RWTH Aachen

Department of Computer Science
Technical Report

Tree-Like Grammars and Separation Logic

Christoph Matheja, Christina Jansen and Thomas Noll

ISSN 0935–3232 · Aachener Informatik-Berichte · AIB-2015-12

RWTH Aachen · Department of Computer Science · September 2015

Tree-Like Grammars and Separation Logic

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) [18, 2, 14] and hyperedge replacement grammars (HRG) [10, 13].

While these formalisms are intuitive and expressive, important problems are undecidable. In particular, the *entailment problem* for SL [1], i.e. the question of whether all models of a formula φ are also models of another formula ψ (written $\varphi \models \psi$), as well as its graph-theoretical counterpart, the *inclusion problem* for HRGs [10], 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 [11], often restrict themselves to the analysis of list-like data structures, where the entailment problem is known to be decidable [2]. VeriFast [15] 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 ($\mathtt{SL}_{\mathtt{btw}}$) [14].

Approaches based on graph grammars suffer from the undecidability of the related inclusion problem: Lee et al. [17] propose the use of graph grammars for shape analysis, but their approach is restricted to trees. The tool Juggrnaut [13] 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. [12]). 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. [19]) which justifies the alternative name “context-free graph grammars”.

Most of our results stand on two pillars. 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 [5], exactly the word languages definable in monadic second-order logic (`MSO`).

Lemma 1 (Courcelle [6]) *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 and $\underline{H}$ denotes the relational structure associated with the hypergraph H. $L(G)$ and $L(\varphi)$ denote the language generated by the grammar G and the set of models of the formula φ, respectively.

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 [9] and Jansen et al. [16].

Lemma 2 (Jansen et al. [16]) *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 hypergraph H, $H \in L(G)$ 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 7) which is equivalent to TLGs generating heaps rather than arbitrary graphs.

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 of whether an *arbitrary* `SL` formula φ entails an $\mathtt{SL}_{\mathtt{tl}}$ formula ψ, are decidable.

2. Although negation and conjunction are restricted to pure formulae, $\mathtt{SL_{tl}}$ 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 $\mathtt{SL_{tl}}$-definable. In particular, we show that $\mathtt{SL_{tl}}$ is strictly more expressive than $\mathtt{SL_{btw}}$. The same holds for an entirely syntactic fragment of TLGs, called Δ-DSGs, and a corresponding fragment of $\mathtt{SL_{tl}}$.

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. Section 5 studies an entirely syntactic fragment of TLGs and compares its expressive power to Courcelle's regular graph grammars. We quickly recapitulate how $\mathtt{SL}$ formulae is translated into DSGs in Section 6. Our results on TLGs are transferred to $\mathtt{SL}$ and discussed in Section 7. Finally, Section 8 concludes.

Some technical details and proofs are omitted in the paper and are provided in the appendix instead.

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.

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 $\boxplus$ 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 $t|_x : \{y \mid x.y \in \mathrm{dom}(t)\} \to S : y \mapsto t(x.y)$. With each tree t, we associate the relational structure $\underline{t} := (\mathrm{dom}(t), (\mathrm{succ}_i)_{i \in [\Re(S)]}, (\mathrm{tlb}_s)_{s \in S})$ where $\mathrm{succ}_i := \{(x, x.i) \in \mathrm{dom}(t) \times \mathrm{dom}(t)\}$ and $\mathrm{tlb}_s := \{x \in \mathrm{dom}(t) \mid t(x) = s\}$.

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 $\texttt{MSO}_2$ 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. [8]). Note that the semantics of src, tgt and E_s has been given explicitly in the definition of $\underline{H}$. We allow other operators like conjunction $\wedge$, implication $\rightarrow$ and universal quantification $\forall$ that can be obtained from the given ones. To keep formulae readable in the remainder of this paper, we make use of the following shortcuts which are rather straightforward to define in $\texttt{MSO}_2$. If $\mathcal{W}$ is a finite set of variables, $\exists \mathcal{W} : \varphi$ denotes the existential quantification of all variables in $\mathcal{W}$. $\exists! x : \varphi(x) := \exists x : \varphi(x) \wedge \forall y : \varphi(y) \rightarrow x = y$ denotes the unique existential quantification of x and $\Pi(X, Y_1, \ldots, Y_m)$ denotes that the sets $Y_1, \ldots, Y_m$ form a partition of X. The formulae $V(x)$ and $E(x)$ state that x is a node or an edge, respectively, and $\mathrm{inc}(x, y)$ is satisfied if and only if x is an edge incident to a node y. If $X_1, \ldots, X_m$ are second order variables and $I \subseteq [m]$, $X_I(x)$ denotes the disjunction $\bigvee_{i \in I} X_i(x)$. Finally, $\mathrm{tree}(X, x)$ means that the nodes in X form the domain of a relational tree structure $\underline{t}$ with root x (cf. [14] for a formal definition).

Heaps Similarly to the typical RAM model, a heap is understood as a set of locations $\mathit{Loc} := \mathbb{N}$, whose values are interpreted as pointers to other locations. Formally, we define a *heap* as a partial mapping $h : \mathit{Loc} \rightarrow \mathit{Loc} \uplus \{\mathbf{null}\}$. The set of all heaps is denoted by $\texttt{Hp}$. Let Σ be a finite set of *selectors* equipped with an injective ordering function $\mathrm{cn} : \Sigma \rightarrow [0, |\Sigma| - 1]$. We assume a heap to consist of a single kind of objects with a fixed set of fields Σ which are modelled by reserving exactly $|\Sigma|$ successive locations.

Separation Logic with Recursive Definitions We consider a fragment of Separation Logic, similar to Separation Logic with recursive definitions in [14, 16], in which negation $\neg$, $\texttt{true}$, and conjunction $\wedge$ in spatial formulae are disallowed. Let *Pred* be a set of predicate names. The *syntax of* $\texttt{SL}$ is given by:

$E ::= x \mid \mathbf{null}$	
$P ::= x = y \mid x \neq y \mid P \wedge P$	pure formulae
$F ::= \mathbf{emp} \mid x.s \mapsto E \mid F * F \mid \exists x : F \mid \sigma(x_1, ..., x_n)$	spatial formulae
$S ::= F \mid S \vee S \mid S \wedge P$	$\texttt{SL}$ formulae

where $x, y, x_1, ..., x_n \in \mathit{Var}$, $s \in \Sigma$ and $\sigma \in \mathit{Pred}$. We call $x.s \mapsto E$ a *points-to assertion*, $\sigma(x_1, ..., x_n)$ a *predicate call*.

Note that we do not require all selectors of a given variable to be defined by a single points-to assertion. Furthermore, it is straightforward to add program variables to $\texttt{SL}$, 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 specified 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 [16] for a formal definition.

The semantics of the remaining `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 $\mathtt{MSO}_2$:

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

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 points-to assertion.

From now on, we assume that all existentially quantified variables are eventually allocated. This requirement is similar to the “establishment” condition in [14]. 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 [12, 19].

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

Definition 1 (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 $[H] = (V_H, E)$, $E := \{(\mathrm{att}_H(e)(1), \mathrm{lab}_H(e), \mathrm{att}_H(e)(2)) \mid e \in E_H\}$.

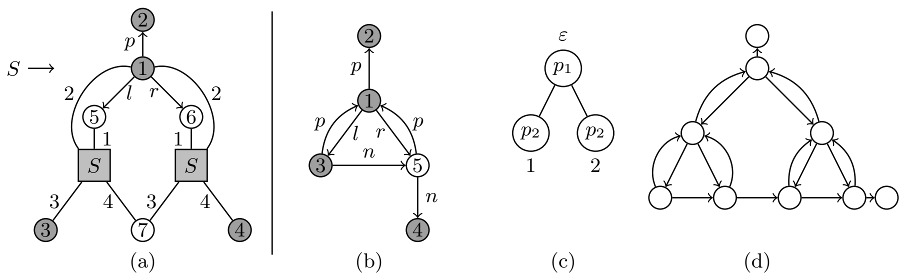
Example 1. As an example, consider the HG illustrated in Figure 1(a) (right). For referencing purpose, we provide a unique index $i \in [|V|]$ inside of each node u_i represented by a circle. External nodes are shaded. For simplicity, we assume them to be ordered according to the provided index. Terminal 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. Let K and H be HGs with disjoint sets of nodes and edges. Intuitively, a non-terminal hyperedge e of K is replaced by H by first removing e, adding H to K and identifying the nodes of K originally attached to e with the sequence of external nodes of H. This is formally expressed by a quotient.

Definition 2 (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 3 (Hyperedge Replacement). *Let $H, K \in \mathtt{HG}_{\Sigma_N}$ 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 $u \;{}_{H,e}{\approx}_K\; v$.*



(a) production rule p_1 (b) production rule p_2 (c) derivation tree t (d) *yield*(t)

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 4 (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 *and the maximal number of nonterminal hyperedges in any production rule* $p \in P$ *by* $\Bbbk$.

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) and 1(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. The rule graph of p_2 depicts a fully-branched binary tree of height 1 with additional parent-pointers and two n-connected leaves. Later we will see that the HRG describes arbitrary trees of the aforementioned structure.

A key feature of HRGs is that the order in which nonterminal hyperedges are replaced is irrelevant, i.e. HRGs are confluent (cf. [12, 19]). 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 5 (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* t *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 6 (HR Language). *The* language *generated by* $G \in \mathit{HRG}$ *is the set* $L(G) = \{\mathit{yield}(t) \mid t \in \mathit{D}(G)\}$.

Example 4. The HRG *TLL*, provided in Figure 1, generates the language 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. [19] below Theorem 3.10).

Lemma 3 *For each $G \in$ HRG, $D(G)$ 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 7 (Merged Nodes). *Let $G \in$ HRG, $t \in D(G)$, $x, y \in \mathrm{dom}(t)$ 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 *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 *yield*(t).

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

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

Proof. By complete induction over the height of derivation trees. ☐

4 Tree-Like Grammars

This section introduces tree-like grammars (TLG), a fragment of HRGs which can be translated into $\mathtt{MSO}_2$. The main idea of TLGs 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 [6].
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. We call $E^{\Sigma}_{t(x)}$ the *characteristic edges* of x.

Hence, given some $\mathtt{MSO}_2$ 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 $H \cong [K]_{/\simeq_t} \cong yield(t)$, i.e. graphs that resulted from hyperedge replacement steps where exactly the $[K]_{/\simeq_t}$-equivalent nodes were merged, are models of φ'.

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 $\downarrow_H$. 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 [12] (if words are canonically encoded by edge-labelled graphs).

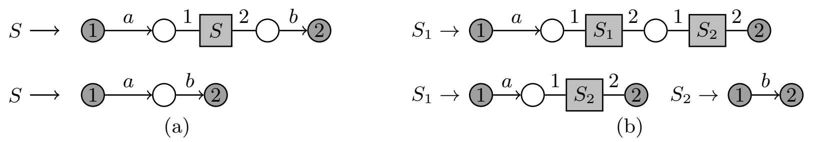


Fig. 2. Two HRGs generating the language $\{a^n.b^n \mid n \geq 1\}$ of string-like graphs.

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. 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 8 (Basic Tree-Like Hypergraphs). *$H \in \mathtt{HG}_{\Sigma_N}$ is a* basic *tree-like HG if $\perp_H \in \lfloor \mathrm{att}_H(e) \rfloor$ 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, 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.

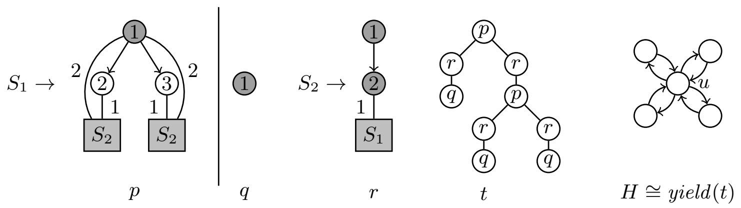


Fig. 3. An HRG G where production rules p, q, r map to tree-like HGs (left) and a generated graph $H \in L(G)$ (right)

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 $\perp_{t(x)} \simeq_t \perp_{t(y)}$. The set of all HGs in $\bigcup_{X \in N} L(G[X])$ containing merged anchor nodes is denoted by $\mathcal{M}(G)$.

Definition 9 (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 $\mathtt{TLG}$.*

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. Since this construction is straightforward, but rather tedious, we skip it for space reasons. 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.

The condition $\mathcal{M}(G) = \emptyset$ is, admittedly, not syntactic. However, it is possible to automatically derive the largest subset of graphs generated by an HRG that satisfies it.

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

Our main result on TLGs is the following.

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

In order to prove this theorem, we have to encode derivation trees in graphs generated by TLGs. We associate each terminal edge with a unique, characteristic edge of some anchor node and use the constructed derivation tree t to verify that two nodes are identical if and only if they have been merged due to hyperedge replacement, i.e. are equivalent with respect to $\simeq_t$ (see Lemma 4).

Before we present the main steps of the proof, we state two important consequences of Theorem 1 and Lemma 1.

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

Proof. It is straightforward that TLGs are closed under union. By Lemma 1, for each $G = (\Sigma_N, P, S) \in \mathtt{HRG}$ and $\mathtt{MSO}_2$ sentence φ, there exists $G' = ({\Sigma_N}', P', S') \in \mathtt{HRG}$ such that $L(G') = \{H \in L(G) \mid \underline{H} \models \varphi\}$. In particular, each $q \in P(G')$ is isomorphic to some $p \in P'$ except for the names of nonterminals. Thus, if G is a TLG, so is G'. Then the claim follows from Theorem 1 and the fact that $\mathtt{MSO}_2$ is closed under conjunction and negation. $\square$

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.*

Proof. By Theorem 1, there exists an $\mathtt{MSO}_2$ sentence φ such that $L(G) = \{H \in \mathtt{HG}_\Sigma \mid \underline{H} \models \varphi\}$. Applying Lemma 1, we can construct an HRG G'' with $L(G'') = \{H \in L(G') \mid \underline{H} \models \neg\varphi\}$. Then, $L(G') \subseteq L(G)$ if and only if $L(G'') = \emptyset$ which is decidable due to Lemma 3. $\square$

In the following constructions, we use two sets of second-order variables $\mathcal{A}$ and $\mathcal{S}$. Each set variable $A_p \in \mathcal{A} = \{A_p \mid p \in P\}$ collects all anchor nodes corresponding to a position in a derivation tree labelled by $p \in P$. Each set $S_e \in \mathcal{S}$ collects all edges corresponding to the characteristic edge $e \in E_p^\Sigma$ of positions labelled by p in a derivation tree. Formally, we introduce an extended set of edge labels $\mathcal{S} := \{S_e \mid e \in E_p^\Sigma, p \in P\}$ together with functions $\ell : \mathcal{S} \to \Sigma : S_e \mapsto \mathrm{lab}_p(e)$ and $\rho : \mathcal{S} \to P : S_e \mapsto p$. Let $\mathcal{W} = \mathcal{A} \uplus \mathcal{S}$.

To improve readability in the following constructions, we introduce an auxiliary formula which takes three variables x, y, z and is satisfied iff x corresponds to a characteristic edge $e \in E_p^{\Sigma}$ of anchor node y and is additionally attached to z. Formally, let $e \in E_p^{\Sigma}$ and $char_e(x, y, z) := E_{\ell(S_e)}(x) \wedge S_e(x) \wedge A_{\rho(e)}(y) \wedge \mathrm{src}(y, x) \wedge \mathrm{tgt}(x, z)$ if $\perp_p = \mathrm{att}_p(e)(1)$ ($\perp_p = \mathrm{att}_p(e)(2)$ with y and z swapped in src and tgt). The first auxiliary lemma to prove Theorem 1 states that derivation trees can be encoded in every graph generated by a TLG.

Lemma 5 *For each TLG $G = (\Sigma_N, P, S)$, a set of derivation trees of G is $\mathtt{MSO}_2$-definable with parameters $\mathcal{W}$ in every $H \in \mathtt{HG}_\Sigma$, i.e. there exists a definition scheme $(\psi, (succ_i)_{i\in[\Bbbk]}, (\mathrm{tlb}_p)_{p\in P})$ such that for each $H \in \mathtt{HG}_\Sigma$ with $H, \jmath \models \psi(\mathcal{W})$, $\underline{t} := (dom, (\jmath(succ_i))_{i\in[\Bbbk]}, (\jmath(\mathrm{tlb}_p))_{p\in P})$ is a relational tree structure representing a tree $t \in \mathtt{D}(G)$, where dom is given by the union of all nodes satisfying exactly one of the formulae tlb_p, $p \in P$.*

Proof. Let $k_p := |E_p^N|$ for each $p \in P$ and $edge_e(x, y) := \exists z : char_e(z, x, y)$. We provide a definition scheme $(\psi, (succ_i)_{i\in[\Bbbk]}, (\mathrm{tlb}_p)_{p\in P})$ to embed tree structures (see Section 2) in graphs and ensure that each tree is in $\mathtt{D}(G)$. These formulae are defined as follows:

- $\mathrm{tlb}_p(x, \mathcal{W}) := A_p(x)$ if $p \in I$,
- $\mathrm{tlb}_p(x, \mathcal{W}) := A_p(x) \wedge \exists y \bigvee_{i\in[\Bbbk]} succ_i(y, x)$ if $p \notin I$,
- $succ_i(x, y, \mathcal{W}) := \bigvee_{p\in P} A_p(x) \wedge \bigwedge_{e\in E_{p,i}} edge_e(x, y) \wedge A_{J(p,i)}(y)$,
- $\psi(\mathcal{W}) := \exists X, r : shape(\mathcal{W}, X, r) \wedge succ(\mathcal{W}, X)$,
- $shape(\mathcal{W}, X, r) := A_I(r) \wedge \Pi(X, \mathcal{A}) \wedge tree(X, r)$,
- $succ(\mathcal{W}, X) := \bigwedge_{p\in P} \forall x : A_p(x) \to exactSucc_p(x, \mathcal{W})$,
- $exactSucc_p(x, \mathcal{W}) := \bigwedge_{j\in[k_p]} \exists y : succ_j(x, y, \mathcal{W}) \wedge \bigwedge_{k_p\le j\le\Bbbk} \neg\exists y : succ_j(x, y, \mathcal{W})$

where $I = \{p \in P \mid \mathrm{lhs}(p) = S\}$, $J(p, i) = \{q \in P \mid \mathrm{lhs}(q) = \mathrm{tlb}_p(e_i), e_i \in E_p^{\Sigma}\}$. Let $\jmath$ be an interpretation of $\mathcal{W}$, X and r and $H \in \mathtt{HG}_\Sigma$. The graph encoded by the definition scheme from above is $K = (V, E)$ where

$$V := \{u \in V_H \mid \text{ there exists a unique } p \in P \text{ s.t. } \mathrm{tlb}_p(u, \jmath(\mathcal{W})) \text{ holds}\},$$
$$E := \{(u, i, v) \in V \times [\Bbbk] \times V \mid succ_i(u, v, \jmath(\mathcal{W})) \text{ holds}\}.$$

With each graph K as above, we associate the function $t : \tau(V_K) \to P, t(x) = p$ if and only if there exists $u \in V_K$ with $\mathrm{tlb}_p(u, \jmath(\mathcal{W}))$ and $\tau(u) = x$, where $\tau : V \to \mathbb{N}^\star$, $\tau(\jmath(r)) = \varepsilon$ and $\tau(v) = \tau(u).i$ if $(u, i, v) \in E$, otherwise. Since, $\mathcal{A}$ partitions X and the nodes in X form a tree with root r and successor relations $succ_i$, t is a tree. Then, $t \in \mathtt{D}(G)$ for each graph K with $\underline{K}, \jmath \models \psi(\mathcal{W})$ follows by complete induction over $|V|$. □

In particular, for each $H \in L(G)$, $\mathcal{W}$ can be chosen such that a derivation tree $t \in \mathtt{D}(G)$ with $yield(t) \cong H$ is defined in $\underline{H}$. This is formalised in the following.

Definition 10. *Let $G = (\Sigma_N, P, S)$ be a TLG, $p \in P$ and $E_{p,i} := \{S_e \in \mathcal{S} \mid e \in E_p^{\Sigma}, ctxt_p(i) \in \lfloor \mathrm{att}_p(e) \rfloor\}$ be the set of all extended labels of edges incident to the i-th context node of p.*

A derivation tree $t \in \mathtt{D}(G)$ is contained *in $H \in L(G)$ if there exists a set $T \subseteq V_H$ and a bijection $f : T \to \mathrm{dom}(t)$ such that for each $u \in T$ either $f(u) = \varepsilon$ or $f(u) = x.i$, $x \in \mathrm{dom}(t)$, $i \in \mathbb{N}$ and $E_{t(x),i} = \{S_e \in \mathcal{S} \mid f^{-1}(x) \xrightarrow{e} u\}$.*

Lemma 6 *For each TLG $G = (\Sigma_N, P, S)$, $H \in L(G)$ and $t \in \mathtt{D}(G)$ with $\mathit{yield}(t) \cong H$, t is contained in H.*

Proof. It suffices to choose T as the set of all nodes in V_H that correspond to anchor nodes in $\mathit{yield}(t) \cong H$ and set $f(u) = f(v).i$ if v corresponds to the i-th context node of u. More precisely, we construct a TLG G' generating the graph consisting only of anchor nodes and edges between anchor nodes defined inside a single production rule. For each production rule $p \in P$, we obtain a corresponding production rule q as follows.

1. Make every nonterminal hyperedge e_i an edge of rank 1 attached to $\mathrm{ctxt}(i)$.
2. Take the subgraph of $\mathrm{rhs}(p)$ induced by the nodes $\downarrow$ and ctxt.
3. For each $i \in [|\mathrm{ctxt}|]$, replace all edges between $\downarrow$ and $\mathrm{ctxt}(i)$ by a single terminal edge from $\downarrow$ to $\mathrm{ctxt}(i)$ labelled $E_{p,i}$.

By construction, for each $t \in \mathtt{D}(G)$, there exists $t' \in \mathtt{D}(G')$ with $\mathrm{dom}(t) = \mathrm{dom}(t')$ and $t'(x) = t(x)'$. Clearly, $\mathit{yield}(t')$ encodes a subgraph of $\mathit{yield}(t)$. Then, the desired set T and function f are given by $T = V_{\mathit{yield}(t')}$ and $f(u) = x.i$ if there exists a node $v \in T$ and $p \in P$ with $u \xrightarrow{E_{p,i}} v$ and $f(u) = \varepsilon$, otherwise. ☐

We need one more ingredient before proving Theorem 1: How to use a derivation tree t to check whether two nodes are identical with respect to $\simeq_t$?

Let $t \in \mathtt{D}(G)$. Obviously, each pair of positions $x, y \in \mathrm{dom}(t)$, where $y \prec x$, defines a unique (bottom-up) path $\mathit{path}(x, y) := x_1 \dots x_m$ in t where $x_1 = x, x_m = y$ and $x_i = x_{i+1}.j_i$ for $j_i \in \mathbb{N}$, $i \in [1, m-1]$. Its corresponding *trace* is given by $\mathrm{trace}(x_1 \dots x_m) := (j_1, t(x_1)) \dots (j_{m-1}, t(x_{m-1}))(j, t(x_m))$, where $j = 0$ if $x_m = \varepsilon$. We define a finite (word) automaton running on traces of derivation trees to check whether two given nodes – the i-th external node in $t(x_1)$ and the node $\mathrm{att}(e_j)(k)$ in $t(x_m)$ – are merged. In particular, this merge automaton depends on the HRG G and the indices i, j, k only, but not on a specific derivation tree.

Definition 11 (Merge Automaton). *Let $G = (\Sigma_N, P, S) \in \mathtt{HRG}$, $i, j \in [\Re(G)]$ and $k \in [\Bbbk]$. The* merge automaton *$\mathfrak{A}_{i,j,k} = (Q, \Gamma, 1, \delta, F)$ running on traces of derivation trees of G is given by the set of states $Q := \{1\} \uplus (\{1, 0\} \times [\Re(G)] \times [\Bbbk])$, input alphabet $\Gamma = [0, \Re(G)] \times P$, initial state q_0, final states $F := \{(0, j, k)\}$ and the partial transition function $\delta : Q \times \Gamma \to Q$, defined by:*

$$
\begin{aligned}
\delta(1, (x, p)) &= (1, i, x) && \text{if } x > 0,\\
\delta(1, (0, p)) &= (0, i, j) && \text{if } \mathrm{lhs}(p) = S, E_p^N = \emptyset \text{ and } i = j,\\
\delta((1, x, y), (z, p)) &= (1, x', z) && \text{if } \mathrm{att}_p(e_y)(x) = \mathrm{ext}_p(x'),\\
\delta((1, x, y), (z, p)) &= (0, j, k) && \text{if } u := \mathrm{att}_p(e_y)(x) = \mathrm{att}_p(e_j)(k)\\
& && \text{and } (z = 0 \text{ or } u \notin \lfloor \mathrm{ext}_p \rfloor).
\end{aligned}
$$

Intuitively, the automaton records the indices i, j, k in its state space and updates them according to the rules of hyperedge replacement while moving through a derivation tree.

Lemma 7 *Let $G = (\Sigma_N, P, S) \in \mathtt{HRG}$, $t \in \mathtt{D}(G)$ and $x, y \in \mathrm{dom}(t)$ such that $y \prec x$. Furthermore, let $u = \mathrm{ext}_{t(x)}(i)$ and $v = \mathrm{att}_{t(y)}(e_j)(k)$. Then, $u \sim_t v$ if and only if $\mathfrak{A}_{i,j,k}$ accepts $\mathrm{trace}(\mathit{path}(x, y))$.*

Proof. By complete induction over the length of trace($path(x, y)$). □

We now turn to the proof of Theorem 1, in which the automaton from above is used to verify whether two nodes are merged with each other.

Proof (of Theorem 1). Let $G = (\Sigma_N, P, S) \in \mathtt{TLG}$. W.l.o.g., we make two assumptions in order to reduce the technical effort. First, we assume that every production rule is "well-formed", i.e. for each production rule p, all nodes in ext_p and $\mathrm{att}_p(e)$, $e \in E_p^N$, are distinct. It is a classical result that each HRG can be transformed into an equivalent one satisfying this condition (cf. [12], Theorem 4.6). In particular, this transformation does not change tree-likeness of an HRG. Second, we assume for each $p \in P$ that $\mathrm{rhs}(p)$ does not contain *inner* nodes, i.e. nodes which are neither external nor attached to any nonterminal hyperedge. Otherwise, we introduce a new nonterminal symbol $X \notin \Sigma_N$ with $rk_{\Sigma_N}(X) = 1$ and a single production rule mapping X to a single external node. Then, we attach a new nonterminal hyperedge labelled X to each inner node.

We have to define an $\mathtt{MSO}_2$ sentence φ_G such that $H \in L(G)$ iff $\underline{H} \models \varphi_G$. In order to construct φ_G, a set of derivation trees is encoded in H using the definition scheme $(\psi, (succ_i)_{1 \le i \le \Bbbk}, (\mathrm{tlb}_p)_{p \in P})$ from Lemma 5. Recall that this definition scheme depends on a set of parameters $\mathcal{W} = \mathcal{A} \uplus \mathcal{S}$. An evaluation of the parameters $\mathcal{W}$ determines a fixed derivation tree t. Due to Lemma 6, the values of $\mathcal{W}$ can be chosen such that $yield(t) \cong H$ if $H \in L(G)$. Then, φ_G is given by

$$\varphi_G := \mathrm{conn} \wedge \exists \mathcal{W} : \psi(\mathcal{W}) \wedge \mathrm{edges}(\mathcal{W}) \wedge \mathrm{anchors}(\mathcal{W}) \wedge \mathrm{mergeNodes}(\mathcal{W}), \qquad (1)$$

where all subformulae except for $\psi(\mathcal{W})$ are needed to assert $yield(t) \cong H$. In the following, we explain the components of φ_G in more detail.

1. All nodes of H are incident to at least one edge. Together with the conditions on edges provided in (2), this ensures that every model of φ_G is a connected graph. Formally, $\mathrm{conn} := \forall x \exists y : \mathrm{inc}(x, y)$.
2. Every edge corresponds to exactly one characteristic edge of an anchor node. In particular, $\mathcal{S}$ forms a partition of all edges E of H.

$$\mathrm{edges}(\mathcal{W}) := \Pi(E, \mathcal{S}) \wedge \bigwedge_{S_e \in \mathcal{S}} \forall x : S_e(x) \rightarrow \exists y, z : char_e(x, y, z) \qquad (2)$$

3. Every anchor node $u \in \mathrm{tlb}_p$ has exactly the characteristic edges E_p^Σ. Furthermore, if two edges $e, e' \in E_p^\Sigma$ are attached to the same nodes in $\mathrm{rhs}(p)$, then the same holds for the corresponding two edges of u:

$$\mathrm{anchors}(\mathcal{W}) := \bigwedge_{p \in P} \forall x : \mathrm{tlb}_p(x, \mathcal{W}) \rightarrow \alpha_p(x, \mathcal{W}) \qquad (3)$$

$$\alpha_p(x, \mathcal{W}) := \bigwedge_{u \in V_p \setminus \mathrm{free}(p)} \beta_p^u(x, \mathcal{W}) \wedge \bigwedge_{e \notin E_p^\Sigma} \forall y, z : \neg char_e(y, x, z)$$

$$\beta_p^u(x, \mathcal{W}) := \exists y : \bigwedge_{\substack{e \in E_p^\Sigma \\ \perp_p \xrightarrow{e} u}} \exists! z : char_e(z, x, y) \wedge \bigwedge_{\substack{e \in E_p^\Sigma \\ \neg \perp_p \xrightarrow{e} u}} \forall z : \neg char_e(z, x, y)$$

4. Two nodes $u \in V_{t(x)}, v \in V_{t(y)}$, where $x \neq y$, represent the same node in V_H iff they are merged according to the rules of hyperedge replacement (see Lemma 4). Since G is well-formed and every node is either external or attached to some nonterminal hyperedge, this is the case iff u, v are merged with the same non-external node. In this setting, external nodes of $t(\varepsilon)$ are a corner case and can be considered non-external.
Given an anchor node $x \in \mathrm{dom}(t)$, an inspection of the characteristic edges of x allows us to verify whether a node corresponds to $\mathrm{ext}_{t(x)}(i)$ or $\mathrm{att}_{t(x)}(e_j)(k)$, where $i, j, k \in \mathbb{N}$.
Furthermore, given two anchor nodes x, y, a merge automaton $\mathfrak{A}_{i,j,k}$ running on trace($path(x, y)$) accepts iff the i-th external node of $t(x)$ is merged with the k-th node attached to the j-th nonterminal hyperedge of $t(y)$ (see Lemma 7). Applying Büchi's Theorem [5] yields a corresponding $\mathtt{MSO}_2$ formula for each automaton $\mathfrak{A}_{i,j,k}$. By $\mathfrak{A}_{i,j,k}(x, y, \mathcal{W})$, we denote an $\mathtt{MSO}_2$ formula which is satisfied iff the run of the respective merge automaton on the unique path from x to y in the encoded derivation tree is accepting.
The $\mathtt{MSO}_2$ formula mergeNodes($\mathcal{W}$) then requires for each pair u, v of nodes that $u = v$ iff both represent the same external node of $t(\varepsilon)$ or there exists an anchor node $w \in \mathcal{A}_p$, $p \in P$, such that $\mathfrak{A}_{i,j,k}(v, w, \mathcal{W})$ as well as $\mathfrak{A}_{i',j,k}(v, w, \mathcal{W})$ holds for some $i, i', k \in \Re(G)$ and $j \in \Bbbk$.

To complete the proof, we show how traces are encoded in derivation trees, we provide a formal definition of mergedNodes($\mathcal{W}$) and we prove the correctness of our construction.

Encoding of traces in derivation trees We encode a trace from x to y as a finite word in a derivation tree using a definition scheme $(\vartheta, min, max, \mathrm{succ}, (\mathrm{lab}_{l,p})_{p\in P, i\in[\Bbbk]})$ with parameters $\mathcal{P} = \{x, y\}$:

- The first and last element of the trace are min and max given by $min(z, \mathcal{P}) := z = x$, $max(z, \mathcal{P}) := z = y$.
- The label of each element is defined as

$$\mathrm{lab}_{l,p}(z, \mathcal{P}) := \mathrm{tlb}_p(z) \wedge \exists z' : \mathrm{succ}(z', z, \mathcal{P}) \wedge \mathrm{succ}_l(z', z)$$

  if $l \neq 0$ and $\mathrm{lab}_{l,p}(z, \mathcal{P}) := \mathrm{tlb}_p(z) \wedge \forall z' : \bigwedge_{i\in\Bbbk} \neg\mathrm{succ}_i(z', z)$ if $l = 0$.
- The domain formula $\vartheta(\mathcal{P}) := \bigwedge_{(i,p)\neq(p,q)} \forall z : \neg(\mathrm{lab}_{i,p}(z, \mathcal{P}) \wedge \mathrm{lab}_{i,p}(z, \mathcal{P}))$ asserts that no element of a trace is labelled twice.
- The binary successor relation is given by

$$\mathrm{succ}(z_1, z_2, \mathcal{P}) := y \rightsquigarrow z_2 \wedge \bigwedge_{i\in\Bbbk} \mathrm{succ}_i(z_2, z_1) \wedge z_1 \rightsquigarrow x$$

  where $z_1 \rightsquigarrow z_2$ denotes that z_2 is reachable from z_1 using the successor relations of the derivation tree, i.e. $z_1 \rightsquigarrow z_2$ is defined as

$$\forall X : [z_1 \in X \wedge (\forall u, v : (u \in X \wedge \bigwedge_{i\in\Bbbk} \mathrm{succ}_i(u, v)) \rightarrow v \in X)] \rightarrow z_2 \in X.$$

The task of transforming a finite (string) automaton into an $\mathtt{MSO}$ formula defined over models of finite words is classical (cf. [5]). The $\mathtt{MSO}_2$ formula corresponding to a merge automaton $\mathfrak{A}_{i,j,k}$ running on a trace as encoded above is denoted by $\mathfrak{A}_{i,j,k}(x, y, \mathcal{W})$ where $\mathcal{W}$ are the parameters used in the definition scheme of derivation trees.

Formal construction of mergeNodes($\mathcal{W}$) Furthermore, we have to formally specify the $\mathtt{MSO}_2$ formula mergeNodes($\mathcal{W}$) to ensure that nodes are merged with each other in compliance with Lemma 4. mergeNodes($\mathcal{W}$) is defined as

$$\forall x, y : V(x) \wedge V(y) \to [x = y \leftrightarrow (\mathrm{sameExt}(x, y, \mathcal{W}) \vee \mathrm{merged}(x, y, \mathcal{W})]. \quad (4)$$

Figure 4 provides formal definitions of the required auxiliary formulae which are briefly explained below.

- $\mathrm{sameExt}(x, y, \mathcal{W})$: This formula checks whether $\jmath(x)$ and $\jmath(x)$ represent the same external node of the root of the derivation tree. These nodes are the only ones which may not be merged with any node due to hyperedge replacement. Thus, we have to treat them as a corner case.
- $\mathrm{merged}(x, y, \mathcal{W})$: This formula checks whether x and y are merged with each other by requiring the existence of a common non-external node $u = \mathrm{att}_{t(z)}(e_j)(k)$ for some anchor node $\jmath(z)$ both are merged with. In case the guessed node is non-free, it also asserts that u is equal to $\jmath(x)$ and $\jmath(x)$.
- $\mathrm{merge}_{j,k}(x, y, \mathcal{W})$: This formula first determines the anchor node of x and then applies a merge automaton to verify that x and u have to be merged.
- $\mathrm{free}_{j,k,p}(x, z, \mathcal{W})$: This formula deals with a corner case in which a non-external node of anchor node $\jmath(z)$ is non-free and asserts that it is identical to $\jmath(x)$.
- $\mathrm{ext}_i(x, y, \mathcal{W})$: This formula holds iff y corresponds the i-th external node of the anchor node $\jmath(x)$.
- $att_{j,k,p}(x, y, \mathcal{W})$: This formula holds iff y corresponds to the k-th node attached to the j-th nonterminal hyperedge of $t(z)$ where z is the position in the encoded derivation tree represented by anchor node x.

Correctness proof It remains to show the correctness of the construction described above, i.e. $\underline{H} \models \varphi_G$ if and only if $H \in L(G)$. We concentrate on the direction $\underline{H} \models \varphi_G$ implies $H \in L(G)$. The converse direction is rather straightforward, because $\mathcal{W}$ can be always be chosen such that a derivation tree of H is encoded by Lemma 6. Thus, let $\underline{H} \models \varphi_G$ and $t \in \mathtt{D}(G)$ be a derivation tree encoded by the definition scheme $(\psi, (\mathrm{succ}_i)_{i \in [\![\mathbb{k}]\!]}, (\mathrm{tlb}_p)_{p \in P})$ in Lemma 5 for an interpretation $\jmath$. Due to (2), every node in H can be reached from a node in the encoded derivation tree using at most one (characteristic) edge. We show the following stronger proposition by complete induction over the height of an encoded derivation tree t:

$$\underline{H} \models \varphi_{G[X]} \text{ implies } H \cong \mathit{yield}(t) \in L(G[X]) \text{ for all } X \in N.$$

$h = 0$: By (2), the set of all edges is partitioned by $\mathcal{S}$ and each edge of H is attached to at least one anchor node, i.e. a node v such that $\mathrm{tlb}_p(v, \jmath(\mathcal{W}))$ holds. Furthermore, by (3), exactly the characteristic edges E_p^{Σ} are associated with a node v satisfying $\mathrm{tlb}_p(v, \jmath(\mathcal{W}))$. Since $h = 0$ and we assumed the absence of inner nodes, $\mathrm{dom}(t)$ is a singleton set and $t(\varepsilon)$ contains only external nodes and no nonterminal hyperedges. Hence, $\mathrm{equal}(x, y, \mathcal{W})$ in (4) is satisfied if and only if $\jmath(x) = \jmath(y)$, i.e. $H \cong t(\varepsilon) \cong \mathit{yield}(t)$.

$$\mathrm{sameExt}(x,y,\mathcal{W}) := \exists z : \bigvee_{p\in I, i\in[rk_{\Sigma_N}(S)]} [\mathrm{ext}_{p,i}(z,x,\mathcal{W}) \wedge \mathrm{ext}_{p,i}(z,y,\mathcal{W})]$$

$$\mathrm{merged}(x,y,\mathcal{W}) := \bigvee_{\substack{p\in P,\\ \mathtt{att}(e_j)(k)\in V_p}} \exists z : \mathrm{tlb}_p(z,\mathcal{W}) \wedge \mathit{free}_{j,k,p}(x,z,\mathcal{W}) \wedge \mathrm{merge}_{j,k}(x,z,\mathcal{W}) \wedge \mathrm{merge}_{j,k}(y,z,\mathcal{W})$$

$$\mathrm{merge}_{j,k}(x,y,\mathcal{W}) := \bigvee_{i\in[\Re]} \exists x_1 : \mathrm{ext}_i(x_1,x,\mathcal{W}) \wedge \mathfrak{A}_{i,j,k}(x_1,y,\mathcal{W})$$

$$\mathrm{free}_{j,k,p}(x,z,\mathcal{W}) := \begin{cases} \mathrm{true} & \text{if } \mathrm{att}_p(e_j)(k) \in \mathrm{free}(p) \\ \exists z_1 : \mathrm{att}_{j,k,p}(z,z_1,\mathcal{W}) \wedge z_1 = x & \text{otherwise} \end{cases}$$

$$\mathrm{ext}_i(x,y,\mathcal{W}) := \bigvee_{p\in P, rk_{\Sigma_N}(p)\geq i} \mathrm{ext}_{p,i}(x,y,\mathcal{W})$$

$$\mathrm{ext}_{p,i}(x,y,\mathcal{W}) := \mathrm{tlb}_p(x,\mathcal{W}) \wedge \begin{cases} x = y & \text{if } i = 1 \\ \bigwedge_{e\in E_p^\Sigma : \downarrow_p \xrightarrow{e} \mathrm{ext}_p(i)} \exists z : \mathit{char}_e(z,x,y) & \text{otherwise} \end{cases}$$

$$\mathrm{att}_{j,k,p}(x,y,\mathcal{W}) := \mathrm{tlb}_p(x) \wedge \bigwedge_{e\in E_p^\Sigma : \downarrow_p \xrightarrow{e} \mathrm{att}_p(e_j)(k)} \exists z : \mathit{char}_e(z,x,y)$$

Fig. 4. Auxiliary formulae to identify attached, external and free nodes as well as applying Lemma 7 to check whether two nodes are merged according to the rules of hyperedge replacement. Here, $I := \{p \in P \mid \mathrm{lhs}(p) = S\}$.

$h > 0$: By Lemma 5, the definition scheme $(\psi, (\mathrm{succ}_i)_{1\leq i\leq \Bbbk}, (\mathrm{tlb}_p)_{p\in P})$ encodes a derivation tree t with unary labelling relations tlb_p for each $p \in P$ and binary successor relations succ_i for each $1 \leq i \leq \Bbbk$. Let $u_0 = \jmath(r)$ be the root of this tree, $u_1, ..., u_n$ its children and $p_0, ..., p_n \in P$ the production rules such that $u_i \in \mathrm{tlb}_{p_i}$ for $i \in [0,n]$. Let H_0 be the graph containing the node u_0, the characteristic edges $E(u_0)$ of u_0 and all nodes reached from u_0 using these edges. In other words H_0 is isomorphic to $t(\varepsilon)$ except that nonterminal hyperedges and free nodes have been removed. Furthermore, for each $i \in [n]$, consider a subgraph H_i of H which is obtained as follows:

1. Let $\mathcal{A}(i) \subseteq \{v \in V_H \mid v \in \mathrm{tlb}_p, p \in P\}$ be the nodes corresponding to $t|_i$ in H.
2. Remove all characteristic edges $E(v)$ from H, where $v \in \mathrm{tlb}_p \setminus \mathcal{A}(i)$, $p \in P$.
3. Then, H_i is the connected component of the resulting graph containing u_i.

Now, observe that $\underline{H_i} \models \varphi_{G[\mathrm{lhs}(p_i)]}$ for the same interpretation $\jmath$ as before (restricted to the nodes and edges in H_i). In particular, the derivation tree encoded in H_i is $t|_i$, and for each pair of nodes in H_i, there are accepting runs of merge automata on $t|_i$ if there are corresponding runs on t by our initial assumption. Hence, by induction hypothesis, we have $H_i \cong \mathit{yield}(t|_i) \in L_{G[\mathrm{lhs}(p_i)]}$ for each $i \in [n]$. By construction of H_i, $\mathcal{A}(i) \cap \mathcal{A}(j) = \emptyset$ if $i \neq j$. Thus, all edges of H are either characteristic edges of the root node of t or of some node in $\mathcal{A}(i)$, $i \in [n]$, i.e. $E_H \cong \biguplus_{i\in[0,n]} E_{H_i}$.

It remains to verify that exactly the external nodes of $H_1, \ldots, H_r$ are merged with the corresponding attached nodes of H_0. Take two nodes $u \in V_{H_i}$, $v \in V_{H_j}$ for some $i \neq j$. The following cases arise:

– Either u or v does not correspond to an external node of some anchor node in H_i and H_j, respectively. In other words, for all $w \in \mathcal{A}(i)$ and $l \in [1, \Re(P)]$, $\mathrm{ext}_l(w, u, \jmath(\mathcal{W}))$ is violated (or analogously for v and all anchors in $\mathcal{A}(j)$). Then, by Definition 3, u and v may not refer to the same node in *yield*(t) which is ensured by (4), because $\mathrm{merge}_{j,k}(x, y, \mathcal{W})$ only holds if $\jmath(x)$ is an external node.
– Both u and v correspond to external nodes of some anchor in H_i and H_j, but the formula $\mathrm{equal}(u, v, \jmath(\mathcal{W}))$ is violated. By definition of (4), this means there exists no position z in t such that there are accepting runs of merge automata for the some node in $\mathrm{rhs}(t(z))$. By Lemma 7, u and v are not merged due to hyperedge replacement and therefore refer to different nodes in *yield*(t).
– The formula $\mathrm{equal}(u, v, \jmath(\mathcal{W}))$ holds. Then, u and v are external nodes for some anchor nodes in H_i and H_j. Moreover, there are accepting runs of merge automata terminating in the same position z of t for the same attached node in $\mathrm{rhs}(t(z))$. By definition of (4), this implies $u = v$ and by Lemma 7 u and v represent the same node in *yield*(t).

In each of these cases, (4) implies $u = v$ if and only if u and v refer to the same node in *yield*(t). To finish the proof, let $\approx$ be the reflexive, symmetric, and transitive closure of the relation $\{(u, v) \in V_{H_i} \times V_{H_j} \mid \mathrm{equal}(u, v, \jmath(\mathcal{W})) \text{ holds}, i \neq j\}$. Furthermore, let $K_0, K_1, ..., K_n$ be disjoint, but isomorphic copies of $H_0, ..., H_n$. Then, by the previous observation, $u \approx v$ holds if and only if $u \simeq_t v$ holds. Hence, by Lemma 4,

$$H \cong [\biguplus_{i \in [0,n]} K_i]_{/\simeq_t} \cong [\biguplus_{i \in [0,n]} K_i]_{/\approx} \cong \mathit{yield}(t).$$

Since $t \in \mathtt{D}(G)$, we have *yield*$(t) \in L(G)$ which completes the proof. □

5 Regular and CES-Grammars

In this section, we study an entirely syntactic fragment of TLGs, called *context-external separated* TLGs (`CES-TLG`), and compare its expressiveness to Courcelle's regular graph grammars [7].

Definition 12 (CES-Hypergraph). *A hypergraph* $H = (V, E, \mathrm{att}, \mathrm{lab}, \mathrm{ext}) \in \mathit{HG}_{\Sigma_N}$ *is* context-external separated *(CES) if and only if* $\lfloor ctxt \rfloor \cap \lfloor \mathrm{ext} \rfloor = \emptyset$ *and neither ctxt nor* ext *contain repetitions.*

By Definition 3, only external nodes of an HG are merged with attached nodes of another HG during hyperedge replacement. Thus, it is straightforward that $\mathcal{M}(a) = \emptyset$ if every production rule of $G \in \mathtt{HRG}$ maps to a CES-hypergraph.

Definition 13 (`CES-TLG`). $G \in \mathit{HRG}$ *is a* context-external separated *TLG (`CES-TLG`) if each of its production rules maps to a tree-like HG which is also a CES-hypergraph.*

Note that we do not require that the basic tree-like HGs a production rule may be composed of to satisfy the CES property, i.e. tree-like HGs as described in Remark 1 may be used as right-hand sides of production rules. Thus, our recurring example HRG *TLL* illustrated in Figure 1 is a `CES-TLG`.

A regular graph grammar (cf. [7], section 5) is an HRG $G = (\Sigma_N, P, S)$ in which for every production rule $p \in P$,

1. ext_p contains no repetitions,
2. each edge $e \in E_p$ is attached to at least one internal node, i.e. $\lfloor \mathrm{att}_p(e) \rfloor \not\subseteq \lfloor \mathrm{ext}_p \rfloor$,
3. Any pair of nodes $u, v \in V_p$ is connected by a terminal and internal path, i.e. v is reachable in p from u by a path containing no nonterminal hyperedges and no external nodes (except u and v).

We denote the class of regular graph grammars by `RGG`. In particular, Courcelle showed that every regular tree language and sets of series parallel graphs are in `RGG` [7].

Theorem 5. *The class of languages generated by* `CES-TLG`*s is strictly larger than the corresponding class of languages generated by Courcelle's regular graph grammars [7].*

Proof (Sketch). We first observe that repetitions of context nodes can be removed by replacing all hyperedges $e_1, ..., e_k$ sharing the same context node by a new nonterminal hyperedge e (labelled with a fresh nonterminal) that captures the effect of replacing all of these hyperedges (which is always possible, cf. [12]). In particular, if $e_1, ..., e_k$ (except at most one) can only be replaced by CES-hypergraphs, then every production rule replacing the fresh nonterminal contains exactly the same repetitions of context nodes as the graphs replacing $e_1, ..., e_k$, i.e. all repetitions can be removed iteratively. Furthermore, every nonterminal hyperedge occurring on the right-hand side of a production rule of a regular graph grammar is attached to at least one non-free node. Hence, we may assume w.l.o.g. that every production rule of a regular graph grammar G maps to a CES-hypergraph. By Remark 1 and the previous observation, these HGs are also tree-like, i.e. there exists a `CES-TLG` generating the the same language as G. For the converse direction, consider the `CES-TLG` *TLL* shown in Figure 1 as an example of a language not in `RGG`. $\square$

6 Separation Logic and Data Structure Grammars

We quickly recapitulate the close relationship between `SL` and HRGs studied in [16]. To simplify notation, we assume w.l.o.g. that a heap h contains no unused locations. Hence, for a heap containing n objects, we have $dom(h) = [n \cdot |\Sigma|]$. Then every heap is associated with an HC as follows.

Definition 14 (Heap Mapping). *The* heap mapping $\mu : \mathit{Hp} \to \mathit{HC}_\Sigma$ *is given by* $\mu(h) = (V, E, \mathrm{att}, \mathrm{lab}, \varepsilon)$ *with* $V = \lfloor \mathrm{dom}(h) \rfloor$, $E = \{e_{v,s} \mid v \in V, s \in \Sigma, h(v + cn(s)) \in \mathrm{dom}(h)\}$, $\mathrm{att}(e_{v,s}) = v.\lfloor h(v + cn(s)) \rfloor$, *and* $\mathrm{lab}(e_{v,s}) = s$, *where* $\lfloor . \rfloor$ *rounds down to a location in* $\{1 + k \cdot |\Sigma| \mid k \in \mathbb{N}\}$.

With the connection between heaps and HCs established by the heap mapping μ, we call $G = (\Sigma_N, P, S) \in$ `DSG` and a formula $\varphi(x_1, ..., x_n) \in$ `SL` whose predicates are defined in $\Gamma \in \mathit{Env}$ *language-equivalent* if for all $h \in \texttt{Hp}$ and interpretations $\jmath$, $h, \jmath \models \varphi(x_1, ..., x_n)$ if and only if $H \in L(G)$ where H is isomorphic to $\mu(h)$ except that the external nodes are set to $\mathrm{ext} = \jmath(x_1) \ldots \jmath(x_n)$.

In the following, we briefly sketch one direction of Lemma 2 which intuitively allows to translate SL formulae into DSGs and vice versa. Correctness proofs as well as the backward direction are provided in [16].

Lemma 2 (Jansen et al. [16]) *There exists a translation* $\mathrm{env}[\![.]\!] : \mathtt{DSG} \to \mathtt{SL} \times \mathit{Env}$*, such that* G *and* φ *defined over* Γ *are language-equivalent for each* $G \in \mathtt{DSG}$ *with* $\mathrm{env}[\![G]\!] = (\varphi, \Gamma)$*. Conversely, there exists a translation* $\mathrm{hrg}[\![.]\!] : \mathtt{SL} \times \mathit{Env} \to \mathtt{DSG}$ *such that* φ *and* $\mathrm{hrg}[\![\varphi, \Gamma]\!]$ *are language-equivalent for each* $\Gamma \in \mathit{Env}$ *and* $\varphi \in \mathtt{SL}$*.*

Let $\varphi(x_1, ..., x_n)$ be an SL formula defined over an environment $\Gamma \in \mathit{Env}$ and a set of selectors Σ. We provide the construction of $\mathrm{hrg}[\![\varphi(x_1, ..., x_n), \Gamma]\!] \in \mathtt{DSG}$.

For each predicate name $\sigma \in \mathit{Pred}$ defined in Γ, we introduce a nonterminal X_σ. These nonterminals are collected in a set N.

Let $\sigma(x_1, ..., x_m)$ be a disjunct of a predicate definition in Γ and $\mathit{Var}(\sigma)$ the set of variables in this disjunct. We construct a corresponding set $\mathcal{H}$ of *tagged* hypergraphs, i.e. $\mathcal{H}$ consists of pairs (H, tag) where $H \in \mathtt{HG}_{\Sigma_N}$ and $\mathit{tag} : V_H \to 2^{\mathit{Var}(\sigma)}$ maps each node to a set of variables of $\sigma(x_1, ..., x_m)$. This construction is defined inductively by a mapping *tgraph* as shown in Figure 5. Here, $\sim$ denotes the equivalence relation induced by $u \sim v$ if and only if $\mathit{tag}(u) \cap \mathit{tag}(v) \neq \emptyset$. The unification operator $\Downarrow (H, \mathit{tag})$ denotes the quotient according to $\sim$ and is canonically extended to sets of tagged hypergraphs. Furthermore, it is assumed that each graph contains a dedicated node representing the location **null**. The external nodes of H are determined by

$$\mathrm{ext}_H = \mathit{tag}^{-1}(x_1) \ldots \mathit{tag}^{-1}(x_n).\mathit{tag}^{-1}(\mathbf{null}).$$

Here, the last node is used to ensure that all nodes representing the location **null** are merged into one. Then, each pair $(H, \mathit{tag}) \in \mathit{tgraph}(\sigma(x_1, ..., x_n))$ is translated by $\mathrm{hrg}[\![.]\!]$ into a production rule $p = (X_\sigma, H)$. Let P be the set of all production rules obtained by applying the procedure from above to every disjunct of every predicate definition in Γ.

The formula $\varphi(x_1, ..., x_n)$ is treated analogously. We introduce a fresh nonterminal X_φ and production rules $P_\varphi := \{(X_\varphi, H) \mid (H, \mathit{tag}) \in \mathit{tgraph}(\varphi(x_1, ..., x_n))\}$. Then our construction yields the HRG

$$\mathrm{hrg}[\![\varphi(x_1, ..., x_n), \Gamma]\!] = (\Sigma_N \uplus \{X_\varphi\}, P \uplus P_\varphi, X_\varphi).$$

The function *tgraph* yields tagged hypergraphs for each disjunct of each predicate definition in an SL environment. However, if such a disjunct is unsatisfiable, it is not guaranteed that all of these HGs are heap configurations. Similar to Theorem 2, the following theorem allows us to automatically derive the largest subset of heap configurations generated by a given HRG.

Theorem 6. *For every* $G \in \mathtt{HRG}$*, an HRG* G' *can be constructed such that* $L(G') = L(G) \cap \mathtt{HC}_\Sigma$*.*

Proof. By application of the Filter Theorem for HRGs. $\square$

$$
\begin{aligned}
tgraph(\mathbf{emp}) :=& \{(H_\emptyset, tag_\emptyset)\}\\
tgraph(x.s \mapsto \mathbf{null}) :=& \Downarrow \{((\{u,v\},\{e\},\{e \mapsto u.v\},\{e \mapsto s\},\varepsilon),\{u \mapsto \{x\}, v \mapsto \{\mathbf{null}\}\})\}\\
tgraph(x.s \mapsto y) :=& \Downarrow \{((\{u,v\},\{e\},\{e \mapsto u.v\},\{e \mapsto s\},\varepsilon),\{u \mapsto \{x\}, v \mapsto \{y\}\})\}\\
tgraph(\varphi_1 \vee \varphi_2) :=& tgraph(\varphi_1) \cup tgraph(\varphi_2)\\
tgraph(\varphi_1 * \varphi_2) :=& \Downarrow \{(H_1 \uplus H_2, tag_1 \uplus tag_2,) \mid (H_i, tag_i) \in tgraph(\varphi_i), i \in \{1,2\}\}\\
tgraph(\exists x : \varphi) :=& \Downarrow \{(H, tag') \mid (H, tag) \in tgraph(\varphi),\\
& tag' = \{v \mapsto tag(v) \setminus \{x\} \mid v \in V_H\}\}\\
tgraph(\rho(y_1,...,y_k)) :=& \Downarrow \{((\{v_1,...,v_k\},\{e\},\{e \mapsto v_1 \ldots v_k.tag^{-1}(\mathbf{null})\},\\
& \{e \mapsto X_\rho\},\varepsilon),\{v_i \mapsto y_i \mid i \in [1,k]\})\}, \rho \in Pred,\\
tgraph(x = y \wedge \varphi) :=& \Downarrow \{(H, tag') \mid (H, tag) \in tgraph(\varphi), \text{ where}\\
tag' :=& \{v \mapsto tag(v) \cup \{x,y\} \mid \{x,y\} \cap tag(v) \neq \emptyset\}\\
& \uplus \{v \mapsto tag(v) \mid \{x,y\} \cap tag(v) = \emptyset\}\}\\
\Downarrow (H, tag) :=& \left([H]_{/\sim}, \left\{[v]_{/\sim} \mapsto \bigcup_{v \in [v]_{/\sim}} tag(v) \mid v \in V_H\right\}\right)
\end{aligned}
$$

Fig. 5. Inductive construction of tagged hypergraphs from a given SL formula.

Hence, these corner cases can be removed from the obtained grammar. Theorem 6 also provides a solution for the satisfiability problem: First translate an SL environment into an HRG, then apply Theorem 6 to obtain a DSG and solve the emptiness problem using Lemma 3.

It should be noted that an application of Theorem 6 may increase the size of an HRG exponentially in the number of selectors and the maximal arity of predicates.

7 Tree-Like Separation Logic

The close relationship between SL and HRGs leads to portability of the obtained TLG results to analogous SL results.

The largest SL fragment considered in this paper is $\mathtt{SL_{tl}}$, which is obtained from applying Lemma 2 to tree-like DSGs.

Definition 15. *An* $\mathtt{SL_{tl}}$ formula *is an SL formula* $\varphi(x_1, ..., x_n)$, $n \geq 1$, *meeting the following conditions:*

- Anchoredness*: All points-to 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 points-to assertion of* φ*.*
- Distinctness*:* x_1 *is unequal to the first parameter of every predicate call occurring in* Γ*.*

An SL_{tl} *environment is an SL environment* Γ *where every disjunct of every predicate definition is an* SL_{tl} *formula.*

Theorem 7. *For every* $\mathtt{SL_{tl}}$ *formula* φ *defined over an* $\mathtt{SL_{tl}}$ *environment* Γ *there exists a language-equivalent tree-like DSG* G *with* $L(G) = L(\mathrm{hrg}[\![\varphi, \Gamma]\!]) \cap \mathtt{HC}_{\Sigma}$ *and vice versa.*

Proof (Sketch). Applying the translation $\mathrm{hrg}[\![\varphi, \Gamma]\!]$ from Lemma 2 to φ and Γ, one observes that every disjunct of every predicate definition is translated into production rules mapping to (basic) tree-like HGs. In particular, the first parameter of every translated predicate definition corresponds to an anchor node in each of these HGs. Thus, by the distinctness condition of $\mathtt{SL_{tl}}$, it is guaranteed that two anchor nodes of the resulting DSG are never merged. Hence, the obtained DSG is tree-like. The converse direction is obtained analogously by applying $\mathrm{env}[\![.]\!]$ to a tree-like DSG. □

As a consequence, our results shown for TLGs presented in Section 4 also apply to $\mathtt{SL_{tl}}$. Other fragments of TLGs presented in this paper are transferred to corresponding $\mathtt{SL}$ fragments analogously. We omit a formal proof of Theorem 7 for lack of space and provide an example of an $\mathtt{SL_{tl}}$ formula corresponding to the DSG of our running example instead.

Example 9. Consider the $\mathtt{SL_{tl}}$ formula $\varphi := \sigma(x_1, x_2, x_3, x_4)$ defined over an environment Γ consisting of the following predicate definitions.

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

Applying the translation $\mathrm{hrg}[\![\varphi, \Gamma]\!]$ yields a tree-like DSG generating the same language as the HRG *TLL* shown in Figure 1. 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 16 (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* $\perp_p$ *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 8 *Every* Δ*-DSG with* $\emptyset \neq \Delta \subseteq \Sigma$ *is a TLG.*

Proof. If two anchor nodes of a Δ-DSG G are merged, there exists $t \in \mathtt{D}(G)$, $\delta \in \Delta$ and $u \in V_{yield(t)}$ such that u has two outgoing edges labelled with δ, i.e. $G \notin \mathtt{DSG}$. □

A corresponding SL fragment is defined analogously to $\mathtt{SL_{tl}}$ except that the distinctness condition is replaced by a new condition: There exists at least one selector $s \in \Sigma$ such that every disjunct of every predicate definition contains an assertion $x_1.s \mapsto y$, where x_1 is the first paramter of the predicate and y is an arbitrary variable. In terms of expressiveness, we may compare Δ-DSGs to $\mathtt{SL_{btw}}$ [14], which is, to the best of our knowledge, the largest known fragment of SL with a decidable entailment problem.

Theorem 8. *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.*

Proof (Sketch). Every $\mathtt{SL_{btw}}$ environment over a set of selectors Σ can be normalized such that predicates are connected by their first parameter. Then, applying the translation hrg⟦.⟧ from SL to DSGs (see Lemma 2) yields a language-equivalent Σ-DSG G (see Theorem 6, Lemma 2 and Remark 1).

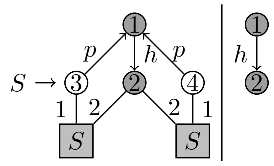


Fig. 6. Tree-like DSG

Conversely, the $\{h\}$-DSG G depicted in Figure 6 generates reversed binary trees with an additional pointer to the head of another data structure. Obviously, $L(G)$ is $\mathtt{SL_{tl}}$-definable but not $\mathtt{SL_{btw}}$-definable, because the number of allocated locations from which the whole heap is reachable is fixed a priori for every $\mathtt{SL_{btw}}$ formula and a corresponding environment. □

8 Conclusion

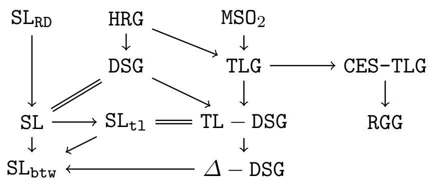


Fig. 7. Fragments of HRG and SL

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. However, the entailment problem or, equivalently, the inclusion problem is undecidable.

We introduced the class TLG of tree-like grammars and showed that every TLG is $\mathtt{MSO_2}$-definable. From this, some remarkable properties, like decidability of the inclusion problem and closure under intersection, directly follow from previous work on context-free and recognisable graph languages [8]. Moreover, the close correspondence between HRGs and SL yields several fragments of SL, in particular $\mathtt{SL_{tl}}$, where an extended entailment problem is decidable. The resulting fragments are more expressive than $\mathtt{SL_{btw}}$, the largest fragment of SL with a decidable entailment problem known so far.

Figure 7 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 $\mathtt{SL_{RD}}$ of Separation Logic with inductive predicate definitions (cf. [14, 1]).

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. We hope that a combined approach – studying $\mathtt{SL}$ as well as HRGs – 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., OâHearn, 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. Büchi, J.R.: Weak second-order arithmetic and finite automata. Mathematical Logic Quarterly **6**(1-6) (1960) 66–92
6. Courcelle, B.: The monadic second-order logic of graphs I: Recognizable sets of finite graphs. Information and Computation **85**(1) (1990) 12–75
7. 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
8. Courcelle, B., Engelfriet, J.: Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach. Number 138. Cambridge University Press (2012)
9. Dodds, M.: From separation logic to hyperedge replacement and back. In: ICGT. Volume 5214 of LNCS. (2008) 484–486
10. Drewes, F., Kreowski, H.J., Habel, A.: Hyperedge replacement graph grammars. In: Handbook of Graph Grammars and Computing by Graph Transformation. (1997) 95–162
11. 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
12. Habel, A.: Hyperedge Replacement: Grammars and Languages. Volume 643 of LNCS. (1992)
13. Heinen, J., Noll, T., Rieger, S.: Juggrnaut: Graph grammar abstraction for unbounded heap structures. ENTCS **266** (2010) 93–107
14. 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
15. 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
16. 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
17. 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
18. Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: LICS. (2002) 55–74
19. Salomaa, A., Rozenberg, G.: Handbook of Formal Languages, Vol. 3: Beyond Words. Springer (1997)

Appendix

Appendix A provides a brief introduction to the filter theorem for HRGs. This theorem is needed in the proofs of Theorem 2 and Theorem 6 which are provided in Appendix B and Appendix C, respectively.

A The Filter Theorem for HRGs

The proofs of Theorem 6 and Theorem 2 apply a version of the filter theorem (Lemma 1) due to Habel [12] which makes use of "compatible" predicates. This section briefly introduces this theorem together with the most important definitions. Details can be found in [12] (Chapter VI).

Let $G = (\Sigma_N, P, S) \in \mathtt{HRG}$. $H \Rightarrow H'$ is a *derivation step* if there exists $e \in E_H^N$ and $(\mathrm{lab}_H(e), K) \in P$ such that $H' \cong H[e/K]$. A derivation $H \Rightarrow^\star H'$ is the application of finitely many derivation steps such that $H' \in \mathtt{HG}_\Sigma$. A *handle* of $X \in N$ is an HG $X^\bullet$ consisting of a single nonterminal hyperedge labelled X attached to $rk_{\Sigma_N}(X)$ distinct nodes. The projection of $H \in \mathtt{HG}_{\Sigma_N}$ to the HG derived from a single nonterminal hyperedge $e \in E_H^N$ is denoted by $H(e)$. Furthermore, for a class $\mathcal{C} \subseteq \mathtt{HRG}$, let $HG^{\mathcal{C}}$ be the set of all HGs generated by some HRG in $\mathcal{C}$.

The following definition is taken verbatim from [12] with minor modifications to fit our notation.

Definition 17 (Compatible Predicates).

1. *Let $\mathcal{C} \subseteq \mathtt{HRG}$, $\mathfrak{I}$ a finite set, called the index set, $\mathtt{PROP}$ a decidable predicate defined on pairs (H, I) with $H \in \mathtt{HG}^{\mathcal{C}}$ and $I \in \mathfrak{I}$, and $\mathtt{PROP'}$ a decidable predicate on triples (R, assign, I) with $R \in \mathtt{HG}^{\mathcal{C}}$, a mapping $E_R^N \to \mathfrak{I}$ and $I \in \mathfrak{I}$. Then $\mathtt{PROP}$ is called $(\mathcal{C}, \mathtt{PROP'})$-*compatible *if for all $G = (\Sigma_N, P, S) \in \mathcal{C}$ and all derivations $X^\bullet \Rightarrow R \Rightarrow^\star H$ with $X \in N$ and $H \in \mathtt{HG}_\Sigma^{\mathcal{C}}$, and for all $I \in \mathfrak{I}$, $\mathtt{PROP}(H, I)$ holds if and only if there is a mapping *assign* $: E_R^N \to \mathfrak{I}$ such that $\mathtt{PROP'}(R, \mathrm{assign}, I)$ holds and $\mathtt{PROP}(H(e), \mathrm{assign}(e))$ holds for all $e \in E_R^N$.*
2. *A predicate $\mathtt{PROP}_0$ on $\mathtt{HG}^{\mathcal{C}}$ is called $\mathcal{C}$-*compatible *if predicates $\mathtt{PROP}$ and $\mathtt{PROP'}$ and an index I_0 exist such that $\mathtt{PROP}$ is $(\mathcal{C}, \mathtt{PROP'})$-compatible and $\mathtt{PROP}_0(H) = \mathtt{PROP}(H, I_0)$ for all $H \in \mathtt{HG}^{\mathcal{C}}$.*

For every $\mathcal{C}$-compatible predicate $\mathtt{PROP}_0$ and $G \in \mathcal{C}$, one can construct a new HRG generating all graphs generated by G which additionally satisfy $\mathtt{PROP}_0$.

Lemma 9 (Filter Theorem for HRGs [12]) *Let $\mathtt{PROP}_0$ be a $\mathcal{C}$-compatible predicate for some class $\mathcal{C}$ of HRGs. For every HRG $G \in \mathcal{C}$, there is an HRG $G_{\mathtt{PROP}_0}$ such that $L(G_{\mathtt{PROP}_0}) = \{H \in L(G) \mid \mathtt{PROP}_0(H) \text{ holds}\}$.*

Moreover, for each production rule p of the HRG $G_{\mathtt{PROP}_0}$, $\mathrm{rhs}(p)$ is isomorphic to some $\mathrm{rhs}(q)$ up to the names of nonterminals, where q is a production rule of G. The size of the new grammar is linear in G and $|\mathfrak{I}|$.

B Proof of Theorem 2

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

Let $\mathcal{T}$ denote the class of all HRGs in which every production rule maps to a (basic) tree-like hypergraph. Given $G \in \mathcal{T}$, we construct a $\mathcal{T}$-compatible predicate $\mathtt{MRG}_0^G(H)$ such that for all $H \in \mathtt{HG}^{\mathcal{T}}$, $\mathtt{MRG}_0^G(H)$ holds if and only if $H \notin \mathcal{M}(G)$. Then, Theorem 2 is a consequence of Lemma 9.

In the following we will see that it is decidable whether $H \in \mathcal{M}(G)$ holds.

Lemma 10 *Let $G \in \mathcal{T}$, $H \in L(G)$ and $\mathtt{D}(G)_H := \{t \in \mathtt{D}(G) \mid \mathit{yield}(t) \cong H\}$. Then (1) $\mathtt{D}(G)_H$ is a finite set and (2) it is decidable whether $H \in \mathcal{M}(G)$.*

Proof. (1) By definition, every production rule p of G maps to a (basic) tree-like hypergraph. Hence, $rhs(p)$ is either a single external node or contains at least one terminal edge. Thus, $|\mathrm{dom}(t)| \leq 2 \cdot |E_H|$ and $\mathtt{D}(G)_H \subseteq \{t \in \mathtt{D}(G) \mid |\mathrm{dom}(t)| \leq 2 \cdot |E_H|\}$ is finite.

(2) Given a derivation tree $t \in \mathtt{D}(G)$ with $\mathit{yield}(t) \cong H$, it is straightforward to check whether $H \in \mathcal{M}(G)$, e.g. by applying Lemma 7 to each pair $x, y \in \mathrm{dom}(t)$ to check whether their anchor nodes are merged. Due to (1), it suffices to check finitely many derivation trees to obtain either $H \in \mathcal{M}(G)$ or $H \notin \mathcal{M}(G)$. $\square$

We now construct $\mathtt{MRG}_0^G(H)$ together with the predicates required by Definition 17. Let $\mathfrak{I} := 2^{[\Re(G)]}$, i.e. every index $I \in \mathfrak{I}$ is a set of integers in $[\Re(G)]$. Intuitively, I remembers which nodes attached to a nonterminal hyperedge have already been merged with anchor nodes. We define two auxiliary sets for a given set $I \in \mathfrak{I}$ and an HG H:

- The set of all nodes in H representing anchor nodes is given by

$$\perp (H) := \{v \in V_H \mid \exists t \in \mathtt{D}(G)_H, x \in \mathrm{dom}(t) : v \approxeq_t \mathrm{ext}_{t(x)}(1)\}.$$

- The set of all nodes attached to e as well as marked by assign is given by

$$V(e) := \{\mathrm{att}_H(e)(j) \mid j \in \mathrm{assign}(e) \cap [rk_{\Sigma_N}(lhs(e))]\} \text{ for each } e \in E_H^N.$$

Now, $\mathtt{MRG'}^G(R, \mathrm{assign}, I)$ holds if and only if $V(e_i) \cap V(e_j) = \emptyset$ for each $i \neq j$, and $\perp_R \notin V(e_i)$, i.e. there exists no pair of nodes already known to be merged with anchor nodes (according to assign and I) which are merged in R. The predicate $\mathtt{MRG}^G(H, I)$ holds if and only if $H \notin \mathcal{M}(G)$ and $I = \perp (H) \cap \lfloor \mathrm{ext}_H \rfloor$. Furthermore, we set $\mathtt{MRG}_0^G(H) := \mathtt{MRG}^G(H, \perp (H) \cap \lfloor \mathrm{ext}_H \rfloor)$.

Lemma 11 *For each $G \in \mathcal{T}$, $\mathtt{MRG}_0^G(H)$ is a $\mathcal{T}$-compatible predicate.*

Proof. Let $p = (X, R)$ be a production rule with $E_R^N = \{e_1, ..., e_n\}$ such that $H \cong R[e_1/H(e_1), \ldots, e_n/H(e_n)]$. We have to show that $\mathtt{MRG}^G(H, I)$ holds if and only if there exists a mapping assign : $E_R^N \to \mathfrak{I}$ such that $\mathtt{MRG'}^G(R, \mathrm{assign}, I)$ holds and $\mathtt{MRG}^G(H(e_i), \mathrm{assign}(e_i))$ holds for each $i \in [n]$.

"$\Rightarrow$": Assume $\mathtt{MRG}^G(H, I)$ holds. We choose assign : $E_R^N \to \mathfrak{I}$ such that $e_i \mapsto \perp (H(e_i)) \cap \lfloor \mathrm{ext}_{H(e_i)} \rfloor$ for each $i \in [n]$. By definition, $H \notin \mathcal{M}(G)$, i.e. $H(e_i) \notin \mathcal{M}(G)$ for each $i \in [1, n]$. Now, assume $\perp_R \in V(e_i)$ for some $i \in [1, n]$. Then, the anchor of R is merged with some node in $V(e_i)$ and by definition of $V(e_i)$, two anchor nodes are merged, i.e. $H \in \mathcal{M}(G)$. Analogously, if there are $1 \leq i < j \leq n$ such that $V_{e_i} \cap V_{e_j} \neq \emptyset$, there is a node $v \in V_R$ which is merged with an anchor node in $H(e_i)$ and $H(e_j)$. Since $H(e_i)$ and $H(e_j)$ have disjoint nodes and edges, this implies $H \in \mathcal{M}(G)$. Both cases lead to a contradiction, i.e. $\mathtt{MRG'}^G(R, \mathrm{assign}, I)$ holds.

"$\Leftarrow$": Conversely, assume there exists a mapping assign : $E_R^N \to \mathfrak{I}$ such that $\mathtt{MRG'}^G(R, \mathrm{assign}, I)$ holds and $\mathtt{MRG}^G(H(e_i), \mathrm{assign}(e_i))$ holds for each $i \in [n]$. By definition, we have $H(e_i) \notin \mathcal{M}(G)$ for each $i \in [n]$. Thus, if $H \in \mathcal{M}(G)$, two cases are possible:

1. Two nodes u, v with $u \in \downarrow (H(e_i))$, $v \in \downarrow (H(e_j))$, $1 \leq i < j \leq n$ are merged.
2. $\downarrow_R$ is merged with some $v \in \downarrow (H(e_i))$, $i \in [1, n]$.

In the first case, there is a node $w \in V_R$ that is merged with u as well as with v, because $H(e_i)$,$H(e_i)$ are disjoint subgraphs of H. Hence, $w \in V(e_i) \cap V(e_j)$. This is a contradiction, because $V(e_i) \cap V(e_j) = \emptyset$ by definition. The second case requires $\downarrow_R \in V(e_i)$ which is also impossible by definition.

Since, the predicates $\texttt{MRG}^G(H, I)$ and $\texttt{MRG'}^G(R, \text{assign}, I)$ are decidable by Lemma 10, it follows that $\texttt{MRG}_0^G(H)$ is a $\mathcal{T}$-compatible predicate. □

It remains to complete the proof of Theorem 2.

Proof. By definition, $\texttt{MRG}_0^G(H) = \texttt{MRG}^G(H, \downarrow (H) \cap \lfloor \text{ext}_H \rfloor)$ holds if and only if $H \in \mathcal{M}(G)$, because the second condition is always satisfied for $I = \downarrow (H) \cap \lfloor \text{ext}_H \rfloor$. Since $\texttt{MRG}_0^G(H)$ is a $\mathcal{T}$-compatible predicate for any HRG $G \in \mathcal{T}$, the claim follows directly from Lemma 9. □

C Proof of Theorem 6

Theorem 6. *For every $G \in$ HRG, an HRG G' can be constructed such that $L(G') = L(G) \cap \texttt{HC}_\Sigma$.*

Let $\mathcal{C}_\Re$ be the class of all HRGs G with $\Re(G) \leq \Re$ and $L(G) \subseteq \texttt{HG}_\Sigma$ for some finite alphabet of terminal symbols Σ. We construct a $\mathcal{C}_\Re$-compatible predicate $\texttt{HCP}_0(H)$ such that for each $H \in \texttt{HC}^{\mathcal{C}_\Re}$, $\texttt{HCP}_0(H)$ holds if and only if $H \in \texttt{HC}$. Then, the claim follows from Lemma 9.

Given $H \in \texttt{HG}_\Sigma$ and $v \in V_H$, we define the auxiliary set $\Gamma(v) := \{s \in \Sigma \mid \exists e \in E : \text{att}(e)(1) = v, \text{lab}_H(e) = s\}$ collecting all labels of outgoing edges of v. The index set of the predicates we have to construct is defined as $\mathfrak{I} := (2^\Sigma)^{\leq \Re}$. Intuitively, every $I \in \mathfrak{I}$ records all terminal outgoing edges that have been added to an external node in previous derivation steps. Analogously, assign(e) collects all terminal symbols for each node v attached to $e \in E_H^N$, outgoing edges of v introduced in further derivation steps may be labelled with without violating the HC property. Now, for each $H \in \texttt{HG}_\Sigma$, we define $\texttt{HCP'}(R, \text{assign}, I)$ to hold if and only if

1. $\text{att}_H(e_1)(1) \neq \text{att}_H(e_2)(1)$ for all $e_1, e_2 \in E_H^\Sigma$ with $\text{lab}_H(e_1) = \text{lab}_H(e_2)$,
2. $\Gamma(\text{ext}_H(k)) \cap I(k) = \emptyset$ for each $k \in [n]$ where $n = |I| = |\text{ext}_H|$,
3. $\Gamma(\text{att}_H(e)(k)) \subseteq \text{assign}(e)(k)$ for each $k \in [rk_{\Sigma_N}(e)]$, $e \in E_H^N$.

Moreover, we define $\texttt{HCP}(H, I) := \texttt{HCP'}(H, \emptyset, I)$ and $\texttt{HCP}_0(H) = \texttt{HCP}(H, \emptyset^{|\text{ext}_H|})$. Clearly, each of these predicates is decidable.

Lemma 12 *$\texttt{HCP}_0(H)$ holds for $H \in \texttt{HG}_{\Sigma_N}^{\mathcal{C}_\Re}$ if and only if $H \in \texttt{HC}_{\Sigma_N}^{\mathcal{C}_\Re}$.*

Proof. If $H \in \texttt{HC}_{\Sigma_N}^{\mathcal{C}_\Re}$, every $v \in V_H$ has at most one outgoing terminal edge $e \in E_H$ with $\text{lab}_H(e) = s$ for each $s \in \Sigma$ which satisfies the first condition. Since $I = \emptyset^{|\text{ext}_H|}$, the second condition is trivially satisfied.

Conversely, if $\texttt{HCP}_0(H)$ holds, we have $\text{att}_H(e_1)(1) \neq \text{att}_H(e_2)(1)$ for all $e_1, e_2 \in E_H$ with $\text{lab}_H(e_1) = \text{lab}_H(e_2) \in \Sigma$ by the first condition. Thus, $H \in \texttt{HC}_{\Sigma_N}^{\mathcal{C}_\Re}$ □

Lemma 13 *$\mathtt{HCP}_0(H)$ is a $\mathcal{C}_{\mathfrak{R}}$-compatible predicate.*

Proof. Let $S^\bullet \Rightarrow R \Rightarrow^\star H$ and $I \in \mathfrak{I}$. We have to show for any $H \in \mathtt{HC}_{\Sigma_N}^{\mathcal{C}_{\mathfrak{R}}}$ that $\mathtt{HCP}_0(H)$ holds if and only if there exists a mapping $\text{assign} : E_R^N \to I$ such that $\mathtt{HCP'}(R, \text{assign}, I)$ holds and for each $e \in E_R^N$, $\mathtt{HCP}(H(e), \text{assign}(e))$ holds.

"$\Rightarrow$": Assume $\mathtt{HCP}_0(H)$ holds. Then the first two conditions of $\mathtt{HCP'}(R, \text{assign}, I)$ are satisfied regardless of our choice of assign. We choose $\text{assign} : E_R^N \to I$ such that the k-th component of $\text{assign}(e)$ contains all $s \in \Sigma$, the k-th attached node of e has an outgoing edge labelled with. Thus, $\text{assign}(e)(k) := \{\text{lab}_R(e') \mid \text{att}_R(e)(k) \in \lfloor \text{att}_R(e') \rfloor\}$ for each $k \in [rk_{\Sigma_N}(e)]$. This trivially satisfies the third condition of $\mathtt{HCP'}(R, \text{assign}, I)$. Moreover, since H satisfies the first condition, $\mathtt{HCP}(H(e), \text{assign}(e))$ holds for each $e \in E_R^N$.

"$\Leftarrow$": Assume there is a function $\text{assign} : E_R^N \to I$ such that $\mathtt{HCP'}(R, \text{assign}, I)$ holds and for each $e \in E_R^N$, $\mathtt{HCP}(H(e), \text{assign}(e))$ holds. Then $H(e) \in \mathtt{HC}_\Sigma$ and $\text{lab}_{H(e)}(\text{ext}_{H(e)}(k)) \cap \text{assign}(e)(k) = \emptyset$ for each $k \in [|\text{ext}_H|]$. By definition, $\text{lab}(\text{att}_R(e)(k)) \subseteq \text{assign}(e)(k)$ for each $k \in [rk_{\Sigma_N}(e)]$. Thus, replacing every nonterminal hyperedge $e \in E_R^N$ by $H(e)$ yields a heap configuration, i.e. $\mathtt{HCP}_0(H)$ holds. □

Proof (of Theorem 6). Let $G = (\Sigma_N, P, S) \in \mathtt{HRG}$ and $rk_{\Sigma_N}(s) = 2$ for each $s \in \Sigma$. Then, $G \in \mathcal{C}_{\mathfrak{R}(G)}$. By Lemma 13, $\mathtt{HCP}_0(H)$ is a $\mathcal{C}_{\mathfrak{R}(G)}$-compatible predicate and by Lemma 12, $\mathtt{HCP}_0(H)$ holds if and only if $H \in \mathtt{HC}_{\Sigma_N}^{\mathcal{C}_{\mathfrak{R}(G)}}$. Thus, applying Lemma 9 yields a DSG G' with $L(G') = \{H \in L(G) \mid \mathtt{HCP}_0(H) \text{ holds}\}$. □

In particular, the construction of G' is linear in the size of G and $|\mathfrak{I}|$ (cf. the proof of Theorem 5.1 in [12]).

Aachener Informatik-Berichte

This list contains all technical reports published during the past three years. A complete list of reports dating back to 1987 is available from:

`http://aib.informatik.rwth-aachen.de/`

To obtain copies please consult the above URL or send your request to:

Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen, Email: `biblio@informatik.rwth-aachen.de`

2012-01	Fachgruppe Informatik: Annual Report 2012
2012-02	Thomas Heer: Controlling Development Processes
2012-03	Arne Haber, Jan Oliver Ringert, Bernhard Rumpe: MontiArc - Architectural Modeling of Interactive Distributed and Cyber-Physical Systems
2012-04	Marcus Gelderie: Strategy Machines and their Complexity
2012-05	Thomas Ströder, Fabian Emmes, Jürgen Giesl, Peter Schneider-Kamp, and Carsten Fuhs: Automated Complexity Analysis for Prolog by Term Rewriting
2012-06	Marc Brockschmidt, Richard Musiol, Carsten Otto, Jürgen Giesl: Automated Termination Proofs for Java Programs with Cyclic Data
2012-07	André Egners, Björn Marschollek, and Ulrike Meyer: Hackers in Your Pocket: A Survey of Smartphone Security Across Platforms
2012-08	Hongfei Fu: Computing Game Metrics on Markov Decision Processes
2012-09	Dennis Guck, Tingting Han, Joost-Pieter Katoen, and Martin R. Neuhäußer: Quantitative Timed Analysis of Interactive Markov Chains
2012-10	Uwe Naumann and Johannes Lotz: Algorithmic Differentiation of Numerical Methods: Tangent-Linear and Adjoint Direct Solvers for Systems of Linear Equations
2012-12	Jürgen Giesl, Thomas Ströder, Peter Schneider-Kamp, Fabian Emmes, and Carsten Fuhs: Symbolic Evaluation Graphs and Term Rewriting — A General Methodology for Analyzing Logic Programs
2012-15	Uwe Naumann, Johannes Lotz, Klaus Leppkes, and Markus Towara: Algorithmic Differentiation of Numerical Methods: Tangent-Linear and Adjoint Solvers for Systems of Nonlinear Equations
2012-16	Georg Neugebauer and Ulrike Meyer: SMC-MuSe: A Framework for Secure Multi-Party Computation on MultiSets
2012-17	Viet Yen Nguyen: Trustworthy Spacecraft Design Using Formal Methods
2013-01 *	Fachgruppe Informatik: Annual Report 2013
2013-02	Michael Reke: Modellbasierte Entwicklung automobiler Steuerungssysteme in Klein- und mittelständischen Unternehmen
2013-03	Markus Towara and Uwe Naumann: A Discrete Adjoint Model for OpenFOAM

2013-04 Max Sagebaum, Nicolas R. Gauger, Uwe Naumann, Johannes Lotz, and Klaus Leppkes: Algorithmic Differentiation of a Complex C++ Code with Underlying Libraries

2013-05 Andreas Rausch and Marc Sihling: Software & Systems Engineering Essentials 2013

2013-06 Marc Brockschmidt, Byron Cook, and Carsten Fuhs: Better termination proving through cooperation

2013-07 André Stollenwerk: Ein modellbasiertes Sicherheitskonzept für die extrakorporale Lungenunterstützung

2013-08 Sebastian Junges, Ulrich Loup, Florian Corzilius and Erika Ábrahám: On Gröbner Bases in the Context of Satisfiability-Modulo-Theories Solving over the Real Numbers

2013-10 Joost-Pieter Katoen, Thomas Noll, Thomas Santen, Dirk Seifert, and Hao Wu: Performance Analysis of Computing Servers using Stochastic Petri Nets and Markov Automata

2013-12 Marc Brockschmidt, Fabian Emmes, Stephan Falke, Carsten Fuhs, and Jürgen Giesl: Alternating Runtime and Size Complexity Analysis of Integer Programs

2013-13 Michael Eggert, Roger Häußling, Martin Henze, Lars Hermerschmidt, René Hummen, Daniel Kerpen, Antonio Navarro Pérez, Bernhard Rumpe, Dirk Thißen, and Klaus Wehrle: SensorCloud: Towards the Interdisciplinary Development of a Trustworthy Platform for Globally Interconnected Sensors and Actuators

2013-14 Jörg Brauer: Automatic Abstraction for Bit-Vectors using Decision Procedures

2013-16 Carsten Otto: Java Program Analysis by Symbolic Execution

2013-19 Florian Schmidt, David Orlea, and Klaus Wehrle: Support for error tolerance in the Real-Time Transport Protocol

2013-20 Jacob Palczynski: Time-Continuous Behaviour Comparison Based on Abstract Models

2014-01 * Fachgruppe Informatik: Annual Report 2014

2014-02 Daniel Merschen: Integration und Analyse von Artefakten in der modellbasierten Entwicklung eingebetteter Software

2014-03 Uwe Naumann, Klaus Leppkes, and Johannes Lotz: dco/c++ User Guide

2014-04 Namit Chaturvedi: Languages of Infinite Traces and Deterministic Asynchronous Automata

2014-05 Thomas Ströder, Jürgen Giesl, Marc Brockschmidt, Florian Frohn, Carsten Fuhs, Jera Hensel, and Peter Schneider-Kamp: Automated Termination Analysis for Programs with Pointer Arithmetic

2014-06 Esther Horbert, Germán Martín García, Simone Frintrop, and Bastian Leibe: Sequence Level Salient Object Proposals for Generic Object Detection in Video

2014-07 Niloofar Safiran, Johannes Lotz, and Uwe Naumann: Algorithmic Differentiation of Numerical Methods: Second-Order Tangent and Adjoint Solvers for Systems of Parametrized Nonlinear Equations

2014-08 Christina Jansen, Florian Göbe, and Thomas Noll: Generating Inductive Predicates for Symbolic Execution of Pointer-Manipulating Programs

2014-09 Thomas Ströder and Terrance Swift (Editors): Proceedings of the International Joint Workshop on Implementation of Constraint and Logic Programming Systems and Logic-based Methods in Programming Environments 2014

2014-14 Florian Schmidt, Matteo Ceriotti, Niklas Hauser, and Klaus Wehrle: HotBox: Testing Temperature Effects in Sensor Networks

2014-15 Dominique Gückel: Synthesis of State Space Generators for Model Checking Microcontroller Code

2014-16 Hongfei Fu: Verifying Probabilistic Systems: New Algorithms and Complexity Results

2015-01 * Fachgruppe Informatik: Annual Report 2015

2015-02 Dominik Franke: Testing Life Cycle-related Properties of Mobile Applications

2015-05 Florian Frohn, Jürgen Giesl, Jera Hensel, Cornelius Aschermann, and Thomas Ströder: Inferring Lower Bounds for Runtime Complexity

2015-06 Thomas Ströder and Wolfgang Thomas (Editors): Proceedings of the Young Researchers' Conference "Frontiers of Formal Methods"

2015-07 Hilal Diab: Experimental Validation and Mathematical Analysis of Cooperative Vehicles in a Platoon

2015-08 Mathias Pelka, Jó Agila Bitsch, Horst Hellbrück, and Klaus Wehrle (Editors): Proceedings of the 1st KuVS Expert Talk on Localization

2015-09 Xin Chen: Reachability Analysis of Non-Linear Hybrid Systems Using Taylor Models

* These reports are only available as a printed version.

Please contact `biblio@informatik.rwth-aachen.de` to obtain copies.