Модел. и анализ информ. систем. Т. **22**, № **2** (2015) 176–196
©*Dekhtyar M. I., Karlov B. N.,* 2015

UDC 519.766:519.713.1

PDA with Independent Counters

Michael Dekhtyar, Boris Karlov [1]

Dept. of Computer Science, Tver State University,
Zhelyabova str., 33, Tver, Russia, 170000.

e-mail: *Michael.Dekhtyar@tversu.ru, bnkarlov@gmail.com*

received February 15, 2015

Keywords: automata, formal grammars and languages, push-down automata with independent counters, projective and non-projective dependency, categorial dependency grammar, multimodal categorial dependency grammar, push-down automata with stacks of independent counters.

Is dedicated to the memory of Alexander Dikovsky (1945–2014)

Push-down automata with independent counters (PDACs) combine the power of PDAs and Petri Nets. They were developed in [21, 15], as a tool of recognition of languages generated by Categorial Dependency Grammars (CDGs). CDGs are classical categorial grammars extended by oriented polarized valencies. They express both projective and non-projective dependencies between the words of a sentence. PDAC is a usual PDA equipped with a finite number of counters. The independence of counters means that their state has no effect on the choice of an automaton move. In the first part of the paper we compare some variants of PDACs and prove the equivalence of two variants of PDAs with independent counters: without syntactic and without semantic ε-loops. Some connections between PDAC-languages and Petri Net languages are noticed. Then we show that PDACs are equivalent to stack+bag push-down automata (SBPA) independently introduced by Søgaard and that ε-acyclic SBPAs recognize exactly CDG-languages.

Multimodal Categorial Dependency Grammars (mmCDGs) were introduced in [4] as an extension of GDGs that allows control of some intersections of dependencies. The class of mmCDG-languages is rich enough and has good closure properties, that it forms AFL. In the second part of the paper we extend PDACs and introduce push-down automata with stacks of independent counters (PDASC). PDASCs extend PDACs twofold: (i) each counter is a stack of integers and (ii) there is a restriction function which allows to diminish a head of a counter only if the heads of all dependent counters are zeros. Our main result says that these PDASCs accept exactly the class of mmCDG-languages.

The article is published in the author's wording.

[1]This work was sponsored by the Russian Foundation of Basic Research (Grants No. 13-01-00643 and 13-01-00382).

Introduction

Push-down automata with independent counters (PDAC) are a natural extension of push-down automata (PDA), the class of automata which recognizes context-free languages (see, e.g. [11, 20]). They were developed by the second author ([21, 15]) as a tool for recognition of languages generated by Categorial Dependency Grammars (CDGs). Because these automata combine PDAs with Petri Nets, we believe that they are an interesting object of study by themselves.

Dependency grammars (DGs) are formal grammars assigning *dependency trees* (DTs) to well-formed sentences. A DT of a sentence is a labeled arrows tree whose nodes are the words of the sentence. A formal description of DGs and the DG syntax was defined by Tesnière [17]. It is well-known that the DTs assigned to constituent structures by context-free (cf-) grammars are always *projective*: the projections of words fill continuous segments. Meanwhile, discontinuous non-projective dependencies are inevitable in languages. They often mark communicative structure and special constructs encoding complex semantic relations.

The necessity of treating non-projective dependencies in grammars led to many proposals on extending cf-grammars and push-down automata. One of the best known is the class of Tree Adjoining Grammars (TAGs) [13]. For this class Embedded Push-Down Automata (EPDA) have been introduced in [18]. EPDAs recognize the class of Tree-Adjoining Languages (TALs). EPDAs extend push-down automata (PDA) by replacing the single push-down store used in PDAs with a stack of non-empty push-down stores. TAGs and EPDAs can handle some non-projective dependencies. But their power is not enough to recognize some languages with the intensive cross-serial dependencies. E.g., it was recently shown in [18] that TALs do not include language $MIX_3 \subset \{a, b, c\}^*$ containing all the words with equal numbers of the symbols a, b and c. A detailed comparison of TAGs, EPDAs and many others extensions of cf-grammars and PDAs can be found in [18, 12].

The *Categorial Dependency Grammars* (**CDGs**) were originally introduced by Dikovsky [6]. Their mathematical properties were studied by Dekhtyar, Dikovsky and Karlov in [2, 3, 21]. They showed that the family of CDG-languages is closed under all AFL operations, except for iteration[2]. Additionally, Karlov defined the class of Push-down Automata with Independent Counters (PDAC) and proved that these automata accept exactly all CDG-languages [21]. PDA with independent counters is a usual PDA equipped with a finite number of counters. The independence of counters means that their state has no effect on the choice of an automaton move. An equivalent class of *Stack+Bag Push-down Automata* was introduced by Søgaard independently [16].

In [7, 4] the multimodal extensions of CDGs (mmCDGs) were defined. They add to CDGs cross prohibition functions which prevent from intersections of some dependencies in DTs. It was shown that mmCDG-languages are closed under iteration, and their expressive power and complexity were investigated.

At the same time, the problem of characterizing the mmCDG-languages by appropriately extending push-down automata remained open. In this paper we provide such a characterization by defining Push-down Automata with Stacks of Independent Counters (PDASC), and extension of PDAC. These automata are push-down automata equipped

[2]The closure under iteration still is unknown.

with finite number of counter stacks. Cells of the stacks contain numbers which can be increased or decreased. At the same time, the contents of the stacks do not influence the selection of the automaton instruction. The automata recognize the words in the language by emptying the stack and zeroing all the counter stacks. Our main result states that PDASC accept exactly all mmCDG-languages.

The rest of this paper is organized as follows: in Section 2 we give definitions of CDGs and mmCDGs and their languages. In Section 3 we consider some variants of PDACs and prove the equivalence of two variants of PDACs with independent counters: without syntactic and without semantic ε-loops. We also notice that PDAC-languages include all intersections of cf-languges and Petri Net languages and that every PDAC-language can be obtained by a homomorphism of the intersection of a cf-language with some special Petri Net language. In Section 4 we show that PDACs are equivalent to stack+bag push-down automata (SBPA) independently introduced by Søgaard and that ε-acyclic SBPA recognize exactly CDG-languages. In Section 5 we introduce a new class of automata: Push-down Automata with Stacks of Independent Counters (PDASCs) without empty loops. PDASCs are illustrated by an automaton which accepts the language $\{a^n b^n c^n\}^+$. In Section 6 we prove our main result which says that PDASCs accept exactly the class of languages generated by mmCDGs.

1. CDG and mmCDG

Similarly to other categorial grammars [1], the *categorial dependency grammars* (*CDG*) may be seen as assignments of dependency types to words. Every dependency type assigned to a word w defines its possible *local neighborhood* in a grammatically correct dependency structure. The neighborhood of w consists of the *incoming* dependency, i.e. the dependency relation d through which w is subordinate to a word g, its *governor*, and also of a sequence of *outgoing* dependencies, i.e. the dependency relations d_i through which w governs subordinate words w_i. In order to formalize the linguistic notion of the syntactic type, we use the notion of the category. Let $\mathbf{C}$ be a nonempty finite set of *elementary categories* (e.g. subject, predicate, complement). The elementary categories can be iterated: for $C \in \mathbf{C}$, C^* means a corresponding *iterated category*. The set of all iterated categories is denoted $\mathbf{C}^*$. Elementary and iterated categories are combined in *base (local) categories* with the constructors $\backslash$ and $/$.

In CDGs and mmCDGs, the non-projective dependencies are expressed using so called polarized valencies. Namely, in order that a word G may govern through a discontinuous dependency d a word D that follows the word G somewhere in the sentence, G should have a type declaring the *positive* valency $\nearrow d$, whereas its subordinate D should have a type declaring the *negative* valency $\searrow d$. Together these *dual* valencies define the discontinuous *right* dependency d. Additionally, there is another pair of *dual* polarized valencies $\nwarrow d$ and $\swarrow d$ which defines the discontinuous *left* dependency d.

Definition 1 *Let $\mathbf{C}$ be a set of* elementary (dependency) categories*. $S \in \mathbf{C}$ is the selected category of sentences.*

For each elementary category d the category d^* is iterated*.*

Each elementary category or ε is base*. If a category C is base and a category α is elementary or iterated, then the categories $[\alpha\backslash C]$ and $[C/\alpha]$ are also base. There are no* *other base categories. The set of base categories over $\boldsymbol{C}$ is denoted $bCat(\mathbf{C})$.*

Polarized valencies are expressions $\swarrow d$, $\searrow d$, $\nwarrow d$, $\nearrow d$, where $d \in \mathbf{C}$. The set of polarized valencies over $\mathbf{C}$ is denoted $V(\mathbf{C})$. Strings $\theta \in Pot(\mathbf{C}) = V(\mathbf{C})^*$ are called potentials.

A (general) category is either base category or has the form C^{θ}, where θ is a potential and C is a base category. The set of general categories over $\mathbf{C}$ is denoted $Cat(\mathbf{C})$.

CDG assigns to each word in its dictionary a finite set of categories.

Definition 2 *A categorial dependency grammar (CDG) is a system $G = (W, \mathbf{C}, S, \lambda)$, where W is a finite set of words, $\mathbf{C}$ is a finite set of elementary categories containing the selected name S (an axiom), λ, called lexicon, is a finite substitution on W such that $\lambda(a) \subset Cat(\mathbf{C})$ for each word $a \in W$.*

If $\lambda(a) = \{\gamma_1, \ldots, \gamma_n\}$, then we write $a \mapsto \gamma_1, \ldots, \gamma_n$.

CDG proofs are defined using the following calculus of dependency types [3].

Definition 3 *Let Γ_1, Γ_2 be strings of categories $Cat(\mathbf{C})^*$, $\theta, \theta_1, \theta_2, \theta_3$ be potentials, α be local category from $bCat(\mathbf{C})$.*

Local dependency rules:
$\mathbf{L^l}$: $\Gamma_1[C]^{\theta_1}[C\backslash\alpha]^{\theta_2}\Gamma_2 \vdash \Gamma_1[\alpha]^{\theta_1\theta_2}\Gamma_2$,
where $C \in \mathbf{C} \cup \{\varepsilon\}$

Iterated dependency rules:
$\mathbf{I^l}$: $\Gamma_1[C]^{\theta_1}[C^*\backslash\alpha]^{\theta_2}\Gamma_2 \vdash \Gamma_1[C^*\backslash\alpha]^{\theta_1\theta_2}\Gamma_2$
$\mathbf{I^l_0}$: $\Gamma_1[C^*\backslash\alpha]^{\theta}\Gamma_2 \vdash \Gamma_1[\alpha]^{\theta}\Gamma_2$,
where $C \in \mathbf{C} \cup \{\varepsilon\}$

Discontinuous dependency rules:
$\mathbf{D^l}$. $\alpha^{P_1(\swarrow C)P(\nwarrow C)P_2} \vdash \alpha^{P_1 P P_2}$,
where the potential $(\swarrow C)P(\nwarrow C)$ satisfies the following pairing rule **FA** *(*first available*):*

$$\mathbf{FA}: \quad P \text{ has no occurrences of } \swarrow C, \nwarrow C.$$

Intuitively, the rules $\mathbf{L^l}$ ($\mathbf{L^r}$) correspond to the classical elimination rules of categorial grammars. Eliminating the argument subtype C they construct the (`projective`) dependency C in which the governor is the word with the functional type and the subordinate is the word with the argument type. At the same time, they concatenate the potentials of these types (if any). The rules $\mathbf{I^l}, \mathbf{I^l_0}$ ($\mathbf{I^r}, \mathbf{I^r_0}$) derive the iterated (projective) dependencies. $\mathbf{I^l}$ ($\mathbf{I^r}$), analogous to the rule $\mathbf{L^l}$ ($\mathbf{L^r}$), may derive $k > 0$ dependencies C and $\mathbf{I^l_0}$ ($\mathbf{I^r_0}$) corresponds to the case $k = 0$. $\mathbf{D^l}$ ($\mathbf{D^r}$) creates *discontinuous dependencies.* It pairs and eliminates dual valencies with name C satisfying the rule **FA** to create the discontinuous dependency C.

When one of these rules is applied, an edge is added into the dependency structure. This edge goes from the governor to the subordinate word and is labeled with the name of the canceled category.

This calculus defines the immediate provability relation $\vdash$ on the strings of categories. Its transitive closure $\vdash^*$ underlies the following definition of CDG-languages.

[3] We show left-oriented rules. The right-oriented rules are symmetrical.

Definition 4 *CDG G generates the language $L(G)$, consisting of all words $w = w_1w_2 \dots w_n \in W^*$, such that for some string of categories $\Gamma \in \lambda(w) = \lambda(w_1)\lambda(w_2)\dots\lambda(w_n)$ there is a proof $\Gamma \vdash^* S$. Let $\mathcal{L}(CDG)$ be the class of all CDG-languages.*

The following example shows a simple CDG that generates a non cf-language.

Example 1 *Let CDG $G_{abc} = (\{a, b, c\}, \{S, A, B, C\}, S, \lambda_{abc})$ where λ_{abc} :*
$a \mapsto [S/A]^{\nearrow A}, [S/B]^{\nearrow A}, [A/A]^{\nearrow A}, [A/B]^{\nearrow A}$
$b \mapsto [B/B]^{\searrow A \nearrow B}, [B/C]^{\searrow A \nearrow B}$
$c \mapsto [C/C]^{\searrow B}, [C]^{\searrow B}$

This grammar generates language $L_{abc} = \{a^nb^nc^n \mid n \geq 1\}$. Fig. 1 shows the categories assignment for the word $a^3b^3c^3$ and the dependency structure of the word built by G_{abc}.

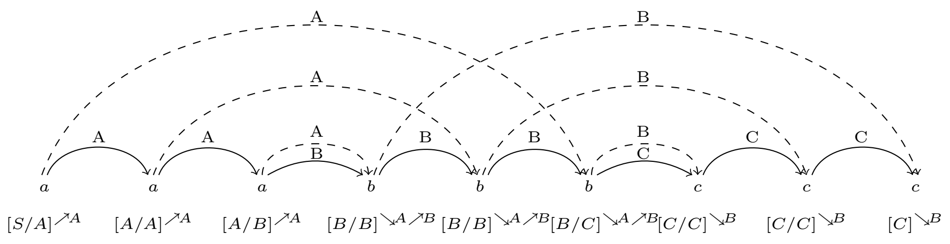

Fig. 1. Dependency structure for $w = aaabbbccc$

A number of properties of CDGs and languages of $\mathcal{L}(CDG)$ were established in [2, 3, 21, 15]. Specifically, $\mathcal{L}(CDG)$ is closed under all Abstract Family of Languages (AFL) [8] operations except iteration[4]. The notion of PDA with independent counters was defined and it was shown that these automata accept exactly CDG-languages.

But it turned out that pairing rule **FA** can not prevent generation of some unwanted dependencies, e.g. dependencies between two words of different sentences. So, in [4] a new class of multimodal categorial dependency grammars (mmCDG) was introduced. mmCDG extends CDG with a cross prohibition function π of type $\pi : \mathbf{C} \to 2^{\mathbf{C}}$. If $D \in \pi(C)$ and $C \in \pi(D)$ then dependencies C and D should not intersect in the dependency structures.

We adopt the definition of mmCDG from [4].

Definition 5 *A multimodal categorial dependency grammar (mmCDG) is a system $G = (W, \mathbf{C}, S, \lambda, \pi)$, where $W, \mathbf{C}, S$ and λ are as in CDGs, and $\pi : \mathbf{C} \to 2^{\mathbf{C}}$ is a cross prohibition function.*

The function π should be symmetrical, i.e. if $C \in \pi(D)$, then $D \in \pi(C)$ for all C and D from **C**.

The calculus of dependency types for mmCDG proofs includes rules $\mathbf{L^l}, \mathbf{L^r}, \mathbf{I^l}, \mathbf{I^r}, \mathbf{I^l_0}, \mathbf{I^r_0}$ and discontinuous dependency rules of the form

$\mathbf{D}_{\mathbf{FA}^l_{C:\pi(C)}}$: $\quad \alpha^{\theta_1(\nearrow C)\theta(\searrow C)P_2} \vdash \alpha^{\theta_1\theta\theta_2}$,

[4]We believe that it is not closed. But it is still an open problem.

where $\theta_1(\nearrow C)\theta(\searrow C)$ satisfies the pairing rule $\mathbf{FA}_{C:\pi(C)}$: θ has no occurrences of $\nearrow C$, $\searrow C$ and also of $\swarrow A, \nwarrow A, \nearrow A, \searrow A$ for all $A \in \pi(C)$.

As in the case of CDG rules, $\mathbf{D}_{\mathbf{FA}^l_{C:\pi(C)}}$ ($\mathbf{D}_{\mathbf{FA}^r_{C:\pi(C)}}$) derive non-projective dependencies. They pair dual valencies C under the negative condition that the resulting discontinuous dependency C *does not intersect the discontinuous dependencies in the set* $\pi(C)$.

Let $\mathcal{L}(mmCDG)$ be the class of all mmCDG-languages.

In [4] it was shown that $\mathcal{L}(mmCDG)$ is closed under all AFL operations including iteration. The following example shows how mmCDG can generated the iteration of the language of Example 1.

Example 2 *Let* $L_1 = L^+_{abc} = \{ a^n b^n c^n \mid n > 0 \}^+$. *It is generated by the following mmCDG* $G_1(\{a,b,c\}, \{S,A,B,C\}, S, \lambda_1, \pi_1)$, *where* λ_1 :
$a \mapsto [S/A]^{\nearrow C \nearrow A}, [S/B]^{\nearrow C \nearrow A}, [A/A]^{\nearrow A}, [A/B]^{\nearrow A}$
$b \mapsto [B/B]^{\searrow A \nearrow B}, [B/C]^{\searrow A \nearrow B}$
$c \mapsto [C/C]^{\searrow B}, [C]^{\searrow B \searrow C}, [C/S]^{\searrow B \searrow C \nearrow C}$
and $\pi_1(A) = \{ C \}$, $\pi_1(B) = \{ C \}$, $\pi_1(C) = \{ A, B \}$, $\pi_1(S) = \emptyset$.

G_1 extends G_{abc} of example 1 with a new pair of dual valencies $\nearrow C$, $\searrow C$ which due to the prohibition function π defends subwords of the form $a^n b^n c^n$ against the penetration of dependencies A and B from the adjacent words. It can be seen on Fig. 2 which presents the dependency structure that G_1 assigns to string $w = aabbccabc$.

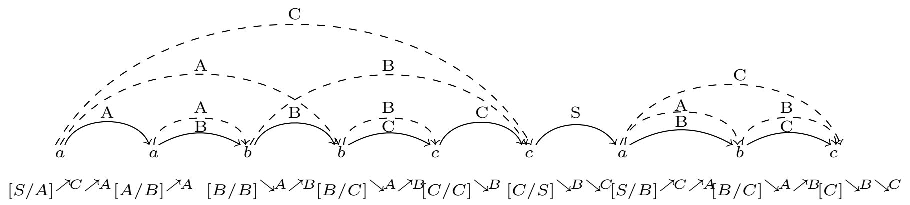
C
A
B
C
A
A
B
B
B
C
C
S
A
B
B
C
a a b b c c a b c
[S/A]↗C↗A [A/B]↗A [B/B]↘A↗B [B/C]↘A↗B [C/C]↘B [C/S]↘B↘C [S/B]↗C↗A [B/C]↘A↗B [C]↘B↘C

Fig. 2. Dependency structure for $w = aabbccabc$

Definition 6 *Let* π *be a cross prohibition function,* θ *be a potential. A potential* θ *is called* balanced with respect to π, *if its projection on every pair of dual polarized valencies is a word of well-matched valencies, and there exists an order on the set of the correct pairs such that they can be removed from potential without violating the restrictions of* π.

In other words, a potential θ is balanced with respect to π, if there is a proof $a^\theta \vdash^* a^\varepsilon$ which uses the rules $\mathbf{D}_{\mathbf{FA}^l_{C:\pi(C)}}$ and $\mathbf{D}_{\mathbf{FA}^r_{C:\pi(C)}}$ only.

It is easy to see that if the cross prohibition function of mmCDG G is empty, i.e. $\pi(C) = \emptyset$ for all $C \in \mathbf{C}$, then discontinuous dependency rules $\mathbf{D}_{\mathbf{FA}^l_{C:\pi(C)}}$ are transformed into the First Available (FA-) principle and then G is a CDG.

In [4] we showed also that $\mathcal{L}(mmCDG)$ includes some non-semilinear languages and that there is a *mmCDG* G such that the membership problem for $L(G)$ is NP-complete. At the same time, until this paper, no class of automata for accepting languages from $\mathcal{L}(mmCDG)$ has been proposed.

2. Push-down Automata with Independent Counters

The notion of PDA with independent counters was introduced in [21, 15]. Informally, the PDA with independent counters is a usual PDA equipped with a finite number of counters. The independence of counters means that their state has no effect on the choice of an automaton move.

Let $\mathbf{Z}$ be the set of all integers and $\mathbf{N}$ be the set of all nonnegative integers.

Definition 7 *A push-down automaton with independent counters (PDAC) is a 7-tuple* $M = \langle \Sigma, Q, \Gamma, q_0, z_0, P, n\rangle$, *where* Σ *is an input alphabet,* Q *is an alphabet of states,* Γ *is a stack alphabet,* $q_0 \in Q$ *is an initial state,* $z_0 \in \Gamma$ *is an initial symbol of the stack,* P *is a set of rules,* n *is a number of counters.*
The rules are of the form $\langle q, a, z, \langle q', \alpha, \bar{v}\rangle\rangle$, *where* $q, q' \in Q$, $a \in \Sigma \cup \{\,\varepsilon\,\}$, $z \in \Gamma$, $\alpha \in \Gamma^*$, $\bar{v} = (v_1, \ldots, v_n) \in \mathbf{Z^n}$ *is a vector of integers.*

Informally speaking, this is a push-down automaton additionally augmented with a finite number of counters. It uses its stack to check the elimination of local categories, and the counters correspond to different types of valencies.

Definition 8 *A configuration of PDAC* $M = \langle \Sigma, Q, \Gamma, q_0, z_0, P, n\rangle$ *is a quadruple* $\langle q, w, \gamma, \bar{u}\rangle$, *where* $q \in Q$, $w \in \Sigma^*$, $\gamma \in \Gamma^*$, $\bar{u} = (u_1, \ldots, u_n) \in \mathbf{N^n}$ *is a vector of nonnegative integers.*
We define a one-step transition: $\langle q, w, \gamma, \bar{u}\rangle \vdash^1_M \langle q', w', \gamma', \bar{u}'\rangle$ *iff there exists a rule* $\langle q, a, z, \langle q', \alpha, \bar{v}\rangle\rangle \in P$ *such that the following three conditions are satisfied:*
1) $w = aw'$,
2) $\gamma = z\gamma''$, $\gamma' = \alpha\gamma''$,
3) $\bar{u}' = \bar{u} + \bar{v}$.
If $\gamma = \varepsilon$ *or some component of* $\bar{u}'$ *is negative, then the step cannot be made.*
The relations $\vdash^n_M$ *of* n*-steps derivations of* M *and* $\vdash^*_M$ *of derivations of* M *are defined as usual.*

In fact, the numbers in the counters are the numbers of currently unpaired left valencies. The positive numbers in the rules correspond to the left valencies, and the negative numbers correspond to the right valencies. The automaton works like a push-down automaton. Additionally it changes the values of the counters on every step, but the step itself is not influenced by these values, which means that the counters are independent.

The language accepted by the push-down automaton with independent counters M can be defined by emptying the stack and zeroing the counters.

Definition 9 *The word* w *is accepted by PDAC* M *iff there exists* $q \in Q$ *such that* $\langle q_0, w, z_0, (0, \ldots, 0)\rangle \vdash^*_M \langle q, \varepsilon, \varepsilon, (0, \ldots, 0)\rangle$.

The language $L(M)$ *accepted by PDAC* M *is the set of all the words accepted by the automaton.*

In general, it is possible that the automaton performs ε-instructions in a cycle and changes the counters. In this case it can increase the counters by an unbounded amount without reading new symbols. But all potentials in the CDGs have finite length. One can propose two kinds of restrictions to avoid such empty loops. The first definition is syntactical and was used in [15].

Definition 10 *We say that PDAC M has a syntactic ε-loop if there exists a sequence of states $q_1, \ldots q_k$ $(k > 1)$ such that $\langle q_i, \varepsilon, z_i, \langle q_{i+1}, \gamma_i, \bar{v}_i\rangle\rangle \in P$ for $1 \le i < k$, $\langle q_k, \varepsilon, z_k, \langle q_1, \gamma_k, \bar{v}_k\rangle\rangle \in P$ and for some $i \in [1, k]$ $\bar{v}_i \ne (0, 0, \ldots, 0)$.*

The second one is semantical and follows Søgaard of [16].

Definition 11 *We say that the PDAC M has a semantic ε-loop if for some $q \in Q$, $\alpha, \beta \in \Gamma^*$, and vectors $\bar{u}, \bar{v}$ there is a derivation of M $\langle q, \varepsilon, \alpha, \bar{u}\rangle \vdash^+ \langle q, \varepsilon, \beta, \bar{v}\rangle$ that consists only of ε-instructions and at least one of these instructions changes at least one counter.*

It is easy to see that if the automaton has no syntactic ε-loops, then it has no semantic ε-loops. The inverse assertion does not hold. E.g. if the automaton has the instructions $\langle q, \varepsilon, a, \langle q_1, a, \bar{u}\rangle\rangle$, $\langle q_1, \varepsilon, b, \langle q, b, \bar{v}\rangle\rangle$, then they form a syntactic ε-loop, but there are no semantic loops because of different stack symbols.

Nevertheless, the following assertion holds.

Theorem 1 *For every PDAC without semantic ε-loops there exists an equivalent PDAC without syntactic ε-loops.*

Proof. Let us consider an arbitrary PDAC without semantic ε-loops $M = \langle \Sigma, Q, Z, q_0, z_0, P, n\rangle$. We may suppose that the initial state q_0 cannot be revisited by the automaton. We build an auxiliary graph $G = (V, E)$, where
$V = \{\langle q, \varepsilon, z, \langle q', \gamma, \bar{v}\rangle\rangle \mid \langle q, \varepsilon, z, \langle q', \gamma, \bar{v}\rangle\rangle \in P\}$, $E = \{(\langle q, \varepsilon, z, \langle q', \gamma, \bar{v}\rangle\rangle, \langle q', \varepsilon, z', \langle q'', \gamma', \bar{v}'\rangle\rangle) \mid \langle q, \varepsilon, z, \langle q', \gamma, \bar{v}\rangle\rangle, \langle q', \varepsilon, z', \langle q'', \gamma', \bar{v}'\rangle\rangle \in V\}$.

Let $p_1, p_2, \ldots, p_k$ be all simple cycles in G. The amount k of such cycles is finite. Let $p_i = \langle q_1, \varepsilon, z_1, \langle q_2, \gamma_1, \bar{v}_1\rangle\rangle, \ldots, \langle q_m, \varepsilon, z_m, \langle q_1, \gamma_m, \bar{v}_m\rangle\rangle$ be one of these cycles. We add $m-1$ new states (i, q_2), (i, q_3), $\ldots$, (i, q_m). These states are different for every cycle. We remove all instructions of the cycle p_i from the set of instructions, and we add the following instructions:
$\langle q_1, \varepsilon, z_1, \langle (i, q_2), \gamma_1, \bar{v}_1\rangle\rangle$,
$\langle (i, q_2), \varepsilon, z_2, \langle (i, q_3), \gamma_2, \bar{v}_2\rangle\rangle$,
$\ldots$
$\langle (i, q_{m-1}), \varepsilon, z_{m-1}, \langle (i, q_m), \gamma_{m-1}, \bar{v}_{m-1}\rangle\rangle$.
Besides, for every instruction $\langle q_j, a, z, \langle q, \gamma, \bar{v}\rangle\rangle$, where $a \in \Sigma$, q_j is one of the states from p_i, $2 \le j \le m$, we add a new instruction $\langle (i, q_j), a, z, \langle q, \gamma, \bar{v}\rangle\rangle$. Let us denote the new automaton M'.

Obviously, M' has no syntactic ε-loops. When M' begins to perform a sequence of ε-instructions, it remembers the ε-path in the states. If it reaches the state (i, q_m), it cannot return to q_1 closing the loop, because the last ε-instruction of the cycle was removed. But M could neither move to q_1 due to absence of semantic ε-loops. If M

"chooses" not to perform the loop, but to read a symbol in the state q_j, M' may "forget" the path which leads it to q_j and continue working exactly like M. Thus, $L(M) = L(M')$. $\square$

Let $\mathcal{L}(PDAC)$ be the class of languages accepted by push-down automata with independent counters and without (syntactic or semantic) empty loops.

Another way is to define acceptance by final state and final counter states. Let pushdown automaton with independent counters and final states (PDACF) $M = \langle \Sigma, Q, \Gamma, q_0, z_0, P, n, F, C_f \rangle$ be PDAC extended with a set of final states $F \subset Q$ and a finite set of accepted counter states $C_f \subset \mathbf{N^n}$.

Definition 12 *The word w is accepted by PDACF M by means of the terminal states if $\langle q_0, w, z_0, (0, \dots, 0)\rangle \vdash_M^* \langle q', \varepsilon, \gamma, \bar{u}\rangle$ for some state $q' \in F$, counters states $\bar{u} \in C_f$ and any stack string γ.*
The language $LF(M)$ accepted by PDACF M is the set of all the words accepted by the automaton.

Let $\mathcal{L}_F(PDACF)$ be the class of languages accepted by PDACF without empty loops. As in the case of PDAs, it can be shown that definitions 9 and 12 are equivalent.

Proposition 1 *$\mathcal{L}(PDAC) = \mathcal{L}_F(PDACF)$.*

In [15] it was proved that the classes $\mathcal{L}(CDG)$ and $\mathcal{L}(PDAC)$ are "almost" equal.

Theorem 2 *1) $\mathcal{L}(PDAC) \subseteq \mathcal{L}(CDG)$.*
2) If $L \in \mathcal{L}(PDAC)$, then $L - \{\varepsilon\} \in \mathcal{L}(CDG)$.

Now we list some properties of $\mathcal{L}(PDAC)$ which follows from theorem 2 and the results obtained in the papers [2, 3, 21, 15]:

- $\mathcal{L}(PDAC)$ includes all cf-languages and some non cf- and non TAG-languages;
- let $W = \{a_1, a_2, \dots, a_n\}$, then languages $L_n = \{a_1^k a_2^k \dots a_n^k \mid k > 0\} (n \geq 1)$ and $MIX_n = \{w \in W^+ \mid |w|_{a_1} = |w|_{a_2} = \dots = |w|_{a_n}\}$ are in $\mathcal{L}(PDAC)$;
- the parsing problem for $\mathcal{L}(PDAC)$ in the general case is NP-complete;
- for languages accepted by PDACs with bounded numbers of counters there is a parsing algorithm that has polynomial complexity;
- $\mathcal{L}(PDAC)$ is closed under union, concatenation, intersection with regular languages, ε-free homomorphisms, and inverses of homomorphisms.

If PDAC does not use its stack, then it turns into a counter automaton without ε-loops. A direct comparison shows that this class of counter automata is equivalent to the class of *prompt Weak Counter Automata (WCA)* defined by Hack in the well-known report [10]. Theorem 9.12 of the report says that the languages generated by prompt WCA are the family of Petri Net languages $\mathcal{L}_0$ [5] completed by ε. Therefore, $\mathcal{L}(PDAC)$ includes all cf-languages as well as all Petri Net languages from $\mathcal{L}_0$. It can be shown that $\mathcal{L}(PDAC)$ also includes the intersection of these classes.

[5] $\mathcal{L}_0$ is the class of all ε-free languages obtained as the set of all terminal label sequences of a ε-free Labelled Petri Net.

Proposition 2 *Let L_1 be a cf-language and L_2 be a Petri Net language from $\mathcal{L}_0$. Then $L = L_1 \cap L_2 \in \mathcal{L}(PDAC)$.*

On the other hand, it follows from the results of [21] that for every $L \in \mathcal{L}(PDAC)$ there exist a cf-language L_1, a Petri Net language $P \in \mathcal{L}_0$ and a homomorphism ϕ such that $L = \phi(L_1 \cap P)$.

In fact, for some n P is the language P_n of words in the alphabet of n pairs of parentheses, whose projections on any single pair of parentheses are well-matched. It is easy to see that P_n can be recognized by our counter automaton with n counters.

3. PDAC and stack+bag push-down automata

In the paper [16] A. Søgaard introduced a *stack+bag push-down automaton* (SBPA) as a 6-tuple $P = \langle \Sigma, Q, \Gamma, q_0, F, \delta \rangle$ where Σ is an input alphabet, Q is an alphabet of states, Γ is a stack alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of the terminal states, and $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \times Q \times (\Gamma^* \cup \{\varepsilon\}) \times \{\{\gamma_1, \dots, \gamma_n\}_M \mid \gamma_1, \dots, \gamma_n \in \Gamma, n \geq 0\}$ is a finite set of transitions, where $\{\dots\}_M$ is a bag or a multiset, i.e. $\{\{\gamma_1, \dots, \gamma_n\}_M \mid \gamma_1, \dots, \gamma_n \in \Gamma, n \geq 0\}$ is the set of multisets over elements of Γ.

A configuration of SBPA has the form $(q, w, \gamma, \gamma') \in Q \times \Sigma^* \times \Gamma^* \times \{\{\gamma_1, \dots, \gamma_n\}_M \mid \gamma_1, \dots, \gamma_n \in \Gamma, n \geq 0\}$, where q is the state the SBPA is currently in, w is the input string still to be processed, γ is the contents of the stack, and γ' is the contents of the bag. The derivability relation is the transitive, reflexive closure ($\vdash^*$) of the following binary relation $\vdash$ over the class of all configurations, where

1. $(q, xw, z\gamma, \gamma') \vdash (q', w, \alpha\gamma, \gamma')$ if $(q', \alpha, \emptyset_M) \in \delta(q, x, z)$ (pop z from stack, push α to stack),

2. $(q, xw, z\gamma, \gamma') \vdash (q', w, \gamma, \alpha' \cup \gamma')$ if $(q', \varepsilon, \alpha') \in \delta(q, x, z)$ (pop z from stack, push α' to bag),

3. $(q, xw, \gamma, \{z\} \cup \gamma') \vdash (q', w, \alpha\gamma, \gamma')$ if $(q', \alpha, \emptyset_M) \in \delta(q, x, z)$ (pop z from bag, push α to stack),

4. $(q, xw, \gamma, \{z\}_M \cup \gamma') \vdash (q', w, \gamma, \alpha' \cup \gamma')$ if $(q', \varepsilon, \alpha') \in \delta(q, x, z)$ (pop z from bag, push α' to bag)

with $x \in \Sigma \cup \{\varepsilon\}$, $z \in \Gamma \cup \{\varepsilon\}$, $\alpha \in \Gamma^*$, and $\alpha' \in \{\{\gamma_1, \dots, \gamma_n\}_M \mid \gamma_1, \dots, \gamma_n \in \Gamma, n \geq 0\}$.

A SBPA S recognizes the language:
$L(S) = \{w \mid (q_0, w, \varepsilon, \emptyset_M) \vdash^* (q, \varepsilon, \varepsilon, \emptyset_M) \wedge q \in F\}$.

The languages that can be recognized by SBPAs are called stack+bag push-down languages.

When comparing the definitions of SBPA and PDAC, one cannot help but notice their similarity. In Theorems 3 and 4 we show that these classes of automata are equivalent.

Theorem 3 *For every SBPA one can effectively construct an equivalent PDAC.*

Proof. Let $S = \langle \Sigma, Q, \Gamma, q_0, F, \delta \rangle$ be a SBPA. Let $\Gamma = \{ z_1, \dots, z_k \}$ and $z_0 \notin \Gamma$ be a new stack symbol. We build a PDAC $M = \langle \Sigma, Q', \Gamma \cup \{ z_0 \}, q_0, P, k \rangle$. Here Q' is a new set of

states. It contains Q and some new states which will be defined later. The number of counters k is $|\Gamma|$.
1) Let $(q', \alpha, \emptyset_M) \in \delta(q, x, z_i)$. This instruction can be performed in two ways. The automaton may replace the top of stack z_i by the word α, or it may remove z from the bag and push α into the stack. We add the following instruction in P.
$\langle q, x, z_i, \langle q', \alpha, \bar{0}\rangle\rangle$
$\langle q, x, u, \langle q', \alpha u, (0, \dots, 0, -1, 0, \dots, 0)\rangle\rangle$ for every $u \in \Gamma \cup \{ z_0 \}$ (here -1 is on i-th place)
The first instruction corresponds to the point 1 of definition of derivability relation for SBPA, and the second one corresponds to the point 3.
2) Let $(q', \varepsilon, \alpha') \in \delta(q, x, z_i)$. The first way to perform the instruction is to pop z_i from the stack and add α' to the bag, the second way is to remove z_i from the bag and add α' there. Let $\bar{v}$ be a vector such that v_j is the number of elements z_j in the bag α' $(j = 1, \dots, |\Gamma|)$. We add the following instructions to P.
$\langle q, x, z_i, \langle q', \varepsilon, \bar{v}\rangle\rangle$
$\langle q, x, u, \langle q'', u, (0, \dots, 0, -1, 0, \dots, 0)\rangle\rangle$ for every $u \in \Gamma \cup \{ z_0 \}$ (-1 is on i-th place)
$q'', \varepsilon, u, \langle q', u, \bar{v}\rangle\rangle$ for every $u \in \Gamma \cup \{ z_0 \}$
Here q'' is a new state, different for every instruction.
For every state $q \in F$ we add the instruction $\langle q, \varepsilon, z_0, \langle q, \varepsilon, \bar{0}\rangle\rangle$. □

Theorem 4 *For every PDAC one can effectively construct an equivalent SBPA.*

Proof. Let $M = \langle \Sigma, Q, \Gamma, q_0, z_0, P, k\rangle$ be a PDAC with k counters. Let $T = \{ t_1, \dots, t_k \}$ be a set of new stack symbols, a and z_0' be new stack symbols, q_0' and q_f be new states. We build a SBPA $S = \langle \Sigma, Q', Z \cup T \cup \{ a, z_0' \}, q_0', \{ q_f \}, \delta\rangle$. Here Q' contains Q, q_0', q_f, and some other states which are described later.
Let the automaton M has an instruction $\langle q, x, z, \langle q', \alpha, (v_1, \dots, v_k)\rangle\rangle$. We divide the vector $\bar{v}$ in two vectors $\bar{v}^+$ and $\bar{v}^-$: $\bar{v} = \bar{v}^+ + \bar{v}^-$, where

$$v_i^+ = \begin{cases} v_i, & \text{if } v_i > 0 \\ 0, & \text{otherwise} \end{cases}, \qquad v_i^- = \begin{cases} v_i, & \text{if } v_i < 0 \\ 0, & \text{otherwise} \end{cases}.$$

The vector $\bar{v}^+$ describes the increase of the counters, and $\bar{v}^-$ describes their decrease. Let $i_1, \dots, i_l$ be the numbers of nonzero coordinate of the vector $\bar{v}^-$, and $j_1, \dots, j_m$ be the numbers of nonzero coordinates of the vector $\bar{v}^+$, $r(s) = |v_s|$, $r = \sum\limits_{i=1}^{k} |v_i^-|$. Let A be a bag containing t_s v_s^+ times for $1 \le s \le k$.

We add the following instructions to δ:
$\delta(q, x, z) = (q_1, az, \emptyset_M)$ (move to q_1 and push into the stack a special symbol a),
$\delta(q_1, \varepsilon, t_{i_1}) = (q_2, \varepsilon, \emptyset_M)$, $\delta(q_2, \varepsilon, t_{i_1}) = (q_3, \varepsilon, \emptyset_M)$, $\dots$,
$\delta(q_{r(i_1)}, \varepsilon, t_{i_1}) = (q_{r(i_1)+1}, \varepsilon, \emptyset_M)$ (decrease i_1-th counter by $r(i_1)$),
$\delta(q_{r(i_1)+1}, \varepsilon, t_{i_2}) = (q_{r(i_1)+2}, \varepsilon, \emptyset_M)$, $\dots$,
$\delta(q_{r(i_1)+r(i_2)}, \varepsilon, t_{i_2}) = (q_{r(i_1)+r(i_2)+1}, \varepsilon, \emptyset_M)$ (decrease i_2-th counter by $r(i_2)$),
$\dots$
$\delta(q_{r(i_1)+\dots+r(i_{l-1})+1}, \varepsilon, t_{i_l}) = (q_{r(i_1)+\dots+r(i_{l-1})+2}, \varepsilon, \emptyset_M)$, $\dots$,
$\delta(q_{r(i_1)+\dots+r(i_l)}, \varepsilon, t_{i_l}) = (q'', \varepsilon, \emptyset_M)$ (decrease i_l-th counter by $r(i_l)$),
$\delta(q', \varepsilon, a) = (q'', \varepsilon, A)$ (increase the remaining counters),
$\delta(q'', \varepsilon, z) = (q', \alpha, \emptyset_M)$ (replace z with α).

These instructions model correctly one instruction of the automaton S, because the symbols from $\Gamma \cup \{ a \}$ never appear in the bag, and the symbols from T never appear in the stack. Besides, we add several additional instructions:
$\delta(q_0', \varepsilon, \varepsilon) = (q_0, z_0 z_0', \emptyset_M)$ (put the symbols $z_0 z_0'$ on the bottom of the stack and prepare for modelling),
$\delta(q, \varepsilon, z_0') = (q_f, \varepsilon, \emptyset_M)$ for every $q \in Q$ (the automaton M emptied the stack; empty the stack, move to q_f). $\square$

In [16] the following class of SPBAs without semantical ε-cycles is considered.

Definition 13 *A SBPA S is called ε-acyclic if it is impossible to apply a transition of the form $\delta(q, \varepsilon, \ldots)$ more than once without reading an element from the input string first.*

So, if S is a ε-acyclic SBPA then it has not the cycles of ε-derivations of the form $\langle q, \varepsilon, \alpha, A\rangle \vdash^+ \langle q, \varepsilon, \beta, B\rangle$.

Let $\mathcal{L}(SBPA)$ be the class of languages accepted by ε-acyclic stack+bag push-down automata. It is not difficult to see that the constructions of the theorems 3 and 4 transform ε-acyclic SBPA into PDA with independent counters without ε-loops, and vice versa. Then from theorem 4 we obtain the following corollary.

Corollary 1 *For every language $L \in \Sigma^*$ the following equivalences hold:*
$L \in \mathcal{L}(SBPA) \Leftrightarrow L \in \mathcal{L}(PDAC) \Leftrightarrow L - \{ \varepsilon \} \in \mathcal{L}(CDG)$.

4. Push-down Automata with Stacks of Independent Counters

In this section we introduce Push-down Automata with Stacks of Independent Counters (PDASCs) without empty loops and prove that they accept exactly the class $\mathcal{L}(mmCDG)$ of languages generated by mmCDGs. PDASCs extend PDACs twofold: each counter is a stack of integers and there is a restriction function which allows to diminish a head of a counter only if the heads of all dependent counters are zeros.

Definition 14 *A Push-down Automaton with Stacks of independent Counters (PDASC) is an octuple $M = \langle W, Q, q_0, Z, z_0, n, \pi, P\rangle$, where*

- *W is a finite set of input (terminal) symbols,*
- *Q is a finite set of states, $q_0 \in Q$ is the start state,*
- *Z is a finite set of stack symbols, and $z_0 \in Z$ is the initial stack symbol,*
- *$n \in \mathbf{N}$ is the number of counter stacks,*
- *$\pi \colon \{ 1, \ldots, n \} \to 2^{\{ 1,\ldots,n \}}$ is a restriction function, it should be symmetrical, i.e. if $y \in \pi(x)$, then $x \in \pi(y)$ for all x and y,*
- *P is a set of instructions of the form $\langle q, a, z, \langle q', \alpha, (i, j)\rangle\rangle$, where $q, q' \in Q$, $a \in W \cup \{ \varepsilon \}$, $z \in Z$, $\alpha \in Z^*$, i is a natural number from 1 to n (a current counter stack), $j \in \{ -1, 0, 1 \}$ defines an execution mode.*

If $y \notin \pi(x)$, then we call the counters x and y independent.

The configuration of the PDASC M is the quadruple $\langle q, w, \gamma, v\rangle$, where $q \in Q$ is the current state, $w \in W^*$ is the part of input which is yet to be recognized, $\gamma \in Z$ is the top of the stack, v is a vector of length n whose components are stacks of natural numbers, i.e. $v \in (\mathbf{N}^+)^n$.

Definition 15 *Let $M = \langle W, Q, q_0, Z, z_0, n, \pi, P\rangle$ be a PDASC. The transition relation $\vdash$ on the set of all configurations of M is defined as follows :*
$\langle q, w, \gamma, v\rangle \vdash \langle q', w', \gamma', v'\rangle$ iff there exists an instruction $\langle q, a, z, \langle q', \alpha, (i, j)\rangle\rangle \in P$ such that $w = aw'$ ($a \in W \cup \{\varepsilon\}$), $\gamma = z\beta$, $\gamma' = \alpha\beta$ for some β, and v' is defined in the following way. Let $v = (\sigma_1, \dots, \sigma_n)$, $v' = (\sigma'_1, \dots, \sigma'_n)$.
1) If $j = 0$, the $v = v'$.
2) If $j = 1$, then σ'_i is obtained from σ_i by increasing the top element of σ_i by one, and for $k \neq i$ $\sigma'_k = \sigma_k$ if the counters i and k are independent, otherwise σ'_k is obtained form σ_k by pushing zero into σ_k.
3) If $j = -1$, then the top elements of all counters dependent on the i-th counter must be equal to zero, and the top element of the i-th counter must be positive. If at least one of these conditions does not hold, then the instruction cannot be applied. If both conditions hold, then σ'_i is obtained from σ_i by subtracting one from the top element of σ_i, for every $k \neq i$ such that the counters i and k are independent $\sigma'_k = \sigma_k$, and for every $k \neq i$ such that the counters i and k are dependent σ'_k is obtained from σ_k by popping its top zero element.

Let $\vdash^*$ be the reflexive transitive closure of $\vdash$.

Note that empty transitions, i.e. transitions that do not read an input symbol, are allowed ($a \in W \cup \{\varepsilon\}$). However, we forbid empty loops.

Let $M = \langle W, Q, q_0, Z, z_0, n, \pi, P\rangle$ be a PDASC. We call it a PDASC *without empty loops* if there are no states $q_1, \dots, q_l \in Q$ such that P contains the instructions $\langle q_i, \varepsilon, z_i, \langle q_{i+1}, \alpha_i, (j_i, k_i)\rangle\rangle$ for $i < l$ and an instruction $\langle q_l, \varepsilon, z_l, \langle q_1, \alpha_l, (j_l, k_l)\rangle\rangle$. In what follows we consider only PDASC without empty loops.

We call the vector of the form $(\mathbf{N}^+)^n$, whose components are stacks of natural numbers, *a configuration of counter stacks*. Let $v_0 = (0; 0; \dots; 0)$ denote the configuration of the counter stacks whose components are n stacks containing one zero each.

Definition 16 *The PDASC M accepts the word w iff $\langle q_0, w, z_0, v_0\rangle \vdash^* \langle q', \varepsilon, \varepsilon, v_0\rangle$ for some state $q' \in Q$.*
Let $L(M)$ be the set of all words accepted by the PDASC M.

The following example shows how the restriction function of PDASC helps accept the iteration of non cf-languages.

Example 3 *Let us consider the language $L_1 = \{\, a^n b^n c^n \mid n > 0 \,\}^+$ from example 1. It is accepted by the following PDASC $M_1 = \langle\{\, a, b, c \,\}, \{\, q_0, q_1, q_2, q_3, q_4 \,\}, q_0, \{\, z_0, a \,\}, z_0, 2, \pi, P\rangle$. Program P consists of the following instructions:*

$\langle q_0, \varepsilon, z_0, \langle q_1, z_0, (1, 1)\rangle\rangle$	$\langle q_2, c, z_0, \langle q_3, z_0, (2, -1)\rangle\rangle$
$\langle q_1, a, z_0, \langle q_1, az_0, (1, 0)\rangle\rangle$	$\langle q_3, c, z_0, \langle q_3, z_0, (2, -1)\rangle\rangle$
$\langle q_1, a, a, \langle q_1, aa, (1, 0)\rangle\rangle$	$\langle q_3, \varepsilon, z_0, \langle q_4, \varepsilon, (1, -1)\rangle\rangle$
$\langle q_1, b, a, \langle q_2, \varepsilon, (2, 1)\rangle\rangle$	$\langle q_3, \varepsilon, z_0, \langle q_0, z_0, (1, -1)\rangle\rangle$
$\langle q_2, b, a, \langle q_2, \varepsilon, (2, 1)\rangle\rangle$	

The restriction function: $\pi(1) = \{ 2 \}$, $\pi(2) = \{ 1 \}$

The following execution shows how M_1 accepts string $w = aabbccabc$. Note that all stacks are increased from right to left and two stacks of counters c_1, c_2 are shown as $(c_1; c_2)$.
$\langle q_0, w, z_0, (0;0)\rangle \vdash \langle q_1, w, z_0, (1;0,0)\rangle \vdash \langle q_1, abbccabc, az_0, (1;0,0)\rangle \vdash$
$\langle q_1, bbccabc,\ aaz_0, (1;00)\rangle \vdash \langle q_2, bccabc, az_0, (0,1;1,0)\rangle \vdash \langle q_2, ccabc, z_0, (0,0,1;2,0)\rangle \vdash$
$\langle q_3, cabc, z_0, (0,1;1,0)\rangle \vdash \langle q_3, abc, z_0, (1;0,0)\rangle \vdash \langle q_0, abc, z_0, (0;0)\rangle \vdash$
$\langle q_1, abc, z_0, (1;0,0)\rangle \vdash \langle q_1, bc, az_0, (1;0,0)\rangle \vdash \langle q_2, c, z_0, (0,1;1,0)\rangle \vdash$
$\langle q_3, \varepsilon, z_0, (1;0,0)\rangle \vdash \langle q_4, \varepsilon, \varepsilon, (0;0)\rangle$

5. PDASCs and mmCDGs

We are now ready to establish the relationships between push-down automata with stacks of independent counters and mmCDG languages.

It is not hard to see that for every mmCDG G one can efficiently construct an equivalent mmCDG G' whose categories do not contain polarized valencies with the left polarities $\swarrow$ and $\nwarrow$. Therefore we assume that all the grammars which we consider below do not have such polarities.

We associate with a potential θ some counter stacks configuration $c(\theta)$ as follows.

Definition 17 *Let $G = \langle W, \mathbf{C}, S, \lambda, \pi\rangle$ be a mmCDG, and $\mathbf{C} = \{ A_1, \ldots, A_r \})$. We define an auxiliary PDASC $M(\mathbf{C}, \pi) = \langle \mathbf{C}, \{ q \}, q, \{ z_0 \}, z_0, r, \pi', P\rangle$ as follows.*
1) $j \in \pi'(i)$ iff $A_j \in \pi(A_i)$
2) For every i we include in P an instruction $\langle q, \nearrow A_i, z_0, \langle q, z_0, (i, 1)\rangle\rangle$ and an instruction $\langle q, \searrow A_i, z_0, \langle q, z_0, (i, -1)\rangle\rangle$.
For a potential θ let $c(\theta)$ be a counter configuration such that $\langle q, \theta, z_0, v_0\rangle \vdash^* \langle q, \varepsilon, z_0, c(\theta)\rangle$.

In fact, $c(\theta)$ is the counter configuration which is obtained from θ if we treat every valency $\nearrow A_i$ as a command to increase the i-th counter, and every valency $\searrow A_i$ as a command to decrease the i-th counter.

Lemma 1 *i) $c(\theta)$ is defined iff there exists a potential θ' such that $\theta\theta'$ is balanced.*
ii) The potential θ is balanced iff $c(\theta) = v_0$.

This lemma is proved by the straight induction on the length of θ.

The following definition proposes a transformation of mmCDGs into cf-grammars with the similar derivations.

Definition 18 *Let $G = \langle W, \mathbf{C}, S, \lambda, \pi\rangle$ be a mmCDG. We denote the cf-grammar $G' = \langle \Sigma, N, S, R\rangle$ as $CF(G)$, where:*
$\Sigma = \{ w^\theta \mid w \mapsto [\alpha]^\theta \in \delta$ for some $\alpha \}$;
N is the set of all local subcategories from δ ;
R is defined in the following way:

$[\alpha] \to w^\theta \in R \Leftrightarrow w \mapsto [\alpha]^\theta \in \delta$	$[A^*\backslash\alpha] \to [A][A^*\backslash\alpha] \in R \Leftrightarrow [A^*\backslash\alpha] \in N$
$[\alpha] \to [A][A\backslash\alpha] \in R \Leftrightarrow [A\backslash\alpha] \in N$	$[\alpha] \to [\alpha/A^*] \in R \Leftrightarrow [\alpha/A^*] \in N$
$[\alpha] \to [\alpha/A][A] \in R \Leftrightarrow [\alpha/A] \in N$	$[\alpha/A^*] \to [\alpha/A^*][A] \in R \Leftrightarrow [\alpha/A^*] \in N$
$[\alpha] \to [A^*\backslash\alpha] \in R \Leftrightarrow [A^*\backslash\alpha] \in N$	

If the categories in G have no potentials, this construction simply transforms a classical categorial grammar into an equivalent cf-grammar ([1]).

The following assertion relates the derivations of G and $CF(G)$.

Lemma 2 *Let G be a mmCDG, $G' = CF(G)$, $\alpha \in N$. Then $\alpha \Rightarrow^*_{G'} a_1^{\theta_1} \dots a_n^{\theta_n} \in \Sigma^*$ iff there exist categories $\gamma_1 = \alpha_1^{\theta_1} \in \delta(a_1), \dots, \gamma_n = \alpha_n^{\theta_n} \in \delta(a_n)$ such that $\gamma_1 \dots \gamma_n \vdash^*_G \alpha^{\theta_1 \dots \theta_n}$.*

Corollary 2 *Let G be a mmCDG, $G' = CF(G)$. Then $w_1 \dots w_n \in L(G)$ iff $w_1^{\theta_1} \dots w_n^{\theta_n} \in L(G')$ for some $\theta_1 \dots, \theta_n$ and the potential $\theta_1 \dots \theta_n$ is balanced.*

Both the lemma and the corollary are proved exactly as Lemma 1 and Corollary 1 in [15].

Theorem 5 *For every mmCDG G one can effectively construct a PDASC without empty loops M such that $L(G) = L(M)$.*

Proof. Let $G = \langle W, \mathbf{C}, S, \lambda, \pi \rangle$ be a mmCDG, $\mathbf{C} = \{ A_1, \dots, A_r \}$. We transform it into an auxiliary cf-grammar $G' = CF(G) = \langle \Sigma, N, S, R \rangle$ (Definition 18). By Corollary 2 $a_1 \dots a_n \in L(G)$ iff $S \Rightarrow^*_{G'} a_1^{\theta_1} \dots a_n^{\theta_n}$ and the potential $\theta_1 \dots \theta_n$ is balanced with respect to π. Let $G'' = \langle \Sigma, N', S, R' \rangle$ be a cf-grammar in Greibach normal form [9] and equivalent to G'. Now we construct a PDASC $M = \langle W, Q, q, \mathbf{C}, S, r, \pi', P \rangle$. The function π' is defined as follows: $j \in \pi'(i)$ iff $A_j \in \pi(A_i)$.
Let $\rho \colon A \to a^\theta B_1 \dots B_t \in R'$, where $\theta = p_1 A_{i_1} \dots p_l A_{i_l}$, $p_i \in \{ \nearrow, \searrow \}$ are polarities. Then we include into Q new states $q_1^\rho, \dots, q_l^\rho$, and we include into P the following set of instructions.
$\langle q, \varepsilon, A, \langle q_1^\rho, A, (i_1, j_1) \rangle \rangle$,
$\langle q_1^\rho, \varepsilon, A, \langle q_2^\rho, A, (i_2, j_2) \rangle \rangle$,
$\dots$
$\langle q_{l-1}^\rho, \varepsilon, A, \langle q_l^\rho, A, (i_l, j_l) \rangle \rangle$,
$\langle q_l^\rho, a, A, \langle q, B_1 \dots B_t, (i, 0) \rangle \rangle$.
Here $j_k = 1$ if $p_k = \nearrow$, and $j_k = -1$ if $p_k = \searrow$.

The following lemma relates derivations of G'' with computations of M.

Lemma 3 *i) If $S \Rightarrow^*_{G''} a_1^{\theta_1} \dots a_n^{\theta_n} Z_1 \dots Z_m$ and $\theta = \theta_1 \dots \theta_n$ is prefix of some balanced potential, then $\langle q, a_1 \dots a_n w, S, v_0 \rangle \vdash^*_M \langle q, w, Z_1 \dots Z_m, c(\theta) \rangle$.*
ii) Let $a_i^{\theta_i} \in \Sigma$ for $1 \le i \le n$ and $\theta = \theta_1 \dots \theta_n$. If $\langle q, a_1 \dots a_n w, S, v_0 \rangle \vdash^*_M \langle q, w, Z_1 \dots Z_m, c(\theta) \rangle$, then θ is a prefix of some balanced potential and $S \Rightarrow^*_{G''} a_1^{\theta_1} \dots a_n^{\theta_n} Z_1 \dots Z_m$.

Proof. i) Induction on the length j of the derivation in G''.
Base case. $j = 0$. By definition $\langle q, w, S, v_0 \rangle \vdash^0_M \langle q, w, S, v_0 \rangle$.
Inductive step. Let $S \Rightarrow^j_{G''} a_1^{\theta_1} \dots a_j^{\theta_j} Z_1 \dots Z_s$, then by the inductive hypothesis there is a derivation $\langle q, a_1 \dots a_j a_{j+1} w, S, v_0 \rangle \vdash^*_M \langle q, a_{j+1} w, Z_1 \dots Z_s, c(\theta) \rangle$, where $\theta = \theta_1 \dots \theta_j$. After this, the rule $\rho \colon Z_1 \to a_{j+1}^{\theta_{j+1}} B_1 \dots B_t$ was used. First of all, the automaton performs the instructions $\langle q, \varepsilon, Z_1, \langle q_1^\rho, Z_1, (i_1, j_1) \rangle \rangle$, $\dots$, $\langle q_{l-1}^\rho, \varepsilon, Z_1, \langle q_l^\rho, Z_1, (i_l, j_l) \rangle \rangle$. They do not change the stack, and they add the potential θ_{j+1} to the stacks of counters. Then the instruction $\langle q_l^\rho, a, Z_1, \langle q, B_1 \dots B_t, (i, 0) \rangle \rangle$ is applied. The resulting configuration is $\langle q, w, B_1 \dots B_t Z_2 \dots Z_s, c(\theta\theta_{j+1}) \rangle$.

ii) Induction on the number j of steps of the automaton M.
Base case. $j = 0$. There are no steps. By definition $S \Rightarrow^0_{G''} S$.
Inductive step. Let $\langle q, a_1 \dots a_j a_{j+1} w, S, v_0\rangle \vdash^*_M \langle q, a_{j+1}w, Z_1 \dots Z_s,\ c(\theta)\rangle$, then by the inductive hypothesis $S \Rightarrow^*_{G''} a_1^{\theta_1} \dots a_j^{\theta_j} Z_1 \dots Z_s$ and $\theta_1 \dots \theta_j$ is a prefix of a balanced potential. First, let us assume that after this the automaton uses an ε-instruction $\langle q, \varepsilon, Z_1, \langle q_1^\rho, Z_1, (i_1, j_1)\rangle\rangle$. Then the sequence of ε-instructions that follow the first one is unambiguously defined, since the new states are different for every rule of the grammar G''. When the automaton reaches a state q_l^ρ, it uses an instruction of the form $\langle q_l^\rho, a, Z_1, \langle q, B_1 \dots B_t, (i, 0)\rangle\rangle$. This sequence of instructions was obtained from the rule $\rho\colon Z_1 \to a_{j+1}^{\theta_{j+1}} B_1 \dots B_t$. By construction the change of the stacks of counters corresponds to the potential θ_{j+1}. Thus, we get the configuration $\langle q, w, B_1 \dots B_t Z_2 \dots Z_s, c(\theta\theta_{j+1})\rangle$. Since the automaton was able to perform the instructions, the potential $\theta_1 \dots \theta_{j+1}$ is a prefix of some balanced potential. Also $S \Rightarrow^*_{G''} a_1^{\theta_1} \dots a_{j+1}^{\theta_{j+1}} B_1 \dots B_t Z_2 \dots Z_s$. □

It follows from Corollary 1 and this lemma that the following five statements are equivalent.
1) $a_1 \dots a_n \in L(G)$
2) $S \Rightarrow^*_{G'} a_1^{\theta_1} \dots a_n^{\theta^n}$ and the potential $\theta_1 \dots \theta_n$ is balanced.
3) $S \Rightarrow^*_{G''} a_1^{\theta_1} \dots a_n^{\theta^n}$ and the potential $\theta_1 \dots \theta_n$ is balanced.
4) $\langle q, a_1 \dots a_n, S, v_0\rangle \vdash^*_M \langle q, \varepsilon, \varepsilon, v_0\rangle$
5) $a_1 \dots a_n \in L(M)$
Therefore, $L(M) = L(G)$. □

Now we prove the converse.

The following definition presents a transformation of cf-grammars into mmCDGs with the similar derivations.

Definition 19 *Let $G' = \langle \Sigma, N, S, R\rangle$ be a cf-grammar in Greibach normal form, where the elements of Σ are of the form w^θ. We denote by $mmCDG(G', \pi)$ the mmCDG $G = \langle W, N, S, \lambda, \pi\rangle$, where $W = \{ w \mid w^\theta \in \Sigma$ for some $\theta \}$ and δ is defined in the following way:*
$w \mapsto [X]^\theta \in \delta \Leftrightarrow X \to w^\theta \in R$,
$w \mapsto [X/Y]^\theta \in \delta \Leftrightarrow X \to w^\theta Y \in R$,
$w \mapsto [X/Z/Y]^\theta \in \delta \Leftrightarrow X \to w^\theta Y Z \in R$.

If G has no potentials, then $mmCDG(G)$ is simply a categorial grammar equivalent to G [1].

The following assertion relates the derivations of G' and $CF(G)$.

Lemma 4 *Let $G = mmCDG(G', \pi)$. Then $X \Rightarrow^*_{G'} a_1^{\theta_1} \dots a_n^{\theta_n} \in \Sigma^*$ iff there exist categories $\gamma_1 = \alpha_1^{\theta_1} \in \delta(a_1), \dots\ \gamma_n = \alpha_n^{\theta_n} \in \delta(a_n)$ such that $\gamma_1 \dots \gamma_n \vdash^*_G [X]^{\theta_1 \dots \theta_n}$.*

Corollary 3 *Let G' be a cf-grammar in Greibach normal form, π be a cross prohibition function, $G = mmCDG(G', \pi)$. Then $w_1 \dots w_n \in L(G)$ iff $w_1^{\theta_1} \dots w_n^{\theta_n} \in L(G')$ for some $\theta_1, \dots, \theta_n$ and the potential $\theta_1 \dots \theta_n$ is balanced with respect to π.*

These two assertions are proved along the lines of Lemma 2 and Corollary 2 in [15].

Theorem 6 *For every PDASC without empty loops M one can effectively construct a mmCDG G such that $L(G) = L(M) \setminus \{ \varepsilon \}$.*

Proof. Let $M = \langle W, Q, q_0, Z, z_0, k, \pi, P\rangle$ be a PDASC without empty loops. The transformation of the automaton into a grammar is performed in several stages.

1) We construct an auxiliary graph $G = (V, E)$, where V is a set of all ε-instructions of M, and the edge connects two instructions if and only if these two instructions can be performed one after another, i.e. E contains all edges of the form $(\langle q, \varepsilon, z, \langle q', \alpha, (i, j)\rangle\rangle, \langle q', \varepsilon, z', \langle q'', \alpha', (i', j')\rangle\rangle)$. Since M has no empty loops, the graph G is acyclic. Hence, the set of all paths in this graph is finite. Now for every ε-path r we define corresponding potential $\theta(r)$. The i-th valency of $\theta(r)$ is $\nearrow A_j$ if the i-th instruction of r has form of $(j, 1)$, it is $\searrow A_j$ if the i-th instruction is of the form $(j, -1)$, and ε, otherwise. We call the ε-path r complete if no ε-instructions can be applied immediately before or after it. More precisely, this means that if the first instruction from r is of the form $\langle p, \varepsilon, z_1, \langle p_1, \alpha_1, (i_1, j_1)\rangle$ and the last one is of the form $\langle q_2, \varepsilon, z_2, \langle q, \alpha_2, (i_2, j_2)\rangle\rangle$, then there are no instructions of the form $\langle p', \varepsilon, z, \langle p, \alpha, (i, j)\rangle\rangle$ or $\langle q, \varepsilon, z, \langle q', \alpha, (i, j)\rangle\rangle$. Now we rename the states of M in such a way that no two complete ε-paths from G had common states. In order to do this, we remove all ε-instructions from P, and for every instruction $\langle p, \varepsilon, z, \langle q, \alpha, (i, j)\rangle\rangle$ in the l-th complete path we add a new instruction $\langle p^l, \varepsilon, z, \langle q^l, \alpha, (i, j)\rangle\rangle$, where the states p^l, q^l are unique for all paths. If the state q^l is now the first state of l-th ε-path, then for every instruction $\langle p, a, z, \langle q, \alpha, (i, j)\rangle\rangle$ $(a \neq \varepsilon)$ we add an instruction $\langle p, a, z, \langle q^l, \alpha, (i, j)\rangle\rangle$. If the state p^l is the last state of l-th ε-path, then for every instruction $\langle p, a, z, \langle q, \alpha, (i, j)\rangle\rangle$ $(a \neq \varepsilon)$ we add an instruction $\langle p^l, a, z, \langle q, \alpha, (i, j)\rangle\rangle$. As the result, we get the new automaton $M_1 = \langle W, Q_1, q_0, Z, z_0, k, \pi, P_1\rangle$. It is easy to see that M_1 is equivalent to M.

2) Now we duplicate all the states of M_1. Let $\overline{Q}_1 = \{\, \overline{q} \mid q \in Q_1 \,\}$ be a set of copies, $Q_2 = Q_1 \cup \overline{Q}_1$. For every $a \in W \cup \{\, \varepsilon \,\}$ and for every instruction $\langle p, a, z, \langle q, \alpha, (i, j)\rangle\rangle \in P_1$ we include into P_2 this instruction and its copy, obtained from original by replacing states by their copies, i.e. the instruction $\langle \overline{p}, a, z, \langle \overline{q}, \alpha, (i, j)\rangle\rangle$. If $a \in W$, then we also include an instruction $\langle p, a, z, \langle \overline{q}, \alpha, (i, j)\rangle\rangle$. Let $M_2 = \langle W, Q_2, q_0, Z, z_0, k, \pi, P_2\rangle$. On every step M_2 "knows" whether it has already read the first symbol of the word or not. It is easy to see that $L(M_2) = L(M_1)$.

3) Let $1 \le i \le k$, $j \in \{\, -1, 0, 1 \,\}$. Then $\theta(i, j)$ denotes $\nearrow A_i$ if $j = 1$, it denotes $\searrow A_i$ if $j = -1$, and it denotes ε otherwise. Now we can construct an auxiliary cf-grammar $G_1 = \langle \Delta, N, S, R\rangle$.
Δ includes all the symbols of the form a^θ appearing below in the rules of R.
$N = \{\, [qzq'] \mid q, q' \in Q_2, z \in Z \,\} \cup \{\, S \,\}$. R contains the following rules.
a) Let $\langle \overline{p}, \varepsilon, z, \langle \overline{q}, z_1 \dots z_t, (i, j)\rangle\rangle$ be an instruction, $\overline{p}, \overline{q} \in \overline{Q}_1$. Then we include into R the rules $[\overline{p}zq_t] \to [\overline{q}z_1q_1][q_1z_2q_2]\dots[q_{t-1}z_tq_t]$ for all $q_1, \dots, q_t \in Q_2$.
Let $\langle p, \varepsilon, z, \langle q, z_1 \dots z_t, (i, j)\rangle\rangle$ be an instruction, $p, q \in Q_1$. Then we include into R the rules $[pzq_t] \to [qz_1q_1][q_1z_2q_2]\dots[q_{t-1}z_tq_t]$ for all $q_1, \dots, q_t \in Q_2$.
b) Let $\langle \overline{p}, a, z, \langle \overline{q}, z_1 \dots z_t, (i, j)\rangle\rangle$ be an instruction, $\overline{p}, \overline{q} \in \overline{Q}_1$, $a \in W$. Let r be a ε-path whose first instruction has $\overline{q}$ as its first state. Then we include into R the rules
$[\overline{p}zq_t] \to a^{\theta(i,j)\theta(r)}[\overline{q}z_1q_1][q_1z_2q_2]\dots[q_{t-1}z_tq_t]$ for all $q_1, \dots, q_t \in Q_2$.
c) Let $\langle p, a, z, \langle \overline{q}, z_1 \dots z_t, (i, j)\rangle\rangle$ be an instruction, $p \in Q_1$, $\overline{q} \in \overline{Q}_1$, $a \in W$. Let r be a ε-path ending with the state p, and let r' be a ε-path beginning with $\overline{q}$. Then we include into R the rules $[pzq_t] \to a^{\theta(r)\theta(i,j)\theta(r')}[\overline{q}z_1q_1][q_1z_2q_2]\dots[q_{t-1}z_tq_t]$ for all $q_1, \dots, q_t \in Q_2$.
d) We include into R the rules $S \to [q_0z_0q]$ for every $q \in Q_2$.

Lemma 5 *i) Let* $\langle q_0, a_1 \dots a_n, z_0, v_0\rangle \vdash^*_{M_2} \langle q, \varepsilon, z_1 \dots z_t, v\rangle$ *and no* ε*-step can be made*

from q. *Then* $[q_0z_0q_t] \Rightarrow^*_{G_1} a_1^{\theta_1}\dots a_n^{\theta_n}[qz_1q_1][q_1z_2q_2]\dots[q_{t-1}z_tq_t]$ *for some* $\theta_1,\dots,\theta_n$, $q_1,\dots,$ q_t, *and* $c(\theta_1\dots\theta_n) = v$.
ii) Let $[q_0z_0q_t] \Rightarrow^*_{G_1} a_1^{\theta_1}\dots a_n^{\theta_n}[qz_1q_1]\dots[q_{t-1}z_tq_t]$ *be the left derivation in* G_1. *Suppose that* $c(\theta_1\dots\theta_n)$ *is defined and there are not* ε*-instructions starting with* q*, i.e. instructions of the form* $\langle q,\varepsilon,z,\langle\dots\rangle\rangle$. *Then* $\langle q_0,a_1\dots a_n,z_0,v_0\rangle \vdash^*_{M_2} \langle q,\varepsilon,z_1\dots z_t,c(\theta_1\dots\theta_n)\rangle$.

Proof. First of all let us notice that without the potentials these two assertions are the same as for standard transformation of a push-down automaton into an equivalent cf-grammar. Therefore it is enough to prove only the part of the assertions considering the counters and the potentials.
i) Induction on n.
Base case. Let $n = 1$. Then the automaton performs ε-instructions of some complete ε-path r beginning with q_0 and ending with q_1, then it performs an instruction $\langle q_1,a_1,z,\langle q_2,\alpha,(i,j)\rangle\rangle$, and finally it performs ε-instructions of some complete ε-path r' beginning with q_2 and ending with q. The rules of G_1 corresponding to ε-instructions have empty potentials, and the rule for a_1 has by construction a potential $\theta = \theta(r)\theta(i,j)\theta(r')$. We see that $c(\theta) = v$.
Inductive step. Suppose that there is a derivation of M: $\langle q_0,a_1\dots a_na_{n+1},z_0,v_0\rangle \vdash^*_M$ $\langle q_1,a_{n+1},\beta,v_1\rangle \vdash^*_M \langle q,\varepsilon,z_1\dots z_t,v\rangle$. After reaching state q_1 the automaton performs an instruction $\langle q_1,a_{n+1},z_1,\langle q_2,\alpha,(i,j)\rangle\rangle$, and then it performs the ε-instructions of some complete ε-path r beginning with q_2 and ending with q. By construction the rule corresponding to the first instruction has the symbol a_{n+1} with the potential $\theta(i,j)\theta(r)$, and the rules corresponding to ε-instructions have no potentials. Then the total potential is $\theta = \theta_1\dots\theta_n\theta(i,j)\theta(r)$, and $c(\theta) = c(\theta_1\dots\theta_n\theta(i,j)\theta(r)) = v$ because $c(\theta_1\dots\theta_n) = v_1$ by induction hypothesis.
ii) Induction on n.
Base case. If $n = 1$, then the derivation begins with rules without terminal symbols corresponding to some complete ε-path r, then it contains a rule of the form $[pzq_t] \to a_1^{\theta_1}[qz_1q_1][q_1z_2q_2]\dots[q_{t-1}z_tq_t]$, and then it ends with the rules corresponding to some complete ε-path r'. Let $\langle p,a_1,z_1,\langle q,\alpha,(i,j)\rangle\rangle$ be an instruction. Then by construction $\theta = \theta(r)\theta(i,j)\theta(r')$. Therefore, $c(\theta) = v$.
Inductive step. Let $[pzq_t] \Rightarrow^*_{G_1} a_1^{\theta_1}\dots a_n^{\theta_n}\xi \Rightarrow^*_{G_1} a_1^{\theta_1}\dots a_n^{\theta_n}a_{n+1}^{\theta_{n+1}}\xi'$, where $\xi,\xi' \in N^+$. After obtaining $a_1^{\theta_1}\dots a_n^{\theta_n}$ some rule of the form $A \to a_{n+1}^{\theta_{n+1}}\eta$ is applied, where $\eta \in N^*$, and then the derivation uses rules without terminal symbols corresponding to some complete ε-path r. The first rule corresponds to some instruction $\langle q',a_{n+1},z_1,\langle q'',\alpha,(i,j)\rangle\rangle$. Then by construction $\theta_{n+1} = \theta(i,j)\theta(r)$. Having read the symbols $a_1,\dots,a_n$, the automaton gets the counter stacks configuration $v_1 = c(\theta_1\dots\theta_n)$ (by induction hypothesis). After reading a_{n+1} the automaton gets the counter stacks configuration $c(\theta_1\dots\theta_{n+1})$. □

Lemma 6 *i) Let* $\langle q_0,a_1\dots a_n,z_0,v_0\rangle \vdash^*_{M_2} \langle q,\varepsilon,\varepsilon,v\rangle$ *and no* ε*-step can be made from state* q. *Then* $[q_0z_0q] \Rightarrow^*_{G_1} a_1^{\theta_1}\dots a_n^{\theta_n}$ *for some* $\theta_1,\dots,\theta_n$, $q_1,\dots q_t$, *and* $c(\theta_1\dots\theta_n) = v$.
ii) Let $[q_0z_0q] \Rightarrow^*_{G_1} a_1^{\theta_1}\dots a_n^{\theta_n}$ *be the left derivation in* G_1. *Suppose that* $c(\theta_1\dots\theta_n)$ *is defined and there are not* ε*-instructions starting with* q*, i.e. instructions of the form* $\langle q,\varepsilon,z,\langle\dots\rangle\rangle$. *Then* $\langle q_0,a_1\dots a_n,z_0,v_0\rangle \vdash^*_{M_2} \langle q,\varepsilon,\varepsilon,c(\theta_1\dots\theta_n)\rangle$.

Like in previous lemma, the assertion without stacks is proved in standard way. In order to get the statements concerning potentials and counters, one should use the proof similar to the one in the previous lemma, but consider the final step of derivation.

Now, let G_2 be a cf-grammar in Greibach normal form and equivalent to G_1. Let $G = mmCDG(G_2, \pi')$ (Definition 19). Then it follows from corollary 3 and lemmas 1 and 6 that the following statements are equivalent.
1) $a_1 \ldots a_n \in L(M)$
2) $\langle q_0, a_1 \ldots a_n, z_0, v_0\rangle \vdash_M^* \langle q, \varepsilon, \varepsilon, v_0\rangle$ for some $q \in Q_2$
3) $S \Rightarrow_{G_1} [q_0 z_0 q] \Rightarrow_{G_1}^* a_1^{\theta_1} \ldots a_n^{\theta_n}$ for some $\theta_1, \ldots, \theta_n$ such that $c(\theta_1 \ldots \theta_n) = v_0$
4) $S \Rightarrow_{G_2}^* a_1^{\theta_1} \ldots a_n^{\theta_n}$ for some $\theta_1, \ldots, \theta_n$ such that $c(\theta_1 \ldots \theta_n) = v_0$
5) $a_1 \ldots a_n \in L(G)$

Therefore, $L(G) = L(M)$. □

Theorems 5 and 6 lead to the following

Corollary 4 *1) $\mathcal{L}(mmCDG) \subseteq \mathcal{L}(PDASC)$.*
2) If $L \in \mathcal{L}(PDASC)$, then $L - \{\varepsilon\} \in \mathcal{L}(mmCDG)$.

Now all properties of $\mathcal{L}(mmCDG)$ established in [4] hold also for $\mathcal{L}(PDASC)$. Especially, $\mathcal{L}(PDASC)$ is closed under union, concatenation, iteration, intersection with regular languages, ε-free homomorphisms, and inverses of homomorphisms, i.e. all AFL operations [8], it includes some non-semilinear languages, there is NP-complete language $G \in \mathcal{L}(PDASC)$.

References

[1] Bar-Hillel Y., Gaifman H., Shamir E., "On categorial and phrase structure grammars", *Bull. Res. Council Israel*, **9F** (1960), 1–16.

[2] Dekhtyar M., Dikovsky A., "Categorial dependency grammars", Proc. of Int. Conf. on Categorial Grammars, 2004, 76–91.

[3] Dekhtyar M., Dikovsky A., "Generalized categorial dependency grammars", *Pillars of Compute Science: Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, LNCS, **4800**, 2008, 230–255.

[4] Dekhtyar M., Dikovsky A., Karlov B., "Iterated dependencies and Kleene iteration", Proc. of the 15th Conference on Formal Grammar (FG 2010), LNCS, **7395**, Copenhagen, Denmark, 2012, 66–81.

[5] Dikovsky A., "Grammars for local and long dependencies", In Proc. of the Intern. Conf. ACL'2001, 2001, 156–163.

[6] Dikovsky A., "Dependencies as categories", Proc. of Workshop Recent Advances in Dependency Grammars". In conjunction with COLING 2004, 2004, 90–97.

[7] Dikovsky A., "Proc. of the 12th Conference on Formal Grammar", 2007, 1–12.

[8] Ginsburg S., Greibach S., "Abstract families of languages", *Mem. Amer. Math. Soc.*, 1969, № 87, 1–32.

[9] Greibach S., "A new normal-form theorem for context-free phrase structure grammars", *Journal of the ACM*, **12** (1965), 42–52.

[10] Hack M., "Petri Net Languages", MIT, Lab. for Computer Science, Technical Report 159, 1976.

[11] Hopcroft J., Ullman J., *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, 1979.

[12] Kallmeyer L., *Parsing Beyond Context-Free Grammars, Cognitive Technologies*, Springer-Verlag, Berlin Heidelberg, 2010.

[13] Joshi A., Levy L., Takahashi M., "Tree adjunct grammar", *Journal of Computer and System Sciences*, 1975, № 10(1), 136–163.

[14] Kanazawa M., Salvati S., "Mix is not a tree-adjoining language", *The 50th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*: *Long Papers*, **1**, Jeju Island, Korea, 2012, 666–674.

[15] Karlov B., "Abstract automata and a normal form for categorial dependency grammars", Proc. of the 7th International Conference on Logical Aspects of Computational Linguistics (LACL 2012), LNCS, **7351**, Nantes, France, 2012, 86–102.

[16] Søgaard A., "A linear time extension of deterministic push-down automata", Proc. of the 17th Nordic Conference of Computational Linguistics NODALIDA 2009, NEALT Proceedings Series, **4**, eds. K. Jokinen, E. Bick, 2009, 182–189.

[17] Tesnière L., *Éléments de syntaxe structurale. Librairie C*, Klincksieck, Paris, 1959.

[18] Vijay-Shanker K., *A Study of Tree Adjoining Grammars*, Ph.D. thesis, University of Pennsylvania, 1987.

[19] Vijay-Shanker K., Weir D. J., "The equivalence of four extensions of context-free grammars", *Mathematical Systems Theory*, 1994, № 27(6), 511–546.

[20] Gladkiy A. V., *Formalnye grammatiki i yazyki*, Nauka, Moskva, 1973, (in Russian).

[21] Karlov B. N., "Normalnye formy i avtomaty dlya kategorialnykh grammatik zavisimostey", *Vestnik Tverskogo gosudarstvennogo universiteta*, "Prikladnaya matematika", 2008, 23–43, (in Russian).

МП-автоматы с независимыми счётчиками

Дехтярь М. И., Карлов Б. Н.

Тверской государственный университет
170000 Россия, г. Тверь, ул. Желябова, 33

Ключевые слова: автоматы, формальные грамматики и языки, магазинные автоматы с независимыми счётчиками, проективная и непроективная зависимость, категориальная грамматика зависимостей, мультимодальная категориальная грамматика зависимостей, магазинные автоматы со стеками независимых счётчиков

Магазинные автоматы с независимыми счётчиками (МПНС) объединяют возможности МП-автоматов и сетей Петри. Они были предложены в работах [21, 15] как средство для распознавания языков, порождаемых категориальными грамматиками зависимостей (КГЗ). КГЗ представляют собой классические категориальные грамматики, расширенные ориентированными поляризованными валентностями. Они позволяют выразить как проективные, так и непроективные зависимости между словами предложения. МПНС — это обычный МП-автомат, к которому добавлено конечное число счётчиков. Независимость счётчиков означает, что их содержимое не влияет на выбор очередного действия автомата. В первой части статьи мы сравниваем несколько вариантов определения МПНС и доказываем эквивалентность двух вариантов МПНС: без синтаксических пустых циклов и без семантических пустых циклов. Отмечаем также некоторые связи между МПНС-языками и языками сетей Петри. Мы показываем, что МПНС эквивалентны стек+бэг МП-автоматам (СБМПА), предложенным независимо Сёгаардом (Søgaard), и что СБМПА без пустых циклов распознают в точности КГЗ-языки. Мультимодальные категориальные грамматики зависимостей (ммКГЗ) были введены в [4] как расширения КГЗ, позволяющие управлять пересечениями некоторых зависимостей. Класс ммКГЗ-языков достаточно богат и обладает многими свойствами замкнутости, в частности, он образует абстрактное семейство языков. Во второй части статьи мы расширяем МПНС и определяем МП-автоматы со стеками независимых счётчиков (МПСНС). Это расширение двоякое: (1) каждый счётчик представляет стек натуральных чисел и (2) добавляется функция, которая позволяет уменьшать число на вершине стека счётчика, только если вершины всех связанных с ним счётчиков равны нулю. Наш основной результат утверждает, что МПСНС допускают в точности класс ммКГЗ-языков.

Сведения об авторах:
Дехтярь Михаил Иосифович,
Тверской государственный университет,
д-р физ.-мат. наук, доцент, код ORCID 0000-0002-2609-4397,
Карлов Борис Николаевич,
Тверской государственный университет,
канд. физ.-мат. наук, код ORCID 0000-0002-4340-2435