ELSEVIER

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Theoretical Computer Science 337 (2005) 183–216

Theoretical Computer Science

www.elsevier.com/locate/tcs

On state-alternating context-free grammars☆

Etsuro Moriya[a,*], Dieter Hofbauer[b], Maria Huber[b], Friedrich Otto[b]

[a] *Department of Mathematics, School of Education, Waseda University, 1-6-1, Nishi-Waseda, Shinjuku-ku, Tokyo 169-8050, Japan*

[b] *Fachbereich Mathematik/Informatik, Universität Kassel, 34109 Kassel, Germany*

Received 22 February 2004; received in revised form 15 September 2004; accepted 30 December 2004

Communicated by M. Ito

Abstract

State-alternating context-free grammars are introduced, and the language classes obtained from them are compared to the classes of the Chomsky hierarchy as well as to some well-known complexity classes. In particular, state-alternating context-free grammars are compared to alternating context-free grammars (Theoret. Comput. Sci. 67 (1989) 75–85) and to alternating pushdown automata. Further, various derivation strategies are considered, and their influence on the expressive power of (state-)alternating context-free grammars is investigated.

© 2005 Elsevier B.V. All rights reserved.

MSC: 68Q45; 68Q10

Keywords: Alternating context-free grammar; Context-free grammar with states; State-alternating context-free grammar; Alternating pushdown automaton

1. Introduction

Alternation is a powerful generalization of non-determinism that has led to many interesting results in automata and complexity theory. It was first introduced by Chandra and

☆ The main part of this work was done while Etsuro Moriya was visiting the Fachbereich Mathematik/Informatik, Universität Kassel.

* Corresponding author.

E-mail addresses: moriya@waseda.jp (E. Moriya), dieter@theory.informatik.uni-kassel.de (D. Hofbauer), maria@theory.informatik.uni-kassel.de (M. Huber), otto@theory.informatik.uni-kassel.de (F. Otto).

0304-3975/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2004.12.029

Stockmeyer [2,3] for general Turing machines and by Ladner et al. [10,11] for pushdown automata. By alternation the well-known deterministic hierarchy

$$\mathsf{LOGSPACE} \subseteq \mathsf{P} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXPTIME} \subseteq \mathsf{EXPSPACE} \subseteq \cdots$$

shifts by exactly one level, as according to [2,3],

$$\begin{aligned}&\mathsf{ALOGSPACE} = \mathsf{P},\\ &\mathsf{APTIME} = \mathsf{PSPACE},\\ &\mathsf{APSPACE} = \mathsf{EXPTIME},\\ &\mathsf{AEXPTIME} = \mathsf{EXPSPACE}.\end{aligned}$$

Further, the class $\mathsf{ALINSPACE} := \mathcal{L}(\mathsf{ALBA})$ of languages that are accepted by alternating Turing machines within linear space, that is, by the so-called *alternating linear bounded automata* ALBA, coincides with the class $\mathcal{L}(\mathsf{APDA})$ of languages that are accepted by *alternating pushdown automata* (APDA, for short). This class in turn coincides with the deterministic time complexity class $\mathsf{ETIME} := \bigcup_{c>0} \mathsf{DTIME}(c^n)$ [2,11]. This result holds in fact for alternating pushdown automata with one-way input as well as with two-way input.

(Non-alternating) pushdown automata accept exactly the context-free languages, while non-deterministic Turing machines with linear space bounds accept exactly the context-sensitive languages. Hence, one would like to also obtain a grammatical characterization for the class of languages $\mathcal{L}(\mathsf{APDA})$ that are accepted by alternating pushdown automata. This question was first addressed by Moriya in [14] by considering alternating context-free grammars.

Definition 1.1. *An alternating context-free grammar* is given through a 5-tuple $G = (V, U, \Sigma, P, S)$, where V is a set of variables (or non-terminals), $U \subseteq V$ is a set of *universal* variables, Σ is a set of terminals, S is the start symbol, and P is a set of context-free productions. The variables in $V \setminus U$ are called *existential variables*.

The *derivation relation* $\Rightarrow_G$ that is induced by G on the set of *sentential forms* $(V \cup \Sigma)^*$ is defined as follows. Let $\alpha, \beta \in (V \cup \Sigma)^*$, and let $A \in V$. If A is an existential variable, and $(A \to \gamma) \in P$, then $\alpha A \beta \Rightarrow_G \alpha\gamma\beta$. If, however, A is a universal variable, and $(A \to \gamma_i)$ $(1 \leqslant i \leqslant k)$ are all the productions from P with left-hand side A, then $\alpha A \beta \Rightarrow_G (\alpha\gamma_1\beta, \ldots, \alpha\gamma_k\beta)$, that is, all productions with left-hand side A are applied simultaneously. In this way a derivation is not a linear chain, but it has the form of a tree. A terminal word w can be derived from G, if there exists a finite derivation tree in this sense such that the root is labelled with the start symbol S and all leaves are labelled with the string w. As usual $L(G)$ denotes the set of terminal words derived from G.

In the following we will denote the class of alternating context-free grammars by ACFG, and $\mathcal{L}(\mathsf{ACFG})$ will denote the class of languages that are generated by these grammars. Further, $\mathcal{L}(\varepsilon\text{-}\mathsf{free}\text{-}\mathsf{ACFG})$ will denote the class of languages that are generated by alternating context-free grammars without ε-rules. We use $\mathcal{L}_{\mathsf{lm}}(\mathsf{ACFG})$ and $\mathcal{L}_{\mathsf{lm}}(\varepsilon\text{-}\mathsf{free}\text{-}\mathsf{ACFG})$ to denote the classes of languages that are generated by these grammars using the *leftmost derivation strategy*, which requires that in each step of a derivation the leftmost variable of the current sentential form must be rewritten. Finally, by lin-ACFG we denote the class of *linear* alternating context-free grammars, that is, those alternating context-free grammars for which each rule contains at most one variable occurrence in its right-hand side.

In [14], it is claimed that a language is accepted by an alternating pushdown automaton if and only if it can be generated by an alternating context-free grammar, but unfortunately the arguments given in that paper contain some serious flaws that have not been overcome to this day. One of the problems stems from the fact that in an alternating context-free grammar the derivation strategy chosen makes a difference in contrast to the situation for context-free grammars. In particular, for an alternating context-free grammar, the set of words generated by *leftmost* derivations is in general a proper subset of the set of all words that can be generated by that grammar.

Nevertheless, some interesting partial results have been obtained. First, Chen and Toda [4] presented complexity theoretical characterizations of the language classes $\mathcal{L}_{\mathsf{lm}}(\mathsf{lin\text{-}ACFG})$ and $\mathcal{L}_{\mathsf{lm}}(\varepsilon\text{-}\mathsf{free\text{-}ACFG})$ by showing that

$$\mathsf{P} = \mathsf{LOG}(\mathcal{L}_{\mathsf{lm}}(\mathsf{lin\text{-}ACFG})) \text{ and } \mathsf{PSPACE} = \mathsf{LOG}(\mathcal{L}_{\mathsf{lm}}(\varepsilon\text{-}\mathsf{free\text{-}ACFG})),$$

where $\mathsf{LOG}(\mathcal{L})$ denotes the closure of the language class $\mathcal{L}$ under log-space reductions. For a linear grammar each derivation is necessarily leftmost, and so the first result above can be restated as $\mathsf{P} = \mathsf{LOG}(\mathcal{L}(\mathsf{lin\text{-}ACFG}))$. Then Ibarra, Jiang, and Wang gave a grammatical characterization for $\mathcal{L}(\mathsf{APDA})$ in [8] by showing that $\mathcal{L}(\mathsf{APDA}) = \mathcal{L}(\mathsf{linear\text{-}erasing\text{-}ACFG})$, where an alternating context-free grammar G is said to be *linear erasing* if there is a constant c such that every string of length n in the language generated by G has a derivation tree containing only sentential forms of length at most $c \cdot n$. However, Ibarra, Jiang, and Wang require in addition that the grammar introduces *endmarkers* for the terminal strings generated, that is, the language they consider consists of all terminal strings w such that the string $\$w\$$ is generated by the grammar. While the inclusion of $\mathcal{L}(\mathsf{linear\text{-}erasing\text{-}ACFG})$ in $\mathcal{L}(\mathsf{APDA})$ remains valid even without these endmarkers, it is not clear whether the converse inclusion does, as the simulation of an alternating linear-bounded automaton by a linear-erasing alternating context-free grammar given in [8] crucially depends on the use of these endmarkers.

Here we will consider a new variant of alternating grammars, a variant that is obtained by combining the notion of *grammars with states* with the notion of alternation. Context-free grammars with states, abbreviated as ECFG, were introduced by Kasai in [9]. We will shortly discuss these grammars and the language classes they generate in Section 2. Then we will define the so-called *state-alternating context-free grammars*, abbreviated as sACFG, which are obtained from the grammars with states by distinguishing between universal and existential states. We will see that these grammars can be interpreted as a generalization of both the ECFGs and the ACFGs.

In Section 3, we will derive a lower bound for the expressive power of ACFGs by presenting an example of a language that is generated by an ACFG in leftmost mode as well as in unrestricted mode, but that cannot be written as the intersection of finitely many context-free languages. As a consequence, we obtain that $\mathcal{L}_{\mathsf{lm}}(\mathsf{ACFG}) \cap \mathcal{L}(\mathsf{ACFG})$ properly includes the class of languages that are intersections of finitely many context-free languages. In Section 4, we will compare sACFGs to ACFGs. In particular, we will obtain the basic fact that sACFGs are at least as powerful as ACFGs in their generative capacity. In Section 5, we will consider various restricted versions of sACFGs and analyze the complexity of the language classes generated by leftmost derivations. For example we will show that in leftmost mode, ε-free sACFGs only generate deterministic context-sensitive languages. Then, in Section 6, we will derive a grammatical characterization for the language class $\mathcal{L}(\mathsf{APDA})$ in terms of sACFGs by showing that $\mathcal{L}(\mathsf{APDA})$ coincides with $\mathcal{L}_{\mathsf{lm}}(\mathsf{sACFG})$. In Section 7, we will compare the classes of languages to each other that are generated by sACFGs using different derivation modes. Finally, in Section 8, we will address the problem of deciding for a given (s)ACFG G and a fixed derivation mode m, whether the language $L_{\mathsf{m}}(G)$ coincides with the language $L(G)$. The paper closes with a discussion of the results obtained and some open problems. In appendix, we include a diagram depicting the known inclusion relations among the major language classes discussed in the paper and some well-known language and complexity classes.

2. State-alternating context-free grammars

Throughout this paper we will make use of the following notational convention. For a grammar G of any type, we denote by $L(G)$ the language generated by that grammar, and for a class of grammars C we denote by $\mathcal{L}(\mathsf{C})$ the family of all languages that are generated by grammars from that class. For any derivation mode m, $L_{\mathsf{m}}(G)$ and $\mathcal{L}_{\mathsf{m}}(\mathsf{C})$ denote the language and the family of languages, respectively, that are generated by only using the derivation mode m. Here we will encounter the *leftmost* mode, denoted by lm, the *leftish* mode, denoted by lt, and the *rightmost* mode, denoted by rm. Further, by using the prefix ε-free- we indicate that only grammars without ε-rules are considered. Analogously, for any automaton A, $L(A)$ is the language accepted by A, and for a class of automata C, $\mathcal{L}(\mathsf{C})$ is the class of languages that are accepted by automata from that class. Further, for reasons of simplicity we will mainly consider only languages that do not contain the empty word ε.

As mentioned before, context-free grammars with states, abbreviated as ECFG, were introduced by Kasai [9].

Definition 2.1. An ECFG is given through a 6-tuple $G = (Q, V, \Sigma, P, S, q_0)$, where Q is a finite set of *states*, V is a finite set of variables, Σ is a finite set of terminals, $S \in V$ is the start symbol, $q_0 \in Q$ is the initial state, and P is a finite set of productions of the form $(p, A) \to (q, \alpha)$, where $p, q \in Q$, $A \in V$, and $\alpha \in (V \cup \Sigma)^*$.

The *derivation relation* induced by G is defined through $(p, \beta A\gamma) \Rightarrow (q, \beta\alpha\gamma)$ for all $\beta, \gamma \in (V \cup \Sigma)^*$ and $((p, A) \to (q, \alpha)) \in P$. The language generated by G is the set

$$L(G) := \{ w \in \Sigma^* \mid (q_0, S) \Rightarrow^* (p, w) \text{ for some } p \in Q \}.$$

A production of the form $(p, A) \to (q, \varepsilon)$ is called an ε-rule, and an ECFG is called ε-free, if it does not contain any ε-rules.

The context-free grammars with states are obviously a generalization of the context-free grammars. However, if we require that each step is performed in a *leftmost* fashion, that is, $\beta \in \Sigma^*$ in the above definition of $\Rightarrow$, then the derivations of an ECFG can be simulated by a pushdown automaton. On the other hand, it is easily seen that for ECFGs different derivation strategies will in general give different languages. In addition to the leftmost and the unrestricted derivation modes, we will be interested in the *leftish* derivation mode. Here a derivation step $(p, \beta A\gamma) \Rightarrow (q, \beta\alpha\gamma)$ of an ECFG is called *leftish*, if no rule can be applied to the prefix β. Thus, β may contain occurrences of variables, but under the current state p none of them can be rewritten with a production from P. In fact, under the various derivation modes the ECFGs are equivalent in expressive power to the matrix grammars ([13,17], see also, e.g., [6, Sections 1.4 and 2.2]). Hence, concerning the language classes generated by the various types of ECFGs, we have the following results.

Proposition 2.2.
(a) $\varepsilon\text{-free-CFL} = \mathcal{L}_{\mathrm{lm}}(\varepsilon\text{-free-ECFG}) \subsetneq \mathcal{L}(\varepsilon\text{-free-ECFG}) \subsetneq \mathcal{L}_{\mathrm{lt}}(\varepsilon\text{-free-ECFG}) = \mathrm{CSL}$.
(b) $\mathrm{CFL} = \mathcal{L}_{\mathrm{lm}}(\mathrm{ECFG}) \subsetneq \mathcal{L}(\mathrm{ECFG}) \subsetneq \mathcal{L}_{\mathrm{lt}}(\mathrm{ECFG}) = \mathrm{RE}$.
(c) *The language classes* $\mathcal{L}_{\mathrm{lt}}(\varepsilon\text{-free-ECFG})$ *and* $\mathcal{L}(\mathrm{ECFG})$ *are incomparable under inclusion.*

Here CFL (CSL,RE) denotes the class of context-free (context-sensitive, recursively enumerable) languages.

Now we come to the announced definition of a new type of alternating grammar, combining the notion of alternation with that of a context-free grammar with states.

Definition 2.3. An *extended alternating context-free grammar*, EACFG for short, is a context-free grammar $G = (Q, V, U, \Sigma, P, S, q_0, F)$ with states, in which a subset U of the set of variables V is designated as *universal variables*, and a subset F of the set of states Q is designated as final states.

The *derivation relation* $\Rightarrow_G^*$ defined by G is the reflexive and transitive closure of the relation $\Rightarrow_G$ that is defined as follows. Let $(p, \beta A\gamma)$ be a sentential form of G, where $p \in Q$, $\beta, \gamma \in (V \cup \Sigma)^*$, and $A \in V$. If A is an existential variable, that is, $A \in V \setminus U$, and $(p, A) \rightarrow (q, \alpha)$ is a production from P, then $(p, \beta A\gamma) \Rightarrow_G (q, \beta\alpha\gamma)$. If, however, A is a universal variable, and if $(p, A) \rightarrow (q_i, \alpha_i)$ $(1 \leqslant i \leqslant k)$ are all the productions with left-hand side (p, A), then

$$(p, \beta A\gamma) \Rightarrow_G ((q_1, \beta\alpha_1\gamma), \ldots, (q_k, \beta\alpha_k\gamma)).$$

Hence, in the latter case all the rules with left-hand side (p, A) are applied in parallel, and following this step all the resulting sentential forms are rewritten further, independently of each other. In this way a *derivation tree* is obtained from G in analogy to the computation tree that is associated with an alternating automaton and its input.

The language $L(G)$ that is generated by G consists of all those words $w \in \Sigma^*$ for which there exists a derivation tree such that the root is labelled with the pair (q_0, S) and each leaf is labelled with a pair of the form (p, w) with $p \in F$. Here we remark that the labels of different leaves may differ in the first component, that is, in the final state, but that they must agree in the second component, that is, in the terminal string generated.

There is a slight difference in the way the states are used in ECFGs and in EACFGs, as the latter distinguish between final and non-final states. However, it can be shown that it would

$$\begin{tikzcd}
& \text{EACFG} \arrow[r, leftrightarrow, "\text{altern. states vs.}", "\text{altern. variables}"'] & \text{sACFG} \\
\text{ACFG} \arrow[ur, "\text{states}"'] & & \\
& \text{ECFG} \arrow[uu, "\text{alternation}"] \arrow[uur, "\text{alternation}"'] & \\
\text{CFG} \arrow[uu, "\text{alternation}"] \arrow[ur, "\text{states}"'] & &
\end{tikzcd}$$

Fig. 1. The taxonomy of alternating context-free grammars.

not make a difference in the expressive power of ECFGs, if they also made this distinction (see Lemmas 4.7 and 4.8). EACFGs are obtained from ACFGs by introducing states, that is, in essentially the same way as ECFGs are obtained from context-free grammars. Also EACFGs can be seen as being obtained from ECFGs by distinguishing between universal and existential variables, that is, in essentially the same way as ACFGs are obtained from context-free grammars. Hence, the EACFGs unify these two generalizations of context-free grammars.

As it will turn out, however, the concept of the EACFG is equivalent to the following concept, where alternation is governed by states and not by variables.

Definition 2.4. A *state-alternating context-free grammar*, sACFG for short, is an ECFG in which we distinguish between existential and universal states, and in which we mark certain states as *final*. Let $G = (Q, U, V, \Sigma, P, S, q_0, F)$ be such a grammar, where $U \subseteq Q$ is the set of *universal* states and $F \subseteq Q$ is the set of *final* states.

The *derivation relation* $\Rightarrow_G^*$ is defined on the set $Q \times (V \cup \Sigma)^*$ of *extended sentential forms*. Let $p \in Q$ and $\alpha \in (V \cup \Sigma)^*$. If p is an existential state, that is, $p \in Q \setminus U$, then $(p, \alpha) \Rightarrow_G (q, \alpha_1\beta\alpha_2)$, if $\alpha = \alpha_1 A \alpha_2$, and there exists a production of the form $(p, A) \to (q, \beta)$. If p is a universal state, α has the factorization $\alpha = \alpha_1 A \alpha_2$, and $(p, A) \to (q_i, \beta_i)$ $(1 \leqslant i \leqslant k)$ are all the productions with left-hand side (p, A), then

$$(p, \alpha) \Rightarrow_G ((q_1, \alpha_1\beta_1\alpha_2), \ldots, (q_k, \alpha_1\beta_k\alpha_2)),$$

that is, all these productions are applied in parallel to the chosen occurrence of the variable A, and following this step all these sentential forms are rewritten further, independently of each other. In this way a derivation tree is obtained.

The language $L(G)$ that is generated by G consists of all words $w \in \Sigma^*$ for which there exists a derivation tree such that the root is labelled with (q_0, S) and all leaves are labelled with pairs of the form (p, w) with $p \in F$. Note that, as for an EACFG, the labels of different leaves may differ in their first components.

Fig. 1 puts the various generalizations of context-free grammars introduced so far into perspective.

Below we will see that sACFGs are actually equivalent in expressive power to EACFGs.

Lemma 2.5. *For each* EACFG G, *an* sACFG G' *can be constructed such that* $L_{\mathsf{m}}(G) = L_{\mathsf{m}}(G')$ *holds for each derivation mode* m. *Moreover, if* G *is* ε*-free and*/or (*right-*) *linear, then so is* G'.

Proof. Let $G = (Q, V, U, \Sigma, P, S, q_0, F)$ be an EACFG. From G we construct an sACFG G' that accepts the same language as G. For each state of G, G' will have an existential as well as a universal state. Derivation steps of G involving existential variables will be simulated in G' by using existential states. Derivation steps of G, however, that involve universal variables, will be simulated by G' by first changing from the current existential state to the corresponding universal state, and by then simulating the universal derivation step of G. Accordingly, we take $G' := (Q^{\exists} \cup Q^{\forall}, Q^{\forall}, V, \Sigma, P', S, q_0^{\exists}, F^{\exists})$, where $Q^{\exists} := \{ q^{\exists} \mid q \in Q \}$, $F^{\exists} := \{ q^{\exists} \mid q \in F \}$, and $Q^{\forall} := \{ q^{\forall} \mid q \in Q \}$, and we define the set P' of productions of G' as follows:

(1) For $X \in V \setminus U$, if $(p, X) \to (q, \alpha)$ is in P, where $p, q \in Q$ and $\alpha \in (V \cup \Sigma)^*$, then $(p^{\exists}, X) \to (q^{\exists}, \alpha)$ is included in P'.
(2) For each $X \in U$, if $(p, X) \to (q, \alpha)$ is in P, where $p, q \in Q$ and $\alpha \in (V \cup \Sigma)^*$, then $(p^{\exists}, X) \to (p^{\forall}, X)$ and $(p^{\forall}, X) \to (q^{\exists}, \alpha)$ are included in P'.

Clearly, if G is ε-free and/or (right-) linear, then so is G'. We will see that, for any derivation tree T of G that generates a terminal word w, there is a derivation tree T' of G' that generates the same word, and vice versa.

The tree T' is obtained from T inductively as follows. The root of T is labelled with the pair (q_0, S), while the root of T' is labelled with the pair $(q_0^{\exists}, S)$. Now let v be a node of T with label $(p, \alpha X \gamma)$, where $p \in Q$, $X \in V$, and $\alpha, \gamma \in (V \cup \Sigma)^*$, such that under the derivation mode m the distinguished variable occurrence of X is to be rewritten next, and assume that T' contains a corresponding node v' with label $(p^{\exists}, \alpha X \gamma)$. If the variable X is existential, then the node v has a single son v_1 with label $(q, \alpha\beta\gamma)$, where $(p, X) \to (q, \beta)$ is a production of P. Accordingly, the node v' will get a son v_1' with label $(q^{\exists}, \alpha\beta\gamma)$, which corresponds to an application of the corresponding production from group (1). If the variable X is universal, then the node v will have sons $v_1, \ldots, v_k$ with labels $(q_i, \alpha\beta_i\gamma)$, where $(p, X) \to (q_i, \beta_i)$, $1 \leqslant i \leqslant k$, are the productions of P with left-hand side (p, X). Now the node v' of T' will get the single son v'' with label $(p^{\forall}, \alpha X \gamma)$, which corresponds to an application of a rule from group (2), and the node v'' will get sons $v_1', \ldots, v_k'$ with labels $(q_i^{\exists}, \alpha\beta_i\gamma)$, $1 \leqslant i \leqslant k$, corresponding to the applications of the rules of group (2) with left-hand side $(p^{\forall}, X)$. Hence, we see that $L_{\mathsf{m}}(G) \subseteq L_{\mathsf{m}}(G')$ holds.

Conversely, if $w \in L_{\mathsf{m}}(G')$, then there is a derivation tree T' for G' such that the root is labelled with $(q_0^{\exists}, S)$ and each leaf is labelled with $(q^{\exists}, w)$ for some $q \in F$. Each application of a production from group (1) corresponds directly to an application of a production from P. Further, each production of the form $(p^{\exists}, X) \to (p^{\forall}, X)$ must be followed by an application of a production from group (2) that corresponds to a production from P. Hence, it is easily seen that from T' we obtain a derivation tree T for G such that the root is labelled with (q_0, S) and each leaf is labelled with (q, w) for some $q \in F$. Thus, we see that the languages $L_{\mathsf{m}}(G)$ and $L_{\mathsf{m}}(G')$ coincide. □

Lemma 2.6. *For each* sACFG G, *an* EACFG G' *can be constructed such that* $L_{\mathsf{m}}(G) = L_{\mathsf{m}}(G')$ *holds for each derivation mode* m. *Moreover, if* G *is* ε*-free and/or (right-) linear, then so is* G'.

Proof. Let $G = (Q, U, V, \Sigma, P, S, q_0, F)$ be an sACFG. From G we construct an EACFG G' that generates the same language as G. For each variable of G, G' will have an existential and a universal variable. The start symbol of G' will be an existential variable, and as long as G uses existential states in a derivation, G' will use only existential variables. If, however, G changes into a universal state in the course of a derivation, then G' can replace the (existential) variable that is to be rewritten next according to the derivation mode m by its universal variant, and using that universal variable it can simulate the current universal derivation step of G. Accordingly, we take $G' := (Q \cup Q', V^{\exists} \cup V^{\forall}, V^{\forall}, \Sigma, P', S^{\exists}, q_0, F)$, where $Q' := \{ q' \mid q \in U \}$, $V^{\exists} := \{ X^{\exists} \mid X \in V \}$ and $V^{\forall} := \{ X^{\forall} \mid X \in V \}$. Further, let $\varphi^{\exists} : (V \cup \Sigma)^* \to (V^{\exists} \cup \Sigma)^*$ be the morphism that replaces each occurrence of each variable X by the variable $X^{\exists}$. Then P' is defined as follows:
(1) For $p \in Q \setminus U$, if $(p, X) \to (q, \alpha)$ is in P, where $X \in V$, $q \in Q$ and $\alpha \in (V \cup \Sigma)^*$, then $(p, X^{\exists}) \to (q, \varphi^{\exists}(\alpha))$ is included in P'.
(2) For each $p \in U$, if $(p, X) \to (q, \alpha)$ is in P, where $X \in V$, $q \in Q$ and $\alpha \in (V \cup \Sigma)^*$, then $(p, X^{\exists}) \to (p', X^{\forall})$ and $(p', X^{\forall}) \to (q, \varphi^{\exists}(\alpha))$ are included in P'.
Clearly, if G is ε-free and/or (right-) linear, then so is G'. As in the proof of the previous lemma it can now be shown that, for each derivation tree T of G that generates a terminal word w, there is a derivation tree T' of G' that generates the same word, and vice versa. Thus, the languages $L_{\mathsf{m}}(G)$ and $L_{\mathsf{m}}(G')$ coincide. □

From these two lemmata we obtain the following result.

Corollary 2.7. EACFG *and* sACFG *have exactly the same expressive power.*

Thus, in the main part of the paper we will not discuss EACFGs anymore, but consider sACFGs instead.

3. A lower bound for $\mathcal{L}_{\mathsf{lm}}(\mathsf{ACFG})$

Here, we will establish a lower bound for the generative capacity of ACFGs.

For each integer $k \geqslant 1$, let CFL_k denote the class of languages that can be written as the intersection of k context-free languages, and let $\mathsf{CFL}_\omega := \bigcup_{k \geqslant 0} \mathsf{CFL}_k$. According to Liu and Weiner [12] the classes CFL_k form an infinite hierarchy within the class of context-sensitive languages.

Example 3.1. For $i = 1, 2$, let $G_i = (V_i, U_i, \Sigma, P_i, S_i)$ be an ACFG, where we assume that $V_1 \cap V_2 = \emptyset$. Let S be a new variable, and let $G := (V, U, \Sigma, P, S)$ be defined by taking $V := V_1 \cup V_2 \cup \{S\}$, $U := U_1 \cup U_2 \cup \{S\}$, and $P := P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}$. Then it is easily seen that $L_{\mathsf{m}}(G) = L_{\mathsf{m}}(G_1) \cap L_{\mathsf{m}}(G_2)$ for each derivation mode m.
On the other hand, if we take $U := U_1 \cup U_2$, that is, the start symbol S of G is taken to be an existential variable, then $L_{\mathsf{m}}(G) = L_{\mathsf{m}}(G_1) \cup L_{\mathsf{m}}(G_2)$ for each derivation mode m.

This shows that, for each derivation mode m, the language class $\mathcal{L}_{\mathsf{m}}(\mathsf{ACFG})$ is closed under intersection and union. As each context-free grammar can be regarded as an ACFG with no universal variables, we obtain the following inclusion.

Observation 3.2. $\mathsf{CFL}_\omega \subseteq \mathcal{L}_{\mathsf{lm}}(\mathsf{ACFG}) \cap \mathcal{L}(\mathsf{ACFG})$.

In the following we will prove that this is actually a proper inclusion by considering a sequence of example languages L_k ($k \geqslant 2$) and L_ω. For $k \geqslant 2$, let $\Sigma_k := \{a_1, a_2, \ldots, a_k\}$, and let L_k be the language

$$L_k := \{ (a_1^{i_1} \cdot a_2^{i_2} \cdot \ldots \cdot a_k^{i_k})^2 \mid i_j \geqslant 1 \} \subseteq \Sigma_k^*.$$

Further let

$$L_\omega := \{ (a^{i_1} \cdot b^1 \cdot a^{i_2} \cdot b^2 \cdot \ldots \cdot a^{i_k} \cdot b^k)^2 \mid k \geqslant 0, i_j \geqslant 1 \} \subseteq \{a, b\}^*.$$

Liu and Weiner [12] proved that, for each $k \geqslant 2$,

$$\{(a_1^{i_1} \cdot a_2^{i_2} \cdot \ldots \cdot a_k^{i_k})^2 \mid i_j \geqslant 0\} \in \mathsf{CFL}_k \setminus \mathsf{CFL}_{k-1}.$$

From their proof the following lemma follows easily.

Lemma 3.3. *For each* $k \geqslant 2$, $L_k \in \mathsf{CFL}_k \setminus \mathsf{CFL}_{k-1}$.

For each $k \geqslant 2$, we define a partial mapping $\varphi_k : \{a, b\}^* \to \Sigma_k^*$ with domain $\mathsf{dom}(\varphi_k) := (a^+ \cdot b^1 \cdot a^+ \cdot b^2 \cdot \ldots \cdot a^+ \cdot b^k)^2$ by

$$\varphi_k : a^{i_1} \cdot b^1 \cdot \ldots \cdot a^{i_k} \cdot b^k \cdot a^{j_1} \cdot b^1 \cdot \ldots \cdot a^{j_k} \cdot b^k \mapsto a_1^{i_1} \cdot \ldots \cdot a_k^{i_k} \cdot a_1^{j_1} \cdot \ldots \cdot a_k^{j_k}.$$

Note that it follows from the definition that $\varphi_k(w)$ is undefined for each word w that does *not* belong to the set $(a^+ \cdot b^1 \cdot a^+ \cdot b^2 \cdot \ldots \cdot a^+ \cdot b^k)^2$. Obviously, φ_k is an injective mapping satisfying $\varphi_k(L_\omega) = L_k$. Further, there exists a *generalized sequential machine* (GSM) (see, e.g., [7]) that, given a string $w \in \mathsf{dom}(\varphi_k)$ as input, computes the string $\varphi_k(w)$. On the other hand there is the following negative result.

Lemma 3.4. $L_\omega \notin \mathsf{CFL}_\omega$.

Proof. Assume that $L_\omega = \bigcap_{i=1}^m N_i$ for some context-free languages N_i, $1 \leqslant i \leqslant m$. We may assume without loss of generality that $N_i \subseteq \{a, b\}^*$. Then we obtain the following equalities from the injectivity of φ_{m+1}:

$$L_{m+1} = \varphi_{m+1}(L_\omega) = \varphi_{m+1}\left(\bigcap_{i=1}^m N_i\right) = \bigcap_{i=1}^m \varphi_{m+1}(N_i).$$

Since CFL is closed under GSM mappings, this contradicts the fact that $L_{m+1} \notin \mathsf{CFL}_m$ (Lemma 3.3). □

In contrast to the result above, we will now see that the language L_ω is generated by an ACFG. Actually we have the following result.

Lemma 3.5. $L_\omega \in \mathcal{L}_{\mathsf{lm}}(\mathsf{ACFG}) \cap \mathcal{L}(\mathsf{ACFG})$.

Proof. First, in order to simplify the discussion, we allow an arbitrary string w consisting of terminals and variables to be used as the start string for an ACFG. We denote by $L_{\mathsf{lm}}(G, w)$ the language generated by G from the initial string w using the leftmost derivation mode, and by $L(G, w)$ we denote the language generated by G from the initial string w by the unrestricted derivation mode. It is not difficult to give context-free grammars G_1 to G_4 with pairwise disjoint sets of variables that generate the following languages over $\Sigma := \{a, b\}$:

$$L_{\mathsf{lm}}(G_1, B) = a^+ \cdot b^+,$$

$$L_{\mathsf{lm}}(G_2, C) = a^+ \cdot b^+ \cdot a^+ \cdot ba \cdot \Sigma^*,$$

$$L_{\mathsf{lm}}(G_3, M) = \bigcup_{m \geqslant 1} (a^m \cdot b^+ \cdot a \cdot \Sigma^* \cdot b \cdot a^m),$$

$$L_{\mathsf{lm}}(G_4, T) = \{\varepsilon\} \cup a \cdot \Sigma^*,$$

where B, C, M, and T are variables.

Secondly, let

$$J := \bigcup_{n \geqslant 1} (a^+ \cdot b^n \cdot a^+ \cdot b^{n+1}),$$

$$I_1 := (J^2)^* \cup \bigcup_{k \geqslant 0} (J^k \cdot a^+ \cdot b^+ \cdot a^+ \cdot b \cdot J^k),$$

$$I_2 := a^+ \cdot b \cdot (J^2)^* \cdot a^+ \cdot b^+ \cup \bigcup_{k \geqslant 0} (a^+ \cdot b \cdot J^k \cdot a^+ \cdot b^+ \cdot a^+ \cdot b \cdot J^k \cdot a^+ \cdot b^+) \cup \{\varepsilon\},$$

and

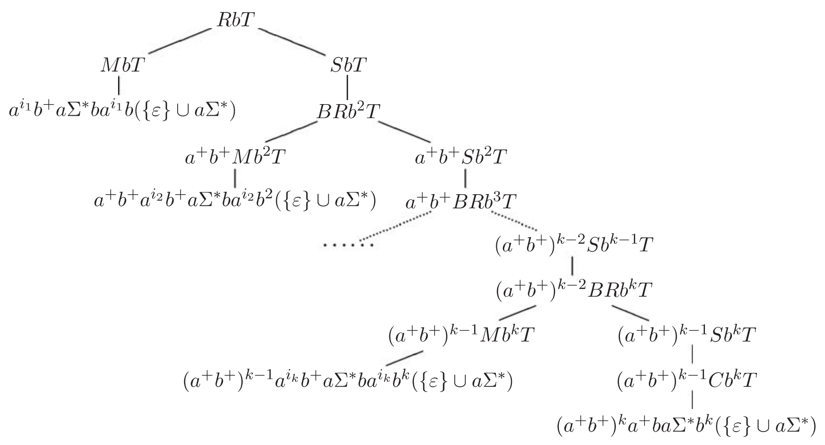
$$I := \bigcup_{k \geqslant 1} (a^+ \cdot b^1 \cdot a^+ \cdot b^2 \cdot \ldots \cdot a^+ \cdot b^k)^2 \cup \{\varepsilon\}.$$

Then we have the following equality.

Claim. $I = I_1 \cap I_2$.

Proof. Let $w := a^{r_1} \cdot b \cdot a^{r_2} \cdot b^2 \cdot \ldots \cdot a^{r_k} \cdot b^k \cdot a^{s_1} \cdot b \cdot a^{s_2} \cdot b^2 \cdot \ldots \cdot a^{s_k} \cdot b^k$ be an element of the language I. For each $i = 1, \ldots, k-1$, $a^{r_i} \cdot b^i \cdot a^{r_{i+1}} \cdot b^{i+1} \in J$ and $a^{s_i} \cdot b^i \cdot a^{s_{i+1}} \cdot b^{i+1} \in J$. Thus, if $k = 2l$ for some $l > 0$, then $w \in (J^2)^k$, and therewith $w \in I_1$. Further, $a^{r_2} \cdot b^2 \cdot a^{r_3} \cdot b^3 \cdot \ldots \cdot a^{r_{k-1}} \cdot b^{k-1} \in J^{l-1}$ and $a^{s_2} \cdot b^2 \cdot a^{s_3} \cdot b^3 \cdot \ldots \cdot a^{s_{k-1}} \cdot b^{k-1} \in J^{l-1}$, implying that $w \in a^+ \cdot b \cdot J^{l-1} \cdot a^+ \cdot b^+ \cdot a^+ \cdot b \cdot J^{l-1} \cdot a^+ \cdot b^+$, which means that $w \in I_2$.

If $k = 2l+1$, then $a^{r_1} \cdot b \cdot a^{r_2} \cdot b^2 \cdot \ldots \cdot a^{r_{k-1}} \cdot b^{k-1} \in J^l$ and $a^{s_2} \cdot b^2 \cdot a^{s_3} \cdot b^3 \cdot \ldots \cdot a^{s_k} \cdot b^k \in J^l$, which shows that $w \in J^l \cdot a^+ \cdot b^+ \cdot a^+ \cdot b \cdot J^l$. Hence, also in this case $w \in I_1$. Further,



RbT
MbT
SbT
$a^{i_1}b^+a\Sigma^*ba^{i_1}b(\{\varepsilon\}\cup a\Sigma^*)$
BRb^2T
$a^+b^+Mb^2T$
$a^+b^+Sb^2T$
$a^+b^+a^{i_2}b^+a\Sigma^*ba^{i_2}b^2(\{\varepsilon\}\cup a\Sigma^*)$
$a^+b^+BRb^3T$
$(a^+b^+)^{k-2}Sb^{k-1}T$
$(a^+b^+)^{k-2}BRb^kT$
$(a^+b^+)^{k-1}Mb^kT$
$(a^+b^+)^{k-1}Sb^kT$
$(a^+b^+)^{k-1}a^{i_k}b^+a\Sigma^*ba^{i_k}b^k(\{\varepsilon\}\cup a\Sigma^*)$
$(a^+b^+)^{k-1}Cb^kT$
$(a^+b^+)^ka^+ba\Sigma^*b^k(\{\varepsilon\}\cup a\Sigma^*)$

Fig. 2. A schematic representation of a typical derivation tree for G.

$a^{r_2} \cdot b^2 \cdot a^{r_3} \cdot b^3 \cdot \ldots \cdot a^{r_k} \cdot b^k \cdot a^{s_1} \cdot b \cdot a^{s_2} \cdot b^2 \cdot \ldots \cdot a^{s_{k-1}} \cdot b^{k-1} \in (J^2)^l$, and so $w \in I_2$. Hence, we see that $I \subseteq I_1 \cap I_2$.

Conversely, assume that $w \in I_1 \cap I_2$. Let n be the number of factors of the form $a^+ \cdot b^+$ in w. From the definition of I_1, we see immediately that n is an even number. So let $n = 2k$. If $k = 0$, then $w = \varepsilon$, and so $w \in I$. So assume that $k \geqslant 1$. Hence, w has the form

$$w = a^{r_1} \cdot b^{t_1} \cdot \ldots \cdot a^{r_k} \cdot b^{t_k} \cdot a^{r_{k+1}} \cdot b^{t_{k+1}} \cdot \ldots \cdot a^{r_{2k}} \cdot b^{t_{2k}},$$

where all exponents are positive integers. As $w \in I_1$, we see that there are two cases. Either k is even, and then $a^{r_i} \cdot b^{t_i} \cdot a^{r_{i+1}} \cdot b^{t_{i+1}} \in J$ for each $i \in \{1, 3, 5, \ldots, 2k-1\}$, or $k = 2l + 1$, and then $a^{r_i} \cdot b^{t_i} \cdot a^{r_{i+1}} \cdot b^{t_{i+1}} \in J$ for each $i \in \{1, 3, 5, \ldots, 2l-1\}$ and each $i \in \{k+2, k+4, \ldots, 2k-1\}$, and $t_{k+1} = 1$. As $w \in I_2$, too, it follows in each case that $t_i = t_{k+i} = i$ for $i = 1, 2, \ldots, k$. Hence, we can conclude that $w \in I$. $\Box$

Obviously J, I_1, and I_2 are context-free. Thus, there exists an ACFG G_5 such that $L_{\mathsf{lm}}(G_5) = L(G_5) = I$ (see Example 3.1).

Finally, we consider the ACFG G which has all the productions of G_1 to G_4 together with the following productions:

$$R \to M,\ R \to S,\ S \to C,\ S \to BRb.$$

Here R is a new universal variable and S is a new existential variable. We consider the languages $L_{\mathsf{lm}}(G, RbT)$ and $L(G, RbT)$ that are generated by G from the initial string RbT in leftmost and in unrestricted derivation mode, respectively. In Fig 2 an informal pictorial description of a typical derivation tree for G with respect to the unrestricted derivation mode is given, and in Fig. 3 a leftmost derivation tree for the string $a^3ba^2b^2a^3ba^2b^2$ is depicted. We see that within these derivation trees certain dependencies are established between pairs of powers of a occurring in the string generated. For example, the first block $a^{i_1}b^+$ is related to a block $a^{i_1}b$, and the second block $a^{i_2}b^+$ is related to a block $a^{i_2}b^2$. Thus, if we restrict

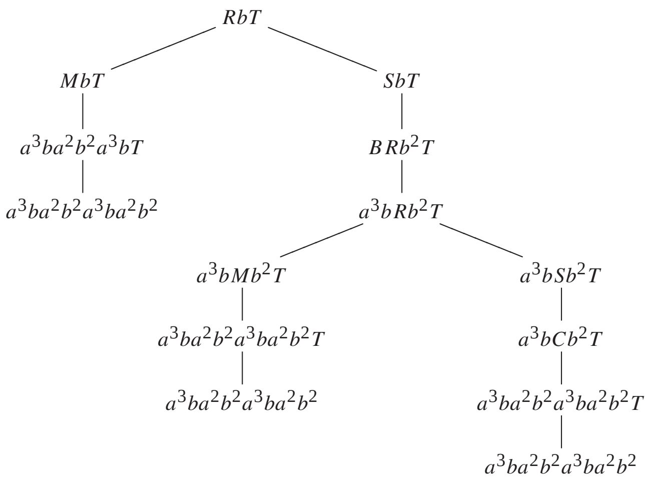


Fig. 3. A leftmost derivation tree for $a^3ba^2b^2a^3ba^2b^2$ in G.

our attention to strings that in addition belong to the language I, then we see by inspection that

$$\begin{aligned} L_\omega &= I \cap \big(\{\varepsilon\} \cup \{a^iba^ib \mid i > 0\} \cup L_{\mathsf{lm}}(G, RbT)\big) \\ &= I \cap \big(\{\varepsilon\} \cup \{a^iba^ib \mid i > 0\} \cup L(G, RbT)\big). \end{aligned}$$

Obviously $\{\varepsilon\}$ and $\{a^iba^ib \mid i \geqslant 1\}$ can be generated by context-free grammars. As $\mathcal{L}_{\mathsf{lm}}(\mathsf{ACFG})$ and $\mathcal{L}(\mathsf{ACFG})$ are both closed under union and intersection (see Example 3.1), it follows that L_ω belongs to $\mathcal{L}_{\mathsf{lm}}(\mathsf{ACFG})$ as well as to $\mathcal{L}(\mathsf{ACFG})$. $\square$

From Observation 3.2 and Lemmas 3.4 and 3.5, we obtain the main result of this section.

Theorem 3.6. $\mathsf{CFL}_\omega \subsetneq \mathcal{L}_{\mathsf{lm}}(\mathsf{ACFG}) \cap \mathcal{L}(\mathsf{ACFG})$.

However, the following problem remains unanswered.

Open Problem 1. Does L_ω separate $\mathcal{L}_{\mathsf{lm}}(\mathsf{ACFG})$ or $\mathcal{L}(\mathsf{ACFG})$ from the Boolean closure of CFL?

4. Basic properties of ACFGs and sACFGs

Next, we will establish some basic properties of the language classes that are generated by various kinds of ACFGs and sACFGs. We will mainly concentrate on the leftmost derivation strategy, but some other strategies will also be considered at various places. In particular, we will establish a normal form result for both these types of grammars.

As each ACFG can be interpreted as an EACFG with only a single state, we obtain the following result from Lemma 2.5.

Lemma 4.1. *For each* ACFG G, *we can construct an* sACFG G' *such that* $L_{\mathsf{m}}(G) = L_{\mathsf{m}}(G')$ *holds for each derivation mode* m. *Moreover, if* G *is* ε*-free and/or (right-) linear, then so is* G'.

Open Problem 2. Does the converse of Lemma 4.1 hold, that is, can each sACFG be simulated by an ACFG? Observe that this is equivalent to asking whether each EACFG is equivalent to an EACFG with only a single state. For the leftish mode this is not possible (see Section 7), but is it at least possible for the leftmost derivation mode?

At least for linear grammars we do have the converse of Lemma 4.1. Observe that for linear grammars all derivation modes are equivalent.

Lemma 4.2. *For each linear* sACFG G, *we can construct a linear* ACFG G' *such that* G *and* G' *generate the same language. Moreover, if* G *is right-linear and/or* ε*-free, then so is* G'.

Proof. Let $G = (Q, U, V, \Sigma, P, S, q_0, F)$ be a linear sACFG. We obtain a linear ACFG G' that simulates G by introducing variables that combine the variables of G with the states of G. The linear ACFG $G' := (V', U', \Sigma, P', S')$ is defined as follows:
- $V' := \{ [q, A] \mid q \in Q, A \in V \} \cup \{E\}$, where E is a new symbol,
- $U' := \{ [q, A] \mid q \in U, A \in V \}$,
- $S' := [q_0, S]$, and
- P' is obtained as follows:

(1) If $(p, A) \to (q, xBy)$ is in P, where $A, B \in V$ and $x, y \in \Sigma^*$, then $[p, A] \to x[q, B]y$ is included in P'.

(2) If $(p, A) \to (q, x)$ is in P, where $A \in V$, $x \in \Sigma^*$, and $q \in F$, then $[p, A] \to x$ is included in P'.

(3) If $(p, A) \to (q, x)$ is in P, where $A \in V$, $x \in \Sigma^*$, and $q \in Q \setminus F$, then $[p, A] \to E$ is included in P'.

Obviously the productions of group (3) cannot be used in any successful G'-derivation. They correspond to applications of G-productions of the form $(p, A) \to (q, x)$ with $x \in \Sigma^*$ and $q \in Q \setminus F$, which cannot occur in any successful G-derivation, either. However, these productions have to be included in P', as otherwise certain universal derivation steps of G' would be successful although the corresponding derivation steps of G are not.

Clearly, G' is right-linear and/or ε-free, if G is. Now let w be a terminal string such that $w \in L(G)$. Then there is a derivation tree T of G with root labelled by (q_0, S) and each leaf labelled by (q, w) for some final state q. From T we obtain a derivation tree T' of G' by replacing each node v with label (p, xAy) $(p \in Q, x, y \in \Sigma^*, A \in V)$ by a node v' with label $x[p, A]y$, and by replacing each node v with label (p, x) $(p \in F, x \in \Sigma^*)$ by a node v' with label x. Then T' witnesses the fact that $w \in L(G')$, that is, $L(G) \subseteq L(G')$ holds.

As the productions of group (2) are added to P' only for those productions of P for which the state entered during the production considered is final, we see that all terminal strings that can be generated by G' can also be generated by G, that is, we have $L(G) = L(G')$. □

The above lemmas yield the following consequences.

Theorem 4.3.
(a) $\mathcal{L}(\varepsilon\text{-free-right-lin-ACFG}) = \mathcal{L}(\varepsilon\text{-free-right-lin-sACFG})$ *and*
$\mathcal{L}(\text{right-lin-ACFG}) = \mathcal{L}(\text{right-lin-sACFG})$.
(b) $\mathcal{L}(\varepsilon\text{-free-lin-ACFG}) = \mathcal{L}(\varepsilon\text{-free-lin-sACFG})$ *and*
$\mathcal{L}(\text{lin-ACFG}) = \mathcal{L}(\text{lin-sACFG})$.
(c) $\mathcal{L}_\mathsf{m}(\varepsilon\text{-free-ACFG}) \subseteq \mathcal{L}_\mathsf{m}(\varepsilon\text{-free-sACFG})$ *and* $\mathcal{L}_\mathsf{m}(\text{ACFG}) \subseteq \mathcal{L}_\mathsf{m}(\text{sACFG})$, *where* m *is any of the leftmost*, *the leftish*, *the rightmost or the unrestricted derivation modes*.

Here, the *rightmost derivation mode* rm is defined in analogy to the leftmost derivation mode.

Based on Example 3.1 it is easily seen that the language class $\mathcal{L}(\text{lin-ACFG})$ is closed under intersection. As the non-context-free language $\{\, a^n b^n c^n \mid n \geqslant 1 \,\}$ is easily described as the intersection of two linear languages, it follows that the class LIN of linear languages is properly contained in $\mathcal{L}(\text{lin-ACFG})$.

We close this section by establishing normal forms for ACFGs and sACFGs. First we will see that for these grammars a normal form exists that is similar to the Chomsky normal form for context-free grammars, and then we will show that we can assume without loss of generality that all states of any sACFG are final.

Definition 4.4. A production $(A \to \alpha)$ of an ACFG $G = (V, U, \Sigma, P, S)$ or a production $(p, A) \to (q, \alpha)$ of an sACFG $G = (Q, U, V, \Sigma, P, S, q_0, F)$, respectively, is a *unit-production* if α is a variable.

The ACFG or the sACFG G, respectively, is said to be in *weak Chomsky normal form* if it satisfies the following conditions:
(1) each production $(A \to \alpha)$ or $(p, A) \to (q, \alpha)$, respectively, satisfies the condition that $\alpha \in (V \cup V^2 \cup \Sigma \cup \{\varepsilon\})$;
(2) for each variable $A \in V$ or for each pair $(p, A) \in Q \times V$, respectively, if there are two or more productions with left-hand side A or (p, A), respectively, then all these productions are unit-productions.
Thus, if a (state-) alternating context-free grammar is in weak Chomsky normal form, then it is only for unit-productions that it plays a role whether the actual variable (or state) is universal or existential.

Lemma 4.5. *For each* ACFG G, *we can construct an* ACFG G' *in weak Chomsky normal form such that* G *and* G' *are equivalent with respect to the leftmost*, *the leftish and the unrestricted derivation modes*. *In addition*, *if* G *is* ε*-free*, *then so is* G'.

Proof. Let $G = (V, U, \Sigma, P, S)$ be an ACFG. Following the standard construction of a context-free grammar in Chomsky normal form from a given context-free grammar, we will replace each production of P that is not in weak Chomsky normal form by a sequence of new productions. However, we have to take the existence of universal variables into account, which means that this technique has to be adopted accordingly.

First of all we introduce new existential variables $V_\Sigma := \{\hat{a} \mid a \in \Sigma\}$ and take $V' := V \cup V_\Sigma$. Then in each production of P we replace each occurrence of each terminal symbol $a \in \Sigma$ by an occurrence of the corresponding new variable $\hat{a}$. The resulting set of productions is called P_1. Next we define the set of new productions

$$P_2 := \{\hat{a} \to a \mid a \in \Sigma\}.$$

The grammar $G_1 := (V', U, \Sigma, P_1 \cup P_2, S)$ is then an ACFG such that each production is either of the form described in (1) above or it is of the form $A \to B_1 B_2 \cdots B_k$ for some $A, B_1, \ldots, B_k \in V'$ and some integer $k > 2$. Further, it is easily seen that $L_{\mathsf{lm}}(G) = L_{\mathsf{lm}}(G_1)$, as for each new variable $\hat{a} \in V_\Sigma$, there is exactly one production in $P_1 \cup P_2$ with left-hand side $\hat{a}$. Thus, under the leftmost derivation mode a sentential form $x\hat{a}\beta$, where $x \in \Sigma^*$ and $\beta \in V'^*$, has a single successor only, which is the sentential form $xa\beta$.

Next we replace the productions of the form $r := (A \to B_1 B_2 \cdots B_k)$, where A and $B_1, \ldots, B_k$ are variables and $k > 2$. For each production of this form, we introduce new (existential) variables $C_{r,2}, \ldots, C_{r,k-1}$ and replace the original production by the following productions:

$$\begin{array}{lll} A & \to & C_{r,k-1} B_k, \\ C_{r,k-1} & \to & C_{r,k-2} B_{k-1}, \\ & \cdots & \\ C_{r,3} & \to & C_{r,2} B_3, \\ C_{r,2} & \to & B_1 B_2. \end{array}$$

As the new variables are existential, and as for each production of P_1 to be replaced, a set of new variables is chosen, it is easily seen that the resulting grammar G_2 is an ACFG that satisfies condition (1) of the weak Chomsky normal form, and that G_2 is equivalent to G with respect to leftmost derivations.

Finally, for each variable A, if there are $k > 1$ productions

$$A \to \alpha_1, A \to \alpha_2, \ldots, A \to \alpha_k$$

in G_2, then we introduce k new existential variables $D_1, \ldots, D_k$ and replace these productions by the following new productions:

$$A \to D_i, D_i \to \alpha_i, \quad 1 \leqslant i \leqslant k.$$

The resulting ACFG is called G'. Obviously, it is in weak Chomsky normal form, and as each variable D_i occurs on the left-hand side of a single production of G' only, we see that the G'-derivation $A \Rightarrow D_i \Rightarrow \alpha_i$ is just the replacement of the G_2-step $A \Rightarrow \alpha_i$. Hence, in leftmost mode G' generates the same language as the original grammar G.

From the above construction we see that the grammar G' is ε-free, if G is.

Observe that for any ACFG G, $L_{\mathsf{lt}}(G) = L_{\mathsf{lm}}(G)$ holds. Further, it is easily seen that the above construction also works for the unrestricted derivation mode, as all the newly introduced variables are existential. $\square$

A symmetric construction yields the corresponding result for the rightmost derivation mode. Also for sACFGs a corresponding result holds.

Lemma 4.6. *For each* sACFG G, *we can construct an* sACFG G' *in weak Chomsky normal form such that G and G' are equivalent with respect to the leftmost, the leftish and the unrestricted derivation modes. In addition, if G is ε-free, then so is G'.*

The construction of G' from G is almost the same as the construction in the proof of Lemma 4.5. The only difference consists in the fact that a single new existential state is needed that is used to ensure that a sequence of steps of G' that simulate a single step of G is executed completely before the simulation of the next step of G begins.

Again a symmetric construction yields the corresponding result for the rightmost derivation mode. Next we will prove that for sACFGs, the notion of final states is not of particular importance. They have been introduced here, as in certain cases they are quite handy to simplify the construction of an sACFG for a particular language, but we can do without them. As ECFGs are a special case of sACFGs, this shows in particular that it does not matter whether ECFGs are defined with or without final states. In the proof we will already use Lemma 4.6.

Lemma 4.7. *For each* sACFG G, *we can construct an* sACFG G' *in weak Chomsky normal form such that G and G' are equivalent with respect to the leftmost derivation mode and all states of G' are final. In addition, if G is ε-free, then so is G'.*

Proof. Let $G = (Q, U, V, \Sigma, P, S, q_0, F)$ be an sACFG. By the previous lemma we can assume without loss of generality that G is in weak Chomsky normal form. We define an sACFG $G' := (Q, U, V', \Sigma, P', \bar{S}, q_0, Q)$ by taking $V' := V \cup \{ \bar{A} \mid A \in V \} \cup \{T\}$, where $\bar{A}$ $(A \in V)$ and T are new variables, and $P' := P \cup P_1$, where P_1 contains the following productions:
(1) $(p, \bar{A}) \to (q, B\bar{C})$, if $(p, A) \to (q, BC)$ is in P,
(2) $(p, \bar{A}) \to (q, \bar{B})$, if $(p, A) \to (q, B)$ is in P,
(3) $(p, \bar{A}) \to (q, a)$, if $(p, A) \to (q, a)$ is in P and $q \in F$,
(4) $(p, \bar{A}) \to (q, T)$, if $(p, A) \to (q, a)$ is in P and $q \notin F$.
The idea underlying this construction is as follows. In each sentential form within a derivation tree the rightmost variable is marked. For this the variables $\bar{A}$ $(A \in V)$ are used. Now this variable can eventually be rewritten into a terminal symbol only by applying a production from group (3). This, however, means that this step, which under the leftmost derivation mode ends the actual branch of the derivation tree, corresponds to a derivation step in the grammar G that enters a final state.

Now let us consider a derivation tree for a word $w \in L_{\mathrm{lm}}(G')$. Then each inner node of this tree is labelled by a pair $(p, xX\bar{A})$, where $p \in Q$, $x \in \Sigma^*$, $X \in V^*$, and $A \in V$, while each leaf is labelled with a pair (p, w) for some $p \in Q$. As we can get rid of the variables of the form $\bar{A}$ only by applying productions from group (3), we actually see that, for the label (p, w) of each leaf, $p \in F$ holds. Thus, by replacing each occurrence of $\bar{A}$ $(A \in V)$ by A, we obtain a derivation tree that witnesses that $w \in L_{\mathrm{lm}}(G)$ holds.

Hence, we see that $L_{\mathsf{lm}}(G') \subseteq L_{\mathsf{lm}}(G)$. Also the converse inclusion is easily verified. Thus, we see that $L_{\mathsf{lm}}(G') = L_{\mathsf{lm}}(G)$ holds.

Observe that the productions of group (4) are necessary to cover the case that the state p is universal. If in G all productions with left-hand side (p, A) are applied simultaneously, where A is the last (and therewith rightmost) occurrence of a variable in the actual sentential form, then a G-production of the form $(p, A) \to (q, a)$ with $q \notin F$ will result in a deadend in the derivation tree generated, that is, this tree will not generate a valid terminal string. However, if the corresponding production $(p, \bar{A}) \to (q, T)$ was missing from G', then the resulting G'-derivation tree would have no branch that corresponds to the deadend in the G-tree mentioned, and hence, it might lead to generating a valid terminal string.

Obviously, if G is ε-free, then so is G'. □

The above construction makes essential use of the fact that leftmost derivations are considered. However, the corresponding result also holds for the leftish and the unrestricted derivation modes.

Lemma 4.8. *For each* sACFG G, *we can construct an* sACFG G' *such that G and G' are equivalent with respect to leftish and unrestricted derivations and all states of G' are final. In addition, if G is ε-free, then so is G'.*

Proof. Actually the construction is much simpler than the one for the leftmost derivation mode. Let $G = (Q, U, V, \Sigma, P, S, q_0, F)$ be an sACFG. We define an sACFG $G' := (Q, U, V', \Sigma, P', S, q_0, Q)$ by taking $V' := V \cup \{\hat{a} \mid a \in \Sigma \cup \{\varepsilon\}\}$ and

$$\begin{aligned} P' := & \{(q, A) \to (p, \phi(\alpha)) \mid ((q, A) \to (p, \alpha)) \in P\} \\ & \cup \{(q, \hat{a}) \to (q, a) \mid q \in F, a \in \Sigma \cup \{\varepsilon\}\}, \end{aligned}$$

where $\phi : (V \cup \Sigma)^* \to (V')^*$ is the mapping that replaces each occurrence of each terminal symbol a within a non-empty string by an occurrence of the variable $\hat{a}$, and that maps the empty string to the variable $\hat{\varepsilon}$.

Each derivation tree in G obviously corresponds to a derivation tree of G'. Observe that here it is important that we do not consider leftmost derivations, as a variable of the form $\hat{a}$ ($a \in \Sigma \cup \{\varepsilon\}$) can be rewritten into the terminal string a only under a final state. On the other hand, to each derivation tree of G', we can associate a derivation tree of G that yields the same terminal string by simply replacing each occurring symbol $\hat{a}$ by a and by forgetting about the applications of those rules of G' that rewrite these particular variables. It follows that $L(G) = L(G')$ and $L_{\mathsf{lt}}(G) = L_{\mathsf{lt}}(G')$ hold.

If G does not contain any ε-rules, then we do not need the symbol $\hat{\varepsilon}$ in G', and accordingly G' will then also be ε-free. □

5. Upper bounds for some subclasses of $\mathcal{L}_{\mathsf{lm}}(\mathsf{sACFG})$

In this section, we consider upper bounds for some subclasses of $\mathcal{L}_{\mathsf{lm}}(\mathsf{sACFG})$.

Theorem 5.1. *The language class* $\mathcal{L}(\textsf{right-lin-ACFG})$ *coincides with the class* REG *of regular languages*.

Proof. As each regular language is generated by a right-linear grammar, the inclusion from right to left is obvious. Conversely, let $L \subseteq \Sigma^*$ be the language that is generated by the right-linear ACFG $G = (V, U, \Sigma, P, S)$. We will show that there exists an alternating finite-state acceptor for the language L. As alternating finite-state acceptors only accept regular languages [2], this shows that L is regular.

First we transform the ACFG G as follows. For each universal variable $A \in U$ and for each production $A \to a_1 \dots a_k B$ from P, where $a_1, \dots, a_k \in \Sigma$, $k \geqslant 1$, and $B \in V$ or $B = \varepsilon$, we introduce k new existential variables $A_1, \dots, A_k$. Then we replace the production $A \to a_1 \dots a_k B$ by the following group of new productions:

$$A \to A_1,\ A_1 \to a_1 A_2,\ \dots,\ A_{k-1} \to a_{k-1} A_k,\ A_k \to a_k B.$$

Further, for each existential variable C and each production $C \to c_1 \dots c_k D$ from P, where $c_1, \dots, c_k \in \Sigma$, $k > 1$, and $D \in V$ or $D = \varepsilon$, we introduce new existential variables $C_1, \dots, C_{k-1}$, and we replace the production $C \to c_1 \dots c_k D$ by the following group of productions:

$$C \to c_1 C_1,\ C_1 \to c_2 C_2,\ \dots,\ C_{k-2} \to c_{k-1} C_{k-1},\ C_{k-1} \to c_k D.$$

Then all rules of the resulting right-linear ACFG $G' := (V', U, \Sigma, P', S)$ are of the form $A \to B$, $A \to aB$, $A \to a$ or $A \to \varepsilon$, where $A, B \in V'$ and $a \in \Sigma$. In addition, if A is universal, then all productions with left-hand side A are of the first or the fourth form. Thus, no universal derivation step directly generates a terminal symbol.

Now from the grammar G' we construct an alternating finite-state acceptor M by applying the standard construction. The states of M correspond to the variables of G'. In particular, the universal states of M correspond to the universal variables, and the existential states of M correspond to the existential variables. Further, the transitions of M correspond to the productions of G'. In addition, we introduce a new state F that serves as a final state, and that is entered by each transition corresponding to a production of the form $A \to a$ with $A \in V$ and $a \in \Sigma \cup \{\varepsilon\}$. The finite-state acceptor M will have ε-transitions, if G' has productions of the form $A \to B$ with $A, B \in V'$ or $A \to \varepsilon$.

From the properties of G' we see that the universal states of M only admit ε-transitions. Now let $w \in \Sigma^*$. It is easily seen that there exists a successful derivation tree for w in G' if and only if there is an accepting computation tree of M on input w. Thus, M accepts the language L, that is, L is indeed regular. $\square$

By Theorem 4.3(a) this has the following consequence.

Corollary 5.2. $\mathcal{L}(\textsf{right-lin-ACFG}) = \mathcal{L}(\textsf{right-lin-sACFG}) = \textsf{REG}$.

From Theorem 4.3(b) we know that $\mathcal{L}(\textsf{lin-ACFG})$ and $\mathcal{L}(\textsf{lin-sACFG})$ coincide. On the other hand Chen and Toda have shown that the closure of $\mathcal{L}(\textsf{lin-ACFG})$ under log-space reductions coincides with the complexity class P [4]. Thus, their result can be restated as follows.

Corollary 5.3. $\mathsf{LOG}(\mathcal{L}(\text{lin-ACFG})) = \mathsf{LOG}(\mathcal{L}(\text{lin-sACFG})) = \mathsf{P}$.

As $\mathsf{P} = \mathsf{ALOGSPACE}$ [2], Corollary 5.3 can be viewed as the counterpart (with respect to alternation) of the well-known result by Sudborough [18] that

$$\mathsf{NLOGSPACE} = \mathsf{LOG}(\mathcal{L}(\text{lin-CFG})),$$

where NLOGSPACE denotes the class of languages that are accepted by non-deterministic Turing machines within logarithmic space. Below we will repeatedly refer to the complexity class DLINSPACE, which is the class of languages that are accepted by deterministic Turing machines within linear space.

We now turn our attention to ε-free sACFGs. In fact, we first consider sACFGs that are in addition *unit-production-free*, that is, they do not contain any unit-productions.

Lemma 5.4. $\mathcal{L}_{\mathsf{lm}}(\varepsilon\text{-free-unit-production-free-sACFG}) \subseteq \mathsf{DLINSPACE}$.

Proof. Let $G = (Q, U, V, \Sigma, P, S, q_0, F)$ be an sACFG that is ε- and unit-production-free. Thus, each production $(p, A) \to (q, \alpha)$ of G satisfies $|\alpha| \geqslant 2$ or $\alpha \in \Sigma$. Hence, for each word $w \in L(G)$, if $|w| = n$, then each path from the root to a leaf in each G-derivation tree of w has length at most $2n - 1$, and so each G-derivation tree of w has height at most $2n - 1$.

Each node of a G-derivation tree is labelled with a pair of the form (p, α), where $p \in Q$ and $\alpha \in (V \cup \Sigma)^+$. If α does not contain any variables, then this node is a terminal leaf. Otherwise, it is either an existential or a universal node, depending on the type of the state p.

We say that a node is *successful* if it can be part of a leftmost G-derivation tree for w. Our goal is to verify whether the root is successful. When creating a leftmost G-derivation tree, we can distinguish between the following situations, depending on the type of the actual node:

(1) If it is a terminal node with label (p, u), then it is successful if and only if $u = w$ and $p \in F$.
(2) If it is an existential node with label $(p, uA\gamma)$, where $u \in \Sigma^*$, $A \in V$, and $\gamma \in (V \cup \Sigma)^*$, then it is successful if and only if there exists a production with left-hand side (p, A) such that the node is successful that is obtained from the actual node by applying this production.
(3) If it is a universal node with label $(p, uA\gamma)$, where $u \in \Sigma^*$, $A \in V$, and $\gamma \in (V \cup \Sigma)^*$, then it is successful if and only if all those nodes are successful that are obtained from the actual node by applying all the rules with left-hand side (p, A).

Further, in the latter two cases we can abort the search, if $|uA\gamma| > |w|$.

We now construct a linearly space-bounded deterministic Turing machine T that, given a word $w \in \Sigma^+$ as input, tries to construct a leftmost G-derivation tree for w in depth-first order by using all possible applications of rules in a systematic way. The above bound is used to limit the depth to which the search continues.

In order to realize this depth-first search, the Turing machine T has four tapes. The first of these tapes will contain the given input. It will not be changed in the course of the computation, as it will be used to check the labels of terminal nodes encountered during the search. Because of the intended linear space bound, T cannot possibly store the complete G-derivation tree for w. Instead it only stores the current sentential form together with information on the 'address' of the actual node. For this, the second tape will contain a description of the path in the currently created partial leftmost G-derivation tree that leads to the actual node. Observe that this node is uniquely described by the sequence of rules that have been applied on the path from the root of the tree to this particular node. Thus, from this information all ancestors of the current node can be recomputed on demand. The third tape will contain the sentential form of G that is the label of the actual node, and finally the fourth tape will be used as scratch paper for performing auxiliary calculations.

Let $w \in \Sigma^+$ be the given input, and let $n := 2 \cdot |w| - 1$. For each integer k, $1 \leqslant k \leqslant n$, the initial part of length k of a path in the G-derivation tree of w (and therewith a node in this tree) can be described uniquely by a sequence of the form

$$(Q_1 : i_1), (Q_2 : i_2), \ldots, (Q_k : i_k),$$

where i_j is the serial number (with respect to an arbitrarily chosen linear ordering of the productions of G) of the production $(p_{i_j}, A_{i_j}) \to (q_{i_j}, \alpha_{i_j})$ that is used in step j, and Q_j is the quantifier $\forall$, if the state p_{i_j} is universal, and it is the quantifier $\exists$, if the state p_{i_j} is existential, $1 \leqslant j \leqslant k$. Further, two productions i and j are said to be *compatible* if they apply to the same configurations in leftmost derivation mode, that is, $p_i = p_j$ and $A_i = A_j$ hold simultaneously.

The Turing machine T will use its second tape as a pushdown store that stores the sequence of pairs $(Q_1 : i_1), (Q_2 : i_2), \ldots, (Q_k : i_k)$ that describe the path in the leftmost G-derivation tree that leads to the actual node. Here i_1 is the index of the rule that was applied to the start symbol on the actual path, and i_k is the index of the rule through which the actual sentential form (q_{i_k}, α) has been obtained. Accordingly, the pair $(Q_k : i_k)$ will be on the top of the stack. Observe that the quantifier Q_k corresponds to the type (existential or universal) of the parent node of the actual node, and as such it gives information on how to proceed once it has been determined whether or not the actual node is successful.

The computation of the Turing machine T is now described by the following procedure. Within this procedure **accept** is equated with the Boolean value *true*, and **reject** stands for the value *false*. Further, the stack operations **push**, **top**, and **pop** used refer always to tape 2, and the items that are pushed, retrieved or popped from the stack, respectively, are always pairs of the form $(Q : i)$, where $Q \in \{\exists, \forall\}$ and $i \in \{1, \ldots, |P|\}$. Further, a boolean variable *result* is used to compute (and store) the result of the computation. For each node it will be set to the value **accept** if this node is successful. The computation halts once it becomes clear whether the root of the tree, which is labelled with the pair (q_0, S), is successful. In the affirmative this happens only when the stack (that is, tape 2) has become empty, and the root has been recognized as a successful node, while in the negative this might be realized much sooner. To manage the depth-first search another boolean variable named *first* is being used. For each node encountered during the search process this variable has value *true* if the actual node is just being encountered for the first time, and it gets the value *false* once the actual node is being revisited.

```
boolean function simulate($w$); (* $w \in \Sigma^+$ is the given input. *)
begin boolean result, first;
    stack tape2 $\leftarrow$ empty; (* tape2 is used as a stack *)
    (* INITIALISATION *)
    choose $i$ to be the index of the first production of $G$ that is applicable to the
    initial sentential form $(q_0, S)$;
    if $q_0$ is universal then $Q \leftarrow \forall$ else $Q \leftarrow \exists$;
    push($Q : i$);
    first $\leftarrow$ true;
    result $\leftarrow$ reject;
    while not empty tape2 do (* SIMULATION *)
    begin if first = true then
        begin
            on tape3, simulate the leftmost $G$-derivation that is described by
            the current content of tape2, starting from the initial sentential form
            $(q_0, S)$, and let $(q, \alpha)$ be the resulting sentential form;
            first $\leftarrow$ false;
            if $\alpha \in \Sigma^*$ then (* $\alpha$ is a terminal string *)
                if $\alpha = w$ and $q \in F$ then result $\leftarrow$ accept
                else result $\leftarrow$ reject
            else (* $\alpha$ is NOT a terminal string *)
                if $\exists i$ : production $i$ is applicable to $(q, \alpha)$
                    and $|\alpha| \leqslant |w|$ then
                      begin
                          choose $i$ to be the smallest index of a production of $G$
                          that is applicable to the sentential form $(q, \alpha)$, and let
                          $Q$ be the quantifier corresponding to the state $q$;
                          push($Q : i$);
                          first $\leftarrow$ true;
                      end
                else result $\leftarrow$ reject;
          end
          else (* first = false *)
          begin $(Q : i) \leftarrow$ top; (* Read the topmost element from the stack *)
              pop;     (* $(Q : i)$ is removed from the stack *)
              if $((Q = \exists$ and result = reject$)$ or $(Q = \forall$ and result = accept$))$
                  and $\exists j > i$ : productions $i$ and $j$ are compatible then
              begin choose the smallest $j > i$ with this property;
                  push $(Q : j)$;
                  first $\leftarrow$ true;
              end
          end
     end;
     return result;
end.
```

Given a string $w \in \Sigma^+$ as input, the Turing machine T performs a traversal of the leftmost derivation tree of G for w in depth-first order. Each time a sentential form is generated, which is indicated by pushing a new pair $(Q : i)$ onto tape 2, the Boolean variable *first* is set to *true*. In the next traversal of the while-loop T tries to extend the derivation, if the sentential form is not terminal already, or it compares the result of this branch of the derivation tree to the given input w, if the actual sentential form is terminal. Observe that a terminal sentential form ends the actual branch of the derivation tree, and that this branch is successful, if the string generated coincides with the input string w and if the actual state is final, and that this branch fails, if the string generated differs from w or if the actual state is not final. When T reaches the same pair $(Q : i)$ again during the backtracking process, which is indicated by the truth value *false* of the variable *first*, then this step can be removed if the result of the corresponding subtree is already known, which is the case if Q is existential and the current result is **accept**, or if Q is universal and the current result is **reject**. Otherwise the corresponding brother configurations have to be examined in order to determine the result of the actual branch correctly. To do so the step encoded by $(Q : i)$ is replaced by its next older brother, if such a brother exists. It follows that the language $L(T)$ accepted by T coincides with the language $L_{\mathsf{lm}}(G)$.

As T simulates leftmost G-derivations for the given input w, the content of tape 2 is bounded in length by the number $c \cdot n$ for some constant c. As tape 3 always contains a sentential form of G, it is also bounded in length by $|w|$. Hence, T can indeed be realized in a linearly space-bounded manner. □

Next we want to generalize the above result to ε-free sACFGs. As an intermediate step we introduce the following notion. An sACFG G is said to have *bounded unit-productions* if there exists a constant $c \geqslant 1$ such that, for each string $w \in L_{\mathsf{lm}}(G)$, there exists a leftmost G-derivation tree for w such that the number of applications of unit-productions on each path of this derivation tree is bounded from above by the number $c \cdot |w|$.

If G is ε-free and has bounded unit-productions with respect to the constant c, then for each string $w \in L_{\mathsf{lm}}(G)$, there exists a leftmost G-derivation tree for w such that on each path of G, there are at most $2 \cdot |w| - 1$ many applications of non-unit-productions and $c \cdot |w|$ many applications of unit-productions. Thus, this leftmost G-derivation tree for w has height at most $(2 + c) \cdot |w| - 1$. Hence, the arguments in the proof of Lemma 5.4 apply to G. This yields the following consequence.

Corollary 5.5. $\mathcal{L}_{\mathsf{lm}}(\varepsilon\text{-free-bounded-unit-production-sACFG}) \subseteq \mathsf{DLINSPACE}$.

Based on the notion of bounded unit-productions, we can now establish the following interesting result.

Theorem 5.6. $\mathcal{L}_{\mathsf{lm}}(\varepsilon\text{-free-sACFG}) \subseteq \mathsf{DLINSPACE}$.

Proof. It suffices by the previous corollary to show that an ε-free sACFG has necessarily bounded unit-productions. So let $G = (Q, U, V, \Sigma, P, S, q_0, F)$ be an ε-free sACFG. We choose the constant $c := 2 \cdot |Q| \cdot |V|$, and claim that G has bounded unit-productions with respect to this constant c.

Let $w \in L_{\mathsf{lm}}(G)$, $|w| = n$, and let T be a G-derivation tree for w corresponding to a leftmost derivation. Assume that this tree contains a path p on which there are more than $c \cdot n = 2 \cdot |Q| \cdot |V| \cdot n$ applications of unit-productions. As G is ε-free, p contains at most $2n - 1$ applications of non-unit-productions. Hence, p can be partitioned into (at most) $2n - 1$ subpaths $p_1, \ldots, p_{2n-1}$ such that

- each subpath p_i contains only applications of unit-productions, and
- subpath p_i is connected to subpath p_{i+1} $(1 \leqslant i < 2n - 2)$ by an application of a non-unit-production, and
- subpath p_{2n-1} ends with a non-unit-production that generates a terminal sentential form.

Thus, there is an index i such that subpath p_i contains more than $|Q| \cdot |V|$ applications of unit-productions, that is, this subpath can be decomposed as

$$p_i = p_i^{(1)} \to (q, uA\alpha) \to (r, uB\alpha) \to p_i^{(2)} \to (q, uA\alpha) \to (r, uB\alpha) \to p_i^{(3)},$$

where $(q, A) \to (r, B)$ is the first unit-production that is repeated on p_i. By deleting the subpath $(r, uB\alpha) \to p_i^{(2)} \to (q, uA\alpha)$ we obtain an equivalent derivation tree. This is true even if this subpath contains universal steps, in which case we also delete all the subtrees generated along this subpath. By using such replacements repeatedly we eventually obtain a leftmost G-derivation tree for w that does not contain more than $c \cdot n$ unit-productions on any path. Thus, we see that G has indeed bounded unit-productions. $\square$

This result together with Theorem 4.3(c) and the result of Chen and Toda on ε-free ACFGs [4] has the following consequence.

Corollary 5.7. $\mathsf{LOG}(\mathcal{L}_{\mathsf{lm}}(\varepsilon\text{-free-ACFG})) = \mathsf{LOG}(\mathcal{L}_{\mathsf{lm}}(\varepsilon\text{-free-sACFG})) = \mathsf{PSPACE}$.

6. Characterizing language classes by automata

The original purpose for introducing ACFGs was to give a grammatical characterization for the language class $\mathcal{L}(\mathsf{APDA})$ [14]. Such a characterization will be derived in this section in terms of sACFGs with leftmost derivations.

Definition 6.1. An *alternating pushdown automaton*, APDA for short, M is given by an 8-tuple $(Q, U, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, where Q is a finite set of states, $U \subseteq Q$ is a set of universal states, Σ is an input alphabet, Γ is a pushdown alphabet, $q_0 \in Q$ is the initial state, $Z_0 \in \Gamma$ is the bottom marker for the pushdown store, $F \subseteq Q$ is a set of accepting (or final) states, and $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to \mathcal{P}_{\mathsf{fin}}(Q \times \Gamma^*)$ is a transition function.

A *configuration* of M is described by a triple (q, u, α), where $q \in Q$ is the current state, $u \in \Sigma^*$ is the remaining part of the input with the input head scanning the first symbol of u, and α is the current content of the pushdown store with the first letter of α being the symbol on the top of the pushdown store and the last letter of α being the symbol on the bottom of the pushdown store. As usual the *initial configuration* for an input $w \in \Sigma^*$ is the triple (q_0, w, Z_0), and a *final configuration* has the form (q, ε, α) with $q \in F$ and $\alpha \in \Gamma^*$.

An input $w \in \Sigma^*$ is *accepted* by M, if there is a successful computation tree of M on that input, that is, there is a finite tree the nodes of which are labelled by configurations of M such that the following conditions are satisfied:

(1) the root is labelled with the initial configuration (q_0, w, Z_0);
(2) each leaf is labelled with a final configuration;
(3) if a non-leaf is labelled by $(q, au, Z\alpha)$, where $q \in Q \setminus U$, $a \in \Sigma \cup \{\varepsilon\}$, $Z \in \Gamma$, then it has a single successor that is labelled by $(p, u, \beta\alpha)$ for some $(p, \beta) \in \delta(q, a, Z)$;
(4) if a non-leaf is labelled by $(q, au, Z\alpha)$, where $q \in U$, $a \in \Sigma \cup \{\varepsilon\}$, $Z \in \Gamma$, and if $\delta(q, a, Z) = \{(p_1, \beta_1), \dots, (p_m, \beta_m)\}$, then it has m successor nodes labelled by $(p_1, u, \beta_1\alpha), \dots, (p_m, u, \beta_m\alpha)$, respectively.

Instead of accepting by final state as defined above, an APDA can also *accept by empty pushdown*, which means that each leaf of a successful computation tree is labelled with a configuration of the form $(q, \varepsilon, \varepsilon)$, where q is any state from Q.

As mentioned in the introduction, it has been shown in [11] that $\mathcal{L}(\mathsf{APDA}) = \mathsf{ETIME}$. To derive the intended grammatical characterization the following two technical lemmas are needed.

Lemma 6.2. *For each* APDA M, *there exists an* APDA M' *such that* $L(M) = L(M')$, *and all final states of* M' *are existential.*

Proof. Let $M = (Q, U, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be an APDA. If $U \cap F = \emptyset$, then already all final states of M are existential, and there is nothing to do.

Assume that $U \cap F \neq \emptyset$. Then we construct an APDA $M' := (Q', U', \Sigma, \Gamma', \delta', q_0', Z_0', F')$ from M by creating two new states q_{f} and q_{u} for each state $q \in U \cap F$. The state q_{f} will be a final state that is existential, and the state q_{u} will be a non-final state that is universal. Further, the original state q will be turned into a non-final existential state. In addition, a new initial state q_0' and a new bottom marker Z_0' for the pushdown store are introduced. The main idea of the construction of M' is the following. Whenever M' is in state q, it must choose non-deterministically whether the role of q as a universal state or its role as a final state is needed. This is done by executing an ε-step that takes M' into state q_{u} or q_{f}, respectively. Accordingly, M' is defined as follows:

- $Q' := Q \cup \{ q_{\mathrm{f}}, q_{\mathrm{u}} \mid q \in U \cap F \} \cup \{q_0'\}$,
- $U' := (U \setminus F) \cup \{ q_{\mathrm{u}} \mid q \in U \cap F \}$,
- $F' := (F \setminus U) \cup \{ q_{\mathrm{f}} \mid q \in U \cap F \}$,
- $\Gamma' := \Gamma \cup \{Z_0'\}$,
- $\delta'(q_0', \varepsilon, Z_0') := (q_0, Z_0 Z_0')$,
- $\delta'(q, a, Z) := \delta(q, a, Z)$ if $q \in Q \setminus (U \cap F)$, $a \in \Sigma \cup \{\varepsilon\}$, $Z \in \Gamma$,
- $\delta'(q, \varepsilon, Z) := \{(q_{\mathrm{f}}, Z), (q_{\mathrm{u}}, Z)\}$ if $q \in U \cap F$, $Z \in \Gamma'$,
- $\delta'(q_{\mathrm{u}}, a, Z) := \delta(q, a, Z)$ if $q \in U \cap F$, $a \in \Sigma \cup \{\varepsilon\}$, $Z \in \Gamma$.

Each time a state $q \in U \cap F$ is encountered in the course of a computation of M', either the universal state q_{u} is entered, thus continuing the simulation of M, or the final state q_{f} is entered, which ends the simulation. The new bottom marker Z_0' for the pushdown store of M' is introduced in order to enable M' to enter a final state, even if in the corresponding computation of M a universal final state p is entered in a move that empties the pushdown store. It is now straightforward to verify that M' accepts the same language as M. $\square$

The next lemma establishes a kind of normal form for APDAs.

Lemma 6.3. *If a language L is accepted by an* APDA *by final state, then L is also accepted by an* APDA *N by empty pushdown, where, in addition, each universal transition of N is an ε-transition, that is, N satisfies the following condition*:
$(*)$ *if p is a universal state of N, and $\delta(p, a, Z) \neq \emptyset$, then $a = \varepsilon$.*

Proof. Let M be an APDA accepting the language L. By the previous lemma we can assume without loss of generality that no final state of M is universal. Now by applying the standard technique for pushdown automata (see, e.g., [7, Theorem 5.1]), we can convert M into an equivalent APDA $M_1 := (Q, U, \Sigma, \Gamma, \delta, q_0, Z_0)$ which accepts by empty pushdown. Observe that here it is essential that no final state of M is universal, as by this construction ε-transitions that empty the pushdown store are introduced for all final states.

Next, we introduce a new existential state $\hat{p}$ for each universal state p of M_1 and replace each occurrence of p in the right-hand side of any transition by $\hat{p}$. Further, we introduce new universal states $[p, a]$ ($a \in \Sigma \cup \{\varepsilon\}$), and introduce the following additional transitions for all $a \in \Sigma \cup \{\varepsilon\}$ and $Z \in \Gamma$:

$$\begin{aligned} \delta_N(\hat{p}, a, Z) \quad &:= \{([p, a], Z)\}, \\ \delta_N([p, a], \varepsilon, Z) &:= \delta(p, a, Z). \end{aligned}$$

Let N denote the resulting APDA. Clearly N satisfies the condition $(*)$. To see that N is equivalent to M_1 we note that a universal transition $\delta(p, a, Z)$ of M_1 is simulated by N by first entering the existential state $\hat{p}$, then choosing non-deterministically the transition $(\hat{p}, a, Z) \mapsto ([p, a], Z)$, and then executing the universal ε-transition $\delta_N([p, a], \varepsilon, Z)$. Since M_1 accepts by empty pushdown, and since N does not empty its pushdown unless M_1 does so, N and M_1 accept the same language by empty pushdown. $\square$

Based on this technical result we can now establish the following characterization.

Theorem 6.4. $\mathcal{L}_{\mathsf{lm}}(\mathsf{sACFG}) = \mathcal{L}(\mathsf{APDA})$.

Proof. For the inclusion $\mathcal{L}_{\mathsf{lm}}(\mathsf{sACFG}) \subseteq \mathcal{L}(\mathsf{APDA})$, let $G = (Q, U, V, \Sigma, P, S, q_0, F)$ be an sACFG. By Lemma 4.7 we may assume that G is in weak Chomsky normal form and that $F = Q$ holds. As APDAs are allowed to make ε-transitions, that is, the head on the input tape may remain stationary during certain transitions, the standard technique (see, e.g., [7]) can be used to construct an APDA M from G such that M accepts the language $L_{\mathsf{lm}}(G)$. Observe the fact that, whenever several different productions of G are applicable to the same sentential form, all the possible productions are unit-productions, which means that all existential or universal transitions of M that entail a non-deterministic choice are in fact ε-transitions.

The proof for the converse inclusion $\mathcal{L}(\mathsf{APDA}) \subseteq \mathcal{L}_{\mathsf{lm}}(\mathsf{sACFG})$ is actually simpler than the one for the non-alternating case, as sACFGs can use their own states to simulate the state transitions of APDAs. Let $M = (Q, U, \Sigma, \Gamma, \delta, q_0, Z_0, Q)$ be an APDA. By Lemma 6.3 we can assume that M accepts by empty pushdown, and that all universal transitions of M are ε-transitions. Assume that M is in a configuration $(p, ay, Z\alpha)$, where p is the current state, ay ($a \in \Sigma \cup \{\varepsilon\}$, $y \in \Sigma^*$) is the remaining suffix (the unconsumed part) of the input, and $Z\alpha$ ($Z \in \Gamma,\ \alpha \in \Gamma^*$) is the current content of the pushdown store. Further, let x be the prefix of the input that has already been consumed by M before reaching the current configuration. Then we construct an sACFG G which has the same universal and existential states as M and that behaves as follows. Corresponding to the current configuration $(p, ay, Z\alpha)$ of M, G will derive the sentential form $(p, xZ\alpha)$. If $\delta(p, a, Z)$ contains (q, β), then G has the production $(p, Z) \to (q, a\beta)$. Thus, $(q, xa\beta\alpha)$ is directly derived from $(p, xZ\alpha)$ by rewriting the leftmost variable Z into $xZ\alpha$. Observe that by Lemma 6.3, if p is universal, then for any $Z \in \Gamma$, the universal productions of G with left-hand side (p, Z) correspond one to one to the universal transitions $\delta(p, \varepsilon, Z)$ of M. It follows easily that the languages $L_{\mathrm{lm}}(G)$ and $L(M)$ coincide. □

By Lemma 4.1 and the result on $\mathcal{L}(\mathsf{APDA})$ from [11] the above characterization yields the following consequence.

Corollary 6.5. $\mathcal{L}_{\mathrm{lm}}(\mathsf{ACFG}) \subseteq \mathcal{L}_{\mathrm{lm}}(\mathsf{sACFG}) = \mathcal{L}(\mathsf{APDA}) = \mathsf{ETIME}$.

7. Comparisons of derivation strategies

So far we have mostly considered leftmost derivations for ACFGs and sACFGs, but of course there are many strategies to select an occurrence of a variable in a sentential form to apply a production. Here, we compare the expressive power of the ACFGs and the sACFGs with respect to the leftmost, the leftish and the unrestricted derivation modes. First we turn to the *leftish* derivation strategy, which differs significantly in expressive power from the leftmost strategy, as we will see below.

Theorem 7.1. (a) $\mathcal{L}_{\mathrm{lt}}(\varepsilon\text{-free-sACFG}) \subseteq \mathcal{L}(\mathsf{APDA})$.
(b) $\mathcal{L}_{\mathrm{lt}}(\mathsf{sACFG}) = \mathsf{RE}$.

Proof. (a) It is known (see [2]) that $\mathcal{L}(\mathsf{APDA}) = \mathcal{L}(\mathsf{ALBA})$, where ALBA stands for *alternating linear bounded automata*. Hence, in order to prove that $\mathcal{L}_{\mathrm{lt}}(\varepsilon\text{-free-sACFG}) \subseteq \mathcal{L}(\mathsf{APDA})$, it suffices to present an ALBA M which accepts the language generated by a given ε-free sACFG G in leftish mode. By Lemma 4.8 we can assume that all states of G are final.

The input tape of M is divided into two tracks. Throughout the computation M retains the input string on the first track, and it utilizes the second track to simulate leftish derivations of G. Naturally, M holds the actual state of G in its finite control.

M simulates each G-derivation step by a sequence of moves, called a *cycle*. At the beginning of a cycle, the read/write head of M is at the left end of the input tape. M begins the cycle by searching the sentential form on the second track from left to right for the leftmost variable A to which a production of G can be applied. Once M finds this variable, it applies an appropriate production $(p, A) \to (q, \alpha)$ of G, thereby replacing the variable A by the string α. This step is existential or universal, depending on whether the state p of G is existential or universal, respectively. As α may be of length larger than one, M may have to shift the suffix of the inscription of the second track to the right in order to have enough space for storing α. Further, M stores the state q in its finite control. M ends the current cycle by moving its read/write head back to the left end of the input tape.

If M reaches the right end of the input tape without finding any variable on the second track to which a production of G applies, then M either accepts (if there is no variable at all in the sentential form and if the contents of the two tracks coincide) or rejects (otherwise). It easily follows that $L(M) = L_{\mathsf{lt}}(G)$.

As the sACFG G does not contain any ε-productions, each configuration of M uses only the space provided by the given input. Thus, M is linearly space-bounded, that is, it is indeed an ALBA. Hence, $\mathcal{L}_{\mathsf{lt}}(\varepsilon\text{-free-sACFG}) \subseteq \mathcal{L}(\mathsf{ALBA}) = \mathcal{L}(\mathsf{APDA})$ follows.

(b) As each ECFG (see Section 2) can be regarded as an sACFG with only existential states, it follows from Proposition 2.2(b) that $\mathcal{L}_{\mathsf{lt}}(\mathsf{sACFG}) \supseteq \mathcal{L}_{\mathsf{lt}}(\mathsf{ECFG}) = \mathsf{RE}$. The converse inclusion is obvious. $\square$

Together with Theorem 6.4 this yields the following inclusion.

Corollary 7.2. $\mathcal{L}_{\mathsf{lt}}(\varepsilon\text{-free-sACFG}) \subseteq \mathcal{L}_{\mathsf{lm}}(\mathsf{sACFG})$.

Open Problem 3. Does the converse of the inclusion above hold, too?

Concerning the expressive power of the various derivation modes for the sACFGs, we have the following inclusions.

Theorem 7.3.
(a) $\mathcal{L}_{\mathsf{lm}}(\varepsilon\text{-free-sACFG}) \subseteq \mathcal{L}_{\mathsf{lt}}(\varepsilon\text{-free-sACFG})$ *and*
$\mathcal{L}(\varepsilon\text{-free-sACFG}) \subseteq \mathcal{L}_{\mathsf{lt}}(\varepsilon\text{-free-sACFG})$.
(b) $\mathcal{L}_{\mathsf{lm}}(\mathsf{sACFG} \subseteq \mathcal{L}_{\mathsf{lt}}(\mathsf{sACFG})$ *and* $\mathcal{L}(\mathsf{sACFG}) \subseteq \mathcal{L}_{\mathsf{lt}}(\mathsf{sACFG})$.

Proof. The inclusions in (b) are immediate consequences of Theorem 7.1(b). It remains to verify the inclusions in (a). The first of them is a consequence of Proposition 2.2 (a) and Theorem 5.6, as each ECFG is also an sACFG.

It remains to prove the inclusion $\mathcal{L}(\varepsilon\text{-free-sACFG}) \subseteq \mathcal{L}_{\mathsf{lt}}(\varepsilon\text{-free-sACFG})$. For an arbitrary ε-free sACFG $G = (Q, U, V, \Sigma, P, S, q_0, F)$, we will construct an ε-free sACFG G' that in leftish mode simulates the unrestricted derivations of G. The idea of the simulation is as follows.

For each variable X, a new variable $\bar{X}$ is introduced, and for each variable or terminal A, a new variable A' is introduced. Further, for each state q, two new existential states E_q and $\bar{E}_q$ are introduced. When $(p, \beta X \gamma)$ is the current sentential form of G, where $p \in Q$, $\beta, \gamma \in (V \cup \Sigma)^*$, and $X \in V$, and the displayed occurrence of the variable X is the one that is to be rewritten next, then this derivation step is simulated by a sequence of derivation steps of G'. This sequence starts with the sentential form $(E_p, \beta X \gamma)$. As G' has to work in leftish mode, we need some preparatory steps that change the current sentential form in such a way that the displayed occurrence of the variable X becomes the leftmost to which a production of G' is applicable. For this each variable Y that occurs within the prefix β is replaced by an occurrence of the new variable $\bar{Y}$. After that has been done, the state E_p changes into the state p, and then the actual derivation step of G is simulated by a leftish derivation step of G'. However, to complete the simulation of the G-derivation step, the variables of the form $\bar{Y}$ contained in β must again be replaced by the original variables. This is done using the states of the form $\bar{E}_q$ and the new variables of the form A'.

We now describe G' in detail. We take $G' := (Q', U, V', \Sigma, P', S, q_0, F')$, where

$$\begin{aligned}
Q' &:= Q \cup \{ E_p, \bar{E}_p \mid p \in Q \},\\
V' &:= V \cup \{ \bar{X} \mid X \in V \} \cup \{ A' \mid A \in V \cup \Sigma \},\\
F' &:= \{ E_p \mid p \in F \},
\end{aligned}$$

and let P' consist of the following rules:

$$\begin{array}{llll}
(1) & (E_p, X) \to (E_p, \bar{X}) & \text{for each } p \in Q \text{ and } X \in V,\\
(2) & (E_p, X) \to (p, X) & \text{for each } p \in Q \text{ and } X \in V,\\
(3) & (p, X) \to (\bar{E}_q, A'\alpha) & \text{if } ((p, X) \to (q, A\alpha)) \in P,\\
(4) & (\bar{E}_p, \bar{X}) \to (\bar{E}_p, X) & \text{for each } p \in Q \text{ and } X \in V,\\
(5) & (\bar{E}_p, A') \to (E_p, A) & \text{for each } p \in Q \text{ and } A \in V \cup \Sigma.
\end{array}$$

Now if $(p, \beta X\gamma) \Rightarrow_G (q, \beta A\alpha\gamma)$ is an arbitrary derivation step in G, where $p, q \in Q$, $X \in V$, $A \in V \cup \Sigma$, and $\alpha, \beta, \gamma \in (V \cup \Sigma)^*$, then

$$\begin{array}{rll}
(E_p, \beta X\gamma) & \Rightarrow^*_{(1)} (E_p, \bar{\beta} X\gamma) & \text{(Replace each variable } Y \text{ in } \beta \text{ by } \bar{Y})\\
& \Rightarrow_{(2)} (p, \bar{\beta} X\gamma) & \text{(Change the state } E_p \text{ into } p)\\
& \Rightarrow_{(3)} (\bar{E}_q, \bar{\beta} A'\alpha\gamma) & \text{(Simulate the } G\text{-derivation step)}\\
& \Rightarrow^*_{(4)} (\bar{E}_q, \beta A'\alpha\gamma) & \text{(Replace each variable } \bar{Y} \text{ in } \bar{\beta} \text{ by } Y)\\
& \Rightarrow_{(5)} (E_q, \beta A\alpha\gamma) & \text{(Replace the variable } A' \text{ by } A).
\end{array}$$

is the corresponding simulation in G', where $\bar{\beta}$ denotes the string obtained from β by replacing each variable Y by its barred variant $\bar{Y}$. Note that in the productions of group (3) it is essential that P does not contain any ε-productions. It is now easily verified that $L_{\mathrm{lt}}(G') = L(G)$. $\square$

Corollary 7.4.

(a) $\mathcal{L}_{\mathrm{lm}}(\varepsilon\text{-free-ACFG}) = \mathcal{L}_{\mathrm{lt}}(\varepsilon\text{-free-ACFG}) \subseteq \mathcal{L}_{\mathrm{lt}}(\varepsilon\text{-free-sACFG})$.

(b) $\mathcal{L}_{\mathrm{lm}}(\mathrm{ACFG}) = \mathcal{L}_{\mathrm{lt}}(\mathrm{ACFG}) \subsetneq \mathcal{L}_{\mathrm{lt}}(\mathrm{sACFG})$.

Proof. The inclusions follow from Lemma 4.1. Further, we know from Corollary 6.5 and Theorem 7.1(b) that

$$\mathcal{L}_{\mathrm{lm}}(\mathrm{sACFG}) = \mathcal{L}(\mathrm{APDA}) = \mathrm{ETIME} \subsetneq \mathrm{RE} = \mathcal{L}_{\mathrm{lt}}(\mathrm{sACFG}). \qquad \square$$

Unfortunately, many questions concerning the inclusions between the language classes considered so far remain open. In particular we have the following open question.

Open Problem 4. Does $\mathcal{L}_{\mathsf{lm}}(X) \subseteq \mathcal{L}(X)$ or its converse hold, where X is any of the classes (ε-free-) (s)ACFG?

Observe that

$$\mathcal{L}(\varepsilon\text{-free-sACFG}) \subseteq \mathcal{L}_{\mathsf{lt}}(\varepsilon\text{-free-sACFG}) \subseteq \mathcal{L}_{\mathsf{lm}}(\text{sACFG})$$

by Corollary 7.2 and Theorem 7.3(a), while

$$\mathcal{L}_{\mathsf{lm}}(\text{ECFG}) \subsetneq \mathcal{L}(\varepsilon\text{-free-ECFG}) \subsetneq \mathcal{L}_{\mathsf{lt}}(\varepsilon\text{-free-ECFG})$$

by Proposition 2.2, which shows that various inclusion results do not carry over from ECFGs to sACFGs.

In the remainder of this section, we will illustrate the differences between the various derivation modes by considering some examples of (s)ACFGs. Analogously to Section 3 we denote by $L_{\mathsf{rm}}(G, w)$ and $L_{\mathsf{lt}}(G, w)$ the languages that are generated by the grammar G from the initial string w with respect to rightmost and leftish derivations, respectively. If arbitrary derivations are used, then the generated language is denoted by $L(G, w)$. Obviously, for each sACFG G and each string w, we have

$$L_{\mathsf{lm}}(G, w) \subseteq L(G, w), \quad L_{\mathsf{rm}}(G, w) \subseteq L(G, w), \quad \text{and} \quad L_{\mathsf{lt}}(G, w) \subseteq L(G, w).$$

The following example simultaneously separates $L_{\mathsf{lm}}(G)$, $L_{\mathsf{rm}}(G)$, $L_{\mathsf{lt}}(G)$, and $L(G)$.

Example 7.5. Let $G_1 := (\{\exists, \forall\}, \{\forall\}, \{S, A, B, C\}, \{a\}, P, S, \exists, \{\exists, \forall\})$ be the sACFG with the following productions:

$$\begin{array}{lll}
(\exists, S) \to (\exists, ABC), & (\exists, A) \to (\exists, \varepsilon), & (\exists, C) \to (\exists, \varepsilon), \\
(\exists, S) \to (\forall, ABC), & (\exists, B) \to (\forall, \varepsilon), & (\forall, B) \to (\exists, a), \\
(\exists, A) \to (\forall, a), & (\exists, C) \to (\exists, a), & (\forall, B) \to (\exists, \varepsilon).
\end{array}$$

The possible leftmost, rightmost, and leftish derivation trees are shown in Fig. 4, where dotted lines represent possible choices (that is, only one of them is to be chosen), and solid lines represent universal branches (that is, all the branches must be chosen at each such node). As, in addition, there is an unrestricted derivation

$$(\exists, S) \Rightarrow (\exists, ABC) \Rightarrow (\exists, BC) \Rightarrow (\exists, B) \Rightarrow (\forall, \varepsilon),$$

we see that the languages generated by G_1 in the leftmost, rightmost, leftish, and unrestricted derivation modes are $L_{\mathsf{lm}}(G_1) = \{aa\}$, $L_{\mathsf{rm}}(G_1) = \emptyset$, $L_{\mathsf{lt}}(G_1) = \{a, aa\}$, and $L(G_1) = \{\varepsilon, a, aa\}$, respectively. $\square$

Further, for each ACFG G and each string w, we have

$$L_{\mathsf{lm}}(G, w) \subseteq L(G, w) \quad \text{and} \quad L_{\mathsf{rm}}(G, w) \subseteq L(G, w).$$

The following example simultaneously separates $L_{\mathsf{lm}}(G)$, $L_{\mathsf{rm}}(G)$, and $L(G)$.

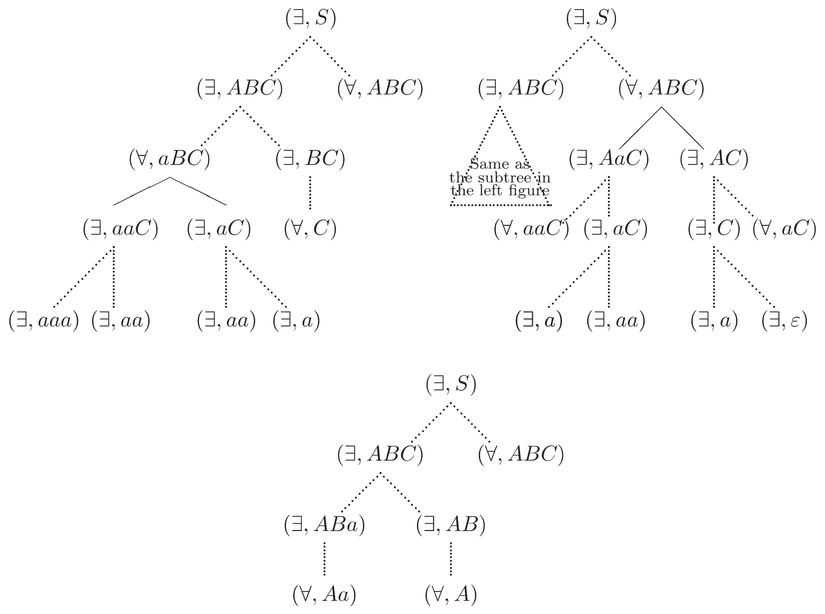


Fig. 4. Leftmost (top left), leftish (top right), and rightmost (bottom) derivation trees.

Example 7.6. Consider the ACFG

$$G_2 := (\{S, A_1, A_2, A_3, E_1, E_2, E_3\}, \{A_1, A_2, A_3\}, \{a, a', b, b'\}, P, S),$$

where P contains the following productions:

$$\begin{array}{lllll} S \to A_1E_2E_3, & E_1 \to \varepsilon, & E_1 \to a, & A_1 \to a', & A_1 \to a'a, \\ S \to E_1A_2E_3, & E_2 \to a, & E_2 \to b, & A_2 \to a, & A_2 \to b, \\ S \to E_1E_2A_3, & E_3 \to \varepsilon, & E_3 \to b, & A_3 \to b', & A_3 \to bb'. \end{array}$$

Then, for each derivation mode m,

$$L_{\mathsf{m}}(G_2) = L_{\mathsf{m}}(G_2, A_1E_2E_3) \cup L_{\mathsf{m}}(G_2, E_1A_2E_3) \cup L_{\mathsf{m}}(G_2, E_1E_2A_3).$$

(1) $L(G_2, A_1E_2E_3) = \{a'ab\}$, and this string is derivable only by first expanding A_1, since

$$L(G_2, a'E_2E_3) \cap L(G_2, a'aE_2E_3) = \{a'ab\},$$
$$L(G_2, A_1aE_3) \cup L(G_2, A_1bE_3) = \emptyset,$$
$$L(G_2, A_1E_2b) \cup L(G_2, A_1E_2) = \emptyset.$$

(2) $L(G_2, E_1A_2E_3) = \{ab\}$, and this string is derivable only by first expanding A_2, since

$$L(G_2, E_1aE_3) \cap L(G_2, E_1bE_3) = \{ab\},$$
$$L(G_2, A_2E_3) \cup L(G_2, aA_2E_3) = \emptyset,$$
$$L(G_2, E_1A_2b) \cup L(G_2, E_1A_2) = \emptyset.$$

(3) $L(G_2, E_1E_2A_3) = \{abb'\}$, and this string is derivable only by first expanding A_3, since

$$\begin{aligned}&L(G_2, E_1E_2bb') \cap L(G_2, E_1E_2b') = \{abb'\},\\ &L(G_2, E_2A_3) \cup L(G_2, aE_2A_3) = \emptyset,\\ &L(G_2, E_1aA_3) \cup L(G_2, E_1bA_3) = \emptyset.\end{aligned}$$

Thus, we see that $L(G_2) = \{a'ab, ab, abb'\}$, $L_{\mathsf{lm}}(G_2) = \{a'ab\}$, and $L_{\mathsf{rm}}(G_2) = \{abb'\}$.

8. An undecidability result

Let G be an (s)ACFG, and let m be any of the leftmost, rightmost or leftish derivation modes. Then $L_{\mathsf{m}}(G) \subseteq L(G)$, but in general the converse inclusion does not hold. Accordingly, we are interested in the following decision problem:

INSTANCE : An (s)ACFG G.
QUESTION : Is $L(G) = L_{\mathsf{m}}(G)$?

For this problem we have the following undecidability result.

Theorem 8.1. *Let* m *denote one of the derivation modes leftmost, rightmost, or leftish. Then it is undecidable in general whether the equality* $L(G) = L_{\mathsf{m}}(G)$ *holds for a given* (s)ACFG G.

Proof. By Lemma 4.1, it suffices to consider the case that G is an ACFG.

Given two context-free grammars G_1 and G_2, one can easily construct an ACFG G_0 such that $L(G_0) = L(G_1) \cap L(G_2)$. As it is undecidable in general whether the intersection of two context-free languages is empty [7], it is also undecidable in general whether the intersection of two context-free languages contains a non-empty word. It follows that it is undecidable in general whether the language $L(G)$ generated by an ACFG G contains a non-empty word.

We will now reduce this problem to the problem of deciding whether the languages $L(G')$ and $L_{\mathsf{m}}(G')$ coincide for an ACFG G'. So let G be an ACFG over the terminal alphabet $\{a, b\}$, and let S be its start symbol. We define an ACFG G' with start symbol S' by adding to G the new existential variables E, E', the universal variables S', A_a, A_b, and the following productions:

$$\begin{array}{llll} S' \to S, & S' \to EE', & E' \to A_a, & E' \to A_b,\\ E \to \varepsilon, & E \to aE, & E \to bE, & \\ A_a \to \varepsilon, & A_a \to a, & A_b \to \varepsilon, & A_b \to b. \end{array}$$

It is easily verified that

$$\begin{aligned}&L(G', E) = \{a, b\}^*,\\ &L(G', EE') = L_{\mathsf{rm}}(G', EE') = \{a, b\}^+, \text{ and}\\ &L_{\mathsf{lm}}(G', EE') = \emptyset.\end{aligned}$$

As $L(G', S) = L(G, S)$, we see that $L(G', S') = L(G, S) \cap L(G', EE')$. Further, we have that $L_{\mathsf{lm}}(G', S') = L_{\mathsf{lm}}(G', S) \cap L_{\mathsf{lm}}(G', EE') = \emptyset$. Hence,

$$L(G', S') = L_{\mathsf{lm}}(G', S') \quad \text{iff} \quad L(G', S') = \emptyset \quad \text{iff} \quad L(G, S) \cap \{a, b\}^{+} = \emptyset,$$

and by our remark above the last equality is undecidable.

The same proof applies to the leftish derivation mode, and a symmetric construction yields the result for the rightmost derivation mode. □

9. Concluding remarks

The main result of this paper is the characterization of the language class $\mathcal{L}(\mathsf{APDA})$ by sACFGs with leftmost derivation mode (Theorem 6.4). Unfortunately, this result does not answer the original question as to whether or not alternating context-free grammars correspond in expressive power to alternating pushdown automata.

We have further seen that for (s)ACFGs the expressive power depends on the chosen derivation mode (see Section 7). However, many questions about the exact relationships between the many language classes that are obtained by choosing various derivation modes remain open. Also only few closure properties for the various language classes defined by (s)ACFGs are currently known.

Recently a variant of the context-free grammars has been considered under the name of *conjunctive grammars* [15]. While in the derivation process of an alternating context-free grammar the application of a universal step splits the derivation into several independent sentential forms, each of which is then processed independently of all the others, the derivation process in a conjunctive grammar keeps all the different right-hand sides obtained by applying a universal step in a common context. Thus, the effect of a universal step in a conjunctive grammar is only *local* in contrast to the situation in an alternating context-free grammar. It is shown in [15] that many of the standard constructions of context-free grammars carry over to conjunctive grammars, and consequently the languages generated by them are recognizable in polynomial time. On the other hand, the class of languages generated by conjunctive grammars is quite expressive, as it properly contains all those languages that are obtained as intersections of finitely many context-free languages. However, the exact relationship between the class of conjunctive languages and the languages generated by ACFGs remains to be determined. Intuitively ACFGs should be more powerful than conjunctive grammars, but it is not even clear whether each conjunctive language is generated by an ACFG. However, in the linear case these concepts yield the same language class.

For future work it also remains to study the language classes that are obtained by (state-) alternating variants of non-context-free phrase structure grammars. Here growing context-sensitive grammars [5], context-sensitive grammars, monotone grammars and arbitrary phrase structure grammars come to mind. For example, in [16] two of the authors have studied the computational power of the alternating variant of the shrinking two-pushdown automaton of Buntrock and Otto [1].

Appendix

The diagram below depicts the known inclusion relations between some of the language classes discussed in the paper and some well-known language and complexity classes. By $\xrightarrow{}$ (dotted) we denote an inclusion, $\longrightarrow$ denotes a proper inclusion, and $=\!=\!=$ denotes equality.

$$\begin{tikzcd}
 & & & \text{RE} \arrow[d, equal] & & \\
 & & & \mathcal{L}_{\mathrm{lt}}(\text{sACFG}) \arrow[d, equal] & & \\
 & & & \mathcal{L}_{\mathrm{lt}}(\text{ECFG}) & & \\
 & \mathcal{L}(\text{sACFG}) \arrow[urr, dotted] & & \text{ETIME} \arrow[u] \arrow[d, equal] & & \\
 & & & \text{ALINSPACE} \arrow[d, equal] & & \\
 & & & \mathcal{L}_{\mathrm{lm}}(\text{sACFG}) \arrow[d, equal] & & \text{P} \arrow[uull] \arrow[d, equal] \\
 & & \mathcal{L}_{\mathrm{lt}}(\varepsilon\text{-free-sACFG}) \arrow[ur, dotted] & \mathcal{L}(\text{APDA}) & & \text{LOG}(\mathcal{L}(\text{lin-sACFG})) \arrow[d, equal] \\
 & & & & & \text{LOG}(\mathcal{L}(\text{lin-ACFG})) \\
 & & & \mathcal{L}_{\mathrm{lt}}(\varepsilon\text{-free-ECFG}) \arrow[uul, dotted] \arrow[uu, dotted] \arrow[d, equal] & & \\
\mathcal{L}(\text{ACFG}) \arrow[uuuuuur, dotted] & \mathcal{L}(\varepsilon\text{-free-sACFG}) \arrow[uuuuuu, dotted] \arrow[uuur, dotted] & & \text{CSL} & \mathcal{L}_{\mathrm{lm}}(\text{ACFG}) \arrow[uuul, dotted] & \\
 & & \mathcal{L}(\text{ECFG}) \arrow[uuuuuuul, dotted] & \text{DLINSPACE} \arrow[u, dotted] & & \\
 & & & \mathcal{L}_{\mathrm{lm}}(\varepsilon\text{-free-sACFG}) \arrow[u, dotted] & & \\
 & \mathcal{L}(\varepsilon\text{-free-ACFG}) \arrow[uuul, dotted] \arrow[uuu, dotted] & \mathcal{L}(\varepsilon\text{-free-ECFG}) \arrow[uuul, dotted] \arrow[uu, dotted] \arrow[uuuur] & & & \\
 & & & \mathcal{L}_{\mathrm{lm}}(\varepsilon\text{-free-ACFG}) \arrow[uu, dotted] \arrow[uuuur, dotted] & & \\
 & & & & & \mathcal{L}(\text{lin-sACFG}) \arrow[uuuuul, dotted] \arrow[uuuuuuu, dotted] \arrow[d, equal] \\
 & & & \mathcal{L}_{\mathrm{lm}}(\varepsilon\text{-free-ECFG}) \arrow[uuull] \arrow[uuul] \arrow[uu] \arrow[d, equal] & & \mathcal{L}(\text{lin-ACFG}) \\
 & & & \mathcal{L}_{\mathrm{lm}}(\text{ECFG}) \arrow[d, equal] & & \\
 & & & \text{CFL} & & \\
 & & & \text{LIN} \arrow[u] \arrow[uuurr] & & \\
 & & & \text{REG} \arrow[u] \arrow[d, equal] & & \\
 & & & \mathcal{L}(\text{right-lin-sACFG}) \arrow[d, equal] & & \\
 & & & \mathcal{L}(\text{right-lin-ACFG}) & &
\end{tikzcd}$$

Acknowledgements

The first author was supported in part by Waseda University Grant for Special Research Projects #2002C-006, which he gratefully acknowledges. The authors also want to thank Hartmut Messerschmidt for many fruitful discussions on the results presented here, and they thank the anonymous referees for very helpful comments.

References

[1] G. Buntrock, F. Otto, Growing context-sensitive languages and Church–Rosser languages, Inform. Comput. 141 (1998) 1–36.
[2] A.K. Chandra, D.C. Kozen, L.J. Stockmeyer, Alternation, J. Assoc. Comput. Mach. 28 (1981) 114–133.
[3] A.K. Chandra, L.J. Stockmeyer, Alternation, in: Proc. 17th FOCS, IEEE Computer Society Press, Silver Spring, MD, 1976, pp. 98–108.
[4] Z.Z. Chen, S. Toda, Grammatical characterizations of P and PSPACE, IEICE Trans. Inform. Systems E 73 (1990) 1540–1548.
[5] E. Dahlhaus, M. Warmuth, Membership for growing context-sensitive grammars is polynomial, J. Comput. System Sci. 33 (1986) 456–472.
[6] J. Dassow, G. Păun, Regulated Rewriting in Formal Language Theory, Monographs in Theoretical Computer Science, Vol. 18, Springer, Berlin, 1989.
[7] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, MA, 1979.
[8] O.H. Ibarra, T. Jiang, H. Wang, A characterization of exponential-time languages by alternating context-free grammars, Theoret. Comput. Sci. 99 (1992) 301–313.
[9] T. Kasai, An infinite hierarchy between context-free and context-sensitive languages, J. Comput. System Sci. 4 (1970) 492–508.
[10] R.E. Ladner, R.J. Lipton, L.J. Stockmeyer, Alternating pushdown automata, in: Proc. 19th FOCS, IEEE Computer Society Press, Silver Spring, MD, 1978, pp. 92–106.
[11] R.E. Ladner, R.J. Lipton, L.J. Stockmeyer, Alternating pushdown and stack automata, SIAM J. Comput. 13 (1984) 135–155.
[12] L. Liu, P. Weiner, An infinite hierarchy of intersections of context-free languages, Math. Systems Theory 7 (2) (1973) 185–192.
[13] E. Moriya, Some remarks on state grammars and matrix grammars, Inform. Control 23 (1973) 48–57.
[14] E. Moriya, A grammatical characterization of alternating pushdown automata, Theoret. Comput. Sci. 67 (1989) 75–85.
[15] A. Okhotin, Conjunctive grammars, J. Automata, Languages Combin. 6 (2001) 519–535.
[16] F. Otto, E. Moriya, Shrinking alternating two-pushdown automata, IEICE Trans. Inform. Systems E87-D(4) (2004) 959–966.
[17] A. Salomaa, Matrix grammars with a leftmost restriction, Inform. Control 20 (1972) 143–149.
[18] H. Sudborough, A note on tape-bounded complexity classes and linear context-free languages, J. Assoc. Comput. Mach. 22 (1975) 499–500.