeeds, and every spatial atom of φ_1 occurs in exactly one sub-formula obtained during the first step (this constraint is required by the semantics of the separating conjunction), then the entailment holds.

For the entailment $post(Inv) \Rightarrow Inv'$, the first step has two goals which consist in computing two sub-formulas of $post(Inv)$ that entail $\exists M_1'.\ bsthole(\texttt{root}, M_0, \texttt{t}', M_1')$ and respectively, $\exists M_1''.\ bst(\texttt{t}', M_1'')$. This renaming of existential variables requires adding the equality $M_1 = M_1' = M_1''$ to Inv'. The second goal, for $\exists M_1''.\ bst(\texttt{t}', M_1'')$, can be solved easily since this predicate almost matches the sub-formula $bst(\texttt{t}', M_2)$. This matching generates the constraint $M_1'' = M_2$, which provides an instantiation of the existential variable M_1'' useful in proving the entailment between the data constraints.

Computing a sub-formula that entails $\exists M_1'.\ bsthole(\texttt{root}, M_0, \texttt{t}', M_1')$ requires a non-trivial lemma. Thus, according to the syntactic criteria defined in Sec. 5, the predicate *bsthole* enjoys the following *composition lemma*:

$$\begin{aligned}\big(\exists F, M.\ bsthole(\texttt{root}, M_0, F, M) * bsthole(F, M, \texttt{t}', M_1')\big) \qquad (11) \\ \Rightarrow bsthole(\texttt{root}, M_0, \texttt{t}', M_1').\end{aligned}$$

[3] The existential quantifiers in φ_1 can be removed using skolemization.

Intuitively, this lemma states that composing two heap structures described by *bsthole* results in a structure that satisfies the same predicate. The particular relation between the arguments of the predicate atoms in the left-hand side is motivated by the fact that the parameters F and M are supposed to represent "ports" for composing $\mathit{bsthole}(\mathtt{root}, M_0, F, M)$ with some other similar heap structures. This property of F and M is characterized syntactically by the fact that, roughly, F (resp. M) occurs only once in the body of each inductive rule of *bsthole*, and F (resp. M) occurs only in an equality with $\mathtt{root}$ (resp. M_0) in the base rule (we are referring to the rules (6)–(8) with the parameters of *bsthole* substituted by $(\mathtt{root}, M_0, F, M)$).

Therefore, the first goal reduces to finding a sub-formula of $\mathit{post}(\mathit{Inv})$ that implies the premise of (11) where M_1' remains existentially-quantified. Recursively, we apply the same strategy of enumerating spatial atoms and finding sub-formulas that entail them. However, we are relying on the fact that all the existential variables denoting the root locations of spatial atoms, e.g., F in lemma (11), occur as argument in the only spatial atom rooted at the same location as the conclusion, i.e., $\mathtt{root}$ in lemma (11). Therefore, in the first sub-goal $\exists F, M.\ \mathit{bsthole}(\mathtt{root}, M_0, F, M)$, matches the atom $\mathit{bsthole}(\mathtt{root}, M_0, \mathtt{t}, M_1)$ under the constraint $F = \mathtt{t} \wedge M = M_1$. This constraint is used in solving the second sub-goal, which now becomes $\exists M_1'.\ \mathit{bsthole}(\mathtt{t}, M_1, \mathtt{t}', M_1')$.

The second sub-goal can be proved by unfolding *bsthole* twice, using first the rule (8) and then the rule (6), and by matching the resulting spatial atoms with those in $\mathit{post}(\mathit{Inv})$ one by one. This completes the proof of $\mathit{post}(\mathit{Inv}) \Rightarrow \mathit{Inv}'$ since the sub-formulas generated for the two initial goals are disjoint and they cover all the spatial atoms of $\mathit{post}(\mathit{Inv})$. Also, assuming that the existential M_1 from Inv' is instantiated with M_2 from $\mathit{post}(\mathit{Inv})$ (fact automatically deduced in the first step), the data constraints in $\mathit{post}(\mathit{Inv})$ entail those in Inv'.

3 Separation Logic with Inductive Definitions

Let LVar be a set of *location variables*, interpreted as heap locations, and DVar a set of *data variables*, interpreted as data values stored in the heap, (multi)sets of values, etc. In addition, let $\mathsf{Var} = \mathsf{LVar} \cup \mathsf{DVar}$. The domain of heap locations is denoted by $\mathbb{L}$ while the domain of the variables in DVar is generically denoted by $\mathbb{D}$. Let $\mathcal{F}$ be a set of pointer fields, interpreted as functions $\mathbb{L} \rightharpoonup \mathbb{L}$, and $\mathcal{D}$ a set of data fields, interpreted as functions $\mathbb{L} \rightharpoonup \mathbb{D}$. The syntax of the Separation Logic fragment considered in this paper is defined in Table 1.

Formulas are interpreted over pairs (s, h) formed of a *stack* s and a *heap* h. The stack s is a function giving values to a finite set of variables (location or data variables) while the heap h is a function mapping a finite set of pairs (ℓ, pf), where ℓ is a location and pf is a pointer field, to locations, and a finite set of pairs (ℓ, df), where df is a data field, to values in $\mathbb{D}$. In addition, h satisfies that for each $\ell \in \mathbb{L}$, if $(\ell, \mathit{df}) \in \mathsf{dom}(h)$ for some $d \in \mathcal{D}$, then $(\ell, \mathit{pf}) \in \mathsf{dom}(h)$ for some $\mathit{pf} \in \mathcal{F}$. Let $\mathsf{dom}(h)$ denote the domain of h, and $\mathsf{ldom}(h)$ denote the set of $\ell \in \mathbb{L}$ such that $(\ell, \mathit{pf}) \in \mathsf{dom}(h)$ for some $\mathit{pf} \in \mathcal{F}$.

Table 1. The syntax of the Separation Logic fragment

$X, Y, E \in \mathsf{LVar}$	location variables	$\rho \subseteq (\mathcal{F} \times \mathsf{LVar}) \cup (\mathcal{D} \times \mathsf{DVar})$	
$\vec{F} \in \mathsf{Var}^*$	vector of variables	$P \in \mathcal{P}$	predicates
$x \in \mathsf{Var}$	variable	Δ	formula over data variables

$$\Pi ::= X = Y \mid X \neq Y \mid \Delta \mid \Pi \wedge \Pi \qquad \text{pure formulas}$$

$$\Sigma ::= \mathtt{emp} \mid E \mapsto \rho \mid P(E, \vec{F}) \mid \Sigma * \Sigma \qquad \text{spatial formulas}$$

$$\varphi ::= \Pi \wedge \Sigma \mid \varphi \vee \varphi \mid \exists x.\ \varphi \qquad \text{formulas}$$

Formulas are conjunctions between a pure formula Π and a spatial formula Σ. Pure formulas characterize the stack s using (dis)equalities between location variables, e.g., a stack models $x = y$ iff $s(x) = s(y)$, and constraints Δ over data variables. We let Δ unspecified, though we assume that they belong to decidable theories, e.g., linear arithmetic or quantifier-free first order theories over multisets of values. The atom `emp` of spatial formulas holds iff the domain of the heap is empty. The *points-to atom* $E \mapsto \{(f_i, x_i)\}_{i \in \mathcal{I}}$ specifies that the heap contains exactly one location E, and for all $i \in \mathcal{I}$, the field f_i of E equals x_i, i.e., $h(s(E), f_i) = s(x_i)$. The fragment is parameterized by a set $\mathcal{P}$ of *inductively defined predicates*, described hereafter.

Let $P \in \mathcal{P}$. An *inductive definition* of P is a finite set of rules of the form $P(E, \vec{F}) ::= \exists \vec{Z}. \Pi \wedge \Sigma$, where $\vec{Z} \in \mathsf{Var}^*$ is a tuple of variables. A rule R is called a *base rule* if Σ contains no predicate atoms. Otherwise, it is called an *inductive rule*. A base rule R is called a *spatial-empty* base rule if $\Sigma = \mathtt{emp}$. Otherwise, it is called a *spatial-nonempty* base rule. For instance, the predicate *bst* in Section 2 is defined by one spatial-empty base rule and one inductive rule.

For each predicate $P(E, \vec{F}) \in \mathcal{P}$, we distinguish the first parameter E from the other parameters. Intuitively, E represents the root of the heap structure described by $P(E, \vec{F})$. We consider several restrictions on the rules defining a predicate $P(E, \vec{F}) \in \mathcal{P}$. Thus, for each rule $P(E, \vec{F}) ::= \exists \vec{Z}. \Pi \wedge \Sigma$,

- If the rule is inductive, then
  - *One points-to atom*: Σ contains exactly one points-to atom $E \mapsto \rho$, for some ρ. In addition, for each field $f \in \mathcal{F}$ (resp. $d \in \mathcal{D}$), ρ contains at most one occurrence of f (resp. d).
  - *Connectedness*: For each predicate atom $Q(E_1, \vec{F_1})$ in Σ, there is $Z \in \mathsf{LVar}$ such that $\Pi \models E_1 = Z$ and Z occurs in ρ.
- If the rule is a spatial-nonempty base rule, then Σ contains exactly one points-to atom $E \mapsto \rho$, for some ρ.

Since we disallow the use of negations on top of the spatial atoms, the semantics of the predicates in $\mathcal{P}$ is defined as usual as a least fixed-point.

We say that a formula ψ_1 *entails* another formula ψ_2, denoted by $\psi_1 \Rightarrow \psi_2$, iff every model of ψ_1 is also a model of ψ_2. In addition, $\psi_1 \Leftrightarrow \psi_2$ is used to denote the conjunction of $\psi_1 \Rightarrow \psi_2$ and $\psi_2 \Rightarrow \psi_1$.

4 A Proof Strategy Based on Lemmas

We introduce a proof strategy based on lemmas for entailments $\varphi_1 \Rightarrow \exists\vec{X}.\varphi_2$, where φ_1, φ_2 are quantifier-free, and $\vec{X} \in \mathsf{DVar}^*$. We consider quantifier-free left-hand sides φ_1 since the existential variables from this part of the entailment can be skolemized. In addition, we restrict our considerations to the situation that only data variables are quantified in the right-hand side[4]. W.l.o.g., we assume that every variable in $\vec{X}$ occurs in at most one spatial atom of φ_2 (multiple occurrences of the same variable can be removed by introducing fresh variables and new equalities in the pure part). Also, we assume that φ_1 and φ_2 are of the form $\Pi \wedge \Sigma$. In the general case, our proof strategy checks that for every disjunct φ_1' of φ_1, there is a disjunct φ_2' of φ_2 s.t. $\varphi_1' \Rightarrow \exists\vec{X}.\varphi_2'$.

Our proof strategy is defined by the recursive procedure $\mathsf{slice}(\varphi_1, \exists\vec{X}.\varphi_2)$ in Alg. 1. The procedure computes a sub-formula ψ of φ_1 and two pure formulas $\mathcal{C}_r$, called *rely*, and $\mathcal{C}_g$, called *guarantee*, such that $\psi \wedge \mathcal{C}_r \Rightarrow \varphi_2 \wedge \mathcal{C}_g$. Intuitively, the constraint $\mathcal{C}_r$ defines the instantiations of the existential variables $\vec{X}$ over variables (terms) of φ_1; the constraint $\mathcal{C}_g$ describes the possible relations between data variables of φ_1 assumed while computing the sub-formula ψ. When ψ is syntactically the same as φ_1, the entailment $\varphi_1 \Rightarrow \exists\vec{X}.\ \varphi_2$ holds.

When φ_2 contains at least two *spatial* atoms, slice is called recursively (line 3) on each spatial atom from φ_2, preserving the existential quantifiers over $\vec{X}$. Then, it checks (line 7) that the pure part of φ_1, together with the rely constraints obtained from the recursive calls, implies the pure part of φ_2 and the guarantees from the recursive calls. The second condition at line 7 ensures that the semantics of the separating conjunction is preserved by checking that every spatial atom of φ_1 occurs in at most one sub-formula returned by the recursive calls. If both tests succeed, slice returns the pure part of φ_1 conjuncted to all the spatial sub-formulas obtained from the recursive calls, and the conjunction of all the rely, and respectively, guarantee constraints (line 8).

If φ_2 contains only one spatial atom, slice checks whether φ_1 contains a spatial atom that matches the spatial atom in φ_2 (line 10), using the function $\mathsf{matchAtom}$. If φ_2 is an atom $\exists\vec{X}.\ E \mapsto \rho$, then $\mathsf{matchAtom}$ searches for an atom $E' \mapsto \rho'$ of φ_1 that (1) has the same root, i.e., $\mathsf{Pure}(\varphi_1) \models E = E'$, (2) for every pointer field f, if $(f, X) \in \rho$, then there is $(f, Y) \in \rho'$, and vice versa. $\mathsf{matchAtom}$ returns such an atom, if it exists, or an error value $\bot$ otherwise. Moreover, in the positive case, $\mathsf{matchAtom}$ computes a rely constraint $\mathcal{C}_r$ and a guarantee constraint $\mathcal{C}_g$ (which are sets of equality constraints between data terms) such that $\big(\mathsf{Pure}(\varphi_1) \wedge E' \mapsto \rho' \wedge \mathcal{C}_r\big) \Rightarrow E \mapsto \rho \wedge \mathcal{C}_g$. For example, consider the call of $\mathsf{matchAtom}$ with the following formulas:

$$\varphi_1 ::= X = Y \wedge w = w' \wedge E \mapsto \{(f, Y), (d_1, v), (d_2, w)\}$$
$$\varphi_2 ::= \exists v'.\ E \mapsto \{(f, X), (d_1, v'), (d_2, w')\},$$

where d_1 and d_2 are data fields. $\mathsf{matchAtom}$ returns the points-to atom from φ_1, $\mathcal{C}_r : v = v'$, and $\mathcal{C}_g : w = w'$. The basic principle here is to add an equality

[4] We believe that this restriction is reasonable for the verification conditions appearing in practice and all the benchmarks in our experiments are of this form.

1 **Procedure** $\mathsf{slice}\,(\,\varphi_1,\ \exists\vec{X}.\ \varphi_2\,)$
2   **if** $|\mathsf{Spatial}(\varphi_2)| > 1$ **then**
3   **foreach** *spatial atom* A *of* $\mathsf{Spatial}(\varphi_2)$ *different from* `emp` **do**
4       $(\varphi^A, \mathcal{C}_r^A, \mathcal{C}_g^A) \leftarrow \mathsf{slice}(\varphi_1, \exists\vec{X}.\ A)$;
5   **if** $\varphi^A = \bot$ **then return** $\bot$;
6     $\mathcal{C}_r \leftarrow \bigwedge_A \mathcal{C}_r^A$; $\mathcal{C}_g \leftarrow \bigwedge_A \mathcal{C}_g^A$;
7   **if** $\mathsf{Pure}(\varphi_1) \wedge \mathcal{C}_r \models \mathsf{Pure}(\varphi_2) \wedge \mathcal{C}_g$ *and* $\forall A, B.\ \varphi_A \bowtie \varphi_B$ **then**
8   **return** $(\ \mathsf{Pure}(\varphi_1) \wedge \star_A\ \varphi_A,\ \mathcal{C}_r,\ \mathcal{C}_g\ )$
9   **else**
10     $(A, \mathcal{C}_r, \mathcal{C}_g) \leftarrow \mathsf{matchAtom}(\varphi_1, \exists\vec{X}.\ \mathsf{Spatial}(\varphi_2))$;
11   **if** $A \neq \bot$ *and* $\mathsf{Pure}(\varphi_1) \wedge \mathcal{C}_r \models \mathsf{Pure}(\varphi_2) \wedge \mathcal{C}_g$ **then**
12   **return** $(A, \mathcal{C}_r, \mathcal{C}_g)$
13   **else if** $A = \bot$ *and* $\mathsf{Spatial}(\varphi_2)$ *is a points-to atom* **then return** $\bot$;
14   **else if** $A = \bot$ *and* $\mathsf{Spatial}(\varphi_2)$ *is a predicate atom, say* $P(E, \vec{F})$ **then**
15   **foreach** *lemma* $L \triangleq \exists\vec{Z}.\ \Pi \wedge \Sigma \Rightarrow P(E, \vec{F})$ **do**
16         $(A_1, \mathcal{C}_r, \mathcal{C}_g) \leftarrow \mathsf{matchAtom}(\varphi_1, \exists\vec{X}\exists\vec{Z}.\ root(L))$;
17   **if** $A_1 \neq \bot$ **then**
18           $\exists\vec{Z}'.\Pi' \wedge \Sigma' \leftarrow \mathsf{quantElmt}(\,\exists\vec{X}\exists\vec{Z}.\Pi \wedge (\Sigma \setminus root(L)),\ \mathcal{C}_r\,)$;
19           $(\varphi, \mathcal{C}_r', \mathcal{C}_g') \leftarrow \mathsf{slice}(\varphi_1 \setminus A_1, \exists\vec{Z}'.\ \Pi' \wedge \Sigma')$;
20   **if** $\varphi \neq \bot$ *and* $\mathsf{Pure}(\varphi_1) \wedge \mathcal{C}_r \wedge \mathcal{C}_r' \wedge \Pi \models \mathsf{Pure}(\varphi_2) \wedge \mathcal{C}_g \wedge \mathcal{C}_g'$ **then**
21   **return** $(\,\mathsf{Pure}(\varphi_1) \wedge A_1 * \varphi,\ \mathcal{C}_r \wedge \mathcal{C}_r' \wedge \Pi,\ \mathcal{C}_g \wedge \mathcal{C}_g')$;
22   **return** $\bot$

Algorithm 1: The procedure slice. Given a formula $\varphi \triangleq \exists\vec{X}.\ \Pi \wedge \Sigma$, $\mathsf{Pure}(\varphi) = \Pi$ and $\mathsf{Spatial}(\varphi) = \Sigma$. The size of Σ, $|\Sigma|$, is the number of its atoms. Also, $\varphi_A \bowtie \varphi_B$ denotes the fact that φ_A and φ_B don't share spatial atoms.

deduced from the matching to $\mathcal{C}_g$, if it involves only the free variables of φ_2, and to $\mathcal{C}_r$, otherwise.

The output of $\mathsf{matchAtom}$ for predicate atoms is computed in a similar way: The predicate atom from φ_2, say $P(E', \vec{F}')$, is matched to some atom $P(E, \vec{F})$ from φ_1 such that $\mathsf{Pure}(\varphi_1) \models E = E'$. For instance, $\mathsf{matchAtom}$ called with

$$\varphi_1 ::= X = Y \wedge w = w' \wedge P(E, Y, v, w) \text{ and } \varphi_2 ::= \exists v'.\ P(E, X, v', w'),$$

returns the spatial atom of φ_1 and the same rely and guarantee as above.

If φ_2 contains a predicate atom, say $P(E, \vec{F})$, which doesn't match an atom of φ_1 (line 14), slice proceeds by applying *lemmas* (line 15). Straightforward lemmas correspond to the inductive rules defining the predicate P: every rule $P(E, \vec{F}) ::= \exists\vec{Z}.\Pi \wedge \Sigma$ defines a lemma $\exists\vec{Z}.\Pi \wedge \Sigma \Rightarrow P(E, \vec{F})$. More complex lemmas are defined in Sec. 5 and 6.

Essentially, applying a lemma $L \triangleq \varphi_L \Rightarrow P(E, \vec{F})$ means that the initial proof goal, i.e., proving entailment of $\exists\vec{X}.\ P(E, \vec{F})$, is reduced to proving the entailment of $\exists\vec{X}.\varphi_L$. The formula φ_L may contain existentially-quantified *location* variables. Finding suitable instantiations for these variables relies on a natural assumption that φ_L contains a unique spatial atom, denoted by $root(L)$, rooted at E (either a points-to atom $E \mapsto \rho$ or a predicate atom $Q(E, \ldots)$), that includes the occurrences of all the root variables of the points-to atoms and all the first parameters of the predicate atoms. This assumption holds for all the inductive rules defining predicates in our fragment (a consequence of the connectedness constraint) and for all the lemmas defined in Sec. 5 and 6. The atom $root(L)$ is matched with an atom rooted at E from φ_1 using the function matchAtom (line 16). When this matching is possible, matchAtom returns a spatial atom A_1 of φ_1 rooted at some E', with $E' = E$ implied by $\mathsf{Pure}(\varphi_1)$, together with rely and guarantee constraints.

The rely constraints $\mathcal{C}_r$ returned by matchAtom are used to eliminate some of the existential variables from the current goal $\exists \vec{X} \exists \vec{Z}.\ \Pi \wedge (\Sigma \setminus root(L))$. The atom $root(L)$ is removed from φ_L since it has been already matched to an atom of φ_1. The procedure quantElmt is responsible for this quantifier elimination and returns a formula $\exists \vec{Z}'.\ \Pi' \wedge \Sigma'$ with $\vec{Z}' \subseteq \vec{Z}$, which is equivalent to $\exists \vec{X} \exists \vec{Z}.\ \Pi \wedge (\Sigma \setminus root(L)) \wedge \mathcal{C}_r$ (line 18). For instance, the procedure quantElmt can substitute a quantified variable X_i with a variable Y occurring in φ_1, if $X_i = Y$ is a conjunct of $\mathcal{C}_r$. Then, the procedure slice is called recursively on the sub-formula $\varphi_1 \setminus A_1$ and the simplified consequence $\exists \vec{Z}'.\ \Pi' \wedge \Sigma'$. The final output of this case is defined at line 21.

To exemplify the use of the lemmas, consider the following input of slice corresponding to the entailment stating that two cells linked by the `next` pointer field, and storing ordered data values, form a sorted list segment:

$$\begin{aligned}
\varphi_1 &::= x_1 \neq \mathsf{nil} \wedge x_2 \neq \mathsf{nil} \wedge v_1 < v_2 \wedge x_1 \mapsto \{(\mathtt{next}, x_2), (\mathtt{data}, v_1)\} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad * \; x_2 \mapsto \{(\mathtt{next}, \mathsf{nil}), (\mathtt{data}, v_2)\} \\
\varphi_2 &::= \exists M.\ lseg(x_1, M, \mathsf{nil}, \emptyset) \wedge v_2 \in M,
\end{aligned}$$

where $lseg$ has been defined in Sec. 1 (eq. (1)–(2)). The first lemma to be applied corresponds to the inductive rule of $lseg$, i.e., eq. (2) (page II): $\exists X, M_1, v.\ x_1 \mapsto \{(\mathtt{next}, X), (\mathtt{data}, v)\} * lseg(X, M_1, \mathsf{nil}, \emptyset) \wedge M = \{v\} \cup M_1 \wedge v \leq M_1 \Rightarrow lseg(x_1, M, \mathsf{nil}, \emptyset)$. Therefore, the input of matchAtom is φ_1 and the atom $\exists X, v.\ x_1 \mapsto \{(\mathtt{next}, X), (\mathtt{data}, v)\}$ from the lemma. The output of matchAtom is the points-to atom of φ_1 rooted at x_1, the rely $\mathcal{C}_r : X = x_2 \wedge v = v_1$, and $\mathcal{C}_g$ is empty (true). The rely $\mathcal{C}_r$ is used to eliminate the quantifiers over X and v, the quantifier $\exists M$ being eliminated by simply deleting the constraint $M = \{v_1\} \cup M_1$, and slice is called at line 19 with inputs

$$\begin{aligned}
\varphi_1' &::= x_1 \neq \mathsf{nil} \wedge x_2 \neq \mathsf{nil} \wedge v_1 < v_2 \wedge x_2 \mapsto \{(\mathtt{next}, \mathsf{nil}), (\mathtt{data}, v_2)\} \\
\varphi_2' &::= \exists M_1.\ lseg(x_2, M_1, \mathsf{nil}, \emptyset) \wedge v_1 \leq M_1
\end{aligned}$$

This recursive call returns the whole formula φ_1' together with the rely $\mathcal{C}_r' : M_1 = \{v_2\}$, and an empty guarantee. Note that the conjunction of $\mathcal{C}_r$, $\mathcal{C}_r'$, and the pure part of the lemma, in particular, the constraint $M = \{v\} \cup M_1$, implies the constraint $v_2 \in M$ from φ_2. Therefore, the entailment $\varphi_1 \Rightarrow \varphi_2$ holds. The full explanation of this examples is given in Appendix A.

The following result states the correctness of slice. Moreover, since we assume a finite set of lemmas, and every application of a lemma L removes one spatial atom from φ_1 (the atom matched to $root(L)$), the termination of slice is guaranteed. In general, slice is incomplete.

Theorem 1. *Let* φ_1 *and* $\exists\vec{X}.\varphi_2$ *be two formulas. If* $\mathsf{slice}(\varphi_1, \exists\vec{X}.\varphi_2) = (\varphi_1, \mathcal{C}_r, \mathcal{C}_g)$*, then* $\varphi_1 \Rightarrow \exists\vec{X}.\varphi_2$*.*

5 Composition Lemmas

As we have seen in the motivating example, the predicate $bsthole(E, M_1, F, M_2)$ satisfies the property that composing two heap structures described by this predicate results in a heap structure satisfying the same predicate. We call this property a *composition lemma*. We define simple and uniform syntactic criteria which, if they are satisfied by a predicate, then the composition lemma holds. Further extensions to allow, e.g., trees with parent node, are discussed in Appendix B.

The main idea is to divide the parameters of inductively defined predicates into three categories: The *source* parameters $\vec{\alpha} = (E, C)$, the *hole* parameters $\vec{\beta} = (F, H)$, and the *static* parameters $\vec{\xi} \in \mathsf{Var}^*$, where $E, F \in \mathsf{LVar}$ are called the source and resp., the hole location parameter, and $C, H \in \mathsf{DVar}$ are called the cumulative and resp., the hole data parameter[5].

Let $\mathcal{P}$ be a set of inductively defined predicates and $P \in \mathcal{P}$ with the parameters $(\vec{\alpha}, \vec{\beta}, \vec{\xi})$. Then P is said to be *syntactically compositional* if the inductive definition of P contains *exactly one base rule*, and *at least one inductive rule*, and the rules of P are of one of the following forms:

– Base rule: $P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) ::= \bigwedge\limits_{1=i}^{2} \alpha_i = \beta_i \wedge \mathtt{emp}$,
– Inductive rule: $P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) ::= \exists\vec{Z}.\ \Pi \wedge \Sigma$, with (a) $\Sigma \triangleq E \mapsto \rho * \Sigma_r * P(\vec{\gamma}, \vec{\beta}, \vec{\xi})$, (b) Σ_r contains only predicate atoms, (c) $\gamma \subseteq \vec{Z}$, and (d) the variables in $\vec{\beta}$ don't occur elsewhere in $\Pi \wedge \Sigma$.

$P \in \mathcal{P}$ with the parameters $(\vec{\alpha}, \vec{\beta}, \vec{\xi})$ is said to be *semantically compositional* if the entailment $\exists\beta.\ P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) * P(\vec{\beta}, \vec{\gamma}, \vec{\xi}) \Rightarrow P(\vec{\alpha}, \vec{\gamma}, \vec{\xi})$ holds.

Theorem 2. *Let* $\mathcal{P}$ *be a set of inductively defined predicates. If* $P \in \mathcal{P}$ *is syntactically compositional, then* P *is semantically compositional.*

The proof of Theorem 2 is done by induction on the size of the domain of the heap structures as follows. Suppose $(s, h) \models P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) * P(\vec{\beta}, \vec{\gamma}, \vec{\xi})$, then either $s(\vec{\alpha}) = s(\vec{\beta})$ or $s(\vec{\alpha}) \neq s(\vec{\beta})$. If the former situation occurs, then $(s, h) \models P(\vec{\alpha}, \vec{\gamma}, \vec{\xi})$ follows immediately. Otherwise, the predicate $P(\vec{\alpha}, \vec{\beta}, \vec{\xi})$ is unfolded by using some inductive rule of P, and the induction hypothesis can be applied to a sub-heap of smaller size. The fact $(s, h) \models P(\vec{\alpha}, \vec{\gamma}, \vec{\xi})$ is deduced from the hypothesis, i.e., P is syntactically compositional.

Remark 1. The static parameters are useful to define universal properties of the data structures, e.g., all the data values in the binary search tree are greater than a data value represented by the static data parameter v.

[5] For simplicity, we assume that $\vec{\alpha}$ and $\vec{\beta}$ consist of exactly one location parameter and one data parameter.

6 Derived Lemmas

Theorem 2 provides a mean to obtain lemmas for one single syntactically compositional predicate. In the following, based on the syntactic compositionality, we demonstrate how to derive additional lemmas describing relationships between different predicates. We identify three categories of derived lemmas: "completion" lemmas, "stronger" lemmas, and "static-parameter contraction" lemmas.

6.1 The "completion" lemmas

We first consider the "completion" lemmas which describe relationships between incomplete data structures (e.g. binary search trees with one hole) and complete data structures (e.g. binary search trees). For example, the following lemma exists for the predicate *bsthole* and *bst*:

$$\exists F, M_2.\ bsthole(E, M_1, F, M_2) * bst(F, M_2) \Rightarrow bst(E, M_1).$$

Moreover, notice that the rules (i.e., lemmas) in the definition of *bst* can be obtained from those of *bsthole* by replacing (F, M_2) with $(\mathsf{nil}, \emptyset)$, and M_1 with M. These observations can be generalized to arbitrary syntactically compositional predicates as follows.

Let $P \in \mathcal{P}$ be a syntactically compositional predicate with the parameters $(\vec{\alpha}, \vec{\beta}, \vec{\xi})$, and $P' \in \mathcal{P}$ a predicate with the parameters $(\vec{\alpha}, \vec{\xi})$. Then P' is said to be a *completion* of P with respect to a pair of constants $\vec{c} = c_1 c_2$ if the rules of P' are obtained from the rules of P by setting $\vec{\beta} = \vec{c}$. More precisely,

- P' contains only one base rule, and this base rule is obtained from the base rule of P by replacing β_i with c_i for each $i : 1 \leq i \leq 2$,
- for each inductive rule of P', say $P'(\vec{\alpha}, \vec{\xi}) ::= \exists \vec{Z'}.\Pi' \wedge \Sigma'$, there exists a rule of P of the form $P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) ::= \exists \vec{Z}.\Pi \wedge E \mapsto \rho * \Sigma_r * P(\vec{\gamma}, \vec{\beta}, \vec{\xi})$, s.t. $|\vec{Z'}| = |\vec{Z}|$ and $\Pi' \wedge \Sigma'$ is $(\Pi \wedge E \mapsto \rho * \Sigma_r * P'(\vec{\gamma}, \vec{\xi}))[\vec{c}/\vec{\beta}, \vec{Z'}/\vec{Z}]$,
- for each inductive rule of P, say $P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) ::= \exists \vec{Z}.\Pi \wedge E \mapsto \rho * \Sigma_r * P(\vec{\gamma}, \vec{\beta}, \vec{\xi})$, there is an inductive rule of P' of the form $P'(\vec{\alpha}, \vec{\xi}) ::= \exists \vec{Z'}.\Pi' \wedge \Sigma'$, satisfying the same conditions as above.

Note that in the above definition, the occurrences of P in Σ_r (if there are any) are not replaced by P'.

Theorem 3. *Let $P \in \mathcal{P}$ be a syntactically compositional predicate with the parameters $(\vec{\alpha}, \vec{\beta}, \vec{\xi})$, and $P' \in \mathcal{P}$ with the parameters $(\vec{\alpha}, \vec{\xi})$. If P' is a completion of P with respect to $\vec{c}$, then $P'(\vec{\alpha}, \vec{\xi}) \Leftrightarrow P(\vec{\alpha}, \vec{c}, \vec{\xi})$ and $\exists \vec{\beta}.\ P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) * P'(\vec{\beta}, \vec{\xi}) \Rightarrow P'(\vec{\alpha}, \vec{\xi})$ hold.*

6.2 The "stronger" lemmas

We illustrate this class of lemmas on the example of BST. Let $natbsth(E, M_1, F, M_2)$ be the predicate defined by the same rules as $bsthole(E, M_1, F, M_2)$ (i.e., eq. (6)–(8)), except that $M_3 \geq 0$ is added to the body of each inductive rule (i.e., eq. (7) and (8)). Then we say that *natbsth* is *stronger* than *bsthole*, since for each rule R of *natbsth*, there is a rule R' of *bsthole*, such that the body of R entails the body of R'. This "stronger" relation guarantees that the following lemma hold:

$$natbsth(E, M_1, F, M_2) \Rightarrow bsthole(E, M_1, F, M_2)$$

$$\exists E_2, M_2.\ natbsth(E_1, M_1, E_2, M_2) * bsthole(E_2, M_2, E_3, M_3) \Rightarrow bsthole(E_1, M_1, E_3, M_3).$$

In general, for two syntactically compositional predicates $P, P' \in \mathcal{P}$ with the same set of parameters $(\vec{\alpha}, \vec{\beta}, \vec{\xi})$, P' is said to be *stronger* than P if for each inductive rule $P'(\vec{\alpha}, \vec{\beta}, \vec{\xi}) ::= \exists \vec{Z}.\ \Pi' \wedge E \mapsto \rho * \Sigma_r * P'(\vec{\gamma}, \vec{\beta}, \vec{\xi})$, there is an inductive rule $P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) ::= \exists \vec{Z}.\ \Pi \wedge E \mapsto \rho * \Sigma_r * P(\vec{\gamma}, \vec{\beta}, \vec{\xi})$ such that $\Pi' \Rightarrow \Pi$ holds. The following result is a consequence of Thm. 2.

Theorem 4. *Let $P, P' \in \mathcal{P}$ be two syntactically compositional predicates with the same set of parameters $(\vec{\alpha}, \vec{\beta}, \vec{\xi})$. If P' is stronger than P, then the entailments $P'(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \Rightarrow P(\vec{\alpha}, \vec{\beta}, \vec{\xi})$ and $\exists \vec{\beta}.\ P'(\vec{\alpha}, \vec{\beta}, \vec{\xi}) * P(\vec{\beta}, \vec{\gamma}, \vec{\xi}) \Rightarrow P(\vec{\alpha}, \vec{\gamma}, \vec{\xi})$ hold.*

6.3 The "static-parameter contraction" lemmas

Let *tailbsth*(E, M_1, F, M_2) be the predicate defined by the same rules as *bsthole*(E, M_1, F, M_2), with the modification that the points-to atom in each inductive rule is replaced by $E \mapsto \{(\texttt{left}, X), (\texttt{right}, Y), (\texttt{tail}, F), (\texttt{data}, v)\}$. Notice that *tailbsth* is not syntactically compositional since F occurs in the points-to atom of the inductive rules. Moreover, let *stabsth*(E, M_1, F, M_2, B) be the predicate defined by the same rules as *bsthole*(E, M_1, F, M_2), with the modification that the points-to atom in each inductive rule is replaced by $E \mapsto \{(\texttt{left}, X), (\texttt{right}, Y), (\texttt{tail}, B), (\texttt{data}, v)\}$, and the atom *bsthole*$(Y, M_4, F, M_2)$ (resp. *bsthole*(X, M_3, F, M_2)) is replaced by *stabsth*(Y, M_4, F, M_2, B) (resp. *stabsth*(X, M_3, F, M_2, B)). Clearly, the predicate *stabsth* is syntactically compositional.

From the above description, it is easy to observe that the inductive definition of *tailbsth*(E, M_1, F, M_2) can be obtained from that of *stabsth*(E, M_1, F, M_2, B) by replacing B with F. Then the lemma *tailbsth*$(E, M_1, F, M_2) \Leftrightarrow$ *stabsth*(E, M_1, F, M_2, F) holds. From this, we further deduce the lemma

$$\exists E_2, M_2.\ stabsth(E_1, M_1, E_2, M_2, E_3) * tailbsth(E_2, M_2, E_3, M_3) \Rightarrow tailbsth(E_1, M_1, E_3, M_3).$$

We call the replacement of B by F in the inductive definition of *stabsth* the "static-parameter contraction". This idea can be generalized to arbitrary syntactically compositional predicates as follows.

Let $P \in \mathcal{P}$ be a syntactically compositional predicate with the parameters $(\vec{\alpha}, \vec{\beta}, \vec{\xi})$, $P' \in \mathcal{P}$ be an inductive predicate with the parameters $(\vec{\alpha}, \vec{\beta}, \vec{\xi'})$, $\vec{\xi} = \xi_1 \dots \xi_k$, and $\vec{\xi'} = \xi'_1 \dots \xi'_l$. Then P' is said to be a *static-parameter contraction* of P if the rules of P' are obtained from those of P by setting ξ_i with β_j for some $\xi_i \in \vec{\xi}$ and $\beta_j \in \vec{\beta}$ such that ξ_i and β_j have the same data type. More precisely, the base rules of P' are those of P, in addition, there is a function $prj : \{1, \dots, k\} \to \{0, 1, 2\}$ such that the following conditions hold.

– $|prj^{-1}(0)| = l$ and $\vec{\xi'} = prj_0(\vec{\xi})$, where $prj_0(\vec{\xi})$ is the tuple obtained from $\vec{\xi}$ by keeping the elements ξ_i such that $prj(i) = 0$ and removing all the others.
– For each $i : 1 \leq i \leq k$ s.t. $prj(i) \neq 0$, ξ_i and $\beta_{prj(i)}$ have the same data type.
– For each inductive rule of P', say $P'(\vec{\alpha}, \vec{\beta}, \vec{\xi'}) ::= \exists \vec{Z}. \Pi' \wedge \Sigma'$, there is an inductive rule of P of the form $P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) ::= \exists \vec{Z}. \Pi \wedge E \mapsto \rho * \Sigma_r * P(\vec{\gamma}, \vec{\beta}, \vec{\xi})$, s.t. $\Pi' \wedge \Sigma'$ is obtained from $\Pi \wedge E \mapsto \rho * \Sigma_r * P(\vec{\gamma}, \vec{\beta}, \vec{\xi})$ by first replacing $P(\vec{\gamma}, \vec{\beta}, \vec{\xi})$ with $P'(\vec{\gamma}, \vec{\beta}, \vec{\xi'})$, then replacing ξ_i with $\beta_{prj(i)}$ for each $i : 1 \leq i \leq k$ such that $prj(i) \neq 0$.
– For each inductive rule of P, say $P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) ::= \exists \vec{Z}. \Pi \wedge E \mapsto \rho * \Sigma_r * P(\vec{\gamma}, \vec{\beta}, \vec{\xi})$, there is an inductive rule of P' of the form $P'(\vec{\alpha}, \vec{\beta}, \vec{\xi'}) ::= \exists \vec{Z}. \Pi' \wedge \Sigma'$, satisfying the same conditions as above.

The function prj is called the *contraction function* of the static-parameter contraction. Notice that in the above definition, the occurrences of P in Σ_r (if there are any) are not replaced by P'.

Suppose $\vec{\beta} \in \mathsf{LVar} \times \mathsf{DVar}$, $\vec{\xi'} = \xi'_1 \ldots \xi'_l$, $prj : \{1, \ldots, k\} \to \{0, 1, 2\}$ such that $|prj^{-1}(0)| = l$, then the $(prj, \vec{\beta})$-*extension* of $\vec{\xi'}$, denoted by $ext_{(prj, \vec{\beta})}(\vec{\xi'})$, is defined as $\vec{\xi} = \xi_1 \ldots \xi_k$ such that $prj_0(\vec{\xi}) = \vec{\xi'}$ and for each position $i : 1 \leq i \leq k$ such that $prj(i) \neq 0$, $\xi_i = \beta_{prj(i)}$.

Theorem 5. *Let $P \in \mathcal{P}$ be a syntactically compositional predicate with the parameters $(\vec{\alpha}, \vec{\beta}, \vec{\xi})$ and $P' \in \mathcal{P}$ be an inductive predicate with the parameters $(\vec{\alpha}, \vec{\beta}, \vec{\xi'})$. If P' is a static-parameter contraction of P with the contraction function prj, then $P'(\vec{\alpha}, \vec{\beta}, \vec{\xi'}) \Leftrightarrow P(\vec{\alpha}, \vec{\beta}, ext_{prj, \vec{\beta}}(\vec{\xi'}))$ and $\exists \vec{\beta}.\ P(\vec{\alpha}, \vec{\beta}, ext_{(prj, \vec{\gamma})}(\vec{\xi'})) * P'(\vec{\beta}, \vec{\gamma}, \vec{\xi'}) \Rightarrow P'(\vec{\alpha}, \vec{\gamma}, \vec{\xi'})$ hold.*

7 Experimental results

We have extended our tool SPEN [26] with the proof strategy proposed in this paper. The entailments are written in an extension of the SMTLIB format used in the competition SL-COMP'14 for solvers for separation logic. It provides as output SAT, UNSAT or UNKNOWN, and a diagnosis for all these cases.

The solver starts with a normalization step, based on the boolean abstractions described in [11], which saturates the input formulas with (dis)equalities between location variables implied by the semantics of separating conjunction. The entailments of data constraints are translated into satisfiability problems in the theory of arrays, discharged using the SMT solver Z3 [10].

We have applied the approach proposed on two sets of problems[6]:

RDBI: verification conditions for proving the correctness of iterative procedures (delete, insert, search) over data structures storing integer data: sorted lists, binary search trees (BST), AVL trees, and red black trees (RBT).

SL-COMP'14: pure shape problems in the SL-COMP'14 benchmark. These problems use syntactically compositional inductive predicates.

[6] `http://www.liafa.univ-paris-diderot.fr/spen/benchmarks.html`

Table 2. Experimental results on benchmark RDBI

Data structure	Procedure	#VC	Lemma (#b, #r, #p, #c, #d)	$\Rightarrow_{\mathbb{D}}$	Time (s) SPEN	Z3
sorted lists	`search`	4	(6, 8, 17, 3, 1)	6	0.10	0.05
	`insert`	8	(12, 18, 30, 12, 0)	20	0.33	0.10
	`delete`	4	(6, 10, 16, 6, 1)	10	0.15	0.05
BST	`search`	4	(9, 11, 27, 9, 1)	6	0.20	0.05
	`insert`	14	(18, 21, 33, 14, 0)	24	0.63	0.20
	`delete`	25	(30, 37, 106, 25, 0)	68	1.49	0.51
AVL	`search`	4	(10, 12, 17, 13, 1)	6	0.23	0.15
	`insert`	22	(20, 27, 66, 23, 0)	48	1.94	0.63
RBT	`search`	4	(9, 12, 27, 12, 1)	6	0.23	0.15
	`insert`	21	(32, 25, 126, 22, 0)	78	2.94	0.93

Table 3. Experimental results on benchmark SL-COMP'14

Data structure	#VC	Lemma (#b, #r, #p, #c, #d)	Time-SPEN(s) SPEN-comp	SPEN-TA
Nested linked lists	16	(17,47,14,8,0)	0.81	0.65
Skip lists 2 levels	4	(5,11,1,1,0)	0.18	0.18
Skip lists 3 levels	10	(16,32,29,17,0)	0.47	0.42

Table 2 provides experimental data[7] for **RDBI**. The column #VC gives the number of verification conditions considered for each procedure. The column Lemma summarizes the number and the type of lemmas applied for each set of problems: #b and #r are the number lemmas corresponding to base and resp., inductive rules, #c and #d are the number of composition and resp., derived lemmas, and #p is the number of predicates matched syntactically, without applying lemmas. Also, $\Rightarrow_{\mathbb{D}}$ provides the number of entailments between pure constraints generated by SPEN; Time-SPEN gives the "wall clock time" spent by SPEN on all problems excepting the time taken to solve the data constraints by Z3, which is given on the column Time-Z3.

For the benchmark **SL-COMP'14**, Table 3 provides for comparison the time spent by the decision procedure [11] on the same set of problem.

8 Related work

There have been many works on the verification of programs manipulating mutable data structures in general and the use of separation logic, e.g., [1–5, 7–9, 11–18, 21, 22, 24, 27]. In the following, we discuss those which are closer to our approach.

The prover SLEEK [7, 18] and the natural proof approach DRYAD [20, 23] provide proof strategies for proving entailments of SL formulas. These strategies are also based on lemmas, relating inductive definitions, but differently from our approach, these lemmas are supposed to be given by the user (SLEEK can prove

[7] Our experiments were performed on an Intel Core 2 Duo2.53 GHz processor with 4 GiB DDR3 1067 MHz running a virtual machine with Fedora 20 (64-bit).

the correctness of the lemmas once they are provided). Our approach is able to discover and synthesize the lemmas systematically, efficiently, and automatically. Furthermore, the inductive definitions used in our paper enable succinct lemmas, far less complex than for instance, the lemmas used in DRYAD, which include complex constraints on data variables and the magic wand.

The method of cyclic proofs [5] can prove the entailment of two SL formulas describing relationships between inductive predicates (or lemmas in our terminology) by using induction on the size of the heaps, but it is the users' task to input these formulas. In addition, their lemmas focus on the shape part, but do not contain data or size constraints. However, there are examples of intricate lemmas which can be proved using the cyclic proofs methodology but not with our approach, e.g., lemmas concerning the predicate $RList$ which is defined by unfolding the list segments from the end (instead of the beginning).

The tool SLIDE [14, 15] provides decision procedures for fragments of SL based on reductions to the language inclusion problem of tree automata. Their fragments contain no data or size constraints. In addition, the EXPTIME lower bound complexity is an important obstacle for scalability. Our previous work [11] introduces a decision procedure based on reductions to the membership problem of tree automata which however is not capable of dealing with data constraints.

The tool GRASShopper [21, 22] is based on translations of SL fragments to first-order logic with reachability predicates, and the use of SMT solvers to deal with the latter. The advantage is the integration with other SMT theories to reason about data. However, this approach considers a very limited class of inductive definitions, for linked lists and trees. It is unclear whether it can be generalized to allow user-defined inductive predicates, as in our work.

The truncation point approach [12] provides a method to specify and verify programs based on separation logic with inductive definitions. The approach allows inductive definitions describing truncated data structures with multiple holes, but it cannot deal with data constraints. Our approach can also be extended to cover such inductive definitions.

9 Conclusion

We proposed a novel approach for automating program proofs based on Separation Logic with inductive definitions. This approach consists of (1) efficiently checkable criteria for recognizing inductive definitions that satisfy crucial lemmas in such proofs and (2) a novel proof strategy for applying these lemmas. The proof strategy relies on syntactic matchings of spatial atoms and SMT solvers for dealing with data constraints. We have implemented the approach as an extension of our tool SPEN and applied it successfully to a representative set of examples, coming from iterative procedures for binary search trees or lists.

In the future, we plan to investigate extensions of our approach to formulas with a more general boolean structure or using more general inductive definitions. Concerning the latter, we plan to investigate whether some ideas from [23] could be used to extend our proof strategy. From a practical point of view, apart from improving the implementation of our proof strategy, we plan to integrate it into the program analysis framework Celia [6].

References

1. Parosh Aziz Abdulla, Luks Holk, Bengt Jonsson, Ondrej Lengl, Cong Quy Trinh, and Toms Vojnar. Verification of heap manipulating programs with ordered data by extended forest automata. In *ATVA 2013*, pages 224–239, 2013.
2. Timos Antonopoulos, Nikos Gorogiannis, Christoph Haase, Max I. Kanovich, and Joël Ouaknine. Foundations for decision problems in separation logic with general inductive predicates. In *FoSSaCS*, pages 411–425, 2014.
3. Ittai Balaban, Amir Pnueli, and Lenore D. Zuck. Shape analysis by predicate abstraction. In *VMCAI*, pages 164–180, 2005.
4. Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. Symbolic execution with separation logic. In *APLAS'05*, pages 52–68, 2005.
5. James Brotherston, Dino Distefano, and Rasmus Lerchedahl Petersen. Automated cyclic entailment proofs in separation logic. In *Proceedings of the 23rd International Conference on Automated Deduction*, CADE'11, pages 131–146, 2011.
6. CELIA. `http://www.liafa.univ-paris-diderot.fr/celia`.
7. Wei-Ngan Chin, Cristina David, Huu Hai Nguyen, and Shengchao Qin. Automated verification of shape, size and bag properties via user-defined predicates in separation logic. *Sci. Comput. Program.*, 77(9):1006–1036, 2012.
8. Adam Chlipala. Mostly-automated verification of low-level programs in computational separation logic. *SIGPLAN Not.*, 46(6):234–245, 2011.
9. Byron Cook, Christoph Haase, Joël Ouaknine, Matthew Parkinson, and James Worrell. Tractable reasoning in a fragment of separation logic. In *Proceedings of the 22Nd International Conference on Concurrency Theory*, CONCUR'11, pages 235–249, 2011.
10. Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.
11. Constantin Enea, Ondřej Lengál, Mihaela Sighireanu, and Tomáš Vojnar. Compositional entailment checking for a fragment of separation logic. In *APLAS*, Lecture Notes in Computer Science. Springer, 2014.
12. Bolei Guo, Neil Vachharajani, and David I. August. Shape analysis with inductive recursion synthesis. In *PLDI*, pages 256–265, 2007.
13. Lukáš Holík, Ondřej Lengál, Adam Rogalewicz, Jiří Šimáček, and Tomáš Vojnar. Fully automated shape analysis based on forest automata. In *CAV'13*, pages 740–755, 2013.
14. Radu Iosif, Adam Rogalewicz, and Jiri Simacek. The tree width of separation logic with recursive definitions. In *Proceedings of the 24th International Conference on Automated Deduction*, CADE'13, pages 21–38, 2013.
15. Radu Iosif, Adam Rogalewicz, and Tomás Vojnar. Deciding entailments in inductive separation logic with tree automata. In *ATVA 2014*, pages 201–218, 2014.
16. Shachar Itzhaky, Anindya Banerjee, Neil Immerman, Ori Lahav, Aleksandar Nanevski, and Mooly Sagiv. Modular reasoning about heap paths via effectively propositional formulas. In *POPL 2014*, pages 385–396, 2014.
17. Shuvendu Lahiri and Shaz Qadeer. Back to the future: Revisiting precise program verification using smt solvers. *SIGPLAN Not.*, 43(1):171–182, 2008.
18. Huu Hai Nguyen and Wei-Ngan Chin. Enhancing program verification with lemmas. In *Proceedings of the 20th International Conference on Computer Aided Verification*, CAV '08, pages 355–369, 2008.

19. Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In *CSL '01*, pages 1–19, 2001.
20. Edgar Pek, Xiaokang Qiu, and P. Madhusudan. Natural proofs for data structure manipulation in C using separation logic. *SIGPLAN Not.*, 49(6):440–451, June 2014.
21. Ruzica Piskac, Thomas Wies, and Damien Zufferey. Automating separation logic using SMT. In *Proceedings of the 25th International Conference on Computer Aided Verification*, CAV'13, pages 773–789, 2013.
22. Ruzica Piskac, Thomas Wies, and Damien Zufferey. Automating separation logic with trees and data. In *CAV 2014*, pages 711–728, 2014.
23. Xiaokang Qiu, Pranav Garg, Andrei Stefănescu, and Parthasarathy Madhusudan. Natural proofs for structure, data, and separation. *SIGPLAN Not.*, 48(6):231–242, 2013.
24. Zvonimir Rakamaric, Jesse Bingham, and Alan J. Hu. An inference-rule-based decision procedure for verification of heap-manipulating programs with mutable data and cyclic data structures. In *VMCAI'07*, pages 106–121, 2007.
25. John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *LICS '02*, pages 55–74, 2002.
26. SPEN. `http://www.liafa.univ-paris-diderot.fr/spen`.
27. Karen Zee, Viktor Kuncak, and Martin Rinard. Full functional verification of linked data structures. In *PLDI '08*, pages 349–361, 2008.

A Full example of Section 4

We provide here the full execution of slice on the input considered in Section 4.

The input corresponds to the entailment stating that two cells linked by the `next` pointer field, and storing ordered data values, form a sorted list segment:

$$\varphi_1 ::= x_1 \neq \mathsf{nil} \wedge x_2 \neq \mathsf{nil} \wedge v_1 < v_2 \wedge x_1 \mapsto \{(\mathtt{next}, x_2), (\mathtt{data}, v_1)\} \\ * \ x_2 \mapsto \{(\mathtt{next}, \mathsf{nil}), (\mathtt{data}, v_2)\}$$

$$\varphi_2 ::= \exists M, M'.\ \mathit{lseg}(x_1, M, \mathsf{nil}, \emptyset) \wedge v_2 \in M,$$

where *lseg* has been defined in Sec. 1 (eq. (1)–(2)).

The first recursive call of slice: Because φ_2 has only one spatial atom ($\mathit{lseg}(x_1, M, \mathsf{nil}, \emptyset)$) which fails to be matched to an atom of φ_1 by matchAtom at line 10, the lemmas of *lseg* are tried at line 15. The only lemma including a points-to atom from x_1 is the inductive rule of *lseg*, i.e., eq. (2) (page II), where the pure part of the lemma $\Pi^0 : v^1 \leq M_1^1 \wedge M = M_1^1 \cup \{v^1\}$.

Therefore, matchAtom is called at line 16 with the input φ_1 and $\exists X^1, v^1.\ x_1 \mapsto \{(\mathtt{next}, X^1), (\mathtt{data}, v^1)\}$ (α-conversion is applied to existential variables). The output of matchAtom is the points-to atom of φ_1 rooted at x_1, the rely $\mathcal{C}_r^1 : X^1 = x_2 \wedge v^1 = v_1$ and the guarantee $\mathcal{C}_g^1$ is empty (true). The rely $\mathcal{C}_r^1$ is used to eliminate the quantifiers over X^1 and v^1; in addition, quantifier over M are also eliminated to satisfy the constraint $\vec{Z'} \subseteq \vec{Z}$.

The second recursive call of slice: is done at line 19 with the following inputs:

$$\varphi_1^1 ::= x_1 \neq \mathsf{nil} \wedge x_2 \neq \mathsf{nil} \wedge v_1 < v_2 \wedge x_2 \mapsto \{(\mathtt{next}, \mathsf{nil}), (\mathtt{data}, v_2)\}$$

$$\varphi_2^1 ::= \exists M_1^1.\ \mathit{lseg}(x_2, M_1^1, \mathsf{nil}, \emptyset) \wedge\ v_1 \leq M_1^1.$$

By applying again lemma (2) to the atom $\mathit{lseg}(x_2, M_1^1, \mathsf{nil}, \emptyset)$, we have that $\Pi^1 : v^2 \leq M_1^2 \wedge M_1^1 = M_1^2 \cup \{v^2\}$. The matchAtom is called at line 16 with φ_1^1 and $\exists X^2, v^2 . x_2 \mapsto \{(\mathtt{next}, X^2), (\mathtt{data}, v^2)\}$. The output of matchAtom is the points-to atom of φ_1^1 rooted at x_2, the rely $\mathcal{C}_r^2 : X^2 = \mathsf{nil} \wedge v^2 = v_2$ and the empty guarantee $\mathcal{C}_g^2$. The quantifiers elimination at line 18 returns the simplified formula wrt $\mathcal{C}_r^2$:

$$\exists M_1^2.\ \mathit{lseg}(\mathsf{nil}, M_1^2, \mathsf{nil}, \emptyset) \wedge\ v_2 \leq M_1^2.$$

The third recursive call of slice: is done at line 19 with the following inputs:

$$\varphi_1^2 ::= x_1 \neq \mathsf{nil} \wedge x_2 \neq \mathsf{nil} \wedge v_1 < v_2 \wedge \mathtt{emp}$$

$$\varphi_2^2 ::= \exists M_1^2.\ \mathit{lseg}(\mathsf{nil}, M_2^1, \mathsf{nil}, \emptyset) \wedge\ v_2 \leq M_1^2.$$

The third lemma applied is the base rule (1) of *lseg*, which has $\Pi^2 : M_1^2 = \emptyset$ and matchAtom generates empty (true) rely $\mathcal{C}_r^3$ and guarantee $\mathcal{C}_g^3$.

The fourth (final) recursive call of slice: is done at line 19 with the following inputs:

$$\varphi_1^3 ::= x_1 \neq \mathsf{nil} \wedge x_2 \neq \mathsf{nil} \wedge v_1 < v_2 \wedge \mathtt{emp}$$

$$\varphi_2^3 ::= \mathtt{true} \wedge \mathtt{emp}$$

By matching the atom $\mathtt{emp}$ at line 10, it generates empty (true) $\mathcal{C}_r^4$ and $\mathcal{C}_g^4$. Because $\mathsf{Pure}(\varphi_1^3) \models \mathsf{Pure}(\varphi_2^3)$ is valid, the fourth recursive call returns at line 12, to the line 20 in the 3rd recursive call with the triple $(\mathtt{emp}, {\mathcal{C}'_r}^3 : \mathtt{true}, {\mathcal{C}'_g}^3 : \mathtt{true})$.

The constraint on data tested at line 20 in the 3rd recursive call is $\mathsf{Pure}(\varphi_1^2) \wedge {\mathcal{C}'_r}^3 \wedge \mathcal{C}_r^3 \wedge \Pi^2 \models \mathsf{Pure}(\varphi_2^2)$, i.e., $x_1 \neq \mathsf{nil} \wedge x_2 \neq \mathsf{nil} \wedge v_1 < v_2 \wedge M_1^2 = \emptyset \Rightarrow v_2 \leq M_1^2$, which is valid. Notice that $v_2 \leq \emptyset$ is trivially true. Then, the 3rd recursive call returns the triple $(\mathtt{emp}, {\mathcal{C}'_r}^2 : M_1^2 = \emptyset, {\mathcal{C}'_g}^2 : \mathtt{true})$ at line 20 of the second recursive call.

The constraint on data tested at line 20 in the 2nd recursive call is $\mathsf{Pure}(\varphi_1^1) \wedge {\mathcal{C}'_r}^2 \wedge \mathcal{C}_r^2 \wedge \Pi^1 \models \mathsf{Pure}(\varphi_2^1)$, i.e., $\mathsf{Pure}(\varphi_1^1) \wedge M_1^2 = \emptyset \wedge X^2 = \mathsf{nil} \wedge v^2 = v_2 \wedge v^2 \leq M_1^2 \wedge M_1^1 = M_1^2 \cup \{v^2\} \models v_1 \leq M_1^1$, which is also valid. Then, the 2nd recursive call returns the triple $(x_2 \mapsto \{(\mathtt{next}, \mathsf{nil}), (\mathtt{data}, v_2)\}, {\mathcal{C}'_r}^1 : \mathcal{C}_r^2 \wedge {\mathcal{C}'_r}^2 \wedge M_1^1 = \{v^2\} \cup M_1^2 \wedge v^2 \leq M_1^2, {\mathcal{C}'_g}^1 : \mathtt{true})$ at line 20 of the first recursive call.

The constraint on data tested at line 20 in the 1st recursive call is $\mathsf{Pure}(\varphi_1) \wedge {\mathcal{C}'_r}^1 \wedge \mathcal{C}_r^1 \wedge \Pi^0 \models \mathsf{Pure}(\varphi_2)$, i.e., $\mathsf{Pure}(\varphi_1) \wedge {\mathcal{C}'_r}^1 \wedge X^1 = x_2 \wedge v^1 = v_1 \wedge v^1 \leq M_1^1 \wedge M = M_1^1 \cup \{v^1\} \models v_2 \in M$ which is valid. The first call returns the triple $(x_1 \mapsto \{(\mathtt{next}, X^1), (\mathtt{data}, v^1) * x_2 \mapsto \{(\mathtt{next}, \mathsf{nil}), (\mathtt{data}, v_2)\}, \mathcal{C}'_r : \mathcal{C}_r^1 \wedge {\mathcal{C}'_r}^1 \wedge M = \{v^1\} \cup M_1^1 \wedge v^1 \leq M_1^1, \mathcal{C}'_g : \mathtt{true})$ at line 20 of the first recursive call.

B Extensions of the lemmas

In this section, we discuss how the the basic idea of syntactical compositionality can be extended in various ways.

Multiple location and data parameters.

At first, we would like to emphasize that although we restrict our discussions on compositional predicates $P(\vec{\alpha}, \vec{\beta}, \vec{\xi})$ to the special case that $\vec{\alpha}$ (resp. $\vec{\beta}$) contain only two parameters: one location parameter, and one data parameter. But all the results about the lemmas can be generalized smoothly to the situation that $\vec{\alpha}$ and $\vec{\beta}$ contain multiple location and data parameters.

Pseudo-composition lemmas.

We then consider syntactically pseudo-compositional predicates.

We still use the binary search trees to illustrate the idea.

Suppose *neqbsthole* is the predicate defined by the same rules as *bsthole*, with the modification that $E \neq F$ is added to the body of each inductive rule. Then *neqbsthole* is not syntactically compositional anymore and the composition lemma

$$\exists E_2, M_2.\ neqbsthole(E_1, M_1, E_2, M_2) * neqbsthole(E_2, M_2, E_3, M_3) \Rightarrow neqbsthole(E_1, M_1, E_3, M_3)$$

does not hold. This is explained as follows: Suppose $h = h_1 * h_2$ (where $h = h_1 * h_2$ denotes that h_1 and h_2 are domain disjoint and h is the union of h_1 and h_2), $(s, h_1) \models neqbsthole(E_1, M_1, E_2, M_2)$ and $(s, h_2) \models neqbsthole(E_2, M_2, E_3, M_3)$, in addition, both $\mathsf{ldom}(h_1)$ and $\mathsf{ldom}(h_2)$ are nonempty. Then from the inductive definition of *neqbsthole*, we deduce that $s(E_1) \neq s(E_2)$ and $s(E_2) \neq s(E_3)$. On the other hand, $(s, h) \models bsthole1(E_1, M_1, E_3, M_3)$ requires that $s(E_1) \neq s(E_3)$, which cannot be inferred from $s(E_1) \neq s(E_2)$ and $s(E_2) \neq s(E_3)$ in general. Nevertheless, the entailment

$$\begin{aligned}&\exists E_2, M_2. neqbsthole(E_1, M_1, E_2, M_2) * neqbsthole(E_2, M_2, E_3, M_3) * \\&\qquad E_3 \mapsto ((left, X), (right, Y), (data, v)) \Rightarrow \\&\qquad neqbsthole(E_1, M_1, E_3, M_3) * E_3 \mapsto ((left, X), (right, Y), (data, v))\end{aligned}$$

holds since the information $E_1 \neq E_3$ can be inferred from the fact that E_3 is allocated and separated from E_1. Therefore, intuitively, in this situation, the composition lemma can be applied under the condition that we already know that $E_1 \neq E_3$. We call this as pseudo-compositionality. Our decision procedure can be generalized to apply the pseudo-composition lemmas when proving the entailment of two formulas.

Data structures with parent pointers.

Next, we show how our ideas can be generalized to the data structures with parent pointers, e.g. doubly linked lists or trees with parent pointers. We use binary search trees with parent pointers to illustrate the idea. We can define the predicates $prtbst(E, Pr, M)$ and $prtbsthole(E, Pr_1, M_1, F, Pr_2, M_2)$ to describe respectively binary search trees with parent pointers and binary search trees with parent pointers and one hole. The intuition of E, F are still the source and the hole, while Pr and Pr_1 (resp. Pr_2) are the parent of E (resp. F) (the definition of $prtbst$ is omitted here).

$$\begin{aligned}prtbsthole(E, Pr_1, M_1, F, Pr_2, M_2) &::= E = F \wedge \mathtt{emp} \wedge Pr_1 = Pr_2 \wedge M_1 = M_2 \\prtbsthole(E, Pr_1, M_1, F, Pr_2, M_2) &::= \exists X, Y, M_3, M_4, v. \\&\quad E \mapsto \{(left, X), (right, Y), (parent, Pr_1), (data, v)\} \\&\quad * prtbst(X, E, M_3) * prtbsthole(Y, E, M_4, F, Pr_2, M_2) \\&\quad \wedge M_1 = \{v\} \cup M_3 \cup M_4 \wedge M_3 < v < M_4 \\prtbsthole(E, Pr_1, M_1, F, Pr_2, M_2) &::= \exists X, Y, M_3, M_4, v. \\&\quad E \mapsto \{(left, X), (right, Y), (parent, Pr_1), (data, v)\} \\&\quad * prtbsthole(X, E, M_3, F, Pr_2, M_2) * prtbst(Y, E, M_4) \\&\quad \wedge M_1 = \{v\} \cup M_3 \cup M_4 \wedge M_3 < v < M_4\end{aligned}$$

Then the predicate $prtbsthole$ enjoys the composition lemma

$$\begin{aligned}&\exists E_2, Pr_2, M_2.\ prtbsthole(E_1, Pr_1, M_1, E_2, Pr_2, M_2) * \\&\qquad prtbsthole(E_2, Pr_2, M_2, E_3, Pr_3, M_3) \Rightarrow \\&\qquad\qquad prtbsthole(E_1, Pr_1, M_1, E_3, Pr_3, M_3).\end{aligned}$$

Multiple points-to atoms in inductive rules.

In addition, we would like to remark that the constraint that each inductive rule contains only one points-to atom can be lifted, without affecting the compositionality. For instance, we can define the predicate $lsegeven$ for list segments of even length as follows,

$$\begin{aligned}lsegeven(E, F) &::= E = F \wedge \mathtt{emp}, \\lsegeven(E, F) &::= \exists X, Y.\ E \mapsto (next, X) * X \mapsto (next, Y) * lsegeven(Y, F).\end{aligned}$$

Then *lsegeven* still enjoys the composition lemma

$$\exists E_2.\ lsegeven(E_1, E_2) * lsegeven(E_2, E_3) \Rightarrow lsegeven(E_1, E_3).$$

On the other hand, the predicate *lsegodd* for list segments of the odd length does not enjoy the composition lemma. The definition of $lsegodd(E, F)$ can be obtained from that of $lsegeven(E, F)$ by replacing the base rule with the rule $lsegodd(E, F) ::= E \mapsto (next, F)$. This counterexample suggests that in order to guarantee the compositionality lemma, the syntactical compositionality should be carefully generalized as follows: The inductive rules may be generalized to contain several points-to atoms, but the base rule should not be changed.

Points-to atom in base rules.
Finally, we discuss the constraint that the base rule of a syntactically compositional predicate has an empty spatial atom. We use the aforementioned predicates *lsegeven* and *lsegodd* to illustrate the idea. The only difference between the inductive definition of *lsegeven* and and that of *lsegodd* is as follows: The base rule of *lsegodd* is $E \mapsto (next, F)$, while that of *lsegeven* is $E = F$. From this, we deduce that

$$lsegodd(E, F) \Leftrightarrow \exists X.\ E \mapsto \{(next, X)\} * lsegeven(X, F).$$

This idea can be generalized to arbitrary syntactically compositional predicates.

C Proofs in Section 5

Theorem 2. *Suppose that $\mathcal{P}$ is a set of inductively defined predicates. If $P \in \mathcal{P}$ is syntactically compositional, then P is semantically compositional.*

Proof. Suppose P is syntactically compositional and has parameters $(\vec{\alpha}, \vec{\beta}, \vec{\xi})$.
It is sufficient to prove the following claim.

For each pair (s, h), if $(s, h) \models P(\vec{\alpha}_1, \vec{\alpha}_2, \vec{\xi'}) * P(\vec{\alpha}_2, \vec{\alpha}_3, \vec{\xi'})$, then $(s, h) \models P(\vec{\alpha}_1, \vec{\alpha}_3, \vec{\xi'})$.

We prove the claim by induction on the size of $\mathsf{ldom}(h)$.
Suppose for each $i : 1 \leq i \leq 3$, $\vec{\alpha}_i = E_i v_i$.
Since $(s, h) \models P(\vec{\alpha}_1, \vec{\alpha}_2, \vec{\xi'}) * P(\vec{\alpha}_2, \vec{\alpha}_3, \vec{\xi'})$, there are h_1, h_2 such that $h = h_1 * h_2$, $(s, h_1) \models P(\vec{\alpha}_1, \vec{\alpha}_2, \vec{\xi'})$, and $(s, h_2) \models P(\vec{\alpha}_2, \vec{\alpha}_3, \vec{\xi'})$.

If $(s, h_1) \models \bigwedge\limits_{i=1}^{2} \alpha_{1,i} = \alpha_{2,i} \wedge \mathtt{emp}$, then $\mathsf{ldom}(h_1) = \emptyset$, and $h_2 = h$. From this, we deduce that $(s, h) \models P(\vec{\alpha}_1, \vec{\alpha}_3, \vec{\xi'})$.

Otherwise, there are a recursive rule of P, say $P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) ::= \exists \vec{X}. \mathit{\Pi} \wedge E \mapsto \rho * \mathit{\Sigma}_r * P(\vec{\gamma}, \vec{\beta}, \vec{\xi})$, and an extension of s, say s', such that $(s', h_1) \models \mathit{\Pi}' \wedge E_1 \mapsto \rho' * \mathit{\Sigma}'_r * P(\vec{\gamma'}, \vec{\alpha}_2, \vec{\xi'})$, where $\mathit{\Pi}', \rho', \mathit{\Sigma}'_r, \gamma'$ are obtained from $\mathit{\Pi}, \rho, \mathit{\Sigma}_r, \gamma$ by replacing $\vec{\alpha}, \vec{\beta}, \vec{\xi}$ with $\vec{\alpha_1}, \vec{\alpha}_2, \vec{\xi'}$ respectively. From this, we deduce that there are $h_{1,1}, h_{1,2}, h_{1,3}$ such that $h_1 = h_{1,1} * h_{1,2} * h_{1,3}$, $(s', h_{1,1}) \models E \mapsto \rho'$, $(s', h_{1,2}) \models \mathit{\Sigma}'_r$, and $(s', h_{1,3}) \models P(\vec{\gamma'}, \vec{\alpha}_2, \vec{\xi'})$. Then $(s', h_{1,3} * h_2) \models P(\vec{\gamma'}, \vec{\alpha}_2, \vec{\xi'}) * P(\vec{\alpha}_2, \vec{\alpha}_3, \vec{\xi'})$. From the induction hypothesis, we deduce that $(s', h_{1,3} * h_2) \models P(\vec{\gamma'}, \vec{\alpha}_3, \vec{\xi'})$. Then $(s', h_{1,1} * h_{1,2} * h_{1,3} * h_2) \models \Pi' \wedge E_1 \mapsto \rho' * \Sigma'_r * P(\vec{\gamma'}, \vec{\alpha}_3, \vec{\xi'})$. We then deduce that $(s, h) \models \exists \vec{X}. \Pi' \wedge E_1 \mapsto \rho' * \Sigma'_r * P(\vec{\gamma'}, \vec{\alpha}_3, \vec{\xi'})$.

To prove $(s, h) \models P(\vec{\alpha}_1, \vec{\alpha}_3, \vec{\xi'})$, it is sufficient to prove that $(s, h) \models \exists \vec{X}. \Pi'' \wedge E_1 \mapsto \rho'' * \Sigma''_r * P(\vec{\gamma''}, \vec{\alpha}_3, \vec{\xi'})$, where $\Pi'', \rho'', \Sigma''_r, \vec{\gamma''}$ are obtained from $\Pi, \rho, \Sigma_r, \vec{\gamma}$ by replacing $\vec{\alpha}, \vec{\beta}, \vec{\xi}$ with $\vec{\alpha}_1, \vec{\alpha}_3, \vec{\xi'}$ respectively.

From the fact that no variables from $\vec{\beta}$ occur in Π, ρ, Σ_r, or $\vec{\gamma}$, we know that $\Pi'' = \Pi'$, $\rho'' = \rho'$, $\Sigma''_r = \Sigma'_r$, and $\vec{\gamma''} = \vec{\gamma'}$. Since $(s, h) \models \exists \vec{X}. \Pi' \wedge E_1 \mapsto \rho' * \Sigma'_r * P(\vec{\gamma'}, \vec{\alpha}_3, \vec{\xi'})$, we have already proved that $(s, h) \models \exists \vec{X}. \Pi'' \wedge E_1 \mapsto \rho'' * \Sigma''_r * P(\vec{\gamma''}, \vec{\alpha}_3, \vec{\xi'})$. The proof is done. $\square$

D Proofs in Section 6

Theorem 3. Let $P \in \mathcal{P}$ be a syntactically compositional predicate with the parameters $(\vec{\alpha}, \vec{\beta}, \vec{\xi})$, and $P' \in \mathcal{P}$ with the parameters $(\vec{\alpha}, \vec{\xi})$. If P' is a completion of P with respect to $\vec{c}$, then $P'(\vec{\alpha}, \vec{\xi}) \Leftrightarrow P(\vec{\alpha}, \vec{c}, \vec{\xi})$ and $\exists \vec{\beta}.\ P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) * P'(\vec{\beta}, \vec{\xi}) \Rightarrow P'(\vec{\alpha}, \vec{\xi})$ hold.

Proof. The fact $P'(\vec{\alpha}, \vec{\xi}) \Leftrightarrow P(\vec{\alpha}, \vec{c}, \vec{\xi})$ can be proved easily by an induction on the size of the domain of the heap structures.

The argument for $\exists \vec{\beta}.\ P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) * P'(\vec{\beta}, \vec{\xi}) \Rightarrow P'(\vec{\alpha}, \vec{\xi})$ goes as follows: Suppose $(s, h) \models P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) * P'(\vec{\beta}, \vec{\xi})$. Then there are h_1, h_2 such that $h = h_1 * h_2$, $(s, h_1) \models P(\vec{\alpha}, \vec{\beta}, \vec{\xi})$, and $(s, h_2) \models P'(\vec{\beta}, \vec{\xi})$. From the fact that $P'(\vec{\beta}, \vec{\xi}) \Leftrightarrow P(\vec{\beta}, \vec{c}, \vec{\xi})$, we know that $(s, h_2) \models P(\vec{\beta}, \vec{c}, \vec{\xi})$. Therefore, $(s, h) \models P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) * P(\vec{\beta}, \vec{c}, \vec{\xi})$. From Theorem 2, we deduce that $(s, h) \models P(\vec{\alpha}, \vec{c}, \vec{\xi})$. From the fact $P(\vec{\alpha}, \vec{c}, \vec{\xi}) \Leftrightarrow P'(\vec{\alpha}, \vec{\xi})$, we conclude that $(s, h) \models P'(\vec{\alpha}, \vec{\xi})$. $\square$

Theorem 4. *Let $P, P' \in \mathcal{P}$ be two syntactically compositional inductively defined predicates with the same set of parameters $(\vec{\alpha}, \vec{\beta}, \vec{\xi})$. If P' is stronger than P, then the entailment $P'(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \Rightarrow P(\vec{\alpha}, \vec{\beta}, \vec{\xi})$ and $\exists \vec{\beta}.\ P'(\vec{\alpha}, \vec{\beta}, \vec{\xi}) * P(\vec{\beta}, \vec{\gamma}, \vec{\xi}) \Rightarrow P(\vec{\alpha}, \vec{\gamma}, \vec{\xi})$ hold.*

Proof. We first show that $P'(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \Rightarrow P(\vec{\alpha}, \vec{\beta}, \vec{\xi})$. By induction on the size of $\mathsf{ldom}(h)$, we prove the following fact: For each (s, h), if $(s, h) \models P'(\vec{\alpha}, \vec{\beta}, \vec{\xi})$, then $(s, h) \models P(\vec{\alpha}, \vec{\beta}, \vec{\xi})$.

Suppose $(s, h) \models P'(\vec{\alpha}, \vec{\beta}, \vec{\xi})$.

If $(s, h) \models \bigwedge\limits_{i=1}^{2} \alpha_i = \beta_i \wedge \mathtt{emp}$, since P' and P have the same base rule, we deduce that $(s, h) \models P(\vec{\alpha}, \vec{\beta}, \vec{\xi})$.

Otherwise, there are a recursive rule of P', say $P'(\vec{\alpha}, \vec{\beta}, \vec{\xi}) ::= \exists \vec{X}. \Pi' \wedge E \mapsto \rho * \Sigma_r * P'(\vec{\gamma}, \vec{\xi})$, and an extension of s, say s', such that $(s', h) \models \Pi' \wedge E \mapsto \rho * \Sigma_r * P'(\vec{\gamma}, \vec{\beta}, \vec{\xi})$. Then there are h_1, h_2, h_3 such that $h = h_1 * h_2 * h_3$, $(s', h_1) \models$

$E \mapsto \rho$, $(s', h_2) \models \Sigma_r$, and $(s', h_3) \models P'(\vec{\gamma}, \vec{\beta}, \vec{\xi})$. From the induction hypothesis, we deduce that $(s', h_3) \models P(\vec{\gamma}, \vec{\beta}, \vec{\xi})$. Moreover, from the assumption, we know that there is a recursive rule of P of the form $P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) ::= \exists \vec{X}.\Pi \wedge E \mapsto \rho * \Sigma_r * P(\vec{\gamma}, \vec{\xi})$, such that $\Pi' \Rightarrow \Pi$ holds. Then it follows that $(s', h_1 * h_2 * h_3) \models \Pi \wedge E \mapsto \rho * \Sigma_r * P(\vec{\gamma}, \vec{\beta}, \vec{\xi})$. We then deduce that $(s, h) \models \exists \vec{X}.\Pi \wedge E \mapsto \rho * \Sigma_r * P(\vec{\gamma}, \vec{\beta}, \vec{\xi})$. From this, we conclude that $(s, h) \models P(\vec{\alpha}, \vec{\beta}, \vec{\xi})$.

We then prove the second claim of the theorem.

From the argument above, we know that $P'(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \Rightarrow P(\vec{\alpha}, \vec{\beta}, \vec{\xi})$ holds. Then $P'(\vec{\alpha}, \vec{\beta}, \vec{\xi}) * P(\vec{\beta}, \vec{\gamma}, \vec{\xi}) \Rightarrow P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) * P(\vec{\beta}, \vec{\gamma}, \vec{\xi})$ holds. In addition, from Theorem 2, we know that $P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) * P(\vec{\beta}, \vec{\gamma}, \vec{\xi}) \Rightarrow P(\vec{\alpha}, \vec{\gamma}, \vec{\xi})$ holds. Therefore,we conclude that $P'(\vec{\alpha}, \vec{\beta}, \vec{\xi}) * P(\vec{\beta}, \vec{\gamma}, \vec{\xi}) \Rightarrow P(\vec{\alpha}, \vec{\gamma}, \vec{\xi})$. $\square$

Theorem 5. *Let $P \in \mathcal{P}$ be a syntactically compositional predicate with the parameters $(\vec{\alpha}, \vec{\beta}, \vec{\xi})$ and $P' \in \mathcal{P}$ be an inductive predicate with the parameters $(\vec{\alpha}, \vec{\beta}, \vec{\xi'})$. If P' is a static-parameter contraction of P with the contraction function prj, then $P'(\vec{\alpha}, \vec{\beta}, \vec{\xi'}) \Leftrightarrow P(\vec{\alpha}, \vec{\beta}, ext_{prj, \vec{\beta}}(\vec{\xi'}))$ and $\exists \vec{\beta}.\ P(\vec{\alpha}, \vec{\beta}, ext_{(prj, \vec{\gamma})}(\vec{\xi'})) * P'(\vec{\beta}, \vec{\gamma}, \vec{\xi'}) \Rightarrow P'(\vec{\alpha}, \vec{\gamma}, \vec{\xi'})$ hold.*

Proof. The first claim can be proved by induction on the size of the domain of the heap structures.

The argument for the second claim goes as follows: From the fact that $P'(\vec{\beta}, \vec{\gamma}, \vec{\xi'}) \Leftrightarrow P(\vec{\beta}, \vec{\gamma}, ext_{(prj, \vec{\gamma})}(\vec{\xi'}))$, we deduce that

$$\begin{array}{l} P(\vec{\alpha}, \vec{\beta}, ext_{(prj, \vec{\gamma})}(\vec{\xi'})) * P'(\vec{\beta}, \vec{\gamma}, \vec{\xi'}) \Rightarrow \\ \quad P(\vec{\alpha}, \vec{\beta}, ext_{(prj, \vec{\gamma})}(\vec{\xi'})) * P(\vec{\beta}, \vec{\gamma}, ext_{(prj, \vec{\gamma})}(\vec{\xi'})). \end{array}$$

From Theorem 2, we know that

$$P(\vec{\alpha}, \vec{\beta}, ext_{(prj, \vec{\gamma})}(\vec{\xi'})) * P(\vec{\beta}, \vec{\gamma}, ext_{(prj, \vec{\gamma})}(\vec{\xi'})) \Rightarrow P(\vec{\alpha}, \vec{\gamma}, ext_{(prj, \vec{\gamma})}(\vec{\xi'})).$$

Then the second claim follows from the fact that $P(\vec{\alpha}, \vec{\gamma}, ext_{(prj, \vec{\gamma})}(\vec{\xi'})) \Leftrightarrow P'(\vec{\alpha}, \vec{\gamma}, \vec{\xi'})$. $\square$