Conjunctive Grammars and Alternating Pushdown Automata
(Extended Abstract)

Tamar Aizikowitz and Michael Kaminski

Department of Computer Science
Technion – Israel Institute of Technology
{tamarai,kaminski}@cs.technion.ac.il

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.

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 looking at the dual form of these models (e.g., Co-NP), universal acceptance conditions lead to conjunction: all computations must accept. This type of acceptance is useful in such fields as concurrent programming where *all* processes must meet correctness demands. In this paper we explore several extensions of models for context-free languages which 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 [1], 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 [1,2], making the model practical from a computational standpoint, and therefore of interest for applications in various fields such as programming languages, etc. In this paper we introduce a new model of alternating automata, *Synchronized Alternating Pushdown Automata* (SAPDA), which is equivalent to the Conjunctive Grammar model.[1]

The concept of alternating automata models was first introduced by Chandra et.al. in [3]. In these models, computations alternate between existential and universal modes of acceptance, hence their name. This behavior is achieved

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

W. Hodges and R. de Queiroz (Eds.): WoLLIC 2008, LNAI 5110, pp. 44–55, 2008.
© Springer-Verlag Berlin Heidelberg 2008

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 [3].

Alternating Pushdown Automata (APDA) were also introduced in [3] and were further explored in [4]. 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 [3,4].

It is well known that Context-Free Grammars and Pushdown Automata are equivalent, e.g., see [5, pp. 115–119]. Yet, the APDA model is stronger than the CG model [1]. 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 [6,7] that Linear Conjunctive Grammars, a subfamily of the Conjunctive Grammars, are equivalent to Trellis Automata [8], 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 define the Conjunctive Grammar model. In Section 3 we introduce our SAPDA model. Section 4 details our main results, namely the equivalence of the CG and SAPDA models. Sections 5 and 6 contain discussions of mildly context-sensitive languages and related work respectively, and Section 7 is a short conclusion of our work.

2 Conjunctive Grammars

The following definitions are taken from [1].

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

1. V, Σ *are disjoint finite sets of non-terminal and terminal symbols respectively.*
2. $S \in V$ *is the designated start symbol.*
3. P *is a finite set of rules of the form* $A \to (\alpha_1 \& \ldots \& \alpha_n)$ *s.t.* $A \in V$ *and* $\alpha_i \in (V \cup \Sigma)^*$. *If* $n = 1$ *then we write* $A \to \alpha$.

Definition 2. Conjunctive Formulas *are defined over the alphabet* $V \cup \Sigma \cup \{(,), \&\}$. *The set of conjunctive formulas corresponding to a grammar* G *is defined as follows:*

1. ϵ *is a formula.*
2. *Every symbol in* $V \cup \Sigma$ *is a formula.*
3. *If* $\mathcal{A}$ *and* $\mathcal{B}$ *are formulas then* $\mathcal{AB}$ *is a formula.*
4. *If* $\mathcal{A}_1, \ldots, \mathcal{A}_n$ *are formulas then* $(\mathcal{A}_1 \& \cdots \& \mathcal{A}_n)$ *is a formula.*

Notation. Below we use the following notation: σ, τ denote terminal symbols, u, w, y denote terminal words, X, Y denote non-terminal symbols, α, β denote non-terminal words, and $\mathcal{A}, \mathcal{B}$ denote conjunctive formulas.

Definition 3. *For every conjunctive formula* $\mathcal{A} = (\mathcal{B}_1 \& \ldots \& \mathcal{B}_n)$, $\mathcal{B}_i s$, $i = 1, \ldots, n$, *are called are called* conjuncts *of* $\mathcal{A}$,[2] *and* $\mathcal{A}$ *is called the* enclosing formula. *If* $\mathcal{B}_i$ *contains no* &*-s, then* $\mathcal{B}_i$ *is a* simple conjunct.

Definition 4. *Given a grammar* G, *the relation of* immediate derivability *on the set of conjunctive formulas,* $\Rightarrow_G$, *is defined as follows:*

1. $s_1 X s_2 \Rightarrow_G s_1(\alpha_1 \& \cdots \& \alpha_n) s_2$ *for all* $X \to (\alpha_1 \& \cdots \& \alpha_n) \in P$
2. $s_1(w \& \cdots \& w) s_2 \Rightarrow_G s_1 w s_2$ *for all* $w \in T^*$

where $s_i \in (V \cup T \cup \{(,), \&\})^*$. *As usual,* $\Rightarrow_G^*$ *is the reflexive transitive closure of* $\Rightarrow_G$, *and the language of a grammar* G *is defined as* $L(G) = \{w \in T^* | S \Rightarrow_G^* w\}$. *We refer to (1) as* production *rules and to (2) as* contraction *rules.*

Informally, a terminal word w is derived from a formula $(\mathcal{A}_1 \& \cdots \& \mathcal{A}_n)$ if and only if it is derived from each $\mathcal{A}_i$.

Example 1. The following conjunctive grammar derives the non-context-free *multiple-agreement* language $\{a^n b^n c^n | n \in \mathbb{N}\}$. $G = (V, T, P, S)$ where:

- $V = \{S, A, B, C, X, Y\}$, $T = \{a, b, c\}$,
- P contains the following derivation rules:

$$\begin{aligned} S &\to (C \ \& \ A) \\ C &\to Cc \mid X \ ; \ A \to aA \mid Y \\ X &\to aXb \mid \epsilon \ ; \ Y \to bYc \mid \epsilon \end{aligned}$$

The derivation of the word *aabbcc* is as follows:

$$\begin{aligned} S &\Rightarrow (C \ \& \ A) \Rightarrow (Cc \ \& \ A) \Rightarrow (Ccc \ \& \ A) \Rightarrow (Xcc \ \& \ A) \\ &\Rightarrow (aXbcc \ \& \ A) \Rightarrow (aaXbbcc \ \& \ A) \Rightarrow (aabbcc \ \& \ A) \\ &\Rightarrow (aabbcc \ \& \ aA) \Rightarrow (aabbcc \ \& \ aaA) \Rightarrow (aabbcc \ \& \ aaY) \\ &\Rightarrow (aabbcc \ \& \ aabYc) \Rightarrow (aabbcc \ \& \ aabbYcc) \Rightarrow (aabbcc \ \& \ aabbcc) \Rightarrow aabbcc \end{aligned}$$

Example 2. The following linear conjunctive grammar derives the non-context-free *cross-agreement* language $\{a^n b^m c^n d^m | n, m \in \mathbb{N}\}$. $G = (V, T, P, S)$ where:

- $V = \{S, A, B, C, D, X, Y\}$, $T = \{a, b, c, d\}$,
- P contains the following derivation rules:

$$\begin{aligned} S &\to (A \ \& \ D) \\ A &\to aA \mid X \ ; \ D \to Dd \mid Y \\ X &\to bXd \mid C \ ; \ Y \to aYc \mid B \\ C &\to cC \mid \epsilon \ ; \ B \to bB \mid \epsilon \end{aligned}$$

[2] Note that this definition is different from Okhotin's definition in [1].

The non-terminal A derives words of the form $a^i b^m c^j d^m$ for $m, i, j \in \mathbb{N}$, while D derives words of the form $a^n b^i c^n d^j$ for $n, i, j \in \mathbb{N}$. Therefore, the conjunction of the two generates the cross-agreement language.

Example 3. The following linear conjunctive grammar, taken from [1] derives the non-context-free *reduplication* language with a center marker $\{wcw \mid w \in \{a,b\}^*\}$. $G = (V, T, P, S)$ where:

- $V = \{S, A, B, C, D, E\}$, $T = \{a, b, c\}$,
- P contains the following derivation rules:

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

The non-terminal C verifies that the lengths of the words before and after the c marker are equal, while D validates that the letters are the same. For a more detailed description see [1].

2.1 Linear Conjunctive Grammars

Okhotin defines in [1] a sub-family of conjunctive grammars called *Linear Conjunctive Grammars* (LCG). The definition is analogues 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 particularly efficient parsing algorithms [6], making them practical from a computational standpoint. Okhotin proved in [7] that LCGs are equivalent to Trellis Automata.

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

- $X \rightarrow (u_1 Y_1 v_1 \ \& \cdots \& \ u_n Y_n v_n)$; $u_i, v_i \in T^*$, $X, Y_i \in V$
- $X \rightarrow w$; $w \in T^*$, $X \in V$

The grammars presented in Examples 1, 2, 3 are all linear.

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 [3,4], SAPDA have both the power of existential and universal choice.

We use a different (equivalent) definition from the existential and universal state-sets one presented in [3]. Instead, transitions are made to a conjunction of states. The model is non-deterministic, therefore several different conjunctions may be possible from a given configuration. 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.[3] 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 the domain of* δ *is* $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$*. For every such* (q, σ, X)*,* δ *is a finite subset of*

$$\{(q_1, \alpha_1) \wedge \cdots \wedge (q_n, \alpha_n) \mid q_i \in Q,\ \alpha_i \in \Gamma^*,\ n \in \mathbb{N}\}\ .$$

Everything else is defined as in the standard PDA model; Q *is a finite set of states,* Γ, Σ *are the stack and input alphabets respectively,* $q_0 \in Q$ *is the initial state and* $\bot \in \Gamma$ *is the initial stack symbol, see, e.g., [5, pp. 107–112].*

We describe the current state of the automaton as a labelled 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 labelled tree. Each internal node is labelled* $\alpha \in \Gamma^*$ *denoting the stack-branch contents, and each leaf node is labelled* (q, w, α) *where* $q \in Q$ *denotes the current state,* $w \in \Sigma^*$ *denotes the remaining input to be read and* $\alpha \in \Gamma^*$ *denotes 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, it is denoted by the label of that node. For example, if a configuration T has a single node labelled (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 8. *Let* A *be an SAPDA and let* T*,* T' *be two configurations of* A*. We write* $T \vdash_A T'$ *(*A *is omitted if understood from the context), if:*

- *There exists a leaf node* v *in* T *s.t.* $T(v) = (q, \sigma w, X\alpha)$ *and a transition* $(q_1, \alpha_1) \wedge \cdots \wedge (q_k, \alpha_k) \in \delta(q, \sigma, X)$ *s.t.:*
  - *If* $k = 1$ *then* T' *can be obtained from* T *by relabelling* v *s.t.* $T'(v) = (q_1, w, \alpha_1\alpha)$*.*

[3] 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.

  - *If $k > 1$ then T' can be obtained from T by relabelling v s.t. $T'(v) = \alpha$, and adding k child nodes to v, $v_1, \ldots, v_k$ s.t. $T'(v_j) = (q_j, w, \alpha_j)$ for $j = 1, \ldots, k$.*
- *There is a node v in T s.t. $T(v) = \alpha$, v has k children $v_1, \ldots, v_k$ s.t. all v_js, $j = 1, \ldots, k$, are leaves labelled with the same (p, w, ϵ), and T' can be obtained from T by removing nodes v_j and relabelling v s.t. $T'(v) = (p, w, \alpha)$.*

We denote by $T \vdash_A^* T'$ the reflexive transitive closure of $\vdash_A$.

Definition 9. *Let A be an SAPDA and let $w \in \Sigma^*$.*

- *An* initial configuration *of A on w is the configuration $(q_0, w, \bot)$.*
- *An* accepting configuration *of A is a configuration of the form (q, ϵ, ϵ).*
- *A* computation *of A on w is a series 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 labelled (q, ϵ, α), i.e., the entire input string has been read.*
- *An* accepting computation *of A on w is a computation where the final configuration T_n is accepting.*[4]

The language of A, denoted $L(A)$, is the set of all $w \in \Sigma^*$ s.t. A has an accepting computation on w.

Example 4. The SAPDA, $A = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$, accepts the non-context-free language $\{w \mid \#_a(w) = \#_b(w) = \#_c(w)\}$ over $\Sigma = \{a, b, c\}$, where $Q = \{q_0, q_1, q_2\}$, $\Gamma = \{\bot, \bot_1, \bot_2, a, b, c\}$ and δ is defined as follows:

- $\delta(q_0, \epsilon, \bot) = \{(q_1, \bot_1) \wedge (q_2, \bot_2)\}$
- $\delta(q_i, \sigma, \bot_i) = \{(q_i, \sigma\bot_i)\}$, $(i, \sigma) \in \{1\} \times \{a, b\} \cup \{2\} \times \{b, c\}$
- $\delta(q_i, \sigma, \sigma) = \{(q_i, \sigma\sigma)\}$, $(i, \sigma) \in \{1\} \times \{a, b\} \cup \{2\} \times \{b, c\}$
- $\delta(q_1, \sigma_j, \sigma_k) = \{(q_1, \epsilon)\}$, $(\sigma_j, \sigma_k) \in \{(a, b), (b, a)\}$
- $\delta(q_2, \sigma_j, \sigma_k) = \{(q_2, \epsilon)\}$, $(\sigma_j, \sigma_k) \in \{(b, c), (c, b)\}$
- $\delta(q_i, \epsilon, \bot_i) = \{(q_0, \epsilon)\}$, $i \in \{1, 2\}$

The first step of the computation opens two branches, one for verifying that $\#_a = \#_b$ and one for verifying that $\#_b = \#_c$. 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 *abbcccabb*. The left branch has read *abbcc* and shows that one more *b*-s than *a*-s have been read, while the right branch has read *abbccc* and shows that two more *c*-s than *b*-s have been read. Figure 2 shows the configuration describing the state of the automaton.

[4] Note that this is acceptance by empty stack. It is possible to define acceptance by accepting states. Let $F \subseteq Q$ be a set of accepting states. An accepting configuration is of the form (q, ϵ, α) where $q \in F$. Both models of acceptance are equivalent.

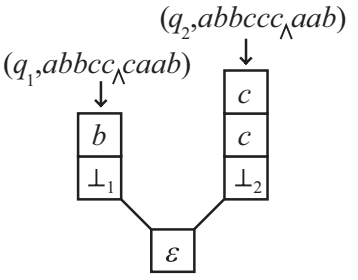

Fig. 1. Intermediate state of a computation on *abbcccaab*

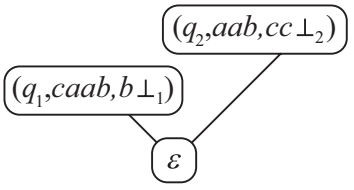

Fig. 2. The configuration matching the state in Fig. 1

4 Main Results

In this section we state the main results of our paper, namely that the SAPDA and CG models are equivalent.

Theorem 1. *If a language is generated by a CG then it is accepted by a SAPDA.*

Theorem 2. *If a language is accepted by an SAPDA then it is generated by a CG.*

The proofs of Theorems 1 and 2 are extensions of the classical ones,[5] see, e.g., [5, Theorem 5.3, pp. 115–116] and [5, Theorem 5.4, pp. 116–119] respectively. Both proofs are omitted due to lack of space.

5 Mildly Context-Sensitive Languages

The field of 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 [9].

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[6]. It is clear that there is a strong relation between the class of languages derived by conjunctive grammars (and accepted by SAPDA) and the class of mildly context sensitive languages. The first criterion of MCSL is obviously met, as both CG and SAPDA

[5] The proof of Theorem 1 is more involved, and requires several preliminary steps.

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

contain their context free counterparts, CFG and PDA respectively. The third criterion is also met by Okhotin's proof that CG membership is polynomial.

Multiple-agreement and cross-agreement are covered as shown in Examples 1, 2 respectively. Reduplication with a center marker is shown in Example 3. Okhotin has conjectured that reduplication without a center marker cannot be generated by any CG. However, this is still an open problem.

Surprisingly, it is the fourth criterion of semi-linearity which is not met, as demonstrated by the following example due to Okhotin [10].

Example 5. The following linear conjunctive grammar derives the non-context-free language $\{ba^2ba^4\cdots ba^{2^n}b | n \in \mathbb{N}\}$. $G = (V, T, P, S)$ where:

- $V = \{S, A, B, C, D, U, V\}$, $T = \{a, b\}$,
- P contains the following derivation rules:

$$S \rightarrow (U \ \& \ V) \mid b$$
$$U \rightarrow Ua \mid Ub \mid b \ ; \ V \rightarrow Ab \mid (B \ \& \ D)$$
$$A \rightarrow aA \mid a \ ; \ B \rightarrow Ba \mid Bb \mid Cb$$
$$C \rightarrow aCa \mid baa \ ; \ D \rightarrow aD \mid bV$$

For more details regarding this example see [10].

The language generated in Example 5 has super-linear growth, which means that, in this respect, CG and SAPDA accept some languages not characterized as mildly context-sensitive. In this respect, it may be that the CG and SAPDA models are too strong for natural language processing.

6 Related Work

6.1 Alternating Grammars

Moriya introduced in [11] the concept of Alternating Context-Free Grammars (ACFG), as a suggested grammatization for APDA. In ACFG, when a conjunctive rule is applied in a derivation, the currently derived formula is duplicated, and each duplicate continues its derivation independently. Therefore, a derivation of a grammar is in fact a tree where a conjunctive rule with k conjuncts yields k child nodes in the derivation tree. The root of the derivation tree is always labelled with the start symbol S. If all leaves of a derivation tree are labelled w then the tree is a derivation of w.

The difference between Moriya's ACFGs and Okhotin's CGs is that conjunctions in CGs are *local* leaving the rest of the thus far derived formula untouched whereas in ACFGs, when a conjunctive rule is applied, the entire formula is duplicated. It is the *locality* of conjunctions in CGs which renders them so similar in many respects to context-free grammars.

In [11], Moriya claimed that ACFG are equivalent to APDA. However, Ibarra et.al. showed in [12] that the equivalence proof was flawed. Namely, the proof was based on the claim that leftmost derivations of ACFGs are equivalent to general derivations. This claim, however, is surprisingly false. The question of

whether ACFG and APDA are equivalent remains an open one. It would seem that ACFG are stronger than CGs (as APDAs are stronger than SAPDAs) but this too has yet to be proven. Possibly the introduction of an automata model for CG can help solve some of these questions.

6.2 Conjunction in Lambek Categorial Grammars

Categorial Grammar is a formal system for analyzing the syntax and semantics of both formal and natural languages. Categorial grammars contain only a small set of universal rules, which are applicable to *all* languages; the differences between languages stemming solely from the lexicon. The universal rules of categorial grammars are treated as a logical calculus. Therefore, the syntactic analysis of an expression is reduced to a logical derivation. For more information see [13].

There are several frameworks for categorial grammars, each defining a set of universal derivation rules. One of the most widely accepted is the (associative) Lambek-calculus (**L**) as defined in [14]. Figure 3 shows the calculus, where Γ (Γ_i) denotes a finite sequence of categories, and c (c_i) denotes a single category. The expression $\Gamma \triangleright c$ is understood as: Γ is reducible to c in the given calculus.

$$(Ax) \quad c \triangleright c$$

$$\frac{\Gamma_1 \triangleright c_1 \qquad \Gamma_2\ c_2\ \Gamma_3 \triangleright c_3}{\Gamma_2\ \Gamma_1\ (c_1 \rightarrow c_2)\ \Gamma_3 \triangleright c_3}\ (\rightarrow \mathrm{L}) \qquad \frac{c_1\ \Gamma \triangleright c_2}{\Gamma \triangleright (c_1 \rightarrow c_2)}\ (\rightarrow \mathrm{R})$$

$$\frac{\Gamma_1 \triangleright c_1 \qquad \Gamma_2\ c_2\ \Gamma_3 \triangleright c_3}{\Gamma_2\ (c_2 \leftarrow c_1)\ \Gamma_1\ \Gamma_3 \triangleright c_3}\ (\leftarrow \mathrm{L}) \qquad \frac{\Gamma\ c_1 \triangleright c_2}{\Gamma \triangleright (c_2 \leftarrow c_1)}\ (\leftarrow \mathrm{R})$$

Fig. 3. Lambek Calculus – Sequent calculus style

Definition 10. *A* Lambek categorial grammar *is a tuple* $G = (\Sigma, \mathcal{B}, c_0, \alpha)$ *where:*

- Σ *is a finite set, the* alphabet
- $\mathcal{B}$ *is a set of* basic categories *with an associated category system* $\mathcal{C}$ *(the reflexive-transitive closure of* $\mathcal{B}$ *under* $\rightarrow$ *and* $\leftarrow$*)*
- c_0 *is the* target category
- $\alpha : \Sigma \longrightarrow P_f(\mathcal{C})$ *is the* lexicon*, a mapping assigning each terminal symbol in* Σ *a finite non-empty subset of categories from* $\mathcal{C}$

The language generated by G *is defined to be:*

$$L(G) = \{w = \sigma_1 \dots \sigma_n \in \Sigma^+ \mid \exists c_1 \dots c_n : c_i \in \alpha[\sigma_i], i = 1 \dots n, \vdash_L c_1 \dots c_n \triangleright c_0\}$$

Lambek categorial grammars have been proven to derive exactly the class of context free languages, see [15,16,17]. However, as context-free languages do not cover all natural language structures, several extensions have been explored. One

$$\frac{\Gamma_1\ c_1\ \Gamma_2 \triangleright c_3}{\Gamma_1\ c_1\ \cap\ c_2\ \Gamma_2 \triangleright c_3}\ (\cap\ \mathrm{L}_1) \qquad \frac{\Gamma_1\ c_1\ \Gamma_2 \triangleright c_3}{\Gamma_1\ c_2\ \cap\ c_1\ \Gamma_2 \triangleright c_3}\ (\cap\ \mathrm{L}_2)$$

$$\frac{\Gamma \triangleright c_1 \qquad \Gamma \triangleright c_2}{\Gamma \triangleright c_1\ \cap\ c_2}\ (\cap\ \mathrm{R})$$

Fig. 4. Kanazawa's intersective conjunction rules

such extension is Kanazawa's work [18] in which he suggests an enrichment of the Lambek calculus with intersective conjunction (see Fig. 4).

Similarly to CG, Kanazawa's grammars accept all finite intersections of context-free languages, e.g., the multiple-agreement and cross-agreement languages etc. Kanazawa also shows that his extended grammars accept languages not obtained as a finite intersection of context-free languages by proving that they can accept the langauge $L = \{a^{2^{n^2}} \mid n \in \mathbb{N}\}$.[7] This is a super-linear language, similarly to the CG generated language from Example 5.

It would be interesting to compare Kanazawa's model to CG and SAPDA as there seem to be many similarities between them. Categorial grammars are used mainly in the field of computational linguistics, so such a comparison could have a bearing on natural language processing as well as formal language theory.

7 Concluding Remarks

We have introduced a synchronized model of Alternating Pushdown Automata, SAPDA, which is equivalent to the CG model. As the exact class of languages generated by CG-s 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.

An interesting direction for further research is the exploration of the relation between LCG and SAPDA. It is a well known result, due to Ginsberg and Spanier [19], that linear grammars are equivalent to 1turn-PDA. 1turn-PDA are a subfamily of PDA where in each computation the stack hight switches only once from non-decreasing to decreasing. A similar notion of 1turn-SAPDA can be defined, where each stack branch can make only one turn in the course of a computation. Our initial results point towards an equivalence between 1turn-SAPDA and LCG. If this equivalence holds, it will deepen the correlation between SAPDA and CG, strengthening the claim that SAPDA are a natural model for CG.

In [20], 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 also prove interesting to explore the general case of finite-turn SAPDA, perhaps finding models for closures of linear conjunctive languages under regular operations.

[7] L is not a finite intersection of context-free languages, as all unary context-free languages are regular, and regular languages are closed under intersection [18].

Acknowledgments

The authors are grateful to Nissim Francez for his remarks on the first draft of this paper. The comments of the anonymous referee which considerably improved the presentation are greatly appreciated.

The work of Michael Kaminski was supported by the fund for promotion of research at the Technion.

References

1. Okhotin, A.: Conjunctive grammars. Journal of Automata, Languages and Combinatorics 6(4), 519–535 (2001)
2. Okhotin, A.: A recognition and parsing algorithm for arbitrary conjunctive grammars. Theoretical Computer Science 302, 81–124 (2003)
3. Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. Journal of the ACM 28(1), 114–133 (1981)
4. Ladner, R.E., Lipton, R.J., Stockmeyer, L.J.: Alternating pushdown and stack automata. SIAM Journal on Computing 13(1), 135–155 (1984)
5. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Reading (1979)
6. Okhotin, A.: Efficient automaton-based recognition for linear conjunctive languages. International Journal of Foundations of Computer Science 14(6), 1103–1116 (2003)
7. Okhotin, A.: On the equivalence of linear conjunctive grammars and trellis automata. RAIRO Theoretical Informatics and Applications 38(1), 69–88 (2004)
8. Culik II, K., Gruska, J., Salomaa, A.: Systolic trellis automata, i and ii. International Journal of Computer Mathematics 15 & 16(1 & 3–4), 3–22, 195–212 (1984)
9. Vijay-Shanker, K., Weir, D.J.: The equivalence of four extensions of context-free grammars. Mathematical Systems Theory 27(6), 511–546 (1994)
10. Okhotin, A.: On the closure properties of linear conjunctive languages. Theor. Comput. Sci. 299(1-3), 663–685 (2003)
11. Moriya, E.: A grammatical characterization of alternating pushdown automata. Theoretical Computer Science 67(1), 75–85 (1989)
12. Ibarra, O.H., Jiang, T., Wang, H.: A characterization of exponential-time languages by alternating context-free grammars. Theoretical Computer Science 99(2), 301–313 (1992)
13. Moortgat, M.: Categorial type logics. In: van Benthem, J., ter Meulen, A. (eds.) Handbook of Logic and Language, Elsevier, Amsterdam (1997)
14. Lambek, J.: The mathematics of sentance structure. American Mathematical Monthly 65(3), 154–170 (1958)
15. Bar-Hillel, Y., Gaifman, C., Shamir, E.: On categorial and phrase structure grammars. Bulletin of the Research Council of Israel 9(F), 1–16 (1960)
16. Cohen, J.M.: The equivalence of two concepts of categorial grammar. Information and Control 10, 475–484 (1967)
17. Pentus, M.: Lambek grammars are context free. In: Proc. of 8th Ann. IEEE Symp. on Logic in Computer Science, pp. 429–433 (1993)

18. Kanazawa, M.: The lambek calculus enriched with additional connectives. Journal of Logic, Language and Information 1(2), 141–171 (1992)
19. Ginsgurg, S., Spanier, E.H.: Finite-turn pushdown automata. SIAM Journal on Control 4(3), 429–453 (1966)
20. Kutrib, M., Malcher, A.: Finite-turn pushdown automata. Discreet Applied Mathematics 155, 2152–2164 (2007)