A Separation Logic with Data: Small Models and Automation[⋆]

Jens Katelaan[1], Dejan Jovanović[2], and Georg Weissenbacher[1]

[1] TU Wien, Vienna, Austria
[2] SRI International

Abstract. Separation logic has become a stock formalism for reasoning about programs with dynamic memory allocation. We introduce a variant of separation logic that supports lists and trees as well as inductive constraints on the data stored in these structures. We prove that this logic has the small model property, meaning that for each satisfiable formula there is a small domain in which the formula is satisfiable. As a consequence, the satisfiability and entailment problems for our fragment are in NP and coNP, respectively. Leveraging this result, we describe a polynomial SMT encoding that allows us to decide satisfiability and entailment for our separation logic.

1 Introduction

Separation logic is a popular formalism to describe the state and shape of dynamically allocated data structures and is used for the verification of programs that manipulate the heap. The formalism prominently features in Facebook's static analyzer Infer [6], which is successfully deployed on an industrial scale to analyze the memory safety of millions of lines of imperative code. This impressive scalability is facilitated by the logic's separating conjunction operator ($*$), which allows the decomposition of the program heap into disjoint regions and thus enables compositional reasoning by isolating data structures modified by a given code fragment from unaffected portions of the memory (the frame). Moreover, separation logic provides recursive predicates which describe the shape of dynamically allocated data structures such as linked lists and trees and thus enable reasoning about programs with unbounded heap.

The high expressiveness of separation logic, however, comes at the cost of undecidability [7]. Consequently, the use of separation logic in deductive verification [19,20] and symbolic execution [4] is often restricted to decidable fragments. To obtain decidability, most of these fragments adhere to at least one of the following restrictions: they only support lists [3,12,1,8,18]; they can only express structural constraints but not constraints on data stored in structures [3,10,11,5]; or they are not closed under Boolean operators, but only under

[⋆] The research presented in this paper was supported by the Austrian Science Fund (FWF) under project W1255-N23, the Austrian National Research Network S11403-N23 (RiSE), and the National Science Foundation (NSF) grant 1528153.

separating conjunction [3,10,5,13]. Yet, the computational complexity is still daunting in many cases: deciding satisfiability is ExpTime-hard for the fragments in [5,13], for instance, and [10] and the Strand logic [14] rely on a reduction of structural constraints to monadic second-order logic. Other separation logics and related formalisms are undecidable altogether [22].

We present a decidable separation logic that aims to strike a balance between expressiveness and computational complexity. Our logic supports list and tree segments as well as arbitrary data constraints, allowing us to describe common data structures such as binary search trees and max-heaps. Notably, the spatial formulas of our fragment are closed under classical Boolean operators including negation, allowing us to decide satisfiability and entailment.

This decidability result is established by showing that the structural part of our separation logic has the *small-model property*. In particular, we provide a bound that is linear in the number of variables in the formula. As a consequence, the satisfiability problem of our fragment is in NP and entailment in coNP (exploiting closure under negation). Moreover, the explicit bound provided by our small-model theorem enables a range of SAT/SMT-encodings of our separation logic. We characterize the properties that such encodings must satisfy and provide a complete polynomial-size encoding of our logic.

While we are not the first to propose an encoding of separation logic into SMT ([12] implements a reachability theory in SMT, lists with length constraints are encoded in [18], and an encoding supporting the magic wand operator but not recursive predicates is described in [23]), our approach lifts a number of requirements of encodings of comparable expressiveness and complexity [19,20].

First, the encodings in [19,20] are based on theories for reachability in function graphs [12,25] as well as a theory of finite sets, whereas we rely only on theories supported by off-the-shelf SMT solvers. Second, in [20], reasoning about trees relies on ghost variables representing pointers to parent nodes. Reachability predicates are restricted to these parent fields. In our approach, we support reasoning about left and right descendants. Third, our encoding can be easily combined with arbitrary data theories, whereas the data constraints in [19,20] are harder to generalize, since they depend on local theory extensions. Fourth, our logic supports reasoning about tree segments via a notion of stop points, thus generalizing the structural properties about tree-like structures that can be expressed in the logic compared to [20].

The paper is structured as follows. In Section 2, we introduce $\mathbf{SL}_{\mathsf{data}}$, a separation logic with support for lists, trees, and data, but without constraints on the data stored inside the lists and trees. We prove the small-model property for $\mathbf{SL}_{\mathsf{data}}$ in Section 3. In Section 4, we introduce $\mathbf{SL}^*_{\mathsf{data}}$, an extension of $\mathbf{SL}_{\mathsf{data}}$ in which the data inside of list and tree structures can be constrained by formulas from the data theory. We show how to lift the small-model property to this extended setting. In Section 5, we present a polynomial encoding of $\mathbf{SL}^*_{\mathsf{data}}$ into SMT. We conclude in Section 6. [3]

[3] Due to lack of space some proofs and additional material are omitted and can be found in the extended version.

$$\begin{array}{rll} t & := \mathbf{null} \mid x \in \mathcal{X} & \\ A_{Spatial} & ::= t \rightarrow_f t \mid \mathsf{list}(t, \mathbf{s}) \mid \mathsf{tree}(t, \mathbf{s}) \mid \mathcal{F}_{\mathsf{loc}} \mid \mathcal{F}_{\mathsf{data}} & \text{Spatial atoms} \\ F_{Spatial} & ::= A_{Spatial} \mid F_{Spatial} * F_{Spatial} & \text{Spatial formulas} \\ F & ::= F_{Spatial} \mid \neg F \mid F \vee F \mid F \wedge F & \mathbf{SL}_{\mathsf{data}} \text{ formulas} \end{array}$$

Fig. 1: Syntax of the core separation logic $\mathbf{SL}_{\mathsf{data}}$ with lists, trees, and data.

2 Separation Logic with Lists, Trees, and Data

In this section we introduce the core fragment of our separation logic of lists, trees, and data. Our approach is parametric with respect to a background theory $\mathcal{T}_{\mathsf{data}}$ of the data domain, and a background theory $\mathcal{T}_{\mathsf{loc}}$ of the location domain. We denote with $\mathcal{F}_{\mathsf{data}}$ and $\mathcal{F}_{\mathsf{loc}}$ the sets of all quantifier-free $\mathcal{T}_{\mathsf{data}}$-formulas and $\mathcal{T}_{\mathsf{loc}}$-formulas, respectively. The background theories can be instantiated with any first-order theory with equality, as usual in satisfiability modulo theories (see, e.g., [2]). We denote this logic with $\mathbf{SL}_{\mathsf{data}}$.

Syntax. We work in a many-sorted logic with equality. The signature of $\mathbf{SL}_{\mathsf{data}}$ contains the sorts $\mathcal{S} = \{\mathsf{loc}, \mathsf{data}\}$, representing locations and data, respectively. We assume a countable infinite set of (sorted) variables $\mathcal{X}$ and a dedicated constant **null** of sort loc. We denote with $\mathbf{s}$ a vector $\langle s_1, \ldots, s_n \rangle$ of variables from $\mathcal{X}$, and write ϵ for an empty vector, and $\mathbf{s}_1 \cdot \mathbf{s}_2$ for the concatenation of two vectors.

Let $\mathsf{Fld} = \{\mathsf{n}, \mathsf{l}, \mathsf{r}, \mathsf{d}\}$ be the set of field identifiers corresponding to the next element of a list node, the left and the right child of a binary tree node, and the data field. To each field $f \in \mathsf{Fld}$ we associate a binary points-to predicate $\rightarrow_f$ with the following signatures:

$$\begin{array}{ll} \rightarrow_{\mathsf{n}} : \mathsf{loc} \times \mathsf{loc} \mapsto \mathsf{Bool}\ , & \rightarrow_{\mathsf{l}} : \mathsf{loc} \times \mathsf{loc} \mapsto \mathsf{Bool}\ , \\ \rightarrow_{\mathsf{r}} : \mathsf{loc} \times \mathsf{loc} \mapsto \mathsf{Bool}\ , & \rightarrow_{\mathsf{d}} : \mathsf{loc} \times \mathsf{data} \mapsto \mathsf{Bool}\ . \end{array}$$

The logic includes two inductive predicates list and tree with signatures

$$\mathsf{list} : \mathsf{loc} \times \mathsf{loc}^* \mapsto \mathsf{Bool}\ , \qquad \mathsf{tree} : \mathsf{loc} \times \mathsf{loc}^* \mapsto \mathsf{Bool}\ .$$

The syntax of $\mathbf{SL}_{\mathsf{data}}$ is presented in Figure 1. A formula in $\mathbf{SL}_{\mathsf{data}}$ is a well-sorted Boolean combination of spatial formulas ($F_{Spatial}$). Spatial formulas are constructed by applying the separating conjunction $*$ to the *spatial atoms*. The spatial atoms are $\mathcal{T}_{\mathsf{loc}}$ and $\mathcal{T}_{\mathsf{data}}$ formulas, the points-to predicate $x \rightarrow_f y$, the list predicate $\mathsf{list}(x, \mathbf{s})$ and the tree predicate $\mathsf{tree}(x, \mathbf{s})$. To ease notation, we denote a separating conjunction of several points-to predicates over the same variable x with $x \rightarrow_{\mathsf{p}_1, \ldots, \mathsf{p}_n} (y_1, \ldots, y_n)$. The vector $\mathbf{s}$ is a vector of structural *stop points* delineating the data structure. By abuse of notation, we omit $\mathbf{s}$ when it is empty.

Our logic departs from standard presentations of separation logic (see, e.g., [24]) in several details. First, we do not have **emp**, the empty heap. It can be introduced as syntactic sugar, e.g. $\mathbf{emp} := (\mathbf{null} = \mathbf{null})$. Second, we include an independent points-to predicate for each field of lists and trees to facilitate extensions of the logic to doubly-linked and/or overlaid data structures, see e.g. [9]. Third, our lists and tree fragments represent data structures that start from x and end in stop points $\mathbf{s}$ in an ordered fashion. Additionally—unlike in many decidable separation logics [3,10]—we allow arbitrary Boolean structure outside of the spatial conjunction.

Example 1 (Syntax). Let x, y, z be variables of sort loc, and w be of sort data.

- $\mathsf{list}(x, \langle y \rangle) * \mathsf{list}(y)$ are disjoint list segments from x to y and from y to $\mathbf{null}$.
- $\mathsf{tree}(x, \langle y, z \rangle) * \mathsf{tree}(y) * \mathsf{tree}(z)$ represents a binary tree rooted in x that contains two subtrees y and z ordered from left to right, as specified by $\langle y, z \rangle$.
- $(x \rightarrow_{\mathsf{n,d}} (y, w)) * \mathsf{list}(y) * (w > 0)$ (where $\mathcal{T}_{\mathsf{data}}$ is an arithmetic theory) states that x is a list node with data $w > 0$ pointing to a list with head y.

Semantics. We denote with $f = \{x_1, \dots, x_n \mapsto y_1, \dots, y_n\}$ a partial function that maps x_i to y_i and is otherwise undefined, and write $f = \emptyset$ if f is undefined everywhere. We write $\mathrm{dom}(f)$ and $\mathrm{img}(f)$ for the domain and image of f.

The semantics of $\mathbf{SL}_{\mathsf{data}}$ formulas are defined in terms of heap interpretations. Let $X \subseteq \mathcal{X}$ be a set of variables. A *heap interpretation* $\mathcal{M}$ over X is a map that interprets each sort $\sigma \in \mathcal{S}$ as a non-empty domain $\sigma^{\mathcal{M}}$, each $x \in X \cup \{\mathbf{null}\}$ of sort σ as an element $x^{\mathcal{M}} \in \sigma^{\mathcal{M}}$, and each points-to predicate $\rightarrow_f$ of sort $\sigma_1 \times \sigma_2 \mapsto \mathsf{Bool}$ is interpreted as a partial function $f^{\mathcal{M}} : \sigma_1 \rightharpoonup \sigma_2$ with finite domain such that $f^{\mathcal{M}}(\mathbf{null})$ is undefined (i.e., null may never be allocated) and such that $\mathrm{dom}(\mathsf{n}) \cap (\mathrm{dom}(\mathsf{l}) \cup \mathrm{dom}(\mathsf{r})) = \emptyset$, i.e., a location cannot be both a list and a tree location.

We denote with $\mathcal{M}[x_1, \dots, x_n \mapsto v_1, \dots, v_n]$ a heap interpretation over $X \cup \{x_1, \dots, x_n\}$ that differs from $\mathcal{M}$ only by interpreting the variables x_i as values v_i. Let $\ell_1, \ell_2 \in \mathsf{loc}^{\mathcal{M}}$. We write $\ell_1 \rightarrow_{\mathcal{M}} \ell_2$ if $\ell_2 = f^{\mathcal{M}}(\ell_1)$ for some $f \in \mathsf{Fld}$. We extend this notation to variables and write $x \rightarrow_{\mathcal{M}} y$ if for two variables $x, y \in X$ it holds that $x^{\mathcal{M}} \rightarrow_{\mathcal{M}} y^{\mathcal{M}}$. We denote with $\rightarrow^*_{\mathcal{M}}$ and $\rightarrow^+_{\mathcal{M}}$ the usual Kleene closures of $\rightarrow_{\mathcal{M}}$ and say that ℓ_2 is *reachable from* ℓ_1 if $\ell_1 \rightarrow^*_{\mathcal{M}} \ell_2$.

A location $\ell \in \mathsf{loc}^{\mathcal{M}}$ is an *allocated location* in $\mathcal{M}$ if there exists an $f \in \{\mathsf{n}, \mathsf{l}, \mathsf{r}\}$ such that $\ell \in \mathrm{dom}(f^{\mathcal{M}})$. We define $\mathsf{loc}^{\mathcal{M}}_{\mathsf{list}} := \left\{\ell \in \mathsf{loc}^{\mathcal{M}} \mid \ell \in \mathrm{dom}(\mathsf{n})\right\}$ and $\mathsf{loc}^{\mathcal{M}}_{\mathsf{tree}} := \left\{\ell \in \mathsf{loc}^{\mathcal{M}} \mid \ell \in \mathrm{dom}(\mathsf{l}) \cup \mathrm{dom}(\mathsf{r})\right\}$. Location ℓ is *fully allocated* in $\mathcal{M}$ if it allocates data and either the next pointer or both the left and right pointer, i.e., if $\ell \in (\mathrm{dom}(\mathsf{n}^{\mathcal{M}}) \cup (\mathrm{dom}(\mathsf{l}^{\mathcal{M}}) \cap \mathrm{dom}(\mathsf{r}^{\mathcal{M}}))) \cap \mathrm{dom}(\mathsf{d}^{\mathcal{M}})$. Location ℓ is *labeled* in $\mathcal{M}$ if there exists an $x \in X \cup \{\mathbf{null}\}$ with $x^{\mathcal{M}} = \ell$. Otherwise, ℓ is unlabeled.

The *size of* $\mathcal{M}$, denoted $|M|$, is the number of allocated locations in $\mathcal{M}$.[4] The *size of a formula* F, denoted $|F|$, is defined as the numbers of terms, atomic formulas, and operators in F (including symbols from $\mathcal{T}_{\mathsf{loc}}$ and $\mathcal{T}_{\mathsf{data}}$).

[4] This size notion captures the amount of allocated memory rather than the amount of addressable memory (which is determined by the interpretation of the location domains).

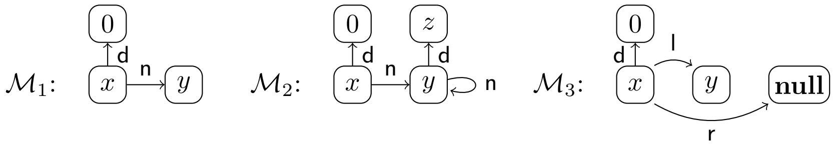
$$\begin{array}{ll}
\mathcal{M} \models x \rightarrow_f y & \text{iff } f^{\mathcal{M}} = \{x^{\mathcal{M}} \mapsto y^{\mathcal{M}}\} \land\ \forall h \in \mathsf{Fld}\ .\ h \neq f \implies h = \emptyset \\
\mathcal{M} \models F_{\mathsf{loc}} & \text{iff } (\mathcal{M} \models_{\mathsf{loc}} F_{\mathsf{loc}}) \land \forall h \in \mathsf{Fld}\ .\ h = \emptyset \\
\mathcal{M} \models F_{\mathsf{data}} & \text{iff } (\mathcal{M} \models_{\mathsf{data}} \mathcal{F}_{\mathsf{loc}}) \land \forall h \in \mathsf{Fld}\ .\ h = \emptyset \\
\mathcal{M} \models F * G & \text{iff } \mathcal{M} = \mathcal{M}_1 \oplus \mathcal{M}_2 \land \mathcal{M}_1 \models F \land \mathcal{M}_2 \models G \\
\mathcal{M} \models F \land G & \text{iff } \mathcal{M} \models F \land \mathcal{M} \models G \\
\mathcal{M} \models F \lor G & \text{iff } \mathcal{M} \models F \lor \mathcal{M} \models G \\
\mathcal{M} \models \neg F & \text{iff not } \mathcal{M} \models F \\
\mathcal{M} \models x \notin \mathbf{s} & \text{iff } \bigwedge_{y \in \mathbf{s}} \mathcal{M} \models x \neq y \\
\mathcal{M} \models \mathsf{pred}(x, \mathbf{s}) & \text{iff } \exists i\ .\ \mathcal{M} \models \mathsf{pred}^i_{\mathbf{s}}(x, \mathbf{s}) \land \bigwedge_{y_1 \neq y_2 \in \mathbf{s}} \mathcal{M} \models y_1 \neq y_2 \\
\mathcal{M} \models \mathsf{pred}^0_{\mathbf{t}}(x, \epsilon) & \text{iff } \mathcal{M} \models x = \mathbf{null} \\
\mathcal{M} \models \mathsf{pred}^0_{\mathbf{t}}(x, \langle y \rangle) & \text{iff } \mathcal{M} \models x = y \\
\mathcal{M} \models \mathsf{list}^i_{\mathbf{t}}(x, \mathbf{s}) & \text{iff } \exists \ell \in \mathsf{loc}^{\mathcal{M}}, d \in \mathsf{data}^{\mathcal{M}}\ . \\
& \quad \mathcal{M}[z, w \mapsto \ell, d] \models x \notin \mathbf{t} * x \rightarrow_{\mathsf{n,d}} (z, w) * \mathsf{list}^{i-1}_{\mathbf{t}}(z, \mathbf{s}) \\
\mathcal{M} \models \mathsf{tree}^i_{\mathbf{t}}(x, \mathbf{s}) & \text{iff } \mathbf{s} = \mathbf{s}_1 \cdot \mathbf{s}_2 \land i = i_1 + i_2 + 1 \\
& \quad \land\ \exists \ell_1, \ell_2 \in \mathsf{loc}^{\mathcal{M}}, d \in \mathsf{data}^{\mathcal{M}}\ .\ \mathcal{M}' = \mathcal{M}[z_1, z_2, w \mapsto \ell_1, \ell_2, d] \\
& \quad \land\ \mathcal{M}' \models x \notin \mathbf{t} * x \rightarrow_{\mathsf{l,r,d}} (z_1, z_2, w) * \mathsf{tree}^{i_1}_{\mathbf{t}}(z_1, \mathbf{s}_1) * \mathsf{tree}^{i_2}_{\mathbf{t}}(z_2, \mathbf{s}_2)
\end{array}$$

Fig. 2: Semantics of the core separation logic $\mathbf{SL}_{\mathsf{data}}$. Variable z is a fresh variable of sort $\mathsf{loc}_{\mathsf{list}}$, z_1 and z_2 are fresh variables of sort $\mathsf{loc}_{\mathsf{tree}}$, and w is a fresh variable of sort data. For brevity we denote with pred either list or tree.

Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be heap interpretations over X that agree on the interpretation of all sorts, variables, and constants. We say that $\mathcal{M}_1$ is a *sub-interpretation* of $\mathcal{M}_2$, written $\mathcal{M}_1 \subseteq \mathcal{M}_2$, if $f^{\mathcal{M}_1} \subseteq f^{\mathcal{M}_2}$ for all fields $f \in \mathsf{Fld}$. $\mathcal{M}_1$ and $\mathcal{M}_2$ are *disjoint interpretations* if for all $f \in \mathsf{Fld}$, $\mathrm{dom}(f^{\mathcal{M}_1}) \cap \mathrm{dom}(f^{\mathcal{M}_2}) = \emptyset$. If $\mathcal{M}_1$ and $\mathcal{M}_2$ are disjoint, we denote with $M_1 \oplus M_2$ the composition of $\mathcal{M}_1$ and $\mathcal{M}_2$ defined by taking the point-wise union of the functions f for each field $f \in \mathsf{Fld}$.

The semantics of a formula $F \in \mathbf{SL}_{\mathsf{data}}$ with respect to a heap interpretation $\mathcal{M}$ is defined inductively over the structure of F, as presented in Figure 2. The semantics of location and data formulas $F_{\mathsf{loc}} \in \mathcal{F}_{\mathsf{loc}}$ and $F_{\mathsf{data}} \in \mathcal{F}_{\mathsf{data}}$ is defined by their interpretation in $\mathcal{T}_{\mathsf{loc}}$ and $\mathcal{T}_{\mathsf{data}}$ (denoted with $\mathcal{M} \models_{\mathsf{loc}} F_{\mathsf{loc}}$ and $\mathcal{M} \models_{\mathsf{data}} F_{\mathsf{data}}$), respectively. Our semantics is *precise* in the usual separation-logic sense (see e.g. [3]), meaning $\mathcal{M} \models x_1 \rightarrow_f x_2$ implies that $x_1 \rightarrow_f x_2$ is the only pointer that is defined in $\mathcal{M}$.

Example 2 (Semantics). Consider the following graphical representations of three heap interpretations.



Each field interpretation corresponds directly to the edges of the graph labeled with that field. For example, in $\mathcal{M}_2$, the next fields of lists are interpreted as

$\mathsf{n}^{\mathcal{M}_2} = \{x^{\mathcal{M}_2}, y^{\mathcal{M}_2} \mapsto y^{\mathcal{M}_2}, y^{\mathcal{M}_2}\}$. In these interpretations we have that

$$\begin{array}{lll} \mathcal{M}_1 \models \mathsf{list}(x, \langle y \rangle), & \mathcal{M}_2 \not\models \mathsf{list}(x, \langle y \rangle), & \mathcal{M}_2 \not\models \mathsf{list}(x, \langle y \rangle) * \mathsf{list}(y, \langle y \rangle), \\ \mathcal{M}_2 \models \mathsf{list}(x, \langle y \rangle) * (y \rightarrow_{\mathsf{n,d}} (y, z)), & & \mathcal{M}_3 \models \mathsf{tree}(x, \langle y \rangle) * (x \neq y). \end{array}$$

3 A Small Model Property for $\mathbf{SL}_{\mathsf{data}}$

In this section, we show that every satisfiable $\mathbf{SL}_{\mathsf{data}}$ formula F is satisfiable by a model with a small interpretation of the loc domain. More precisely, we will derive a size bound that is linear in the number of variables in F. This result has two implications. First, the satisfiability problem for $\mathbf{SL}_{\mathsf{data}}$ is in NP if satisfiability for $\mathcal{T}_{\mathsf{loc}}$ and $\mathcal{T}_{\mathsf{data}}$ is in NP, as we can guess and check a polynomially-sized model.[5] Second, as we will argue in Section 5, the size bound enables encodings of $\mathbf{SL}_{\mathsf{data}}$ into SMT without the need to reason about unbounded reachability.

To derive a tight bound, we distinguish between the *list variables* and the *tree variables* in F. A variable is a list variable if it appears in at least one atom of the form $x_1 \rightarrow_{\mathsf{n}} x_2$ or $\mathsf{list}(x_1, \langle x_2, \ldots, x_k \rangle)$; a variable is a tree variable if it appears in an atom of the form $x_1 \rightarrow_{\mathsf{l}} x_2$, $x_1 \rightarrow_{\mathsf{r}} x_2$ or $\mathsf{tree}(x_1, \langle x_2, \ldots, x_k \rangle)$. The main result in the current section is the following:

Theorem 1 (Small-model property for $\mathbf{SL}_{\mathsf{data}}$). *Let F be a satisfiable $\mathbf{SL}_{\mathsf{data}}$ formula with n_{list} list variables, n_{tree} tree variables, and at most $k \geq 1$ stop locations per* tree *predicate. Then there is a heap interpretation $\mathcal{M}$ that satisfies F such that $|\mathcal{M}| \leq \max(4, 2n_{\mathsf{list}} + (2 + k)n_{\mathsf{tree}})$.*

Example 3. To illustrate the bound of Theorem 1, consider tree variables $x_1, \ldots, x_n$ and $\mathsf{s} = \langle s_1, \ldots, s_n \rangle$, for $n = 2^k$, and the formula $\mathsf{tree}(x_1, \mathsf{s}) * \cdots * \mathsf{tree}(x_n, \mathsf{s})$. A heap that satisfies the above formula needs to accommodate n separate trees that all end in n stop points. The smallest such heap $\mathcal{M}$ would therefore include n full binary trees with n leaves and have the allocated size $|\mathcal{M}| = n(n - 1)$. Although this size is quadratic in n, this is only because we are using n stop points. In practice, the number of (program) variables pointing into a tree structure will be an upper bound for the number of stop points. This number is generally low—for example at most 2 for many typical tree traversal and tree update algorithms. ☐

To prove Theorem 1, we take an arbitrary model $\mathcal{M} \models F$ and transform it into a small model $\mathcal{M}'$ such that $\mathcal{M}' \models F$. We define separate transformations for positive and negative heap interpretations. A positive heap interpretation is one that satisfies a positive spatial formula. More precisely, a heap interpretation $\mathcal{M}$ over X is *positive* if there exists a formula $F = A_1 * \cdots * A_k$ such that $\mathcal{M} \models F$. A heap interpretation that is not positive is *negative.*

Positive heap interpretations are well behaved. In particular, every unlabeled location in a positive interpretation $\mathcal{M}$ is contained in exactly one list or tree in $\mathcal{M}$, as the separating conjunction precludes sharing of unlabeled allocated

[5] This is the case, e.g., in the common case when $\mathcal{T}_{\mathsf{loc}}$ is the theory of equality and $\mathcal{T}_{\mathsf{data}}$ is the theory of linear arithmetic.

locations between multiple data structures. The semantics of the list and tree predicates additionally enforce acyclicity within each data structure and full allocation of all unlabeled locations. We formalize these and related observations in the following lemma.

Lemma 1. *In a positive heap interpretation $\mathcal{M}$ the following holds:*

1. *Every unlabeled allocated location is fully allocated.*
2. *If $\mathcal{M} = \mathcal{M}_1 \oplus \mathcal{M}_2$, with both $\mathcal{M}_1$ and $\mathcal{M}_2$ positive, then every unlabeled and allocated location of $\mathcal{M}$ is allocated in exactly one of $\mathcal{M}_1$ and $\mathcal{M}_2$.*
3. *For every unlabeled allocated location ℓ, there is at exactly one x such that ℓ is reachable from $x^{\mathcal{M}}$ without going through another labeled location, i.e., such that $x \to^{+}_{\mathcal{M}} \ell' \to^{*}_{\mathcal{M}} \ell$ implies that ℓ' is unlabeled.*
4. *Every loop in $\mathcal{M}$ must contain a labeled location: if $\ell_1 \to^{+}_{\mathcal{M}} \ell_1$, then every such closed path can be split into $\ell_1 \to^{*}_{\mathcal{M}} x^{\mathcal{M}} \to^{*}_{\mathcal{M}} \ell_1$ for some $x \in X$.*

Note that by definition, all negative heap interpretations falsify *all* spatial formulas. So if $\mathcal{M}$ is negative and $\mathcal{M} \models F$, we know that $\mathcal{M}$ falsifies all spatial subformulas of F, as would *any* negative heap interpretation. For example, we can replace $\mathcal{M}$ by a small model that contains a loop, because such a model is negative by Lemma 1. Intuitively, this is why all formulas that are satisfied by negative heap interpretations have the small model property.

Lemma 2. *There exists a heap interpretation $\mathcal{M}_0$, with $|\mathcal{M}_0| = 4$, that falsifies all spatial formulas $A_1 * \cdots * A_n$. Moreover, for any negative heap interpretation $\mathcal{M}$ and formula F, if $\mathcal{M} \models F$ then $\mathcal{M}_0 \models F$.*

For the remainder of this section, we assume that F is a formula that is satisfiable in a positive interpretation $\mathcal{M}$ over variables X. First, we define a transformation of heap interpretations that removes a single location from the field interpretations. We then show that, using this transformation, we can minimize $\mathcal{M}$ to a model that is still positive and satisfies F. Finally, we show that the size of this positive minimal model is bounded as in Theorem 1.

Location removal. Let $\ell \in \mathsf{loc}^{\mathcal{M}}$ be an allocated location, i.e., there is at least one field $f \in \{\mathsf{n}, \mathsf{l}, \mathsf{r}\}$ with $\ell \in \mathrm{dom}(f^{\mathcal{M}})$. We say that such a location ℓ_0 is *removable through its field* f if the field f is defined (allocated) at ℓ_0 and for all other fields $g \neq \mathsf{d}$, $g^{\mathcal{M}}(\ell)$ is either **null** or undefined. (Note that this does not preclude that also $f^{\mathcal{M}}(\ell) = \textbf{null}$.) If ℓ is a location removable through its field f, we write $\mathcal{M}' = \mathcal{M} \setminus \{\ell\}$ for the interpretation that mimics $\mathcal{M}$ apart from avoiding the location ℓ, i.e., for all $g \in \{\mathsf{n}, \mathsf{l}, \mathsf{r}\}$ and for all locations ℓ', we define

$$g^{\mathcal{M}'}(\ell') = \begin{cases} g^{\mathcal{M}}(\ell') & \text{if } \ell' \neq \ell \text{ and } g^{\mathcal{M}}(\ell') \neq \ell \ , \\ g^{\mathcal{M}}(\ell) & \text{if } \ell' \neq \ell \text{ and } g^{\mathcal{M}}(\ell') = \ell \\ \bot & \text{if } \ell' = \ell \end{cases}$$

In addition, location ℓ is removed from the data interpretation, i.e., we set $\mathsf{d}^{\mathcal{M}'} = \mathsf{d}^{\mathcal{M}} \setminus \{\ell \mapsto d\}$. Figure 3 illustrates location removal for lists and trees.

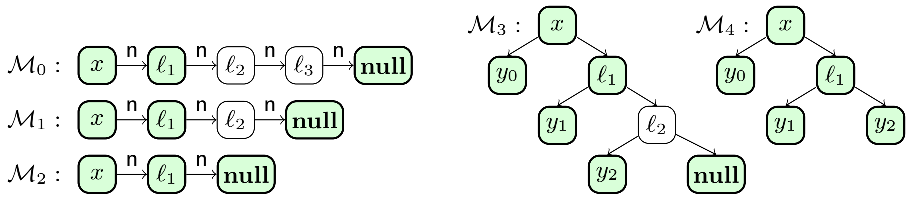


Fig. 3: Subsequent removal of removable non-essential list locations ℓ_3 and ℓ_2 and tree location ℓ_2, transforming $\mathcal{M}_0$ via $\mathcal{M}_1$ into $\mathcal{M}_2$ and $\mathcal{M}_3$ into $\mathcal{M}_4$. Essential locations are displayed in green.

Essential locations. Location removal reduces the allocated size of $\mathcal{M}$. We now characterize the locations that can safely be removed from $\mathcal{M}$ without falsifying the formula F. Assume two distinct labeled locations ℓ_1 and ℓ_2 from $\mathsf{loc}^{\mathcal{M}}$. We call a location $\ell \in \mathsf{loc}^{\mathcal{M}}$ an *induction indicator* for (ℓ_1, ℓ_2) if $\ell_1 \to_{\mathcal{M}} \ell \to^{+}_{\mathcal{M}} \ell_2$. Intuitively, a location is an induction indicator if it is a potential witness of a first step of a longer unrolling of an inductive predicate. An induction indicator cannot be removed as such a removal might change the interpretation of a predicate $x \to_f y$ from false to true. An allocated location ℓ is an *essential location* iff ℓ is a labeled location or ℓ is an induction indicator.

Model minimization. We now proceed to show that if $\mathcal{M}' = \mathcal{M} \setminus \{\ell\}$ for a non-essential, removable location ℓ, then $\mathcal{M} \models F$ if and only if $\mathcal{M}' \models F$.

Lemma 3. *Let $\mathcal{M}$ be a positive heap interpretation over X and let A be a spatial atom. Let ℓ be a non-essential, removable location and let $\mathcal{M}' = \mathcal{M} \setminus \{\ell\}$. Then $A \models \mathcal{M}$ if and only if $\mathcal{M}' \models A$.*

If $F = A_1 * \cdots * A_n$ is a separating conjunction, we use that $\mathcal{M} = \mathcal{M}_1 \oplus \cdots \oplus \mathcal{M}_k$ for some $\mathcal{M}_i$ such that $\mathcal{M}_i \models A_i$. By Lemma 1, ℓ is fully allocated in exactly one $\mathcal{M}_i$, has no direct predecessors outside of $\mathcal{M}_i$, and is removable in $\mathcal{M}_i$. This lets us reduce the case for the separating conjunction to Lemma 3 and conclude:

Lemma 4. *Let $F = A_1 * \cdots * A_n$ be a conjunction of spatial atoms A_i, and let $\mathcal{M}$ be a positive heap interpretation. Let ℓ be a non-essential removable location of $\mathcal{M}$, and let $\mathcal{M}' = \mathcal{M} \setminus \{\ell\}$. Then $\mathcal{M} \models F$ if and only if $\mathcal{M}' \models F$.*

Finally, by a simple induction proof over the Boolean structure of F that relies on Lemma 4, we conclude:

Lemma 5. *Let $\mathcal{M}$ be a positive heap interpretation over X, ℓ be a non-essential removable location, and let $\mathcal{M}' = \mathcal{M} \setminus \{\ell\}$. Then $\mathcal{M} \models F$ if and only if $\mathcal{M}' \models F$.*

It remains to be shown that by iterating location removal, we eventually terminate with a model of a size that satisfies the bound in Theorem 1.

Proof (of Theorem 1). Let $\mathcal{M} \models F$. If F is also satisfied in the small model $\mathcal{M}_0$ from Lemma 2, we are done. Otherwise, consider the DNF form of F. Since

$\mathcal{M} \models F$, there is at least one conjunct $L_1 \wedge \ldots \wedge L_m$ of the DNF such that $\mathcal{M} \models L_i$, for all i. If all L_i are negated, then $\mathcal{M}_0$ satisfies them, and therefore $\mathcal{M}_0 \models F$, contrary to assumption. Therefore, there is at least one positive $L_i = A_1 * \cdots * A_n$, with $\mathcal{M} \models A_1 * \cdots * A_n$. We iterate location removal until we end in a positive model $\mathcal{M}' \models F$ and $\mathcal{M}' \models A_1 * \cdots * A_n$, that has no more removable non-essential locations. We estimate the size of $\mathcal{M}'$. An allocated location from $\mathcal{M}'$ is either essential or non-essential.

There are at most $N_1 = 2n_{\mathsf{list}} + 3n_{\mathsf{tree}}$ allocated essential locations in $\mathcal{M}'$: In $\mathcal{M}'$, every list location has at most one successor location, and every tree location has at most two direct successors (with respect to $\rightarrow_{\mathcal{M}}$). These are potential induction indicators which, taken together with labeled locations, give a total of at most N_1 essential locations.

Now, let ℓ be an allocated but non-essential location in $\mathcal{M}'$. Since $\mathcal{M}' \models A_1 * \cdots * A_n$. Location ℓ must be allocated by one of the A_i atoms. A_i cannot be a $\rightarrow_f$ predicate as ℓ would otherwise be labeled and therefore essential. A_i cannot be a list predicate either, as ℓ would be removable. A_i must therefore be a $\mathsf{tree}(x, \mathsf{s})$ predicate. We claim that both left and right subtree of ℓ must contain (distinct) stop variables s_1 and s_2. If not, then one of the descendants of ℓ would be removable. The location ℓ is therefore the lowest common ancestor of s_1 and s_2. Assuming that the number of stop points in s is at most k, for a tree starting at x, there are at most $k-1$ such common ancestors. Since there can be at most n_{tree} non-empty tree predicates among A_i, there are at most $N_2 = n_{\mathsf{tree}}(k-1)$ allocated non-essential locations in $\mathcal{M}'$. We can thus bound the size of $\mathcal{M}'$ with $N_1 + N_2 = 2n_{\mathsf{list}} + (2+k)n_{\mathsf{tree}}$. $\square$

4 Extending $\mathbf{SL}_{\mathsf{data}}$ with Data Constraints

In this section we add to $\mathbf{SL}_{\mathsf{data}}$ the possibility to constrain the data values in lists and trees by means of passing $\mathcal{T}_{\mathsf{data}}$ formulas as additional parameter to the list and tree predicates. We call the extended logic $\mathbf{SL}^*_{\mathsf{data}}$. Our goal is to reason about data properties of inductive structures that appear frequently in practice, e.g., a list being sorted, or a tree being a binary search tree.

We assume two dedicated fresh variables α and β from $\mathcal{X}$ of sort data to be used exclusively in data predicates. We call a formula $P(\alpha)$ a *unary data predicate*, and a pair $(f, P(\alpha, \beta))$, for $f \in \{\mathsf{n}, \mathsf{l}, \mathsf{r}\}$, a *binary data predicate*. Both types of predicates may also contain other variables from $\mathcal{X} \setminus \{\alpha, \beta\}$. We pass a set of data predicates $\mathcal{P}$ as additional parameter to the predicates, obtaining ternary predicates $\mathsf{list}(x, \mathsf{s}, \mathcal{P})$ and $\mathsf{tree}(x, \mathsf{s}, \mathcal{P})$. As before, for brevity, if either of s or $\mathcal{P}$ is empty, we omit them. Semantics of an inductive data predicate $\mathsf{pred}(x, \mathsf{s}, \mathcal{P})$, in a heap interpretation $\mathcal{M}$, are as follows:

1. The predicate holds in $\mathcal{M}$ only if it holds without the data constraints, i.e., $\mathsf{pred}(x, \mathsf{s})$ must be true in $\mathcal{M}$ and therefore $\mathcal{M}$ describes a pred structure.
2. For each unary data predicate $P(\alpha) \in \mathcal{P}$, all allocated data in $\mathcal{M}$ must satisfy P. More precisely, for all $(\ell, d) \in \mathsf{d}^{\mathcal{M}}$, we have that $\mathcal{M}[\alpha \mapsto d] \models_{\mathsf{data}} P$ holds.

3. For each binary predicate $(f, P(\alpha, \beta)) \in \mathcal{P}$, all allocated data must be related with all of its f-descendants through P. More precisely, for all $(\ell_1, d_1), (\ell_2, d_2) \in \mathsf{d}^{\mathcal{M}}$ such that $f^{\mathcal{M}}(\ell_1) \rightarrow^*_{\mathcal{M}} \ell_2$, $\mathcal{M}[\alpha, \beta \mapsto d_1, d_2] \models_{\mathsf{data}} P$ holds.

Example 4 (Data Predicates). We illustrate the inductive predicates through representative examples of data predicates over lists and trees. The predicates

$$\mathsf{list}(x, \{(\alpha = 0)\}) \ , \qquad \mathsf{list}(x, \{(\mathsf{n}, \alpha \neq \beta)\}) \ , \qquad \mathsf{list}(x, \{(\mathsf{n}, \alpha < \beta)\}) \ ,$$

describe a list with all data values equal to 0, a list with all data values distinct, and a list with data values increasing. The predicates

$$\mathsf{tree}(x, \{(\mathsf{l}, \beta < \alpha), (\mathsf{r}, \beta > \alpha)\}) \ , \qquad \mathsf{tree}(x, \{(\mathsf{l}, \beta < \alpha), (\mathsf{r}, \beta < \alpha)\}) \ ,$$

describe a binary search tree, and a max-heap. Formula $\mathsf{list}(x, \langle m \rangle, \{(\alpha < M)\}) * m \rightarrow_{\mathsf{n},\mathsf{d}} (y, M) * \mathsf{list}(y, \{(\alpha > M)\})$ describes a partitioned list where the left partition contains elements smaller than the pivot m, and the right partition contains elements larger than the pivot m. Formula $\mathsf{list}(x) * \mathsf{list}(y, \{(\alpha \neq a)\}) \wedge \neg(\mathsf{list}(x, \{(\alpha \neq a)\}) * \mathsf{list}(y))$ describes a list x that contains a data value a, and a list y that does not contain a, meaning that the sets of values in the lists x and y are different. □

We now lift the small-model property to full $\mathbf{SL}^*_{\mathsf{data}}$.

Theorem 2 (Small-model property for $\mathbf{SL}^*_{\mathsf{data}}$). *Let F be a satisfiable $\mathbf{SL}^*_{\mathsf{data}}$ formula with n_{list} list variables, n_{tree} tree variables, m_{list} list predicates with data constraints, m_{tree} tree predicates with data constraints, and at most $k \geq 1$ stop locations per* tree *predicate. Then there is a heap interpretation $\mathcal{M}$ that satisfies F such that $|\mathcal{M}| \leq \max(4, 2n_{\mathsf{list}} + (3 + k)n_{\mathsf{tree}} + 2m_{\mathsf{list}} + 2m_{\mathsf{tree}})$.*

Intuitively, the changes to the reasoning are minimal since the data predicates are universal, and the location removal does not invalidate predicates that are true. We only must be careful to ensure that location removal does not change the value of a data predicate from false to true.

Example 5. Let $F = \mathsf{list}(x_1, \langle x_2 \rangle, \{(\mathsf{n}, P_1)\}) \wedge \neg\mathsf{list}(x_1, \langle x_2 \rangle, \{(\mathsf{n}, P_2)\})$, where $P_1 = (\alpha < \beta)$ and $P_2 = (2\alpha \leq \beta)$, and consider the following model $\mathcal{M}_0$.

$$\mathcal{M}_0 : \begin{tikzcd} 2 & 4 & 5 & 8 & \\ x_1 \arrow[u, "\mathsf{d}"] \arrow[r, "\mathsf{n}"] & \ell_1 \arrow[u, "\mathsf{d}"] \arrow[r, "\mathsf{n}"] & \ell_2 \arrow[u, "\mathsf{d}"] \arrow[r, "\mathsf{n}"] & \ell_3 \arrow[u, "\mathsf{d}"] \arrow[r, "\mathsf{n}"] & x_2 \end{tikzcd}$$

We have that $\mathcal{M}_0 \models F$. In addition, locations ℓ_2 and ℓ_3 are not essential and can be removed. But, in models $\mathcal{M}_1 = \mathcal{M}_0 \setminus \{\ell_2\}$ and $\mathcal{M}_2 = \mathcal{M}_0 \setminus \{\ell_3\}$, we have that $\mathcal{M}_1 \not\models F$ but $\mathcal{M}_2 \models F$. □

To avoid the situation described above, we must ensure that for each data predicate $\mathsf{pred}(x, \mathsf{s}, \mathcal{P})$ that is falsified due to some $P \in \mathcal{P}$ being false, we have a designated pair of locations that witness the reason why P is false.

Proof (of Theorem 2). As in the proof of Theorem 1, assume that F has a positive model $\mathcal{M}$, i.e., $\mathcal{M} \models A_1 * \cdots * A_n$ for some A_i from F. In this model, for each falsified data predicate, we designate at most 2 additional locations as essential and proceed with reducing $\mathcal{M}$ to $\mathcal{M}'$ by removing all removable non-essential locations. The model $\mathcal{M}'$ therefore contains at most $N_1 = 2n_{\mathsf{list}} + 3n_{\mathsf{tree}} + 2m_{\mathsf{list}} + 2m_{\mathsf{list}}$ locations marked as essential. We now count the number of allocated non-essential locations ℓ in $\mathcal{M}'$. As in the proof of Theorem 1, location ℓ must be allocated as part of a predicate $\mathsf{pred}(x, \mathbf{s})$, whose interpretation can also contain two associated witness locations w_1 and w_2 with $w_1 \to^+_{\mathcal{M}'} w_2$. Location ℓ is then either the lowest common ancestor of two stop location $s_i^{M'}$ and $s_j^{M'}$, or a lowest common ancestor of some stop locations $s_i^{M'}$ and w_2. Assuming that the number of stop points in $\mathbf{s}$ is at most k, for a tree starting at x there are at most k such common ancestors. Since there can be at most n_{tree} non-empty tree predicates among A_i, there are at most $N_2 = kn_{\mathsf{tree}}$ allocated non-essential locations in $\mathcal{M}'$. We can thus bound the size of $\mathcal{M}'$ with $|\mathcal{M}'| \leq N_1 + N_2 = 2n_{\mathsf{list}} + (3 + k)n_{\mathsf{tree}} + 2m_{\mathsf{list}} + 2m_{\mathsf{tree}}$. □

As opposed to the symbolic-heap family of separation logics (e.g. [3,5]), the logic $\mathbf{SL}^*_{\mathsf{data}}$ is closed under negation, and we can solve the entailment problem $F \models G$ by checking whether $F \wedge \neg G$ is unsatisfiable.

Corollary 1. *If the satisfiability problem for $\mathcal{T}_{\mathsf{data}}$ is in* NP *then the satisfiability problem for $\mathbf{SL}^*_{\mathsf{data}}$ is in* NP*, and the entailment problem for $\mathbf{SL}^*_{\mathsf{data}}$ is in* coNP.

5 Encoding $\mathbf{SL}^*_{\mathsf{data}}$ into SMT

We now present an encoding of $\mathbf{SL}^*_{\mathsf{data}}$ formulas into SMT. We show that every formula $F \in \mathbf{SL}^*_{\mathsf{data}}$ can be encoded in polynomial time (and size) as a formula in the SMT theory of arrays that is satisfiable iff F is satisfiable. Our approach relies on the theory of arrays extended with combinators that can express constant arrays and express point-wise array operations [16]. We denote this theory by $\mathcal{T}_{\mathsf{array}}$. In $\mathcal{T}_{\mathsf{array}}$, it is possible to express universal statements about array elements without relying on quantifiers. Moreover, the satisfiability of generalized array formulas is decidable in NP with effective decision procedures implemented in popular SMT solvers such as Z3 [17] and Boolector [21].

The basic theory of arrays defines functions store and $\cdot[\cdot]$, as usual (see, e.g. [15]). The generalized theory adds a constant combinator $\mathbf{K}$ and a map combinator map such that $\mathbf{K}(c)[i] = c$, for a constant c, and $\mathsf{map}_f(A_1, \ldots, A_n)[i] = f(A_1[i], \ldots, A_n[i])$, for a function f. The array combinators are expressive enough to express basic set-theoretic operations. For example, we can view a set of locations as an array mapping loc to Bool and define set operations as follows

$$\begin{array}{lll} \{x\} = \mathsf{store}(\mathbf{K}(\bot), x, \top) & x \in X = X[x] & \mathsf{empty}(X) = (X = \mathbf{K}(\bot)) \\ X \subseteq Y = \mathsf{map}_{\Rightarrow}(X, Y) & X \cup Y = \mathsf{map}_{\vee}(X, Y) & X \cap Y = \mathsf{map}_{\wedge}(X, Y) \end{array}$$

In the following, we use the set notation as a shorthand for the equivalent array encoding. We denote array variables that represent sets in capital letters (e.g.,

X), and vectors of array variables in boldface (e.g., $\mathbf{X} = \langle X_1, \dots X_n \rangle$). To ease notation we overload predicates over sets to predicates over vectors of sets in a point-wise manner and write, e.g., $\mathsf{empty}(\mathbf{X})$ for $\bigwedge \mathsf{empty}(X_i)$, and $\mathbf{X} = \mathbf{Y} \cup \mathbf{Z}$ for $\bigwedge X_i = Y_i \cup Z_i$.

To each $\mathbf{SL}^*_{\mathsf{data}}$ interpretation $\mathcal{M}_{\text{SL}}$, of size N, we associate an equivalent first-order model $\mathcal{M}_{\text{SMT}}$ in the theory $\mathcal{T}_{\mathsf{array}} \oplus \mathcal{T}_{\mathsf{data}} \oplus \mathcal{T}_{\mathsf{loc}}$ as follows. $\mathcal{M}_{\text{SMT}}$ interprets each sort from $\mathcal{M}_{\text{SL}}$ as the same sort; each partial function $f \in \mathsf{Fld}$ in $\mathcal{M}_{\text{SL}}$ as an array f of the same sort; and the domain of each partial function $f \in \mathsf{Fld}$ in $\mathcal{M}_{\text{SL}}$ as a dedicated set variable X_f. The interpretation $f^{\mathcal{M}_{\text{SMT}}}$ of a field is an array mapping each $\ell \in \mathrm{dom}(f^{\mathcal{M}_{\text{SL}}})$ to $f^{\mathcal{M}_{\text{SL}}}(\ell)$, and to an arbitrary well-sorted value otherwise. The interpretation of $X_f^{\mathcal{M}_{\text{SMT}}}$ is an array representing the set $\mathrm{dom}(f^{\mathcal{M}_{\text{SL}}})$. The interpretation $\mathcal{M}_{\text{SMT}}$ also includes N dedicated location variables $x_1, \dots, x_N$, and a set of locations X interpreted so that $X = X_{\mathsf{n}} \cup X_{\mathsf{l}} \cup X_{\mathsf{r}}$ and $X \subseteq \{x_1, \dots, x_N\}$ holds. Other variables and constants are interpreted in $\mathcal{M}_{\text{SMT}}$ as they are in $\mathcal{M}_{\text{SL}}$.

The following SMT formula $\Delta^N_{\mathbf{SL}}$ defines $\mathbf{SL}^*_{\mathsf{data}}$ heap interpretations of size at most N.

$$\Delta^N_{\mathbf{SL}} \overset{\text{def}}{=} X = \bigcup_{f \in \mathsf{Fld}} X_f \wedge X \subseteq \{x_1, \dots, x_N\} \wedge \mathbf{null} \notin X \wedge \mathsf{empty}(X_{\mathsf{n}} \cap (X_{\mathsf{l}} \cup X_{\mathsf{r}}))$$

Formula $\Delta^N_{\mathbf{SL}}$ makes sure that the allocated heap size is at most N, that $\mathbf{null}$ is not allocated, and that no variable is treated as both a list and a tree location. In the following we always denote with $\mathbf{X} = \langle X_{\mathsf{n}}, X_{\mathsf{l}}, X_{\mathsf{r}}, X_{\mathsf{d}} \rangle$ the vector of dedicated set variables denoting field footprints.

SMT Translation. The encoding function T_N that translates basic $\mathbf{SL}^*_{\mathsf{data}}$ formulas to SMT is shown in Figure 4. We start without inductive predicates, following the approach from [19]. The function T_N takes an $\mathbf{SL}^*_{\mathsf{data}}$ formula F and translates F into an SMT formula $F' = \mathsf{T}_N(F)$ so that F' is satisfiable if and only if F is satisfiable in a model of size at most N. The translation relies on two auxiliary functions: $\mathsf{T}^b_N(F)$, that translates the Boolean structure of F recursively; and $\mathsf{T}^s_N(F, \mathbf{Y})$, that translates spatial formulas. Both functions take as input a formula (and a footprint $\mathbf{Y}$ to define) and return a triple $\langle A, B, Z \rangle$, where A and B together define the semantics of F and Z is the set of all fresh variables introduced by the translation. The encoding is straightforward, with the exception of negation. Let $\mathsf{T}^b_N(F) = \langle A, B, Z \rangle$. In order for our encoding to be correct, we make sure that the following properties hold.

Correctness: $\mathcal{M}^N_{\text{SL}} \models F$ iff $\mathcal{M}^N_{\text{SMT}} \models \exists Z \,.\, A \wedge B$;
Z-Existence: $\exists Z \,.\, B$ is valid; and
Z-Equivalence: $B(Z_1) \wedge B(Z_2) \Rightarrow A(Z_1) = A(Z_2)$ is valid.

The correctness property ensures that the encoding correctly encodes the $\mathbf{SL}^*_{\mathsf{data}}$ semantics: F is true in a heap interpretation of size N iff it is true in the corresponding SMT model. Z-Existence and Z-Equivalence make sure that the encoding can accommodate negation: the B part of the translation is a "definition"

$$
\begin{aligned}
\mathsf{T}^s_N(F_{\mathsf{loc}}, \mathbf{Y}) &= \langle F_{\mathsf{loc}}, \mathsf{empty}(\mathbf{Y}), \emptyset\rangle \\
\mathsf{T}^s_N(F_{\mathsf{data}}, \mathbf{Y}) &= \langle F_{\mathsf{data}}, \mathsf{empty}(\mathbf{Y}), \emptyset\rangle \\
\mathsf{T}^s_N(x \to_f y, \mathbf{Y}) &= \langle f(x) = y, Y_f = \{x\} \wedge \mathsf{empty}(\mathbf{Y} \setminus Y_f), \emptyset\rangle \\
\mathsf{T}^s_N(F_1 * F_2, \mathbf{Y}) &= \text{let } \mathbf{Y}_1, \mathbf{Y}_2 \text{ be fresh,} \\
&\quad \text{let } \langle A_1, B_1, Z_1\rangle = \mathsf{T}^s_N(F_1, \mathbf{Y}_1), \langle A_2, B_2, Z_2\rangle = \mathsf{T}^s_N(F_2, \mathbf{Y}_2) \text{ in} \\
&\quad \langle A_1 \wedge A_2 \wedge \mathsf{empty}(\mathbf{Y}_1 \cap \mathbf{Y}_2), B_1 \wedge B_2 \wedge \mathbf{Y} = \mathbf{Y}_1 \cup \mathbf{Y}_2, Z_1 \cup Z_2 \cup \mathbf{Y}_1 \cup \mathbf{Y}_2\rangle \\
\mathsf{T}^b_N(F) &= \text{let } \mathbf{Y} \text{ be fresh, } \langle A, B, Z\rangle = \mathsf{T}^s_N(F, \mathbf{Y}) \text{ in } \langle A \wedge \mathbf{X} = \mathbf{Y}, B, Z \cup \mathbf{Y}\rangle \\
\mathsf{T}^b_N(\neg F) &= \text{let } \langle A, B, Z\rangle = \mathsf{T}^b_N(F) \text{ in } \langle \neg A, B, Z\rangle \\
\mathsf{T}^b_N(F_1 \wedge F_2) &= \text{let } \langle A_1, B_1, Z_1\rangle = \mathsf{T}^b_N(F_1), \langle A_2, B_2, Z_2\rangle = \mathsf{T}^b_N(F_2) \text{ in} \\
&\quad \langle A_1 \wedge A_2, B_1 \wedge B_2, Z_1 \cup Z_2\rangle \\
\mathsf{T}^b_N(F_1 \vee F_2) &= \text{let } \langle A_1, B_1, Z_1\rangle = \mathsf{T}^b_N(F_1), \langle A_2, B_2, Z_2\rangle = \mathsf{T}^b_N(F_2) \text{ in} \\
&\quad \langle A_1 \vee A_2, B_1 \wedge B_2, Z_1 \cup Z_2\rangle \\
\mathsf{T}_N(F) &= \text{let } \mathsf{T}^b_N(F) = \langle A, B, Z\rangle \text{ in } A \wedge B \wedge \Delta^N_{\mathbf{SL}}
\end{aligned}
$$

Fig. 4: SMT encoding for the core fragment of $\mathbf{SL}_{\mathsf{data}}$ without inductive predicates.

of the fresh variables Z: variables Z can be assigned in each model to satisfy B, in a way that the A part cannot distinguish. These properties allow us to ensure correctness of the translation of negation $\neg F$. Assuming that $\mathsf{T}^b_N(F) = \langle A, B, Z\rangle$, we can derive the encoding of negation as

$$
\begin{aligned}
\mathcal{M}^N_{\mathrm{SL}} &\models \neg F && \text{iff} && \mathcal{M}^N_{\mathrm{SL}} \not\models F && \text{iff} \quad (1)\\
\mathcal{M}^N_{\mathrm{SMT}} &\not\models \exists Z.A \wedge B && \text{iff} && \mathcal{M}^N_{\mathrm{SMT}} \models \neg\exists Z.A \wedge B && \text{iff} \quad (2)\\
\mathcal{M}^N_{\mathrm{SMT}} &\models \forall Z.B \Rightarrow \neg A && \text{iff} && \mathcal{M}^N_{\mathrm{SMT}} \models \exists Z.B \wedge \neg A \ \ , && \quad (3)
\end{aligned}
$$

where the equivalence (3) follows from Z-existence and Z-equivalence.

Lists and Trees. The translation of an inductive predicate $\mathsf{T}^s_N(\mathsf{pred}(x, \mathbf{s}, \mathcal{P}), \mathbf{Y})$, with $\mathbf{s} = \langle s_1, \ldots, s_k\rangle$, for model sizes of at most N, introduces fresh binary predicates $r^Z_1, \ldots, r^Z_N$, and a fresh location set Z. These fresh predicates are meant to represent reachability in up to N steps within the set Z. Location set Z will represent all nodes reachable from x allocated within the predicate. Throughout the remainder of the section, we assume that x, $\mathbf{s}$, $\mathcal{P}$, $\mathbf{Y}$, Z, r^Z_i and N are fixed and in scope of all definitions. We also assume sets of fields F_{pred} and $F^{\mathsf{d}}_{\mathsf{pred}}$, defined as $F_{\mathsf{list}} = \{\mathsf{n}\}$ and $F^{\mathsf{d}}_{\mathsf{list}} = \{\mathsf{n}, \mathsf{d}\}$, or $F_{\mathsf{tree}} = \{\mathsf{l}, \mathsf{r}\}$ and $F^{\mathsf{d}}_{\mathsf{tree}} = \{\mathsf{l}, \mathsf{r}, \mathsf{d}\}$.

To fully define translation function T^s_N, we will define auxiliary helper formulas. To start with, we define the following functions for convenience: $\mathsf{isstop}(x) \overset{\text{def}}{=} x = \mathbf{null} \vee \bigvee_{s \in \mathbf{s}} x = s$, defining stop nodes; $S(x, y) \overset{\text{def}}{=} \bigvee_{f \in F_{\mathsf{pred}}} f[x] = y$, defining a successor node; and $\mathsf{defineY} \overset{\text{def}}{=} \bigwedge_{f \in F^{\mathsf{d}}_{\mathsf{pred}}} Y_f = Z \wedge \bigwedge_{f \in \mathsf{Fld} \setminus F^{\mathsf{d}}_{\mathsf{pred}}} Y_f = \emptyset$, defining pred-relevant elements of $\mathbf{Y}$ in terms of the footprint Z.

Although reachability is not expressible in first-order logic, since we are only interested in finite reachability with respect to the model elements $x_1, \dots, x_N$, we can define the reachability predicates r^Z_K, for $1 \leq K \leq N$, with the following formulas.

$$R_1 \overset{\text{def}}{=} \bigwedge_{1 \leq i,j \leq N} r^Z_1(x_i, x_j) \Leftrightarrow (x_i \in Z \wedge \neg \mathsf{isstop}(x_j) \wedge S(x_i, x_j))$$

$$R_K \overset{\text{def}}{=} \bigwedge_{1 \leq i,j \leq N} r^Z_K(x_i, x_j) \Leftrightarrow (r^Z_{K-1}(x_i, x_j) \vee \bigvee_{1 \leq k \leq n} (r^Z_{K-1}(x_i, x_k) \wedge r^Z_1(x_k, x_j))$$

$$\mathsf{reachability} \overset{\text{def}}{=} R_1 \wedge R_2 \wedge \cdots \wedge R_N$$

In addition, we define the function $r^Z_N(x, y, f) \overset{\text{def}}{=} f[x] = y \vee (f[x] \in Z \wedge r^Z_N(f[x_i], x_j))$ to denote that y is reachable from x through f as the first step. We can now define the formula footprint that asserts that the set Z (the footprint of pred) is defined as the set of locations reachable from x.

$$\begin{aligned}
&\mathsf{emptyZ} \overset{\text{def}}{=} \mathsf{isstop}(x) \vee (\textstyle\bigwedge_{1 \leq i \leq N} x \neq x_i) \\
&\mathsf{footprint} \overset{\text{def}}{=} Z \subseteq \{x_1, \dots, x_N\} \wedge (\mathsf{emptyZ} \Rightarrow Z = \emptyset) \wedge \\
&\qquad \wedge \Big(\neg \mathsf{emptyZ} \Rightarrow \textstyle\bigwedge_{1 \leq i \leq N} \big((x_i \in Z) \Leftrightarrow \big((x_i = x) \vee r^Z_N(x, x_i)\big)\big)\Big)
\end{aligned}$$

Next, we define the formula structure that ensures that the elements of the pred are part of an acyclic data structure, starting at x, with no sharing except for the potential **null** node.

$$\begin{aligned}
\mathsf{oneparent} \overset{\text{def}}{=} &\bigwedge_{1 \leq i \leq N} x_i \in Z \Rightarrow \bigwedge_{f \neq g \in F_{\mathsf{pred}}} (f[x_i] = g[x_i] \Rightarrow f[x_i] = \mathbf{null}) \\
\wedge &\bigwedge_{1 \leq j \leq N} x_j \in Z \wedge x_i \neq x_j \Rightarrow \bigwedge_{f,g \in F_{\mathsf{pred}}} (f[x_i] = g[x_j] \Rightarrow f[x_i] = \mathbf{null})
\end{aligned}$$

$$\mathsf{structure} \overset{\text{def}}{=} (\neg \mathsf{isstop}(x) \Rightarrow x \in Z) \wedge \mathsf{oneparent} \wedge \neg r^Z_N(x, x)$$

For ensuring stop node properties, we define a formula $\mathsf{stops}^{\mathsf{pred}}$, that asserts that stop nodes of pred are pairwise different, occur exactly once, are the only leaves of the structure, and, for trees, are ordered the same way as prescribed by the vector $\mathbf{s} = \langle s_1, \dots, s_k \rangle$.

$$\mathsf{stopseq} \overset{\text{def}}{=} (\mathsf{isstop}(x) \Rightarrow \bigwedge_{s \in \mathbf{s}} x = s) \wedge \bigwedge_{1 \leq i < j \leq k} s_i \neq s_j$$

$$\mathsf{stopsoccur} \overset{\text{def}}{=} \neg \mathsf{isstop}(x) \implies \bigwedge_{s \in \mathbf{s}} \bigvee_{1 \leq p \leq N} (x_p \in Z \wedge S(x_p, s))$$

$$\mathsf{stopleaves} \overset{\text{def}}{=} \bigwedge_{1 \leq i \leq N} \bigwedge_{f \in F_{\mathsf{pred}}} (x_i \in Z \wedge f[x_i] \notin Z) \Rightarrow \mathsf{isstop}(f[x_i])$$

$$\mathsf{fstop}(x_p, f, s) \overset{\text{def}}{=} f[x_p] = s \vee \bigvee_{1 \leq c \leq N} r^Z_N(x_p, x_c, f) \wedge x_c \in Z \wedge S(x_c, s)$$

$$\mathsf{ordered} \overset{\text{def}}{=} \bigwedge_{1 \leq i < k} \bigvee_{1 \leq p \leq N} x_p \in Z \wedge \mathsf{fstop}(x_p, \mathsf{l}, s_i) \wedge \mathsf{fstop}(x_p, \mathsf{r}, s_{i+1})$$

We combine the above constraints into $\mathsf{stops}^{\mathsf{list}} \overset{\text{def}}{=} \mathsf{stopsoccur} \wedge \mathsf{stopseq} \wedge \mathsf{stopleaves}$ and $\mathsf{stops}^{\mathsf{tree}} \overset{\text{def}}{=} \mathsf{stopsoccur} \wedge \mathsf{stopseq} \wedge \mathsf{stopleaves} \wedge \mathsf{ordered}$. Finally, we define the data formula that ensures that the data allocated in the predicate respects the given (unary and binary) data predicates.

$$\begin{aligned}
\mathsf{udata}(P) &\overset{\text{def}}{=} \mathsf{map}_{\Rightarrow}(Z, \mathsf{map}_P(\mathsf{d})) = \mathbf{K}(\top) \\
\mathsf{bdata}(f, P) &\overset{\text{def}}{=} \bigwedge_{1 \leq i,j \leq N} x_i, x_j \in Z \wedge r_N^Z(x_i, x_j, f) \Rightarrow P(x_i, x_j) \\
\mathsf{data} &\overset{\text{def}}{=} \bigwedge_{P \in \mathcal{P}} \mathsf{udata}(P) \wedge \bigwedge_{(f,P) \in \mathcal{P}} \mathsf{bdata}_N(f, P)
\end{aligned}$$

Putting all the auxiliary formulas together, we define the translation of inductive predicates $\mathsf{pred} \in \{\mathsf{list}, \mathsf{tree}\}$ to SMT as follows.

$$\begin{aligned}
\mathsf{T}_N^s(\mathsf{pred}(x, \mathbf{s}, \mathcal{P}), \mathbf{Y}) = {} & \text{let } r_1^Z, \ldots, r_N^Z, Z \text{ be fresh} \\
& \text{let } A = \mathsf{structure} \wedge \mathsf{stops}^{\mathsf{pred}} \wedge \mathsf{data} \\
& \text{let } B = \mathsf{reachability} \wedge \mathsf{footprint} \wedge \mathsf{defineY} \ \mathsf{in} \\
& \langle A, B, \{r_1^Z, \ldots, r_N^Z, Z\} \rangle
\end{aligned}$$

It is important to note that the formulas R_K only ensure that the predicates r_K^Z are fully defined on the set $\{x_1, \ldots, x_N\}$ and can be interpreted arbitrarily elsewhere. Nevertheless, this is sufficient for the translation to be correct. By inspection, it can be seen that the A part of the translation cannot distinguish two interpretations of r_K^Z that differ only outside of $\{x_1, \ldots, x_N\}$. This is crucial for the correctness of the encoding as it supports the Z-Equivalence property of the translation.

Theorem 3. *Let F be a $\mathbf{SL}^*_{\mathsf{data}}$ formula and N be the bound given by Theorem 2. Then F is $\mathbf{SL}^*_{\mathsf{data}}$-satisfiable if and only if the SMT translation $F' = \mathcal{T}_N(F)$ is satisfiable. Moreover, the translation F' is polynomial in the size of F.*

As $\mathcal{T}_{\mathsf{array}}$ is in NP, this yields an NP decision procedure for $\mathbf{SL}^*_{\mathsf{data}}$ if $\mathcal{T}_{\mathsf{data}}$ is in NP, matching the complexity result from Section 4.

6 Conclusion

We defined a new fragment of separation logic, $\mathbf{SL}^*_{\mathsf{data}}$, which supports lists, trees, and data constraints. $\mathbf{SL}^*_{\mathsf{data}}$ allows us to formalize common data structures such as max-heaps and binary search trees. Despite this expressiveness, satisfiability and entailment of $\mathbf{SL}^*_{\mathsf{data}}$ formulas are decidable in NP and coNP, respectively. This follows from the logic's small-model property: Every model of an $\mathbf{SL}^*_{\mathsf{data}}$ formula can be converted into a small model by removing unnecessary locations. We derived a bound that is linear in the number of variables and thus enables a polynomial encoding into SMT. An implementation, which remains future work, can be based on off-the-shelf SMT solvers. In addition, we plan to extend our approach to doubly-linked and nested data structures, as well as to abduction.

References

1. Bansal, K., Brochenin, R., Lozes, E.: Beyond shapes: Lists with ordered data. In: FOSSACS (2009)
2. Barrett, C.W., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability Modulo Theories. In: Handbook of Satisfiability. IOS Press (2009)
3. Berdine, J., Calcagno, C., O'Hearn, P.W.: A decidable fragment of separation logic. In: FSTTCS (2004)
4. Berdine, J., Calcagno, C., O'Hearn, P.W.: Symbolic execution with separation logic. In: APLAS. Springer (2005)
5. Brotherston, J., Fuhs, C., Pérez, J.A.N., Gorogiannis, N.: A decision procedure for satisfiability in separation logic with inductive predicates. In: CSL-LICS (2014)
6. Calcagno, C., Distefano, D., Dubreil, J., Gabi, D., Hooimeijer, P., Luca, M., O'Hearn, P., Papakonstantinou, I., Purbrick, J., Rodriguez, D.: Moving fast with software verification. In: NFM. pp. 3–11 (2015)
7. Calcagno, C., Yang, H., O'Hearn, P.: Computability and complexity results for a spatial assertion language for data structures. In: FSTTCS (2001)
8. Cook, B., Haase, C., Ouaknine, J., Parkinson, M., Worrell, J.: Tractable reasoning in a fragment of separation logic. In: CONCUR (2011)
9. Drăgoi, C., Enea, C., Sighireanu, M.: Local shape analysis for overlaid data structures. In: SAS (2013)
10. Iosif, R., Rogalewicz, A., Simacek, J.: The tree width of separation logic with recursive definitions. In: CADE (2013)
11. Itzhaky, S., Banerjee, A., Immerman, N., Nanevski, A., Sagiv, M.: Effectively-propositional reasoning about reachability in linked data structures. In: CAV (2013)
12. Lahiri, S., Qadeer, S.: Back to the future: revisiting precise program verification using SMT solvers. POPL (2008)
13. Le, Q.L., Makoto, T., Sun, J., Chin, W.N.: A decidable fragment in separation logic with inductive predicates and arithmetic. In: CAV (2017)
14. Madhusudan, P., Parlato, G., Qiu, X.: Decidable logics combining heap structures and data. In: POPL (2011)
15. McCarthy, J.: Towards a mathematical science of computation. In: IFIP Congress (1962)
16. de Moura, L., Bjørner, N.: Generalized, efficient array decision procedures. In: FMCAD. pp. 45–52 (2009)
17. de Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: TACAS (2008)
18. Navarro Pérez, J., Rybalchenko, A.: Separation logic modulo theories. In: APLAS (2013)
19. Piskac, R., Wies, T., Zufferey, D.: Automating separation logic using SMT. In: CAV (2013)
20. Piskac, R., Wies, T., Zufferey, D.: Automating separation logic with trees and data. In: CAV. Springer (2014)
21. Preiner, M., Niemetz, A., Biere, A.: Lemmas on demand for lambdas. In: DIFTS@FMCAD. CEUR Workshop Proceedings, vol. 1130. CEUR-WS.org (2013)
22. Qiu, X., Garg, P., Ştefănescu, A., Madhusudan, P.: Natural proofs for structure, data, and separation. In: PLDI (2013)
23. Reynolds, A., Iosif, R., King, T.: A decision procedure for separation logic in SMT. In: ATVA (2016)
24. Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: LICS (2002)
25. Totla, N., Wies, T.: Complete instantiation-based interpolation. JAR 57(1) (2016)