数理解析研究所講究録 1375 巻 2004 年 92-98

On Alternating Context-Free Grammars – Old and New Versions and Their Characterizations (An Extended Abstract)

(Alternating CFG 再び – 新旧種とその特徴付け)

早稲田大学・教育学部　守屋　悦朗 (Etsuro Moriya)*
Department of Mathematics, School of Education
Waseda University, Shinjuku-ku, Tokyo, 169-8050, Japan
and
ドイツ・カッセル大学　Dieter Hofbauer, Maria Huber, Friedrich Otto
Fachbereich Mathematik/Informatik,Universität Kassel, 34109 Kassel, Germany

Abstract

A new type of 'alternating 'context-free grammar', called *state-alternating context-free grammar* is introduced, and the generative power of it is compared to some well-known language classes, especially to the alternating context-free grammar of Moriya (1989) and to the alternating pushdown automaton. Further, various derivation strategies are considered, and their influence on the expressive power of (state-) alternating context-free grammars is investigated.

1 Introduction

Alternation is a powerful generalization of nondeterminism that has led to many interesting results in automata and complexity theory. It was first introduced by Chandra, Kozen and Stockmeyer [1, 2] for general Turing machines and by Ladner, Lipton and Stockmeyer [7, 8] for pushdown automata. It is known that the class of languages $\mathcal{L}$(ALBA) accepted by alternating Turing machines in linear space (the so-called *alternating linear bounded automata* ALBA) coincides with the class $\mathcal{L}$(APDA) of languages accepted by the *alternating pushdown automata* APDA, which in turn coincides with the deterministic time complexity class EXPTIME := $\bigcup_{c>0}$ DTIME(c^n) [2, 8].

The (non-alternating) pushdown automata accept exactly the context-free languages, while the nondeterministic 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 $\mathcal{L}$(APDA) accepted by the alternating pushdown automata. This question was first addressed by Moriya [10] by considering alternating context-free grammars. Here an *alternating context-free grammar* is a grammar $G = (V, U, \Sigma, P, S)$, where V is a set of variables (or nonterminals), $U \subseteq V$ is a set of *universal* variables, while the variables in $V \setminus U$ are called *existential*, Σ is a set of terminals, S is the start symbol, and P is a set of context-free productions. In the derivation process an existential variable is rewritten as usual, but a universal variable is rewritten by applying all productions with that variable as left-hand side simultaneously, thus giving a finite number of successor sentential forms. 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 the above sense such that the root is labelled with the start symbol and all leaves are labelled with w.

In the following we will denote the class of alternating context-free grammars by ACFG, and the class of languages generated by them by $\mathcal{L}$(ACFG). Further, $\mathcal{L}$(ε-free-ACFG) will de-

*This author was supported in part by Waseda University Grant for Special Research Projects #2002C-006.

note the class of languages generated by alternating context-free grammars without ε-rules, and $\mathcal{L}$(linear-ACFG) the class of languages generated by *linear* alternating context-free grammars. Finally we use $\mathcal{L}_{\mathsf{lm}}$(ACFG) and $\mathcal{L}_{\mathsf{lm}}$($\varepsilon$-free-ACFG) to denote the corresponding classes of languages generated by the leftmost derivation strategy.

In [10] 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 G, the set of words generated by *leftmost* derivations is in general a proper subset of the set of words that can be generated by G. Nevertheless some interesting partial results have been obtained. First, Chen and Toda [3] presented complexity theoretical characterizations of the language classes $\mathcal{L}_{\mathsf{lm}}$(linear-ACFG) and $\mathcal{L}_{\mathsf{lm}}$($\varepsilon$-free-ACFG) by showing that

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

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 $\mathsf{P} = \mathsf{LOG}(\mathcal{L}(\text{linear-ACFG}))$. Then Ibarra, Jiang, and Wang gave a grammatical characterization for $\mathcal{L}$(APDA) in [5] by showing that $\mathcal{L}$(APDA) = $\mathcal{L}$(linear-erasing-ACFG), where an alternating context-free grammar G is called *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 $\mathcal{L}$(linear-erasing-ACFG) in $\mathcal{L}$(APDA) = $\mathcal{L}$(ALBA) remains valid even without the 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 [5] crucially depends on the use of these endmarkers.

We will consider a new variant of alternating context-free grammar, 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 SCFG, were introduced by Kasai in [6]. A *state-alternating context-free grammar*, sACFG for short, will be an SCFG 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, and let (p, α) be a sentential form. If p is an existential state, that is, $p \in Q \setminus U$, then we choose an occurrence of a variable A in α, say $\alpha = \alpha_1 A \alpha_2$, and a production of the form $(p, A) \to (q, \beta)$ and derive $(q, \alpha_1 \beta \alpha_2)$ from (p, α). If, however, p is a universal state, and $(p, A) \to (q_1, \beta_1), \ldots, (p, A) \to (q_s, \beta_s)$ are all the productions with lefthand side (p, A), then from $(p, \alpha_1 A \alpha_2)$ we obtain all the sentential forms $(q_1, \alpha_1 \beta_1 \alpha_2), \ldots, (q_s, \alpha_1 \beta_s \alpha_2)$ in parallel, and following this step all these 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. By $L(G)$ we denote the language consisting 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 $p \in F$. Here we remark that the labels of different leaves may differ in the first component, that is, in the state, but that they

must agree in the second component, that is, in the terminal string.

The notion of *leftish* derivations was introduced for the first time in [6]. We also consider leftish derivations for sACFG and we use $\mathcal{L}_{\mathsf{lt}}(\mathsf{ACFG})$ and $\mathcal{L}_{\mathsf{lt}}(\varepsilon\text{-free-ACFG})$ to denote the classes of languages generated by the leftish derivation strategy. Here a derivation step $(p, \beta A\gamma) \Rightarrow (q, \beta\alpha\gamma)$ 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.

In Section 2 we will compare the sACFGs to the ACFGs. In Section 3 we will present 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. In Section 4 we will consider various restricted versions of sACFGs and analyze the complexity of the language classes generated by leftmost derivations. Then, in Section 5, we will derive characterizations for the language classes $\mathcal{L}_{\mathsf{lm}}(\mathsf{sACFG})$ and $\mathcal{L}_{\mathsf{lm}}(\mathsf{ACFG})$ in terms of alternating pushdown automata. Finally in Section 6 we will consider the classes of languages that are generated by sACFGs by using leftish derivations and unrestricted derivations, respectively.

2 Basic Properties of ACFGs and sACFGs

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

Example 2.1. For $i = 1, 2$, let $G_i := (V_i, U_i, \Sigma, P_i, S_i)$ be an ACFG, where we assume that $V_1 \cap V_2 = \emptyset$, 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 G generates the language $L(G_1) \cap L(G_2)$. This shows that, for any derivation mode m, the language class $\mathcal{L}_{\mathsf{m}}(\mathsf{ACFG})$ is closed under intersection. It follows analogously that $\mathcal{L}_{\mathsf{m}}(\mathsf{sACFG})$ is closed under intersection. Further, as each context-free grammar can be regarded as an ACFG with no universal variables, it follows that $\mathcal{L}_{\mathsf{m}}(\mathsf{ACFG})$ contains each language that can be written as the intersection of finitely many context-free languages.

The next lemma shows that any ACFG can be simulated by an sACFG. Here in addition to the leftmost, leftish and unrestricted derivation modes mentioned before, we also consider the *rightmost* derivation mode, which is a trivial analogy to the leftmost one. We denote by $\mathcal{L}_{\mathsf{rm}}(\mathsf{X})$ the class of languages generated by grammars of type X under the rightmost derivation strategy.

Lemma 2.2. *For each* ACFG G, *we can construct an* sACFG G' *such that* $L_{\mathsf{m}}(G) = L_{\mathsf{m}}(G')$ *holds, where* m *is the leftmost, the rightmost, the leftish or the unrestricted derivation mode. Moreover, if* G *is* ε*-free and/or* (*right-*) *linear, then so is* G'.

Currently we don't know whether the converse of Lemma 2.2 holds. At least for linear grammars we do have the converse of Lemma 2.2.

Lemma 2.3. *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'.

The above lemmas give the following consequences.

Theorem 2.4. (a) $\mathcal{L}(\varepsilon\text{-free-right-linear-ACFG}) = \mathcal{L}(\varepsilon\text{-free-right-linear-sACFG})$ *and* $\mathcal{L}(\text{right-linear-ACFG}) = \mathcal{L}(\text{right-linear-sACFG})$.
(b) $\mathcal{L}(\varepsilon\text{-free-linear-ACFG}) = \mathcal{L}(\varepsilon\text{-free-linear-sACFG})$ *and* $\mathcal{L}(\text{linear-ACFG}) = \mathcal{L}(\text{linear-sACFG})$.

(c) $\mathcal{L}_{\mathsf{m}}(\varepsilon\text{-free-ACFG}) \subseteq \mathcal{L}_{\mathsf{m}}(\varepsilon\text{-free-sACFG})$ *and* $\mathcal{L}_{\mathsf{m}}(\mathsf{ACFG}) \subseteq \mathcal{L}_{\mathsf{m}}(\mathsf{sACFG})$, *where* m *is any of the leftmost, the leftish, the rightmost or the unrestricted derivation mode.*

We close this section with two results concerning the form of sACFGs. The first result states that for these grammars a normal form similar to the Chomsky normal form exists. We call a production of an sACFG $G = (Q, U, V, \Sigma, P, S, q_0, F)$ a *unit-production* if it is of the form $(p, A) \to (q, B)$ for some $p, q \in Q$ and $A, B \in V$. The sACFG G is said to be in *weak Chomsky normal form* if it satisfies the following conditions:

(1) each production $(p, A) \to (q, \alpha)$ from P satisfies $\alpha \in (V \cup V^2 \cup \Sigma \cup \{\varepsilon\})$;

(2) for each pair $(p, A) \in Q \times V$, if there are two or more productions with lefthand side (p, A), then all these productions are unit-productions.

Thus, it is only for unit-productions that it plays a role whether the actual state is universal or existential. The second result states that we can assume without loss of generality that all states of an sACFG are final.

Lemma 2.5. *For each derivation mode* m *and each* ACFG (sACFG) G, *we can construct an* ACFG (sACFG) G' *in weak Chomsky normal form such that* G *and* G' *are equivalent with respect to derivation mode* m, *and all states of* G' *are final in case* G' *is an* sACFG. *In addition, if* G *is* ε*-free, then so is* G'.

3 A Lower Bound for $\mathcal{L}_{\mathsf{lm}}(\mathsf{ACFG})$

In Section 2 we have seen that $\mathcal{L}_{\mathsf{lm}}(\mathsf{ACFG})$ is a lower bound for the language class $\mathcal{L}_{\mathsf{lm}}(\mathsf{sACFG})$. Here we will give a lower bound for $\mathcal{L}_{\mathsf{lm}}(\mathsf{ACFG})$. As we have seen in Example 2.1, $\mathcal{L}_{\mathsf{lm}}(\mathsf{ACFG})$ includes the class of languages that can be obtained as the intersections of finitely many context-free languages. Let CFL_k denote the class of *k-intersection languages*, that is, the class of languages that can be written as the intersection of k context-free languages, and let $\mathsf{CFL}_\omega := \bigcup_{k \geq 0} \mathsf{CFL}_k$. Liu and Weiner [9] proved that the classes CFL_k form an infinite hierarchy within the class of context-sensitive languages. From Example 2.1 we obtain the following inclusion.

Observation 3.1. $\mathsf{CFL}_\omega \subseteq \mathcal{L}_{\mathsf{lm}}(\mathsf{ACFG}) \cap \mathcal{L}(\mathsf{ACFG})$.

We will consider a sequence of example languages L_k $(k \geq 2)$, and L_ω. For $k \geq 2$, let $\Sigma_k := \{a_1, a_2, \ldots, a_k\}$, and let L_k be the language

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

Further let L_ω be the language

$$\{(a^{i_1} b^1 a^{i_2} b^2 \ldots a^{i_k} b^k)^2 \mid k \geq 0, i_j \geq 1\} \subseteq \{a, b\}^*.$$

Liu and Weiner [9] proved that

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

A slight modification of their proof gives the following lemma.

Lemma 3.2. *For each* $k \geq 2$, $L_k \in \mathsf{CFL}_k \setminus \mathsf{CFL}_{k-1}$.

On the other hand, concerning L_ω we can show the following lemmas.

Lemma 3.3. $L_\omega \notin \mathsf{CFL}_\omega$.

Lemma 3.4. $L_\omega \in \mathcal{L}_{\mathsf{lm}}(\mathsf{ACFG}) \cap \mathcal{L}(\mathsf{ACFG})$.

From Observation 3.1 and Lemmas 3.3 and 3.4, we obtain the main result of this section.

Theorem 3.5. $\mathsf{CFL}_\omega \subsetneq \mathcal{L}_{\mathsf{lm}}(\mathsf{ACFG}) \cap \mathcal{L}(\mathsf{ACFG})$.

4 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 4.1. $\mathcal{L}$(right-linear-ACFG) *coincides with the class* REG *of regular languages.*

By Theorem 2.4(a) this gives the following consequence.

Corollary 4.2. $\mathcal{L}$(right-linear-ACFG) = $\mathcal{L}$(right-linear-sACFG) = REG.

Combining Theorem 2.4 (b) with the result of Chen and Toda on $\mathcal{L}$(linear-ACFG) [3], we obtain the following corollary.

Corollary 4.3. LOG($\mathcal{L}$(linear-ACFG)) = LOG($\mathcal{L}$(linear-sACFG)) = P.

Since P = ALOGSPACE [2], Corollary 4.3 can be viewed as the counterpart (with respect to alternation) of the well-known result by Sudborough [11] that NLOGSPACE = LOG ($\mathcal{L}$(linear-CFG)), where NLOGSPACE (ALOGSPACE, respectively) denote the class of languages that are accepted by nondeterministic (alternating, respectively) Turing machines in 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. Note that NLINSPACE (DLINSPACE, respectively) coincides with the class of context-sensitive languages (deterministic context-sensitive languages, respectively).

Finally, we turn our attention to the ε-free sACFGs. We say that an sACFG G has *bounded unit-productions* if there exists a constant $c \geq 1$ such that, for each $w \in L_{\mathsf{lm}}(G)$, there exists a leftmost G-derivation tree for w such that the number of applications of unit-productions (i.e., productions of the form $(p, A) \to (q, B)$, where p, q are states and A, B are variables) on each path of this derivation tree is bounded from above by the number $c \cdot |w|$. Based on the notion of bounded unit-productions, we can now establish the following interesting result.

Theorem 4.4. $\mathcal{L}_{\mathsf{lm}}(\varepsilon\text{-free-sACFG}) \subseteq$ DLINSPACE.

This result together with Theorem 2.4 (c) and the result of Chen and Toda on ε-free ACFGs [3] yields the following result.

Corollary 4.5. LOG($\mathcal{L}_{\mathsf{lm}}(\varepsilon$-free-ACFG)) = LOG($\mathcal{L}_{\mathsf{lm}}(\varepsilon$-free-sACFG)) = PSPACE.

5 Characterizing Language Classes by Automata

The original purpose for introducing the ACFGs was to give a characterization for the language class $\mathcal{L}$(APDA) [10]. Such a characterization will be derived in this section in terms of the sACFGs with leftmost derivations. We begin by restating the definition of the APDA in short.

An *alternating pushdown automaton*, APDA for short, M is given through 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 actual 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 and the last letter of α being the symbol on the bottom of the pushdown. 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 an inner node 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 node that it labelled by $(p, u, \beta\alpha)$ for some $(p, \beta) \in \delta(q, a, Z)$;

(4) if an inner node is labelled by $(q, au, Z\alpha)$, where $q \in U$, $a \in \Sigma \cup \{\varepsilon\}$, $Z \in \Gamma$, and if $\delta(q, a, Z) = \{(p_1, \beta_1), \ldots, (p_m, \beta_m)\}$, then this node has m successor nodes labelled by $(p_1, u, \beta_1\alpha), \ldots, (p_m, u, \beta_m\alpha)$, respectively.

By $L(M)$ we denote the language consisting

of all strings that are accepted by M.

Next we establish two technical lemmas on APDAs.

Lemma 5.1. *For each* APDA M, *there exists an* APDA M' *such that* $L(M) = L(M')$, *and all final states of* M' *are existential.*

Lemma 5.2. *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.*

Based on these technical results we can now establish the following characterization.

Theorem 5.3. $\mathcal{L}_{\mathsf{lm}}(\mathsf{sACFG}) = \mathcal{L}(\mathsf{APDA})$.

By Lemma 2.2 and the result on $\mathcal{L}$(APDA) from [8] the above characterization yields the following consequence.

Corollary 5.4. $\mathcal{L}_{\mathsf{lm}}(\mathsf{ACFG}) \subseteq \mathcal{L}_{\mathsf{lm}}(\mathsf{sACFG}) = \mathcal{L}(\mathsf{APDA}) = \mathsf{EXPTIME}$.

We will next derive a characterization of the language class $\mathcal{L}_{\mathsf{lm}}$(ACFG) in terms of a variant of the alternating pushdown automata, the so-called *stack-alternating pushdown automaton*, stackAPDA. A stackAPDA is a pushdown automaton which has a single state only and whose pushdown symbols are divided into two types, *universal* and *existential* ones. Further, a stackAPDA accepts by empty pushdown. Thus, a stackAPDA is denoted by a 5-tuple $M = (\Sigma, \Gamma, U, \delta, Z_0)$, where Σ and Γ are finite sets of input and pushdown symbols, respectively, $U \subseteq \Gamma$ is a set of universal pushdown symbols, and $Z_0 \in \Gamma$ is the initial pushdown symbol. The transition relation δ is a partial function from $(\Sigma \cup \{\varepsilon\}) \times \Gamma$ into the finite subsets of Γ^*. A configuration $(x, Z\alpha) \in \Sigma^* \times \Gamma\Gamma^*$ of M represents the current content x on the input tape and the current content $Z\alpha$ on the pushdown store, where the input head is on the leftmost symbol of x and Z is the topmost symbol on the pushdown store. The initial configuration on input x is (x, Z_0). For a given input x, a *computation tree* for M is a finite rooted tree the nodes of which are labelled with configurations. It is defined in the same way as that for an ordinary alternating pushdown automaton, except that the pushdown symbol on the top of the pushdown store of M plays the counterpart to the universal or existential states of an ordinary alternating pushdown automaton. Thus, an input x is accepted by the stackAPDA M, if there is a computation tree for M such that the root is labelled with (x, Z_0), and each leaf is labelled with the pair $(\varepsilon, \varepsilon)$.

It is important to note that if a is an input symbol and Z is a universal stack symbol of a stackAPDA M, and if M contains the transitions $\delta(a, Z) = \{\beta_1, \ldots, \beta_p\}$ as well as $\delta(\varepsilon, Z) = \{\gamma_1, \ldots, \gamma_q\}$, then a node π of a computation tree of M that is labelled with a configuration of the form $(ax, Z\alpha)$ has either p sons labelled with $(x, \beta_1\alpha), \ldots, (x, \beta_p\alpha)$, respectively, or q sons labelled with $(ax, \gamma_1\alpha), \ldots, (ax, \gamma_q\alpha)$, respectively. If we relax the condition, requiring that in the above situation both the a- and the ε-transitions must be applied, then we say that the stackAPDA M works in *relaxed mode.*

Lemma 5.5. $\mathcal{L}$(stackAPDA) $= \mathcal{L}_{\mathsf{relaxed}}$(stackAPDA), *where* $\mathcal{L}_{\mathsf{relaxed}}$(stackAPDA) *denotes the class of languages that are accepted by* stackAPDA*s in relaxed mode.*

Based on the normal form result for ACFGs (Lemma 2.5) and the above result, we now derive the following characterization.

Theorem 5.6. $\mathcal{L}_{\mathsf{lm}}(\mathsf{ACFG}) = \mathcal{L}(\mathsf{stackAPDA})$.

Summarizing Corollary 5.4 and Theorem 5.6 we have the following result.

Corollary 5.7. $\mathcal{L}(\mathsf{stackAPDA}) = \mathcal{L}_{\mathsf{lm}}(\mathsf{ACFG}) \subseteq \mathcal{L}_{\mathsf{lm}}(\mathsf{sACFG}) = \mathcal{L}(\mathsf{APDA})$.

6 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.

Theorem 6.1. (a) $\mathcal{L}_{\mathsf{lt}}(\varepsilon\text{-free-sACFG}) \subseteq \mathcal{L}(\mathsf{APDA})$.
(b) $\mathcal{L}_{\mathsf{lt}}(\mathsf{sACFG}) = \mathsf{RE}$.

Corollary 6.2. $\mathcal{L}_{\mathsf{lt}}(\varepsilon\text{-free-sACFG}) \subseteq \mathcal{L}_{\mathsf{lm}}(\mathsf{sACFG})$.

As to the expressive power of the various derivation modes for the sACFGs, we have

Theorem 6.3. (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* $\mathcal{L}(\mathsf{sACFG}) \subseteq \mathcal{L}_{\mathsf{lt}}(\mathsf{sACFG})$.

Corollary 6.4. (a) $\mathcal{L}_{\mathsf{lm}}(\varepsilon\text{-free-ACFG}) = \mathcal{L}_{\mathsf{lt}}(\varepsilon\text{-free-ACFG}) \subseteq \mathcal{L}_{\mathsf{lt}}(\varepsilon\text{-free-sACFG})$.
(b) $\mathcal{L}_{\mathsf{lm}}(\mathsf{ACFG}) = \mathcal{L}_{\mathsf{lt}}(\mathsf{ACFG}) \subsetneq \mathcal{L}_{\mathsf{lt}}(\mathsf{sACFG})$.

Observe that $\mathcal{L}(\varepsilon\text{-free-sACFG}) \subseteq \mathcal{L}_{\mathsf{lt}}(\varepsilon\text{-free sACFG}) \subseteq \mathcal{L}_{\mathsf{lm}}(\mathsf{sACFG})$ holds by Corollary 6.2 and Theorem 6.3 (a).

References

[1] A.K. Chandra and L.J. Stockmeyer, Alternation, in *Proc. of 17th FOCS*, pp.98–108, IEEE Computer Society Press, 1976.

[2] A.K. Chandra, D.C. Kozen and L.J. Stockmeyer, Alternation, *Journal of the ACM*, 28: 114–133, 1981.

[3] Z.Z. Chen and S. Toda, Grammatical characterizations of P and PSPACE, *The Transactions of the IEICE*, E 73: 1540–1548, 1990.

[4] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, M.A., 1979.

[5] O.H. Ibarra, T. Jiang and H. Wang, A characterization of exponential-time languages by alternating context-free grammars, *Theoretical Computer Science*, 99: 301–313, 1992.

[6] T. Kasai, An infinite hierarchy between context-free and context-sensitive languages, *Journal of Computer and System Sciences*, 4: 492–508, 1970.

[7] R.E. Ladner, R.J. Lipton and L.J. Stockmeyer, Alternating pushdown automata, in *Proc. of 19th FOCS*, pp.92–106, IEEE Computer Society Press, 1978.

[8] R.E. Ladner, R.J. Lipton and L.J. Stockmeyer, Alternating pushdown and stack automata, *SIAM Journal on Computing*, 13: 135–155, 1984.

[9] L. Liu and P. Weiner, An infinite hierarchy of intersections of context-free languages, *Mathematical Systems Theory*, 7: 185–192, 1973.

[10] E. Moriya, A grammatical characterization of alternating pushdown automata, *Theoretical Computer Science*, 67: 75–85, 1989.

[11] H. Sudborough, A note on tape-bounded complexity classes and linear context-free languages, *Journal of the ACM*, 22: 499–500, 1975.

Appendix

The diagram below depicts the known inclusion relations between some of the important language classes discussed in the paper and some well-known language and complexity classes. Here —— denotes an inclusion, ⟶ denotes a proper inclusion, and ═══ denotes equality.