The Tree Width of Separation Logic with Recursive Definitions

Radu Iosif[1], Adam Rogalewicz[2], and Jiri Simacek[2]

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

Abstract. Separation Logic is a widely used formalism for describing dynamically allocated linked data structures, such as lists, trees, etc. The decidability status of various fragments of the logic constitutes a long standing open problem. Current results report on techniques to decide satisfiability and validity of entailments for Separation Logic(s) over lists (possibly with data). In this paper we establish a more general decidability result. We prove that any Separation Logic formula using rather general recursively defined predicates is decidable for satisfiability, and moreover, entailments between such formulae are decidable for validity. These predicates are general enough to define (doubly-) linked lists, trees, and structures more general than trees, such as trees whose leaves are chained in a list. The decidability proofs are by reduction to decidability of Monadic Second Order Logic on graphs with bounded tree width.

1 Introduction

Separation Logic (SL) [17] is a general framework for describing dynamically allocated mutable data structures generated by programs that use pointers and low-level memory allocation primitives. The logics in this framework are used by an important number of academic (SPACE INVADER [1], SLEEK [16] and PREDATOR [9]), as well as industrial-scale (INFER [7]) tools for program verification and certification. These logics are used both externally, as property specification languages, or internally, as e.g., abstract domains for computing invariants, or for proving verification conditions. The main advantage of using SL when dealing with heap manipulating programs, is the ability to provide compositional proofs, based on the principle of *local reasoning* i.e., analyzing different sections (e.g., functions, threads, etc.) of the program, that work on disjoint parts of the global heap, and combining the analysis results a-posteriori.

The basic language of SL consists of two kinds of atomic propositions describing either (i) the empty heap, or (ii) a heap consisting of an allocated cell, connected via a separating conjunction primitive. Hence a basic SL formula can describe only a heap whose size is bounded by the size of the formula. The ability of describing unbounded data structures is provided by the use of *recursive definitions*. Figure 1 gives several common examples of recursive data structures definable in this framework.

The main difficulty that arises when using Separation Logic with Recursive Definitions (SLRD) to reason automatically about programs is that the logic, due to its expressiveness, does not have very nice decidability properties. Most dialects used in practice restrict the language (e.g., no quantifier alternation, the negation is used in a

$$\begin{aligned}
list(hd,tl) &::= emp \wedge hd = tl \\
&\mid\ \exists x.\ hd \mapsto x * list(x,tl) \\
dll(hd,p,tl) &::= emp \wedge hd = tl \\
&\mid\ \exists x.\ hd \mapsto (x,p) * dll(x,hd,tl) \\
tree(root) &::= emp \wedge root = nil \\
&\mid\ \exists l,r.\ root \mapsto (l,r) * tree(l) * tree(r) \\
tll(x,ll,lr) &::= x \mapsto (nil,nil,lr) \wedge x = ll \\
&\mid\ \exists l,r,z.\ x \mapsto (l,r,nil) * tll(l,ll,z) \\
&\quad * tll(r,z,lr)
\end{aligned}$$

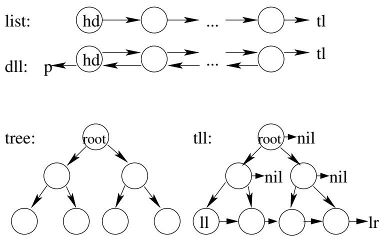

Fig. 1. Examples of recursive data structures definable in SLRD.

very restricted ways, etc.) and the class of models over which the logic is interpreted (typically singly-linked lists, and slight variations thereof). In the same way, we apply several natural restrictions on the syntax of the recursive definitions, and define the fragment SLRD$_{btw}$, which guarantees that all models of a formula in the fragment have *bounded tree width*. Indeed, this ensures that the satisfiability and entailment problems in this fragment are decidable *without any restrictions on the type of the recursive data structures considered*.

In general, the techniques used in proving decidability of Separation Logic are either proof-based ([16, 2]), or model-based ([5, 8]). It is well-known that automata theory, through various automata-logics connections, provides a unifying framework for proving decidability of various logics, such as (W)SkS, Presburger Arithmetic or MSO over certain classes of graphs. In this paper we propose an automata-theoretic approach consisting of two ingredients. First, SLRD$_{btw}$ formulae are translated into equivalent Monadic Second Order (MSO) formulae over graphs. Second, we show that the models of SLRD$_{btw}$ formulae have the *bounded tree width* property, which provides a decidability result by reduction to the satisfiability problem for MSO interpreted over graphs of bounded tree width [18], and ultimately, to the emptiness problem of tree automata.

Related Work The literature on defining decidable logics for describing mutable data structures is rather extensive. Initially, first-order logic with transitive closure of one function symbol was introduced in [11] with a follow-up logic of reachability on complex data structures, in [19]. The decision procedures for these logics are based on reductions to the decidability of MSO over finite trees. Along the same lines, the logic PALE [15] goes beyond trees, in defining trees with edges described by regular routing expressions, whose decidability is still a consequence of the decidability of MSO over trees. More recently, the CSL logic [4] uses first-order logic with reachability (along multiple selectors) in combination with arithmetic theories to reason about shape, path lengths and data within heap structures. Their decidability proof is based on a small model property, and the algorithm is enumerative. In the same spirit, the STRAND logic [14] combines MSO over graphs, with quantified data theories, and provides decidable fragments using a reduction to MSO over graphs of bounded tree width.

On what concerns SLRD [17], the first (proof-theoretic) decidability result on a restricted fragment defining only singly-linked lists was reported in [2], which describe a coNP 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]. Recently, the entailment problem for SLRD over lists has been reduced to graph homomorphism in [8], and can be solved in PTIME. This method has been extended to reason nested and overlaid lists in [10]. The logic SLRD_{btw}, presented in this paper is, to the best of our knowledge, the first decidable SL that can define structures more general than lists and trees, such as e.g. trees with parent pointers and linked leaves.

2 Preliminaries

For a finite set S, we denote by $\|S\|$ its cardinality. We sometimes denote sets and sequences of variables as $\mathbf{x}$, the distinction being clear from the context. If $\mathbf{x}$ denotes a sequence, $(\mathbf{x})_i$ denotes its i-th element. For a partial function $f : A \rightharpoonup B$, and $\bot \notin B$, we denote $f(x) = \bot$ the fact that f is undefined at some point $x \in A$. By $f[a \leftarrow b]$ we denote the function λx . if $x = a$ then b else $f(x)$. The domain of f is denoted $dom(f) = \{x \in A \mid f(x) \neq \bot\}$, 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, we denote by $f \oplus g$ their union.

Stores, Heaps and States. We consider $PVar = \{u, v, w, \ldots\}$ to be a countable infinite set of *pointer variables* and $Loc = \{l, m, n, \ldots\}$ to be a countable infinite set of *memory locations*. Let $nil \in PVar$ be a designated variable, $null \in Loc$ be a designated location, and $Sel = \{1, \ldots, \mathcal{S}\}$, for some given $\mathcal{S} > 0$, be a finite set of natural numbers, called *selectors* in the following.

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

Given a state $S = \langle s, h \rangle$, s is called the *store* and h the *heap*. For any $k \in Sel$, we write $h_k(\ell)$ instead of $(h(\ell))(k)$, and $\ell \xrightarrow{k} \ell'$ for $h_k(\ell) = \ell'$. We sometimes call a triple $\ell \xrightarrow{k} \ell'$ an *edge*, and k is called a *selector*. Let $Img(h) = \bigcup_{\ell \in Loc} img(h(\ell))$ be the set of locations which are destinations of some selector 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), and *dangling* in $\langle s, h \rangle$ if $\ell \in [img(s) \cup Img(h)] \setminus dom(h)$, i.e., it is either 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 a state $S = \langle s, h \rangle$.

Trees. Let Σ be a finite label 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 $p, q \in \mathbb{N}^*$. 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: $t(p.i) \neq \bot \Rightarrow \forall 0 \leq j < i \ . \ t(p.j) \neq \bot$. 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 $i \in \mathbb{N}$. Also q is a successor of p, or equivalently, p is the parent of q, denoted $p = parent(q)$ if $q = p.i$, for some $i \in \mathbb{N}$.

We will sometimes denote by $\mathcal{D}(t) = \{-1, 0, \ldots, N\}$ the *direction alphabet* of t, where $N = \max\{i \in \mathbb{N} \mid p.i \in dom(t)\}$. The concatenation of positions is defined over $\mathcal{D}(t)$ with the convention that $p.(-1) = q$ if and only if $p = q.i$ for some $i \in \mathbb{N}$. We denote $\mathcal{D}_+(t) = \mathcal{D}(t) \setminus \{-1\}$. A *path* in t, from p_1 to p_k, is a sequence $p_1, p_2, \ldots, p_k \in dom(t)$ of pairwise distinct positions, such that either $p_i = parent(p_{i+1})$ or $p_{i+1} = parent(p_i)$, for all $1 \leq i < k$. Notice that a path in the tree can also link sibling nodes, not just ancestors to their descendants, or viceversa. However, a path may not visit the same tree position twice.

Tree Width. A state (Def. 1) can be seen as a directed graph, whose nodes are locations, and whose edges are defined by the selector relation. Some nodes are labeled by program variables (*PVar*) and all edges are labeled by selectors (*Sel*). The notion of tree width is then easily adapted from generic labeled graphs to states. Intuitively, the tree width of a state (graph) measures the similarity of the state to a tree.

Definition 2. *Let $S = \langle s, h \rangle$ be a state. A* tree decomposition *of S is a tree $t : \mathbb{N}^* \rightharpoonup_{fin} 2^{loc(S)}$, labeled with sets of locations from $loc(S)$, with the following properties:*

1. *$loc(S) = \bigcup_{p \in dom(t)} t(p)$, the tree covers the locations of S*
2. *for each edge $l_1 \xrightarrow{s} l_2$ in S, there exists $p \in dom(t)$ such that $l_1, l_2 \in t(p)$*
3. *for each $p, q, r \in dom(t)$, if q is on a path from p to r in t, then $t(p) \cap t(r) \subseteq t(q)$*

The width of the decomposition is $w(t) = \max_{p \in dom(t)}\{\|t(p)\| - 1\}$. The tree width of S is $tw(S) = \min\{w(t) \mid t$ is a tree decomposition of $S\}$.

A set of states is said to have *bounded tree width* if there exists a constant $k \geq 0$ such that $tw(S) \leq k$, for any state S in the set. Figure 2 gives an example of a graph (left) and a possible tree decomposition (right).

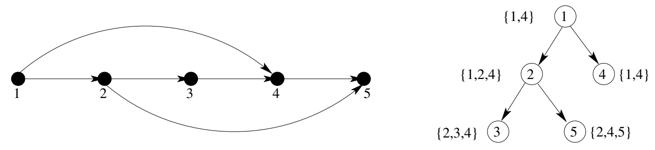

Fig. 2. A graph and a possible tree decomposition of width 2

2.1 Syntax and Semantics of Monadic Second Order Logic

Monadic second-order logic (MSO) on states is a straightforward adaptation of MSO on labeled graphs [13]. As usual, we denote first-order variables, ranging over locations, by $x, y, \ldots$, and second-order variables, ranging over sets of locations, by $X, Y, \ldots$. The set of logical MSO variables is denoted by $LVar_{mso}$, where $PVar \cap LVar_{mso} = \emptyset$.

We emphasize here the distinction between the logical variables $LVar_{mso}$ and the pointer variables $PVar$: the former may occur within the scope of first and second order quantifiers, whereas the latter play the role of symbolic constants (function symbols of zero arity). For the rest of this paper, a logical variable is said to be free if it does not occur within the scope of a quantifier. By writing $\varphi(\mathbf{x})$, for an MSO formula φ, and a set of logical variables $\mathbf{x}$, we mean that all free variables of φ are in $\mathbf{x}$.

The syntax of MSO is defined below:

$$\begin{array}{l} u \in PVar;\ x, X \in LVar_{mso};\ k \in \mathbb{N} \\ \varphi ::= x = y \mid var_u(x) \mid edge_k(x,y) \mid null(x) \mid X(x) \mid \varphi \wedge \varphi \mid \neg\varphi \mid \exists x . \varphi \mid \exists X . \varphi \end{array}$$

The semantics of MSO on states is given by the relation $S, \iota, \nu \models_{mso} \varphi$, where $S = \langle s, h \rangle$ is a state, $\iota : \{x, y, z, \ldots\} \rightharpoonup_{fin} Loc$ is an interpretation of the first order variables, and $\nu : \{X, Y, Z, \ldots\} \rightharpoonup_{fin} 2^{Loc}$ is an interpretation of the second order variables. If $S, \iota, \nu \models_{mso} \varphi$ for all interpretations $\iota : \{x, y, z, \ldots\} \rightharpoonup_{fin} Loc$ and $\nu : \{X, Y, Z, \ldots\} \rightharpoonup_{fin} 2^{Loc}$, then we say that S is a *model* of φ, denoted $S \models_{mso} \varphi$. We use the standard MSO semantics [18], with the following interpretations of the vertex and edge labels:

$$\begin{array}{rcl} S, \iota, \nu \models_{mso} null(x) & \iff & \iota(x) = nil \\ S, \iota, \nu \models_{mso} var_u(x) & \iff & s(u) = \iota(x) \\ S, \iota, \nu \models_{mso} edge_k(x,y) & \iff & h_k(\iota(x)) = \iota(y) \end{array}$$

The *satisfiability problem* for MSO asks, given a formula φ, whether there exists a state S such that $S \models_{mso} \varphi$. This problem is, in general, undecidable. However, one can show its decidability on a restricted class of models. The theorem below is a slight variation of a classical result in (MSO-definable) graph theory [18]. For space reasons, all proofs are given in [12].

Theorem 1. *Let $k \geq 0$ be an integer constant, and φ be an MSO formula. The problem asking if there exists a state S such that $tw(S) \leq k$ and $S \models_{mso} \varphi$ is decidable.*

2.2 Syntax and Semantics of Separation Logic

Separation Logic (SL) [17] uses only a set of first order logical variables, denoted as $LVar_{sl}$, ranging over locations. We suppose that $LVar_{sl} \cap PVar = \emptyset$ and $LVar_{sl} \cap LVar_{mso} = \emptyset$. Let Var_{sl} denote the set $PVar \cup LVar_{sl}$. A formula is said to be *closed* if it does not contain logical variables which are not under the scope of a quantifier. By writing $\varphi(\mathbf{x})$ for an SL formula φ and a set of logical variables $\mathbf{x}$, we mean that all free variables of φ are in $\mathbf{x}$.

Basic Formulae. The syntax of basic formula is given below:

$$\begin{array}{l} \alpha \in Var_{sl} \setminus \{nil\};\ \beta \in Var_{sl};\ x \in LVar_{sl} \\ \pi ::= \alpha = \beta \mid \alpha \neq \beta \mid \pi_1 \wedge \pi_2 \\ \sigma ::= emp \mid \alpha \mapsto (\beta_1, \ldots, \beta_n) \mid \sigma_1 * \sigma_2 \text{ , for some } n > 0 \\ \varphi ::= \pi \wedge \sigma \mid \exists x \, . \, \varphi \end{array}$$

A formula of the form $\bigwedge_{i=1}^{n} \alpha_i = \beta_i \wedge \bigwedge_{j=1}^{m} \alpha_j \neq \beta_j$ defined by π in the syntax above is said to be *pure*. If Π is a pure formula, let Π^* denote its *closure*, i.e., the equivalent pure formula obtained by the exhaustive application of the reflexivity, symmetry, and transitivity axioms of equality. A formula of the form $\bigstar_{i=1}^{k} \alpha_i \mapsto (\beta_{i,1}, \ldots, \beta_{i,n})$ defined by σ in the syntax above is said to be *spatial*. The atomic proposition *emp* denotes the empty spatial conjunction. For a spatial formula Σ, let $|\Sigma|$ be the total number of variable occurrences in Σ, e.g. $|emp| = 0$, $|\alpha \mapsto (\beta_1, \ldots, \beta_n)| = n+1$, etc.

The semantics of a basic formula φ is given by the relation $S, \iota \models_{sl} \varphi$ where $S = \langle s, h \rangle$ is a state, and $\iota : LVar_{sl} \rightharpoonup_{fin} Loc$ is an interpretation of logical variables from φ. For a closed formula φ, we denote by $S \models_{sl} \varphi$ the fact that S is a *model* of φ.

$$\begin{array}{lcl}
S, \iota \models_{sl} emp & \iff & dom(h) = \emptyset \\
S, \iota \models_{sl} \alpha \mapsto (\beta_1, \ldots, \beta_n) & \iff & h = \{\langle (s \oplus \iota)(\alpha), \lambda i \,.\, \text{if } i \leq n \text{ then } (s \oplus \iota)(\beta_i) \text{ else } \bot \rangle\} \\
S, \iota \models_{sl} \varphi_1 * \varphi_2 & \iff & S_1, \iota \models_{sl} \varphi_1 \text{ and } S_2, \iota \models_{sl} \varphi_2 \text{ where } S_1 \uplus S_2 = S
\end{array}$$

The semantics of $=$, $\neq$, $\wedge$, and $\exists$ is classical. Here, the notation $S_1 \uplus S_2 = S$ means that S is the union of two states $S_1 = \langle s_1, h_1 \rangle$ and $S_2 = \langle s_2, h_2 \rangle$ whose stacks agree on the evaluation of common program variables ($\forall \alpha \in PVar \,.\, s_1(\alpha) \neq \bot \wedge s_2(\alpha) \neq \bot \Rightarrow s_1(\alpha) = s_2(\alpha)$), and whose heaps have disjoint domains ($dom(h_1) \cap dom(h_2) = \emptyset$) i.e., $S = \langle s_1 \cup s_2, h_1 \oplus h_2 \rangle$. Note that we adopt here the *strict semantics*, in which a points-to relation $\alpha \mapsto (\beta_1, \ldots, \beta_n)$ holds in a state consisting of a single cell pointed to by α, with exactly n outgoing edges towards dangling locations pointed to by $\beta_1, \ldots, \beta_n$, and the empty heap is specified by *emp*.

Every basic formula φ is equivalent to an existentially quantified pair $\Sigma \wedge \Pi$ where Σ is a spatial formula and Π is a pure formula. Given a basic formula φ, one can define its spatial (Σ) and pure (Π) parts uniquely, up to equivalence. A variable $\alpha \in Var$ is said to be *allocated in* φ if and only if $\alpha \mapsto (\ldots)$ occurs in Σ. It is easy to check that an allocated variable may not refer to a dangling location in any model of φ. A variable β is *referenced* if and only if $\alpha \mapsto (\ldots, \beta, \ldots)$ occurs in Σ for some variable α. For a basic formula $\varphi \equiv \Sigma \wedge \Pi$, the *size of* φ is defined as $|\varphi| = |\Sigma|$.

Lemma 1. *Let $\varphi(\mathbf{x})$ be a basic SL formula, $S = \langle s, h \rangle$ be a state, and $\iota : LVar_{sl} \rightharpoonup_{fin} Loc$ be an interpretation, such that $S, \iota \models_{sl} \varphi(\mathbf{x})$. Then $tw(S) \leq \max(|\varphi|, \|PVar\|)$.*

Recursive Definitions. A system $\mathcal{P}$ of *recursive definitions* is of the form:

$$\begin{array}{rcl}
P_1(x_{1,1}, \ldots, x_{1,n_1}) & ::= & |_{j=1}^{m_1} R_{1,j}(x_{1,1}, \ldots, x_{1,n_1}) \\
 & \ldots & \\
P_k(x_{k,1}, \ldots, x_{k,n_k}) & ::= & |_{j=1}^{m_k} R_{k,j}(x_{k,1}, \ldots, x_{k,n_k})
\end{array}$$

where $P_1, \ldots, P_k$ are called *predicates*, $x_{i,1}, \ldots, x_{i,n_i}$ are called *parameters*, and the formulae $R_{i,j}$ are called the *rules* of P_i. Concretely, a rule $R_{i,j}$ 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 Σ is a spatial SL formula over variables $\mathbf{x} \cup \mathbf{z}$, called the *head* of $R_{i,j}$, $\langle P_{i_1}(\mathbf{y}_1), \ldots, P_{i_m}(\mathbf{y}_m) \rangle$ is an ordered sequence of *predicate occurrences*, called the *tail* of $R_{i,j}$ (we assume w.l.o.g. that $\mathbf{x} \cap \mathbf{z} = \emptyset$, and that $\mathbf{y}_k \subseteq \mathbf{x} \cup \mathbf{z}$, for all $k = 1, \ldots, m$), Π is a pure formula over variables $\mathbf{x} \cup \mathbf{z}$.

Without losing generality, we assume that all variables occurring in a rule of a recursive definition system are logical variables from $LVar_{sl}$ – pointer variables can be passed as parameters at the top level. We subsequently denote $head(R_{i,j}) \equiv \Sigma$, $tail(R_{i,j}) \equiv \langle P_{i_k}(\mathbf{y}_k)\rangle_{k=1}^{m}$ and $pure(R_{i,j}) \equiv \Pi$, for each rule $R_{i,j}$. Rules with empty tail are called *base cases*. For each rule $R_{i,j}$ let $\|R_{i,j}\|^{var} = \|\mathbf{z}\| + \|\mathbf{x}\|$ be the number of variables, both existentially quantified and parameters, that occur in $R_{i,j}$. We denote by $\|\mathcal{P}\|^{var} = \max\{\|R_{i.j}\|^{var} \mid 1 \leq i \leq k,\ 1 \leq j \leq m_i\}$ the maximum such number, among all rules in $\mathcal{P}$. We also denote by $\mathcal{D}(\mathcal{P}) = \{-1, 0, \ldots, \max\{|tail(R_{i,j})| \mid 1 \leq i \leq k,\ 1 \leq j \leq m_i\} - 1\}$ the *direction alphabet* of $\mathcal{P}$.

Example. The predicate tll describes a data structure called a *tree with parent pointers and linked leaves* (see Fig. 3(b)). The data structure is composed of a binary tree in which each internal node points to left and right children, and also to its parent node. In addition, the leaves of the tree are kept in a singly-linked list, according to the order in which they appear on the frontier (left to right).

$$\begin{aligned} tll(x,p,leaf_l,leaf_r) ::= {} & x \mapsto (nil,nil,p,leaf_r) \wedge x = leaf_l & (R_1)\\ \mid {} & \exists l,r,z.\ x \mapsto (l,r,p,nil) * tll(l,x,leaf_l,z) * tll(r,x,z,leaf_r) & (R_2) \end{aligned}$$

The base case rule (R_1) allocates leaf nodes. The internal nodes of the tree are allocated by the rule (R_2), where the ttl predicate occurs twice, first for the left subtree, and second for the right subtree. □

Definition 3. *Given a system of recursive definitions $\mathcal{P} = \left\{P_i\ ::= |_{j=1}^{m_i} R_{i,j}\right\}_{i=1}^{n}$, an* unfolding tree of $\mathcal{P}$ rooted at i *is a finite tree t such that:*

1. *each node of t is labeled by a single rule of the system $\mathcal{P}$,*
2. *the root of t is labeled with a rule of P_i,*
3. *nodes labeled with base case rules have no successors, and*
4. *if a node u of t is labeled with a rule whose tail is $P_{i_1}(\mathbf{y}_1) * \ldots * P_{i_m}(\mathbf{y}_m)$, then the children of u form the ordered sequence $v_1, \ldots, v_m$ where v_j is labeled with one of the rules of P_{i_j} for all $j = 1, \ldots, m$.*

Remarks. Notice that the recursive predicate $P(x) ::= \exists y\ .\ x \mapsto y * P(y)$ does not have finite unfolding trees. However, in general a system of recursive predicates may have infinitely many finite unfolding trees. □

In the following, we denote by $\mathcal{T}_i(\mathcal{P})$ the set of unfolding trees of $\mathcal{P}$ rooted at i. An unfolding tree $t \in \mathcal{T}_i(\mathcal{P})$ corresponds to a basic formula of separation logic ϕ_t, called the *characteristic formula* of t, and defined in what follows. For a set of tree positions $P \subseteq \mathbb{N}^*$, we denote $LVar^P = \{x^p \mid x \in LVar,\ p \in P\}$. For a tree position $p \in \mathbb{N}^*$ and a rule R, we denote by R^p the rule obtained by replacing every variable occurrence x in R by x^p. For each position $p \in dom(t)$, we define a formula ϕ_t^p, by induction on the structure of the subtree of t rooted at p:

- if p is a leaf labeled with a base case rule R, then $\phi_t^p \equiv R^p$
- if p has successors $p.1, \ldots, p.m$, and the label of p is the recursive rule $R(\mathbf{x}) \equiv \exists \mathbf{z}\ .\ head(R) * \bigstar_{j=1}^{m} P_{i_j}(\mathbf{y}_j) \wedge pure(R)$, then:

$$\phi_t^p(\mathbf{x}^p) \equiv \exists \mathbf{z}^p\ .\ head(R^p) * \bigstar_{j=1}^{m} [\exists \mathbf{x}_{i_j}^{p.i}\ .\ \phi_t^{p.i}(\mathbf{x}_{i_j}^{p.i}) \wedge \mathbf{y}_j^p = \mathbf{x}_{i_j}^{p.i}] \wedge pure(R^p)$$

In the rest of the paper, we write ϕ_t for ϕ_t^ε. Notice that ϕ_t is defined using the set of logical variables $LVar^{dom(t)}$, instead of $LVar$. However the definition of SL semantics from the previous carries over naturally to this case.

Example. (cont'd) Fig. 3(a) presents an unfolding tree for the *tll* predicate given in the previous example. The characteristic formula of each node in the tree can be obtained by composing the formulae labeling the children of the node with the formula labeling the node. The characteristic formula of the tree is the formula of its root. $\square$

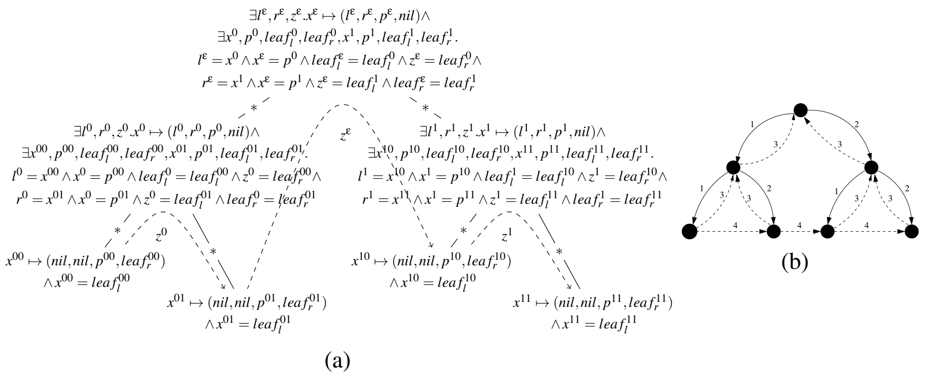
$\exists l^\varepsilon, r^\varepsilon, z^\varepsilon . x^\varepsilon \mapsto (l^\varepsilon, r^\varepsilon, p^\varepsilon, nil) \wedge$
$\exists x^0, p^0, leaf_l^0, leaf_r^0, x^1, p^1, leaf_l^1, leaf_r^1 .$
$l^\varepsilon = x^0 \wedge x^\varepsilon = p^0 \wedge leaf_l^\varepsilon = leaf_l^0 \wedge z^\varepsilon = leaf_r^0 \wedge$
$r^\varepsilon = x^1 \wedge x^\varepsilon = p^1 \wedge z^\varepsilon = leaf_l^1 \wedge leaf_r^\varepsilon = leaf_r^1$
$*$
z^ε
$*$
$\exists l^0, r^0, z^0 . x^0 \mapsto (l^0, r^0, p^0, nil) \wedge$
$\exists x^{00}, p^{00}, leaf_l^{00}, leaf_r^{00}, x^{01}, p^{01}, leaf_l^{01}, leaf_r^{01} .$
$l^0 = x^{00} \wedge x^0 = p^{00} \wedge leaf_l^0 = leaf_l^{00} \wedge z^0 = leaf_r^{00} \wedge$
$r^0 = x^{01} \wedge x^0 = p^{01} \wedge z^0 = leaf_l^{01} \wedge leaf_r^0 = leaf_r^{01}$
$\exists l^1, r^1, z^1 . x^1 \mapsto (l^1, r^1, p^1, nil) \wedge$
$\exists x^{10}, p^{10}, leaf_l^{10}, leaf_r^{10}, x^{11}, p^{11}, leaf_l^{11}, leaf_r^{11} .$
$l^1 = x^{10} \wedge x^1 = p^{10} \wedge leaf_l^1 = leaf_l^{10} \wedge z^1 = leaf_r^{10} \wedge$
$r^1 = x^{11} \wedge x^1 = p^{11} \wedge z^1 = leaf_l^{11} \wedge leaf_r^1 = leaf_r^{11}$
$*$
z^0
$*$
$*$
z^1
$*$
$x^{00} \mapsto (nil, nil, p^{00}, leaf_r^{00}) \wedge x^{00} = leaf_l^{00}$
$x^{01} \mapsto (nil, nil, p^{01}, leaf_r^{01}) \wedge x^{01} = leaf_l^{01}$
$x^{10} \mapsto (nil, nil, p^{10}, leaf_r^{10}) \wedge x^{10} = leaf_l^{10}$
$x^{11} \mapsto (nil, nil, p^{11}, leaf_r^{11}) \wedge x^{11} = leaf_l^{11}$
(a)
1
2
3
3
1
2
3
3
1
2
3
3
4
4
4
(b)

Fig. 3. (a) An unfolding tree for tll predicate and (b) a model of the corresponding formula

Given a system of recursive definitions $\mathcal{P} = \left\{P_i ::= |_{j=1}^{m_i} R_{i,j}\right\}_{i=1}^{n}$, the semantics of a recursive predicate P_i is defined as follows:

$$S, \iota \models_{sl} P_i(x_{i,1}, \ldots, x_{i,n_i}) \iff S, \iota^\varepsilon \models_{sl} \phi_t(x_{i,1}^\varepsilon, \ldots, x_{i,n_i}^\varepsilon), \text{ for some } t \in \mathcal{T}_i(\mathcal{P}) \qquad (1)$$

where $\iota^\varepsilon(x_{i,j}^\varepsilon) \stackrel{def}{=} \iota(x_{i,j})$ for all $j = 1, \ldots, n_i$.

Remark. Since the recursive predicate $P(x) ::= \exists y \, . \, x \mapsto y * P(y)$ does not have finite unfolding trees, the formula $\exists x . P(x)$ is unsatisfiable. $\square$

Top Level Formulae. We are now ready to introduce the fragment of *Separation Logic with Recursive Definitions* (SLRD). A formula in this fragment is an existentially quantified formula of the following form: $\exists \mathbf{z} \, . \, \varphi * P_{i_1} * \ldots * P_{i_n}$, where φ is a basic formula, and P_{i_j} are occurrences of recursive predicates, with free variables in $PVar \cup \mathbf{z}$. The semantics of an SLRD formula is defined in the obvious way, from the semantics of the basic fragment, and that of the recursive predicates.

Example. The following SLRD formulae, with $PVar = \{root, head\}$, describe both the set of binary trees with parent pointer and linked leaves, rooted at *root*, with the leaves linked into a list pointed to by *head*. The difference is that φ_1 describes also a tree containing only a single allocated location:

$$\varphi_1 \equiv tll(root, nil, head, nil)$$
$$\varphi_2 \equiv \exists l, r, x. root \mapsto (l, r, nil, nil) * tll(l, root, head, x) * tll(r, root, x, nil) \quad \square$$

We are interested in solving two problems on SLRD formulae, namely *satisfiability* and *entailment*. The satisfiability problem asks, given a closed SLRD formula φ, whether there exists a state S such that $S \models_{sl} \varphi$. The entailment problem asks, given two closed SLRD formulae φ_1 and φ_2, whether for all states S, $S \models_{sl} \varphi_1$ implies $S \models_{sl} \varphi_2$. This is denoted also as $\varphi_1 \models_{sl} \varphi_2$. For instance, in the previous example we have $\varphi_2 \models_{sl} \varphi_1$, but not $\varphi_1 \models_{sl} \varphi_2$.

In general, it is possible to reduce an entailment problem $\varphi_1 \models \varphi_2$ to satisfiability of the formula $\varphi_1 \wedge \neg\varphi_2$. In our case, however, this is not possible directly, because SLRD is not closed under negation. The decision procedures for satisfiability and entailment is the subject of the rest of this paper.

3 Decidability of Satisfiability and Entailment in SLRD

The decision procedure for the satisfiability and entailment in SLRD is based on two ingredients. First, we show that, under certain natural restrictions on the system of recursive predicates, which define a fragment of SLRD, called SLRD$_{btw}$, all states that are models of SLRD$_{btw}$ formulae have *bounded tree width* (Def. 2). These restrictions are as follows:

1. *Progress*: each rule allocates exactly one variable
2. *Connectivity*: there is at least one selector edge between the variable allocated by a rule and the variable allocated by each of its children in the unfolding tree
3. *Establishment*: all existentially quantified variables in a recursive rule are eventually allocated

Second, we provide a *translation of SLRD$_{btw}$ formulae into equivalent MSO formulae*, and rely on the fact that satisfiability of MSO is decidable on classes of states with bounded tree width.

3.1 A Decidable Subset of SLRD

At this point we define the SLRD$_{btw}$ fragment formally, by defining the three restrictions above. The *progress* condition (1) asks that, for each rule R in the system of recursive definitions, we have $head(R) \equiv \alpha \mapsto (\beta_1, \ldots, \beta_n)$, for some variables $\alpha, \beta_1, \ldots, \beta_n \in Var_{sl}$. The intuition between this restriction is reflected by the following example.

Example. Consider the following system of recursive definitions:

$$ls(x,y) ::= x \mapsto y \mid \exists z, t \,.\, x \mapsto (z, nil) * t \mapsto (nil, y) * ls(z,t)$$

The predicate $ls(x,y)$ defines the set of structures $\{x(\xrightarrow{1})^n z \mapsto t(\xrightarrow{2})^n y \mid n \geq 0\}$, which clearly cannot be defined in MSO. $\square$

The *connectivity* condition (2) is defined below:

Definition 4. *A rule R of a system of recursive definitions, such that $head(R) \equiv \alpha \mapsto (\beta_1, \dots, \beta_n)$ and $tail(R) \equiv \langle P_{i_1}(\mathbf{y}_1), \dots, P_{i_m}(\mathbf{y}_m)\rangle$, $m \geq 1$, is said to be* connected *if and only if the following hold:*

- *for each $j = 1, \dots, m$, $(\mathbf{y}_j)_s = \beta'$, for some $1 \leq s \leq n_{i_j}$, where n_{i_j} is the number of parameters of P_{i_j}*
- *$\beta_t = \beta'$ occurs in $pure(R)^*$, for some $1 \leq t \leq n$*
- *the s-th parameter $x_{i_j,s}$ of P_{i_j} is allocated in the heads of all rules of P_{i_j}.*

In this case we say that between rule R and any rule Q of P_{i_j}, there is a *local edge*, labeled by *selector t*. $\mathcal{F}(R, j, Q) \subseteq Sel$ denotes the set of all such selectors. If all rules of $\mathcal{P}$ are connected, we say that $\mathcal{P}$ is connected.

Example. The following recursive rule, from the previous tll predicate, is connected:

$$\exists l, r, z \,.\, x \mapsto (l, r, p, nil) * tll(l, x, leaf_l, z) * tll(r, x, z, leaf_r) \ (R_2)$$

R_2 is connected because the variable l is referenced in R_2 and it is passed as the first parameter to tll in the first recursive call to tll. Moreover, the first parameter (x) is allocated by all rules of tll. R_2 is connected, for similar reasons. We have $\mathcal{F}(R_2, 1, R_2) = \{1\}$ and $\mathcal{F}(R_2, 2, R_2) = \{2\}$. □

The *establishment* condition (3) is formally defined below.

Definition 5. *Let $P(x_1, \dots, x_n) = |_{j=1}^{m} R_j(x_1, \dots, x_n)$ be a predicate in a recursive system of definitions. We say that a parameter x_i, for some $i = 1, \dots, n$ is* allocated *in P if and only if, for all $j = 1, \dots, m$:*

- *either x_i is allocated in $head(R_j)$, or*
- *(i) $tail(R_j) = \langle P_{i_1}(\mathbf{y}_1), \dots, P_{i_k}(\mathbf{y}_k)\rangle$, (ii) $(\mathbf{y}_\ell)_s = x_i$ occurs in $pure(R_j)^*$, for some $\ell = 1, \dots, k$, and (iii) the s-th parameter of P_{i_ℓ} is allocated in P_{i_ℓ}*

A system of recursive definitions is said to be established *if and only if every existentially quantified variable is allocated.*

Example. Let $llextra(x) ::= x \mapsto (nil, nil) \mid \exists n, e.\ x \mapsto (n, e) * llextra(n)$ be a recursive definition system, and let $\phi ::= llextra(head)$, where $head \in PVar$. The models of the formula ϕ are singly-linked lists, where in all locations of the heap, the first selector points to the next location in the list, and the second selector is dangling i.e., it can point to any location in the heap. These dangling selectors may form a squared grid of arbitrary size, which is a model of the formula ϕ. However, the set of squared grids does not have bounded tree width [18]. The problem arises due to the existentially quantified variables e which are never allocated. □

Given a system $\mathcal{P}$ of recursive definitions, one can effectively check whether it is established, by guessing, for each predicate $P_i(x_{i,1}, \dots, x_{i,n_i})$ of $\mathcal{P}$, the minimal set of parameters which are allocated in P_i, and verify this guess inductively[3]. Then, once the minimal set of allocated parameters is determined for each predicate, one can check whether every existentially quantified variable is eventually allocated.

[3] For efficiency, a least fixpoint iteration can be used instead of a non-deterministic guess.

Lemma 2. *Let $\mathcal{P} = \{P_i ::= |_{j=1}^{m_i} R_{ij}(x_{i,1},\ldots,x_{i,n_i})\}_{i=1}^{k}$ be a established system of recursive definitions, and $S = \langle s,h\rangle$ be a state, such that $S,\iota \models_{sl} P_i(x_{i,1},\ldots,x_{i,n_i})$ for some interpretation $\iota : LVar_{sl} \rightharpoonup_{fin} Loc$ and some $1 \leq i \leq k$. Then $tw(S) \leq \|\mathcal{P}\|^{var}$.*

The result of the previous lemma extends to an arbitrary top-level formula:

Theorem 2. *Let $\mathcal{P} = \{P_i ::= |_{j=1}^{m_i} R_{ij}(x_{i,1},\ldots,x_{i,n_i})\}_{i=1}^{k}$ be a established system of recursive definitions, and $S = \langle s,h\rangle$ be a state, such that $S \models_{sl} \exists \mathbf{z} \,.\, \varphi(\mathbf{y}_0) * P_{i_1}(\mathbf{y}_1) * \ldots * P_{i_n}(\mathbf{y}_n)$, where φ is a basic SL formula, and P_{i_j} are predicates of $\mathcal{P}$, and $\mathbf{y}_i \subseteq \mathbf{z}$, for all $i = 0,1,\ldots,n$. Then $tw(S) \leq \max(\|\mathbf{z}\|, |\varphi|, \|PVar\|, \|\mathcal{P}\|^{var})$.*

4 From SLRD$_{btw}$ to MSO

This section describes the translation of a SL formula using recursively defined predicates into an MSO formula. We denote by $\Pi(X_0,\ldots,X_i,X)$ the fact that $X_0,\ldots,X_i$ is a partition of X, and by $\Sigma(x,X)$ the fact that X is a singleton with x as the only element.

4.1 Converting Basic SL Formulae to MSO

For every SL logical variable $x \in LVar_{sl}$ we assume the existence of an MSO logical variable $\overline{x} \in LVar_{mso}$, which is used to replace x in the translation. For every program variable $u \in PVar \setminus \{nil\}$ we assume the existence of a logical variable $\overline{x_u} \in LVar_{mso}$. The special variable $nil \in LVar_{sl}$ is translated into $\overline{x_{nil}} \in LVar_{mso}$ (with the associated MSO constraint $null(\overline{x_{nil}})$). In general, for any pointer or logical variable $\alpha \in Var_{sl}$, we denote by $\overline{\alpha}$, the logical MSO variable corresponding to it.

The translation of a pure SL formula $\alpha = \beta$, $\alpha \neq \beta$, $\pi_1 \wedge \pi_2$ is $\overline{\alpha} = \overline{\beta}$, $\neg(\overline{\alpha} = \overline{\beta})$, $\overline{\pi_1} \wedge \overline{\pi_2}$, respectively, where $\overline{\pi}(\overline{\alpha_1},\ldots,\overline{\alpha_k})$ is the translation of $\pi(\alpha_1,\ldots,\alpha_k)$. Spatial SL formulae $\sigma(\alpha_1,\ldots,\alpha_k)$ are translated into MSO formulae $\overline{\sigma}(\overline{\alpha_1},\ldots,\overline{\alpha_k},X)$, where X is used for the set of locations allocated in σ. The fact that X actually denotes the domain of the heap, is ensured by the following MSO constraint:

$$Heap(X) \equiv \forall x \bigvee_{i=1}^{\|Sel\|} (\exists y \,.\, edge_i(x,y)) \leftrightarrow X(x)$$

The translation of basic spatial formulae is defined by induction on their structure:

$$\begin{array}{ll}
\overline{emp}(X) & \equiv \forall x \,.\, \neg X(x) \\
\overline{(\alpha \mapsto (\beta_1,\ldots,\beta_n))}(X) & \equiv \Sigma(\overline{\alpha},X) \;\wedge\; \bigwedge_{i=1}^{n} edge_i(\overline{\alpha},\overline{\beta_i}) \;\wedge\; \bigwedge_{i=n+1}^{\|Sel\|} \forall x \,.\, \neg edge_i(\overline{\alpha},x) \\
\overline{(\sigma_1 * \sigma_2)}(X) & \equiv \exists Y \exists Z \,.\, \overline{\sigma_1}(Y) \;\wedge\; \overline{\sigma_2}(Z) \;\wedge\; \Pi(Y,Z,X)
\end{array}$$

The translation of a closed basic SL formula φ in MSO is defined as $\exists X \,.\, \overline{\varphi}(X)$, where $\overline{\varphi}(X)$ is defined as $\overline{(\pi \wedge \sigma)}(X) \equiv \overline{\pi} \wedge \overline{\sigma}(X)$, and $\overline{(\exists x \,.\, \varphi_1)}(X) \equiv \exists \overline{x} \,.\, \overline{\varphi_1}(X)$. The following lemma proves that the MSO translation of a basic SL formula defines the same set of models as the original SL formula.

Lemma 3. *For any state $S = \langle s,h\rangle$, any interpretation $\iota : LVar_{sl} \rightharpoonup_{fin} Loc$, and any basic SL formula φ, we have $S,\iota \models_{sl} \varphi$ if and only if $S,\bar{\iota},\nu[X \leftarrow dom(h)] \models_{mso} \overline{\varphi}(X) \wedge Heap(X)$, where $\bar{\iota} : LVar_{mso} \rightharpoonup_{fin} Loc$ is an interpretation of first order variables, such that $\bar{\iota}(x_u) = s(u)$, for all $u \in PVar$, and $\bar{\iota}(\bar{x}) = \iota(x)$, for all $x \in LVar_{sl}$, and $\nu : LVar_{mso} \rightharpoonup_{fin} 2^{Loc}$ is any interpretation of second-order variables.*

4.2 States and Backbones

The rest of this section is concerned with the MSO definition of states that are models of recursive SL formulae, i.e. formulae involving recursively defined predicates. The main idea behind this encoding is that any part of a state which is the model of a recursive predicate can be decomposed into a tree-like structure, called the *backbone*, and a set of edges between the nodes in this tree. Intuitively, the backbone is a spanning tree that uses only *local edges*. For instance, in the state depicted in Fig. 3(b), the local edges are drawn in solid lines.

Let $P_k(x_1,\ldots,x_n)$ be a recursively defined predicate of a system $\mathcal{P}$, and $S,\iota \models_{sl} P_k(x_1,\ldots,x_n)$, for some state $S = \langle s,h\rangle$ and some interpretation $\iota : LVar_{sl} \rightarrow Loc$. Then $S,\iota \models_{sl} \phi_t$, where $t \in \mathcal{T}_k(\mathcal{P})$ is an unfolding tree, ϕ_t is its characteristic formula, and $\mu : dom(t) \rightarrow dom(h)$ is the bijective tree that describes the allocation of nodes in the heap by rules labeling the unfolding tree. Recall that the direction alphabet of the system $\mathcal{P}$ is $\mathcal{D}(\mathcal{P}) = \{-1,0,\ldots,N-1\}$, where N is the maximum number of predicate occurrences within some rule of $\mathcal{P}$, and denote $\mathcal{D}_+(\mathcal{P}) = \mathcal{D}(\mathcal{P}) \setminus \{-1\}$. For each rule R_{ij} in $\mathcal{P}$ and each direction $d \in \mathcal{D}(\mathcal{P})$, we introduce a second order variable X^d_{ij} to denote the set of locations ℓ such that (i) $t(\mu^{-1}(\ell)) \equiv R_{ij}$ and (ii) $\mu^{-1}(\ell)$ is a d-th child, if $d \geq 0$, or $\mu^{-1}(\ell)$ is the root of t, if $d = -1$. Let $\overrightarrow{\mathbf{X}}$ be the sequence of X^k_{ij} variables, enumerated in some order. We use the following shorthands:

$$X_{ij}(x) \equiv \bigvee_{k\in\mathcal{D}(\mathcal{P})} X^k_{ij}(x) \qquad X_i(x) \equiv \bigvee_{1\leq j\leq m_i} X_{ij}(x) \qquad X^k_i(x) \equiv \bigvee_{1\leq j\leq m_i} X^k_{ij}(x)$$

to denote, respectively, locations that are allocated by a rule R_{ij} (X_{ij}), by a recursive predicate P_i (X_i), or by a predicate P_i, who are mapped to a k-th child (or to the root, if $k = -1$) in the unfolding tree of $\mathcal{P}$, rooted at i (X^k_i).

In order to characterize the backbone of a state, one must first define the local edges:

$$local\_edge^d_{i,j,p,q}(x,y) \equiv \textstyle\bigwedge_{s\in\mathcal{F}(R_{i,j},d,R_{pq})} edge_s(x,y)$$

for all $d \in \mathcal{D}_+(\mathcal{P})$. Here $\mathcal{F}(R_{ij},d,R_{pq})$ is the set of forward local selectors for direction d, which was defined previously – notice that the set of local edges depends on the source and destination rules R_{ij} and R_{pq}, that label the corresponding nodes in the unfolding tree, respectively. The following predicate ensures that these labels are used correctly, and define the successor functions in the unfolding tree:

$$succ_d(x,y,\overrightarrow{\mathbf{X}}) \equiv \bigvee_{\substack{1\ \leq i,p \leq\ M \\ 1\ \leq j \leq\ m_i \\ 1\ \leq q \leq\ m_p}} X_{ij}(x) \wedge X^k_{pq}(y) \wedge local\_edge^d_{i,j,p,q}(x,y)$$

for all $d \in \mathcal{D}_+(\mathcal{P})$. The definition of the backbone of a recursive predicate P_i in MSO follows tightly the definition of the unfolding tree of $\mathcal{P}$ rooted at i (Def. 3):

$$backbone_i(r, \overrightarrow{\mathbf{X}}, T) \equiv tree(r, \overrightarrow{\mathbf{X}}, T) \wedge X_i^{-1}(r) \wedge succ\_labels(\overrightarrow{\mathbf{X}})$$

where $tree(r, \overrightarrow{\mathbf{X}}, T)$ defines a tree[4] with domain T, rooted at r, with successor functions defined by $succ_0, \ldots, succ_{N-1}$, and $succ\_labels$ ensures that the labeling of each tree position (with rules of $\mathcal{P}$) is consistent with the definition of $\mathcal{P}$:

$$succ\_labels(\overrightarrow{\mathbf{X}}) \equiv \bigwedge_{\substack{1 \le i \le M \\ 1 \le j \le m_i}} \begin{array}{l} X_{ij}(x) \to \bigwedge_{d=0}^{r_{ij}-1} \exists y \,.\, X_{k_d}^d(y) \wedge succ_d(x, y, \overrightarrow{\mathbf{X}}) \\ \qquad \wedge \forall y \,.\, \bigwedge_{p=s_{ij}+1}^{\|Sel\|} \neg edge_p(x, y) \end{array}$$

where we suppose that, for each rule R_{ij} of $\mathcal{P}$, we have $head(R_{ij}) \equiv \alpha \mapsto (\beta_1, \ldots, \beta_{s_{ij}})$ and $tail(R_{ij}) = \langle P_{k_1}, \ldots, P_{k_{r_{ij}}} \rangle$, for some $r_{ij} \geq 0$, and some indexing $k_1, \ldots, k_{r_{ij}}$ of predicate occurrences within R_{ij}. The last conjunct ensures that a location allocated in R_{ij} does not have more outgoing edges than specified by $head(R_{ij})$. This condition is needed, since, unlike SL, the semantics of MSO does not impose strictness conditions on the number of outgoing edges.

4.3 Inner Edges

An edge between two locations is said to be *inner* if both locations are allocated in the heap. Let μ be the bijective tree defined in Sec. 4.2. The existence of an edge $\ell \xrightarrow{k} \ell'$ in S, between two arbitrary locations $\ell, \ell' \in dom(h)$, is the consequence of:

1. a basic points-to formula $\alpha \mapsto (\beta_1, \ldots, \beta_k, \ldots, \beta_n)$ that occurs in $\mu(\ell)$
2. a basic points-to formula $\gamma \mapsto (\ldots)$ that occurs in $\mu(\ell')$
3. a path $\mu(\ell) = p_1, p_2, \ldots, p_{m-1}, p_m = \mu(\ell')$ in t, such that the equalities $\beta_k^{p_1} = \delta_2^{p_2} = \ldots = \delta_{m-1}^{p_{m-1}} = \gamma^{p_m}$ are all logical consequences of ϕ_t, for some tree positions $p_2, \ldots, p_{m-1} \in dom(t)$ and some variables $\delta_2, \ldots, \delta_{m-1} \in LVar_{sl}$.

Notice that the above conditions hold only for inner edges. The (corner) case of edges leading to dangling locations is dealt with in [12].

Example. The existence of the edge from tree position 00 to 01 in Fig. 3(b), is a consequence of the following: (1) $x^{00} \mapsto (nil, nil, p^{00}, leaf_r^{00})$, (2) $x^{01} \mapsto (nil, nil, p^{01}, leaf_r^{01})$, and (3) $leaf_r^{00} = z^0 = leaf_l^{01} = x^{01}$. The reason for other dashed edges is similar. □

The main idea here is to encode in MSO the existence of such paths, in the unfolding tree, between the source and the destination of an edge, and use this encoding to define the edges. To this end, we use a special class of tree automata, called *tree-walking automata* (TWA) to recognize paths corresponding to sequences of equalities occurring within characteristic formulae of unfolding trees.

[4] For space reasons this definition can be found in [12].

Tree Walking Automata Given a set of tree directions $\mathcal{D} = \{-1, 0, \ldots, N\}$ for some $N \geq 0$, a tree-walking automaton[5], is a tuple $A = (\Sigma, Q, q_i, q_f, \Delta)$ where Σ is a set of tree node labels, Q is a set of states, $q_i, q_f \in Q$ are the initial and final states, and $\Delta : Q \times (\Sigma \cup \{root\}) \times (\Sigma \cup \{?\}) \rightarrow 2^{Q \times (\mathcal{D} \cup \{\varepsilon\})}$ is the (non-deterministic) transition function. A configuration of A is a pair $\langle p, q \rangle$, where $p \in \mathcal{D}^*$ is a tree position, and $q \in Q$ is a state. A run of A over a Σ-labeled tree t is a sequence of configurations $\langle p_1, q_1 \rangle, \ldots, \langle p_n, q_n \rangle$, with $p_1, \ldots, p_n \in dom(t)$, such that for all $i = 1, \ldots, n-1$, we have $p_{i+1} = p_i.k$, where either:

1. $p_i \neq \varepsilon$ and $(q_{i+1}, k) \in \Delta(q_i, t(p_i), t(p_i.(-1)))$, for $k \in \mathcal{D} \cup \{\varepsilon\}$
2. $p_i = \varepsilon$ and $(q_{i+1}, k) \in \Delta(q_i, \sigma, ?)$, for $\sigma \in \{t(p_i) \cup root\}$ and $k \in \mathcal{D} \cup \{\varepsilon\}$

The run is said to be *accepting* if $q_1 = q_i$, $p_1 = \varepsilon$ and $q_n = q_f$.

Routing Automata For a system of recursive definitions $\mathcal{P} = \left\{P_i(x_{i,1}, \ldots, x_{i,n_i}) ::= |_{j=1}^{m_i} R_{ij}(x_{i,1}, \ldots, x_{i,n_i})\right\}_{i=1}^{k}$, we define the TWA $A_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, Q_{\mathcal{P}}, q_i, q_f, \Delta_{\mathcal{P}})$, where $\Sigma_{\mathcal{P}} = \{R_{ij}^k \mid 1 \leq i \leq k,\ 1 \leq j \leq m_i,\ k \in \mathcal{D}(\mathcal{P})\}$, $Q_{\mathcal{P}} = \{q_x^{var} \mid x \in LVar_{sl}\} \cup \{q_s^{sel} \mid s \in Sel\} \cup \{q_i, q_f\}$. The transition function $\Delta_{\mathcal{P}}$ is defined as follows:

1. $(q_i, k), (q_s^{sel}, \varepsilon) \in \Delta(q_i, \sigma, \tau)$ for all $k \in \mathcal{D}_+(\mathcal{P})$, all $s \in Sel$ and all $\sigma \in \Sigma_{\mathcal{P}} \cup \{root\}$, $\tau \in \Sigma_{\mathcal{P}} \cup \{?\}$ i.e., the automaton first moves downwards chosing random directions, while in q_i, then changes to q_s^{sel} for some non-deterministically chosen selector s.
2. $(q_{\beta_s}^{var}, \varepsilon) \in \Delta(q_s^{sel}, R_{ij}^k, \tau)$ and $(q_f, \varepsilon) \in \Delta(q_\alpha^{var}, R_{ij}^k, \tau)$ for all $k \in \mathcal{D}(\mathcal{P})$ and $\tau \in \Sigma_{\mathcal{P}} \cup \{?\}$ if and only if $head(R_{ij}) \equiv \alpha \mapsto (\beta_1, \ldots, \beta_s, \ldots, \beta_m)$, for some $m > 0$ i.e., when in q_s^{sel}, the automaton starts tracking the destination β_s of the selector s through the tree. The automaton enters the final state when the tracked variable α is allocated.
3. for all $k \in \mathcal{D}_+(\mathcal{P})$, all $\ell \in \mathcal{D}(\mathcal{P})$ and all rules $R_{\ell q}$ of $P_\ell(x_{\ell,1}, \ldots, x_{\ell,n_\ell})$, we have $(q_{x_{\ell,j}}^{var}, k) \in \Delta(q_{y_j}^{var}, R_{ij}^l, \tau)$, for all $\tau \in \Sigma_{\mathcal{P}} \cup \{?\}$, and $(q_{y_j}^{var}, -1) \in \Delta(q_{x_{\ell,j}}^{var}, R_{\ell q}^k, R_{ij}^l)$ if and only if $tail(R_{ij})_k \equiv P_\ell(y_1, \ldots, y_{n_\ell})$ i.e., the automaton moves down along the k-th direction tracking $x_{\ell,j}$ instead of y_j, when the predicate $P_\ell(\mathbf{y})$ occurs on the k-th position in R_{ij}. Symmetrically, the automaton can also move up tracking y_j instead of $x_{\ell,j}$, in the same conditions.
4. $(q_\beta^{var}, \varepsilon) \in \Delta(q_\alpha^{var}, R_{ij}^k, \tau)$ for all $k \in \mathcal{D}(\mathcal{P})$ and all $\tau \in \Sigma_{\mathcal{P}} \cup \{?\}$ if and only if $\alpha = \beta$ occurs in $pure(R_{ij})$ i.e., the automaton switches from tracking α to tracking β when the equality between the two variables occurs in R_{ij}, while keeping the same position in the tree.

The following lemma formalizes the correctness of the TWA construction:

Lemma 4. *Given a system of recursive definitions $\mathcal{P}$, and an unfolding tree $t \in \mathcal{T}_i(\mathcal{P})$ of $\mathcal{P}$, rooted at i, for any $x, y \in LVar_{sl}$ and $p, r \in dom(t)$, we have $\models_{sl} \phi_t \rightarrow x^p = y^r$ if and only if $A_{\mathcal{P}}$ has a run from $\langle p, q_x^{var} \rangle$ to $\langle r, q_y^{var} \rangle$ over t, where ϕ_t is the characteristic formula of t.*

[5] This notion of tree-walking automaton is a slightly modified but equivalent to the one in [3]. We give the translation of TWA into the original definition in [12].

To the routing automaton $A_{\mathcal{P}}$ corresponds the MSO formula $\Phi_{A_{\mathcal{P}}}(r, \overrightarrow{\mathbf{X}}, T, \overrightarrow{\mathbf{Y}})$, where r maps to the root of the unfolding tree, $\overrightarrow{\mathbf{X}}$ is the sequence of second order variables X_{ij}^{k} defined previously, T maps to the domain of the tree, and $\overrightarrow{\mathbf{Y}}$ is a sequence of second-order variables X_q, one for each state $q \in Q_{\mathcal{P}}$. We denote by Y_s^{sel} and Y_f the variables from $\overrightarrow{\mathbf{Y}}$ that correspond to the states q_S^{sel} and q_f, for all $s \in Sel$, respectively. For space reasons, the definition of $\Phi_{A_{\mathcal{P}}}$ is given in [12]. With this notation, we define:

$$inner\_edges(r, \overrightarrow{\mathbf{X}}, T) \equiv \forall x \forall y \bigwedge_{s \in Sel} \exists \overrightarrow{\mathbf{Y}} \,.\, \Phi_{A_{\mathcal{P}}}(r, \overrightarrow{\mathbf{X}}, T, \overrightarrow{\mathbf{Y}}) \wedge Y_s^{sel}(x) \wedge Y_f(y) \rightarrow edge_s(x, y)$$

4.4 Double Allocation

In order to translate the definition of a recursively defined SL predicate $P(x_1, \ldots, x_n)$ into an MSO formula $\overline{P}$, that captures the models of P, we need to introduce a sanity condition, imposing that recursive predicates which establish equalities between variables allocated at different positions in the unfolding tree, are unsatisfiable, due to the semantics of the separating conjunction of SL, which implicitly conjoins all local formulae of an unfolding tree. A double allocation occurs in the unfolding tree t if and only if there exist two distinct positions $p, q \in dom(t)$ and:

1. a basic points-to formula $\alpha \mapsto (\ldots)$ occurring in $t(p)$
2. a basic points-to formula $\beta \mapsto (\ldots)$ occurring in $t(q)$
3. a path $p = p_1, \ldots, p_m = q$ in t, such that the equalities $\alpha^p = \gamma_2^{p_2} = \ldots = \gamma_{m-1}^{p_{m-1}} = \beta^q$ are all logical consequences of ϕ_t, for some tree positions $p_2, \ldots, p_{m-1} \in dom(t)$ and some variables $\gamma_2, \ldots, \gamma_{m-1} \in LVar_{sl}$

The cases of double allocation can be recognized using a routing automaton $B_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, Q'_{\mathcal{P}}, q_i, q_f, \Delta'_{\mathcal{P}})$, whose states $Q'_{\mathcal{P}} = \{q_x^{var} \mid x \in LVar_{sl}\} \cup \{q_0, q_i, q_f\}$ and transitions $\Delta'_{\mathcal{P}}$ differ from $A_{\mathcal{P}}$ only in the following rules:

– $(q_0, \varepsilon) \in \Delta(q_i, \sigma, \tau)$ for all $\sigma \in \Sigma_{\mathcal{P}} \cup \{root\}$ and all $\tau \in \Sigma_{\mathcal{P}} \cup \{?\}$, i.e. after non-deterministically chosing a position in the tree, the automaton enters a designated state q_0, which occurs only once in each run.
– $(q_\alpha^{var}, \varepsilon) \in \Delta(q_0, R_{ij}^{k}, \tau)$ for all $k \in \mathcal{D}(\mathcal{P})$ and all $\tau \in \Sigma_{\mathcal{P}} \cup \{?\}$ if and only if $head(R_{ij}) = \alpha \mapsto (\ldots)$, while in the designated state q_0, the automaton starts tracking the variable α, which is allocated at that position.

This routing automaton has a run over t, which labels one position by q_0 and a distinct one by q_f if and only if two positions in t allocate the same location. Notice that $B_{\mathcal{P}}$ has always a trivial run that starts and ends in the same position – since each position $p \in dom(t)$ allocates a variable α, and $\langle q_i, \varepsilon \rangle, \ldots, \langle q_0, p \rangle, \langle q_\alpha^{var}, p \rangle, \langle q_f, p \rangle$ is a valid run of $B_{\mathcal{P}}$. The predicate system has no double allocation if and only if these are the only possible runs of $B_{\mathcal{P}}$.

The existence of a run of $B_{\mathcal{P}}$ is captured by an MSO formula $\Phi_{B_{\mathcal{P}}}(r, \overrightarrow{\mathbf{X}}, T, \overrightarrow{\mathbf{Y}})$, where r maps to the root of the unfolding tree, $\overrightarrow{\mathbf{X}}$ is the sequence of second order variables X_{ij}^{k} defined previously, T maps to the domain of the tree, and $\overrightarrow{\mathbf{Y}}$ is the sequence of second-order variables Y_q, taken in some order, each of which maps to the set of tree positions visited by the automaton while in state $q \in Q'_{\mathcal{P}}$ – we denote by Y_0 and Y_f the variables from $\overrightarrow{\mathbf{Y}}$ that correspond to the states q_0 and q_f, respectively. Finally, we define the constraint: $no\_double\_alloc(r, \overrightarrow{\mathbf{X}}, T) \equiv \forall \overrightarrow{\mathbf{Y}} \,.\, \Phi_{B_{\mathcal{P}}}(r, \overrightarrow{\mathbf{X}}, T, \overrightarrow{\mathbf{Y}}) \rightarrow Y_0 = Y_f$

4.5 Handling Parameters

The last issue to be dealt with is the role of the actual parameters passed to a recursively defined predicate $P_i(x_{i,1}, \ldots, x_{i,k})$ of $\mathcal{P}$, in a top-level formula. Then, for each parameter $x_{i,j}$ of P_i and each unfolding tree $t \in \mathcal{T}_i(\mathcal{P})$, there exists a path $\varepsilon = p_1, \ldots, p_m \in dom(t)$ and variables $\alpha_1, \ldots, \alpha_m \in LVar_{sl}$ such that $x_{i,j} \equiv \alpha_1$ and $\alpha_\ell^{p_\ell} = \alpha_{\ell+1}^{p_{\ell+1}}$ is a consequence of ϕ_t, for all $\ell = 1, \ldots, m-1$. Subsequently, there are three (not necessarily disjoint) possibilities:

1. $head(t(p_m)) \equiv \alpha_m \mapsto (\ldots)$, i.e. α_m is allocated
2. $head(t(p_m)) \equiv \beta \mapsto (\gamma_1, \ldots, \gamma_p, \ldots, \gamma_\ell)$, and $\alpha_m \equiv \gamma_p$, i.e. α_m is referenced
3. $\alpha_m \equiv x_{i,q}$ and $p_m = \varepsilon$, for some $1 \leq q \leq k$, i.e. α_m is another parameter $x_{i,q}$

Again, we use slightly modified routing automata (one for each of the case above) $C^{i,j}_{\mathcal{P},c} = (\Sigma_{\mathcal{P}}, Q''_{\mathcal{P}}, q_i, q_f, \Delta^{i,j}_c)$ for the cases $c = 1,2,3$, respectively. Here $Q''_{\mathcal{P}} = \{q^{var}_x \mid x \in LVar_{sl}\} \cup \{q^{sel}_s \mid s \in Sel\} \cup \{q^{i,a} \mid 1 \leq a \leq k\} \cup \{q_i, q_f\}$ and $\Delta^{i,j}_c$, $c = 1,2,3$ differ from the transitions of $A_{\mathcal{P}}$ in the following:

- $(q^{i,j}, \varepsilon) \in \Delta^{i,j}_x(q_i, root, ?)$, i.e. the automaton marks the root of the tree with a designated state $q^{i,j}$, that occurs only once on each run
- $(q^{var}_{x_{i,j}}, \varepsilon) \in \Delta^{i,j}_x(q^{i,j}, R^{-1}_{ik}, ?)$, for each rule R_{ik} of P_i, i.e. the automaton starts tracking the parameter variable $x_{i,j}$ beginning with the root of the tree
- $(q_f, \varepsilon) \in \Delta^{i,j}_1(q^{var}_\alpha, R^k_{ij}, \tau)$, for all $k \in \mathcal{D}(\mathcal{P})$, $\tau \in \Sigma_{\mathcal{P}} \cup \{?\}$ iff $head(R_{ij}) \equiv \alpha \mapsto (\ldots)$ is the final rule for $C^{i,j}_{\mathcal{P},1}$
- $(q^{sel}_s, \varepsilon) \in \Delta^{i,j}_2(q^{var}_\gamma, R^k_{ij}, \tau)$, for all $k \in \mathcal{D}(\mathcal{P})$ and $\tau \in \Sigma_{\mathcal{P}} \cup \{?\}$ iff $head(R_{ij}) \equiv \alpha \mapsto (\beta_1, \ldots, \beta_s, \ldots, \beta_n)$ and $\gamma \equiv \beta_s$ i.e., q^{sel}_s is reached in the second case, when the tracked variable is referenced. After that, $C^{i,j}_{\mathcal{P},2}$ moves to the final state i.e., $(q_f, \varepsilon) \in \Delta^{i,j}_2(q^{sel}_s, \sigma, \tau)$ for all $s \in Sel$, all $\sigma \in \Sigma_{\mathcal{P}} \cup \{root\}$ and $\tau \in \Sigma_{\mathcal{P}} \cup \{?\}$
- $(q^{i,a}, \varepsilon) \in \Delta^{i,j}_3(q^{var}_{x_{i,a}}, root, ?)$ and $(q_f, \varepsilon) \in \Delta^{i,j}_3(q_{i,a}, root, ?)$, for each $1 \leq a \leq k$ and $a \neq j$ i.e., are the final moves for $C^{i,j}_{\mathcal{P},3}$

The outcome of this construction are MSO formulae $\Phi_{C^{i,j}_{\mathcal{P},c}}(r, \overrightarrow{\mathbf{X}}, T, \overrightarrow{\mathbf{Y}})$, for $c = 1,2,3$, where r maps to the root of the unfolding tree, respectively, $\overrightarrow{\mathbf{X}}$ is the sequence of second order variables X^k_{ij} defined previously, T maps to the domain of the tree, and $\overrightarrow{\mathbf{Y}}$ is the sequence of second order variables corresponding to states of $Q''_{\mathcal{P}}$ – we denote by $Y_f, Y^{i,a}, Y^{sel}_s \in \overrightarrow{\mathbf{Y}}$ the variables corresponding to the states q_f, $q^{i,a}$, and q^{sel}_s, respectively.

The parameter $x_{i,j}$ of P_i is assigned by the following MSO constraints:

$$\begin{aligned}
param^1_{i,j}(r,\overrightarrow{\mathbf{X}},T) &\equiv \exists\overrightarrow{\mathbf{Y}} \;.\; \Phi_{C^{i,j}_{\mathcal{P},1}} \;\wedge\; Y^{i,j}_0(\bar{x}_{i,j}) \;\wedge\; \forall y \;.\; Y_f(y) \rightarrow \bar{x}_{i,j} = y \\
param^2_{i,j}(r,\overrightarrow{\mathbf{X}},T) &\equiv \exists\overrightarrow{\mathbf{Y}} \;.\; \Phi_{C^{i,j}_{\mathcal{P},2}} \;\wedge\; Y^{i,j}_0(\bar{x}_{i,j}) \;\wedge\; \textstyle\bigwedge_{s\in Sel} \forall y \;.\; Y^{sel}_s(y) \rightarrow edge_s(y,\bar{x}_{i,j}) \\
param^3_{i,j}(r,\overrightarrow{\mathbf{X}},T) &\equiv \exists\overrightarrow{\mathbf{Y}} \;.\; \Phi_{C^{i,j}_{\mathcal{P},3}} \;\wedge\; Y^{i,j}_0(\bar{x}_{i,j}) \;\wedge\; \textstyle\bigwedge_{1\le a\le k} \forall y \;.\; Y^{i,a}(y) \rightarrow \bar{x}_{i,j} = \bar{x}_{i,a}
\end{aligned}$$

where $\bar{x}_{i,j}$ is the first-order MSO variable corresponding to the SL parameter $x_{i,j}$. Finally, the constraint $param_{i,j}$ is conjunction of the $param^c_{i,j}$, $c = 1,2,3$ formulae.

4.6 Translating Top Level SLRD$_{btw}$ Formulae to MSO

We define the MSO formula corresponding to a predicate $P_i(x_{i,1},\ldots,x_{i,n_i})$, of a system of recursive definitions $\mathcal{P} = \{P_1,\ldots,P_n\}$:

$$\begin{aligned}
\overline{P_i}(\bar{x}_{i,1},\ldots,\bar{x}_{i,n_i},T) \equiv \exists r \exists \overrightarrow{\mathbf{X}} \;.\; & backbone_i(r,\overrightarrow{\mathbf{X}},T) \;\wedge\; inner\_edges(r,\overrightarrow{\mathbf{X}},T) \;\wedge \\
& no\_double\_alloc(r,\overrightarrow{\mathbf{X}},T) \;\wedge\; \textstyle\bigwedge_{1\le j\le n_i} param_{i,j}(r,\overrightarrow{\mathbf{X}},T)
\end{aligned}$$

The following lemma is needed to establish the correctness of our construction.

Lemma 5. *For any state $S = \langle s,h\rangle$, any interpretation $\iota : LVar_{sl} \rightarrow_{fin} Loc$, and any recursively defined predicate $P_i(x_1,\ldots,x_n)$, we have $S,\iota \models_{sl} P_i(x_1,\ldots,x_n)$ if and only if $S,\bar{\iota},\nu[T \leftarrow dom(h)] \models_{mso} \overline{P_i}(\overline{x_1},\ldots,\overline{x_k},T) \wedge Heap(T)$, where $\bar{\iota} : LVar_{mso} \rightharpoonup_{fin} Loc$ is an interpretation of first order variables, such that $\bar{\iota}(x_u) = s(u)$, for all $u \in PVar$, and $\bar{\iota}(\bar{x}) = \iota(x)$, for all $x \in LVar_{sl}$, and $\nu : LVar_{mso} \rightharpoonup_{fin} 2^{Loc}$ is any interpretation of second-order variables.*

Recall that a top level SLRD$_{btw}$ formula is of the form: $\varphi \equiv \exists \mathbf{z} \;.\; \phi(\mathbf{y}_0) * P_{i_1}(\mathbf{y}_1) * \ldots P_{i_k}(\mathbf{y}_k)$, where $1 \le i_1,\ldots,i_k \le n$, and $\mathbf{y}_j \subseteq \mathbf{z}$, for all $j = 0,1,\ldots,k$. We define the MSO formula:

$$\overline{\varphi}(X) \equiv \exists\overline{\mathbf{z}}\exists X_{0,\ldots,k} \;.\; \overline{\phi}(\overline{\mathbf{y}_0},X_0) \;\wedge\; \overline{P_{i_1}}(\overline{\mathbf{y}_1},X_1) \;\wedge\; \ldots \;\wedge\; \overline{P_{i_k}}(\overline{\mathbf{y}_k},X_k) \;\wedge\; \Pi(X_0,X_1,\ldots,X_k,X)$$

Theorem 3. *For any state S and any closed SLRD$_{btw}$ formula φ we have that $S \models_{sl} \varphi$ if and only if $S \models_{mso} \exists X \;.\; \overline{\varphi}(X) \;\wedge\; Heap(X)$.*

Theorem 2 and the above theorem prove decidability of satisfiability and entailment problems for SLRD$_{btw}$, by reduction to MSO over states of bounded tree width.

5 Conclusions and Future Work

We defined a fragment of Separation Logic with Recursive Definitions, capable of describing general unbounded mutable data structures, such as trees with parent pointers and linked leaves. The logic is shown to be decidable for satisfiability and entailment, by reduction to MSO over graphs of bounded tree width. We conjecture that the complexity of the decision problems for this logic is elementary, and plan to compute tight upper bounds, in the near future.

Acknowledgement. This work was supported by the Czech Science Foundation (project P103/10/0306) and French National Research Agency (project VERIDYC ANR-09-SEGI-016). We also acknowledge Tomáš Vojnar, Lukáš Holík and the anonymous reviewers for their valuable comments.

References

1. Berdine, J., Calcagno, C., Cook, B., Distefano, D., O'Hearn, P., Wies, T., Yang, H.: Shape analysis for composite data structures. In: Proc. CAV'07. LNCS, vol. 4590. Springer (2007)
2. Berdine, J., Calcagno, C., O'Hearn, P.W.: A decidable fragment of separation logic. In: Proc. of FSTTCS'04. LNCS, vol. 3328. Springer (2004)
3. Bojanczyk, M.: Tree-walking automata. In: Proc. of LATA'08. LNCS, vol. 5196. Springer (2008)
4. Bouajjani, A., Dragoi, C., Enea, C., Sighireanu, M.: A logic-based framework for reasoning about composite data structures. In: Proc. of CONCUR'09. LNCS, vol. 5710. Springer (2009)
5. Bozga, M., Iosif, R., Perarnau, S.: Quantitative separation logic and programs with lists. J. Autom. Reasoning 45(2), 131–16o (2010)
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. pp. 130–139. LICS '10 (2010)
7. Calcagno, C., Distefano, D.: Infer: An automatic program verifier for memory safety of c programs. In: Proc. of NASA Formal Methods'11. LNCS, vol. 6617. Springer (2011)
8. Cook, B., Haase, C., Ouaknine, J., Parkinson, M.J., Worrell, J.: Tractable reasoning in a fragment of separation logic. In: Proc. of CONCUR'11. LNCS, vol. 6901. Springer (2011)
9. Dudka, K., Peringer, P., Vojnar, T.: Predator: A practical tool for checking manipulation of dynamic data structures using separation logic. In: Proc. of CAV'11. LNCS, vol. 6806. Springer (2011)
10. Enea, C., Saveluc, V., Sighireanu, M.: Compositional invariant checking for overlaid and nested linked lists. In: Proc. of ESOP'13. pp. 129–148 (2013)
11. Immerman, N., Rabinovich, A.M., Reps, T.W., Sagiv, S., Yorsh, G.: The boundary between decidability and undecidability for transitive-closure logics. In: Proc of CSL'04. LNCS, vol. 3210. Springer (2004)
12. Iosif, R., Rogalewicz, A., Simacek, J.: The tree width of separation logic with recursive definitions. CoRR abs/1301.5139 (2013)
13. Madhusudan, P., Parlato, G.: The tree width of auxiliary storage. In: Proc. of POPL'11. ACM (2011)
14. Madhusudan, P., Parlato, G., Qiu, X.: Decidable logics combining heap structures and data. In: Proc. of POPL'11 (2011)
15. Møller, A., Schwartzbach, M.I.: The pointer assertion logic engine. In: Proc. of PLDI'01 (June 2001)
16. Nguyen, H.H., Chin, W.N.: Enhancing program verification with lemmas. In: Proc of CAV'08. LNCS, vol. 5123. Springer (2008)
17. Reynolds, J.: Separation Logic: A Logic for Shared Mutable Data Structures. In: Proc. of LICS'02. IEEE CS Press (2002)
18. Seese, D.: The structure of models of decidable monadic theories of graphs. Annals of Pure and Applied Logic 53(2), 169–195 (1991)
19. Yorsh, G., Rabinovich, A.M., Sagiv, M., Meyer, A., Bouajjani, A.: A logic of reachable patterns in linked data-structures. In: Proc. of FoSSaCS'06. LNCS, vol. 3921. Springer (2006)