Tree-Like Grammars and Separation Logic
(Extended Abstract)

Christoph Matheja, Christina Jansen, and Thomas Noll

Software Modeling and Verification Group
RWTH Aachen University, Germany
http://moves.rwth-aachen.de/

Abstract. Separation Logic with inductive predicate definitions (SL) and hyperedge replacement grammars (HRG) are established formalisms to describe the abstract shape of data structures maintained by heap-manipulating programs. Fragments of both formalisms are known to coincide, and neither the entailment problem for SL nor its counterpart for HRGs, the inclusion problem, are decidable in general.
We introduce *tree-like grammars* (TLG), a fragment of HRGs with a decidable inclusion problem. By the correspondence between HRGs and SL, we simultaneously obtain an equivalent SL fragment ($\mathtt{SL}_{\mathtt{tl}}$) featuring some remarkable properties including a decidable entailment problem.

1 Introduction

Symbolic execution of heap-manipulating programs builds upon abstractions to obtain finite descriptions of dynamic data structures, like linked lists and trees. Proposed abstraction approaches employ, amongst others, Separation Logic with inductive predicate definitions (SL) [21, 2, 16] and hyperedge replacement grammars (HRG) [12, 15].

While these formalisms are intuitive and expressive, important problems are undecidable. In particular, the *entailment problem* for SL [5, 1], i.e. the question whether all models of a formula φ are also models of another formula ψ, as well as its graph-theoretical counterpart, the *inclusion problem* for HRGs [12], are undecidable in general. Unfortunately, as stated by Brotherston, Distefano and Peterson [4], "effective procedures for establishing entailments are at the foundation of automatic verification based on Separation Logic". Consequently, SL-based verification tools, such as SLayer [3] and Predator [13], often restrict themselves to the analysis of list-like data structures, where the entailment problem is known to be decidable [2]. VeriFast [17], Hip/Sleek [7] and Cyclist [4] allow general user-specified predicates, but are incomplete and/or require additional user interaction. The largest known fragment of SL featuring both inductive predicate definitions and a decidable entailment problem is Separation Logic with bounded tree width ($\mathtt{SL}_{\mathtt{btw}}$) [16].

Approaches based on graph grammars suffer from the undecidability of the related inclusion problem: Lee et al. [20] propose the use of graph grammars for

shape analysis, but their approach is restricted to trees. JUGGRNAUT [15] allows the user to specify the shape of dynamic data structures by an HRG, but relies on an approximation to check whether newly computed abstractions are subsumed by previously encountered ones. Hence, finding more general fragments of SL and HRGs with good decidability properties is highly desirable.

This paper investigates fragments of HRGs with a decidable inclusion problem. In a nutshell, HRGs are a natural extension of context-free word grammars specifying the replacement of nonterminal-labelled edges by graphs (cf. [14]). Common notions and results for context-free word languages, e.g. decidability of the emptiness problem and existence of derivation trees, can be lifted to HRGs (cf. [22]) which justifies the alternative name "context-free graph grammars".

Most of our results stand on two pillars. To introduce these two pillars as well as to summarise our main results here, the utilisation of some formal notation and concepts is indispensable. Corresponding definitions and detailed explanations can be found in the successive sections. The first pillar is an extension of the well-known fact that context-free word languages are closed under intersection with regular word languages, which are, by Büchi's famous theorem [6], exactly the word languages definable in monadic second-order logic (MSO).

Lemma 1 (Courcelle [8]) *For each HRG G and $\mathtt{MSO}_2$ sentence φ, one can construct an HRG G' such that $L(G') = L(G) \cap L(\varphi) = \{H \in L(G) \mid \underline{H} \models \varphi\}$.*

Here, $\mathtt{MSO}_2$ means MSO over graphs with quantification over nodes and edges.

The second pillar is the close connection between a fragment of HRGs - called data structure grammars (DSG) - and a fragment of SL studied by Dodds [11] and Jansen et al. [18].

Lemma 2 (Jansen et al. [18]) *Every SL formula can be translated into a language-equivalent data structure grammar and vice versa.*

The overall goal of this paper is to develop fragments of HRGs which can be translated into $\mathtt{MSO}_2$. Then it directly follows from Lemma 1 that the resulting classes of languages have a decidable inclusion problem and are closed under union, intersection and difference as well as under intersection with general context-free graph languages. By Lemma 2, we obtain analogous results for equivalent SL fragments.

The largest fragment we propose are *tree-like grammars* (TLG). Intuitively, every graph H generated by a TLG allows to reconstruct one of its derivation trees by identifying certain nodes, the *anchor* nodes, with positions in a derivation tree. Furthermore, each edge of H is uniquely associated with one of these anchor nodes. These properties allow for each graph H generated by a given TLG G to first encode a derivation tree t in $\mathtt{MSO}_2$ and then to verify that H is in fact the graph derived by G according to t. Our main result is that the two informally stated properties from above guarantee $\mathtt{MSO}_2$-definability.

Theorem 1. *For each TLG G, there exists an $\mathtt{MSO}_2$ sentence φ_G such that for each $H \in \mathtt{HG}$, $H \in L(G)$ if and only if $\underline{H} \models \varphi_G$.*

TLGs are introduced in detail in Section 4. Furthermore, we study the fragment of *tree-like Separation Logic* ($\mathtt{SL}_{\mathtt{tl}}$, cf. Section 5) which is equivalent to TLGs generating heaps rather than arbitrary graphs.

Definition 1. *An $\mathtt{SL}_{tl}$ environment is an $\mathtt{SL}$ environment Γ where every disjunct $\varphi(x_1, ..., x_n)$ of every predicate definition meets the following conditions:*

- Anchoredness*: All pointer assertions $y.s \mapsto z$ occurring in φ contain the first parameter x_1 of φ, either on their left-hand or right-hand side, i.e. $x_1 = y$ or $x_1 = z$.*
- Connectedness*: The first parameter of every predicate call in φ occurs in some pointer assertion of φ.*
- Distinctness*: x_1 is unequal to the first parameter of every predicate call occurring in Γ.*

By Lemma 2, our results on TLGs also hold for $\mathtt{SL}_{\mathtt{tl}}$. Thus, $\mathtt{SL}_{\mathtt{tl}}$ has the following remarkable properties:

1. The satisfiability as well as the *extended* entailment problem, i.e. the question whether an *arbitrary* $\mathtt{SL}$ formula φ entails an $\mathtt{SL}_{\mathtt{tl}}$ formula ψ, are decidable.
2. Although negation and conjunction are restricted to pure formulae, $\mathtt{SL}_{\mathtt{tl}}$ is closed under intersection and difference.

Regarding expressiveness, common data structures like (cyclic) lists, trees, in-trees, $n \times k$-grids for fixed k and combinations thereof are $\mathtt{SL}_{\mathtt{tl}}$-definable. In particular, we show that $\mathtt{SL}_{\mathtt{tl}}$ is strictly more expressive than $\mathtt{SL}_{\mathtt{btw}}$. The same holds for an entirely syntactic fragment of TLGs, called Δ-DSGs, and a corresponding fragment of $\mathtt{SL}_{\mathtt{tl}}$.

A full version of this paper containing further details and proofs has recently been submitted to APLAS[1].

The remainder of this paper is structured as follows. Section 2 very briefly recapitulates standard definitions on $\mathtt{SL}$ and $\mathtt{MSO}$, while Section 3 covers essential concepts of hypergraphs and HRGs. The fragment of TLGs and its $\mathtt{MSO}$-definability result is introduced in Section 4. Our results on TLGs are transferred to $\mathtt{SL}$ and discussed in Section 5. Finally, Section 6 concludes.

2 Preliminaries

This section introduces our notation and briefly recapitulates trees, graphs, $\mathtt{MSO}_2$, and $\mathtt{SL}$. On first reading, the well-informed reader might want to skip this part.

[1] `http://pl.postech.ac.kr/aplas2015/`

Notation Given a set S, $S^\star$ denotes all finite sequences over S. For $s, s' \in S^\star$, $s.s'$ denotes their concatenation, the i-th element of s is denoted by $s(i)$ and the set of all of its elements is denoted by $\lfloor s \rfloor$. A ranked alphabet is a finite set S with ranking function $rk_S : S \to \mathbb{N}$ and maximal rank $\Re(S)$. We write $\{x_1 \mapsto y_1, \ldots, x_m \mapsto y_m\}$ to denote a finite (partial) function f with domain $\mathrm{dom}(f) = \{x_1, \ldots, x_m\}$ and co-domain $\{y_1, \ldots, y_m\}$ such that $f(x_i) = y_i$ for each $i \in [m] = [1, m] = \{1, 2, \ldots, m\}$. The operators $\uplus$ and $\biguplus$ denote the disjoint union of two sets and two functions, respectively.

Trees Given a ranked alphabet S, a *tree* over S is a finite function $t : \mathrm{dom}(t) \to S$ such that $\emptyset \neq \mathrm{dom}(t) \subseteq \mathbb{N}^\star$, $\mathrm{dom}(t)$ is prefix closed and for all $x \in \mathrm{dom}(t)$, $\{i \in \mathbb{N} \mid x.i \in \mathrm{dom}(t)\} = [rk_S(t(x))]$. $x \in \mathrm{dom}(t)$ is a (proper) prefix of $y \in \mathrm{dom}(t)$, written $x \prec y$, if $y = x.i.z$ for some $i \in \mathbb{N}$ and $z \in \mathbb{N}^\star$. The subtree of t with root $x \in \mathrm{dom}(t)$ is given by $t|_x : \{y \mid x.y \in \mathrm{dom}(t)\} \to S : y \mapsto t(x.y)$.

Graphs An edge-labelled graph over an alphabet S is a tuple $H = (V, E)$ with a finite set of nodes V and edge relation $E \subseteq V \times S \times V$. With each graph H we associate the relational structure $\underline{H} = (V \uplus E, \mathrm{src}, \mathrm{tgt}, (E_s)_{s \in S})$ where src and tgt are the binary source and target relations given by $\mathrm{src} := \{(u, e) \mid e = (u, s, v) \in E\}$, $\mathrm{tgt} := \{(e, v) \mid e = (u, s, v) \in E\}$. For each $s \in S$, there is a unary relation $E_s := \{(u, s, v) \in E \mid u, v \in V\}$ collecting all edges labelled with s.

Monadic Second-Order Logic over Graphs Given a finite alphabet S, the syntax of $\mathtt{MSO}_2$ is given by:

$$\varphi ::= E_s(x) \mid \mathrm{src}(x, y) \mid \mathrm{tgt}(x, y) \mid X(x) \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \exists x : \varphi \mid \exists X : \varphi \mid x = y$$

where x, y are first-order variables, X is a second-order variable and $s \in S$. For a graph $H = (V, E)$, we write $\underline{H}, \jmath \models \varphi$ iff $\underline{H}$ satisfies φ where $\jmath$ is an interpretation mapping every free first-order variable to an element of $V \uplus E$ and every second-order variable to a subset of either V or E, respectively. The semantics of $\models$ is standard (cf. [10]). Note that the semantics of $\mathrm{src}, \mathrm{tgt}$ and E_s has been given explicitly in the definition of $\underline{H}$.

Heaps Similarly to the typical RAM model, a heap is understood as a set of locations *Loc*, whose values are interpreted as pointers to other locations. Formally, we assume $Loc := \mathbb{N}$ and define a *heap* as a partial mapping $h : Loc \to Loc \uplus \{\mathbf{null}\}$. The set of all heaps is denoted by $\mathtt{He}$. Let Σ be a finite alphabet of *selectors* equipped with an injective ordering function $\mathrm{cn} : \Sigma \to [0, |\Sigma| - 1]$. We assume a heap to consist of objects equipped with finitely many pointer variables which are modelled by reserving exactly $|\Sigma|$ successive locations. Hence, for a heap containing n objects, $dom(h) = [n \cdot |\Sigma|]$.

Separation Logic with Recursive Definitions We consider a fragment of Separation Logic, similar to Separation Logic with recursive definitions in [16, 18], in

which negation $\neg$, $\mathtt{true}$, and conjunction $\wedge$ in spatial formulae are disallowed. Let *Pred* be a set of predicate names. The *syntax of* $\mathtt{SL}$ is given by:

$$
\begin{array}{lr}
E ::= x \mid \mathbf{null} & \\
P ::= x = y \mid P \wedge P & \text{pure formulae} \\
F ::= \mathbf{emp} \mid x.s \mapsto E \mid F * F \mid \exists x : F \mid \sigma(x_1, ..., x_n) & \text{spatial formulae} \\
S ::= F \mid S \vee S \mid S \wedge P & \mathtt{SL} \text{ formulae}
\end{array}
$$

where $x, y, x_1, ..., x_n \in \mathit{Var}$, $s \in \Sigma$ and $\sigma \in \mathit{Pred}$. The formula $x.s \mapsto E$ is called a *pointer assertion*, $\sigma(x_1, ..., x_n)$ a *predicate call*.

Note that we do not require that all selectors of a given variable are defined by a single pointer assertion, i.e. we are less strict about pointer assertions than other fragments proposed in the literature, e.g. in [16]. Furthermore, it is straightforward to add program variables to $\mathtt{SL}$, which we omitted for the sake of simplicity. To improve readability, we write $x.(s_1, \ldots, s_k) \mapsto (y_1, \ldots, y_k)$ as a shortcut for $x.s_1 \mapsto y_1 * \ldots * x.s_k \mapsto y_k$.

Predicate calls are interpreted by means of predicate definitions. A *predicate definition* for $\sigma \in \mathit{Pred}$ is of the form $\sigma(x_1, ..., x_n) := \sigma_1 \vee ... \vee \sigma_m$ where $m, n \in \mathbb{N}$, σ_j is a formula of the form $F \wedge P$, and $x_1, ..., x_n \in \mathit{Var}$ are pairwise distinct and exactly the free variables of σ_j for each $j \in [m]$. The disjunction $\sigma_1 \vee ... \vee \sigma_m$ is called the *body* of the predicate. An *environment* is a set of predicate definitions. *Env* denotes the set of all environments.

The semantics of a predicate call $\sigma(x_1, ..., x_n)$, $\sigma \in \mathit{Pred}$, w.r.t. an environment $\Gamma \in \mathit{Env}$ is given by the predicate interpretation η_Γ. It is defined as the least set of location sequences instantiating the arguments $x_1, \ldots, x_n$ and heaps that fulfil the unrolling of the predicate body. We refer to [18] for a formal definition.

The semantics of the remaining $\mathtt{SL}$ constructs is determined by the standard semantics of first-order logic and the following, where $\jmath$ is an interpretation of variables as introduced for $\mathtt{MSO}_2$:

$$
\begin{array}{ll}
h, \jmath, \eta_\Gamma \models x.s \mapsto \mathbf{null} & \Leftrightarrow \mathrm{dom}(h) = \{\jmath(x) + cn(s)\}, h(\jmath(x) + cn(s)) = \mathbf{null} \\
h, \jmath, \eta_\Gamma \models x.s \mapsto y & \Leftrightarrow \mathrm{dom}(h) = \{\jmath(x) + cn(s)\}, h(\jmath(x) + cn(s)) = \jmath(y) \\
h, \jmath, \eta_\Gamma \models \sigma(x_1, ..., x_n) & \Leftrightarrow ((\jmath(x_1), ..., \jmath(x_n)), h) \in \eta_\Gamma(\sigma) \\
h, \jmath, \eta_\Gamma \models \varphi_1 * \varphi_2 & \Leftrightarrow \exists h_1, h_2 : h = h_1 \uplus h_2, h_1, \jmath, \eta_\Gamma \models \varphi_1,\ h_2, \jmath, \eta_\Gamma \models \varphi_2
\end{array}
$$

A variable $x \in \mathit{Var}$ is said to be *allocated* in a formula if it (or a variable y with $y = x$) occurs on the left-hand side of a pointer assertion.

From now on, we assume that all existentially quantified variables are eventually allocated. This requirement is similar to the "establishment" condition in [16]. With this assumption, the inequality operator for logical variables $x \neq y$ is redundant with respect to the expressive power of the formalism, because $x.s \mapsto z * y.s \mapsto z$ already implies that $\jmath(x) \neq \jmath(y)$ in all heaps satisfying the formula. Thus, we assume that two existentially quantified variables refer to different locations if not stated otherwise by a pure formula.

3 Context-Free Graph Grammars

This section introduces HRGs together with some of their properties relevant for the remainder of this paper. For a comprehensive introduction, we refer to [14, 22].

Let $\Sigma_N := \Sigma \uplus N$ be a ranked alphabet consisting of terminal symbols Σ and nonterminal symbols N.

Definition 2 (Hypergraph). *A* labelled hypergraph *(HG) over Σ_N is a tuple $H = (V, E, \mathrm{att}, \mathrm{lab}, \mathrm{ext})$ where V and E are disjoint sets of nodes and hyperedges,* $\mathrm{att} : E \to V^\star$ *maps each hyperedge to a sequence of attached nodes such that* $|\mathrm{att}(e)| = rk_{\Sigma_N}(\mathrm{lab}(e))$, $\mathrm{lab} : E \to \Sigma_N$ *is a labelling function, and* $\mathrm{ext} \in V^\star$ *a sequence of external nodes. The set of all HGs over Σ_N is denoted by $\mathtt{HG}_{\Sigma_N}$.*

Note that we allow attachments of hyperedges as well as the sequence of external nodes to contain repetitions. Hyperedges with a label from Σ are called *terminal edges*, *nonterminal* otherwise. The set of terminal (nonterminal) hyperedges of an HG H is denoted by E_H^Σ (E_H^N, respectively). In this paper, we assume $rk_{\Sigma_N}(s) = 2$ for each $s \in \Sigma$. Moreover, a hyperedge e with $\mathrm{lab}(e) = s \in \Sigma$ and $\mathrm{att}(e) = u.v$ is interpreted as a directed edge from u to v. The relational structure corresponding to $H \in \mathtt{HG}_\Sigma$ is $\underline{H} := \underline{[H]}$, where the (conventional) graph $[H]$ is defined as $[H] = (V_H, E)$, $E := \{(\mathrm{att}_H(e)(1), \mathrm{lab}_H(e), \mathrm{att}_H(e)(2)) \mid e \in E_H\}$.

Example 1. As an example, consider the HG illustrated in Figure 1(a). For referencing purpose, we provide a unique index $i \in [|V|]$ inside of each node u_i represented by a circle. External nodes are shaded. For simplicity, we assume them to be ordered according to the provided index. Terminal edges are drawn as directed, labelled edges and nonterminal edges as square boxes with their label inside. The ordinals pictured next to the connections of a nonterminal hyperedge denote the position of the attached nodes in the attachment sequence. For example, if e is the leftmost nonterminal hyperedge in Figure 1(a), $\mathrm{att}(e) = u_5.u_1.u_3.u_7$.

Two HGs H, H' are isomorphic, written $H \cong H'$, if they are identical up to renaming of nodes and edges. In this paper, we will not distinguish between isomorphic HGs. The disjoint union of $H, H' \in \mathtt{HG}_{\Sigma_N}$ is denoted by $H \uplus H'$.

The main concept to specify (infinite) sets of HGs in terms of context-free graph grammars is the replacement of a nonterminal hyperedge by a finite HG. Intuitively, a nonterminal hyperedge e is replaced by an HG H by first removing e, inserting a disjoint copy of H and identifying the nodes originally attached to e with the sequence of external nodes of H. This is formally expressed by a quotient.

Definition 3 (Hypergraph Quotient). *Let $H \in \mathtt{HG}_{\Sigma_N}$, $R \subseteq V_H \times V_H$ be an equivalence relation and $[u]_{/R} = \{v \in V_H \mid (u, v) \in R\}$ the equivalence class of $u \in V_H$, which is canonically extended to sequences of nodes. The R-quotient graph of H is $[H]_{/R} = (V, E, \mathrm{att}, \mathrm{lab}, \mathrm{ext})$, where $V = \{[u]_{/R} \mid u \in V_H\}$, $E = E_H$,* $\mathrm{att} = \{e \mapsto [\mathrm{att}_H(e)]_{/R} \mid e \in E_H\}$, $\mathrm{lab} = \mathrm{lab}_H$, $\mathrm{ext} = [\mathrm{ext}_H]_{/R}$.

Definition 4 (Hyperedge Replacement). *Let $H, K \in \mathit{HG}_{\Sigma_N}$ be hypergraphs with disjoint nodes and hyperedges, $e \in E_H^N$ with $rk_{\Sigma_N}(e) = k = |\mathrm{ext}_K|$. Let $V = V_H \uplus V_K$, and ${}_{H,e}{\approx_K} \subseteq V \times V$ be the least equivalence relation containing $\{(\mathrm{att}_H(e)(i), \mathrm{ext}_K(i)) \mid i \in [k]\}$. Then the HG obtained from replacing e by K is $H[e/K] := [(H \setminus \{e\} \uplus K)]_{/\,{}_{H,e}{\approx_K}}$ where $H \setminus \{e\}$ is the HG H in which e has been removed. Moreover, two nodes $u, v \in V$ are* merged *by $H[e/K]$ if $u \neq v$ and $u\ {}_{H,e}{\approx_K}\ v$.*

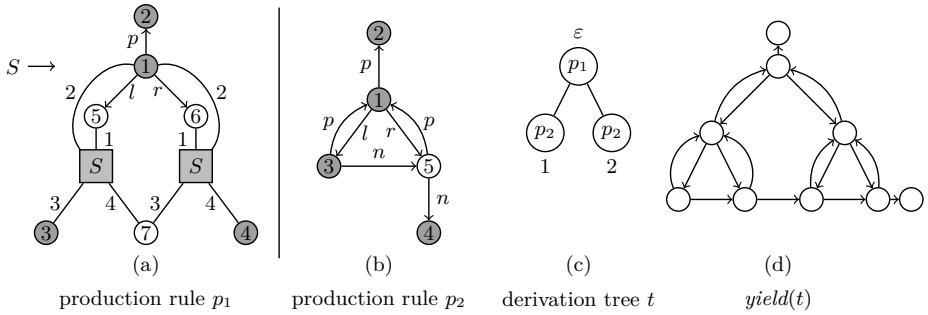

Fig. 1. HRG *TLL* with two production rules p_1 and p_2

We now formally introduce context-free graph grammars based on hyperedge replacement.

Definition 5 (Hyperedge Replacement Grammar). *An HRG is a 3-tuple $G = (\Sigma_N, P, S)$ where Σ_N is a ranked alphabet, $S \in N$ is the initial symbol and $P \subseteq N \times \mathit{HG}_{\Sigma_N}$ is a finite set of production rules such that $rk_{\Sigma_N}(X) = |\mathrm{ext}_H| > 0$ for each $(X, H) \in P$. The class of all HRGs is denoted by HRG.*

Given $p = (X, H) \in P$, we write $\mathrm{lhs}(p)$ and $\mathrm{rhs}(p)$ to denote X and H, respectively. To improve readability, we write p instead of $\mathrm{lhs}(p)$ or $\mathrm{rhs}(p)$ whenever the context is clear.

Example 2. The HRG *TLL* depicted in Figure 1(a),(b) will serve as a running example. It consists of one nonterminal symbol S, four terminal symbols l, r, p, n and two production rules p_1, p_2.

A key feature of HRGs is that the order in which nonterminal hyperedges are replaced is irrelevant, i.e. HRGs are confluent (cf. [14, 22]). Thus, derivations of HRGs can be described by derivation trees. Towards a formal definition, we assume that the nonterminal hyperedges $E_p^N = \{e_1, ..., e_n\}$ of each production rule $p = (X, H)$ are in some (arbitrary, but fixed) linear order, say $e_1, ..., e_n$. For HRG G, $G[X]$ denotes the HRG (Σ_N, P_G, X).

Definition 6 (Derivation Tree). *Let $G = (\Sigma_N, P, S) \in \mathit{HRG}$. The set of all* derivation trees *of G is the least set $\mathit{D}(G)$ of trees over the alphabet P with ranking function $rk_P : P \to \mathbb{N}$ such that $t(\varepsilon) = p$ for some $p \in P$ with $\mathrm{lhs}(p) = S$. Moreover, if $E_p^N = \{e_1, \ldots, e_m\}$, then $rk_P(p) = m$ and $t|_i \in \mathit{D}(G[\mathrm{lab}_p(e_i)])$ for each $i \in [m]$. The* yield *of a derivation tree is given by the HG*

$$\mathit{yield}(t) = t(\varepsilon)[e_1/\mathit{yield}(t|_1), \ldots, e_m/\mathit{yield}(t|_m)].$$

We implicitly assume that the nodes and hyperedges of $t(x)$ and $t(y)$ are disjoint if $x \neq y$. The yield of a derivation tree is also called the *derived* HG according to t.

Example 3. Figure 1(c) illustrates a derivation tree t of the HRG *TLL* in which production rule p_1 has been applied once, and production rule p_2 twice. The labels next to the circles provide the position in $\mathrm{dom}(t)$ while the labels inside indicate the applied production rule. The graph on the right (d) illustrates the shape of $\mathit{yield}(t)$. For simplicity, node indices as well as edge labels are omitted.

The language generated by an HRG consists of all HGs without nonterminal edges that can be derived from the initial nonterminal symbol.

Definition 7 (HR Language). *The language generated by $G \in \mathtt{HRG}$ is the set $L(G) = \{\mathit{yield}(t) \mid t \in \mathtt{D}(G)\}$.*

Example 4. The HRG *TLL*, provided in Figure 1, generates the set of all fully-branched binary trees in which the leaves are connected from left to right and each node has an additional edge to its parent.

Two results for derivation trees are needed in the following. The first result is directly lifted from analogous results for context-free word grammars (cf. [22] below Theorem 3.10).

Lemma 3 *For each $G \in \mathtt{HRG}$, $\mathtt{D}(G)$ is a regular tree language. In particular, the emptiness problem for HRGs is decidable in linear time.*

Furthermore, we generalize the notion of merged nodes to multiple successive applications of hyperedge replacement.

Definition 8 (Merged Nodes). *Let $G \in \mathtt{HRG}$, $t \in \mathtt{D}(G)$, $x, y \in \mathrm{dom}(t)$ such that $x \prec y$, i.e. $y = x.i.z$ for some $i \in \mathbb{N}, z \in \mathbb{N}^\star$, and let $u \in V_{t(x)}$, $v \in V_{t(y)}$. We say that u and v are merged in t, written $u \sim_t v$, if*

- *$z = \varepsilon$ and $u \; {}_{t(x),e_i}{\approx}_{t(x.i)} \; v$, or*
- *$z \neq \varepsilon$ and there exists $w \in V_{t(x.i)}$ such that $u \; {}_{t(x),e_i}{\approx}_{t(x.i)} \; w$ and $w \sim_t v$.*

Example 5. Consider the derivation tree t shown in Figure 1(c) again. In its yield, the node u_7 in $t(\varepsilon)$ is merged with u_4 in $t(1)$ and with u_3 in $t(2)$. In $\mathit{yield}(t)$, this node represents the leftmost leaf of the right subtree.

The relation $\sim_t$ merges exactly the nodes that are identified with each other by $\mathit{yield}(t)$.

Lemma 4 (Merge Lemma) *Given $G \in \mathtt{HRG}$ and $t \in \mathtt{D}(G)$, let $\simeq_t$ denote the least equivalence relation containing $\sim_t$. Then*

$$\mathit{yield}(t) \cong \left[\biguplus_{x \in \mathrm{dom}(t)} \mathrm{rhs}(t(x)) \right]_{/\simeq_t} .$$

4 Tree-Like Grammars

This section introduces tree-like grammars (TLG), a fragment of HRGs which can be translated into $\mathtt{MSO}_2$.

Some further notation is needed. Let $H \in \mathtt{HG}_{\Sigma_N}$ with $E_H^N = \{e_1, \ldots, e_m\}$. We call $\mathrm{ext}_H(1)$ the *anchor* node of H and denote it by $\mathbin{\text{⚓}}_H$. Moreover, the sequence of *context* nodes of H is defined as $\mathrm{ctxt}_H := \mathrm{att}_H(e_1)(1) \ldots \mathrm{att}_H(e_m)(1)$ and the *free* nodes of H are all nodes attached to nonterminal hyperedges only, i.e. $\mathrm{free}(H) := \{u \in V_H \mid \forall e \in E_H^\Sigma : u \notin \lfloor \mathrm{att}_H(e) \rfloor\}$.

We will see that TLGs are constructed such that every anchor node u represents an application of a production rule and thus a position in a derivation tree t. The context nodes represent its children as they are merged with anchor nodes after their corresponding nonterminal hyperedges have been replaced. Consequently, by the characteristic edges of an anchor node u we refer to the characteristic edges $E_{t(x)}^\Sigma$ of a position $x \in \mathrm{dom}(t)$ represented by u. We consider a series of simple graph languages to narrow down the class of TLGs step by step. The first example stems from the fact that every context-free word language can be generated by an HRG [14] (if words are canonically encoded by edge-labelled graphs).

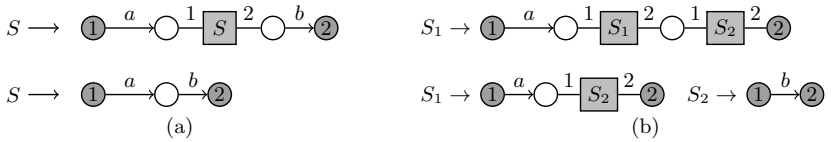

Fig. 2. Two HRGs generating the language $\{a^n.b^n \mid n \geq 1\}$ of string-like graphs.

Example 6. The HRG G shown in Figure 2(a) generates string-like graphs of the form $a^n.b^n$ for each $n \geq 1$. It is well known that the language $L(G)$ is not $\mathtt{MSO}_2$-definable. We observe that for arbitrary hypergraphs $H \in L(G)$ it is not possible to determine a node that is uniquely associated with all terminal edges in the recursive, upper production rule of Figure 2(a) (which is in accordance with the idea behind TLGs formulated at the beginning of this section). This is caused by the intermediate nonterminal hyperedge, which can be replaced by an arbitrarily large HG. Thus, to ensure that TLGs generate $\mathtt{MSO}_2$-definable hypergraphs only, we require that every non-free node (and thus every terminal edge) is reachable from the anchor node using terminal edges only.

However, this requirement is insufficient. For instance, Figure 2(b) depicts an HRG G' with $L(G') = L(G)$ which satisfies the condition from above. G' is obtained by transforming G into the well-known Greibach normal form (for word grammars). In a derivation tree t, a position $x \in \mathrm{dom}(t)$ corresponding to an application of the upper production rule has two children which represent the nonterminal hyperedges labelled with S_1 and S_2, respectively. Since all nodes except for the two leftmost ones are free in this production rule, the parent-child relationship between anchor nodes and context nodes (or any other triple of nodes) cannot be reconstructed in $\mathtt{MSO}_2$. Thus, we additionally require context nodes to be non-free.

In the following we consider *basic* tree-like HGs, which form the building blocks of which a tree-like HG is composed.

Definition 9 (Basic Tree-Like Hypergraphs). *$H \in \mathtt{HG}_{\Sigma_N}$ is a basic tree-like HG if $\mathfrak{t}_H \in \lfloor \mathrm{att}_H(e) \rfloor$ for each $e \in E_H^\Sigma$ and $\lfloor ctxt_H \rfloor \cap free(H) = \emptyset$.*

As a first condition on TLGs, we require right-hand sides of production rules to be (basic) tree-like. In case of string-like graphs, this condition is sufficient to capture exactly the regular word languages (if the direction of edges is ignored), because every such grammar corresponds to a right-linear grammar. If arbitrary graphs are considered, however, there are more subtle cases.

Example 7. Figure 3 (left) depicts an HRG G with three production rules p, q, r. $L(G)$ is the set of "doubly-linked even stars", i.e. a single node u connected by an incoming and an outgoing edge to each of $2n$ nodes for some $n \geq 0$. An HG $H \in L(G)$ is illustrated in Figure 3 (right). Again, $\mathrm{L}(G)$ is not $\mathtt{MSO}_2$-definable. In particular, no derivation tree can be reconstructed from H by identifying nodes (or edges) in H with positions in a derivation tree, because $|V_H| = 5$ and $|E_H| = 8$, but $|\mathrm{dom}(t)| = 9$. The problem emerges from the fact that all anchor nodes are merged with the central node u. Hence, we additionally require that anchor nodes are never merged with each other.

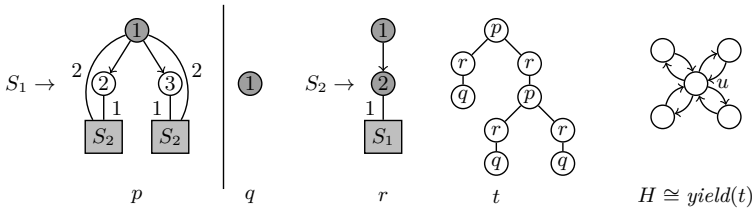

Fig. 3. An HRG G where production rules p, q, r map to tree-like HGs (left) and a generated graph $H \in L(G)$ (right)

Formally, for any $X \in N$, $H \in L(G[X])$ contains merged anchor nodes if for some $t \in \mathtt{D}(G[X])$ with $H \cong yield(t)$, there exist $x, y \in \mathrm{dom}(t)$, $x \neq y$ such that $\mathfrak{t}_{t(x)} \simeq_t \mathfrak{t}_{t(y)}$. The set of all HGs in $\bigcup_{X \in N} L(G[X])$ containing merged anchor nodes is denoted by $\mathcal{M}(G)$.

Definition 10 (Tree-Like Grammar). *$G = (\Sigma_N, P, S) \in \mathtt{HRG}$ is a TLG if $\mathcal{M}(G) = \emptyset$ and for each $p \in P$, $\mathrm{rhs}(p)$ is a basic tree-like HG. The set of all TLGs is denoted by $\mathtt{TLG}$.*

The condition $\mathcal{M}(G) = \emptyset$ is, admittedly, not syntactic. However, it is possible to automatically derive an HRG generating exactly the graphs satisfying it.

Theorem 2. *For each HRG G, one can construct a TLG G' such that $L(G') = L(G) \setminus \mathcal{M}(G)$.*

Remark 1. We call an HG H *tree-like* if it can be composed from basic tree-like HGs, i.e. there exists a TLG G with $L(G) = \{H\}$ (where nonterminals of H are considered to be terminal). Although only basic tree-like HGs are considered in all proofs, our results also hold for tree-like HGs. In particular, if all non-free nodes of an HG H are reachable from the anchor node without visiting an external node, a context node or a nonterminal hyperedge, H is tree-like. Intuitively, the anchor nodes of corresponding TLG production rules are determined by a spanning tree with the anchor of H as root. Analogously, the initial nonterminal S may be mapped to an arbitrary HG provided that it never occurs on the right-hand side of a production rule.

Example 8. According to the previous remark, the recurring example HRG *TLL* illustrated in Figure 1 is a TLG.

As already stated in the introduction, our main result is the following.

Theorem 1. *For each TLG G, there exists an $\mathtt{MSO}_2$ sentence φ_G such that for each $H \in \mathtt{HG}$, $H \in L(G)$ if and only if $\underline{H} \models \varphi_G$.*

An important observation to show this theorem is that every graph H generated by a TLG G has two properties:

1. A derivation tree t of H is $\mathtt{MSO}_2$-definable in H, i.e. TLGs generate recognisable graph languages in the sense of Courcelle [8].
2. Every edge $e \in E_H$ can be uniquely associated in $\mathtt{MSO}_2$ with some $x \in \mathrm{dom}(t)$ corresponding to the production rule $t(x)$ which added e to H.

Hence, given $\mathtt{MSO}_2$ formulae encoding t in H and defining $E^{\Sigma}_{t(x)}$ for each $x \in \mathrm{dom}(t)$, one can easily obtain a formula φ ensuring that all edges in every model of φ are edges introduced by the proper application of a production rule. In particular, $K = \biguplus_{x \in \mathrm{dom}(t)} \mathrm{rhs}(t(x))$ is a model of φ for each $t \in \mathtt{D}(G)$. By Lemma 4, it is sufficient to extend φ to an $\mathtt{MSO}_2$ sentence φ' such that only graphs H with $H \cong [K]_{/\simeq_t} \cong \mathit{yield}(t)$, i.e. graphs that resulted from hyperedge replacement steps where exactly the $[K]_{/\simeq_t}$-equivalent nodes were merged, are models of φ'. For any given pair of nodes, this property can be verfied by a finite (string) automaton running on a path in the derivation tree t.

We collect two direct consequences of Theorem 1 and Lemma 1.

Theorem 3. *The class of languages generated by TLGs is closed under union, intersection and difference.*

Theorem 4. *Given $G \in \mathtt{TLG}$ and $G' \in \mathtt{HRG}$, it is decidable whether $L(G') \subseteq L(G)$. In particular, the inclusion problem for TLGs is decidable.*

5 Tree-Like Separation Logic

As can be seen in Lemma 2, there exists a strong correspondence between $\mathtt{SL}$ and HRGs. This correspondence leads to portability of the obtained TLG results to

analogous SL results. As SL is tailored to reason about heaps, we restrict ourselves to *data structure grammars* (DSG), i.e. HRGs generating heaps only. We denote the class of all DSGs by DSG.

The largest SL fragment considered in this paper is $\mathtt{SL}_{\mathtt{tl}}$ as defined in the introduction (see Definition 1).

Theorem 5. *For every $\mathtt{SL}_{tl}$ formula φ there exists a language-equivalent tree-like DSG G and vice versa.*

Example 9. Consider the $\mathtt{SL}_{\mathtt{tl}}$ formula $\varphi := \sigma(x_1, x_2, x_3, x_4)$ defined over an environment Γ consisting of predicate definitions for two predicate symbols σ and γ.

$$\begin{aligned}\sigma(x_1, x_2, x_3, x_4) := {} & [\exists x_5, x_6, x_7 : x_1.(p, l, r) \mapsto (x_2, x_5, x_6) * \sigma(x_5, x_1, x_3, x_7) \\ & * \sigma(x_6, x_1, x_7, x_4)] \vee [\exists x_5 : x_1.(p, l, r) \mapsto (x_2, x_3, x_5) \\ & * x_3.p \mapsto x_1 * x_5.p \mapsto x_1 * \gamma(x_5, x_3, x_4)] \\ \gamma(x_1, x_2, x_3) := {} & x_2.n \mapsto x_1 * x_1.n \mapsto x_3\end{aligned}$$

Applying Lemma 2 to φ and Γ yields a tree-like DSG generating the same language as the HRG *TLL* shown in Figure 1, i.e. the set of all fully-branched binary trees with linked leaves and parent pointers. In particular, the first disjunct of $\sigma(x_1, x_2, x_3, x_4)$ directly corresponds to the production rule in Figure 1(a), where variable names match with node indices. The other two disjuncts, split across two predicates, translate into basic tree-like HGs and correspond to the second production rule.

We can exploit the additional requirements for DSGs to obtain a simple, yet expressive, *purely syntactical* fragment of TLGs.

Definition 11 (Δ-DSGs). *Let $\Delta \subseteq \Sigma$ be a nonempty set of terminal symbols. Then $G = (\Sigma_N, P, S) \in \mathtt{DSG}$ is a Δ-DSG if for each $p \in P$, $\mathrm{rhs}(p)$ is a tree-like hypergraph and $\mathfrak{k}_p$ has an outgoing edge labelled δ for each $\delta \in \Delta$.*

Example 10. Our example HRG *TLL* shown in Figure 1 is a $\{p, l, r\}$-DSG.

Lemma 5 *Every Δ-DSG with $\emptyset \neq \Delta \subseteq \Sigma$ is a TLG.*

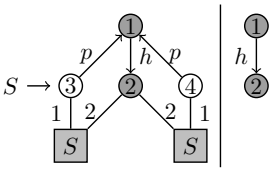

Fig. 4. Tree-like DSG

In terms of expressiveness, we may compare Δ-DSGs to $\mathtt{SL}_{\mathtt{btw}}$ [16], which is, to the best of our knowledge, the largest known fragment of SL with a decidable entailment problem. In particular, consider the $\{h\}$-DSG G depicted in Figure 4 generating reversed binary trees with an additional pointer to the head of another data structure. The language generated by G is not $\mathtt{SL}_{\mathtt{btw}}$-definable, because the number of allocated locations from which the whole heap must be reachable is fixed a priori for every $\mathtt{SL}_{\mathtt{btw}}$ formula and a corresponding environment.

Theorem 6. *Δ-DSGs are strictly more expressive than $\mathtt{SL}_{btw}$, i.e. for every $\mathtt{SL}_{btw}$ formula there exists a language-equivalent Δ-DSG, but not vice versa.*

6 Conclusion

SL and DSGs are established formalisms to describe the abstract shape of dynamic data structures. A substantial fragment of SL is known to coincide with the class DSG. With this relationship, decidability of the satisfiability problem for SL, for instance, follows directly from decidability of its graph theoretic counterpart, the emptiness problem for DSGs. However, the entailment problem or, equivalently, the inclusion problem is undecidable.

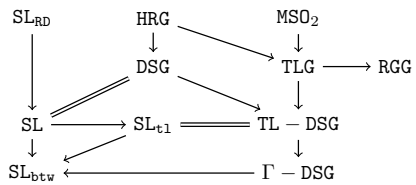

Fig. 5. Relationships between fragments of HRG and SL

We introduced the class TLG of tree-like grammars which generate $\mathtt{MSO}_2$ definable languages only. From this, some remarkable properties, like decidability of the inclusion problem and closure under intersection, directly follow by previous work on context-free and recognisable graph languages. Moreover, the close correspondence between HRGs and SL yields several fragments of SL, in particular $\mathtt{SL}_{\mathtt{tl}}$, where an extended entailment problem is decidable. The resulting fragments are more expressive than $\mathtt{SL}_{\mathtt{btw}}$, the largest fragment of SL with a decidable entailment problem known so far.

Figure 5 depicts an overview of the SL and HRG fragments considered in this paper, where an edge from formalism F_1 to formalism F_2 denotes that the class of languages realizable by F_2 is included in the class of languages realizable by F_1. All of these inclusion relations are strict. For completeness, we also added the class $\mathtt{SL}_{\mathtt{RD}}$ of completely unrestricted Separation Logic with inductive predicate definitions (cf. [16, 1]) and the class RGG of regular graph grammars [9].

With regard to future research, investigating decision procedures and their tractability for the entailment problem for (fragments of) $\mathtt{SL}_{\mathtt{tl}}$ is of great interest. Although the entailment and inclusion problem is effectively decidable for the fragments presented in this paper, our reliance on Courcelle's theorem does not lead to efficient algorithms (see [19] for a recent survey of alternative approaches). Still, a better understanding of the boundary between decidability and undecidability of the entailment problem for SL and the inclusion problem for HRGs might help to obtain more efficient algorithms for specialised fragments. We hope that a combined approach - studying SL as well as context-free graph languages - will lead to further improvements in this area.

References

1. Antonopoulos, T., Gorogiannis, N., Haase, C., Kanovich, M.I., Ouaknine, J.: Foundations for decision problems in separation logic with general inductive predicates. In: FOSSACS. Volume 8412 of LNCS. (2014) 411–425
2. Berdine, J., Calcagno, C., O'Hearn, P.W.: A decidable fragment of separation logic. In: FSTTCS. Volume 3328 of LNCS. (2005) 97–109

3. Berdine, J., Cook, B., Ishtiaq, S.: SLAyer: Memory safety for systems-level code. In: CAV. Volume 6806 of LNCS. (2011) 178–183
4. Brotherston, J., Distefano, D., Petersen, R.L.: Automated cyclic entailment proofs in separation logic. In: CADE-23. Volume 6803 of LNCS. (2011) 131–146
5. Brotherston, J., Kanovich, M.: Undecidability of propositional separation logic and its neighbours. In: LICS. (2010) 130–139
6. Büchi, J.R.: Weak second-order arithmetic and finite automata. Mathematical Logic Quarterly **6**(1-6) (1960) 66–92
7. Chin, W.N., David, C., Nguyen, H.H., Qin, S.: Automated verification of shape, size and bag properties via user-defined predicates in separation logic. Science of Computer Programming **77**(9) (2012) 1006–1036
8. Courcelle, B.: The monadic second-order logic of graphs I: Recognizable sets of finite graphs. Information and Computation **85**(1) (1990) 12–75
9. Courcelle, B.: The monadic second-order logic of graphs V: On closing the gap between definability and recognizability. Theoretical Computer Science **80**(2) (1991) 153–202
10. Courcelle, B., Engelfriet, J.: Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach. Number 138. Cambridge University Press (2012)
11. Dodds, M.: From separation logic to hyperedge replacement and back. In: ICGT. Volume 5214 of LNCS. (2008) 484–486
12. Drewes, F., Kreowski, H.J., Habel, A.: Hyperedge replacement graph grammars. In: Handbook of Graph Grammars and Computing by Graph Transformation. (1997) 95–162
13. Dudka, K., Peringer, P., Vojnar, T.: Predator: A practical tool for checking manipulation of dynamic data structures using separation logic. In: CAV. Volume 6806 of LNCS. (2011) 372–378
14. Habel, A.: Hyperedge Replacement: Grammars and Languages. Volume 643 of LNCS. (1992)
15. Heinen, J., Noll, T., Rieger, S.: Juggrnaut: Graph grammar abstraction for unbounded heap structures. ENTCS **266** (2010) 93–107
16. Iosif, R., Rogalewicz, A., Simacek, J.: The tree width of separation logic with recursive definitions. In: CADE-24. Volume 7898 of LNCS. (2013) 21–38
17. Jacobs, B., Smans, J., Philippaerts, P., Vogels, F., Penninckx, W., Piessens, F.: Verifast: A powerful, sound, predictable, fast verifier for C and Java. In: NFM. Volume 6617 of LNCS. (2011) 41–55
18. Jansen, C., Göbe, F., Noll, T.: Generating inductive predicates for symbolic execution of pointer-manipulating programs. In: ICGT. Volume 8571 of LNCS. (2014) 65–80
19. Langer, A., Reidl, F., Rossmanith, P., Sikdar, S.: Practical algorithms for MSO model-checking on tree-decomposable graphs. Computer Science Review **13-14** (2014) 39–74
20. Lee, O., Yang, H., Yi, K.: Automatic verification of pointer programs using grammar-based shape analysis. In: ESOP. Volume 3444 of LNCS. (2005) 124–140
21. Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: LICS. (2002) 55–74
22. Salomaa, A., Rozenberg, G.: Handbook of Formal Languages, Vol. 3: Beyond Words. Springer (1997)