

Conjunctive Grammars and Alternating Pushdown Automata *Draft Paper*

Tamar Aizikowitz* Michael Kaminski

Department of Computer Science
Technion – Israel Institute of Technology
Haifa 32000
Israel

May 4, 2009

Abstract

In this paper we introduce a variant of alternating pushdown automata, *Synchronized Alternating Pushdown Automata*, which accept the same class of languages as those generated by conjunctive grammars.

Keywords: *conjunctive grammars, alternating automata.*

1 Introduction

Many well known computational models support non-deterministic computations with existential acceptance conditions, thereby leading to an inherent disjunctive quality of the class of languages accepted. When considering the dual form of these models (e.g., Co-NP), universal acceptance conditions lead to conjunction: all computations must accept. This type of acceptance

*Corresponding author, e-mail: tamarai@cs.technion.ac.il, tel: +972 52 3790481, fax: +972 4 8293900

is useful in such fields as concurrent programming where *all* processes must meet correctness demands. In this paper we explore extensions of models for context-free languages that combine both the notion of conjunction and of disjunction, leading to a richer set of languages.

Conjunctive Grammars (CG) are an example of such a model. Introduced by Okhotin in [6], CG are a generalization of context-free grammars. Explicit intersection operations are allowed in rules thereby adding the power of conjunction. CG were shown by Okhotin to accept all finite conjunctions of context-free languages, as well as some additional languages. However, there is no known non-trivial technique to prove a language cannot be derived by a CG, so their exact placing in the Chomsky hierarchy is unknown. Okhotin proved the languages generated by these grammars to be polynomially parsable [6, 7], making the model practical from a computational standpoint, and therefore, of interest for applications in various fields such as, e.g., programming languages. In this paper we introduce a new model of alternating automata, *Synchronized Alternating Pushdown Automata* (SAPDA), and prove that it is equivalent to the Conjunctive Grammar model.¹

The concept of alternating automata models was first introduced by Chandra, Kozen, and Stockmeyer in [1]. In these models, computations alternate between existential and universal modes of acceptance, hence their name. This behavior is achieved by splitting the state set into two disjoint groups, existential states and universal states. The acceptance model dictates that all possible computations from universal states must be accepting, whereas only one must be accepting from existential states. Thus, for a word to be accepted it must meet both disjunctive and conjunctive conditions. In the case of Alternating Finite State Automata and Alternating Turing Machines, the alternating models have been shown to be equivalent in expressive power to their non-alternating counterparts, see [1].

Alternating Pushdown Automata (APDA) were also introduced in [1] and were further explored in [5]. Like Conjunctive Grammars, APDA add the power of conjunction over context-free languages. Therefore, unlike Finite Automata and Turing Machines, here alternation increases the expressiveness of the model. In fact, APDA accept exactly the exponential time languages, see [1, 5].

It is well known that Context-Free Grammars and Pushdown Automata are equivalent, e.g., see [3, pp. 115–119]. Yet, the APDA model is stronger

¹We call two models equivalent if they accept/generate the same class of languages.

than the CG model [6]. Our *Synchronized Alternating Pushdown Automata* are weaker than general APDA and accept exactly the class of languages derived by Okhotin's Conjunctive Grammars. Okhotin showed in [8, 9] that Linear Conjunctive Grammars, a subfamily of the Conjunctive Grammars, are equivalent to Trellis Automata [2]. However, SAPDA are the first class of automata shown to be equivalent to general CG.

The paper is organized as follows. In Section 2 we recall the definition of the Conjunctive Grammar model and in Section 3 we introduce our SAPDA model. Section 4 details our main result, namely the equivalence of the CG and SAPDA models, and Section 5 is a short conclusion of our work.

2 Conjunctive Grammars

The following definitions are taken from [6].

Definition 1 A *Conjunctive Grammar* is a quadruple $G = (V, \Sigma, P, S)$, where

- V, Σ are disjoint finite sets of non-terminal and terminal symbols respectively.
- $S \in V$ is the designated start symbol.
- P is a finite set of rules of the form $A \rightarrow (\alpha_1 \& \cdots \& \alpha_k)$ such that $A \in V$ and $\alpha_i \in (V \cup \Sigma)^*$, $i = 1, \dots, k$. If $k = 1$ then we write $A \rightarrow \alpha$.

Definition 2 *Conjunctive Formulas* over $V \cup \Sigma \cup \{(\cdot), \&\}$ are defined by the following recursion.

- ϵ is a conjunctive formula.
- Every symbol in $V \cup \Sigma$ is a conjunctive formula.
- If \mathcal{A} and \mathcal{B} are formulas, then $\mathcal{A}\mathcal{B}$ is a conjunctive formula.
- If $\mathcal{A}_1, \dots, \mathcal{A}_k$ are formulas, then $(\mathcal{A}_1 \& \cdots \& \mathcal{A}_k)$ is a conjunctive formula.

Notation Below we use the following notation: σ, τ , etc. denote terminal symbols, u, w, y , etc. denote terminal words, A, B , etc. denote non-terminal symbols, α, β , etc. denote non-terminal words, and \mathcal{A}, \mathcal{B} , etc. denote conjunctive formulas. All the symbols above may also be indexed.

Definition 3 For a conjunctive formula $\mathcal{A} = (\mathcal{A}_1 \& \cdots \& \mathcal{A}_k)$, $k \geq 2$, the \mathcal{A}_i s, $i = 1, \dots, k$, are called *conjuncts* of \mathcal{A} ,² and \mathcal{A} is called the *enclosing formula*. If \mathcal{A}_i contains no &s, then it is called a *simple conjunct*.³

Definition 4 For a CG G , the relation of *immediate derivability* on the set of conjunctive formulas, \Rightarrow_G , is defined as follows.

- (1) $s_1 A s_2 \Rightarrow_G s_1 (\alpha_1 \& \cdots \& \alpha_k) s_2$ for all $A \rightarrow (\alpha_1 \& \cdots \& \alpha_k) \in P$, and
- (2) $s_1 (w \& \cdots \& w) s_2 \Rightarrow_G s_1 w s_2$ for all $w \in \Sigma^*$,

where $s_1, s_2 \in (V \cup \Sigma \cup \{(\cdot), \&\})^*$. As usual, \Rightarrow_G^* is the reflexive and transitive closure of \Rightarrow_G , the language $L(\mathcal{A})$ of a conjunctive formula \mathcal{A} is

$$L(\mathcal{A}) = \{w \in \Sigma^* : \mathcal{A} \Rightarrow_G^* w\},$$

and the language $L(G)$ of G is

$$L(G) = \{w \in \Sigma^* : S \Rightarrow_G^* w\}.$$

We refer to (1) as *production* and (2) as *contraction* rules, respectively.

In particular, note that a terminal word w is derived from a conjunctive formula $(\mathcal{A}_1 \& \cdots \& \mathcal{A}_k)$ if and only if it is derived from each \mathcal{A}_i , $i = 1, \dots, k$. Hence, $(\mathcal{A}_1 \& \cdots \& \mathcal{A}_k)$ derives the finite intersection of the languages $L(\mathcal{A}_i)$.

Example 5 ([6, Example 1]) The following conjunctive grammar generates the non-context-free language $\{a^n b^n c^n : n = 0, 1, \dots\}$, called the *multiple agreement* language. $G = (V, \Sigma, P, S)$, where

- $V = \{S, A, B, C, D, E\}$, $\Sigma = \{a, b, c\}$, and
- P consists of the following rules.

$$S \rightarrow (C \& A)$$

$$C \rightarrow Cc \mid D$$

$$A \rightarrow aA \mid E$$

$$D \rightarrow aDb \mid \epsilon$$

²Note that this definition is different from Okhotin's definition in [6].

³Note that, according to this definition, conjunctive formulas which are elements of $(V \cup \Sigma)^*$ are not simple conjuncts or enclosing formulas.

$$E \rightarrow bEc \mid \epsilon$$

The intuition of the above derivation rules is as follows.

$$L(C) = \{a^m b^m c^n : m, n = 0, 1, \dots\},$$

$$L(A) = \{a^m b^n c^n : m, n = 0, 1, \dots\},$$

and

$$L(G) = L(C) \cap L(A) = \{a^n b^n c^n : n = 0, 1, \dots\}.$$

For example, the word $aabbcc$ can be derived as follows.

$$\begin{aligned} S &\Rightarrow (C \& A) \Rightarrow (Cc \& A) \Rightarrow (Ccc \& A) \Rightarrow (Dcc \& A) \Rightarrow (aDbcc \& A) \\ &\Rightarrow (aaDbbcc \& A) \Rightarrow (aabbcc \& A) \Rightarrow (aabbcc \& aA) \\ &\Rightarrow (aabbcc \& aaA) \Rightarrow (aabbcc \& aaE) \Rightarrow (aabbcc \& aabEc) \\ &\Rightarrow (aabbcc \& aabbEcc) \Rightarrow (aabbcc \& aabbcc) \Rightarrow aabbcc \end{aligned}$$

Following is an especially interesting example, due to Okhotin, of a CG which uses recursive conjunctions to derive a language which *cannot* be obtained by a finite conjunction of context-free languages.

Example 6 ([6, Example 2]) The following conjunctive grammar derives the non-context-free language $\{w\$w : w \in \{a, b\}^*\}$, called *reduplication* with a center marker. $G = (V, \Sigma, P, S)$, where

- $V = \{S, A, B, C, D, E\}$, $\Sigma = \{a, b, \$\}$, and
- P consists of the following derivation rules.

$$\begin{aligned} S &\rightarrow (C \& D) \\ C &\rightarrow aCa \mid aCb \mid bCa \mid bCb \mid \$ \\ D &\rightarrow (aA \& aD) \mid (bB \& bD) \mid \$E \\ A &\rightarrow aAa \mid aAb \mid bAa \mid bAb \mid \$Ea \\ B &\rightarrow aBa \mid aBb \mid bBa \mid bBb \mid \$Eb \\ E &\rightarrow aE \mid bE \mid \epsilon \end{aligned}$$

The non-terminal C verifies that the lengths of the words before and after the center marker $\$$ are equal. The non-terminal D derives the language $\{w\$uw \mid w, u \in \{a, b\}\}$. The grammar language is the intersection of these two languages, and is therefore, in fact, the reduplication with a center marker language. For a more detailed description, see [6, Example 2].

3 Synchronized Alternating Pushdown Automata

We define a class of automata called *Synchronized Alternating Pushdown Automata* (SAPDA) as a variation on the standard PDA model. Similarly to standard Alternating Pushdown Automata [1, 5], SAPDA have the power of both existential and universal choice.

Our definition is different from (but equivalent to) the definition in [1], which is based on existential and universal state sets. Instead, in our model, transitions are made to a conjunction of states. The model is non-deterministic, therefore, several different conjunctions of states may be possible from a given configuration.⁴ If all conjunctions are of one state only, the automaton is a standard PDA.

The stack memory of an SAPDA is a tree. Each leaf has a processing head which reads the input and writes to its branch independently. When a multiple-state conjunctive transition is applied, the stack branch splits into multiple branches, one for each conjunct.⁵ The branches process the input independently, however sibling branches must empty synchronously, after which the computation continues from the parent branch.

Definition 7 A *synchronized alternating pushdown automaton* is a tuple $A = (Q, \Sigma, \Gamma, \delta, q_0, \perp)$, where δ is a function that assigns to each element of $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ a finite subset of

$$\{(q_1, \alpha_1) \wedge \cdots \wedge (q_k, \alpha_k) : k = 1, 2, \dots, i = 1, \dots, k, q_i \in Q, \text{ and } \alpha_i \in \Gamma^*\}.$$

Everything else is defined as in the standard PDA model. Namely,

- Q is a finite set of states,
- Σ and Γ are the input and the stack alphabets, respectively,
- $q_0 \in Q$ is the initial state, and
- $\perp \in \Gamma$ is the initial stack symbol,

⁴This type of formulation is standard in the field of Formal Verification, e.g., see [4].

⁵This is similar to the concept of a transition from a universal state in the standard formulation of alternating automata, as all branches must accept.

see, e.g., [3, pp. 107–112].

We describe the current stage of the automaton computation as a labeled tree. The tree encodes the stack contents, the current states of the stack-branches, and the remaining input to be read for each stack-branch. States and remaining inputs are saved in leaves only, as these encode the stack-branches currently processed.

Definition 8 A *configuration* of an SAPDA is a labeled tree. Each internal node is labeled $\alpha \in \Gamma^*$ denoting the stack-branch contents, and each leaf node is labeled (q, w, α) , where

- $q \in Q$ is the current state,
- $w \in \Sigma^*$ is the remaining input to be read, and
- $\alpha \in \Gamma^*$ is the stack-branch contents.

For a node v in a configuration T , we denote the label of v in T by $T(v)$. If a configuration has a single node only,⁶ it is denoted by the label of that node. That is, if a configuration T has a single node labeled (q, w, α) , then T is denoted by (q, w, α) .

At each computation step, a transition is applied to one stack-branch. If a branch empties, it cannot be chosen for the next transition (because it has no top symbol). If all sibling branches are empty, and each branch emptied with the *same* remaining input (i.e., after processing the same portion of the input) and with the same state, the branches are collapsed back to the parent branch.

Definition 9 Let A be an SAPDA and let T, T' be configurations of A . We say that T *yields* T' *in one step*, denoted $T \vdash_A T'$ (A is omitted if understood from the context), if one of the following holds.

- There exists a leaf node v in T , $T(v) = (q, \sigma w, X\alpha)$ and a transition $(q_1, \alpha_1) \wedge \cdots \wedge (q_k, \alpha_k) \in \delta(q, \sigma, X)$ which satisfy the conditions below.
 - If $k = 1$, then T' is obtained from T by relabeling v with $(q_1, w, \alpha_1\alpha)$.

⁶That is, the configuration tree consists of the root only, which is also the only leaf of the tree.

- If $k > 1$, then T' is obtained from T by relabeling v with α , and adding to it k child nodes v_1, \dots, v_k such that $T'(v_i) = (q_i, w, \alpha_i)$, $i = 1, \dots, k$.

In this case we say that the computation step is *based on* $(q_1, \alpha_1) \wedge \dots \wedge (q_k, \alpha_k)$ *applied to* v .

- There is a node v in T , $T(v) = \alpha$, that has k children v_1, \dots, v_k , all of which are leaves labeled the same (p, w, ϵ) , and T' is obtained from T by removing all leaf nodes v_i , $i = 1, \dots, k$ and relabeling v with (p, w, α) .

In this case we say that the computation step is *based on a collapsing of the child nodes of* v .

As usual, we denote by \vdash_A^* the reflexive and transitive closure of \vdash_A .

Definition 10 Let A be an SAPDA and let $w \in \Sigma^*$.

- The *initial* configuration of A on w is the configuration (q_0, w, \perp) .⁷
- An *accepting* configuration of A is a configuration of the form (q, ϵ, ϵ) .
- A *computation* of A on w is a sequence of configurations T_0, \dots, T_n , where
 - T_0 is the initial configuration,
 - $T_{i-1} \vdash_A T_i$ for $i = 1, \dots, n$, and
 - all leaves v of T_n are labeled (q, ϵ, α) , in particular, the entire input string has been read.
- An *accepting* computation of A on w is a computation whose last configuration T_n is accepting.

The language $L(A)$ of A is the set of all $w \in \Sigma^*$ such that A has an accepting computation on w .⁸

⁷That is, the initial configuration of A on w is a one node tree whose only node is labeled (q_0, w, \perp) .

⁸Alternatively, one can extend the definition of A with a set of *accepting* states $F \subseteq Q$ and define acceptance by accepting states, similarly to the classical definition. It can readily be seen that such an extension results in an equivalent model of computation.

Example 11 The SAPDA $A = (Q, \Sigma, \Gamma, \delta, q_0, \perp)$ defined below accepts the non-context-free language

$$\{w : |w|_a = |w|_b = |w|_c\}.$$

- $Q = \{q_0, q_1, q_2\}$,
- $\Sigma = \{a, b, c\}$,
- $\Gamma = \{\perp, a, b, c\}$, and
- δ is defined as follows.
 - $\delta(q_0, \epsilon, \perp) = \{(q_1, \perp) \wedge (q_2, \perp)\}$,
 - $\delta(q_1, \sigma, \perp) = \{(q_1, \sigma\perp)\}$, $\sigma \in \{a, b\}$,
 - $\delta(q_2, \sigma, \perp) = \{(q_2, \sigma\perp)\}$, $\sigma \in \{b, c\}$,
 - $\delta(q_1, \sigma, \sigma) = \{(q_1, \sigma\sigma)\}$, $\sigma \in \{a, b\}$,
 - $\delta(q_2, \sigma, \sigma) = \{(q_2, \sigma\sigma)\}$, $\sigma \in \{b, c\}$,
 - $\delta(q_1, \sigma', \sigma'') = \{(q_1, \epsilon)\}$, $(\sigma', \sigma'') \in \{(a, b), (b, a)\}$,
 - $\delta(q_2, \sigma', \sigma'') = \{(q_2, \epsilon)\}$, $(\sigma', \sigma'') \in \{(b, c), (c, b)\}$,
 - $\delta(q_1, c, X) = \{(q_1, X)\}$, $X \in \{\perp, a, b\}$,
 - $\delta(q_2, a, X) = \{(q_2, X)\}$, $X \in \{\perp, b, c\}$, and
 - $\delta(q_i, \epsilon, \perp) = \{(q_0, \epsilon)\}$, $i = 1, 2$.

The first step of the computation opens two branches, one for verifying that the number of *as* in the input word equals to the number of *bs*, and the other for verifying that the number of *bs* equals to the number of *cs*. If both branches manage to empty their stack then the word is accepted.

Figure 1 shows the contents of the stack tree at an intermediate stage of a computation on the word *abbccaab*.

The left branch has read *abbccc* and indicates that one more *bs* than *as* have been read, while the right branch has read *abb* and indicates that two more *cs* than *bs* have been read. Figure 2 shows the configuration corresponding the above computation stage of the automaton.

We now consider the following example of an SAPDA which accepts the non-context-free language $\{w\$uw : w, u \in \{a, b\}\}$. Note that the intersection of this language with $\{u\$v : u, v \in \{a, b\} \wedge |u| = |v|\}$ is the *reduplication*

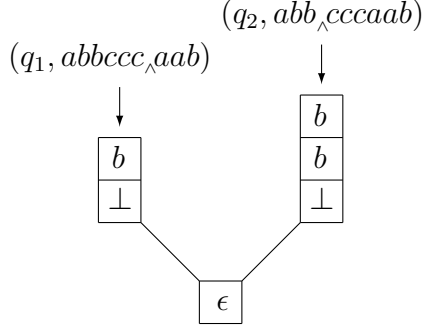


Figure 1: Intermediate stage of a computation on $abbccaab$.

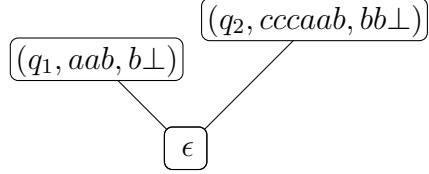


Figure 2: The configuration corresponding to Figure 1.

with a center marker language. As the latter language is context-free, and SAPDA are closed under intersection, the construction can easily be modified to accept the reduplication language.

The example is of particular interest as it showcases the model's ability to utilize recursive conjunctive transitions, allowing it to accept languages which are not finite intersections of context-free languages. Moreover, the example gives additional intuition towards understanding Okhotin's grammar for the reduplication language as presented in Example 6. The below automaton accepts the language derived by the non-terminal D in the grammar.

Example 12 (cf. Example 6) The SAPDA $A = (Q, \Sigma, \Gamma, \delta, q_0, \perp)$ defined below accepts the non-context-free language

$$\{w\$uw : w, u \in \{a, b\}^*\}.$$

- $Q = \{q_0, q_e\} \cup \{q_\sigma^1 : \sigma \in \{a, b\}\} \cup \{q_\sigma^2 : \sigma \in \{a, b\}\},$
- $\Sigma = \{a, b, \$\},$
- $\Gamma = \{\perp, \$\},$ and
- δ is defined as follows.
 1. $\delta(q_0, \sigma, \perp) = \{(q_\sigma^1, \perp) \wedge (q_0, \perp)\}, \sigma \in \{a, b\}$
 2. $\delta(q_\sigma^1, \tau, X) = \{(q_\sigma^1, \$X)\}, \sigma, \tau \in \{a, b\}, X \in \Gamma,$
 3. $\delta(q_0, \$, \perp) = \{(q_e, \epsilon)\},$

4. $\delta(q_\sigma^1, \$, X) = \{(q_\sigma^2, X)\}$, $\sigma \in \{a, b\}$, $X \in \Gamma$,
5. $\delta(q_\sigma^2, \tau, X) = \{(q_\sigma^2, X)\}$, $\sigma, \tau \in \{a, b\} : \sigma \neq \tau$, $X \in \Gamma$,
6. $\delta(q_\sigma^2, \sigma, X) = \{(q_\sigma^2, X), (q_e, X)\}$, $\sigma \in \{a, b\}$, $X \in \Gamma$,
7. $\delta(q_e, \sigma, \$) = \{(q_e, \epsilon)\}$, $\sigma \in \{a, b\}$,
8. $\delta(q_e, \epsilon, \perp) = \{(q_e, \epsilon)\}$

The computations of the automaton have two main phases: before and after the \$ sign is encountered in the input. In the first phase, each input letter σ which is read leads to a conjunctive transition (transition 1) that opens two new stack-branches. One new branch continues the recursion, while the second checks that the following condition is met.

Assume σ is the n -th letter from the \$ sign. If so, the new stack branch opened during the transition on σ will verify that the n -th letter from the end of the input is also σ . This way, if the computation is accepting, the word will in fact be of the form $w\$uw$. To be able to check this property, the branch must know σ and σ 's relative position (n) to the \$ sign. To “remember” σ , the state of the branch head is q_σ^1 (the 1 superscript denoting that the computation is in the first phase). To find n , the branch adds a \$ sign to its stack for each input character read (transition 2), until the \$ is encountered in the input. Therefore, when the \$ is read, the number of \$s in the stack branch will be the number of letters between σ and the \$ sign in the first half of the input word.

Once the \$ is read, the branch perpetuating the recursion ceases to open up new branches, and instead transitions to q_e and empties its stack (transition 3). All the other branches denote that they have moved to the second phase of the computation by transitioning to states q_σ^2 (transition 4). From this point onward, each branch “waits” to see the σ encoded in its state in the input (transition 5). Once it does encounter σ , it can either ignore it and continue to look for another σ in the input (in case there are repetitions in w of the same letter), or it can “guess” that this is the σ which is n letters from the end of the input, and move to state q_e (transition 6).

After transitioning to q_e , one \$ is emptied from the stack for every input character read. If in fact σ was the right number of letters from the end, the \perp sign of the stack branch will be exposed exactly when the last input letter is read. At this point, an ϵ -transition is applied which empties the stack branch (transition 8).

If all branches successfully “guess” their respective σ symbols then the computation will reach a configuration where all leaf nodes are labeled $(q_e, \epsilon, \epsilon)$. From here, successive branch collapsing steps can be applied until an accepting configuration is reached.

Consider a computation on the word $abb\$babb$. Figure 3 shows the contents of the stack tree after all branches have read the prefix ab . The rightmost branch is the branch perpetuating the recursion. The leftmost branch remembers seeing a in the input, and has since counted one letter. The middle branch remembers seeing b in the input, and has not yet counted any letters.

Figure 4 shows the contents of the stack tree after all branches have read the prefix $abb\$bab$. The rightmost branch, has stopped perpetuating the recursion, transitioned to q_e , and emptied its stack. The leftmost branch correctly “guessed” that the a read was the a it was looking for. Subsequently, it transitioned to q_e and removed one $\$$ from its stack for the b that was read afterwards. The second branch from the left correctly ignored the first b after the $\$$ sign, and only transitioned to q_e after reading the second b . The second branch from the right is still waiting to find the correct b , and is therefore still in state q_b^2 .

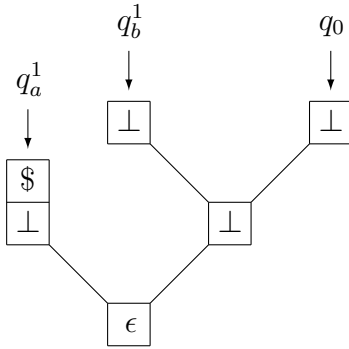


Figure 3: Stack tree contents after all branches have read ab .

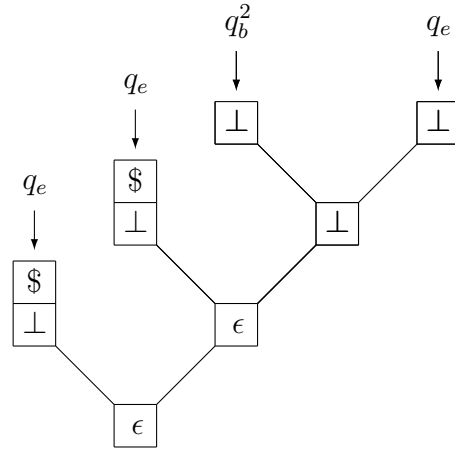


Figure 4: Stack tree contents after all branches have read $abb\$ba$.

4 Equivalence of CG and SAPDA

In this section we present the main results of our paper – equivalence of the CG and SAPDA models.

Theorem 13 *A language is generated by a CG if and only if it is accepted by a SAPDA.*

The proof of Theorem 13 is presented in Sections 4.1 (the “if” part) and 4.2 (the “only if” part). It is an extension of the classical one, see, e.g., [3, pp. 115–119], but, in contrast to the classical case, the proof of the “only if” part of the theorem is more involved and requires several preliminary steps.

4.1 Proof of the “if” part of Theorem 13

Let $A = (Q, \Sigma, \Gamma, \delta, q_0, \perp)$ be an SAPDA. Consider the CG $G_A = (V, \Sigma, P, S)$, where

- $V = Q \times \Gamma \times Q \cup \{S\}$,⁹ and
- P contains the following rules.
 - $S \rightarrow [q_0, \perp, p]$, for all $p \in Q$, and
 - $[q, X, p] \rightarrow \sigma([p_1, Y_{1,1}, q_{1,2}] \cdots [q_{1,m_1}, Y_{1,m_1}, p] \& \cdots$
 $\cdots \& [p_k, Y_{k,1}, q_{k,2}] \cdots [q_{k,m_k}, Y_{k,m_k}, p])$
 for all $q \in Q$, $\sigma \in \Sigma \cup \{\epsilon\}$, $X \in \Gamma$, all transitions

$$(p_1, Y_{1,1} \cdots Y_{1,m_1}) \wedge \cdots \wedge (p_k, Y_{k,1} \cdots Y_{k,m_k}) \in \delta(q, \sigma, X),$$

and all $q_{i,j} \in Q$, $i = 1, \dots, k$, $j = 1, \dots, m_i$.

The proof of the equality $L(G_A) = L(A)$ is based on Proposition 14 below.

Proposition 14 *Let $w \in \Sigma^*$. Then $(q, w, X) \vdash_A^* (p, \epsilon, \epsilon)$ if and only if $[q, X, p] \Rightarrow^* w$.*

⁹Renaming S , if necessary, we may assume that $S \notin Q \times \Gamma \times Q$.

Proof We start with the proof of the “only if” part of the proposition by induction on the number of computation steps of A .

Basis: If X is emptied in one step, then $w = \sigma \in \Sigma \cup \{\epsilon\}$ and $(p, \epsilon) \in \delta(q, \sigma, X)$. By the definition of P , $[q, X, p] \rightarrow \sigma$ is a rule of G_A and, therefore, $[q, X, p] \Rightarrow \sigma$.

Induction step: Assume the proposition holds for all computations of A shorter than n and let $(q, \sigma w, X) \vdash_A^n (p, \epsilon, \epsilon)$, where $\sigma \in \Sigma \cup \{\epsilon\}$. Let the first transition applied be

$$(p_1, Y_{1,1} \cdots Y_{1,m_1}) \wedge \cdots \wedge (p_k, Y_{k,1} \cdots Y_{k,m_k}) \in \delta(q, \sigma, X)$$

and let v_1, \dots, v_k be the k new leaf nodes added to the (root) configuration.¹⁰ The last step of the computation empties these branches, as the computation terminates as soon as they are collapsed. Therefore, steps 2 through $n - 1$ of the computation empty all v_i sub-trees for $i = 1, \dots, k$, and, because of the synchronization constraint, emptying each tree “consumes” the entire remaining portion w of the input. Each transition in this part of the computation is applied to one of the v_i sub-trees. Since there is no interaction between these sub-trees, we may assume that the first sub-tree v_1 is emptied in the first n_1 steps, then the next sub-tree v_2 is emptied in the next n_2 steps, and so forth until the last sub-tree v_k is emptied in the last n_k steps, after which the last step of the derivation collapses all sub-trees v_i , $i = 1, \dots, k$. Since $n_1 + \cdots + n_k = n - 2$, each n_i is smaller than n .

Using the same argument as in the corresponding classical proof, see [3, Theorem 5.4, pp. 116–119], we obtain that for each sub-tree v_i , $i = 1, \dots, k$, there exist states $q_{i,1}, \dots, q_{i,m_i+1}$, $q_{i,1} = p_i$, such that for each $i = 1, \dots, m_i$, the sub-computation emptying the sub-tree of v_i uses sub-computations of the form

$$(q_{i,j}, w_{i,j} \cdots w_{i,m_i}, Y_{i,j} \cdots Y_{i,m_i}) \vdash_A^* (q_{i,j+1}, w_{i,j+1} \cdots w_{i,m_i}, Y_{i,j+1} \cdots Y_{i,m_i}),$$

where $Y_{i,j+1}$ is exposed as the top symbol of the stack branch only at the last step of the computation. Therefore,

$$(q_{i,j}, w_{i,j}, Y_{i,j}) \vdash_A^* (q_{i,j+1}, \epsilon, \epsilon).$$

¹⁰Here and hereafter we assume that $k \geq 2$, because the case of $k = 1$ is treated exactly as in the proofs of [3, Theorems 5.3 and 5.4, pp. 115–119].

Each of these sub-computations is shorter than n . Hence, by the induction hypothesis, $[q_{i,j}, Y_{i,j}, q_{i,j+1}] \Rightarrow^* w_{i,j}$. It follows that

$$[p_1, Y_{i,1}, q_{i,2}] \cdots [q_{i,m_i}, Y_{i,m_i}, q_{m_i+1}] \Rightarrow^* w_{i,1} \cdots w_{i,m_i} = w_i$$

for some subword w_i of w .

The last computation step empties all sub-trees v_i , $i = 1, \dots, k$. Therefore, all states q_{m_i+1} , $i = 1, \dots, k$, are the same state p , and all w_i are w , because all branches begin with w as the remaining input and end with ϵ . Applying the rule

$$\begin{aligned} [q, X, p] \rightarrow \sigma([p_1, Y_{1,1}, q_{1,2}] \cdots [q_{1,m_1}, Y_{1,m_1}, p]) \& \cdots \\ \cdots \& [p_k, Y_{k,1}, q_{k,2}] \cdots [q_{k,m_k}, Y_{k,m_k}, p]) \end{aligned}$$

we obtain the desired derivation

$$[q, X, p] \Rightarrow^* \sigma(w \& \cdots \& w) \Rightarrow \sigma w.$$

We prove the “if” part of the proposition by induction on the derivation length.

Basis: The only one-step derivations deriving a terminal word in G_A are of the form $[q, X, p] \Rightarrow \sigma$, where $\sigma \in \Sigma \cup \{\epsilon\}$. By the definition of P , $(\epsilon, p) \in \delta(q, \sigma, X)$, implying $(q, \sigma, X) \vdash_A (p, \epsilon, \epsilon)$.

Induction step: Assume the proposition holds for all derivations of length less than n and let $[q, X, p] \Rightarrow^n \sigma w$, where the first step of the derivation is

$$[q, X, p] \rightarrow \sigma([p_1, Y_{1,1}, q_{1,2}] \cdots [q_{1,m_1}, Y_{1,m_1}, p]) \& \cdots$$

$$\cdots \& [p_k, Y_{k,1}, q_{k,2}] \cdots [q_{k,m_k}, Y_{k,m_k}, p])$$

and, therefore, the last step of the derivation is $\sigma(w \& \cdots \& w) \Rightarrow \sigma w$. It follows that each above conjunct derives w in less than n steps. Therefore, for each $i = 1, \dots, k$, each variable $[q_{i,j}, Y_{i,j}, p_{i,j+1}]$, $i = 1, \dots, m_i$, where $q_{i,1} = p_i$, derives a word $w_{i,j}$ such that $w_{i,1} \cdots w_{i,m_i} = w$. Each of these derivations is shorter than n . Therefore, by the induction hypothesis, there exist sub-computations of the form

$$(q_{i,j}, w_{i,j}, Y_{i,j}) \vdash_A^* (q_{i,j+1}, \epsilon, \epsilon).$$

By combining these together, we obtain

$$(p_i, w, Y_{i,1} \cdots Y_{i,m_i}) \vdash_A^* (p, \epsilon, \epsilon). \quad (1)$$

From the first derivation step used and the definition of P ,

$$(p_1, Y_{1,1} \cdots Y_{1,m_1}) \wedge \cdots \wedge (p_k, Y_{k,1} \cdots Y_{k,m_k}) \in \delta(q, \sigma, X).$$

Therefore, there is a transition from the configuration $(q, \sigma w, X)$ to a tree with a root labeled ϵ and k root children, each labeled $(p_i, w, Y_{i,1} \cdots Y_{i,m_i})$, $i = 1, \dots, k$. In this tree, each branch, from left to right, can be emptied using (1). When the last branch empties, all sibling branches are in state p with no remaining input. Therefore, the branches are collapsed and the automaton's last configuration is (p, ϵ, ϵ) . \blacksquare

Let $w \in L(A)$. That is, there exists an accepting computation $(q_0, w, \perp) \vdash_A^* (p, \epsilon, \epsilon)$, $p \in Q$, of A on w . By Proposition 14, $[q_0, \perp, p] \Rightarrow^* w$. Using the rule $S \rightarrow [q_0, \perp, p]$, we obtain $S \Rightarrow^* w$, implying $w \in L(G_A)$.

Let $w \in L(G_A)$. That is, $S \Rightarrow^* w$. By the definition of P , the first step of the derivation is of the form $S \Rightarrow [q_0, \perp, p]$. Therefore, $[q_0, \perp, p] \Rightarrow^* w$. By Proposition 14, A has an accepting computation on w , implying $w \in L(A)$.

Consequently, $L(A) = L(G_A)$ and the proof of the “if” part of Theorem 13 is complete.

4.2 Proof of the “only if” part of Theorem 13

For the purposes of our proof we assume that grammars do not contain ϵ -rules. Okhotin proved in [6] that it is possible to remove such rules from the grammar, with the exception of an ϵ -rule from the start symbol in the case where ϵ is in the grammar language. We also assume that the start symbol does not appear in the right-hand side of any rule. This can be achieved by augmenting the grammar with a *new* start symbol S' , as is done in the classical case.

Let $G = (V, \Sigma, P, S)$ be a conjunctive grammar. Consider the single-state SAPDA $A_G = (\Sigma, \Gamma, \delta, q_0, \perp)$,¹¹ where

- $\Gamma = V \cup \Sigma$ and $\perp = S$,

¹¹We omit the state component of both A_G and δ .

- $\delta(\epsilon, A) = \{\alpha_1 \wedge \cdots \wedge \alpha_k : A \rightarrow (\alpha_1 \& \cdots \& \alpha_k) \in P\}$,¹² and
- $\delta(\sigma, \sigma) = \{\epsilon\}$.

We shall prove that $L(A_G) = L(G)$.

The proof is an extension of the classical one, see, e.g., [3, Theorem 5.3, pp. 115–116]. In the classical proof, a correlation is shown between the input and stack contents of the automaton and the leftmost sentential forms of the grammar. In the case of SAPDA, the stack is a tree. Therefore, as a preliminary step, we define a leftmost derivation, a tree representation, and a simplified form for conjunctive formulas. These definitions will later assist us in showing a similar correlation.

Definition 15 The Σ -*prefix* of a word $\alpha \in (V \cup \Sigma)^*$ is the longest prefix of α belonging to Σ^* .

The following definition introduces the basic notions needed for the proof of the “only if” part of Theorem 13.

Definition 16 *Leftmost* conjunctive formulas and *leftmost* derivations are defined by the following simultaneous recursion.

- The elements of $(V \cup \Sigma)^+$ are leftmost (conjunctive) formulas and one-step derivations from them in which a rule is applied to their leftmost variable are leftmost derivations.
- Let \mathcal{A} be a leftmost formula. Then a one-step leftmost derivation from \mathcal{A} is either contraction or a rule applied to the leftmost variable of a simple conjunct of \mathcal{A} .

The one-step leftmost derivation relation is denoted \Rightarrow_L , and, as usual, its reflexive and transitive closure is denoted \Rightarrow_L^* .¹³

We shall need the following alternative description of leftmost formulas.

Lemma 17 *Each leftmost formula is of the form $u(\mathcal{A}_1 \& \cdots \& \mathcal{A}_k)\alpha$, where $u \in \Sigma^*$, all \mathcal{A}_i s, $i = 1, \dots, k$ are leftmost, and $\alpha \in (V \cup \Sigma)^*$.*

¹²Since G does not contain ϵ -rules, $\alpha_1, \dots, \alpha_k \in \Gamma^+$.

¹³Note that leftmost formulas are defined with respect to the underlying conjunctive grammar.

Remark 18 In particular, leftmost formulas do not contain subformulas of the form $(\mathcal{A}'_1 \& \cdots \& \mathcal{A}'_{k'})\mathcal{A}(\mathcal{A}''_1 \& \cdots \& \mathcal{A}''_{k''})$.

Note that leftmost formulas and leftmost derivations have been defined by “recursion from the end,” whereas Lemma 17 describes leftmost formulas by “recursion from the beginning.”

Proof(of Lemma 17) Let \mathcal{A} be a leftmost formula and let $\alpha \in (V \cup \Sigma)^*$ be such that $\alpha \Rightarrow_L^* \mathcal{A}$. The proof is by induction on the length n of a leftmost derivation of \mathcal{A} from α .

Basis: $n = 0$. Then $\mathcal{A} \in (V \cup \Sigma)^+$, and the lemma follows with $k = 1$, $u = \epsilon$, $\mathcal{A}_1 = \mathcal{A}$, and $\alpha = \epsilon$.

Induction step: Assume that the proposition is true for $n \geq 1$ and let $\alpha \Rightarrow_L^{n+1} \mathcal{A}$. Then,

$$\alpha \Rightarrow_L \mathcal{B}_1 \Rightarrow_L \mathcal{B}_2 \Rightarrow_L \cdots \Rightarrow_L \mathcal{B}_n \Rightarrow_L \mathcal{A}.$$

If for some $i = 1, \dots, n$, $\mathcal{B}_i \in (V \cup \Sigma)^+$, then, by the induction hypothesis, \mathcal{A} is in the desired form, because $\mathcal{B}_i \Rightarrow_L^{\leq n} \mathcal{A}$.

Otherwise, $\mathcal{B}_1 = u(\alpha_1 \& \cdots \& \alpha_k)\alpha'$, and $\mathcal{B}_n = u(\mathcal{A}_1 \& \cdots \& \mathcal{A}_k)\alpha'$, where $\alpha_i \Rightarrow_L^* \mathcal{A}_i$, $i = 1, \dots, k$. Thus, by the induction hypothesis, all \mathcal{A}_i , $i = 1, \dots, k$, are leftmost. Since \mathcal{A} is not derived from \mathcal{B}_n by contraction, $\mathcal{A} = u(\mathcal{A}'_1 \& \cdots \& \mathcal{A}'_k)\alpha'$, where for some $m = 1, \dots, k$, $\mathcal{A}_m \Rightarrow_L \mathcal{A}'_m$, and $\mathcal{A}'_i = \mathcal{A}_i$ for $i \neq m$. Thus, all \mathcal{A}'_i , $i = 1, \dots, k$, are leftmost and the proof is complete. \blacksquare

Similarly to context-free grammars, in CGs, the *order* in which independent derivation rules are applied, does not affect the derived word. Therefore, we have the following lemma.

Lemma 19 Let $\alpha \in (V \cup \Sigma)^*$ and $w \in \Sigma^+$ be such that $\alpha \Rightarrow^* w$. Then $\alpha \Rightarrow_L^* w$.

Next, we define the *simplified form* of a leftmost formula.

Definition 20 Let \mathcal{A} be a leftmost formula. The *simplified form* of \mathcal{A} , denoted $\mathcal{S}(\mathcal{A})$ is defined by the following recursion.

- If $\mathcal{A} = \alpha$ for some $\alpha \in (V \cup \Sigma)^*$, then $\mathcal{S}(\alpha) = \alpha$.
- Else, $\mathcal{S}(u(\mathcal{A}_1 \& \cdots \& \mathcal{A}_k)\alpha)$ is $(\mathcal{S}(u\mathcal{A}_1) \& \cdots \& \mathcal{S}(u\mathcal{A}_k))\alpha$, cf. Lemma 17 and Remark 18.

Note that, by Lemma 17, Definition 20 covers all possible leftmost formulas.

Intuitively, the simplified form of a leftmost formula is obtained by pushing all prefixes of enclosing subformulas, which, by Lemma 17, are words over Σ , into the level of the simple conjuncts. For example, the simplified form of the formula

$$ab(cA \ \& \ c(dB \ \& \ dD))ABac$$

is the formula

$$(abcA \ \& \ (abcdB \ \& \ abcdD))ABac$$

obtained by “pushing in” ab and c .

Remark 21 Note that to each simple conjunct of a leftmost formula \mathcal{A} corresponds a simple conjunct of $\mathbf{S}(\mathcal{A})$ and vice versa. The correspondence is defined by the following recursion.

- Let $\mathcal{A} = \alpha \in (V \cup \Sigma)^*$. By Definition 20, $\mathbf{S}(\alpha) = \alpha$. Therefore, the only simple conjunct of \mathcal{A} , α , corresponds to itself.
- Let \mathcal{A} be of the form $u(\mathcal{A}_1 \ \& \ \dots \ \& \ \mathcal{A}_k)\alpha$, where $u \in \Sigma^*$ and all conjuncts $\mathcal{A}_1, \dots, \mathcal{A}_k$ are leftmost, and $\alpha \in (V \cup \Sigma)^*$, and let $\beta \in (V \cup \Sigma)^*$ be a simple conjunct of \mathcal{A} . Then, for some $i = 1, \dots, k$, β is a simple conjunct of $u\mathcal{A}_i$ and its corresponding simple conjunct in $\mathbf{S}(u\mathcal{A}_i)$ is also the corresponding simple conjunct in $\mathbf{S}(\mathcal{A})$.

In addition, as simplification “pushes in” Σ -prefixes, it follows that the simple conjunct of $\mathbf{S}(\mathcal{A})$ corresponding to the simple conjunct β of \mathcal{A} is of the form $v\beta$, where $v \in \Sigma^*$.

In particular, if \mathcal{A} is of the form $u(\alpha_1 \ \& \ \dots \ \& \ \alpha_k)\alpha$, where $u \in \Sigma^*$ and $\alpha, \alpha_i \in (V \cup \Sigma)^*$, $i = 1, \dots, k$, then the simple conjunct of $\mathbf{S}(\mathcal{A})$ corresponding to the simple conjunct α_i of \mathcal{A} is $u\alpha_i$, $i = 1, \dots, k$.

For any leftmost formula, the simplification process always terminates with a unique result. Simplification can be easily shown to maintain the language of the formula, and, therefore, we have the following lemma.

Lemma 22 *If \mathcal{A} is a leftmost formula, then $\mathbf{S}(\mathcal{A})$ is unique and $L(\mathbf{S}(\mathcal{A})) = L(\mathcal{A})$.*

Lemma 23 *Let \mathcal{A} be a leftmost formula and let $w \in L(\mathcal{A})$. Then the Σ -prefixes of the simple conjuncts of $\mathbf{S}(\mathcal{A})$ are prefixes of w .*

The proof of the lemma easily follows from Definitions 4 and 20 and is omitted.

As we want to show a correlation between leftmost formulas and the stack tree structure, we must define a tree structure for the former as well. The tree structure of leftmost formulas is naturally motivated by Definition 20.

Definition 24 The *tree representation* of a leftmost formula \mathcal{A} , denoted $T(\mathcal{A})$, is recursively defined as follows:

- If $\mathcal{A} = \alpha$, where $\alpha \in (V \cup \Sigma)^*$, then $T(\mathcal{A})$ has a single node labeled α .
- If $\mathcal{A} = u(\mathcal{A}_1 \& \cdots \& \mathcal{A}_k)\alpha$, cf. Lemma 17, then $T(\mathcal{A})$ consists of a root node labeled α with the tree representations of $\mathcal{S}(u\mathcal{A}_i)$, $i = 1, \dots, k$, appended as children, see Figure 5.

Again, by Lemma 17, Definition 24 covers all leftmost formulas.

Example 25 The simplified form of the leftmost formula

$$ab(cA \& (cdB \& cC)D \& cdE)FG$$

is

$$(abcA \& (abcdB \& abcC)D \& abcdE)FG$$

and, therefore, its tree representation is as shown in Figure 6.¹⁴

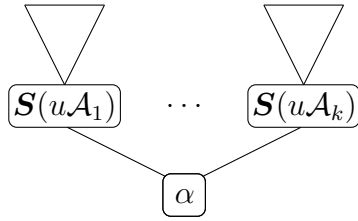


Figure 5: Tree representation of $u(\mathcal{A}_1 \& \cdots \& \mathcal{A}_k)\alpha$.

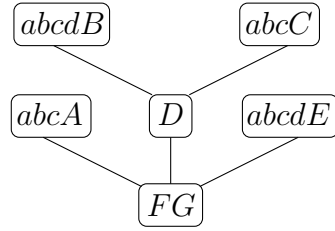


Figure 6: Tree representation of $ab(cA \& (cdB \& cC)D \& cdE)FG$.

The proof of the “only if” part of Theorem 13 is based on Propositions 26 and 31 below.

¹⁴Note that the Σ -prefixes of the simple conjuncts in this formula are prefixes of the same word $abcd$. It is so, because the language derived from this formula is non-empty.

Proposition 26 *Let $u \in \Sigma^*$, $T_0 = (u, \perp)$, and let $T_0 \vdash^* T$. Let the label of a leaf v of T be (u_v, α_v) , and let y_v such that $y_v u_v = u$.¹⁵ If for some leaf v' , $u_{v'} = \epsilon$, then there exists a formula \mathcal{A} such that $S \Rightarrow_L^* \mathcal{A}$, and relabeling each leaf v in T with $y_v \alpha_v$ results in $T(\mathcal{A})$.*

We say that the above formula \mathcal{A} *matches* T .

Proof The proof is by induction on the length of the computation of A_G .

Basis: If the number of computation steps is zero, then $u = \epsilon$ and $T = S$. Therefore, the proposition holds for $\mathcal{A} = S$, and the zero-step derivation.

Induction step: Assume the proposition holds for every computation shorter than n . Let $T_0 \vdash^{n-1} T_{n-1} \vdash T_n$ be a computation on u such that T_n satisfies the proposition prerequisites.

If $T_{n-1} \vdash T_n$ by an ϵ -transition, then, since no input was read, the set of the remaining input components of the labels of the leaves of T_{n-1} equals to that of the leaves of T_n . Therefore, T_{n-1} also satisfies the proposition prerequisites for u , that is, for some leaf v' , $u_{v'} = \epsilon$. Thus, by the induction hypothesis, there exists a leftmost formula \mathcal{B} matching T_{n-1} such that $S \Rightarrow_L^* \mathcal{B}$. Since the last computation step of A_G on u is an ϵ -transition, it is of the form $\alpha_1 \wedge \dots \wedge \alpha_k \in \delta(\epsilon, A)$ that is applied to a leaf v of T_{n-1} labeled $(u_v, A\alpha'_v)$, where $A\alpha'_v = \alpha_v$, say. Therefore, in T_n , v is an internal node labeled α'_v and it has k child nodes labeled (u_v, α_i) , $i = 1, \dots, k$. Since \mathcal{B} matches T_{n-1} , the node v in $T(\mathcal{B})$ is labeled $y_v A\alpha'_v$. By the definition of δ , $A \rightarrow (\alpha_1 \& \dots \& \alpha_k) \in P$, and by applying this rule to the corresponding A in \mathcal{B} we obtain a leftmost formula \mathcal{A} matching T_n such that $S \Rightarrow_L^* \mathcal{A}$. This is so, because T_{n-1} and T_n have the same tree structure, the “matching” labels of their nodes are the same “modulo v and its child leaves,” the label v in $T(\mathcal{A})$ is α'_v , and the labels of the child leaves of v in $T(\mathcal{A})$ are $y_v \alpha_i$, $i = 1, \dots, k$.

If $T_{n-1} \vdash T_n$ by a transition on some $\sigma \in \Sigma$, then this transition is $\delta(\sigma, \sigma) = \{\epsilon\}$. Therefore, the labels of all leaves in T_n are the same as those in T_{n-1} , except for leaf v (to which the transition is applied) whose label changed from $(\sigma u_v, \sigma \alpha_v)$ to (u_v, α_v) . We examine the possible cases.

- In T_{n-1} there is at least one leaf whose remaining input is empty. In this case, T_{n-1} satisfies the proposition prerequisites for u , and by the

¹⁵Such u_v s exists, because, by the definition of \vdash (Definition 9), all u_v s are suffixes of u .

induction hypothesis, there exists a leftmost formula \mathcal{A} matching T_{n-1} such that $S \Rightarrow_L^* \mathcal{A}$. We contend that \mathcal{A} matches T_n as well. Since T_{n-1} and T_n have the same tree structure and the labels of their nodes different from v are the same, it suffices to show that the label of v in $T(\mathcal{A})$ is $y_v \alpha_v$.

The label of v in T_{n-1} is $(\sigma u_v, \sigma \alpha_v)$. Therefore, by the induction hypothesis, its label in $T(\mathcal{A})$ is $y'_v \sigma \alpha_v$, where $y'_v \sigma u_v = u$. That is, $y_v = y'_v \sigma$, implying $y'_v \sigma \alpha_v = y_v \alpha_v$, and the contention follows.

- In T_{n-1} all leaves have (non-empty) input left to read, and let $u = u' \sigma$. Then, T_{n-1} satisfies the proposition prerequisites for u' , because the remaining input at the leaf v is empty. By the induction hypothesis, there exists a leftmost formula \mathcal{A} matching T_{n-1} such that $S \Rightarrow_L^* \mathcal{A}$.

In T_n , $u_v = \epsilon$, implying $y_v = u = u' \sigma$. Again, since T_{n-1} and T_n have the same tree structure and the labels of their nodes different from v are the same, we have only to show that the label of v in $T(\mathcal{A})$ is $y_v \alpha_v$. Indeed, since the label of v in T_{n-1} is $(\epsilon, \sigma \alpha_v)$, its label in $T(\mathcal{A})$ is $u' \sigma \alpha_v = u \alpha_v = y_v \alpha_v$.

If $T_{n-1} \vdash T_n$ collapses a set of sibling leaves in T_{n-1} , then these leaves have the same label (u', ϵ) , $u' \in \Sigma^$. In addition, since no input was read in the transition, T_{n-1} satisfies the proposition prerequisites for u . Thus, by the induction hypothesis, there exists a leftmost formula \mathcal{B} matching T_{n-1} such that $S \Rightarrow_L^* \mathcal{B}$. In particular, the leaves in $T(\mathcal{B})$ “matching” the leaves of T_{n-1} collapsed in the computation step $T_{n-1} \vdash T_n$ of A_G , are all labeled the same word $y' \in \Sigma^*$ such that $y' u' = u$. Therefore, \mathcal{B} contains a subformula of the form $(w' \& \cdots \& w')$ for some suffix w' of y' .¹⁶ Let \mathcal{A} be the leftmost formula obtained from \mathcal{B} by applying contraction to this subformula $(w' \& \cdots \& w')$. Then, $S \Rightarrow_L^* \mathcal{A}$ and $T(\mathcal{A})$ matches T_n . Like in the case of an ϵ -transition, this is so, because T_{n-1} and T_n have the same tree structure and the “matching” labels of their nodes are the same “modulo v and its child leaves.” ■*

We precede the second auxiliary proposition (Proposition 31) with a number of necessary definitions and related results.

Definition 27 below is motivated by Remark 21.

¹⁶Actually, the y' 's are the corresponding conjuncts of w' in $\mathcal{S}(\mathcal{B})$.

Definition 27 Let \mathcal{A} be a leftmost formula. The Σ -*prefix* of \mathcal{A} , denoted $M(\mathcal{A})$, is the longest word over Σ that is a prefix of each simple conjunct in $S(\mathcal{A})$.

For example, the Σ -prefix of $ab(cA \& c(dE \& dF))A$ is abc . It is not $abcd$, because the latter is not a prefix of the leftmost simple conjunct $abcA$ in the simplified form of the formula.

It follows from the definition that, for all leftmost formulas \mathcal{A} and \mathcal{B} such that $\mathcal{A} \Rightarrow_L^* \mathcal{B}$, $M(\mathcal{A})$ is a prefix of $M(\mathcal{B})$. In particular, if $\mathcal{A} \Rightarrow^* w \in \Sigma^*$, then $M(\mathcal{A})$ is a prefix of w .

Lemma 28 Let \mathcal{A} be a leftmost formula such that $L(\mathcal{A}) \neq \emptyset$. Then $M(\mathcal{A})$ is the shortest Σ -prefix of a simple conjunct in $S(\mathcal{A})$.

Proof Let $w \in L(\mathcal{A})$. By Lemma 23, all Σ -prefixes of the simple conjuncts in $S(\mathcal{A})$ are prefixes of w . Thus, the shortest Σ -prefix of a simple conjunct in $S(\mathcal{A})$ is a prefix of all other Σ -prefixes and, therefore, is $M(\mathcal{A})$. ■

Finally, we shall need the notion of *strong* leftmost derivation.

Definition 29 We say that a leftmost derivation is *strong*, if at each step the rule or contraction is applied to a simple conjunct α whose corresponding conjunct in the simplified form of the formula has the shortest Σ -prefix. Namely, the derivation steps are as follows.

- If α is a terminal word, i.e., $\alpha \in \Sigma^+$,¹⁷ then contraction is applied on the enclosing formula.

Note that, if contraction cannot be applied, then, in the enclosing formula, either all conjuncts are terminal words, but not the same, or there is a simple conjunct α' containing a variable. Since there are no ϵ -rules and α has the shortest Σ -prefix, $\alpha' = \alpha\beta$, for some formula $\beta \in (V \cup \Sigma)^+$. Therefore, α will be a proper prefix of any terminal word derived from α' . Either way, the language of the enclosing formula is empty, and the case can be disregarded.

- Else, a production rule is applied to the leftmost variable of α .

¹⁷Recall that we assume that the grammar contains no ϵ -rules.

The one-step strong leftmost derivation relation is denoted \Rightarrow_{SL} , and, as usual, its reflexive and transitive closure is denoted \Rightarrow_{SL}^* .

The idea behind the notion of strong leftmost derivations is to always develop one of the least developed simple conjuncts, which are those with “the shortest Σ -prefix in the simplified form.” By Proposition 31 below, this type of derivation is correlated with the computation of the SAPDA constructed from the grammar.

Again, similarly to context-free grammars, in CGs, the *order* in which independent derivation rules are applied, does not affect the derived word. Therefore, the following lemma immediately follows from Lemma 19.

Lemma 30 *Let $\alpha \in (V \cup \Sigma)^*$ and $w \in \Sigma^+$ be such that $\alpha \Rightarrow^* w$. Then $\alpha \Rightarrow_{SL}^* w$.*

Proposition 31 *Let $S \Rightarrow_{SL}^* \mathcal{A}$, $L(\mathcal{A}) \neq \emptyset$, and let $u = \mathbf{M}(\mathcal{A})$. Then there exists a computation of A_G on u ending in a configuration T such that all leaves v of T are labeled (ϵ, α) , $\alpha \in (V \cup \Sigma)^*$, and replacing each such label with $u\alpha$, results in the tree representation of \mathcal{A} .*

We denote the result of the above transformation of T by $u \cdot T$. Thus, the proposition states that $u \cdot T = T(\mathcal{A})$.

Proof The proof is by induction on the length of the leftmost derivation of \mathcal{A} from S .

Basis: In zero derivation steps, \mathcal{A} is S and, therefore, $u = \epsilon$. Thus, the proposition holds for the zero-step computation (with only the initial configuration).

Induction step: Assume the proposition holds for all leftmost derivations shorter than n and let $S \Rightarrow_{SL}^n \mathcal{A}$. It follows that there exists a leftmost formula \mathcal{B} such that $S \Rightarrow_{SL}^{n-1} \mathcal{B} \Rightarrow_{SL} \mathcal{A}$. Let $u = \mathbf{M}(\mathcal{B})$. By the induction hypothesis, there exists a computation of A_G on u ending in a configuration T such that $u \cdot T = T(\mathcal{B})$. We examine the different cases based on the rule used in the last derivation step $\mathcal{B} \Rightarrow_{SL} \mathcal{A}$.

- Assume that \mathcal{A} is derived from \mathcal{B} by *contraction*, and let the contracted formula be of the form $(w \ \& \ \cdots \ \& \ w)$. Since contraction is applied to the formula with the shortest Σ -prefix, which by Lemma 28, is $u = \mathbf{M}(\mathcal{B})$, $u = vw$, for some $v \in \Sigma^*$. Thus, in $T(\mathcal{B})$, the leaves

corresponding to the contracted conjuncts are all labeled u . Since $u \cdot T = T(\mathcal{B})$, by the induction hypothesis, the corresponding leaves in T are labeled (ϵ, ϵ) . Therefore, these may be collapsed resulting in a configuration T' such that $u \cdot T' = T(\mathcal{A})$.

- Assume that \mathcal{A} is derived from \mathcal{B} by application of the rule $A \rightarrow (\alpha_1 \& \cdots \& \alpha_k)$. Since the derivation is a strong leftmost derivation, A is the leftmost variable in a simple conjunct α having the shortest Σ -prefix, which, by Lemma 28, is u . Therefore, the simple conjunct α' in $\mathcal{S}(\mathcal{B})$ corresponding to α is of the form $uA\beta$. By the induction hypothesis, the leaf v in T corresponding to α' is labeled $(\epsilon, A\beta)$, because after prepending u the label of v is $uA\beta$.

By the definition of A_G , $\alpha_1 \wedge \cdots \wedge \alpha_k \in \delta(\epsilon, A)$. Let T' be the configuration resulting in applying this transitions to v in T . Then, $T'(v) = \beta$ and v has k child nodes v_1, \dots, v_k in T' , each labeled (ϵ, α_i) for $i = 1, \dots, k$. Since the leaves v_i in $T(\mathcal{A})$ are labeled $u\alpha_i$, $u \cdot T' = T(\mathcal{A})$.

So, in both cases, after applying appropriate transitions we have $u \cdot T' = T(\mathcal{A})$. If $u = \mathbf{M}(\mathcal{A})$, then we are done. If, however, $\mathbf{M}(\mathcal{A}) = uy$ for some $y \in \Sigma^+$,¹⁸ then all leaves v in T' and $T(\mathcal{A})$ are labeled $(\epsilon, y\alpha_v)$ and $uy\alpha_v$, respectively, for some $\alpha_v \in \Gamma^*$. Therefore, by appending y to u we obtain a computation of A_G on the prefix u of uy ($= \mathbf{M}(\mathcal{A})$) ending in a configuration where all leaves v are labeled $(y, y\alpha_v)$. By repeatedly applying transitions of the form $\delta(\sigma, \sigma) = \{\epsilon\}$ to all leaves, all symbols of y can be emptied. That is, after reading the whole uy , A_G is in the configuration T'' , where all leaves v are labeled (ϵ, α_v) , implying $uy \cdot T'' = T(\mathcal{A})$. ■

At last we have arrived at the proof of the “only if” part of Theorem 13.

Let $w \in L(A_G)$. That is, there is a computation of A_G on w ending in the accepting configuration (ϵ, ϵ) . By Proposition 26, there exists a leftmost formula \mathcal{A} matching (ϵ, ϵ) such that $S \Rightarrow_L^* \mathcal{A}$. Therefore, $T(\mathcal{A})$ consists of a single node labeled w . The only leftmost formula having this tree representation is w itself. That is, $S \Rightarrow^* w$, implying $L(A_G) \subseteq L(G)$.

Conversely, let $w \in L(G)$. Then, by Lemma 30, $S \Rightarrow_{SL}^* w$. Since w consists of terminals only, it is in simplified form, $\mathbf{M}(w) = w$, and its tree representation has a single node labeled w . Therefore, by Proposition 31,

¹⁸Since $\mathcal{B} \Rightarrow_{SL} \mathcal{A}$, $\mathbf{M}(\mathcal{B})$ is a prefix of $\mathbf{M}(\mathcal{A})$.

there exists a computation of A_G on w ending in the accepting configuration (ϵ, ϵ) . That is, $w \in L(A_G)$, implying $L(G) \subseteq L(A_G)$.

Consequently, $L(A_G) = L(G)$ and the proof of the “only if” part of the theorem is complete.

From the “if” part of Theorem 13 and the proof of its “only if” part we obtain the following immediate corollary.

Corollary 32 *For every SAPDA there is a single-state SAPDA that accepts the same language.*

In other words, allowing SAPDA to have any finite number of states does not make the computation model stronger than that of one-state SAPDA.

5 Concluding Remarks

We have introduced a synchronized model of Alternating Pushdown Automata, SAPDA, which is equivalent to the CG model. We have also shown that, as in the classical case, the single-state and multi-state variants of the model are equivalent. Through the equivalence with CG, certain properties of SAPDA can be deduced, such as accepted languages, closure properties, etc. As the exact class of languages generated by CGs is not yet known, the exact class of languages accepted by SAPDA is not known either. Perhaps the formalization as an automaton will help find methods to prove that languages are *not* accepted by the model, thus answering some open questions.

References

- [1] A. K. Chandra, D. C. Kozen, L. J. Stockmeyer, Alternation, Journal of the ACM 28 (1) (1981) 114–133.
- [2] K. Culik II, J. Gruska, A. Salomaa, Systolic Trellis automata, I and II, International Journal of Computer Mathematics 15 and 16 (1 and 3–4) (1984) 195–212 and 3–22.
- [3] J. E. Hopcroft, J. D. Ullman, Introduction to Automata Theory, Languages and Computation, Addison-Wesley, 1979.

- [4] O. Kupferman, M. Y. Vardi, Weak alternating automata are not that weak, *ACM Transaction on Computational Logic* 2 (3) (2001) 408–429.
- [5] R. E. Ladner, R. J. Lipton, L. J. Stockmeyer, Alternating pushdown and stack automata, *SIAM Journal on Computing* 13 (1) (1984) 135–155.
- [6] A. Okhotin, Conjunctive grammars, *Journal of Automata, Languages and Combinatorics* 6 (4) (2001) 519–535.
- [7] A. Okhotin, A recognition and parsing algorithm for arbitrary conjunctive grammars, *Theoretical Computer Science* 302 (2003) 81–124.
- [8] A. Okhotin, Efficient automaton-based recognition for linear conjunctive languages, *International Journal of Foundations of Computer Science* 14 (6) (2003) 1103–1116.
- [9] A. Okhotin, On the equivalence of linear conjunctive grammars and trellis automata, *RAIRO Theoretical Informatics and Applications* 38 (1) (2004) 69–88.