Automating Program Proofs Based on Separation Logic with Inductive Definitions

Constatin Enea\textsuperscript{1}, Mihaela Sighireanu\textsuperscript{1} and Zhilin Wu\textsuperscript{2,1}

\textsuperscript{1} LIAFA, Université Paris Diderot, France
\textsuperscript{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 trees, 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 $M$:

\[
\begin{align*}
\text{lsseg}(E, M, F) &::= E = F \land \text{emp} \land M = \emptyset \\
\text{lsseg}(E, M, F) &::= (\exists X, v, M_1. E \mapsto \{(\text{next}, X), (\text{data}, v)\} \ast \text{lsseg}(X, M_1, F) \\
&\land v \leq M_1 \land M = M_1 \cup \{v\}
\end{align*}
\]

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

\[
\exists E_2. \text{lsseg}(E_1, M_1, E_2) \ast \text{lsseg}(E_2, M_2, E_3) \land M_1 \leq M_2 \Rightarrow \exists M. \text{lsseg}(E_1, M, E_3).
\]

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 $\text{lsseg}$, with an additional parameter $M'$:

\[
\begin{align*}
\text{lsseg}(E, M, F, M') &::= E = F \land \text{emp} \land M = M' \\
\text{lsseg}(E, M, F, M') &::= (\exists X, v, M_1. E \mapsto \{(\text{next}, X), (\text{data}, v)\} \ast \text{lsseg}(X, M_1, F, M') \\
&\land v \leq M_1 \land M = M_1 \cup \{v\}
\end{align*}
\]

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:

\[
\exists E_2, M_2. \text{lsseg}(E_1, M_1, E_2, M_2) \ast \text{lsseg}(E_2, M_2, E_3, M_3) \Rightarrow \text{lsseg}(E_1, M_1, E_3, M_3).
\]
proof strategy using such lemmas, based on simple syntactic matchings of spa-
tial atoms (points-to atoms or predicate atoms like \texttt{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 bench-
marks, e.g., the verification conditions generated from the iterative procedures
for searching, inserting, or deleting elements in binary search trees, 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:

\[
\begin{align*}
\text{bst}(E, M) & := E = \text{nil} \land \text{emp} \land M = \emptyset \quad (4)
\text{bst}(E, M) & := \exists X, Y, M_1, M_2, v. E \Rightarrow \{(\text{left}, X), (\text{right}, Y), (\text{data}, v)\} \quad (5)
\end{align*}
\]

\[
\begin{align*}
\text{bst}(X, M_1) \land \text{bst}(Y, M_2) \land E \neq \text{nil} \land M = \{v\} \cup M_1 \cup M_2 \land M_1 < v < M_2
\end{align*}
\]

\[
\begin{align*}
\text{bsthole}(E, M_1, F, M_2) & := E = F \land \text{emp} \land M_1 = M_2, \quad (6)
\text{bsthole}(E, M_1, F, M_2) & := \exists X, Y, M_3, M_4, v. E \Rightarrow \{(\text{left}, X), (\text{right}, Y), (\text{data}, v)\} \quad (7)
\end{align*}
\]

\[
\begin{align*}
\text{bsthole}(X, M_3) \land \text{bsthole}(Y, F, M_4, M_2) \land M_1 = \{v\} \cup M_3 \cup M_4 \land M_1 < v < M_4
\end{align*}
\]

\[
\text{bsthole}(X, F, M_3, M_4) \land \text{bsthole}(Y, M_4) \land M_1 = \{v\} \cup M_3 \cup M_4 \land M_1 < v < M_4
\]

The predicate \( \text{bst}(E, M) \) 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 vari-
ablesses (e.g. \( E = \text{nil} \)) 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 val-
ues stored in the tree, and state the sortedness
property of BSTs.

The precondition of \text{search} is \( \text{bst}(\text{root}, M_0) \),
where \( M_0 \) is a ghost variable denoting the multi-
set of values stored in the tree, while its postcondition is \( \text{bst}(\text{root}, M_0) \land (\text{key} \in M_0 \rightarrow \text{ret} = 1) \land (\text{key} \notin M_0 \rightarrow \text{ret} = 0) \), where \( \text{ret} \) 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 sub-
heaps: the tree rooted at \( t \), and the truncated tree rooted at \text{root} which contains a “hole” \( t \).

To specify the invariant of this loop, we define another predicate
\( \text{bsthole}(E, M_1, F, M_2) \) describing “truncated” BSTs with one hole \( F \) as follows:

\[
\begin{align*}
\text{bsthole}(E, M_1, F, M_2) & := E = F \land \text{emp} \land M_1 = M_2, \quad (6)
\text{bsthole}(E, M_1, F, M_2) & := \exists X, Y, M_3, M_4, v. E \Rightarrow \{(\text{left}, X), (\text{right}, Y), (\text{data}, v)\} \quad (7)
\end{align*}
\]

\[
\begin{align*}
\text{bsthole}(X, M_3) \land \text{bsthole}(Y, F, M_4, M_2) \land M_1 = \{v\} \cup M_3 \cup M_4 \land M_1 < v < M_4
\end{align*}
\]

\[
\text{bsthole}(X, F, M_3, M_4) \land \text{bsthole}(Y, M_4) \land M_1 = \{v\} \cup M_3 \cup M_4 \land M_1 < v < M_4
\]
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:

$$Inv := \exists M_1. \; bst\text{hole}(\text{root}, M_0, t, M_1) \land (\text{key} \in M_0 \leftrightarrow \text{key} \in M_1). \tag{9}$$

We illustrate that such inductive definitions are appropriate for automated reasoning, by taking the following branch of the loop: \textit{assume}($t != \text{NULL}$); \textit{assume}($t->\text{data} > \text{key}$); $t' = t->\text{left}$ (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:

$$\exists M_1, Y, v, M_2, M_3. \; bst\text{hole}(\text{root}, M_0, t, M_1) \land t \mapsto \{(\text{left}, t'), (\text{right}, Y), (\text{data}, v)\} \land bst(t', M_2) \land bst(Y, M_3) \land M_1 = \{v\} \cup M_2 \cup M_3 \land v < M_3 \land (\text{key} \in M_0 \leftrightarrow \text{key} \in M_1) \land v > \text{key}. \tag{10}$$

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 $\varphi_1 \Rightarrow \exists X. \varphi_2$, where $X$ contains only data variables\(^3\). The strategy is based on two steps: (i) enumerating spatial atoms $A$ from $\varphi_2$, and for each of them, carving out a sub-formula $\varphi_A$ of $\varphi_1$ that entails $A$, and (ii) proving that the data constraints from $\varphi_A$ imply those from $\varphi_2$ (this can be done using SMT solvers). The step (i) may generate constraints on the existentially-quantified variables $X$ in $\varphi_2$ that are used in the second step. If the step (ii) succeeds, and every spatial atom of $\varphi_1$ 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 $\exists M_1'. \; bst\text{hole}(\text{root}, M_0, t', M'_1)$ and respectively, $\exists M_2'. \; bst(t', M'_2)$. This renaming of existential variables requires adding the equality $M_1 = M'_1 = M''_1$ to $Inv'$. The second goal, for $\exists M_1'. \; bst(t', M'_2)$, can be solved easily since this predicate almost matches the sub-formula $bst(t', M_2)$. 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 $\exists M_1'. \; bst\text{hole}(\text{root}, M_0, t', M'_1)$ requires a non-trivial lemma. Thus, according to the syntactic criteria defined in Sec. 5, the predicate $bst\text{hole}$ enjoys the following composition lemma:

$$\left( \exists F. \; bst\text{hole}(\text{root}, M_0, F, M) \land bst\text{hole}(F, M, t', M'_1) \right) \Rightarrow bst\text{hole}(\text{root}, M_0, t', M'_1). \tag{11}$$

\(^3\) The existential quantifiers in $\varphi_1$ can be removed using skolemization.
Intuitively, this lemma states that composing two heap structures described by \texttt{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 \texttt{bsthole}(\texttt{root}, \texttt{M}_0, \texttt{F}, \texttt{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 \texttt{bsthole}, and \( F \) (resp. \( M \)) occurs only in an equality with \texttt{root} (resp. \texttt{M}_0) in the base rule (we are referring to the rules (6)–(8) with the parameters of \texttt{bsthole} substituted by (\texttt{root}, \texttt{M}_0, \texttt{F}, \texttt{M})).

Therefore, the first goal reduces to finding a sub-formula of \( \texttt{post}(\texttt{Inv}) \) that implies the premise of (11) where \( \texttt{M}'_1 \) 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., \texttt{root} in lemma (11). Therefore, in the first sub-goal \( \exists F, M. \texttt{bsthole}(\texttt{root}, \texttt{M}_0, \texttt{F}, \texttt{M}) \), matches the atom \texttt{bsthole}(\texttt{root}, \texttt{M}_0, \texttt{t}, \texttt{M}_1) under the constraint \( F = \texttt{t} \land M = M_1 \). This constraint is used in solving the second sub-goal, which now becomes \( \exists \texttt{M}'_1. \texttt{bsthole}(\texttt{t}, \texttt{M}_1, \texttt{t}', \texttt{M}'_1) \).

The second sub-goal can be proved by unfolding \texttt{bsthole} twice, using first the rule (8) and then the rule (6), and by matching the resulting spatial atoms with those in \( \texttt{post}(\texttt{Inv}) \) one by one. This completes the proof of \( \texttt{post}(\texttt{Inv}) \Rightarrow \texttt{Inv}' \) since the sub-formulas generated for the two initial goals are disjoint and they cover all the spatial atoms of \( \texttt{post}(\texttt{Inv}) \). Also, assuming that the existential \( \texttt{M}_1 \) from \( \texttt{Inv}' \) is instantiated with \( \texttt{M}_2 \) from \( \texttt{post}(\texttt{Inv}) \) (fact automatically deduced in the first step), the data constraints in \( \texttt{post}(\texttt{Inv}) \) entail those in \( \texttt{Inv}' \).

3 Separation Logic with Inductive Definitions

Let \( \texttt{LVar} \) be a set of location variables, interpreted as heap locations, and \( \texttt{DVar} \) a set of data variables, interpreted as data values stored in the heap, (multi)sets of values, etc. In addition, let \( \texttt{Var} = \texttt{LVar} \cup \texttt{DVar} \). The domain of heap locations is denoted by \( \texttt{L} \) while the domain of the variables in \( \texttt{DVar} \) is generically denoted by \( \texttt{D} \). Let \( \mathcal{F} \) be a set of pointer fields, interpreted as functions \( \texttt{L} \rightarrow \texttt{L} \), and \( \mathcal{D} \) a set of data fields, interpreted as functions \( \texttt{L} \rightarrow \texttt{D} \). 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 \((\ell, \text{pf})\), where \( \ell \) is a location and \( \text{pf} \) is a pointer field, to locations, and a finite set of pairs \((\ell, \text{df})\), where \( \text{df} \) is a data field, to values in \( \texttt{D} \). In addition, \( h \) satisfies that for each \( \ell \in \texttt{L} \), if \((\ell, \text{df}) \in \text{dom}(h)\) for some \( d \in \mathcal{D} \), then \((\ell, \text{pf}) \in \text{dom}(h)\) for some \( \text{pf} \in \mathcal{F} \). Let \( \text{dom}(h) \) denote the domain of \( h \), and \( \text{ldom}(h) \) denote the set of \( \ell \in \texttt{L} \) such that \((\ell, \text{pf}) \in \text{dom}(h)\) for some \( \text{pf} \in \mathcal{F} \).
Table 1. The syntax of the Separation Logic fragment

\[
X, Y, E \in \text{LVar} \text{ location variables } \rho \subseteq (F \times \text{LVar}) \cup (D \times \text{DVar}) \\
\vec{F} \in \text{Var}^* \text{ vector of variables } \ P \in \mathcal{P} \text{ predicates} \\
x \in \text{Var} \text{ variable } \Delta \text{ formula over data variables} \\
\Pi ::= X = Y \mid X \neq Y \mid \Delta \mid \Pi \land \Pi \text{ pure formulas} \\
\Sigma ::= \text{emp} \mid E \mapsto \rho \mid P(E, \vec{F}) \mid \Sigma \ast \Sigma \text{ spatial formulas} \\
\varphi ::= \Pi \land \Sigma \mid \varphi \lor \varphi \mid \exists x. \varphi \text{ formulas}
\]

Formulas are conjunctions between a pure formula \(\Pi\) and a spatial formula \(\Sigma\). 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 \(\Delta\) over data variables. We let \(\Delta\) 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 \(E \mapsto \rho\) specifies that the heap contains exactly one location \(E\), and for all \(i \in I\), 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 

\[P(E, \vec{F}) ::= \exists \vec{Z}.\Pi \land \Sigma\]

where \(\vec{Z} \in \text{Var}^*\) is a tuple of variables. A rule \(R\) is called a base rule if \(\Sigma\) contains no predicate atoms. Otherwise, it is called an inductive rule. A base rule \(R\) is called a spatial-empty base rule if \(\Sigma = \text{emp}\). 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 \(P(E, \vec{F}) ::= \exists \vec{Z}.\Pi \land \Sigma\),

- If the rule is inductive, then
  - **One points-to atom**: \(\Sigma\) contains exactly one points-to atom \(E \mapsto \rho\), for some \(\rho\). In addition, for each field \(f \in F\) (resp. \(d \in D\)), \(\rho\) contains at most one occurrence of \(f\) (resp. \(d\)).
  - **Connectedness**: For each predicate atom \(Q(E_1, \vec{F}_1)\) in \(\Sigma\), there is \(Z \in \text{LVar}\) such that \(\Pi \models E_1 = Z\) and \(Z\) occurs in \(\rho\).
- If the rule is a spatial-nonempty base rule, then \(\Sigma\) contains exactly one points-to atom \(E \mapsto \rho\), for some \(\rho\).

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 \(\psi_1\) **entails** another formula \(\psi_2\), denoted by \(\psi_1 \Rightarrow \psi_2\), iff every model of \(\psi_1\) is also a model of \(\psi_2\). 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 \( \varphi_1 \Rightarrow \exists X. \varphi_2 \), where \( \varphi_1, \varphi_2 \) are quantifier-free, and \( X \in \text{DVar}^* \). We consider quantifier-free left-hand sides \( \varphi_1 \) 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 \( X \) occurs in at most one spatial atom of \( \varphi_2 \) (multiple occurrences of the same variable can be removed by introducing fresh variables and new equalities in the pure part). Also, we assume that \( \varphi_1 \) and \( \varphi_2 \) are of the form \( \Pi \land \Sigma \). In the general case, our proof strategy checks that for every disjunct \( \varphi'_1 \) of \( \varphi_1 \), there is a disjunct \( \varphi'_2 \) of \( \varphi_2 \) s.t. \( \varphi'_1 \Rightarrow \exists X. \varphi'_2 \).

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

When \( \varphi_2 \) contains at least two spatial atoms, \( \text{slice} \) is called recursively (line 3) on each spatial atom from \( \varphi_2 \), preserving the existential quantifiers over \( X \).

Then, it checks (line 7) that the pure part of \( \varphi_1 \), together with the rely constraints obtained from the recursive calls, implies the pure part of \( \varphi_2 \) 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 \( \varphi_1 \) occurs in at most one sub-formula returned by the recursive calls. If both tests succeed, \( \text{slice} \) returns the pure part of \( \varphi_1 \) 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 \( \varphi_2 \) contains only one spatial atom, \( \text{slice} \) checks whether \( \varphi_1 \) contains a spatial atom that matches the spatial atom in \( \varphi_2 \) (line 10), using the function \( \text{matchAtom} \). If \( \varphi_2 \) is an atom \( \exists X. E \Rightarrow \rho \), then \( \text{matchAtom} \) searches for an atom \( E' \Rightarrow \rho' \) of \( \varphi_1 \) that (1) has the same root, i.e., \( \text{Pure}(\varphi_1) \models E = E' \), (2) for every pointer field \( f \), if \( (f, X) \in \rho \), then there is \( (f, Y) \in \rho' \), and vice versa. \( \text{matchAtom} \) returns such an atom, if it exists, or an error value \( \bot \) otherwise. Moreover, in the positive case, \( \text{matchAtom} \) computes a rely constraint \( C_r \) and a guarantee constraint \( C_g \) (which are sets of equality constraints between data terms) such that \((\text{Pure}(\varphi_1) \land E' \Rightarrow \rho' \land C_r) \Rightarrow E \Rightarrow \rho \land C_g \). For example, consider the call of \( \text{matchAtom} \) with the following formulas:

\[
\varphi_1 ::= X = Y \land w = w' \land E \Rightarrow \{(f, Y), (d_1, v), (d_2, w)\}
\]

\[
\varphi_2 ::= \exists \nu. E \Rightarrow \{(f, X), (d_1, v'), (d_2, w')\},
\]

where \( d_1 \) and \( d_2 \) are data fields. \( \text{matchAtom} \) returns the points-to atom from \( \varphi_1 \), \( C_r : v = v' \), and \( C_g : w = w' \). 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.
The predicate atom from $C$ from cation variables. Finding suitable instantiations for these variables relies on a entailment of proof goal, i.e., proving entailment of $\exists P$. Lemmas correspond to the inductive rules defining the predicate $\phi$. If $\phi$ contains a predicate atom, say $\phi \land \vec{Z}. \Pi$, then $\phi$ is a predicate atom, say $P(E, \vec{F})$ then

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

$$\text{deduced from the matching to } C_g, \text{ if it involves only the free variables of } \varphi_2, \text{ and to } C_r, \text{ otherwise.}$$

The output of $\text{matchAtom}$ for predicate atoms is computed in a similar way: The predicate atom from $\varphi_2$, say $P(E', \vec{F}')$, is matched to some atom $P(E, \vec{F})$ from $\varphi_1$ such that $\text{Pure}(\varphi_1) \models E = E'$. For instance, $\text{matchAtom}$ called with $\varphi_1 ::= X = Y \wedge w = w' \wedge P(E, Y, v, w)$ and $\varphi_2 ::= \exists v'. P(E, X, v', w')$, returns the spatial atom of $\varphi_1$ and the same rely and guarantee as above.

If $\varphi_2$ contains a predicate atom, say $P(E, \vec{F})$, which doesn’t match an atom of $\varphi_1$ (line 14), slice proceeds by applying lemmas (line 15). Straightforward lemmas correspond to the inductive rules defining the predicate $P$: every rule $P(E, \vec{F}) ::= \exists Z. \Pi \land \Sigma$ defines a lemma $\exists \vec{Z}. \Pi \land \Sigma \Rightarrow P(E, \vec{F})$. More complex lemmas are defined in Sec. 5 and 6.

Essentially, applying a lemma $L \equiv \varphi_L \Rightarrow P(E, \vec{F})$ means that the initial proof goal, i.e., proving entailment of $\exists X. P(E, \vec{F})$, is reduced to proving the entailment of $\exists X. \varphi_L$. The formula $\varphi_L$ may contain existentially-quantified location variables. Finding suitable instantiations for these variables relies on a
natural assumption that \( \varphi_L \) contains a unique spatial atom, denoted by \( \text{root}(L) \), rooted at \( E \) (either a points-to atom \( E \mapsto \rho \) or a predicate atom \( Q(E_1, \ldots) \)), 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 \( \text{root}(L) \) is matched with an atom rooted at \( E \) from \( \varphi_1 \) using the function \( \text{matchAtom} \) (line 16). When this matching is possible, \( \text{matchAtom} \) returns a spatial atom \( A_1 \) of \( \varphi_1 \) rooted at some \( E' \), with \( E' = E \) implied by \( \text{Pure}(\varphi_1) \), together with rely and guarantee constraints.

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

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

\[
\varphi_1 := x_1 \neq \text{nil} \land x_2 \neq \text{nil} \land v_1 < v_2 \land x_1 \mapsto \{(\text{next}, x_2), (\text{data}, v_1)\}
\]

\[
\varphi_2 := \exists M. \text{bseg}(x_1, M, \text{nil}, \emptyset) \land v_2 \in M,
\]

where \( \text{bseg} \) has been defined in Sec. 1 (eq. (1)–(2)). The first lemma to be applied corresponds to the inductive rule of \( \text{bseg} \), i.e., eq. (2) (page II):

\[
\exists X, M_1, v. \ x_1 \mapsto \{(\text{next}, X), (\text{data}, v)\} \land \text{bseg}(X, M_1, \text{nil}, \emptyset) \land M = \{v\} \cup M_1 \land v \leq M_1 \Rightarrow \text{bseg}(x_1, M, \text{nil}, \emptyset). \]

Therefore, the input of \( \text{matchAtom} \) is \( \varphi_1 \) and the atom \( \exists X, v. x_1 \mapsto \{(\text{next}, X), (\text{data}, v)\} \) from the lemma. The output of \( \text{matchAtom} \) is the points-to atom of \( \varphi_1 \) rooted at \( x_1 \), the rely \( C_r : X = x_2 \land v = v_1 \), and \( C_g \) is empty (true). The rely \( C_r \) 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 \( \text{slice} \) is called at line 19 with inputs

\[
\varphi_1' := x_1 \neq \text{nil} \land x_2 \neq \text{nil} \land v_1 < v_2 \land x_2 \mapsto \{(\text{next}, \text{nil}), (\text{data}, v_2)\}
\]

\[
\varphi_2 := \exists M_1. \text{bseg}(x_2, M_1, \text{nil}, \emptyset) \land v_1 \leq M_1
\]

This recursive call returns the whole formula \( \varphi_1' \) together with the rely \( C_r' : M_1 = \{v_2\} \), and an empty guarantee. Note that the conjunction of \( C_r, C_r' \), and the pure part of the lemma, in particular, the constraint \( M = \{v\} \cup M_1 \), implies the constraint \( v_2 \in M \) from \( \varphi_2 \). 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 \( \text{slice} \). Moreover, since we assume a finite set of lemmas, and every application of a lemma \( L \) removes one
spatial atom from $\varphi_1$ (the atom matched to root($L$)), the termination of slice is guaranteed. In general, slice is incomplete.

**Theorem 1.** Let $\varphi_1$ and $\exists \vec{X}. \varphi_2$ be two formulas. If $\text{slice}(\varphi_1, \exists \vec{X}. \varphi_2) = (\varphi_1, C_r, C_g)$, then $\varphi_1 \Rightarrow \exists \vec{X}. \varphi_2$.

### 5 Composition Lemmas

As we have seen in the motivating example, the predicate $\text{bsthole}(E, M_1, F, M_2)$ 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 $\vec{\xi} \in \text{Var}^*$, where $E,F \in \text{LVar}$ are called the cumulative and resp., the hole location parameter, and $C,H \in \text{DVar}$ are called the source and resp., the hole data parameter.

Let $P$ 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:** $P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) := \bigwedge_{i=1}^2 \alpha_i = \beta_i \land \text{emp}$.
- **Inductive rule:** $P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) := \exists \vec{Z}. \Pi \land \Sigma$, with (a) $\Sigma \triangleq E \mapsto \rho \ast \Sigma_r \ast P(\vec{\gamma}, \vec{\beta}, \vec{\xi})$, (b) $\Sigma_r$ contains only predicate atoms, (c) $\vec{\gamma} \subseteq \vec{Z}$, and (d) the variables in $\vec{\beta}$ don’t occur elsewhere in $\Pi \land \Sigma$.

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

- 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:

\[ \exists F, M_2. \text{bsthole}(E, M_1, F, M_2) \ast \text{bst}(F, M_2) \Rightarrow \text{bst}(E, M_1). \]

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 \((\text{nil}, \emptyset)\), 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 \(\beta_i\) with \(c_i\) for each \(i : 1 \leq i \leq 2\),
- for each inductive rule of \(P'\), say \(P'(\vec{\alpha}, \vec{\xi}) := \exists Z'. \Pi' \land \Sigma'\), there exists a rule of \(P\) of the form \(P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) := \exists Z. \Pi \land E \mapsto \rho \ast \Sigma \ast P(\vec{\gamma}, \vec{\beta}, \vec{\xi})\), s.t. \(|Z'| = |Z|\) and \(\Pi' \land \Sigma'\) is \((\Pi \land E \mapsto \rho \ast \Sigma_r \ast P'(\vec{\gamma}, \vec{\xi}))[\vec{c}/\vec{\beta}, Z'/Z]\),
- for each inductive rule of \(P\), say \(P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) := \exists \tilde{Z}. \Pi \land E \mapsto \rho \ast \Sigma \ast P(\vec{\gamma}, \vec{\beta}, \vec{\xi})\), there is an inductive rule of \(P'\) of the form \(P'(\vec{\alpha}, \vec{\xi}) := \exists \tilde{Z}'. \Pi' \land \Sigma'\), satisfying the same conditions as above.

Note that in the above definition, the occurrences of \(P\) in \(\Sigma_r\) (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 \(\exists \vec{\beta}. P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \ast P'(\vec{\beta}, \vec{\xi}) \Rightarrow P'(\vec{\alpha}, \vec{\xi})\) hold.

6.2 The “stronger” lemmas

We illustrate this class of lemmas on the example of BST. Let natbsth be the predicate defined by the same rules as bsthole (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:

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

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 $P'(\vec{\alpha}, \vec{\beta}, \vec{\xi}) := \exists Z. R' \land E \Rightarrow \rho * \Sigma_r * P'(\vec{\gamma}, \vec{\beta}, \vec{\xi})$, there is an inductive rule $P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) := \exists Z. R \land E \Rightarrow \rho * \Sigma_r * P(\vec{\gamma}, \vec{\beta}, \vec{\xi})$ such that $R' \Rightarrow R$ 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 $E \mapsto \{(\text{left}, X), (\text{right}, Y), (\text{tail}, F), (\text{data}, v)\}$. 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 $E \mapsto \{(\text{left}, X), (\text{right}, Y), (\text{tail}, B), (\text{data}, v)\}$, 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

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

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 $(\vec{\alpha}, \vec{\beta}, \vec{\xi}')$, $\vec{\xi} = \xi_1 \ldots \xi_k$, and $\vec{\xi}' = \xi'_1 \ldots \xi'_k$. 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 $\xi_i$ with $\beta_j$ for some $\xi_i \in \vec{\xi}$ and $\beta_j \in \vec{\beta}$ such that $\xi_i$ and $\beta_j$ have the same data type. More precisely, the base rules of $P'$ are those of $P$, in addition, there is a function $prj : \{1, \ldots, k\} \rightarrow \{0, 1, 2\}$ such that the following conditions hold.
– \(|prj\mathbf{^{-1}}(0)| = l\) and \(\vec{\xi}' = prj_0(\vec{\xi})\), where \(prj_0(\vec{\xi})\) is the tuple obtained from \(\vec{\xi}'\) by keeping the elements \(\xi_i\) such that \(prj(i) = 0\) and removing all the others.
– For each \(i: 1 \leq i \leq k\) s.t. \(prj(i) \neq 0\), \(\xi_i\) and \(\beta_{prj(i)}\) have the same data type.
– For each inductive rule of \(P'\), say \(P'(\vec{\alpha}, \vec{\beta}, \vec{\xi}') \iff \exists \vec{Z}. \Pi' \land \Sigma'\), there is an inductive rule of \(P\) of the form \(P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \iff \exists \vec{Z}. \Pi \land \rho \star \Sigma \star P(\vec{\gamma}, \vec{\beta}, \vec{\xi})\), s.t. \(\Pi' \land \Sigma'\) is obtained from \(\Pi \land \rho \star \Sigma \star P(\vec{\gamma}, \vec{\beta}, \vec{\xi})\) by first replacing \(P(\vec{\gamma}, \vec{\beta}, \vec{\xi})\) with \(P'(\vec{\gamma}, \vec{\beta}, \vec{\xi}')\), then replacing \(\xi_i\) 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 \(P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \iff \exists \vec{Z}. \Pi \land E \rightarrow \rho \star \Sigma \star P(\vec{\gamma}, \vec{\beta}, \vec{\xi})\), there is an inductive rule of \(P'\) of the form \(P'(\vec{\alpha}, \vec{\beta}, \vec{\xi}') \iff \exists \vec{Z}. \Pi' \land \Sigma'\), satisfying the same conditions as above.

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

Suppose \(\vec{\beta} \in \text{LVar} \times \text{DVar}\), \(\vec{\xi}' = \xi'_1, \ldots, \xi'_l\), \(prj: \{1, \ldots, k\} \rightarrow \{0, 1, 2\}\) such that \(|prj^{-1}(0)| = l\), then the \((prj, \vec{\beta})\)-extension of \(\vec{\xi}'\), denoted by \(\text{ext}_{(prj, \vec{\beta})}(\vec{\xi}')\), is defined as \(\vec{\xi} = \xi'_1, \ldots, \xi'_k\) such that \(prj(0) = \vec{\xi}\) 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 \(P'(\vec{\alpha}, \vec{\beta}, \vec{\xi}') \Rightarrow P(\vec{\alpha}, \vec{\beta}, \text{ext}_{prj, \vec{\beta}}(\vec{\xi}'))\) and \(\exists \vec{\beta}. P(\vec{\alpha}, \vec{\beta}, \text{ext}_{prj, \vec{\beta}}(\vec{\xi}')) \star P'(\vec{\alpha}, \vec{\beta}, \vec{\xi}') \Rightarrow P'(\vec{\alpha}, \vec{\beta}, \vec{\xi}')\) hold.

### 7 Experimental results

We have extended our tool \textsc{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 \textsc{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.

Table 2: Experimental results on benchmark RDBI

<table>
<thead>
<tr>
<th>Data structure</th>
<th>Procedure</th>
<th>#VC</th>
<th>Lemma (#b, #r, #p, #c, #d)</th>
<th>=&gt;D</th>
<th>Time (s)</th>
<th>spen</th>
<th>Z3</th>
</tr>
</thead>
<tbody>
<tr>
<td>sorted lists</td>
<td>search</td>
<td>4</td>
<td>(6, 8, 17, 3, 1)</td>
<td>6</td>
<td>0.10</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td></td>
<td>insert</td>
<td>8</td>
<td>(12, 18, 30, 12, 0)</td>
<td>20</td>
<td>0.33</td>
<td>0.10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>delete</td>
<td>4</td>
<td>(6, 10, 16, 6, 1)</td>
<td>10</td>
<td>0.15</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>BST</td>
<td>search</td>
<td>4</td>
<td>(9, 11, 27, 9, 1)</td>
<td>6</td>
<td>0.20</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td></td>
<td>insert</td>
<td>14</td>
<td>(18, 21, 33, 14, 0)</td>
<td>24</td>
<td>0.63</td>
<td>0.20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>delete</td>
<td>25</td>
<td>(30, 37, 106, 25, 0)</td>
<td>68</td>
<td>1.49</td>
<td>0.51</td>
<td></td>
</tr>
<tr>
<td>AVL</td>
<td>search</td>
<td>4</td>
<td>(10, 12, 17, 13, 1)</td>
<td>6</td>
<td>0.23</td>
<td>0.15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>insert</td>
<td>22</td>
<td>(20, 27, 66, 23, 0)</td>
<td>48</td>
<td>1.94</td>
<td>0.63</td>
<td></td>
</tr>
<tr>
<td>RBT</td>
<td>search</td>
<td>4</td>
<td>(9, 12, 27, 12, 1)</td>
<td>6</td>
<td>0.23</td>
<td>0.15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>insert</td>
<td>21</td>
<td>(32, 25, 126, 22, 0)</td>
<td>78</td>
<td>2.94</td>
<td>0.93</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Data structure</th>
<th>#VC</th>
<th>Lemma (#b, #r, #p, #c, #d)</th>
<th>Time-spent(s) spen-comp</th>
<th>spen-TA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nested linked lists</td>
<td>16</td>
<td>(17, 47, 14, 8, 0)</td>
<td>0.81</td>
<td>0.65</td>
</tr>
<tr>
<td>Skip lists 2 levels</td>
<td>4</td>
<td>(5, 11, 1, 1, 0)</td>
<td>0.18</td>
<td>0.18</td>
</tr>
<tr>
<td>Skip lists 3 levels</td>
<td>10</td>
<td>(16, 32, 29, 17, 0)</td>
<td>0.47</td>
<td>0.42</td>
</tr>
</tbody>
</table>

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, \(\Rightarrow_D\) provides the number of entailments between pure constraints generated by spen; Time-spent 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 Duo 2.53 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 terminol-
ogy) 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 ver-
ify 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 ex-
tended to cover such inductive definitions.

9 Conclusion
We proposed a novel approach for automating program proofs based on Sepa-
ration Logic with inductive definitions. This approach consists of (1) efficiently
checkable criteria for recognizing inductive definitions that satisfy crucial lem-
mas 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 ex-
tension 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 defini-
tions. 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

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:

\[
\varphi_1 := x_1 \neq \text{nil} \land x_2 \neq \text{nil} \land v_1 < v_2 \land x_1 \mapsto \{(\text{next}, x_2), (\text{data}, v_1)\}
\]

* \(x_2 \mapsto \{(\text{next}, \text{nil}), (\text{data}, v_2)\}\)

\[
\varphi_2 := \exists M, M'. \text{lseg}(x_1, M, \text{nil}, \emptyset) \land v_2 \in M,
\]

where \(\text{lseg}\) has been defined in Sec. 1 (eq. (1)–(2)).

The first recursive call of slice: Because \(\varphi_2\) has only one spatial atom \((\text{lseg}(x_1, M, \text{nil}, \emptyset))\) which fails to be matched to an atom of \(\varphi_1\) by \text{matchAtom} at line 10, the lemmas of \(\text{lseg}\) are tried at line 15. The only lemma including a points-to atom from \(x_1\) is the inductive rule of \(\text{lseg}\), i.e., eq. (2) (page II), where the pure part of the lemma \(\Pi^0 : v^1 \subseteq M_1 \land M = M_1 \cup \{v^1\}\).

Therefore, \text{matchAtom} is called at line 16 with the input \(\varphi_1\) and \(\exists X^1, v^1. x_1 \mapsto \{(\text{next}, X^1), (\text{data}, v^1)\}\) (\(\alpha\)-conversion is applied to existential variables). The output of \text{matchAtom} is the points-to atom of \(\varphi_1\) rooted at \(x_1\), the rely \(C^1_{\text{r}} : X^1 = x_2 \land v^1 = v_1\) and the guarantee \(C^1_{\text{g}}\) is empty (true). The rely \(C^1_{\text{r}}\) is used to eliminate the quantifiers over \(X^1\) and \(v^1\); in addition, quantifier over \(M\) are also eliminated to satisfy the constraint \(Z^1 \subseteq \bar{Z}\).

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

\[
\varphi_1 := x_1 \neq \text{nil} \land x_2 \neq \text{nil} \land v_1 < v_2 \land x_1 \mapsto \{(\text{next}, \text{nil}), (\text{data}, v_2)\}
\]

\[
\varphi_2 := \exists M_1. \text{lseg}(x_2, M_1, \text{nil}, \emptyset) \land v_1 \leq M_1^1.
\]

By applying again lemma (2) to the atom \(\text{lseg}(x_2, M_1^1, \text{nil}, \emptyset)\), we have that \(\Pi^1 : v^2 \subseteq M_1^2 \land M_1^1 = M_1^2 \cup \{v^2\}\). The \text{matchAtom} is called at line 16 with \(\varphi_1\) and \(\exists X^2, v^2. x_2 \mapsto \{(\text{next}, X^2), (\text{data}, v^2)\}\). The output of \text{matchAtom} is the points-to atom of \(\varphi_1\) rooted at \(x_2\), the rely \(C^2_{\text{r}} : X^2 = \text{nil} \land v^2 = v_2\) and the empty guarantee \(C^2_{\text{g}}\). The quantifiers elimination at line 18 returns the simplified formula wrt \(C^2_{\text{r}}\):

\[
\exists M_1^2. \text{lseg}(\text{nil}, M_1^2, \text{nil}, \emptyset) \land v_2 \leq M_1^2.
\]

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

\[
\varphi_1 := x_1 \neq \text{nil} \land x_2 \neq \text{nil} \land v_1 < v_2 \land \text{emp}
\]

\[
\varphi_2 := \exists M_2^1. \text{lseg}(\text{nil}, M_2^1, \text{nil}, \emptyset) \land v_2 \leq M_2^1.
\]

The third lemma applied is the base rule (1) of \(\text{lseg}\), which has \(\Pi^2 : M_2^1 = \emptyset\) and \text{matchAtom} generates empty (true) rely \(C^3_{\text{r}}\) and guarantee \(C^3_{\text{g}}\).

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

\[
\varphi_1 := x_1 \neq \text{nil} \land x_2 \neq \text{nil} \land v_1 < v_2 \land \text{emp}
\]

\[
\varphi_2 := \text{true} \land \text{emp}
\]
By matching the atom emp at line 10, it generates empty (true) $C^4_3$ and $C^4_2$. Because $\text{Pure}(\varphi_1^2) \models \text{Pure}(\varphi_2^2)$ is valid, the fourth recursive call returns at line 12, to the line 20 in the 3rd recursive call with the triple $(\text{emp}, C^4_1 : \text{true}, C^4_3 : \text{true})$

The constraint on data tested at line 20 in the 3rd recursive call is $\text{Pure}(\varphi_1^1) \land C^1_4 \land C^1_2 \land \Pi^1 \models \text{Pure}(\varphi_2^2)$, i.e., $x_1 \neq \text{nil} \land x_2 \neq \text{nil} \land v_1 < v_2 \land M^1_2 = \emptyset \Rightarrow v_2 \leq M^1_2$, which is valid. Notice that $v_2 \leq 0$ is trivially true. Then, the 3rd recursive call returns the triple $(\text{emp}, C^4_1 : M^1_2 = \emptyset, C^4_2 : \text{true})$ at line 20 of the second recursive call.

The constraint on data tested at line 20 in the 2nd recursive call is $\text{Pure}(\varphi_1^1) \land C^1_4 \land C^1_2 \land \Pi^1 \models \text{Pure}(\varphi_2^1)$, i.e., $\text{Pure}(\varphi_1^1) \land M^1_2 = \emptyset \land X^2 = \text{nil} \land v^2 = v_2 \land v^1 \leq M^1_2 \land M = M^1_2 \cup \{v^1\} = v_2 \in M)$, which is also valid. Then, the 2nd recursive call returns the triple $(x_2 \mapsto \{(\text{next}, \text{nil}), (\text{data}, v_2)\}, C^1_4 : C^1_3 \land C^1_1 \land M^1_1 = \{v^2\} \cup M^1_2 \land v^2 \leq M^1_2, C^1_2 : \text{true})$ at line 20 of the first recursive call.

The constraint on data tested at line 20 in the 1st recursive call is $\text{Pure}(\varphi_1^1) \land C^1_4 \land C^1_2 \land \Pi^1 \models \text{Pure}(\varphi_2^1)$, i.e., $\text{Pure}(\varphi_1^1) \land C^1_3 \land X^1 = x_2 \land V^1 = v_2 \land v^1 \leq M^1_2 \land M = M^1_2 \cup \{v^1\} = v_2 \in M$ which is valid. The first call returns the triple $(x_1 \mapsto \{(\text{next}, X^1), (\text{data}, v^1) \land x_2 \mapsto \{(\text{next}, \text{nil}), (\text{data}, v_2)\}, C^1_4 : C^1_3 \land C^1_1 \land M = \{v^1\} \cup M^1_2 \land v^1 \leq M^1_2, C^1_2 : \text{true})$ at line 20 of the first recursive call.

**B Extensions of the lemmas**

In this section, we discuss how the basic idea of syntactical compositional liveness 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 $\text{neqbsthole}$ is the predicate defined by the same rules as $\text{bsthole}$, with the modification that $E \neq F$ is added to the body of each inductive rule. Then $\text{neqbsthole}$ is not syntactically compositional anymore and the composition lemma

$$\exists E_2, M_2. \text{neqbsthole}(E_1, M_1, E_2, M_2) \land \text{neqbsthole}(E_2, M_2, E_3, M_3) \Rightarrow \text{neqbsthole}(E_1, M_1, E_3, M_3)$$

does not hold. This is explained as follows: Suppose $h = h_1 \ast h_2$ (where $h = h_1 \ast h_2$ denotes that $h_1$ and $h_2$ are domain disjoint and $h$ is the union of $h_1$ and $h_2$), $(s, h_1) \models \text{neqbsthole}(E_1, M_1, E_2, M_2)$ and $(s, h_2) \models \text{neqbsthole}(E_2, M_2, E_3, M_3)$, in addition, both $\text{idom}(h_1)$ and $\text{idom}(h_2)$ are nonempty. Then from the inductive definition of $\text{neqbsthole}$, 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 \text{bshole1}\((E_1, M_1, E_3, M_3)\) 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
\[
\exists E_2, M_2. \text{negbshole}\((E_1, M_1, E_2, M_2) \land \text{negbshole}\((E_2, M_2, E_3, M_3)\) \land 

E_3 \mapsto ((\text{left}, X), (\text{right}, Y), (\text{data}, v)) \Rightarrow 
\]
\[
\text{negbshole}\((E_1, M_1, E_3, M_3)\) \land E_3 \mapsto ((\text{left}, X), (\text{right}, Y), (\text{data}, v))
\]
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 \(\text{prtbst}(E, Pr, M)\) and \(\text{prtbsthole}(E, Pr_1, M_1, F, Pr_2, M_2)\) 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 \(\text{prtbst}\) is omitted here).

\[
\text{prtbsthole}(E, Pr_1, M_1, F, Pr_2, M_2) ::= E = F \land \text{emp} \land Pr_1 = Pr_2 \land M_1 = M_2
\]
\[
\text{prtbsthole}(E, Pr_1, M_1, F, Pr_2, M_2) ::= \exists X, Y, M_3, M_4, v.
\]
\[
E \mapsto \{(\text{left}, X), (\text{right}, Y), (\text{parent}, Pr_1), (\text{data}, v)\} 
\land M_1 = \{v\} \cup M_3 \cup M_4 \land M_3 < v < M_4
\]
\[
\text{prtbsthole}(E, Pr_1, M_1, F, Pr_2, M_2) ::= \exists X, Y, M_3, M_4, v.
\]
\[
E \mapsto \{(\text{left}, X), (\text{right}, Y), (\text{parent}, Pr_1), (\text{data}, v)\} 
\land M_2 = \{v\} \cup M_3 \cup M_4 \land M_3 < v < M_4
\]

Then the predicate \(\text{prtbsthole}\) enjoys the composition lemma
\[
\exists E_2, Pr_2, M_2. \text{prtbsthole}\((E_1, Pr_1, M_1, E_2, Pr_2, M_2) \land 
\text{prtbsthole}\((E_2, Pr_2, M_2, E_3, Pr_3, M_3) \Rightarrow 
\text{prtbsthole}\((E_1, Pr_1, M_1, E_3, Pr_3, M_3)\).
\]

**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 compo-
stationality. For instance, we can define the predicate \(\text{lsegeven}\) for list segments
of even length as follows,

\[
\text{lsegeven}(E, F) ::= E = F \land \text{emp},
\]
\[
\text{lsegeven}(E, F) ::= \exists X, Y. E \mapsto (\text{next}, X) \land X \mapsto (\text{next}, Y) \land \text{lsegeven}(Y, F).
\]
Then \( \text{lsegeven} \) still enjoys the composition lemma

\[
\exists E_2. \text{lsegeven}(E_1, E_2) \land \text{lsegodd}(E_2, E_3) \Rightarrow \text{lsegeven}(E_1, E_3).
\]

On the other hand, the predicate \( \text{lsegodd} \) for list segments of the odd length does not enjoy the composition lemma. The definition of \( \text{lsegodd}(E, F) \) can be obtained from that of \( \text{lsegeven}(E, F) \) by replacing the base rule with the rule

\[
\text{lsegodd}(E, F) ::= E \mapsto (\text{next}, F).
\]

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 \( \text{lsegeven} \) and \( \text{lsegodd} \) to illustrate the idea. The only difference between the inductive definition of \( \text{lsegeven} \) and that of \( \text{lsegodd} \) is as follows: The base rule of \( \text{lsegodd} \) is \( E \mapsto (\text{next}, F) \), while that of \( \text{lsegeven} \) is \( E = F \). From this, we deduce that

\[
\text{lsegodd}(E, F) \Leftrightarrow \exists X. E \mapsto \{(\text{next}, X)\} \land \text{lsegeven}(X, F).
\]

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 \((s, h) \models P(\vec{\alpha}_1, \vec{\alpha}_2, \vec{\xi}) \land P(\vec{\alpha}_2, \vec{\alpha}_3, \vec{\xi})\), then \((s, h) \models P(\vec{\alpha}_1, \vec{\alpha}_3, \vec{\xi})\).

We prove the claim by induction on the size of \( \text{ldom}(h) \).

Suppose for each \( i : 1 \leq i \leq 3, \vec{\alpha}_i = E_{i\alpha_1} \).

Since \((s, h) \models P(\vec{\alpha}_1, \vec{\alpha}_2, \vec{\xi}) \land P(\vec{\alpha}_2, \vec{\alpha}_3, \vec{\xi})\), there are \( h_1, h_2 \) such that \( h = h_1 \land h_2 \), \((s, h_1) \models P(\vec{\alpha}_1, \vec{\alpha}_2, \vec{\xi})\), and \((s, h_2) \models P(\vec{\alpha}_2, \vec{\alpha}_3, \vec{\xi})\).

If \((s, h_1) \models \bigwedge_{i=1}^2 \alpha_{1, i} = \alpha_{2, i} \land \text{emp} \), then \( \text{ldom}(h_1) = 0 \), 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 \( P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) ::= \exists X. \Pi \land E \mapsto \rho \land \Sigma_r \land P(\vec{\gamma}, \vec{\beta}, \vec{\xi}) \), and an extension of \( s \), say \( s' \), such that \((s', h_1) \models \Pi' \land E_1 \mapsto \rho' \land \Sigma_r' \land P(\vec{\gamma}', \vec{\beta}, \vec{\xi}) \), where \( \Pi', \rho', \Sigma_r', \gamma' \) are obtained from \( \Pi, \rho, \Sigma_r, \gamma \) by replacing \( \vec{\alpha}, \vec{\beta}, \vec{\xi} \) with \( \alpha_{1, 1} \alpha_{2, 2} \), \( \vec{\xi} \) respectively. From this, we deduce that there are \( h_{1, 1}, h_{1, 2}, h_{1, 3} \) such that \( h_1 = h_{1, 1} \land h_{1, 2} \land h_{1, 3}, (s', h_{1, 1}) \models E \mapsto \rho', (s', h_{1, 2}) \models \Sigma_r' \).
and \((s', h_{1,3}) \models P(\vec{\gamma}, \vec{\alpha}_2, \vec{\xi})\). Then \((s', h_{1,3} \ast h_2) \models P(\vec{\gamma}, \vec{\alpha}_2, \vec{\xi}) \ast P(\vec{\alpha}_2, \vec{\alpha}_3, \vec{\xi})\).

From the induction hypothesis, we deduce that \((s', h_{1,3} \ast h_2) \models P(\vec{\gamma}, \vec{\alpha}_3, \vec{\xi})\).

Then \((s', h_{1,1} \ast h_{1,2} \ast h_{1,3} \ast h_2) \models \Pi' \land E_1 \rightarrow \rho' \ast \Sigma_3' \ast P(\vec{\gamma}, \vec{\alpha}_3, \vec{\xi})\). We then deduce that \((s, h) \models \exists \vec{X}.\Pi' \land E_1 \rightarrow \rho' \ast \Sigma_3' \ast P(\vec{\gamma}, \vec{\alpha}_3, \vec{\xi})\).

To prove \((s, h) \models P(\vec{\alpha}_1, \vec{\alpha}_3, \vec{\xi})\), it is sufficient to prove that \((s, h) \models \exists \vec{X} .\Pi'' \land E_1 \rightarrow \rho'' \ast \Sigma_3'' \ast P(\vec{\gamma}, \vec{\alpha}_3, \vec{\xi})\), where \(\Pi'', \rho'', \Sigma_3'', \vec{\gamma}\) are obtained from \(\Pi, \rho, \Sigma_3, \vec{\gamma}\) 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 \(\Pi, \rho, \Sigma_3, \vec{\gamma}\), we know that \(\Pi'' = \Pi', \rho'' = \rho', \Sigma_3'' = \Sigma_3', \vec{\gamma}'' = \vec{\gamma}\). Since \((s, h) \models \exists \vec{X} .\Pi' \land E_1 \rightarrow \rho' \ast \Sigma_3' \ast P(\vec{\gamma}, \vec{\alpha}_3, \vec{\xi})\), we have already proved that \((s, h) \models \exists \vec{X} .\Pi'' \land E_1 \rightarrow \rho'' \ast \Sigma_3'' \ast P(\vec{\gamma}, \vec{\alpha}_3, \vec{\xi})\). 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 \(P'(\vec{\alpha}, \vec{\xi}) \iff P(\vec{\alpha}, \vec{c}, \vec{\xi}) \land \exists \vec{\beta}. P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \ast P'(\vec{\beta}, \vec{\xi}) \Rightarrow P'(\vec{\alpha}, \vec{\xi})\) hold.

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

The argument for \(\exists \vec{\beta}. P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \ast P'(\vec{\beta}, \vec{\xi}) \Rightarrow P'(\vec{\alpha}, \vec{\xi})\) goes as follows: Suppose \((s, h) \models P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \ast P'(\vec{\beta}, \vec{\xi}) \Rightarrow P'(\vec{\alpha}, \vec{\xi})\) then there are \(h_1, h_2\) such that \(h = h_1 \ast h_2\), \((s, h_1) \models P(\vec{\alpha}, \vec{\beta}, \vec{\xi})\), and \((s, h_2) \models P'(\vec{\beta}, \vec{\xi})\). From the fact that \(P'(\vec{\beta}, \vec{\xi}) \iff P(\vec{\beta}, \vec{c}, \vec{\xi})\), we know that \((s, h_2) \models P(\vec{\beta}, \vec{c}, \vec{\xi})\). Therefore, \((s, h) \models P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \ast P'(\vec{\beta}, \vec{\xi})\).

From Theorem 2, we deduce that \((s, h) \models P(\vec{\alpha}, \vec{c}, \vec{\xi})\). From the fact \(P(\vec{\alpha}, \vec{c}, \vec{\xi}) \iff P'(\vec{\alpha}, \vec{\xi})\), 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 \(\exists \vec{\beta}. P'(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \ast P'\vec{\beta}, \vec{\xi}) \Rightarrow P(\vec{\alpha}, \vec{\gamma}, \vec{\xi})\) 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 \(\text{dom}(h)\), 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 \((s, h) \models \bigwedge_{i=1}^{2} \alpha_i = \beta_i \land \text{emp}\), 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 \(P'(\vec{\alpha}, \vec{\beta}, \vec{\xi}) := \exists \vec{X}.\Pi' \land E \rightarrow \rho \ast \Sigma_3 \ast P'(\vec{\gamma}, \vec{\beta}, \vec{\xi})\), and an extension of \(s\), say \(s'\), such that \((s', h) \models \Pi' \land E \rightarrow \rho \ast \Sigma_3 \ast P'(\vec{\gamma}, \vec{\beta}, \vec{\xi})\). Then there are \(h_1, h_2, h_3\) such that \(s = h_1 \ast h_2 \ast h_3\), \((s', h_1) \models \Pi' \land E \rightarrow \rho \ast \Sigma_3 \ast P'(\vec{\gamma}, \vec{\beta}, \vec{\xi})\).
E \mapsto \rho, (s', h_2) \models \Sigma_r, and (s', h_3) \models P'(\vec{r}, \vec{\beta}, \vec{\xi}). From the induction hypothesis, we deduce that (s', h_3) \models P(\vec{r}, \vec{\beta}, \vec{\xi}). Moreover, from the assumption, we know that there is a recursive rule of P of the form \( P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) ::= \exists \vec{\alpha}. P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \). Then it follows that (s', h_1 \ast h_2 \ast h_3) \models \Sigma_r \ast P(\vec{r}, \vec{\beta}, \vec{\xi}). We then deduce that (s, h) \models P(\vec{r}, \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 \( P'(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \ast P(\vec{\beta}, \vec{\gamma}, \vec{\xi}) \Rightarrow P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \ast P(\vec{\beta}, \vec{\gamma}, \vec{\xi}) \) holds. In addition, from Theorem 2, we know that \( P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \ast P(\vec{\beta}, \vec{\gamma}, \vec{\xi}) \Rightarrow P(\vec{\alpha}, \vec{\gamma}, \vec{\xi}) \) holds. Therefore, we conclude that \( P'(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \ast P(\vec{\beta}, \vec{\gamma}, \vec{\xi}) \Rightarrow P(\vec{\alpha}, \vec{\gamma}, \vec{\xi}) \).

\[ \Box \]

**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 \( \text{prj} \), then \( P'(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \Leftrightarrow P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \) holds. In addition, from Theorem 2, we know that \( P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \ast P(\vec{\beta}, \vec{\gamma}, \vec{\xi}) \Rightarrow P(\vec{\alpha}, \vec{\gamma}, \vec{\xi}) \) holds. Therefore, we conclude that \( P'(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \ast P(\vec{\beta}, \vec{\gamma}, \vec{\xi}) \Rightarrow P(\vec{\alpha}, \vec{\gamma}, \vec{\xi}) \) 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 \( P'(\vec{\beta}, \vec{\gamma}, \vec{\xi}) \Leftrightarrow P(\vec{\beta}, \vec{\gamma}, \vec{\xi}) \), we deduce that

\[
P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \ast P'(\vec{\beta}, \vec{\gamma}, \vec{\xi}) \Rightarrow P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \ast P(\vec{\beta}, \vec{\gamma}, \vec{\xi}).
\]

From Theorem 2, we know that

\[
P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \ast P'(\vec{\beta}, \vec{\gamma}, \vec{\xi}) \Rightarrow P(\vec{\alpha}, \vec{\gamma}, \vec{\xi}).
\]

Then the second claim follows from the fact that \( P(\vec{\alpha}, \vec{\gamma}, \vec{\xi}) \Leftrightarrow P'(\vec{\alpha}, \vec{\gamma}, \vec{\xi}). \) \[ \Box \]