Deciding Entailments in Inductive Separation Logic with Tree Automata

Radu Iosif[1], Adam Rogalewicz[2], and Tomáš Vojnar[2]

[1] University Grenoble Alpes, CNRS, VERIMAG, Grenoble, France
[2] FIT, Brno University of Technology, IT4Innovations Centre of Excellence, Czech Republic

Abstract. Separation Logic (SL) with inductive definitions is a natural formalism for specifying complex recursive data structures, used in compositional verification of programs manipulating such structures. The key ingredient of any automated verification procedure based on SL is the decidability of the entailment problem. In this work, we reduce the entailment problem for a non-trivial subset of SL describing trees (and beyond) to the language inclusion of tree automata (TA). Our reduction provides tight complexity bounds for the problem and shows that entailment in our fragment is EXPTIME-complete. For practical purposes, we leverage from recent advances in automata theory, such as inclusion checking for non-deterministic TA avoiding explicit determinization. We implemented our method and present promising preliminary experimental results.

1 Introduction

Separation Logic (SL) [22] is a logical framework for describing recursive mutable data structures. The attractiveness of SL as a specification formalism comes from the possibility of writing higher-order *inductive definitions* that are natural for describing the most common recursive data structures, such as singly- or doubly-linked lists (SLLs/DLLs), trees, hash maps (lists of lists), and more complex variations thereof, such as nested and overlaid structures (e.g. lists with head and tail pointers, skip-lists, trees with linked leaves, etc.). In addition to being an appealing specification tool, SL is particularly suited for compositional reasoning about programs. Indeed, the principle of *local reasoning* allows one to verify different elements (functions, threads) of a program, operating on disjoint parts of the memory, and to combine the results a-posteriori, into succinct verification conditions.

However, the expressive power of SL comes at the price of undecidability [6]. To avoid this problem, most SL dialects used by various tools (e.g. SPACE INVADER [2], PREDATOR [9], or INFER [7]) use hard-coded predicates, describing SLLs and DLLs, for which entailments are, in general, tractable [8]. For graph structures of bounded tree width, a general decidability result was presented in [14]. Entailment in this fragment is EXPTIME-hard, as proven in [1].

In this paper, we present a novel decision procedure for a restriction of the decidable SL fragment from [14], describing recursive structures in which *all edges are local with respect to a spanning tree*. Examples of such structures include SLLs, DLLs, trees and trees with parent pointers, etc. For structures outside of this class (e.g. skip-lists or trees with linked leaves), our procedure is sound (namely, if the answer of the procedure is positive, then the entailment holds), but not complete (the answer might be negative and the entailment could still hold). In terms of program verification, such a lack of completeness in the entailment prover can lead to non-termination or false positives, but will not cause unsoundness (i.e. classify a buggy program as correct).

The method described in the paper belongs to the class of *automata-theoretic* decision techniques: We translate an entailment problem $\varphi \models \psi$ into a language inclusion problem $\mathcal{L}(A_\varphi) \subseteq \mathcal{L}(A_\psi)$ for tree automata (TA) A_φ and A_ψ that (roughly speaking) encode the sets of models of φ and ψ, respectively. Yet, a naïve translation of the inductive definitions of SL into TA encounters a *polymorphic representation* problem: the same set of structures can be defined in several different ways, and TA simply mirroring the definition will not report the entailment. For example, DLLs with selectors `next` and `prev` for the next and previous nodes, respectively, can be described by a forward unfolding of the inductive definition: $\mathtt{DLL}(head, prev, tail, next) \equiv \exists x.\ head \mapsto (x, prev) * \mathtt{DLL}(x, head, tail, next) \mid \mathbf{emp} \wedge head = tail \wedge prev = next$, as well as by a backward unfolding of the definition: $\mathtt{DLL}_{rev}(head, prev, tail, next) \equiv \exists x.\ tail \mapsto (next, x) * \mathtt{DLL}_{rev}(head, prev, x, tail) \mid \mathbf{emp} \wedge head = tail \wedge prev = next$. Also, one can define a DLL starting with a node in the middle and unfolding backward to the left of this node and forward to the right: $\mathtt{DLL}_{mid}(head, prev, tail, next) \equiv \exists x, y, z\,.\ \mathtt{DLL}(y, x, tail, next) * \mathtt{DLL}_{rev}(head, prev, z, x)$. The circular entailment: $\mathtt{DLL}(\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}) \models \mathtt{DLL}_{rev}(\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}) \models \mathtt{DLL}_{mid}(\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}) \models \mathtt{DLL}(\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d})$ holds, but a naïve structural translation to TA might not detect this fact. To bridge this gap, we define a closure operation on TA, called *canonical rotation*, which adds all possible representations of a given inductive definition, encoded as a tree automaton.

The translation from SL to TA provides also tight complexity bounds, showing that entailment in the local fragment of SL with inductive definitions is EXPTIME-complete. Moreover, we implemented our method using the VATA [17] tree automata library, which leverages from recent advances in non-deterministic language inclusion for TA [4], and obtained quite encouraging experimental results.

Related Work. Given the large body of literature on logics for describing mutable data structures, we need to restrict this section to the related work that focuses on SL [22]. The first (proof-theoretic) decidability result for SL on a restricted fragment defining only SLLs was reported in [3], which describe a co-NP algorithm. The full basic SL without recursive definitions, but with the magic wand operator was found to be undecidable when interpreted *in any memory model* [6]. A PTIME entailment procedure for SL with list predicates is given in [8]. Their method was extended to reason about nested and overlaid lists in [11]. More recently, entailments in an important SL fragment with hardcoded SLL/DLL predicates were reduced to Satisfiability Modulo Theories (SMT) problems, leveraging from recent advances in SMT technology [20,18]. The work reported in [10] deals with entailments between inductive SL formulae describing nested list structures. It uses a combination of graphs and TA to encode models of SL, but it does not deal with the problem of polymorphic representation. Recently, a decision procedure for entailments in a fragment of multi-sorted first-order logic with reachability, hard-coded trees and frame specifications, called GRIT (Graph Reachability and Inverted Trees) has been reported in [21]. Due to the restriction of the transitive closure to one function symbol (parent pointer), the expressive power of their logic, without data constraints, is strictly lower than ours (regular properties of trees cannot be encoded in GRIT). However, GRIT can be extended with data, which has not been, so far, considered for SL.

Closer to our work on SL with user-provided *inductive definitions* is the fragment used in the tool SLEEK, which implements a semi-algorithmic entailment check, based on unfoldings and unifications [19]. Along this line of work, the theorem prover CYCLIST builds entailment proofs using a sequent calculus. Neither SLEEK nor CYCLIST are complete for a given fragment of SL, and, moreover, these tools do not address the polymorphic representation problem.

Our previous work [14] gave a general decidability result for SL with inductive definitions interpreted over graph-like structures, under several necessary restrictions, based on a reduction from SL to Monadic Second Order Logic (MSOL) on graphs of bounded tree width. Decidability of MSOL on such graphs relies on a combinatorial reduction to MSOL on trees (see [12] for a proof of Courcelle's theorem). Altogether, using the method from [14] causes a blowup of several exponentials in the size of the input problem and is unlikely to produce an effective decision procedure.

The work [1] provides a rather complete picture of complexity for the entailment in various SL fragments with inductive definitions, including EXPTIME-hardness of the decidable fragment of [14], but provides no upper bound. The EXPTIME-completeness result in this paper provides an upper bound for a fragment of *local definitions*, and strengthens the EXPTIME-hard lower bound as well, i.e. it is showed that even the entailment between local definitions is EXPTIME-hard.

2 Definitions

The set of natural numbers is denoted by $\mathbb{N}$. If $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ and $\mathbf{y} = \langle y_1, \ldots, y_m \rangle$ are tuples, $\mathbf{x} \cdot \mathbf{y} = \langle x_1, \ldots, x_n, y_1, \ldots, y_m \rangle$ denotes their concatenation, $|\mathbf{x}| = n$ denotes the length of $\mathbf{x}$, and $(\mathbf{x})_i = x_i$ denotes the i-th element of $\mathbf{x}$. For a partial function $f : A \rightharpoonup B$, and $\perp \notin B$, we denote by $f(x) = \perp$ the fact that f is undefined at some point $x \in A$. The domain of f is denoted $dom(f) = \{x \in A \mid f(x) \neq \perp\}$, and the image of f is denoted as $img(f) = \{y \in B \mid \exists x \in A \ . \ f(x) = y\}$. By $f : A \rightharpoonup_{fin} B$, we denote any partial function whose domain is finite. Given two partial functions f, g defined on disjoint domains, i.e. $dom(f) \cap dom(g) = \emptyset$, we denote by $f \oplus g$ their union.

States. We consider $Var = \{x, y, z, \ldots\}$ to be a countably infinite set of *variables* and $\mathbf{nil} \in Var$ be a designated variable. Let Loc be a countably infinite set of locations and $null \in Loc$ be a designated location.

Definition 1. *A* state *is a pair $\langle s, h \rangle$ where $s : Var \rightharpoonup Loc$ is a partial function mapping pointer variables into locations such that $s(\mathbf{nil}) = null$, and $h : Loc \rightharpoonup_{fin} \mathbb{N} \rightharpoonup_{fin} Loc$ is a finite partial function such that (i) $null \notin dom(h)$ and (ii) for all $\ell \in dom(h)$ there exists $k \in \mathbb{N}$ such that $(h(\ell))(k) \neq \perp$.*

Given a state $S = \langle s, h \rangle$, s is called the *store* and h the *heap*. For any $l, l' \in Loc$, we write $\ell \xrightarrow{k}_S \ell'$ instead of $(h(\ell))(k) = \ell'$ for any $k \in \mathbb{N}$ called a *selector*. We call the triple $\ell \xrightarrow{k}_S \ell'$ an *edge* of S. When the S subscript is obvious from the context, we sometimes omit it. Let $Img(h) = \bigcup_{\ell \in Loc} img(h(\ell))$ be the set of locations which are destinations of some edge in h. A location $\ell \in Loc$ is said to be *allocated* in $\langle s,h \rangle$ if $\ell \in dom(h)$ (i.e. it is the source of an edge). The location is called *dangling* in $\langle s,h \rangle$ if $\ell \in [img(s) \cup Img(h)] \setminus dom(h)$, i.e. it is referenced by a store variable or reachable from an allocated location in the heap, but it is not allocated in the heap itself. The set $loc(S) = img(s) \cup dom(h) \cup Img(h)$ is the set of all locations either allocated or referenced in the state S.

For any two states $S_1 = \langle s_1, h_1 \rangle$ and $S_2 = \langle s_2, h_2 \rangle$ such that (i) s_1 and s_2 agree on the evaluation of common variables ($\forall x \in dom(s_1) \cap dom(s_2) \,.\, s_1(x) = s_2(x)$) and (ii) h_1 and h_2 have disjoint domains ($dom(h_1) \cap dom(h_2) = \emptyset$), we denote by $S_1 \uplus S_2 = \langle s_1 \cup s_2, h_1 \oplus h_2 \rangle$ the *disjoint union* of S_1 and S_2. The disjoint union is undefined if one of the above conditions does not hold.

Trees and Tree Automata. Let Σ be a countable alphabet and $\mathbb{N}^*$ be the set of sequences of natural numbers. Let $\varepsilon \in \mathbb{N}^*$ denote the empty sequence and $p.q$ denote the concatenation of two sequences $p, q \in \mathbb{N}^*$. We say that p is a *prefix* of q if $q = p.q'$ for some $q' \in \mathbb{N}^*$. A set $X \subseteq \mathbb{N}^*$ is *prefix-closed* iff $p \in X \Rightarrow q \in X$ for each prefix q of p.

A *tree* t over Σ is a finite partial function $t : \mathbb{N}^* \rightharpoonup_{fin} \Sigma$ such that $dom(t)$ is a finite prefix-closed subset of $\mathbb{N}^*$ and, for each $p \in dom(t)$ and $i \in \mathbb{N}$, we have $t(p.i) \neq \bot$ only if $t(p.j) \neq \bot$, for all $0 \leq j < i$. The sequences $p \in dom(t)$ are called *positions* in the following. Given two positions $p, q \in dom(t)$, we say that q is the i-th successor (child) of p if $q = p.i$, for some $i \in \mathbb{N}$. We denote by $\mathcal{D}(t) = \{-1, 0, \ldots, N\}$ the *direction alphabet* of t, where $N = \max\{i \in \mathbb{N} \mid \exists p \in \mathbb{N}^* \,.\, p.i \in dom(t)\}$, and we let $\mathcal{D}_+(t) = \mathcal{D}(t) \setminus \{-1\}$. By convention, we have $(p.i).(-1) = p$, for all $p \in \mathbb{N}^*$ and $i \in \mathcal{D}_+(t)$. Given a tree t and a position $p \in dom(t)$, we define the *arity* of the position p as $\#_t(p) = \max\{d \in \mathcal{D}_+(t) \mid p.d \in dom(t)\} + 1$.

A (finite, non-deterministic, bottom-up) *tree automaton* (abbreviated as TA in the following) is a quadruple $A = \langle Q, \Sigma, \Delta, F \rangle$, where Σ is a finite alphabet, Q is a finite set of *states*, $F \subseteq Q$ is a set of *final states*, Σ is an alphabet, and Δ is a set of *transition rules* of the form $\sigma(q_1, \ldots, q_n) \to q$, for $\sigma \in \Sigma$, and $q, q_1, \ldots, q_n \in Q$. Given a tree automaton $A = \langle Q, \Sigma, \Delta, F \rangle$, for each rule $\rho = (\sigma(q_1, \ldots, q_n) \to q)$, we define its size as $|\rho| = n + 1$. The size of the tree automaton is $|A| = \sum_{\rho \in \Delta} |\rho|$. A *run* of A over a tree $t : \mathbb{N}^* \rightharpoonup_{fin} \Sigma$ is a function $\pi : dom(t) \to Q$ such that, for each node $p \in dom(t)$, where $q = \pi(p)$, if $q_i = \pi(p.i)$ for $1 \leq i \leq n$, then Δ has a rule $(t(p))(q_1, \ldots, q_n) \to q$. We write $t \xRightarrow{\pi} q$ to denote that π is a run of A over t such that $\pi(\varepsilon) = q$. We use $t \Longrightarrow q$ to denote that $t \xRightarrow{\pi} q$ for some run π. The *language* of A is defined as $\mathcal{L}(A) = \{t \mid \exists q \in F,\ t \Longrightarrow q\}$.

2.1 Separation Logic

The syntax of *basic formulae* of Separation Logic (SL) is given below:

$$\begin{aligned}
&\alpha \ \in \ Var \setminus \{\mathbf{nil}\};\ x \ \in \ Var;\\
&\Pi ::= \alpha = x \mid \Pi_1 \wedge \Pi_2\\
&\Sigma \ ::= \mathbf{emp} \mid \alpha \mapsto (x_1, \ldots, x_n) \mid \Sigma_1 * \Sigma_2 \text{ , for some } n > 0\\
&\varphi \ ::= \Sigma \wedge \Pi \mid \exists x \,.\, \varphi
\end{aligned}$$

A formula of the form $\bigwedge_{i=1}^{n} \alpha_i = x_i$ defined by the Π nonterminal in the syntax above is said to be *pure*. The atomic proposition **emp**, or any formula of the form $\bigstar_{i=1}^{k} \alpha_i \mapsto$

$(x_{i,1},\ldots,x_{i,n_i})$, for some $k > 0$, is said to be *spatial*. A variable x is said to be *free* in φ if it does not occur under the scope of any existential quantifier. We denote by $FV(\varphi)$ the set of free variables. A variable $\alpha \in FV(\Sigma) \setminus \{\mathbf{nil}\}$ is said to be *allocated* (respectively, *referenced*) in a spatial formula Σ if it occurs on the left-hand (respectively, right-hand) side of a proposition $\alpha \mapsto (x_1,\ldots,x_n)$ of Σ.

In the following, we shall use two equality relations. The *syntactic equality*, denoted $\sigma \equiv \varsigma$, means that σ and ς are the same syntactic object (formula, variable, tuple of variables, etc.). On the other hand, by writing $x =_\Pi y$, for two variables $x,y \in Var$ and a pure formula Π, we mean that the equality of the values of x and y is implied by Π.

A system of *inductive definitions* (inductive system) $\mathcal{P}$ is a set of rules of the form

$$\left\{P_i(x_{i,1},\ldots,x_{i,n_i}) \equiv |_{j=1}^{m_i} R_{i,j}(x_{i,1},\ldots,x_{i,n_i})\right\}_{i=1}^{k} \quad (1)$$

where $\{P_1,\ldots,P_k\}$ is a set of *predicates*, $x_{i,1},\ldots,x_{i,n_i}$ are called *formal parameters*, and the formulae $R_{i,j}$ are called the *rules* of P_i. Each rule is of the form $R_{i,j}(\mathbf{x}) \equiv \exists \mathbf{z} . \Sigma * P_{i_1}(\mathbf{y}_1) * \ldots * P_{i_m}(\mathbf{y}_m) \ \wedge \ \Pi$, where $\mathbf{x} \cap \mathbf{z} = \emptyset$, and the following holds:

1. $\Sigma \not\equiv \mathbf{emp}$ is a non-empty spatial formula[1], called the *head* of $R_{i,j}$.
2. $P_{i_1}(\mathbf{y}_1),\ldots,P_{i_m}(\mathbf{y}_m)$ is a tuple of *predicate occurrences*, called the *tail* of $R_{i,j}$, where $|\mathbf{y}_j| = n_{i_j}$, for all $1 \leq j \leq m$.
3. Π is a pure formula, restricted such that, for all formal parameters $\beta \in \mathbf{x}$, we allow only equalities of the form $\alpha =_\Pi \beta$, where α is allocated in Σ.[2]
4. for all $1 \leq r,s \leq m$, if $x_{i,k} \in \mathbf{y}_r$, $x_{i,l} \in \mathbf{y}_s$, and $x_{i,k} =_\Pi x_{i,l}$, for some $1 \leq k,l \leq n_i$, then $r = s$; a formal parameter of a rule cannot be passed to two or more subsequent occurrences of predicates in that rule.[3]

The size of a rule R is denoted by $|R|$ and defined inductively as follows: $|\alpha = x| = 1$, $|\mathbf{emp}| = 1$, $|\alpha \mapsto (x_1,\ldots,x_n)| = n+1$, $|\varphi \bullet \psi| = |\varphi| + |\psi|$, $|\exists x . \varphi| = |\varphi| + 1$, and $|P(x_1,\ldots,x_n)| = n$. Here, $\alpha \in Var \setminus \{\mathbf{nil}\}$, $x,x_1,\ldots,x_n \in Var$, and $\bullet \in \{*,\wedge\}$. The size of an inductive system (1) is defined as $|\mathcal{P}| = \sum_{i=1}^{k}\sum_{j=1}^{m_i} |R_{i,j}|$. A *rooted system* $\langle \mathcal{P}, P_i \rangle$ is an inductive system $\mathcal{P}$ with a designated predicate $P_i \in \mathcal{P}$.

Example 1. To illustrate the use of inductive definitions (with the above restrictions), we first show how to define a predicate $\mathtt{DLL}(hd,p,tl,n)$ describing doubly-linked lists of length at least one. As depicted on the top of Fig. 1, the formal parameter hd points to the first allocated node of such a list, p to the node pointed to by the *prev* selector of hd, tl to the last node of the list (possibly equal to hd), and n to the node pointed to by the *next* selector from tl. This predicate can be defined as follows: $\mathtt{DLL}(hd,p,tl,n) \equiv hd \mapsto (n,p) \ \wedge \ hd = tl \mid \exists x.\ hd \mapsto (x,p) * \mathtt{DLL}(x,hd,tl,n)$.

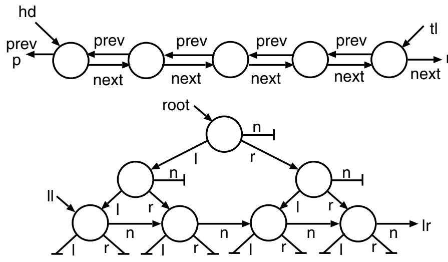

Fig. 1. Top: A DLL. Bottom: A TLL

[1] In practice, we allow frontier or root rules to have **emp**ty heads.

[2] This restriction can be lifted at the expense of an exponential blowup in the size of the TA.

[3] The restriction can be lifted by testing double allocation as in [14] (with an exponential cost).

Another example is the predicate $\mathtt{TLL}(r,ll,lr)$ describing binary trees with linked leaves whose root is pointed to by the formal parameter r, the left-most leaf is pointed to by ll, and the right-most leaf points to lr as shown in the bottom of Fig. 1: $\mathtt{TLL}(r,ll,lr) \equiv r \mapsto (\mathbf{nil},\mathbf{nil},lr) \,\wedge\, r = ll \mid \exists x,y,z.\ r \mapsto (x,y,\mathbf{nil}) * \mathtt{TLL}(x,ll,z) * \mathtt{TLL}(y,z,lr)$. ■

The semantics of SL is given by the *model relation* $\models$, defined inductively, on the structure of formulae, as follows:

$$\begin{array}{lcl}
S \models \mathbf{emp} & \iff & dom(h) = \emptyset \\
S \models \alpha \mapsto (x_1,\ldots,x_n) & \iff & s = \{(\alpha,\ell_0),(x_1,\ell_1),\ldots,(x_n,\ell_n)\} \text{ and} \\
 & & h = \{\langle \ell_0, \lambda i \,.\, \text{if } 1 \le i \le n \text{ then } \ell_i \text{ else } \bot\rangle\} \\
 & & \text{for some } \ell_0,\ell_1,\ldots,\ell_n \in Loc \\
S \models \varphi_1 * \varphi_2 & \iff & S_1 \models \varphi_1 \text{ and } S_2 \models \varphi_2 \text{ for some } S_1,S_2 : S_1 \uplus S_2 = S \\
S \models \exists x \,.\, \varphi & \iff & \langle s[x \leftarrow \ell], h\rangle \models \varphi \text{ for some } \ell \in Loc \\
S \models P_i(x_{i,1},\ldots,x_{i,n_i}) & \iff & S \models R_{i,j}(x_{i,1},\ldots,x_{i,n_i}), \text{ for some } 1 \le j \le m_i, \text{ in (1)}
\end{array}$$

The semantics of $=$ and $\wedge$ are classical for first order logic. Note that we adopt here the *strict semantics*, in which a points-to relation $\alpha \mapsto (x_1,\ldots,x_n)$ holds in a state consisting of a single cell pointed to by α that has exactly n outgoing edges $s(\alpha) \xrightarrow{k}_S s(x_k)$, $1 \le k \le n$, leading either towards the single allocated location $s(\alpha)$ (if $s(x_k) = s(\alpha)$) or towards dangling locations (if $s(x_k) \neq s(\alpha)$). The empty heap is specified by **emp**.

A state S is a model of a predicate P_i iff it is a model of one of its rules $R_{i,j}$. For a state S that is a model of $R_{i,j}$, the inductive definition of the semantics implies existence of a finite *unfolding tree*: this is a tree labeled with rules of the system in such a way that, whenever a node is labeled by a rule with a tail $P_{i_1}(\mathbf{y}_1),\ldots,P_{i_m}(\mathbf{y}_m)$, it has exactly m children such that the j-th child, for $1 \le j \le m$, is labeled with a rule of P_{i_j} (see the middle part of Fig. 2—a formal definition is given in [16].

Given an inductive system $\mathcal{P}$, predicates $P_i(x_1,\ldots,x_n)$ and $P_j(y_1,\ldots,y_n)$ of $\mathcal{P}$ with the same number of formal parameters n, and a tuple of variables $\mathbf{x}$ where $|\mathbf{x}| = n$, the *entailment problem* is defined as follows: $P_i(\mathbf{x}) \models_{\mathcal{P}} P_j(\mathbf{x}) \ :\ \forall S \,.\, S \models P_i(\mathbf{x}) \Rightarrow S \models P_j(\mathbf{x})$.

2.2 Connectivity, Spanning Trees and Local States

In this section, we define two conditions ensuring that entailments in the restricted SL fragment can be decided effectively. The notion of a *spanning tree* is central for these definitions. Informally, a state S has a spanning tree t if all allocated locations of S can be placed in t such that there is always an edge in S in between every two locations placed in a parent-child pair of positions (see Fig. 2 for two spanning trees).

Definition 2. *Given a state $S = \langle s,h\rangle$, a* spanning tree *of S is a bijective tree $t : \mathbb{N}^* \to dom(h)$ such that $\forall p \in dom(t) \forall d \in \mathcal{D}_+(t) \,.\, p.d \in dom(t) \Rightarrow \exists k \in \mathbb{N} \,.\, t(p) \xrightarrow{k}_S t(p.d)$.*

Given an inductive system $\mathcal{P}$, let $S = \langle s,h\rangle$ be a state and $P_i \in \mathcal{P}$ be an inductive definition such that $S \models P_i$. Our first restriction, called *connectivity* (Def. 3), ensures that the unfolding tree of the definition of P_i is also a spanning tree of S (cf. Fig. 2, middle). In other words, each location $\ell \in dom(h)$ is created by an atomic proposition of the form $\alpha \mapsto (x_1,\ldots,x_n)$ from the unfolding tree of the definition P_i, and, moreover, by Def. 2, there exists an edge $\ell \xrightarrow{k}_S \ell'$ for any parent-child pair of positions in this tree (cf. the `next` edges in Fig. 2).

For a basic quantifier-free SL formula $\varphi \equiv \Sigma \wedge \Pi$ and two variables $x, y \in FV(\varphi)$, we say that y is φ-*reachable* from x iff there is a sequence $x =_\Pi \alpha_0, \ldots, \alpha_m =_\Pi y$, for some $m \geq 0$, such that, for each $0 \leq i < m$, $\alpha_i \mapsto (\beta_{i,1}, \ldots, \beta_{i,p_i})$ is an atomic proposition in Σ, and $\beta_{i,s} =_\Pi \alpha_{i+1}$, for some $1 \leq s \leq p_i$. A variable $x \in FV(\Sigma)$ is called a *root* of Σ if every variable $y \in FV(\Sigma)$ is φ-reachable from x.

Definition 3. *Given a system $\mathcal{P} = \{P_i \equiv |_{j=1}^{m_i} R_{i,j}\}_{i=1}^n$ of inductive definitions, a rule $R_{i,j}(x_{i,1}, \ldots, x_{i,k}) \equiv \exists \mathbf{z} \,.\, \Sigma * P_{i_1}(\mathbf{y}_1) * \ldots * P_{i_m}(\mathbf{y}_m) \wedge \Pi$ of a predicate $P_i(x_{i,1}, \ldots, x_{i,k})$ is* connected *iff there exists a formal parameter $x_{i,\ell}$ of P_i, $1 \leq \ell \leq k$, such that (i) $x_{i,\ell}$ is a root of Σ and (ii) for each $j = 1, \ldots, m$, there exists $0 \leq s < |\mathbf{y}_j|$ such that $(\mathbf{y}_j)_s$ is $(\Sigma \wedge \Pi)$-reachable from $x_{i,\ell}$ and $x_{i_j,s}$ is a root of the head of each rule of P_{i_j}. The system $\mathcal{P}$ is said to be* connected *if all its rules are connected.*

For instance, the `DLL` and `TLL` systems from Ex. 1 are both connected. Our second restriction, called *locality*, ensures that every edge $\ell \xrightarrow{k}_S \ell'$, between allocated locations $\ell, \ell' \in dom(h)$, involves locations that are mapped to a parent-child pair of positions in some spanning tree of S.

Definition 4. *Let $S = \langle s, h \rangle$ be a state and $t : \mathbb{N}^* \rightharpoonup dom(h)$ be a spanning tree of S. An edge $\ell \xrightarrow{k}_S \ell'$ with $\ell, \ell' \in dom(h)$ is said to be* local *w.r.t. a spanning tree t iff there exist $p \in dom(t)$ and $d \in \mathcal{D}(t) \cup \{\varepsilon\}$ such that $t(p) = \ell$ and $t(p.d) = \ell'$. The tree t is a* local spanning tree *of S iff t is a spanning tree of S and S has only local edges w.r.t. t. The state S is* local *iff it has a local spanning tree.*

For instance, the `DLL` system of Ex. 1 is local, while the `TLL` system is not (e.g. the `n` edges between leaves cannot be mapped to parent-child pairs in the spanning tree that is obtained by taking the `l` and `r` edges of the TLL). In this paper, we address the locality problem by giving a sufficient condition (a syntactic check of the inductive system, prior to the generation of TA) able to decide the locality on all of the practical examples considered (Sec. 3.2). The decidability of locality of general inductive systems is an interesting open problem, considered for future research.

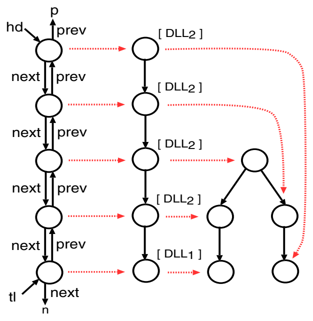

Fig. 2. Two spanning trees of a `DLL`. The middle one is an unfolding tree when labeled by $\mathtt{DLL}_1 \equiv hd \mapsto (n, p) \wedge hd = tl$ and $\mathtt{DLL}_2 \equiv \exists x.\ hd \mapsto (x, p) * \mathtt{DLL}(x, hd, tl, n)$.

Definition 5. *A system $\mathcal{P} = \{P_i(x_{i,1}, \ldots, x_{i,n_i})\}_{i=1}^k$ is said to be* local *if and only if each formal parameter $x_{i,j}$ of a predicate P_i is either (i) allocated in each rule of P_i and $(\mathbf{y})_j$ is referenced at each occurrence $P_i(\mathbf{y})$, or (ii) referenced in each rule of P_i and $(\mathbf{y})_j$ is allocated at each occurrence $P_i(\mathbf{y})$.*

This gives a sufficient (but not necessary) condition ensuring that any state S, such that $S \models P_i$, has a local spanning tree, if $\mathcal{P}$ is a connected local system. The condition is effective and easily implemented (see Sec. 3.2) by the translation from SL to TA.

3 From Separation Logic to Tree Automata

The first step of our entailment decision procedure is building a TA for a given inductive system. Roughly speaking, the TA we build recognizes unfolding trees of the inductive system. The alphabet of such a TA consists of small basic SL formulae describing the neighborhood of each allocated variable, together with a specification of the connections between each such formula and its parent and children in the unfolding tree. Each alphabet symbol in the TA is called a *tile*. Due to technical details related to the encoding of states as trees of SL formulae, the most space in this section is dedicated to the definition of tiles. Once the tile alphabet is defined, the states of the TA correspond naturally to the predicates of the inductive system, and the transition rules correspond to the rules of the system.

3.1 Tiles, Canonical Tiles, and Quasi-canonical Tiles

A *tile* is a tuple $T = \langle \varphi, \mathbf{x}_{-1}, \mathbf{x}_0, \ldots, \mathbf{x}_{d-1} \rangle$, for some $d \geq 0$, where φ is a basic SL formula, and each $\mathbf{x}_i$ is a tuple of pairwise distinct variables, called a *port*. We further assume that all ports contain only free variables from φ and that they are pairwise disjoint. The variables from $\mathbf{x}_{-1}$ are said to be *incoming*, the ones from $\mathbf{x}_0, \ldots, \mathbf{x}_{d-1}$ are said to be *outgoing*, and the ones from $\mathbf{par}(T) = FV(\varphi) \setminus (\mathbf{x}_{-1} \cup \ldots \cup \mathbf{x}_{d-1})$ are called *parameters*. The *arity* of a tile $T = \langle \varphi, \mathbf{x}_{-1}, \ldots, \mathbf{x}_{d-1} \rangle$ is the number of outgoing ports, denoted by $\#(T) = d$. We denote $\mathbf{form}(T) \equiv \varphi$ and $\mathbf{port}_i(T) \equiv \mathbf{x}_i$, for all $-1 \leq i < d$.

Given tiles $T_1 = \langle \varphi, \mathbf{x}_{-1}, \ldots, \mathbf{x}_{d-1} \rangle$ and $T_2 = \langle \phi, \mathbf{y}_{-1}, \ldots, \mathbf{y}_{e-1} \rangle$ such that $FV(\varphi) \cap FV(\phi) = \emptyset$, we define the *i-composition*, for some $0 \leq i < d$, such that $|\mathbf{x}_i| = |\mathbf{y}_{-1}|$: $T_1 \circledast_i T_2 = \langle \psi, \mathbf{x}_{-1}, \ldots \mathbf{x}_{i-1}, \mathbf{y}_0, \ldots, \mathbf{y}_{e-1}, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_{d-1} \rangle$ where $\psi \equiv \exists \mathbf{x}_i \exists \mathbf{y}_{-1} \,.\, \varphi * \phi \wedge \mathbf{x}_i = \mathbf{y}_{-1}$.[4] For a position $q \in \mathbb{N}^*$ and a tile T, we denote by $T^{\langle q \rangle}$ the tile obtained by renaming each variable x in the ports of T by $x^{\langle q \rangle}$. A tree t labeled with tiles corresponds to a tile defined inductively, for any $p \in dom(t)$, as: $\mathcal{T}(t,p) = t(p)^{\langle p \rangle} \circledast_0 \mathcal{T}(t,p.0) \circledast_1 \mathcal{T}(t,p.1) \ldots \circledast_{\#(p)-1} \mathcal{T}(t,p.(\#_t(p)-1))$. The SL formula $\Phi(t) \equiv \mathbf{form}(\mathcal{T}(t,\varepsilon))$ is said to be the *characteristic formula* of t.

Canonical Tiles. We first define a class of tiles that encode local states (Def. 4) with respect to the underlying tile-labeled spanning trees. We denote by $T = \langle (\exists z)\ z \mapsto (y_0, \ldots, y_{m-1}) \wedge \Pi, \mathbf{x}_{-1}, \ldots, \mathbf{x}_{d-1} \rangle$ a tile whose spatial formula is either (i) $\exists z \,.\, z \mapsto (y_0, \ldots, y_{m-1})$ or (ii) $z \mapsto (y_0, \ldots, y_{m-1})$ with $z \in \mathbf{par}(T)$. A tile $T = \langle (\exists z)\ z \mapsto (y_0, \ldots, y_{m-1}) \wedge \Pi,\ \mathbf{x}_{-1}, \ldots, \mathbf{x}_{d-1} \rangle$ is said to be *canonical* if each port $\mathbf{x}_i$ can be factorized as $\mathbf{x}_i^{fw} \cdot \mathbf{x}_i^{bw}$ (distinguishing *forward* links going from the root to the leaves and *backward* links going in the opposite direction, respectively) such that:

1. $\mathbf{x}_{-1}^{bw} \equiv \langle y_{h_0}, \ldots, y_{h_k} \rangle$, for some ordered sequence $0 \leq h_0 < \ldots < h_k < m$, i.e. the backward incoming tuple consists only of variables referenced by the unique allocated variable z, ordered by the corresponding selectors.
2. For all $0 \leq i < d$, $\mathbf{x}_i^{fw} \equiv \langle y_{j_0}, \ldots, y_{j_{k_i}} \rangle$, for some ordered sequence $0 \leq j_0 < \ldots < j_{k_i} < m$. As above, each forward outgoing tuple consists of variables referenced by the unique allocated variable z, ordered by the corresponding selectors.

[4] For two tuples $\mathbf{x} = \langle x_1, \ldots, x_k \rangle$ and $\mathbf{y} = \langle y_1, \ldots, y_k \rangle$, we write $\mathbf{x} = \mathbf{y}$ for $\bigwedge_{i=1}^{k} x_i = y_i$.

3. For all $0 \leq i, j < d$, if $(\mathbf{x}_i^{fw})_0 \equiv y_p$ and $(\mathbf{x}_j^{fw})_0 \equiv y_q$, for some $0 \leq p < q < m$ (i.e. $y_p \not\equiv y_q$), then $i < j$. This means that the forward outgoing tuples are ordered by the selectors referencing their first element.
4. $(\mathbf{x}_{-1}^{fw} \cup \mathbf{x}_0^{bw} \cup \ldots \cup \mathbf{x}_{d-1}^{bw}) \cap \{y_0, \ldots, y_{m-1}\} = \emptyset$ and $\Pi \equiv \mathbf{x}_{-1}^{fw} = z \wedge \bigwedge_{i=0}^{d-1} \mathbf{x}_i^{bw} = z$.[5]

We denote by $\mathbf{port}_i^{fw}(T)$ and $\mathbf{port}_i^{bw}(T)$ the tuples $\mathbf{x}_i^{fw}$ and $\mathbf{x}_i^{bw}$, respectively, for all $-1 \leq i < d$. The set of canonical tiles is denoted as $\mathcal{T}^c$.

Definition 6. *A tree* $t : \mathbb{N}^* \rightharpoonup_{fin} \mathcal{T}^c$ *is called* canonical *iff* $\#(t(p)) = \#_t(p)$ *for any* $p \in dom(t)$ *and, moreover, for each* $0 \leq i < \#_t(p)$, $|\mathbf{port}_i^{fw}(t(p))| = |\mathbf{port}_{-1}^{fw}(t(p.i))|$ *and* $|\mathbf{port}_i^{bw}(t(p))| = |\mathbf{port}_{-1}^{bw}(t(p.i))|$.

An important property of canonical trees is that each state that is a model of the characteristic formula $\Phi(t)$ of a canonical tree t (i.e. $S \models \Phi(t)$) can be uniquely described by a *local spanning tree* $u : dom(t) \rightarrow Loc$, which has the same structure as t, i.e. $dom(u) = dom(t)$. Intuitively, this is because each variable y_i, referenced in an atomic proposition $z \mapsto (y_0, \ldots, y_{m-1})$ in a canonical tile, is allocated only if it belongs to the backward part of the incoming port $\mathbf{x}_{-1}^{bw}$ or the forward part of some outgoing port $\mathbf{x}_i^{fw}$. In the first case, y_i is equal to the variable allocated by the parent tile, and in the second case, it is equal to the variable allocated by the i-th child. An immediate consequence is that any two models of $\Phi(t)$ differ only by a renaming of the allocated locations, i.e. they are identical up to isomorphism.

Example 2 (cont. of Ex. 1). To illustrate the notion of canonical trees, Fig. 3 shows two canonical trees for a given DLL. The tiles are depicted as big rectangles containing the appropriate basic formula as well as the input and output ports. In all ports, the first variable is in the forward and the second in the backward part.

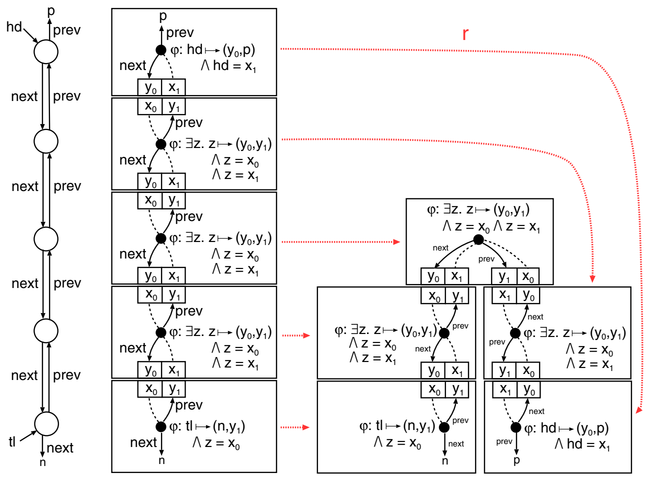

Fig. 3. The DLL from Fig. 1 with two of its canonical trees (related by a canonical rotation r)

■

Quasi-canonical tiles. We next define a class of tiles that encode non-local states in order to extend our decision procedure to handle entailments between non-local inductive systems. In addition to local edges between neighboring tiles, quasi-canonical tiles

[5] For a tuple $\mathbf{x} = \langle x_1, \ldots, x_k \rangle$, we write $\mathbf{x} = z$ for $\bigwedge_{i=1}^{k} x_i = z$.

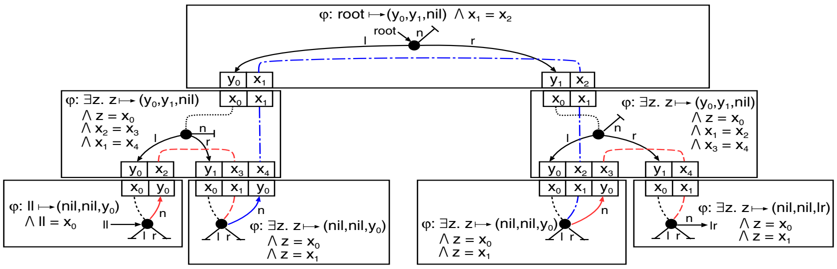

Fig. 4. A quasi-canonically tiled tree for the tree with linked leaves from Fig. 1

allow to define sequences of equalities between remote tiles. This extension is used to specify non-local edges within the state. A tile $T = \langle \varphi \wedge \Pi, \mathbf{x}_{-1}, \ldots, \mathbf{x}_{d-1} \rangle$ is said to be *quasi-canonical* if and only if each port $\mathbf{x}_i$ can be factorized as $\mathbf{x}_i^{fw} \cdot \mathbf{x}_i^{bw} \cdot \mathbf{x}_i^{eq}$, $\langle \varphi,\ \mathbf{x}_{-1}^{fw} \cdot \mathbf{x}_{-1}^{bw},\ \ldots,\ \mathbf{x}_{d-1}^{fw} \cdot \mathbf{x}_{d-1}^{bw} \rangle$ is a canonical tile, Π is pure formula, and:

1. for each $0 \leq i < |\mathbf{x}_{-1}^{eq}|$, either $(\mathbf{x}_{-1}^{eq})_i \in FV(\varphi)$ or $(\mathbf{x}_{-1}^{eq})_i =_\Pi (\mathbf{x}_k^{eq})_j$ for some unique indices $0 \leq k < d$ and $0 \leq j < |\mathbf{x}_k^{fw}|$.
2. for each $0 \leq k < d$ and each $0 \leq j < |\mathbf{x}_k^{eq}|$, either $(\mathbf{x}_k^{eq})_j \in FV(\varphi)$ or exactly one of the following holds: (i) $(\mathbf{x}_k^{eq})_j =_\Pi (\mathbf{x}_{-1}^{eq})_i$ for some unique index $0 \leq i < |\mathbf{x}_{-1}^{eq}|$ or (ii) $(\mathbf{x}_k^{eq})_j =_\Pi (\mathbf{x}_r^{eq})_s$ for some unique indices $0 \leq r < d$ and $0 \leq s < |\mathbf{x}_r^{eq}|$.
3. For any $x, y \in \bigcup_{i=-1}^{d-1} \mathbf{x}_i^{eq}$, we have $x =_\Pi y$ only in one of the cases above.

We denote $\mathbf{port}_i^{eq}(T) \equiv \mathbf{x}_i^{eq}$, for all $-1 \leq i < d$. The set of quasi-canonical tiles is denoted by $\mathcal{T}^{qc}$. The next definition of quasi-canonical trees extends Def. 6 to the case of quasi-canonical tiles.

Definition 7. *A tree* $t : \mathbb{N}^* \rightharpoonup_{fin} \mathcal{T}^{qc}$ *is* quasi-canonical *iff* $\#(t(p)) = \#_t(p)$ *for any* $p \in dom(t)$ *and, moreover, for each* $0 \leq i < \#_t(p)$, $|\mathbf{port}_i^{fw}(t(p))| = |\mathbf{port}_{-1}^{fw}(t(p.i))|$, $|\mathbf{port}_i^{bw}(t(p))| = |\mathbf{port}_{-1}^{bw}(t(p.i))|$, *and* $|\mathbf{port}_i^{eq}(t(p))| = |\mathbf{port}_{-1}^{eq}(t(p.i))|$.

Example 3 (cont. of Ex. 1). For an illustration of the notion of quasi-canonical trees, see Fig. 4, which shows a quasi-canonical tree for the TLL from Fig. 1. The figure uses the same notation as Fig. 3. In all the ports, the first variable is in the forward part, the backward part is empty, and the rest is the equality part. ■

3.2 Building a TA for an Inductive System

In the rest of this section, we consider that $\mathcal{P}$ is a connected inductive system (Def. 3)—our construction will detect and reject disconnected systems. Given a rooted system $\langle \mathcal{P}, P_r \rangle$, the first ingredient of our decision procedure for entailments is a procedure for building a TA that recognizes all unfolding trees of the inductive definition of P_r in the system $\mathcal{P}$. The first steps of the procedure implement a *specialization* of the rooted system with respect to a tuple $\overline{\alpha} = \langle \alpha_1, \ldots, \alpha_{n_r} \rangle$ of actual parameters for P_r, not used in $\mathcal{P}$. For space reasons, the specialization steps are described only informally here (for a detailed description of these steps, see [16]).

The first step is an elimination of existentially quantified variables that occur within equalities with formal parameters or allocated variables from all rules of $\mathcal{P}$. Second, each rule of $\mathcal{P}$ whose head consists of more than one atomic proposition $\alpha \mapsto (x_1,\ldots,x_n)$ is split into several new rules, containing exactly one such atomic proposition. At this point, any disconnected inductive system (Def. 3) passed to the procedure is detected and rejected. The final specialization step consists in propagating the actual parameters $\overline{\alpha}$ through the rules. A formal parameter $x_{i,k}$ of a rule $R_{i,j}(x_{i,1},\ldots,x_{i,n_i}) \equiv \exists \mathbf{z} \,.\, \Sigma * P_{i_1}(\mathbf{y}_1) * \ldots * P_{i_m}(\mathbf{y}_m) \wedge \Pi$ is *directly propagated* to some (unique) parameter of a predicate occurrence P_{i_j}, for some $1 \leq j \leq m$, if and only if $x_{i,k} \notin FV(\Sigma)$ and $x_{i,k} \equiv (\mathbf{y}_{i_j})_\ell$, for some $0 \leq \ell < |\mathbf{y}_{i_j}|$, i.e. $x_{i,k}$ is neither allocated nor pointed to by the head of the rule before being passed on to P_{i_j}. We denote direct propagation of parameters by the relation $x_{i,k} \rightsquigarrow x_{i_j,\ell}$ where $x_{i_j,\ell}$ is the formal parameter of P_{i_j} which is mapped to the occurrence of $(\mathbf{y}_{i_j})_\ell$. We say that $x_{i,k}$ is *propagated* to $x_{r,s}$ if $x_{i,k} \rightsquigarrow^* x_{r,s}$ where $\rightsquigarrow^*$ denotes the reflexive and transitive closure of the $\rightsquigarrow$ relation. Finally, we replace each variable y of $\mathcal{P}$ by the actual parameter α_j provided that $x_{r,j} \rightsquigarrow^* y$. It is not hard to show that the specialization procedure runs in time $O(|\mathcal{P}|)$, hence the size of the output system is increased by a linear factor only.

Example 4 (cont. of Ex. 1). As an example of specialization, let us consider the predicate DLL from Ex. 1, with parameters $\mathtt{DLL}(\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d})$. After the parameter elimination and renaming the newly created predicates, we have a call Q_1 (without parameters) of the following inductive system:

$$\begin{aligned} Q_1() &\equiv \mathtt{a} \mapsto (\mathtt{d},\mathtt{b}) \ \wedge\ \mathtt{a} = \mathtt{c} \mid \exists x.\ \mathtt{a} \mapsto (x,\mathtt{b}) * Q_2(x,\mathtt{a}) \\ Q_2(hd,p) &\equiv hd \mapsto (\mathtt{d},p) \ \wedge\ hd = \mathtt{c} \mid \exists x.\ hd \mapsto (x,p) * Q_2(x,hd) \end{aligned}$$

■

We are now ready to describe the construction of a TA for a specialized rooted system $\langle \mathcal{P}, P_r \rangle$. First, for each predicate $P_j(x_{j,1},\ldots,x_{j,n_j}) \in \mathcal{P}$, we compute several sets of parameters, called *signatures*: $\mathtt{sig}_j^{fw} = \{x_{j,k} \mid x_{j,k}$ is allocated in each rule of P_j, and $(\mathbf{y})_k$ is referenced in each occurrence $P_j(\mathbf{y})$ of $P_j\}$, $\mathtt{sig}_j^{bw} = \{x_{j,k} \mid x_{j,k}$ is referenced in each rule of P_j, and $(\mathbf{y})_k$ is allocated at each occurrence $P_j(\mathbf{y})$ of $P_j\}$, and, finally, $\mathtt{sig}_j^{eq} = \{x_{j,1},\ldots,x_{j,n_j}\} \setminus (\mathtt{sig}_j^{fw} \cup \mathtt{sig}_j^{bw})$. The signatures of an inductive system can be used to implement the *locality test* (Def. 5): the system $\mathcal{P} = \{P_1,\ldots,P_k\}$ is local if and only if $sig_i^{eq} = \emptyset$ for each $1 \leq i \leq k$.

Example 5 (cont. of Ex. 4). The signatures for the system in Ex. 4 are: $\mathtt{sig}_1^{fw} = \mathtt{sig}_1^{bw} = \mathtt{sig}_1^{eq} = \emptyset$ and $\mathtt{sig}_2^{fw} = \{hd\}, \mathtt{sig}_2^{bw} = \{p\}, \mathtt{sig}_2^{eq} = \emptyset$. The fact that, for each $i = 1,2$, we have $\mathtt{sig}_i^{eq} = \emptyset$ implies that the DLL system is local. ■

The procedure for building a TA from a rooted system $\langle \mathcal{P}, P_r \rangle$ with actual parameters $\overline{\alpha}$ is denoted as $\mathrm{SL2TA}(\mathcal{P}, P_r, \overline{\alpha})$ in the following. For each rule $R_{j,\ell}$ in the system, the SL2TA procedure creates a quasi-canonical tile whose incoming and outgoing ports $\mathbf{x}_i$ are factorized as $\mathbf{x}_i^{fw} \cdot \mathbf{x}_i^{bw} \cdot \mathbf{x}_i^{eq}$ according to the precomputed signatures $\mathtt{sig}_j^{fw}$, $\mathtt{sig}_j^{bw}$, and $\mathtt{sig}_j^{eq}$, respectively. The backward part of the input port $\mathbf{x}_{-1}^{bw}$ and the forward parts of the output ports $\{\mathbf{x}_i^{fw}\}_{i \geq 0}$ are sorted according to the order of incoming selector edges from the single points-to formula which constitutes the head of the rule. The output ports $\{\mathbf{x}_i\}_{i \geq 0}$ are sorted within the tile according to the order of the selector edges pointing to $(\mathbf{x}_i^{fw})_0$ for each $i \geq 0$. Finally, each predicate name P_i is associated with a state q_i, and for each inductive rule, the procedure creates a transition rule in the TA. The final state of the TA then corresponds to the root of the system (see Algorithm in [16]). The invariant used to prove the correctness of this construction is that whenever the TA reaches a state q_i it reads an unfolding tree whose root is labeled with a rule $R_{i,j}$ of the definition of a predicate P_i. The following lemma summarizes the TA construction:

Lemma 1. *Given a rooted system $\langle \mathcal{P}, P_r(x_{r,1}, \ldots, x_{r,n_r}) \rangle$ where $\mathcal{P} = \{P_i\}_{i=1}^k$ is a connected inductive system, $1 \leq r \leq k$, and $\overline{\alpha} = \langle \alpha_1, \ldots, \alpha_{n_i} \rangle$ is a tuple of variables not in $\mathcal{P}$, let $A = \mathrm{SL2TA}(\mathcal{P}, P_r, \overline{\alpha})$. Then, for every state S, we have $S \models P_r(\overline{\alpha})$ iff there exists $t \in \mathcal{L}(A)$ such that $S \models \Phi(t)$. Moreover, $|A| = O(|\mathcal{P}|)$.*

Example 6 (cont. of Ex. 5). For the specialized inductive system $\mathcal{P} = \{Q_1, Q_2\}$ from Ex. 4, we obtain the TA $A = \mathrm{SL2TA}(\mathcal{P}, Q_1, \langle \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d} \rangle) = \langle \Sigma, \{q_1, q_2\}, \Delta, \{q_1\} \rangle$ where Δ is shown above. ■

$$\Delta = \left\{ \begin{array}{ll} \langle \mathtt{a} \mapsto (\mathtt{d}, \mathtt{b}) \wedge \mathtt{a} = \mathtt{c}, \emptyset \rangle () \rightarrow q_1 \quad \langle \mathtt{a} \mapsto (x, \mathtt{b}), \emptyset, (x, \mathtt{a}) \rangle (q_2) & \rightarrow q_1 \\ \langle \exists hd'.hd' \mapsto (\mathtt{d}, p) \wedge hd = \mathtt{c} \wedge hd' = hd, (hd, p) \rangle () & \rightarrow q_2 \\ \langle \exists hd'.hd' \mapsto (x, p) \wedge hd' = hd, (hd, p), (x, hd) \rangle (q_2) & \rightarrow q_2 \end{array} \right\}$$

4 Rotation of Tree Automata

In this section we deal with polymorphic representations of states, i.e. situations when a state can be represented by different spanning trees, with different tilings. In this section we show that, for states with local spanning trees only (Def. 4), these trees are related by a *rotation* relation.

4.1 Rotation as a Transformation of TA

We start by defining rotation as a relation on trees. Intuitively, two trees t_1 and t_2 are related by a rotation whenever we can obtain t_2 from t_1 by picking a position $p \in dom(t_1)$ and making it the root of t_2, while maintaining in t_2 all edges from t_1 (Fig. 5).

Definition 8. *Given two trees $t_1, t_2 : \mathbb{N}^* \rightharpoonup_{fin} \Sigma$ and a bijective mapping $r : dom(t_1) \rightarrow dom(t_2)$, we say that t_2 is an r-*rotation *of t_1, denoted by $t_1 \sim_r t_2$ if and only if: $\forall p \in dom(t_1) \forall d \in \mathcal{D}_+(t_1) : p.d \in dom(t_1) \Rightarrow \exists e \in \mathcal{D}(t_2) . r(p.d) = r(p).e$. We write $t_1 \sim t_2$ if there exists a bijective mapping $r : dom(t_1) \rightarrow dom(t_2)$ such that $t_1 \sim_r t_2$.*

An example of a rotation r of a tree t_1 to a tree t_2 such that $r(\varepsilon) = 2$, $r(0) = \varepsilon$, $r(1) = 20$, $r(00) = 0$, and $r(01) = 1$ is shown in Fig. 5. Note that, e.g., for $p = \varepsilon \in dom(t_1)$ and $d = 0 \in \mathcal{D}_+(t_1)$, where $p.d = \varepsilon.0 \in dom(t_1)$, we get $e = -1 \in \mathcal{D}(t_2)$, and $r(\varepsilon.0) = 2.(-1) = \varepsilon$.

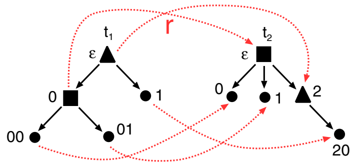

Fig. 5. An example of a rotation

In the rest of this section, we define rotation on canonical and quasi-canonical trees. These definitions are refinements of Def. 8. Namely, the change in the structure of the tree is mirrored by a change in the tile alphabet labeling the tree in order to preserve the state which is represented by the (quasi-)canonical tree.

A *substitution* is an injective partial function $\sigma : Var \rightharpoonup_{fin} Var$. Given a basic formula φ and a substitution σ, we denote by $\varphi[\sigma]$ the result of simultaneously replacing each variable x (not necessarily free) that occurs in φ by $\sigma(x)$. For instance, if $\sigma(x) = y$, $\sigma(y) = z$, and $\sigma(z) = t$, then $(\exists x, y \,.\, x \mapsto (y,z) \wedge z = x)[\sigma] \equiv \exists y, z \,.\, y \mapsto (z,t) \wedge t = y$.

Definition 9. *Given two canonical trees $t,u : \mathbb{N}^* \rightharpoonup_{fin} \mathcal{T}^c$ and a bijective mapping $r : dom(t) \rightarrow dom(u)$, we say that u is a* canonical rotation *of t, denoted $t \sim_r^c u$, if and only if $t \sim_r u$ and there exists a substitution $\sigma_p : Var \rightharpoonup_{fin} Var$ for each $p \in dom(t)$ such that* $\mathbf{form}(t(p))[\sigma_p] \equiv \mathbf{form}(u(r(p)))$ *and, for all $0 \leq i < \#_t(p)$, there exists $j \in \mathcal{D}(u)$ such that $r(p.i) = r(p).j$ and:*

$$\mathbf{port}_i^{fw}(t(p))[\sigma_p] \equiv \textit{if } j \geq 0 \textit{ then } \mathbf{port}_j^{fw}(u(r(p))) \textit{ else } \mathbf{port}_{-1}^{bw}(u(r(p)))$$
$$\mathbf{port}_i^{bw}(t(p))[\sigma_p] \equiv \textit{if } j \geq 0 \textit{ then } \mathbf{port}_j^{bw}(u(r(p))) \textit{ else } \mathbf{port}_{-1}^{fw}(u(r(p)))$$

We write $t \sim^c u$ if there exists a mapping r such that $t \sim_r^c u$.

Example 7 (cont. of Ex. 2). The notion of canonical rotation is illustrated by the canonical rotation r relating the two canonical trees of a DLL shown in Fig. 3. In its case, the variable substitutions are simply the identity in each node. Note, in particular, that when the tile 0 of the left tree (i.e., the second one from the top) gets rotated to the tile 1 of the right tree (i.e., the right successor of the root), the input and output ports get swapped and so do their forward and backward parts. ■

The following lemma is the key for proving completeness of our entailment checking for local inductive systems: if a (local) state is a model of the characteristic formulae of two different canonical trees, then these trees must be related by canonical rotation.

Lemma 2. *Let $t : \mathbb{N}^* \rightharpoonup_{fin} \mathcal{T}^c$ be a canonical tree and $S = \langle s,h \rangle$ be a state such that $S \models \Phi(t)$. Then, for any canonical tree $u : \mathbb{N}^* \rightharpoonup_{fin} \mathcal{T}^c$, we have $S \models \Phi(u)$ iff $t \sim^c u$.*

In the following, we extend the notion of rotation to quasi-canonical trees:

Definition 10. *Given two quasi-canonical trees $t,u : \mathbb{N}^* \rightharpoonup_{fin} \mathcal{T}^{qc}$ and a bijective mapping $r : dom(t) \rightarrow dom(u)$, we say that u is a* quasi-canonical rotation *of t, denoted $t \sim_r^{qc} u$, if and only if $t \sim_r^c u$ and $|\mathbf{port}_i^{eq}(t(p))| = |\mathbf{port}_j^{eq}(u(r(p)))|$ for all $p \in dom(t)$ and all $0 \leq i < \#_t(p)$, $-1 \leq j < \#_t(p)$ such that $r(p.i) = r(p).j$. We write $t \sim^{qc} u$ if there exists a mapping r such that $t \sim_r^{qc} u$.*

The increase in expressivity (i.e. the possibility of defining non-local edges) comes at the cost of a loss of completeness. The following lemma generalizes the necessity direction ($\Leftarrow$) of Lemma 2 for quasi-canonical tiles. Notice that the sufficiency ($\Rightarrow$) direction does not hold in general.

Lemma 3. *Let $t,u : \mathbb{N}^* \rightharpoonup_{fin} \mathcal{T}^{qc}$ be quasi-canonical trees such that $t \sim^{qc} u$. For all states S, if $S \models \Phi(t)$, then $S \models \Phi(u)$.*

Algorithm 1. Rotation Closure of Quasi-canonical TA

```
input a quasi-canonical TA $A = \langle Q, \Sigma, \Delta, F \rangle$
output a TA $A^r$ where:
$\mathcal{L}(A^r) = \{u : \mathbb{N}^* \rightharpoonup_{fin} \mathcal{T}^{qc} \mid \exists t \in \mathcal{L}(A) \,.\, u \sim^{qc} t\}$
function ROTATETA(A)
    $A^r \leftarrow A$
    assume $A^r \equiv \langle Q_r, \Sigma, \Delta_r, F_r \rangle$
    for all $\rho \in \Delta$ do
        assume $\rho \equiv T(q_0, \ldots, q_k) \to q$
        assume $T \equiv \langle \varphi, \mathbf{x}_{-1}, \mathbf{x}_0, \ldots, \mathbf{x}_k \rangle$
        if $\mathbf{x}_{-1} \neq \emptyset$ or $q \notin F$ then
            assume $\mathbf{x}_{-1} \equiv \mathbf{x}_{-1}^{fw} \cdot \mathbf{x}_{-1}^{bw} \cdot \mathbf{x}_{-1}^{eq}$
            if $\mathbf{x}_{-1}^{bw} \neq \emptyset$ then
                $Q^{rev} \leftarrow \{q^{rev} \mid q \in Q\}$
                $(Q_\rho, \Delta_\rho) \leftarrow (Q \cup Q^{rev} \cup \{q_\rho^f\}, \Delta)$
                $p \leftarrow$ POSITIONOF$(\mathbf{x}_{-1}^{bw}, \varphi)$
                $\mathbf{x}_{swap} \leftarrow \mathbf{x}_{-1}^{bw} \cdot \mathbf{x}_{-1}^{fw} \cdot \mathbf{x}_{-1}^{eq}$
                $T_{new} \leftarrow \langle \varphi, \langle\rangle, \mathbf{x}_0, \ldots, \mathbf{x}_p, \mathbf{x}_{swap}, \ldots, \mathbf{x}_k \rangle$
                $\Delta_\rho \leftarrow \Delta_\rho \cup \{T_{new}(q_0 \ldots q_p, q^{rev} \ldots q_k) \to q_\rho^f\}$
                $(\Delta_\rho, \_) \leftarrow$ ROTTR$(q, \Delta, \Delta_\rho, \emptyset, F)$
                $A_\rho \leftarrow \langle Q_\rho, \Sigma, \Delta_\rho, \{q_\rho^f\} \rangle$
                $A^r \leftarrow A^r \cup A_\delta$
    return $A^r$

function ROTTR$(q, \Delta, \Delta_{new}, \mathtt{V}, F)$
    $\mathtt{V} \leftarrow \mathtt{V} \cup \{q\}$
    for all $(U(s_0, \ldots, s_\ell) \to s) \in \Delta$ do
        for all $0 \leq j \leq \ell$ such that $s_j = q$ do
            assume $U = \langle \varphi, \mathbf{x}_{-1}, \mathbf{x}_0, \ldots, \mathbf{x}_j, \ldots, \mathbf{x}_\ell \rangle$
            assume $\mathbf{x}_j \equiv \mathbf{x}_j^{fw} \cdot \mathbf{x}_j^{bw} \cdot \mathbf{x}_j^{eq}$
            if $\mathbf{x}_{-1} = \emptyset$ and $s \in F$ then
                $\mathbf{x}_{swap} \leftarrow \mathbf{x}_j^{bw} \cdot \mathbf{x}_j^{fw} \cdot \mathbf{x}_j^{eq}$
                $U' \leftarrow \langle \varphi, \mathbf{x}_{swap}, \mathbf{x}_0, \ldots, \mathbf{x}_{j-1}, \mathbf{x}_{j+1}, \ldots, \mathbf{x}_\ell \rangle$
                $\Delta_{new} \leftarrow \Delta_{new} \cup \{U'(s_0 \ldots s_{j-1} \ldots s_\ell) \to q^{rev}\}$
            else
                $\mathbf{x}_{-1} \equiv \mathbf{x}_{-1}^{fw} \cdot \mathbf{x}_{-1}^{bw} \cdot \mathbf{x}_{-1}^{eq}$
                if $\mathbf{x}_{-1}^{bw} \neq \emptyset$ then
                    $\mathtt{ports} \leftarrow \langle \mathbf{x}_0, \ldots, \mathbf{x}_{j-1}, \mathbf{x}_{j+1}, \ldots, \mathbf{x}_\ell \rangle$
                    $\mathtt{states} \leftarrow (s_0, \ldots, s_{j-1}, s_{j+1}, \ldots, s_\ell)$
                    $\mathbf{x}_{swap} \leftarrow \mathbf{x}_{-1}^{bw} \cdot \mathbf{x}_{-1}^{fw} \cdot \mathbf{x}_{-1}^{eq}$
                    $p \leftarrow$ INSERTOUTPORT$(\mathbf{x}_{swap}, \mathtt{ports}, \varphi)$
                    INSERTLHSSTATE$(s^{rev}, \mathtt{states}, p)$
                    $U_{new} \leftarrow \langle \varphi, \mathbf{x}_j^{bw} \cdot \mathbf{x}_j^{fw} \cdot \mathbf{x}_j^{eq}, \mathtt{ports} \rangle$
                    $\Delta_{new} \leftarrow \Delta_{new} \cup \{U_{new}(\mathtt{states}) \to q^{rev}\}$
                    if $s \notin \mathtt{V}$ then
                        $(\Delta_{new}, \mathtt{V}) \leftarrow$ ROTTR$(s, \Delta, \Delta_{new}, \mathtt{V}, F)$
    return $(\Delta_{new}, \mathtt{V})$
```

4.2 Implementing Rotation as a Transformation of TA

This section describes the algorithm that produces the closure of a quasi-canonical tree automaton (i.e. a tree automaton recognizing quasi-canonical trees only) under rotation. The result is a TA that recognizes all trees $u : \mathbb{N}^* \rightharpoonup_{fin} \mathcal{T}^{qc}$ such that $t \sim^{qc} u$ for some tree t recognized by the input TA $A = \langle Q, \Sigma, \Delta, F \rangle$. Algorithm 1 (the ROTATETA procedure) describes the rotation closure whose result is a language-theoretic union of A and the TA A_ρ, one for each rule ρ of A. The idea behind the construction of $A_\rho = \langle Q_\rho, \Sigma, \Delta_\rho, \{q_\rho^f\} \rangle$ can be understood by considering a tree $t \in \mathcal{L}(A)$, a run $\pi : dom(t) \to Q$, and a position $p \in dom(t)$, which is labeled with the right hand side of the rule $\rho = T(q_1, \ldots, q_k) \to q$ of A. Then $\mathcal{L}(A_\rho)$ will contain the rotated tree u, i.e. $t \sim_r^{qc} u$, where the significant position p is mapped into the root of u by the rotation function r, i.e. $r(p) = \varepsilon$. To this end, we introduce a new rule $T_{new}(q_0, \ldots, q^{rev}, \ldots, q_k) \to q_\rho^f$ where the tile T_{new} mirrors the change in the structure of T at position p, and $q^{rev} \in Q_\rho$ is a fresh state corresponding to q. The construction of A_ρ continues recursively (procedure ROTTR), by considering every rule of A that has q on the left hand side: $U(q'_1, \ldots, q, \ldots, q'_\ell) \to s$. This rule is changed by swapping the roles of q and s and producing a rule $U_{new}(q'_1, \ldots, s^{rev}, \ldots q'_\ell) \to q^{rev}$ where U_{new} mirrors the change in the structure of U. Intuitively, the states $\{q^{rev} | q \in Q\}$ mark the unique path from the root of u to $r(\varepsilon) \in dom(u)$. The recursion stops when either (i) s is a final state of A, (ii) The tile U does not specify a forward edge in the direction marked by q, or (iii) all states of A have been visited.

Lemma 4. *Let $A = \langle Q, \mathcal{T}^{qc}, \Delta, F \rangle$ be a TA, and $A^r =$ ROTATETA(A) be the TA defining the rotation closure of A. Then $\mathcal{L}(A^r) = \{u \mid u : \mathbb{N}^* \rightharpoonup_{fin} \mathcal{T}^{qc},\ \exists t \in \mathcal{L}(A) \,.\, u \sim^{qc} t\}$. Moreover, $|A^r| = O(|A|^2)$.*

The main result of this paper is given by the following theorem. The entailment problem for inductive systems is reduced, in polynomial time, to a language inclusion problem for tree automata. The inclusion test is always sound (if the answer is yes, the entailment holds), and complete, if the right-hand side is a local system (Def. 4).

Theorem 1. *Let $\mathcal{P} = \left\{P_i \equiv |_{j=1}^{m_i} R_{i,j}\right\}_{i=1}^{k}$ be a connected inductive system. Then, for any two predicates $P_i(x_{i,1},\ldots,x_{i,n_i})$ and $P_j(x_{j,1},\ldots,x_{j,n_j})$ of $\mathcal{P}$ such that $n_i = n_j$, and for any tuple of variables $\overline{\alpha} = \langle \alpha_1,\ldots,\alpha_{n_i}\rangle$ not used in $\mathcal{P}$, the following holds for $A_1 = \mathrm{SL2TA}(\mathcal{P},P_i,\overline{\alpha})$ and $A_2 = \mathrm{SL2TA}(\mathcal{P},P_j,\overline{\alpha})$:*

- **(Soundness)** *$P_i(\overline{\alpha}) \models_{\mathcal{P}} P_j(\overline{\alpha})$ if $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2^r)$ and*
- **(Completness)** *$P_i(\overline{\alpha}) \models_{\mathcal{P}} P_j(\overline{\alpha})$ only if $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2^r)$ provided $\langle \mathcal{P},P_j\rangle$ is local.*

Example 8 (cont. of Ex. 6). When applied on the tree automaton A, the operation of rotation closure produces the tree automaton $A^r = \langle \Sigma, \{q_1,q_2,q_2^{rev},q_{fin}\},\Delta,\{q_1,q_{fin}\}\rangle$ where Δ is shown above. ■

$$\Delta = \left\{\begin{array}{ll}
\langle \mathtt{a} \mapsto (\mathtt{b},\mathtt{d}) \wedge \mathtt{a} = \mathtt{c}, \emptyset\rangle() \to q_1 \quad \langle \mathtt{a} \mapsto (x,\mathtt{b}),\emptyset,(x,\mathtt{a})\rangle(q_2) & \to q_1 \\
\langle \exists hd'.hd' \mapsto (\mathtt{d},p) \wedge hd = \mathtt{c} \wedge hd' = hd,(hd,p)\rangle() & \to q_2 \\
\langle \exists hd'.hd' \mapsto (x,p) \wedge hd' = hd,(hd,p),(x,hd)\rangle(q_2) & \to q_2 \\
\langle \exists hd'.hd' \mapsto (d,p) \wedge hd = c \wedge hd' = hd,\emptyset,(p,hd)\rangle(q_2^{rev}) & \to q_{fin} \\
\langle \mathtt{a} \mapsto (x,\mathtt{b}),(\mathtt{a},x)\rangle() & \to q_2^{rev} \\
\langle \exists hd'.hd' \mapsto (x,p) \wedge hd' = hd,(hd,x),(p,hd)\rangle(q_2^{rev}) & \to q_2^{rev} \\
\langle \exists hd'.hd' \mapsto (x,p) \wedge hd' = hd,\emptyset,(x,hd),(p,hd)\rangle(q_2,q_2^{rev}) & \to q_{fin}
\end{array}\right\}$$

5 Complexity

In this section, we provide tight complexity bounds for the entailment problem in the fragment of SL with inductive definitions under consideration, i.e., with the *connectivity* and *locality* restrictions. The first result shows the need for *connectivity* within the system: allowing disconnected rules leads to undecidability of the entailment problem. As a remark, the general undecidability of entailments for SL with inductive definitions has already been proven in [1]. Our proof stresses the fact that undecidability occurs due the lack of connectivity within some rules.

Theorem 2. *Entailment is undecidable for inductive systems with disconnected rules.*

The second result of this section provides tight complexity bounds for the entailment problem for local connected systems. We must point out that EXPTIME-hardness of entailments in the fragment of [14] was already proved in [1]. The result below is stronger since the fragment under consideration is a restriction of the fragment from [14] obtained by applying the locality condition.

Theorem 3. *Entailment is EXPTIME-complete for local connected inductive systems.*

6 Experiments

We implemented a prototype tool called SLIDE (Separation Logic with Inductive DEfinitions) [15] that takes as input two rooted systems $\langle \mathcal{P}_{lhs},P_{lhs}\rangle$ and $\langle \mathcal{P}_{rhs},P_{rhs}\rangle$ and tests the validity of the entailment $P_{lhs} \models_{\mathcal{P}_{lhs}\cup\mathcal{P}_{rhs}} P_{rhs}$. Table 1 lists the entailment queries on which we tried out our tool; all examples are public and available on the web [15]. The upper part of the table contains local systems, whereas the bottom part contains

Table 1. Experimental results. The upper table contains local systems, while the lower table non-local ones. Sizes of initial TA (col. 3,4) and rotated TA (col. 5) are in numbers of states/transitions.

Entailment $LHS \models RHS$	Answer	$\lvert A_{lhs}\rvert$	$\lvert A_{rhs}\rvert$	$\lvert A^r_{rhs}\rvert$
$\mathtt{DLL}(a,\mathbf{nil},c,\mathbf{nil}) \models \mathtt{DLL}_{rev}(a,\mathbf{nil},c,\mathbf{nil})$	True	2/4	2/4	5/8
$\mathtt{DLL}_{rev}(a,\mathbf{nil},c,\mathbf{nil}) \models \mathtt{DLL}_{mid}(a,\mathbf{nil},c,\mathbf{nil})$	True	2/4	4/8	12/18
$\mathtt{DLL}_{mid}(a,\mathbf{nil},c,\mathbf{nil}) \models \mathtt{DLL}(a,\mathbf{nil},c,\mathbf{nil})$	True	4/8	2/4	5/8
$\exists x,n,b.\ x \mapsto (n,b) * \mathtt{DLL}_{rev}(a,\mathbf{nil},b,x) * \mathtt{DLL}(n,x,c,\mathbf{nil}) \models \mathtt{DLL}(a,\mathbf{nil},c,\mathbf{nil})$	True	3/5	2/4	5/8
$\mathtt{DLL}(a,\mathbf{nil},c,\mathbf{nil}) \models \exists x,n,b.\ x \mapsto (n,b) * \mathtt{DLL}_{rev}(a,\mathbf{nil},b,x) * \mathtt{DLL}(n,x,c,\mathbf{nil})$	False	2/4	3/5	9/13
$\exists y,a.\ x \mapsto (y,\mathbf{nil}) * y \mapsto (a,x) * \mathtt{DLL}(a,y,c,\mathbf{nil}) \models \mathtt{DLL}(x,\mathbf{nil},c,\mathbf{nil})$	True	3/4	2/4	5/8
$\mathtt{DLL}(x,\mathbf{nil},c,\mathbf{nil}) \models \exists y,a.\ x \mapsto (\mathbf{nil},y) * y \mapsto (a,x) * \mathtt{DLL}(a,y,c,\mathbf{nil})$	False	2/4	3/4	8/10
$\exists x,b.\mathtt{DLL}(x,b,c,\mathbf{nil}) * \mathtt{DLL}_{rev}(a,\mathbf{nil},b,x) \models \mathtt{DLL}(a,\mathbf{nil},c,\mathbf{nil})$	True	3/6	2/4	5/8
$\mathtt{DLL}(a,\mathbf{nil},c,\mathbf{nil}) \models \mathtt{DLL}_{0+}(a,\mathbf{nil},c,\mathbf{nil})$	True	2/4	2/4	5/8
$\mathtt{TREE}_{pp}(a,\mathbf{nil}) \models \mathtt{TREE}^{rev}_{pp}(a,\mathbf{nil})$	True	2/4	3/8	6/11
$\mathtt{TREE}^{rev}_{pp}(a,\mathbf{nil}) \models \mathtt{TREE}_{pp}(a,\mathbf{nil})$	True	3/8	2/4	5/10
$\mathtt{TLL}_{pp}(a,\mathbf{nil},c,\mathbf{nil}) \models \mathtt{TLL}^{rev}_{pp}(a,\mathbf{nil},c,\mathbf{nil})$	True	4/8	4/8	13/22
$\mathtt{TLL}^{rev}_{pp}(a,\mathbf{nil},c,\mathbf{nil}) \models \mathtt{TLL}_{pp}(a,\mathbf{nil},c,\mathbf{nil})$	True	4/8	4/8	13/22
$\exists l,r,z.\ a \mapsto (l,r,\mathbf{nil},\mathbf{nil}) * \mathtt{TLL}(l,c,z) * \mathtt{TLL}(r,z,\mathbf{nil}) \models \mathtt{TLL}(a,c,\mathbf{nil})$	True	4/7	4/8	13/22
$\mathtt{TLL}(a,c,\mathbf{nil}) \models \exists l,r,z.\ a \mapsto (l,r,\mathbf{nil},\mathbf{nil}) * \mathtt{TLL}(l,c,z) * \mathtt{TLL}(r,z,\mathbf{nil})$	False	4/8	4/7	13/21

non-local systems. Apart from the DLL and TLL predicates from Sect. 2.1, the considered entailment queries contain the following predicates: $\mathtt{DLL}_{rev}$ (resp. $\mathtt{DLL}_{mid}$) that encodes a DLL from the end (resp. middle), $\mathtt{DLL}_{0+}$ that encodes a possibly empty DLL, $\mathtt{TREE}_{pp}$ encoding trees with parent pointers, $\mathtt{TREE}^{rev}_{pp}$ that encodes trees with parent pointers defined starting with an arbitrary leaf, $\mathtt{TLL}_{pp}$ encoding TLLs with parent pointers, and $\mathtt{TLL}^{rev}_{pp}$ which encodes TLLs with parent pointers starting from their leftmost leaf. Columns $|A_{lhs}|$, $|A_{rhs}|$, and $|A^r_{rhs}|$ of Table 1 provide information about the number of states/transitions of the respective TA. The tool answered all queries correctly (despite the incompleteness for non-local systems), and the running times were all under 1 sec. on a standard PC (Intel Core2 CPU, 3GHz, 4GB RAM).

We also compared the SLIDE tool to the CYCLIST [5] theorem prover on the examples from the CYCLIST distribution [13]. Both tools run in less than 1 sec. on the examples from their common fragment of SL. CYCLIST does not handle examples where rotation is needed, while SLIDE fails on examples that generate an unbounded number of dangling pointers and are outside of the decidable fragment of [14].

7 Conclusion

We presented a novel decision procedure for the entailment problem in a non-trivial subset of SL with inductive predicates, which deals with the problem that the same recursive structure may be represented differently, when viewed from different entry points. To this end, we use a special operation, which closes a given TA representation w.r.t. the rotations of its spanning trees. Our procedure is sound and complete for inductive systems with local edges. We have implemented a prototype tool which we tested through a number of non-trivial experiments, with encouraging results.

Acknowledgment. This work was supported by the Czech Science Foundation under the project 14-11384S, the EU/Czech IT4Innovations Centre of Excellence project CZ.1.05/1.1.00/02.0070, and the internal BUT projects FIT-S-12-1 and FIT-S-14-2486.

References

1. Antonopoulos, T., Gorogiannis, N., Haase, C., Kanovich, M., Ouaknine, J.: Foundations for decision problems in separation logic with general inductive predicates. In: Muscholl, A. (ed.) FOSSACS 2014. LNCS, vol. 8412, pp. 411–425. Springer, Heidelberg (2014)
2. Berdine, J., Calcagno, C., Cook, B., Distefano, D., O'Hearn, P.W., Wies, T., Yang, H.: Shape analysis for composite data structures. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 178–192. Springer, Heidelberg (2007)
3. Berdine, J., Calcagno, C., O'Hearn, P.W.: A decidable fragment of separation logic. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004. LNCS, vol. 3328, pp. 97–109. Springer, Heidelberg (2004)
4. Bouajjani, A., Habermehl, P., Holík, L., Touili, T., Vojnar, T.: Antichain-based universality and inclusion testing over nondeterministic finite tree automata. In: Ibarra, O.H., Ravikumar, B. (eds.) CIAA 2008. LNCS, vol. 5148, pp. 57–67. Springer, Heidelberg (2008)
5. Brotherston, J., Gorogiannis, N., Petersen, R.L.: A generic cyclic theorem prover. In: Jhala, R., Igarashi, A. (eds.) APLAS 2012. LNCS, vol. 7705, pp. 350–367. Springer, Heidelberg (2012)
6. Brotherston, J., Kanovich, M.: Undecidability of propositional separation logic and its neighbours. In: Proceedings of the 2010 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, pp. 130–139 (2010)
7. Calcagno, C., Distefano, D.: Infer: An automatic program verifier for memory safety of C programs. In: Bobaru, M., Havelund, K., Holzmann, G.J., Joshi, R. (eds.) NFM 2011. LNCS, vol. 6617, pp. 459–465. Springer, Heidelberg (2011)
8. Cook, B., Haase, C., Ouaknine, J., Parkinson, M., Worrell, J.: Tractable reasoning in a fragment of separation logic. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 235–249. Springer, Heidelberg (2011)
9. Dudka, K., Peringer, P., Vojnar, T.: Predator: A practical tool for checking manipulation of dynamic data structures using separation logic. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 372–378. Springer, Heidelberg (2011)
10. Enea, C., Lengál, O., Sighireanu, M., Vojnar, T.: Compositional Entailment Checking for a Fragment of Separation Logic. Technical Report FIT-TR-2014-01, FIT, Brno University of Technology (2014)
11. Enea, C., Saveluc, V., Sighireanu, M.: Compositional invariant checking for overlaid and nested linked lists. In: Felleisen, M., Gardner, P. (eds.) ESOP 2013. LNCS, vol. 7792, pp. 129–148. Springer, Heidelberg (2013)
12. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer-Verlag New York, Inc. (2006)
13. Gorogiannis, N.: Cyclist: a cyclic theorem prover framework, `https://github.com/ngorogiannis/cyclist/`
14. Iosif, R., Rogalewicz, A., Simacek, J.: The tree width of separation logic with recursive definitions. In: Bonacina, M.P. (ed.) CADE 2013. LNCS, vol. 7898, pp. 21–38. Springer, Heidelberg (2013)
15. Iosif, R., Rogalewicz, A., Vojnar, T.: Slide: Separation logic with inductive definitions, `http://www.fit.vutbr.cz/research/groups/verifit/tools/slide/`
16. Iosif, R., Rogalewicz, A., Vojnar, T.: Deciding entailments in inductive separation logic with tree automata. CoRR, abs/1402.2127 (2014)
17. Lengal, O., Simacek, J., Vojnar, T.: Vata: a tree automata library, `http://www.fit.vutbr.cz/research/groups/verifit/tools/libvata/`

18. Navarro Pérez, J.A., Rybalchenko, A.: Separation logic modulo theories. In: Shan, C.-C. (ed.) APLAS 2013. LNCS, vol. 8301, pp. 90–106. Springer, Heidelberg (2013)
19. Nguyen, H.H., Chin, W.-N.: Enhancing program verification with lemmas. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 355–369. Springer, Heidelberg (2008)
20. Piskac, R., Wies, T., Zufferey, D.: Automating separation logic using SMT. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 773–789. Springer, Heidelberg (2013)
21. Piskac, R., Wies, T., Zufferey, D.: Automating separation logic with trees and data. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 711–728. Springer, Heidelberg (2014)
22. Reynolds, J.: Separation Logic: A Logic for Shared Mutable Data Structures. In: Proc. of LICS 2002. IEEE CS Press (2002)