Linear Conjunctive Grammars and One-turn Synchronized Alternating Pushdown Automata

Tamar Aizikowitz and Michael Kaminski

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

Abstract. In this paper we introduce a sub-family of synchronized alternating pushdown automata, *one-turn Synchronized Alternating Pushdown Automata*, which accept the same class of languages as those generated by Linear Conjunctive Grammars. This equivalence is analogues to the classical equivalence between one-turn PDA and Linear Grammars, thus strengthening the claim of Synchronized Alternating Pushdown Automata as a natural counterpart for Conjunctive Grammars.

1 Introduction

Context-free languages lay at the very foundations of Computer Science, proving to be one of the most appealing language classes for practical applications. One the one hand, they are quite expressive, covering such syntactic constructs as necessary, e.g., for mathematical expressions. On the other hand, they are polynomially parsable, making them practical for real world applications. However, research in certain fields has raised a need computational models which extend context-free models, without loosing their computational efficiency.

Conjunctive Grammars (CG) are an example of such a model. Introduced by Okhotin in [10], CG are a generalization of context-free grammars which allow explicit intersection operations 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 [10, 11], making the model practical from a computational standpoint, and therefore, of interest for applications in various fields such as, e.g., programming languages.

Alternating automata models were first introduced by Chandra, Kozen and Stockmeyer in [3]. Alternating Pushdown Automata (APDA) were further explored in [9], and shown to accept exactly the exponential time languages. As such, they are too strong a model for Conjunctive Grammars. Synchronized Alternating Pushdown Automata (SAPDA), introduced in [1], are a weakened ver-

sion of Alternating Pushdown Automata, which accept conjunctions of context-free languages. In [1], SAPDA were proven to be equivalent to CG[1].

In [10], Okhotin defined a sub-family of Conjunctive Grammars called *Linear Conjunctive Grammars* (LCG), analogously to the definition of Linear Grammars as a sub-family of Context-free Grammars. LCG are an interesting sub-family of CG as they have especially efficient parsing algorithms, see [12], making them particularly appealing from a computational standpoint. Also, many of the interesting languages derived by Conjunctive Grammars, can in fact be derived by Linear Conjunctive Grammars. In [13], Okhotin proved that LCG are equivalent to a type of *Trellis Automata*.

It is a well-known result, due to Ginsburg and Spanier [5], that Linear Grammars are equivalent to one-turn PDA. One-turn PDA are a sub-family of pushdown automata, where in each computation the stack height switches only once from non-decreasing to non-increasing. That is, once a transition replaces the top symbol of the stack with ϵ, all subsequent transitions may write at most one character.

In this paper we introduce a sub-family of SAPDA, *one-turn Synchronized Alternating Pushdown Automata*, and prove that they are equivalent to Linear Conjunctive Grammars. The equivalence is analogues to the classical equivalence between one-turn PDA and Linear Grammars. This result greatly strengthens the claim of SAPDA as a natural automaton counterpart for Conjunctive Grammars.

The paper is organized as follows. In Section 2 we recall the definitions of Conjunctive Grammars, Linear Conjunctive Grammars, and SAPDA. In Section 3 we introduce one-turn SAPDA as a sub-family of general SAPDA. Section 4 details our main result, namely the equivalence of the LCG and one-turn SAPDA models. Section 5 discusses the relationship between LCG and Mildly Context Sensitive Languages, and Section 6 is a short conclusion of our work.

2 Preliminaries

Following, we recall the definitions of Conjunctive Grammars, Linear Conjunctive Grammars, and Synchronized Alternating Pushdown Automata, as appeared in [1].

2.1 Conjunctive Grammars

The following definitions are taken from [10].

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.*

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

– *P is a finite set of rules of the form $A \to (\alpha_1 \,\&\cdots\, \&\, \alpha_k)$ such that $A \in V$ and $\alpha_i \in (V \cup \Sigma)^*$, $i = 1, \ldots, k$. If $k = 1$ then we write $A \to \alpha$.*

Definition 2. Conjunctive Formulas *over $V \cup \Sigma \cup \{(,), \&\}$ 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 $\mathcal{AB}$ is a conjunctive formula.*
– *If $\mathcal{A}_1, \ldots, \mathcal{A}_k$ 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, $i = 1, \ldots, k$, 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 $A \to (\alpha_1 \,\&\, \cdots \,\&\, \alpha_k) \in P$, and
(2) $s_1(w\& \,\cdots\, \&w)s_2 \Rightarrow_G s_1 w s_2$ for all $w \in \Sigma^*$,

where $s_1, s_2 \in (V \cup \Sigma \cup \{(,), \&\})^*$. As usual, $\Rightarrow_G^*$ is the reflexive and transitive closure of $\Rightarrow_G$,[4] *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.*

Example 1. ([10, Example 1]) The following conjunctive grammar generates the non-context-free language $\{a^n b^n c^n : n = 0, 1, \ldots\}$, 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:
$S \to (C \,\&\, A)$; $C \to Cc \mid D$; $A \to aA \mid E$; $D \to aDb \mid \epsilon$; $E \to bEc \mid \epsilon$

[2] Note that this definition is different from Okhotin's definition in [10].
[3] Note that, according to this definition, conjunctive formulas which are elements of $(V \cup \Sigma)^*$ are not simple conjuncts or enclosing formulas.
[4] In particular, 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 $\mathcal{A}_i$, $i = 1, \ldots, k$.

The intuition of the above derivation rules is as follows.

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

and

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

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

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

2.2 Linear Conjunctive Grammars

Okhotin defined in [10] a sub-family of conjunctive grammars called *Linear Conjunctive Grammars* (LCG) and proved in [13] that they are equivalent to *Trellis* Automata.[5] The definition of LCGs is analogues to the definition of Linear Grammars as a sub-family of Context-free Grammars.

Definition 5. *A conjunctive grammar $G = (V, \Sigma, P, S)$ is said to be* linear *if all rules in P are in one of the following forms.*

- $A \to (u_1 B_1 v_1 \;\&\; \cdots \;\&\; u_k B_k v_k)$; $u_i, v_i \in \Sigma^*$ *and* $A, B_i \in V$, *or*
- $A \to w$; $w \in \Sigma^*$, *and* $A \in V$.

Several interesting languages can be generated by LCGs. In particular, the grammar in Example 1 is linear. Following is a particularly interesting example, due to Okhotin, of a Linear CG which uses recursive conjunctions to derive a language which *cannot* be obtained by a finite conjunction of context-free languages.

Example 2. ([10, Example 2]) The following linear 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.

$$S \to (C \;\&\; D) \quad ; \quad C \to aCa \mid aCb \mid bCa \mid bCb \mid \$$$
$$D \to (aA \;\&\; aD) \mid (bB \;\&\; bD) \mid \$E \quad ; \quad A \to aAa \mid aAb \mid bAa \mid bAb \mid \$Ea$$
$$B \to aBa \mid aBb \mid bBa \mid bBb \mid \$Eb \quad ; \quad E \to aE \mid bE \mid \epsilon$$

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 $\{w\$uw | w, u \in \{a, b\}\}$. The grammar languages is the intersection of these two languages, i.e., the reduplication with a center marker language. For a more detailed description, see [10, Example 2].

[5] As Trellis Automata are not a part of this paper, we omit the definition, which can be found in [4] or [13].

2.3 Synchronized Alternating Pushdown Automata

Following, we recall the definition of *Synchronized Alternating Pushdown Automata* (SAPDA). Introduced in [1], SAPDA are a variation on standard PDA which add the power of conjunction. In the SAPDA 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. If all conjunctions are of one state only, the automaton is a standard PDA.[6]

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.[7] The branches process the input independently, however sibling branches must empty synchronously, after which the computation continues from the parent branch.

Definition 6. *A* Synchronized Alternating Pushdown Automaton *is a tuple* $A = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$*, where* δ *is a function that assigns to each element of* $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ *a finite subset of*

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

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*
- $\bot \in \Gamma$ *is the initial stack symbol,*

see, e.g., [6, 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 7. *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.*

[6] This type of formulation for alternating automata models is equivalent to the one presented in [3], and is standard in the field of Formal Verification, e.g., see [7].

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

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,[8] 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.[9] 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 8. *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 $(q_1, w, \alpha_1\alpha)$.*
  - *If $k > 1$, then T' is obtained from T by relabeling v with α, and adding to it k child nodes $v_1, \ldots, v_k$ such that $T'(v_i) = (q_i, w, \alpha_i)$, $i = 1, \ldots, k$.*

  In this case we say that the computation step is based on $(q_1, \alpha_1) \wedge \cdots \wedge (q_k, \alpha_k)$ applied to v.
- *There is a node v in T, $T(v) = \alpha$, that has k children $v_1, \ldots, v_k$, all of which are leaves labeled the same (p, w, ϵ), and T' is obtained from T by removing all leaf nodes v_i, $i = 1, \ldots, k$ 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 9. *Let A be an SAPDA and let $w \in \Sigma^*$.*

- *The* initial *configuration of A on w is the configuration $(q_0, w, \bot)$.*[10]
- *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 $T_0, \ldots, T_n$, where*
  - *T_0 is the initial configuration,*
  - *$T_{i-1} \vdash_A T_i$ for $i = 1, \ldots, n$, 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.*

[8] That is, the configuration tree consists of the root only, which is also the only leaf of the tree.

[9] Equivalently, all branches can take one step together. This formulation simplifies the configurations, as all branches read the input at the same pace. However, it is less correlated with the grammar model, making equivalence proofs more involved.

[10] That is, the initial configuration of A on w is a one node tree whose only node is labeled $(q_0, w, \bot)$.

The language $L(A)$ of A is the set of all $w \in \Sigma^*$ such that A has an accepting computation on w.[11]

Example 3. The SAPDA $A = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$ defined below accepts the non-context-free language

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

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

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 *abbcccaab*.
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 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 2. The below automaton accepts the language derived by the non-terminal D in the grammar.

[11] Alternatively, 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.

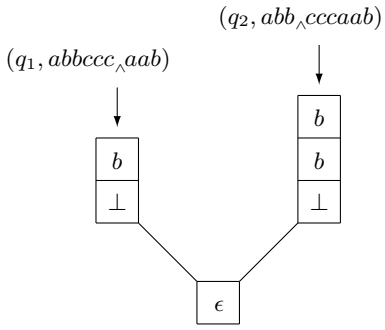

Fig. 1. Intermediate stage of a computation on *abbcccaab*.

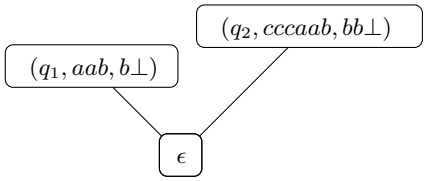

Fig. 2. The configuration corresponding to Figure 1.

Example 4. (cf. Example 2) The SAPDA $A = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$ defined below accepts the non-context-free language

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

- $Q = \{q_0, q_e\} \cup \{q^1_\sigma : \sigma \in \{a, b\}\} \cup \{q^2_\sigma : \sigma \in \{a, b\}\}$,
- $\Sigma = \{a, b, \$\}$,
- $\Gamma = \{\bot, \#\}$, and
- δ is defined as follows.
  1. $\delta(q_0, \sigma, \bot) = \{(q^1_\sigma, \bot) \wedge (q_0, \bot)\}$, $\sigma \in \{a, b\}$
  2. $\delta(q^1_\sigma, \tau, X) = \{(q^1_\sigma, \#X)\}$, $\sigma, \tau \in \{a, b\}$, $X \in \Gamma$,
  3. $\delta(q_0, \$, \bot) = \{(q_e, \epsilon)\}$,
  4. $\delta(q^1_\sigma, \$, X) = \{(q^2_\sigma, X)\}$, $\sigma \in \{a, b\}$, $X \in \Gamma$,
  5. $\delta(q^2_\sigma, \tau, X) = \{(q^2_\sigma, X)\}$, $\sigma, \tau \in \{a, b\} : \sigma \neq \tau$, $X \in \Gamma$,
  6. $\delta(q^2_\sigma, \sigma, X) = \{(q^2_\sigma, X), (q_e, X)\}$, $\sigma \in \{a, b\}$, $X \in \Gamma$,
  7. $\delta(q_e, \sigma, \#) = \{(q_e, \epsilon)\}$, $\sigma \in \{a, b\}$,
  8. $\delta(q_e, \epsilon, \bot) = \{(q_e, \epsilon)\}$

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^1_σ (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 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 is state q_b^2.

3 One-turn Synchronized Alternating Pushdown Automata

It is a well-known result, due to Ginsburg and Spanier [5], that linear grammars are equivalent to one-turn PDA. One-turn PDA are a sub-family of pushdown automata, where in each computation the stack height switches only once from non-decreasing to non-increasing. That is, once a transition replaces the top symbol of the stack with ϵ, all subsequent transitions may write at most one character. A similar notion of one-turn SAPDA can be defined, where each stack branch can make only one turn in the course of a computation.

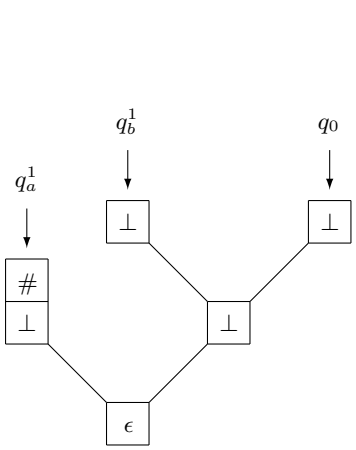

Fig. 3. Stack tree contents after all branches have read *ab*.

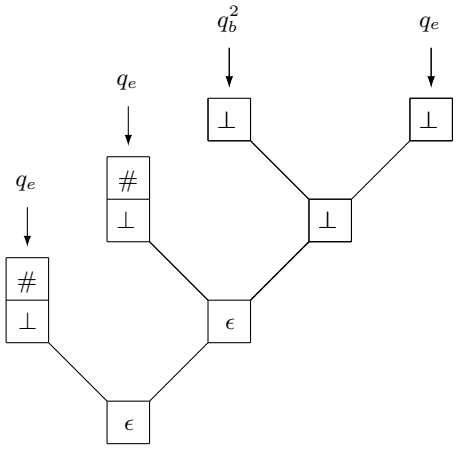

Fig. 4. Stack tree contents after all branches have read *abb*\$*ba*.

Definition 10. *Let A be an SAPDA and let T, T' be configurations of $\mathcal{A}$ such that $T \vdash T'$.*

- *The computation step (or transition) $T \vdash T'$ is called* increasing*, if it is based on $(q_1, \alpha_1) \wedge \cdots \wedge (q_k, \alpha_k)$ such that $k \geq 2$, or is based on (q, α) such that $|\alpha| \geq 2$.*
- *The computation step (or transition) $T \vdash T'$ is called* decreasing*, if it is based on the collapsing of sibling leaves, or is based on (q, ϵ).*
- *The computation step $T \vdash T'$ is called* non-changing*, if it is based on (q, X), $X \in \Gamma$.*

A computation step is non-decreasing (respectively, non-increasing), if it is non-changing or increasing (respectively, decreasing).

Intuitively, a *turn* is a change from non-decreasing computation steps to non-increasing ones. We would like to capture the number of turns each stack branch makes through the course of a computation. Stack branches are determined by nodes in the configuration tree. We first define the segment of the computation in which a specific node appears, and then we can consider the number of turns for that node.

Definition 11. *Let A be an SAPDA , let $T_0 \vdash \cdots \vdash T_n$ be a computation of A, and let v be a node that appears in a configuration of this computation. Let T_i be the first configuration containing v and let T_j be the last configuration containing v, $0 \leq i \leq j \leq n$. The* span *of node v is the sub-computation beginning with configuration T_i and ending with configuration T_j.*

For example, if v is the root node then the span of v is the entire computation $T_0 \vdash \cdots \vdash T_n$.

Within the span of a node v, not all transitions are relevant to v. For example, transitions may be applied to a node v' which is in a different sub-tree of the configuration. The relevant transitions are, first and foremost, those that are applied to v. However, at some point, v may have child nodes. In this case, transitions applied to v's children are not relevant, but the collapsing of these nodes is. This is because the collapsing of child nodes is similar to popping a symbol from the stack (when one considers the combined stack heights of the parent and the children nodes), and can therefore be viewed as a decreasing transition on v.

Definition 12. *Let A be an SAPDA and let $T_0 \vdash \cdots \vdash T_n$ be a computation of A. Let v be a node that appears in (a configuration of) this computation, and let $T_i \vdash \cdots \vdash T_j$ be its span. Let $t_i, \ldots, t_{j-1}$ be the sequence of transitions applied in the computation $T_i \vdash \cdots \vdash T_j$, i.e., $T_k \vdash T_{k+1}$ by transition t_k, $k = i, \ldots, j-1$. The* relevant transitions *on v is the (order-maintaining) subsequence $t_{\ell_1}, \ldots, t_{\ell_m}$ of $t_i, \ldots, t_{j-1}$ such that for each $k = 1, \ldots, m$, one of the following holds.*

- *The computation step $T_{\ell_k} \vdash T_{\ell_k+1}$ is based on some $(q_1, \alpha_1) \wedge \cdots \wedge (q_k, \alpha_k)$ applied to v, or*
- *the computation step $T_{\ell_k} \vdash T_{\ell_k+1}$ is based on a collapsing of child nodes of v.*

Definition 13. *An SAPDA A is* one-turn*, if for each accepting computation $T_0 \vdash_A \cdots \vdash_A T_n$ the following holds. Let v be a node that appears in (a configuration of) this computation and let $t_{\ell_1}, \ldots, t_{\ell_m}$ be the relevant transitions on v. Then there exists $1 \leq k \leq m$ such that*

- *the computation step $T_{\ell_k} \vdash T_{\ell_k+1}$ is decreasing,*
- *for all $1 \leq k' < k$, the computation steps $T_{\ell_{k'}} \vdash T_{\ell_{k'}+1}$ are non-decreasing, and*
- *for all $k \leq k' < m$, the computation steps $T_{\ell_{k'}} \vdash T_{\ell_{k'}+1}$ are non-increasing.*

Informally, the definition states that in every accepting computation, the hight of each stack branch turns exactly once. Note that the requirement of a decreasing step in the computation is not limiting as we are considering acceptance by empty stack. As such, every accepting computation must have at least one decreasing computation step.

Remark 1. Reordering transitions as necessary, we can assume that all transitions on a node v and its descendants, are applied sequentially.[12] By this assumption, all transitions in the span of a node v are applied either to v or to one of its descendants.

If the automaton is one-turn, these transitions can be partitioned into three parts, the first being non-decreasing transitions applied to v, the second being transitions applied to descendants of v, and the third being non-increasing transitions applied to v. The relevant transitions on v, in this case, are the first and

[12] The order of transitions applied to different sub-trees can be changed, without affecting the final configuration reached.

third parts. Note that if the middle section is non-empty then the last transition in the first part must be a conjunctive one which creates child nodes for v (an increasing transition), and the first transition in the final part must be the collapsing of these child nodes (a decreasing transition).

When viewing a classical one-turn PDA as a one-turn SAPDA, there is only one "node" in all configurations, because there are no conjunctive transitions. Therefore, the span of the node is the entire computation, and it is comprised only of the first and third parts, i.e., non-decreasing transitions followed by non-increasing ones, coinciding with the classical notion of a one-turn PDA.

Note that the automaton described in Example 4 is in fact a one-turn SAPDA, while the automaton from Example 3 is not.

4 Linear CG and One-Turn SAPDA

Similarly to the context-free case, one-turn SAPDA and LCG are equivalent.

Theorem 1. *A language is generated by an LCG if and only if it is accepted by a one-turn SAPDA.*

The proof of the "only if" part of the theorem is presented in Section 4.1 and the proof of its "if" part is presented in Section 4.2. For full proofs of the theorems and additional details, see [2].

We precede the proof of Theorem 1 with the following immediate corollary.

Corollary 1. *One-turn SAPDA are equivalent to Trellis Automata.*

While both computational models employ a form of parallel processing, their behavior is quite different. To better understand the relationship between the models, see the proof of equivalence between LCG and Trellis Automata in [13]. As LCG and one-turn SAPDA are closely related, the equivalence proof provides intuition on the relation with one-turn SAPDA as well.

4.1 Proof of the "only if" part of Theorem 1

For the purposes of our proof we assume that grammars do not contain ϵ-rules. Okhotin proved in [10] 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 linear conjunctive grammar. Consider the SAPDA $A_G = (Q, \Sigma, \Gamma, q_0, \bot, \delta)$, where

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

– $Q = \{q_{u'} : \text{for some } A \to (\cdots \ \& \ uBy \ \& \ \cdots) \in P \text{ and some } z \in \Sigma^*,\ u = zu'\}$,[13]
– $q_0 = q_\epsilon$, and
– δ is defined as follows.
  1. $\delta(q_\epsilon, \epsilon, A)$ is the union of
     (a) $\{(q_\epsilon, w) : A \to w \in P\}$ and
     (b) $\{(q_{u_1}, B_1 y_1) \wedge \cdots \wedge (q_{u_k}, B_k y_k) : A \to (u_1 B_1 y_1 \ \& \ \cdots \ \& \ u_k B_k y_k) \in P\}$,
  2. $\delta(q_{\sigma u}, \sigma, A) = \{(q_u, A)\}$,
  3. $\delta(q_\epsilon, \sigma, \sigma) = \{(q_\epsilon, \epsilon)\}$

The proof is an extension of the classical one, see, e.g., [6, Theorem 5.3, pp. 115–116]. The construction is modified so as to result in a one-turn automaton. For examples, transitions such as transition 2 are used to avoid removing symbols from the stack in the increasing phase of the computation.

Following, we prove that A_G is in fact a one-turn SAPDA. Let $T_0 \vdash \cdots \vdash T_n$ be an accepting computation of A_G, let v be a node appearing in (a configuration of) the computation, and let $T_i \vdash \cdots \vdash T_j$, $0 \leq i \leq j \leq n$, be the span of v.

Proposition 1. *In T_i, the contents of the stack of v is of the form Ay, where $A \in \Gamma$ and $y \in \Sigma^*$.*

Proof. If v is the root, then $T_i = T_0$ and v has S in the stack. If v is not the root, then, since T_i is the first configuration containing v, $T_{i-1} \vdash T_i$ was by a transition of type 1(b). Therefore, v contains Ay in its stack. □

Proposition 2. *In T_j, the contents of the stack of v is empty.*

Proof. If v is the root, then $T_j = T_n$ and the proposition follows from the fact that the computation is accepting. If v is not the root, then, since T_i is the last configuration containing v, $T_j \vdash T_{j+1}$ is a collapsing of nodes, one of which is v. Since nodes can only be collapsed with empty stacks, the proposition follows in this case as well. □

Let $t_{\ell_1}, \ldots, t_{\ell_m}$ be the relevant transitions on v. Corollary 2 below immediately follows from Propositions 1 and 2.

Corollary 2. *There exists $k \in \{1, \ldots, m\}$ such that t_{ℓ_k} is decreasing.*

Now, let k be the smallest index such that t_{ℓ_k} is decreasing (i.e., t_{ℓ_k} is first decreasing transition applied to v). There are three possible cases: t_{ℓ_k} is of type 1(a), where $w = \epsilon$; or it is of type 3; or it is a collapsing of child nodes. In all cases, if the stack is not empty after applying the transition, it contains only terminal symbols. This is because type 1(a) transitions remove the non-terminal symbol, and type 3 and collapsing transitions assume the top symbol is in Σ. By the construction of the automaton, if there is a non-terminal symbol in a stack branch, then it is the top symbol. Therefore, if the top symbol is terminal, so are all the rest. It follows that all subsequent relevant transitions can only be of type 3, meaning they are all decreasing, and A_G is, in fact, one-turn.

[13] That is, u' is a suffix of u.

4.2 Proof of the "if" part of Theorem 1

The proof is a variation on a similar proof for the classical case presented in [5]. For the purposes of simplification, we make several assumptions regarding the structure of one-turn SAPDA, namely that they are single-state and that their transitions write at most two symbols at a time. We state that these assumptions are not limiting in the following two lemmas.

Lemma 1. *For each one-turn SAPDA there is an equivalent single-state one-turn SAPDA.*

Definition 14. *Let $A = (\Sigma, \Gamma, \bot, \delta)$ be a single-state SAPDA. We say that A is* bounded *if for all $\sigma \in \Sigma \cup \{\epsilon\}$ and all $X \in \Gamma$ the following holds.*

- *For every $\alpha \in \delta(\sigma, X)$, $|\alpha| \leq 2$.*
- *For every $\alpha_1 \wedge \cdots \wedge \alpha_k \in \delta(\sigma, X)$, $k \geq 2$, $|\alpha_i| = 1$, $i = 1, \ldots, k$.*

Lemma 2. *Every single-state one-turn SAPDA is equivalent to a bounded single-state one-turn SAPDA.*

Let $A = (\Sigma, \Gamma, \bot, \delta)$ be a one-turn SAPDA. By Lemmas 1 and 2, we may assume that A is single-state and bounded. Consider the *linear* conjunctive grammar $G_A = (V, \Sigma, P, S)$, where

- $V = \Gamma \times (\Gamma \cup \{\epsilon\})$,
- $S = [\bot, \epsilon]$, and
- P is the union of the following sets of rules, for all $\sigma \in \Sigma \cup \{\epsilon\}$ and all $X, Y, Z \in \Gamma$.
  1. $\{[X, Y] \to \sigma[Z, \epsilon] : ZY \in \delta(\sigma, X)\}$,
  2. $\{[X, Y] \to \sigma[Z, Y] : Z \in \delta(\sigma, X)\}$,
  3. $\{[X, \epsilon] \to \sigma[Z, \epsilon] : Z \in \delta(\sigma, X)\}$,
  4. $\{[X, \epsilon] \to (\sigma[X_1, \epsilon]\ \&\ \cdots\ \&\ \sigma[X_k, \epsilon]):$ $k \geq 2$ and $X_1 \wedge \cdots \wedge X_k \in \delta(\sigma, X)\}$,[14]
  5. $\{[X, \epsilon] \to \sigma : \epsilon \in \delta(\sigma, X)\}$,
  6. $\{[X, Y] \to [X, Z]\sigma : Y \in \delta(\sigma, Z)\}$, and
  7. $\{[X, \epsilon] \to [X, Z]\sigma : \epsilon \in \delta(\sigma, Z)\}$.

The grammar variables $[X, Y]$ correspond to zero- and one-turn computations starting with X in the stack and ending with Y in the stack. In particular, any word derived from $[\bot, \epsilon]$ is a word with a one-turn emptying computation of A.

The various types of production rules defined, correspond to the different types of transitions of the automaton.

- Rules of type 1 correspond to increasing computation steps,
- rules of type 2, 3 and 6 correspond to non-changing computation steps,
- rules of type 4 correspond to conjunctive transitions,
- rules of type 5 correspond to the turn step, and

[14] Since A is bounded, $X_i \neq \epsilon$, $i = 1, \ldots, k$.

– rules of type 7 correspond to decreasing computation steps.

The correctness of the construction follows form Proposition 3 below, which completes our proof of the "if" part of Theorem 1.

Proposition 3. *For every $X \in \Gamma$, $Y \in \Gamma \cup \{\epsilon\}$, and $w \in \Sigma^*$, $[X, Y] \Rightarrow^* w$ if and only if $(w, X) \vdash^* (\epsilon, Y)$ with exactly one turn.*

5 Mildly Context-Sensitive Languages

Computational linguistics focuses on defining a computational model for natural languages. Originally, context-free languages were considered, and many natural language models are in fact models for context-free languages. However, certain natural language structures that cannot be expressed in context free languages, led to an interest in a slightly wider class of languages which came to be known as *mildly context-sensitive languages* (MCSL). Several formalisms for grammar specification are known to converge to this class [15].

Mildly context sensitive languages are loosely categorized as having the following properties: (1) They contain the context-free languages; (2) They contain such languages as multiple-agreement, cross-agreement and reduplication; (3) They are polynomially parsable; (4) They are semi-linear[15].

There is a strong relation between the class of languages derived by linear conjunctive grammars (and accepted by one-turn SAPDA) and the class of mildly context sensitive languages. The third criterion of MCSL is met by Okhotin's proof that CG membership is polynomial. Multiple-agreement and reduplication with a center marker[16] are shown in Examples 1 and 2 respectively, and an LCG for cross-agreement can be easily constructed. While the first criterion is met for general conjunctive languages, it is not met for the linear sub-family, as there exist context-free languages which are not linear conjunctive [14]. However, every linear context-free language, is of course also a linear conjunctive language.

The fourth criterion of semi-linearity is not met for linear conjunctive languages, and subsequently not for general conjunctive languages as well. In [14], Okhotin presents an LCG for the language $\{ba^2ba^4 \cdots ba^{2^n}b | n \in \mathbb{N}\}$, which has super-linear growth. In this respect, LCG and one-turn SAPDA accept some languages not characterized as mildly context-sensitive.

6 Concluding Remarks

We have introduced one-turn SAPDA as a sub-family of SAPDA, and proven that they are equivalent to Linear Conjunctive Grammars. This supports the claim from [1] that SAPDA are a natural counterpart for CG. The formulation

[15] A language L is *semi-linear* if $\{|w| \mid w \in L\}$ is a finite union of sets of integers of the form $\{l + im \mid i = 0, 1, \dots\}$, $l, m \geq 0$.

[16] Okhotin has conjectured that reduplication without a center marker cannot be generated by any CG. However, this is still an open problem.

as an automaton provides additional insight into Linear Conjunctive Grammars, and may help solve some of the open questions regarding them.

In [8], Kutrib and Malcher explore a wide range of finite-turn automata with and without turn conditions, and their relationships with closures of linear context-free languages under regular operations. It would prove interesting to explore the general case of finite-turn SAPDA, perhaps finding models for closures of linear conjunctive languages under regular operations.

Acknowledgments

This work was supported by the Winnipeg Research Fund.

References

1. Aizikowitz, T., Kaminski, M.: Conjunctive grammars and alternating pushdown automata. In Hodges, W., de Queiroz, R., eds.: The 15th Workshop on Logic, Language, Information and Computation – WoLLIC'2008. Volume 5110 of Lecture Notes in Artificial Intelligence., Heidelberg, Springer Verlag (2008) 30–41
2. Aizikowitz, T., Kaminski, M.: Linear conjunctive grammars and one-turn synchronized alternating pushdown automata. Technical Report CS-2009-15, Technion – Israel Institute of Technology (2009)
3. Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. Journal of the ACM **28**(1) (1981) 114–133
4. Culik II, K., Gruska, J., Salomaa, A.: Systolic Trellis automata, I and II. International Journal of Computer Mathematics **15 and 16**(1 and 3–4) (1984) 195–212 and 3–22
5. Ginsburg, S., Spanier, E.H.: Finite-turn pushdown automata. SIAM Journal on Control **4**(3) (1966) 429–453
6. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley (1979)
7. Kupferman, O., Vardi, M.Y.: Weak alternating automata are not that weak. ACM Transaction on Computational Logic **2**(3) (2001) 408–429
8. Kutrib, M., Malcher, A.: Finite turns and the regular closure of linear context-free languages. Discrete Applied Mathematics **155**(16) (2007) 2152–2164
9. Ladner, R.E., Lipton, R.J., Stockmeyer, L.J.: Alternating pushdown and stack automata. SIAM Journal on Computing **13**(1) (1984) 135–155
10. Okhotin, A.: Conjunctive grammars. Journal of Automata, Languages and Combinatorics **6**(4) (2001) 519–535
11. Okhotin, A.: A recognition and parsing algorithm for arbitrary conjunctive grammars. Theoretical Computer Science **302** (2003) 81–124
12. Okhotin, A.: Efficient automaton-based recognition for linear conjunctive languages. International Journal of Foundations of Computer Science **14**(6) (2003) 1103–1116
13. Okhotin, A.: On the equivalence of linear conjunctive grammars and trellis automata. RAIRO Theoretical Informatics and Applications **38**(1) (2004) 69–88
14. Okhotin, A.: On the closure properties of linear conjunctive languages. Theoretical Computer Science **299**(1-3) (2003) 663–685
15. Vijay-Shanker, K., Weir, D.J.: The equivalence of four extensions of context-free grammars. Mathematical Systems Theory **27**(6) (1994) 511–546