Regulated Tree Automata

Henning Fernau, Martin Vu

Regulated Tree Automata

Henning Fernau and Martin Vu

Universität Trier, FB IV—Abteilung Informatikwissenschaften, CIRT,
54286 Trier, Germany; {fernau,s4vivuuu}@uni-trier.de

Abstract. Regulated rewriting is one of the classical areas in Formal Languages, as tree automata are a classical topic. Somewhat surprisingly, there have been no attempts so far to combine both areas. Here, we start this type of research, introducing regulated tree automata, proving in particular characterizations of the yields of such regulated automata.

Keywords: Regulated rewriting, graph control, tree automata, yield operation

1 Introduction

The area of regulated rewriting, still well-covered by the classical monograph of Dassow and Păun [4], drew its main motivation from the idea to enrich context-free grammars with certain control mechanisms in order to be able to model linguistic features that are not expressible with traditional context-free grammars, yet keeping at least some of the beauties of these. More specifically, programmed grammars, matrix grammars, and grammars with regular control were introduced and studied around 1970. It soon became clear that control involving so-called appearance checks tends to be too powerful in the sense that all recursively enumerable languages can be characterized this way. This somewhat counter-acts the idea of keeping some of the advantages of context-free grammars over, say, Turing machines. Therefore, we are mainly focusing on models without appearance checks in the following. Applications and motivations are also underlined in the relatively recent monograph by Meduna and Zemek [15].

Conversely, finite tree automata have been invented to allow for processing (mostly ordered) trees (as opposed to strings) in a simple manner. Trees not only showed up as a kind of intermediate data structure within compilers, but they are a ubiquitous data structure when it comes to processing semi-structured documents and also for working with natural languages [12,2,13].

Recall the basic well-known link between context-free grammars and derivation trees, often established in practice via considering pushdown automata. However, it is not possible to go this way in connection with regulated rewriting, as already observed by Meduna and Kolar in [14]. The basic reason is that the work of pushdown automata rather corresponds to leftmost (or rightmost) derivations in a very strict sense (called leftmost-1 in [4]). Yet, this strict interpretation does not increase the descriptive power of context-free languages, which counter-acts one of the basic motivations for considering regulated rewriting.

Rather, we are going to follow here the path pioneered by Doner, Thatcher (and also to Wright) [5,16] who showed that the context-free string languages are just the languages that can be obtained by mapping the tree language accepted by some finite tree automaton to its so-called yield, which means that, given an ordered tree, we read the labels of the leaves from left to right. We will obtain similar results for regulated tree grammars in this paper. As tree automata usually come in two working modes (top-down versus bottom-up), our studies also revive the question of generating versus accepting (or analyzing) grammars [1].

We are going to present basic results concerning regulated tree automata and their yields. Due to reasons of space, (straightforward) formal induction arguments are not given here. Most of these can be found in [17].

2 Definitions

2.1 Classical Regulated Rewriting

There are several ways to introduce the basic control mechanisms of regulated rewriting. We are giving a simplified exposition now, basically following [6], adapted to the case of not allowing appearance checks.

A *graph-controlled grammar* is an 8-tuple $G = (V_N, V_T, P, S, \Gamma, \Sigma, \Phi, h)$ where

- (V_N, V_T, P, S) define, as in a phrase structure grammar, the set of nonterminals, terminals, context-free core rules, and the start symbol, respectively;
- Γ is a digraph, i.e., $\Gamma = (U, E)$, with $E \subseteq U \times U$;
- $\Sigma \subseteq U$ are the initial vertices;
- $\Phi \subseteq U$ are the final vertices;
- $h : U \to (2^P \setminus \{\emptyset\})$ relates vertices with rule sets.

We say that $(x, u) \Rightarrow (y, v)$ holds in G with $(x, u), (y, v) \in (V_N \cup V_T)^* \times U$ if, for some $x_1, x_2, \alpha, \beta \in (V_N \cup V_T)^*$,

$$x = x_1 \alpha x_2, \quad y = x_1 \beta x_2, \quad \alpha \to \beta \in h(u), \quad \text{and} \quad (u, v) \in E\,.$$

The reflexive transitive closure of $\Rightarrow$ is denoted by $\overset{*}{\Rightarrow}$. The language generated by G (where P contains only context-free (generating, non-erasing) rules from $V_N \times (V_N \cup V_T)^+$) is defined by

$$L^{gen}(G) = \{x \in V_T^* \mid \exists u \in \Sigma\, \exists v \in \Phi\, ((S, u) \overset{*}{\Rightarrow} (x, v))\}\,.$$

The corresponding language family is written $\mathcal{L}^{gen}(\mathrm{G}, \mathrm{CF}-\varepsilon)$, as we do not allow ε-rules. If P contains only context-free (accepting) rules from $(V_N \cup V_T)^+ \times V_N$, then the language accepted by G is defined by

$$L^{acc}(G) = \{x \in V_T^* \mid \exists u \in \Sigma\, \exists v \in \Phi\, ((x, u) \overset{*}{\Rightarrow} (S, v))\}\,,$$

yielding the language family $\mathcal{L}^{acc}(\mathrm{G}, \mathrm{CF} - \varepsilon)$.

We consider three special cases of graph-controlled grammars in the following.

- A *grammar with regular control* is a graph-controlled grammar where every vertex contains exactly one rule. Usually, these grammars are introduced via regular control languages, but the correspondance with automata graphs is obvious. By $\mathcal{L}^{gen}(\mathrm{rC},\mathrm{CF}-\varepsilon)$, the family of languages generated by context-free grammars with regular control is denoted.
- A *programmed grammar* is a grammar with regular control with no designated initial or final vertices, i.e., formally $\Sigma = \Phi = U$. This means that it is possible to start a derivation in each vertex containing a rule whose left-hand side equals the start symbol S, and it is possible to stop anywhere when a terminal string has been derived. As language families, we obtain, e.g., $\mathcal{L}^{gen}(\mathrm{P},\mathrm{CF}-\varepsilon)$.
- A *matrix grammar* is a grammar with regular control obeying the additional restriction:
  - Initial and final vertices coincide. Only the initial vertices (not necessarily containing rules with left-hand side S) are allowed to have more than one in-going arc. Only predecessors of final vertices are allowed to have more than one out-going arc. Moreover, between every predecessor of a final vertex and every initial vertex, there is an arc.

  As language families, we obtain, e.g., $\mathcal{L}^{gen}(\mathrm{M},\mathrm{CF}-\varepsilon)$.

With literally the same restrictions, we can define, for instance, $\mathcal{L}^{acc}(\mathrm{M},\mathrm{CF}-\varepsilon)$. Recall that all language families introduced in this subsection coincide [4,1].

Remark 1. The formalization of regular control is possibly most different from the one found in traditional textbooks. However, if $\Gamma = (U, E)$ together with Σ, Φ, h defines the control graph structure, then we can relate a finite automaton A with state set U as follows: We have a transition (u, r, v) if $h(u) = \{r\}$; Σ is the set of initial states, and if $u \in \Phi$ and $(u, v) \in E$, then v is a final state of A. Now, if $u_1, u_2, \ldots, u_n$ describes a directed path from $u_1 \in \Sigma$ to $u_n \in \Phi$, then via $\{r_i\} = h(u_i)$ this corresponds to a sequence of rules $r_1 r_2 \ldots r_n$, which, when fed into A, will be accepted. Also the converse construction is possible. Hence, we can in particular assume that the automaton A that describes the set of permitted rule sequences is deterministic.

2.2 Tree Automata

Let $\mathbb{N}$ be the set of nonnegative integers and let $(\mathbb{N}^*, \cdot, \varepsilon)$ (or simply $\mathbb{N}^*$) be the free monoid generated by $\mathbb{N}$. For $y, x \in \mathbb{N}^*$, we write $y \leq x$ iff there is a $z \in \mathbb{N}^*$ with $x = y \cdot z$. "$y < x$" abbreviates: $y \leq x$ and $y \neq x$. As usual, $|x|$ denotes the length of the word x.

We now give the necessary definitions for trees and tree automata. More details can be found, e.g., in [2], where also many examples can be found.

A *ranked alphabet* V is a finite set of symbols together with a finite relation called rank relation $r_V \subset V \times \mathbb{N}$. Define $V_n := \{f \in V \mid (f, n) \in r_V\}$. Since elements in V_n are often considered as *function symbols* (standing for functions of *arity* n), elements in V_0 are also called *constant symbols*. A *tree over* V is a

mapping $t : \Delta_t \to V$, where the domain Δ_t is a finite subset of $\mathbb{N}^*$ such that (1) if $x \in \Delta_t$ and $y < x$, then $y \in \Delta_t$; (2) if $y \cdot i \in \Delta_t$, $i \in \mathbb{N}$, then $y \cdot j \in \Delta_t$ for $1 \leq j \leq i$. An element of Δ_t is also called a *node* of t, where the node ε is the *root* of the tree. Then $t(x) \in V_n$ whenever, for $i \in \mathbb{N}$, $x \cdot i \in \Delta_t$ iff $1 \leq i \leq n$. If $t(x) = A$, A is the *label* of x. Let $V^{\mathtt{t}}$ denote the set of all finite trees over V. By this definition, trees are rooted, directed, acyclic graphs in which every node except the root has one predecessor and the direct successors of any node are linearly ordered from left to right. Interpreting V as a set of function symbols, $V^{\mathtt{t}}$ can be identified with the well-formed terms over V. A *frontier node* in t is a node $y \in \Delta_t$ such there is no $x \in \Delta_t$ with $y < x$. If $y \in \Delta_t$ is not a frontier node, it is called *interior node*. The *depth* of a tree t is defined as $\mathrm{depth}(t) = \max\{|x| \mid x \in \Delta_t\}$, whereas the *size* of t is given by $|\Delta_t|$. Letters will be viewed as trees of size one and depth zero.

We are now going to define a catenation on trees. Let \$ be a new symbol, i.e., $\$ \notin V$, of rank 0. Let $V_{\$}^{\mathtt{t}}$ denote the set of all trees over $V \cup \{\$\}$ which contain exactly one occurrence of label \$. By definition, only frontier nodes can carry the label \$. For trees $u \in V_{\$}^{\mathtt{t}}$ and $t \in (V^{\mathtt{t}} \cup V_{\$}^{\mathtt{t}})$, we define an operation $\#$ to replace the frontier node labelled with \$ of u by t according to

$$u\#t(x) = \begin{cases} u(x), \text{ if } x \in \Delta_u \wedge u(x) \neq \$, \\ t(y), \text{ if } x = z \cdot y \wedge u(z) = \$ \wedge y \in \Delta_t. \end{cases}$$

If $U \subseteq V_{\$}^{\mathtt{t}}$ and $T \subseteq (V^{\mathtt{t}} \cup V_{\$}^{\mathtt{t}})$, then $U\#T := \{u\#t \mid u \in U \wedge t \in T\}$. For $t \in V^{\mathtt{t}}$ and $x \in \Delta_t$, the *subtree* of t at x, denoted by t/x, is defined by $t/x(y) = t(x \cdot y)$ for any $y \in \Delta_{t/x}$, where $\Delta_{t/x} := \{y \mid x \cdot y \in \Delta_t\}$. $\mathrm{ST}(T) := \{t/x \mid t \in T \wedge x \in \Delta_t\}$ is the set of subtrees of trees from $T \subseteq V^{\mathtt{t}}$. Furthermore, for any $t \in V^{\mathtt{t}}$ and any tree language $T \subseteq V^{\mathtt{t}}$, the *quotient* of T and t is defined as:

$$U_T(t) := \begin{cases} \{u \in V_{\$}^{\mathtt{t}} \mid u\#t \in T\}, & \text{if } t \in V^{\mathtt{t}} \setminus V_0, \\ t, & \text{if } t \in V_0. \end{cases}$$

Let V be a ranked alphabet and m be the maximum rank of the symbols in V. A *(bottom-up) (finite-state) tree automaton* over V is a quadruple $A = (Q, V, \delta, F)$ such that Q is a finite state alphabet (disjoint with V_0), $F \subseteq Q$ is a set of final states, and $\delta = (\delta_0, \ldots, \delta_m)$ is an $m+1$-tuple of state transition functions, where $\delta_0(a) = \{a\}$ for $a \in V_0$ and $\delta_k : V_k \times (Q \cup V_0)^k \to 2^Q$ for $k = 1, \ldots, m$. In this definition, the constant symbols at the frontier nodes are taken as sort of initial states. Now, a transition relation (also denoted by δ) can be recursively defined on $V^{\mathtt{t}}$ by letting

$$\delta(f(t_1, \ldots, t_k)) := \begin{cases} \{f\}, & \text{if } k = 0, \\ \bigcup_{q_i \in \delta(t_i), i=1,\ldots,k} \delta_k(f, q_1, \ldots, q_k), & \text{if } k > 0. \end{cases}$$

A tree t is accepted by A iff $\delta(t) \cap F \neq \emptyset$. The tree language accepted by A is denoted by $L^{\mathtt{t}}(A)$. A is *deterministic* if each of the functions δ_k maps each possible argument to a set of cardinality at most one. Deterministic tree automata can be viewed as algorithms for labelling the nodes of a tree with states. Analogously to the case of string automata, it can be shown that nondeterministic

and deterministic bottom-up finite-state tree automata accept the same class of tree languages, namely the *regular tree languages*, at the expense of a possibly exponential state explosion.

Rules are sometimes also written like $f(q_1, \ldots, q_k) \to q$ instead of saying that $q \in \delta_k(f, q_1, \ldots, q_k)$. Sometimes, also *$\varepsilon$-moves* are allowed, written like $q' \to q$, i.e., no part of the tree is consumed, only the state is changed. As in the string case, finite tree automata with ε-moves only accept regular tree languages.

An alternative view on the work of tree automata is that of labelling a tree with states. To this end, we will formally view all symbols from Q as having rank zero, so that they may serve as labels of frontier nodes. Now, A (or more specifically, its transition function δ) defines a derivation relation $\vdash_\delta$ on $(V \cup Q)^{\mathtt{t}}$ by $s \vdash_\delta t$ if $s \neq t$ and there are trees $u \in (V \cup Q)^{\mathtt{t}}_{\$}$, $s' = f(q_1, \ldots, q_k)$, $t' = q$ with $s = u\#s'$, $t = u\#t'$, $f \in V_k$, $q \in \delta_k(f, q_1, \ldots, q_k)$. Clearly, $s \in V^{\mathtt{t}}$ is accepted by a tree automaton $A = (Q, V, \delta, F)$ if $s \vdash^*_\delta q_f$ for some $q_f \in F$. If we consider δ as a set of rules, it makes also sense to define $s \vdash_{\delta'} t$ for subsets of rules $\delta' \subseteq \delta$. We will use this notation when defining regulated tree automata.

It is also possible to define tree automata $A = (Q, V, \delta, I)$ that work top-down. Rules are now of the form $q \to f(q_1, \ldots, q_k)$, and the derivation relation basically reverses the arrows. Hence, also finite top-down tree automata characterize the regular tree languages. However, deterministic finite top-down tree automata are a strictly weaker model.

We already informally recalled the Theorem of Doner, Thatcher (and also to Wright) [5,16]. To formally state it, we provide the necessary key notion: For $t \in V^{\mathtt{t}}$, we define the *yield-operator* $\mathcal{Y}$ as follows:

$$\mathcal{Y}(t) = \begin{cases} t(\varepsilon), & \text{if } t(\varepsilon) \in V_0 \\ \mathcal{Y}(t/1) \cdots \mathcal{Y}(t/k), & \text{if } t(\varepsilon) \in V_k, k > 0 \end{cases}$$

In words, the recursion means that the yield of a tree with a root with k children equals the concatenation of the yields of the trees whose roots are these children. The operator naturally extends to tree languages and tree language families.

Theorem 2 (Doner, Thatcher, Wright). *A string language is context-free if and only if it is the yield of a regular tree language.*

Notice that the proof of this result makes use of the fact that for context-free languages, we can assume that they are generated by some context-free grammar without erasing productions, neglecting the possibility to describe the empty word itself. As it is still an open problem whether or not we can get rid of erasing rules with regulated grammars as introduced in the previous subsection, we restricted our attention to regulated grammars without erasing rules there, as we strive for analogues of Theorem 2 in the following.

2.3 Regulated Tree Automata

We are now defining the central new notion of this paper, combining the two classical worlds so far introduced. Hence, a *graph-controlled finite tree automaton* is an 8-tuple $A = (Q, V, \delta, F, \Gamma, \Sigma, \Phi, h)$ where

- $A' = (Q, V, \delta, F)$ define a finite tree automaton;
- Γ, Σ, Φ define the graph structure as in graph-controlled grammars;
- $h : U \to (2^\delta \setminus \{\emptyset\})$ relates vertices with rule sets; notice that we consider δ as a set of rules here.

We say that $(s, u) \models (t, v)$ holds via A with $(s, u), (t, v) \in (Q \cup V)^{\mathtt{t}} \times U$ if

$$s \vdash_{h(u)} t \quad \text{and} \quad (u, v) \in E\,.$$

The reflexive transitive closure of $\models$ is denoted by $\models^*$. The tree language accepted by A (assuming that A' works bottom-up) is defined by

$$L^{bu}(A) = \{t \in V^{\mathtt{t}} \mid \exists q \in F\, \exists u \in \Sigma\, \exists v \in \Phi((t, u) \models^* (q, v))\}\,.$$

Similarly, we can define acceptance for top-down automata, yielding the language $L^{td}(A)$. This gives the tree language families $\mathcal{L}^{\mathtt{t}}(\mathrm{G}, bu)$ and $\mathcal{L}^{\mathtt{t}}(\mathrm{G}, td)$, depending on whether bottom-up or top-down automata are considered. If we want to explicitly rule out ε-moves, we add $-\varepsilon$ to our notations. As the notions of regular control, matrix and programmed have been introduced in Subsection 2.1 as simple syntactical restrictions of graph control, we can carry over them immediately to regulated finite tree automata, giving, e.g., the notion of a matrix finite tree automaton. This also gives language families such as $\mathcal{L}^{\mathtt{t}}(\mathrm{P}, td, -\varepsilon)$.

3 Basic Results for Regulated Tree Automata

By the definitions themselves, we can conclude (also confer [11], but mind the partially different definitions):

Lemma 3. *Let $\mu \in \{bu, td\}$. Then, we have*

$$\mathcal{L}^{\mathtt{t}}(\mathrm{P}, \mu) \subseteq \mathcal{L}^{\mathtt{t}}(\mathrm{rC}, \mu) \subseteq \mathcal{L}^{\mathtt{t}}(\mathrm{G}, \mu) \quad \textit{and} \quad \mathcal{L}^{\mathtt{t}}(\mathrm{M}, \mu) \subseteq \mathcal{L}^{\mathtt{t}}(\mathrm{rC}, \mu)\,.$$

Proposition 4. *For $C \in \{\mathrm{G}, \mathrm{P}, \mathrm{rC}, \mathrm{M}\}$, $\mathcal{L}^{\mathtt{t}}(C, bu) = \mathcal{L}^{\mathtt{t}}(C, td)$.*

Proof. Recall [2] that for any finite top-down tree automaton A_{td}, one can construct an equivalent finite bottom-up automaton A_{bu} by simply reversing the relation $\vdash$, plus exchanging initial and final states. Similarly, we can simulate $A = (A_{td}, \Gamma, \Sigma, \Phi, h)$ by $A' = (A_{bu}, \Gamma', \Sigma', \Phi', h')$, where $\Gamma' = (U', E')$ is obtained from $\Gamma = (U, E)$ by reversing the arcs, $\Sigma' = \Phi$, $\Phi' = \Sigma$, and h' associates the reversed rule variants of $h(u)$ to $u \in U' = U$. Clearly, if $(\Gamma, \Sigma, \Phi, h)$ satisfies the restrictions imposed by $C \in \{\mathrm{G}, \mathrm{P}, \mathrm{rC}, \mathrm{M}\}$, then $(\Gamma', \Sigma', \Phi', h')$ does so, as well. The converse inclusion is similarly seen. □

Hence, we can from now on consider either the bottom-up or the top-down case, whatever is more convenient to us. We are going to present a sequence of technical lemmas that combine classical ideas from tree automata and from regulated rewriting. Illustrations by examples can be found in the Appendix.

Lemma 5. $\mathcal{L}^{\mathtt{t}}(\mathrm{G}, bu) \subseteq \mathcal{L}^{\mathtt{t}}(\mathrm{rC}, bu)$.

Proof. We only sketch the construction. Consider a graph-controlled finite tree automaton is an 8-tuple $A = (Q, V, \delta, F, \Gamma, \Sigma, \Phi, h)$. We derive an equivalent finite tree automaton with regular control $A_r = (Q, V, \delta, F, \Gamma_r, \Sigma_r, \Phi_r, h_r)$ as follows. Let $\Gamma = (U, E)$. Then, $\Gamma_r = (U_r, E_r)$ with $U_r = \bigcup_{u \in U} \{u\} \times h(u)$, $E_r = \{((u,x),(v,y)) \mid (u,x),(v,y) \in U_r, (u,v) \in E\}$, $\Sigma_r = \bigcup_{u \in \Sigma} \{u\} \times h(u)$, $\Phi_r = \bigcup_{u \in \Phi} \{u\} \times h(u)$, and $h_r((u,x)) = \{x\}$ for $(u,x) \in U_r$. By construction, $|h_r((u,x))| = 1$ for all $(u,x) \in U_r$. Moreover, if $(s,u) \models (t,v)$ holds via A, then $s \vdash_{h(u)} t$ and $(u,v) \in E$, so that for some $x \in h(u)$, $s \vdash_{\{x\}} t$, i.e., $(s,(u,x)) \models (t,(v,y))$ holds via A_r for all $y \in h(v)$. Induction shows the claim. □

Lemma 6. $\mathcal{L}^{\mathtt{t}}(\mathrm{rC}, bu) \subseteq \mathcal{L}^{\mathtt{t}}(\mathrm{P}, bu)$.

Proof. We are modifying bottom-up tree automata with regular control step by step in order to obtain an equivalent programmed control. (i) We can assume that initial vertices (from Σ) have no in-going arcs and that there is only one final vertex (i.e., $|\Phi| = 1$) that has no out-going arcs. This can be easily seen by keeping in mind the relation to regular languages and hence to finite string automata (as control devices) as recalled in Remark 1. (ii) Moreover, by using a shadow state alphabet Q', the finite tree automaton itself can check if the derivation control had started in some $u_i \in \Sigma$ and also that the corresponding rule was used only once. Namely, the starting rules (that have to process a terminal symbol at some leaf node of the tree) will lead to a primed state q' (when it would go to q in the original automaton), and the fact that exactly one primed state was ever entered is then propagated to the root of the tree. Here, it is also necessary to split vertices of the control graph. More specifically, if vertex v contains a rule $f(q_1, \ldots, q_k) \to q$, we create k many twins of v, say, $v_1, \ldots, v_k$, and then v_i contains the rule $f(q_1, \ldots, q_{i-1}, q_i', q_{i+1}, \ldots, q_k) \to q'$ to properly propagate the prime information. This already shows that we can now let any vertex be initial in our control graph without changing the set of accepted trees. (iii) As a further step, we can introduce another shadow state alphabet $\hat{F}$ as a new set of final states and modify the rules associated to the control graph vertices so that (only and exactly) when moving to u_f, with $\{u_f\} = \Phi$, such a final state $\hat{q}$ is entered. This step might involve splitting vertices v that are predecessors of u_f into v and $\hat{v}$, where $\hat{v}$ contains the rule introducing $\hat{q}$, while v contains the rule introducing the corresponding state q. Otherwise, v and $\hat{v}$ have the same predecessors. However, u_f is the only successor of $\hat{v}$ (and not a successor of v), while all other successor vertices that previously existed for v are still successor vertices of v (and not for $\hat{v}$). Now, we can (formally) let every vertex be a final vertex without changing the accepted tree language. Again, the correctness of the construction is seen by induction. □

Remark 7. For all statements made so far in this section, similar results hold when disallowing ε-moves. We refrain from making this explicit. However, this is no longer obvious for the following construction. This also gives a first **open question** in the area of regulated tree automata.

Lemma 8. $\mathcal{L}^{\mathtt{t}}(\mathrm{P}, bu) \subseteq \mathcal{L}^{\mathtt{t}}(\mathrm{M}, bu)$.

Recall that in the classical construction simulating programmed grammars by matrix grammars, the state information is maintained in a special nonterminal. We follow the same idea here, but with tree automata, this is technically more involved due to the absence of erasing rules.

Proof. Consider a programmed automaton $A = (Q, V, \delta, F, \Gamma, \Sigma, \Phi, h)$ with $\Gamma = (U, E)$ and $\Sigma = \Phi = U$. Now, we can also assume that the rules associated to start vertices are of the form $a \to q$ for some terminal $a \in V_0$ and some state q, this way (formally) specifying $\hat{\Sigma} \subseteq \Sigma$. (a) We add rules by $r = a \to [q, u]$ and introduce a new vertex $\hat{v}$, with $h(\hat{v}) = \{r\}$ and $v \in \Sigma$, for those u that are successors of v in Γ. More formally, this means that we introduce for each $v \in \hat{\Sigma}$ as many twins as there are successors of v in Γ. This also defines a new set of initial vertices Σ', with more vertices to be added. All these vertices $\hat{v}$ have outgoing arcs to all vertices from Σ'. (b) For each $q \in Q \cup V_0$ and each $u, v \in U \setminus \Sigma$ with $(u, v) \in E$, we introduce a new rule $[q, u] \to [q, v]$ and a new vertex $[q, u, v]$ into Γ' hosting this new rule. All these vertices also belong to Σ'. (c) Introduce an arc from each vertex $[q, u, v]$ to the vertex u containing the rule $h(u)$. All such vertices u have arcs to all vertices from Σ'. (d) For each $q \in Q \cup V_0$ and each $u, v \in U \setminus \Sigma$ with $(u, v) \in E$ and each $1 \leq i \leq k$ with $h(u) = \{f(p_1, \ldots, p_k) \to p\}$ such that $p_i = q$, we introduce a new rule $f(p_1, \ldots, p_{i-1}, [p_i, u], p_{i+1}, \ldots, p_k) \to [p, v]$; moreover, we create a vertex $[q, i, u, v]$ containing exactly this newly created rule, put it into Σ' and link it to all vertices from Σ'. (e) For each $q \in Q \cup V_0$ and each $u, v \in U \setminus \Sigma$ with $(u, v) \in E$, if $h(u) = \{q \to p\}$, i.e., u hosts an ε-move, then we introduce the new rule $[q, u] \to [p, v]$ and call the vertex hosting this rule $[q, 1, u, v]$ for simplicity. Again, such vertices are put into Σ' and linked to all vertices from Σ'. To summarize, we described a new graph-controlled automaton $A' = (Q', V', \delta', F', \Gamma', \Sigma', \Phi', h')$ with $\Gamma' = (U', E')$, where $Q' \supseteq Q$, $V' \supseteq V$, $\delta' \supseteq \delta$, Σ', $U' \supseteq U$ and E' as specified above. As final state set F', we take $F' = F \times U$. The final vertices (from Φ') contain U and all vertices from Σ' introduced in steps (a), (d) and (e). Clearly, A' is with matrix control. Again, the correctness of the construction is seen by induction. □

We can summarize our results as follows.

Theorem 9. *Let $\mu_M, \mu_P, \mu_{rC}, \mu_G \in \{bu, td\}$. Then, we have*

$$\mathcal{L}^{\mathtt{t}}(\mathrm{M}, \mu_M) = \mathcal{L}^{\mathtt{t}}(\mathrm{P}, \mu_P) = \mathcal{L}^{\mathtt{t}}(\mathrm{rC}, \mu_{rC}) = \mathcal{L}^{\mathtt{t}}(\mathrm{G}, \mu_G) \quad \textit{and}$$

$$\mathcal{L}^{\mathtt{t}}(\mathrm{M}, \mu_M, -\varepsilon) \subseteq \mathcal{L}^{\mathtt{t}}(\mathrm{P}, \mu_P, -\varepsilon) = \mathcal{L}^{\mathtt{t}}(\mathrm{rC}, \mu_{rC}, -\varepsilon) = \mathcal{L}^{\mathtt{t}}(\mathrm{G}, \mu_G, -\varepsilon)\,.$$

This result gives rise to the following natural second **open question**: Is the trivial inclusion $\mathcal{L}^{\mathtt{t}}(\mathrm{G}, td, -\varepsilon) \subseteq \mathcal{L}^{\mathtt{t}}(\mathrm{G}, td)$ strict or not? Recall that for classical finite tree automata, we can dispose of ε-moves as a normal form.

4 Relation to String Languages

We already introduced the yield operator above that allows us to associate strings to trees and hence string languages to tree languages. Recall Theorem 2.

Lemma 10. $\mathcal{Y}(\mathcal{L}^{\mathrm{t}}(\mathrm{G}, bu)) \subseteq \mathcal{L}^{acc}(\mathrm{G}, \mathrm{CF} - \varepsilon)$.

Proof. Consider a graph-controlled finite tree automaton $A = (Q, V, \delta, F, \Gamma, \Sigma, \Phi, h)$ with $\Gamma = (U, E)$. We are going to construct a graph-controlled context-free grammar $G_A = (V_N, V_T, P, q_f, \Gamma, \Sigma, \Phi, h')$ such that $\mathcal{Y}(L^{bu}(A)) = L^{acc}(G_A)$. As it is usually the case with nondeterministic automata, we can assume (without loss of generality) that A only one final (accepting) state, i.e., $F = \{q_f\}$. We construct a simulating accepting grammar G_A as follows: $N = Q$, $T = V_0$, $w \to q \in h'(u)$ if $q \in Q$, $w = w_1 \cdots w_k$, $w_j \in V_0 \cup Q$ whenever $g(w_1, \ldots, w_k) \to q \in h(u)$ for some $g \in V_k$ (*).

Now, each derivation of A producing a certain yield can be simulated by G_A, where the correct labels of inner nodes are guessed during the derivation due to (*). Conversely, these guesses according to (*) label the inner nodes of a derivation tree in a way corresponding to a tree that can be accepted by A.

A formal reasoning would be a relatively tedious exercise, based on the ideas originating from Doner, Thatcher and Wright in the late sixties, which can be also found in any textbook on tree languages. Therefore, we only sketch the basic idea of the inductive step of the proof in the following. Recall that the definition of $\vdash$ transforms trees with leaf labels from $(V_0 \cup Q)$ into trees with leaf labels from $(V_0 \cup Q)$; extending the definition of the yield operator $\mathcal{Y}$ accordingly, this means that sentential forms of G_A are transformed. Notice that the graph control stays the same, which allows the induction to succeed. □

Literally the same construction allows us to state:

Lemma 11. $\mathcal{Y}(\mathcal{L}^{\mathrm{t}}(\mathrm{G}, td)) \subseteq \mathcal{L}^{gen}(\mathrm{G}, \mathrm{CF} - \varepsilon)$.

For the converse direction, we need a normal form result for regulated context-free grammars that might be of independent interest. A graph-controlled context-free grammar $G = (V_N, V_T, P, S, \Gamma, \Sigma, \Phi, h)$ is called *arity-deterministic* if for each nonterminal $A \in N$, there exists a unique number $\alpha(A)$ (called the *arity* of A) such that any rule $A \to w \in P$ (in the generating case) or $w \to A \in P$ (in the accepting case) obeys $|w| = \alpha(A)$.

Theorem 12 (Arity-deterministic normal form). *For any $L \in \mathcal{L}^{acc}(\mathrm{G}, \mathrm{CF}-\varepsilon)$, there exists an arity-deterministic context-free ε-free graph-controlled context-free grammar G accepting L. A similar statement holds for generating grammars.*

As the induction proof given in [8, Theorem 2] can be easily adapted to our case, we omit it here. We only mention that similar results are also true for other forms of control, like matrix grammars.

Lemma 13. $\mathcal{Y}(\mathcal{L}^{\mathrm{t}}(\mathrm{G}, bu)) \supseteq \mathcal{L}^{acc}(\mathrm{G}, \mathrm{CF} - \varepsilon)$.

Proof. [Sketch] Starting with an arity-deterministic graph-controlled context-free grammar G, we can easily interpret its rules as transitions of a controlled finite tree automaton A_G. More specifically, we can consider $V_N \cup V_T$ as a ranked alphabet V, with $V_0 = V_T$. For any rule $w \to A$, we introduce a rule

$\delta_{i,k}(A, w_1, \ldots, w_k) = \{A'\}$, where $|w| = \alpha(A) = k$. Notice that we have to formally distinguish the nonterminal A of arity k from the state A' that has, in a sense, arity zero. A_G accepts derivation trees of G. Conversely, the yield of any tree that derives S' (in A_G) corresponds to a sentential form that derives S (in G). For further details, we refer to [8, Lemma 2]. □

Lemma 14. $\mathcal{Y}(\mathcal{L}^{\mathtt{t}}(\mathrm{G}, td)) \supseteq \mathcal{L}^{gen}(\mathrm{G}, \mathrm{CF} - \varepsilon)$.

Together with the results from the previous section, we conclude a known fact:

Theorem 15. *Let $\mu_M, \mu_P, \mu_{rC}, \mu_G \in \{gen, acc\}$. Then, we have*

$$\mathcal{L}^{\mu_M}(\mathrm{M}, \mathrm{CF} - \varepsilon) = \mathcal{L}^{\mu_P}(\mathrm{P}, \mathrm{CF} - \varepsilon) = \mathcal{L}^{\mu_{rC}}(\mathrm{rC}, \mathrm{CF} - \varepsilon) = \mathcal{L}^{\mu_G}(\mathrm{G}, \mathrm{CF} - \varepsilon)\,.$$

Remark 16. As the constructions presented in this section do not introduce chain rules (into context-free grammars) and as chain rules correspond to ε-moves for tree automata, the open questions formulated in the previous section easily translate into **open questions** in the more classical realm of regulated context-free grammars as follows: *Does there exist a normal form for regulated context-free grammars (without erasing productions) that allows us to avoid chain-rules?* Related to this is another **open question** in the more classical realm of regulated context-free grammars: *Does there exist a Chomsky normal form result?* Loosely speaking, this corresponds to an arity-bounded normal form for derivation trees.

5 Adding Appearance Checks

Appearance checks (or maybe better said applicability checks, see [11]) are one of the key features introduced within regulated rewriting. On the level of graph control, this corresponds to considering bicolored digraphs as control structure [6,18]. For reasons of space, we refrain from giving a formal definition in this extended abstract. Notice that there are two ways of interpreting appearance checks in connection with control by bicolored digraphs, with the choice interpretation that first selects a rule in the rule set $h(u)$ and then checks for applicability of the selected tree rewriting rule, or with the interpretation that only considers $h(u)$ to be not applicable if none of the rules in $h(u)$ is applicable. We signal the choice interpretation by adding a c as a subscript. We can prove:

Theorem 17. *Let $\mu \in \{bu, td\}$. Then, we have*

$$\mathcal{L}^{\mathtt{t}}(\mathrm{M}, \mu, \mathrm{ac}) = \mathcal{L}^{\mathtt{t}}(\mathrm{P}, \mu, \mathrm{ac}) = \mathcal{L}^{\mathtt{t}}(\mathrm{rC}, \mu, \mathrm{ac}) = \mathcal{L}^{\mathtt{t}}(\mathrm{G}, \mu, \mathrm{ac}) = \mathcal{L}^{\mathtt{t}}(\mathrm{G}_c, \mu, \mathrm{ac}).$$

We could state similar results as in Theorem 9 for the case when disallowing ε-moves. We have to distinguish more carefully between the bottom-up and the top-down cases due to the following results.

Theorem 18. *Let $\mu \in \{gen, acc\}$. Then, we have $\mathcal{L}^{\mu}(\mathrm{M}, \mathrm{CF}{-}\varepsilon, \mathrm{ac}) = \mathcal{L}^{\mu}(\mathrm{P}, \mathrm{CF}{-}\varepsilon, \mathrm{ac}) = \mathcal{L}^{\mu}(\mathrm{rC}, \mathrm{CF} - \varepsilon, \mathrm{ac}) = \mathcal{L}^{\mu}(\mathrm{G}, \mathrm{CF} - \varepsilon, \mathrm{ac}) = \mathcal{L}^{\mu}(\mathrm{G}_c, \mathrm{CF} - \varepsilon, \mathrm{ac})$. Moreover, $\mathcal{L}^{gen}(\mathrm{M}, \mathrm{CF}{-}\varepsilon, \mathrm{ac}) = \mathcal{Y}(\mathcal{L}^{\mathtt{t}}(\mathrm{M}, td, \mathrm{ac})) \subsetneq \mathcal{L}^{acc}(\mathrm{M}, \mathrm{CF}{-}\varepsilon, \mathrm{ac}) = \mathcal{Y}(\mathcal{L}^{\mathtt{t}}(\mathrm{M}, bu, \mathrm{ac}))$.*

Corollary 19. $\mathcal{L}^{\mathrm{t}}(\mathrm{M}, td, \mathrm{ac}) \subsetneq \mathcal{L}^{\mathrm{t}}(\mathrm{M}, bu, \mathrm{ac})$.

Proof. The inclusion itself is seen as before; notice that we can simulate a rule $q \to f(q_1, \ldots, q_k)$ that is applied in appearance checking by some ε-move that checks for the presence of q. If the converse inclusion would hold, as well, then the yields of both tree language families would coincide, which contradicts known facts on regulated rewriting, see [1]. □

6 Conclusions

We started investigations on regulated tree automata in this paper. This new way of looking at trees and regulated rewriting opens up quite an ample ground of research. Apart from the concrete open problems mentioned throughout the paper, which mostly also extend to the case admitting appearance checks, we ask the following, more concrete research questions.

- So far, we completely neglected studying closure properties or algorithmic questions of (variants of) regulated tree automata. It seems to be the case that the standard constructions for showing certain closure properties of regular tree languages (see [2]) transfer to the regulated case, but, moreover, we conjecture positive closure results for (general) tree homomorphisms, to give one concrete **open question** in this area.
- There are many relations between regulated rewriting and parallel rewriting; see [4,7]. We are not aware of a theory of parallel tree automata. We would also expect (again) relations to the question of accepting versus generating grammars [9]. Also, the area of grammar systems is barely touched [3,10,8]. Due to the connections between regulated rewriting and cooperating distributed grammar systems (CDGS), see [3], these investigations might also stir some new interest in and even give some new proof ideas for some old open problems. For instance, it is still open whether (context-free) matrix languages can be characterized by CDGS working in $= k$-mode. Can results from tree automata be helpful here? We refer to [8] for results on cooperating distributed tree automata.
- Operations on trees have been one of the cornerstones for developing practically useful mechanisms [13] for formalizing *mild context-sensitivity*. The relations between, for instance, tree adjoining languages and variants of regulated context-free grammars have been largely unexplored until today. Apart from this concrete question, we have the hope that combining tree processing with regulated rewriting mechanisms opens up new (practical) applications of regulated rewriting, also leading to new algorithmic questions.

Finally, we like to mention once more the various open problems in the area of classical regulated rewriting scattered throughout the paper. This should renew the interest of the Formal Language community and also shows that (by far) not all problems in that area are solved. We hope that the approach via considering tree languages might be a way to solve some of these problems. Common to all

these questions is the quest for normal forms. Here, we like to point to random context grammars, where it was shown rather recently that there is a normal form result that gets rid of erasing productions [19].

References

1. Bordihn, H., Fernau, H.: Accepting grammars with regulation. International Journal of Computer Mathematics 53, 1–18 (1994)
2. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Löding, C., Tison, S., Tomassi, M.: Tree Automata, Techniques and Applications (2007), `http://tata.gforge.inria.fr/`
3. Csuhaj-Varjú, E., Dassow, J., Kelemen, J., Păun, G.: Grammar Systems: A Grammatical Approach to Distribution and Cooperation. London: Gordon and Breach (1994)
4. Dassow, J., Păun, Gh.: Regulated Rewriting in Formal Language Theory, EATCS Monographs in Theoretical Computer Science, vol. 18. Springer (1989)
5. Doner, J.: Tree acceptors and some of their applications. Journal of Computer and System Sciences 4(5), 406–451 (1970)
6. Fernau, H.: Graph-controlled grammars as language acceptors. Journal of Automata, Languages and Combinatorics 2(2), 79–91 (1997)
7. Fernau, H.: Parallel grammars: A phenomenology. GRAMMARS 6, 25–87 (2003)
8. Fernau, H.: Cooperating distributed tree automata. In: Bordihn, H., Kutrib, M., Truthe, B. (eds.) Languages Alive; Dassow Festschrift, LNCS, vol. 7300, pp. 75–85. Springer (2012)
9. Fernau, H., Bordihn, H.: Remarks on accepting parallel systems. International Journal of Computer Mathematics 56, 51–67 (1995)
10. Fernau, H., Holzer, M., Bordihn, H.: Accepting multi-agent systems: the case of cooperating distributed grammar systems. Computers and Artificial Intelligence 15(2-3), 123–139 (1996)
11. Freund, R., Kogler, M., Oswald, M.: A general framework for regulated rewriting based on the applicability of rules. In: Kelemen, J., Kelemenová, A. (eds.) Computation, Cooperation, and Life, LNCS, vol. 6610, pp. 35–53. Springer (2011)
12. Gécseg, F., Steinby, M.: Tree Automata. Akadémiai Kiadó (1984)
13. Kallmeyer, L.: Parsing Beyond Context-Free Grammars. Springer (2010)
14. Meduna, A., Kolar, D.: Regulated pushdown automata. Acta Cybernetica 14, 653–664 (2000)
15. Meduna, A., Zemek, P.: Regulated Grammars and Automata. Springer Publishing Company (2014)
16. Thatcher, J.W.: Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. Journal of Computer and System Sciences 1, 317–322 (1967)
17. Vu, M.: Regulierte Grammatiken und regulierte Baumautomaten. Bachelorarbeit, Informatikwissenschaften, Universität Trier, Germany (2016)
18. Wood, D.: Bicolored digraph grammar systems. RAIRO Informatique théorique et Applications/Theoretical Informatics and Applications 1, 45–50 (1973)
19. Zetzsche, G.: On erasing productions in random context grammars. In: Abramsky, S., Gavoille, C., Kirchner, C., auf der Heide, F.M., Spirakis, P.G. (eds.) Automata, Languages and Programming, ICALP 2010, Part II. LNCS, vol. 6199, pp. 175–186. Springer (2010)