Proceedings of the 10th WSEAS International Conference on COMPUTERS, Vouliagmeni, Athens, Greece, July 13-15, 2006 (pp962-967)

Left-most derivation and shadow-pushdown automata for context-sensitive languages

BENEDEK NAGY
Department of Computer Science
University of Debrecen
Egyetem tér 1, Debrecen
HUNGARY

Research Group on Mathematical Linguistics
Rovira I Virgili University
Tarragona
SPAIN

Abstract: - In this paper left-most derivation for context-sensitive grammars is presented based on Penttonen one-sided normal-form. The derivations using grammars in normal form are represented by tree-like graphs. Left-most derivation is defined in the sense of constructing the derivation graph. The concept of the well-known pushdown automata is generalised. A special-type of automata, the so-called shadow-pushdown automata are presented. The work of the automata is based on the left-most derivation of context-sensitive languages. The class of shadow-pushdown automata characterizes exactly the context-sensitive languages.

Key-Words: - formal languages, automata, context-sensitive languages, derivation tree, left-most derivation, normal form, pushdown automata

1 Introduction

The theory of formal languages and automata is one of the most important fields of Theoretical Computer Science. The Chomsky hierarchy of languages, the generative grammars and the corresponding automata-classes are well-known. The regular languages are recognized by (deterministic or nondeterministic) finite automata. The context-free languages can be accepted by (non-deterministic) pushdown automata. The automata-class for context-sensitive languages is the linear bounded Turing machines. Every recursive enumerable language is accepted by a Turing machine. In this correspondence the context-sensitive languages are used as special recursive enumerable languages ([10, 5]). There are several other characterizations of context-sensitive languages known. For instance, they are exactly the frontiers of recognizable picture languages [4]. They coincide to the languages generated by one-dimensional cellular automata [2]. Moreover it is proved that the rational graphs trace context-sensitive languages ([7]) by linear bounded Turing machines. In this paper our approach is closely related to the usual derivation-trees for context-free grammars. We use the context-sensitive class as an extension/generalisation of the context-free case. In the literature the stack automata is known, which is an extension of the pushdown automata ([3]). In this paper, a special extension of the pushdown automata will be shown, which accept exactly the context-sensitive languages. The work of the pushdown automata based on the so-called left-most derivation in context-free grammars. There is exactly one left-most derivation for every derivation tree. Generally the pure concept of derivation tree does not work for context-sensitive grammars, therefore we need a generalised concept of the derivation-trees. The left-most derivations cannot be defined in the usual (sentential form) sense for context-sensitive grammars without loosing the generative power. In this paper based on Penttonen's old result ([9], where he gave one-sided normal form for every context-sensitive grammar) the derivations can be represented in a highly similar way as the derivation trees. In these graphs the so-called context-edges refer for the context-sensitivity. With these derivation graphs left-most derivation can be defined (in derivation graph sense). These left-most derivations give the basis of the work of shadow-pushdown automata.

2 Preliminaries

In this section we recall some basic definitions, notations and facts about context-free and context-sensitive grammars and languages and pushdown automata (based on [3, 1, 6]). The definitions and notion of generative grammars and the generated languages are recalled. A grammar is a construct $G = (N; T; S;H)$, where N, T are the non-terminal and terminal alphabets, with $N \setminus T = \{\}$; they are finite sets. $S \in N$ is a special symbol, called initial letter. H is a finite set of pairs, where a pair uses to

be written in the form $v \to w$ with $v \in \{N \cup T\}^* N \{N \cup T\}^*$ and $w \in \{N \cup T\}^*$. (We used the well-known notation of Kleene-star.) *H* is the finite set of derivation rules. A sequence of letters $v \in \{N \cup T\}^*$ is called a string, while we refer $u \in T^*$ as a (terminal) word. Let *G* be a grammar and *v,w* be two strings. Then $v \Rightarrow w$ is a direct derivation if and only if there exist v_1, v_2, *v'*, *w'* strings such that $v = v_1 v' v_2$, $w = v_1 w' v_2$ and $v' \to w' \in H$. A derivation $v \Rightarrow^* u$ holds if and only if either $v = u$ or there is a finite sequence of strings connect them as $v = v_0, v_1, \ldots v_m = u$ in which $v_i \Rightarrow v_{i+1}$ is a direct derivation for each $0 \le i < m$. The strings can be derived from *S* are the sentential forms. The language generated by a grammar *G* is the set of terminal words can be derived from the initial letter: $L(G) = \{w | S \Rightarrow^* w, w \in T^*\}$. A grammar is context-free if its rules are in the form $A \to v$ with a non-terminal *A* and a string *v*.

For each context-free grammar there is an equivalent grammar (i.e. the generated language can differ at most in the empty word λ) in which all derivation rules are in one of the forms $A \to BC, A \to a \quad A, B, C \in N; a \in T$.

A grammar has only these kinds of rules is in the so-called Chomsky normal form. (From now on we do not care about the fact whether the empty word is in the language L or not.)

A grammar is context-sensitive if all derivation rules are in the form $v_1 A v_2 \to v_1 w v_2$, with strings v_1, v_2, *w*, and non-terminal *A* with $w \neq \lambda$ (except possibly for the rule $S \to \lambda$ in which case *S* does not occur on any right hand side of a rule).

Now we recall the so-called Penttonen normal form for these grammars.

Lemma 1. *Every context-sensitive language can be generated by a grammar whose derivation rules are of the form* $A \to BC, AB \to AC, A \to a$ *, where A, B and C are nonterminals and a is a terminal. This normal form is from ([9]), where it was called one-sided normal form.*

A language is context-free/context-sensitive if it can be generated by a context-free/context-sensitive grammar.

Note, that in this paper we will use pushdown automata only with one state. Moreover we use special form based on normal form, but it is still universal among pushdown automata. So we use one of the simplest descriptions of them.

A pushdown automaton *M* consists of an input tape, a finite control and a stack, formally it is a system $(\Sigma, \Gamma, \delta, Z_0)$, where Σ is a finite alphabet called the input alphabet, Γ is the stack alphabet, Z_0 is a special stack symbol, initially in the bottom of the stack and δ is a mapping form $(\Sigma \cup \{\lambda\}) \times \Gamma$ to finite subsets of Γ^*. The interpretation of δ is the following. The relation $z \in \delta(\lambda, Z)$ with a stack symbol *Z* (where $z \in \Gamma^*$) means that *M* can replace the top of the stack *Z* by *z* without moving its head. The relation $\lambda \in \delta(a, Z)$ with an input symbol *a* and stack symbol *Z* means that *M* can erase the top of the stack *Z* advancing the input head on the symbol *a*. A configuration of a pushdown automata *M* is a pair (*v*, *z*) where $v \in \Sigma^*$ the unprocessed part of the input word and $z \in \Gamma^*$ is the actual contents of the stack. A word *w* is accepted by *M* if there is a finite sequence of configurations starting by (w, Z_0) and finishing by (λ, λ) such that in each step the configuration can be given by δ. The set of all accepted words forms the accepted language of the automaton. Well-known that the (non-deterministic) pushdown automata accept exactly the context-free languages. It is easy to construct a pushdown automaton which accepts exactly the language generated by a grammar in Chomsky normal form. Let $\Sigma = T, \Gamma = N$ and $Z_0 = S$. So, let the automata start with *S* in the stack. Let the transition function be given by the derivation rules in the following way:

$CB \in \delta(\lambda, A)$ if and only if there is a rule $A \to BC$ in the grammar; $\lambda \in \delta(a, A)$ if and only if there is rule $A \to a$.

The automaton accepts a word *w* simulating a left-most derivation in the grammar. (A derivation is called left-most (for context-free grammars) if the first non-terminal occurring in the sentential form is replaced in each step by an appropriate derivation rule.)

In the next section derivation-'tree' and left-most derivation for grammars in Penttonen normal form is shown.

3 Left-most Derivation for Context-sensitive Grammars

The concept of derivation-trees does not work in pure form for context-sensitive grammars and derivations. The neighborhood of a non-terminal can also be important in use of a replacing rule. We use two kinds of edges in these derivation graphs to represent the context-sensitive derivation steps. The original, derivation edges are coming from the replaced non-terminal and going to the new (terminal or nonterminal) symbol(s) given in the right hand side of the used derivation rule. The new type of edges shows the neighborhood of the replaced non-terminals as the used rule requested.We will use the name context-edge. Using grammars in Penttonen normal form the derivation graphs have nice simple structures.

Definition 1. *Let G = (N, T, S, H) be a grammar in Penttonen normal form. The* derivation tree *for a string v is a directed graph build up from labelled nodes (labels from* $N \cup T$ *and two kinds of edges (derivation and context) as follows. The nodes and the derivation edges form a tree graph:*
(1) The root is labelled by S.
(2) Every interior node is labelled by a non-terminal.
(3) The labels of leaves reading from left to right yields v.
Let A be the label of an interior node, and let the string u be its child(ren)'s label(s reading from left to right). Then one of the following conditions is fulfilled.
(i) If there is no context-edge ends at that interior node then $A \to u \in H$.
(ii) If exactly one context edge ends at that interior node (coming from a node labelled by $C \in N$ *) then* $CA \to Cu \in H$.
Every context-edge connects two neighbor branches of the tree (directed from left to right). There are no context-edges crossing each other, i.e. a node x which is an ancestor of a node y must not have an out-context-edge to a node z if there is a context-edge from y to any of the ancestors of z. A derivation tree is said to be finished *if it has only terminal-labelled leaves.*
Observe, that the structure of the graph is simple. For every interior node there is/are one or two child(ren) nodes. In the first case this child can be labelled by a terminal (based on a rule $A \to a (a \in T)$ or can be a non-terminal *C* corresponding to a rule $BA \to BC$ with a non-terminal *B*; while in the latter case these nodes (and their order) must be coincide with a rule in the form $A \to BC (B, C \in N)$. The graph also contains context-edges representing the applied context-sensitive rules. When a node *x* labelled by a non-terminal *A* has only one successor node which is also non-terminal labelled (let its label be *C*), then *x* has exactly one in-context-edge and it is from a node labelled by *B* (satisfying $BA \to BC \in H$). Moreover there are some restrictions for context-edges, namely any of them can cross any other edges (nor derivation, nor context).
Now, for better understanding an example is shown.
Example 1. Let *G* = ({*S, A, B, C, D, E, F, G, I, J, K, L, M, O*}, {*a, b, c*}, *S, H*) be a context sensitive grammar in Penttonen normal form, with rule set *H* = {$S \to AG$, $G \to BC$, $A \to IJ$, $J \to DE$, $EB \to EE$, $EC \to EK$, $K \to FL$, $D \to IM$, $M \to AB$, $BE \to BB$, $BF \to BO$, $O \to CL$, $A \to a$, $B \to b$, $C \to c$, $D \to a$, $E \to b$, $F \to c$, $I \to a$, $;L \to c$}. Fig. 1 shows a possible finished derivation-graph in this system. The derivation edges are represented by solid arrows, while the context-edges are broken.
Theorem 1. *For a grammar G there exists a finished derivation tree for a word w if and only if w in L*(*G*).

Proof. Simulating the derivation by tokens on the graph it can be shown that there is at least one sentential derivation for every derivation tree. In other side there is exactly one derivation tree for each sentential derivation. (For more details we refer to [8]). Therefore finished derivation trees correspond exactly to words of the generated language.

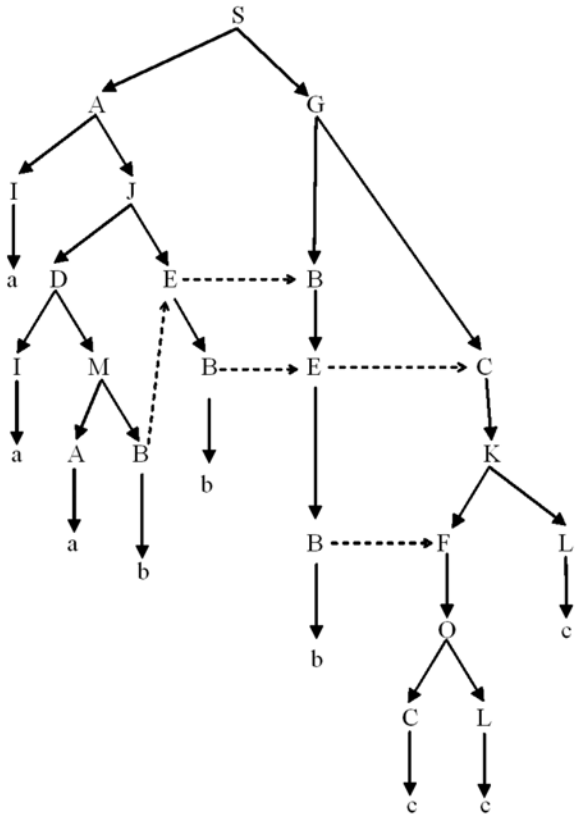

Figure 1. "Derivation-tree" in a grammar in Penttonen normal form

For context-sensitive grammars the left-most derivation is defined only in derivation graph sense opposite to the context-free case in which it is left-most in sentential-form sense as well.
Definition 2. *Let G = (N, T, S, H) a grammar in Penttonen normal form. The* left-most derivation *for a word w is a way of constructing its derivation tree. In each step the left-most non-terminal leaf symbol is used to provide new node(s) by derivation edge(s) (maybe using an in-context-edge).*
It is exactly the same order as a left-most traversal of the derivation tree based only on derivation edges.
Now we give a recursive algorithm which can provide a left-most derivation.
1. First the left-most branch of the graph will be derived up to a terminal leaf.
2. The right branch at the lowest branching will be derived in left-most way.

3. Until there exist not finished branch the previous step is applied.

Note, that the left-most derivation in context-sensitive case usually is not a real derivation in traditional (sentential form) sense. It is a way to construct the derivation tree with context-edges, but it relates to sentential derivations by Theorem 1. Having the derivation graph it is very easy to construct real (sentential) derivations in the following way. In a derivation step do not change any non-terminal from which a context edge starts to a non-terminal not already passed. The left-most derivation for the example is the following. The graph is built up in the next order: $S \to AG$, $A \to IJ$, $I \to a$, it is the left-most branch of the derivation-graph, it is independent of all the remaining part of the graph. The graph is continuing from the left-most non-terminal: $J \to DE$, $D \to IM$, $I \to a$, another branch is finished. The left-most non-terminal leaf is M, so $M \to AB$, $A \to a$. Now $B \to b$. Then E is the left-most non-terminal leaf, and a context is needed from the right-hand-side of the graph already given: $BE \to BB$. The next step is $B \to b$. Now $G \to BC$, and contexts are needed: $EB \to EE$, $BE \to BB$ and $B \to b$ finishes this branch. Now, $EC \to EK$, $K \to FL$, and a context-edge again: $BF \to BO$. Then $O \to CL$, $C \to c$ and $L \to c$ finish these branches, while $L \to c$ finishes the derivation-graph. The left-hand-side part of the graph is never changed, but the right-most non-terminals of the graph may needed as contexts to build up the remaining part. One can imagine that the non-terminals have shadows to the left-neighbor branch. And when a context is needed one can choose among the non-terminals having shadow at this node. These left-most derivations are usually not real derivations in the system in the usual (sentential form) sense, but there is a unique left-most derivation for every derivation tree. Having the left-most derivation or the derivation graph one can give any real derivation of the word represented by the derivation tree.

4 The Shadow-Pushdown Automata

In this section we define a new-type of automata. This, so-called, shadow-pushdown automata class is an extension of the pushdown automata class.

Definition 3. *A (1-state)* shadow-pushdown automaton *M consists of an input tape, a finite control and a stack, formally it is a system* $(\Sigma, \Gamma \cup \overline{\Gamma}, \delta, Z_0)$, *where* Σ *the input alphabet and* Z_0 *is a special stack-symbol as in the case of pushdown-automata.* Γ *is the stack-alphabet and it is extended by symbols* $\overline{\Gamma} = \{\overline{A} \mid A \in \Gamma\}$ *(they are called the shadow symbols of the original ones). The mapping* δ *goes form* $(\Sigma \cup \{\lambda\}) \times \Gamma \times (\overline{\Gamma})^*$ *to finite subsets of* $(\Gamma \cup \overline{\Gamma})^*$. *Initially only the symbol* Z_0 *is in the pushdown-stack. The interpretation of* δ *is the following. The top stack-symbol A is accessed (together with the shadow-symbols above). There are the following cases:*

[reading an input letter] $(\overline{A}) \in \delta(a, A, v)$ *for any* $v \in (\overline{\Gamma})^*$.

[extending the stack] $(\overline{A}uv) \in \delta(\lambda, A, v)$ *with* $u \in \Gamma\Gamma^*$ *for any* $v \in (\overline{\Gamma})^*$.

[using shadow symbol] $(\overline{B}C\overline{A}v_2) \in \delta(\lambda, B, v_1\overline{A}v_2)$ *for any* $v_1, v_2 \in (\overline{\Gamma})^*$. *A configuration is a pair* (*r, z*) *where r is the unprocessed part of the input word and* $z \in (\Gamma \cup \overline{\Gamma})^*$ *is the actual contents of the stack. A word w is accepted by this shadow-pushdown automaton, if there is a sequence of transitions starting* (*w;*Z_0) *and finishing by* (λ, v) *for some* $v \in (\overline{\Gamma})^*$ *according to the mapping* δ *in each step.*

Note, that in the traditional sense δ is not finitely defined, because the set $(\overline{\Gamma})^*$ is not finite. In spite of this fact it can be effectively defined and used. To have an immediate connection between grammars and shadow-pushdown automata, let $\Sigma = T, \Gamma = N$ and $Z_0 = S$.

Lemma 2. *The three kinds of movements of the shadow-pushdown automata correspond to the three kinds of derivation steps of the grammar in Penttonen normal form.*

- steps type [reading an input letter] correspond exactly to rules type $A \to a$,

- steps type [extending the stack] refer exactly for the rules type $A \to u$ *where u contains two non-terminals,*

- steps type [using shadow-symbol] correspond exactly to context-sensitive rules type $AB \to AC$.

There are three cases according to the possible types of derivation rules in Penttonen normal form. For rules type $A \to a$: This step is a context-free step it does not depend on the possible left-contexts, moreover it means that a branch is finished in the derivation and some nodes of the graph (which were shadow non-terminals till this derivation step) cannot be context any more in a left-most derivation represented by this graph. The symbol *A* is replaced by its shadow-symbol $\overline{A}$ that means the following: the derivation and the process are already continued from this non-terminal, but it can be used as a context later, since it has shadow to the next branch. For other context-free rules: $A \to BC$: $(\overline{A}CBv) \in \delta(\lambda, B, v)$ for any $v \in (\overline{\Gamma})^*$. In these step the symbol *A* may be needed as a context later, therefore we leave its shadow

($\overline{A}$) in the stack instead of the original symbol. Now, the most interesting case, simulation of the context-sensitive rules is shown. For rules type $AB \rightarrow AC$: In these steps the deletion of some shadow symbols guarantees the fact, that two context-edges do not cross each-other in the derivation-tree. The stack is 'empty' if there is not any stack symbol $A \in \Gamma$ inside. While the pushdown automata can do only operations pop and push (and, of course, read the input) the shadow-pushdown automata have the following
operations:
- it reads an input letter, deletes the shadow-symbols above the top non-shadow symbol and change this top symbol to shadow symbol.
- it changes the top non-shadow symbol to its shadowed pair and inserts new non-shadow symbols immediately above without changing the shadow-symbols above. It is enough to allow insertion of exactly two symbols.
- it changes the top non-shadow symbol to shadow symbol, puts a new non-shadow symbol immediately above and delete some consecutive shadow symbols immediately above (till a given shadow-symbol). So, in these type of steps the top non-shadow symbol and a shadow symbol above are together used, and shadow-symbols between them are deleted.

Lemma 3. *The configuration* (*r, z*) *of a shadow-pushdown automaton working on the word w represents a derivation tree of the string qt, where q is the processed part of w, i.e. w = qr. Moreover t is a string of non-terminals (non-shadow letters) in the stack reading top down.*

Proof. We use induction by steps. Starting with configuration (*w, S*) the condition holds. Now assume that it is true for (*r, z*). There are three cases according to the type of the next transition to get (*r', z'*) and *q'*. The same cases are used as in the previous lemma.
[reading an input letter] a letter will be removed from *r*, so *q' = qa* and a non-terminal (*A*) is removed from the stack (replaced by its shadow), no other non-shadow elements change. It represents exactly a step in which a new terminal labelled node (leaf) is derived.
[extending the stack] a non-terminal is replaced by its shadow and two new non-terminal pushed into the stack: *r' = r, q' = q* and *t = Ak* is developed to *t' = BCk* meaning that the leaves yield *rBCk* instead of *rAk*. It represent exactly the application of the rule $A \rightarrow BC$ in the derivation tree.
[using shadow-symbol] in these steps *r' = r; q' = q*. The top non-terminal *B* is replaced by a non-terminal *C* yielding *rCk* instead of *rBk* by the leaves. We show, that it works only if there is a possible context *A* at the left neighbor branch. So, the shadow of *A* is in the stack above the modified (i.e. the top) non-terminal. This fact implies that there is a node labelled by *A* on the left neighbor branch, because for every other node of the graph either the symbol representing the node is deeper on the stack than the top non shadow symbol or its shadow symbol was deleted by a step reading an input letter.

Moreover a symbol $\overline{A}$ can be used in the transition if and only if it is not deleted from the stack by any of the previous using shadow-symbol steps. These transitions delete exactly those shadow-symbols from which more context edges cannot start without crossing the context-edge needed for the actual derivation step (from the node labelled by *A* at the left neighbor branch represented by $\overline{A}$ in the stack, to the node labelled by *B*). So, the new configuration refers for the derivation tree for string *qt0* in this case as well. The proof is finished.

Theorem 2. *The shadow-pushdown automata simulate the left-most derivations of a context-sensitive grammar (in Penttonen normal form).*

Proof. Due to Lemmas 2 and 3 it is needed to show only that the order of the steps of the shadow-pushdown automata coincide to the left-most derivation. The automata can check only the top non-shadow symbol which represents the left-most non-terminal leaf of the derivation graph. In each step this non-terminal will be processed. Therefore the theorem holds.
Using the previous results for finished derivation trees we have the following statement.

Corollary 1. *The class of (1-state) shadow-pushdown automata accepts exactly the context-sensitive languages.*

In the remaining part of the section an example is detailed. First the construction of the shadow-pushdown automata for the grammar (and language) shown in Example 1 is provided.
So, the shadow-pushdown automata is the following: ($\{a, b, c\}$, $\{S, A, B, C, D, E, F, G, I, J, K, L, M, \overline{S}, \overline{A}, \overline{B}, \overline{C}, \overline{D}, \overline{E}, \overline{F}, \overline{G}, \overline{I}, \overline{J}, \overline{K}, \overline{L}, \overline{M}\}$, δ ; $S\}$.
The mapping δ is given by Table 1.
Now, let us see how this automaton accepts the word *aaabbbccc*. The numbers of transition steps from Table 1 are used. It starts with (*aaabbbccc, S*) the only step which can be applied results (*aaabbbccc, SGA*). Now there are two choices. Reading the input letter *a* (step 8) will not provide the correct continuation. So, step 2 is used: (*aaabbbccc, SGAJI*). Then step 14 must be used, reading an input *a* we get (*aabbbccc, SGAJI*). The next is step 3 yields (*aabbbccc, SGAJEDI*). Now step 5 or 11 can continue the process. Step 5 is used: (*aabbbccc, SGAJEDMII*). The only choice to continue is step 14: (*abbbccc, SGAJEDMI*). (Now we deleted the top shadow-symbol and the non-shadow *I* changed to *I*.) Step 6 follows: (*abbbccc, SGAJEDMBAI*). Now step 8 is preferred: (*bbbccc, SGAJEDMBA*).

	input tape	top stack symbol	shadow symbols over	new stack top-string
0	λ	S		$\overline{S}GA$
1	λ	G	v	$\overline{G}CBv$
2	λ	A	v	$\overline{A}JIv$
3	λ	J	v	$\overline{J}EDv$
4	λ	K	v	$\overline{K}LFv$
5	λ	D	v	$\overline{D}MIv$
6	λ	M	v	$\overline{M}BAv$
7	λ	O	v	$\overline{O}LCv$
8	a	A	v	$\overline{A}$
9	b	B	v	$\overline{B}$
10	c	C	v	$\overline{C}$
11	a	D	v	$\overline{D}$
12	b	E	v	$\overline{E}$
13	c	F	v	$\overline{F}$
14	a	I	v	$\overline{I}$
15	c	L	v	$\overline{L}$
16	λ	B	$v_1\overline{E}v_2$	$\overline{B}E\overline{E}v_2$
17	λ	C	$v_1\overline{E}v_2$	$\overline{C}K\overline{E}v_2$
18	λ	E	$v_1\overline{B}v_2$	$\overline{E}B\overline{B}v_2$
19	λ	F	$v_1\overline{B}v_2$	$\overline{F}O\overline{B}v_2$

Table 1. The mapping δ with the letter (or the empty word) read from the input tape, the top stack (non-shadow) symbol and the shadow symbols above this symbol. The resulted new top-string of the stack can be seen in the last column.

The possibilities are step 9 and 16, but step 16 is not a real possibility, because there is no *E* in the stack above *B*. Therefore step 9 is used and we have: (*bbccc, SGAJEDMB*). Now either step 12 or step 18 can be used. Let us step by the context-sensitive step and use shadow: (*bbccc, SGAJEBB*). Now two shadow symbols are deleted from the stack between the original *E* and the used shadow *b*. Now step 9 must use and gives (*bccc, SGAJEB*). Now the only possible computation uses step 1: (*bccc, SGCBAJEB*). Now step 9 and 16 are possible and step 16 is used. (*bccc, SGCBEEB*). Now step 12 and step 18 work. We use step 18 and it yields (*bccc, SGCBEBB*). The only way to continue is step 9: (*ccc, SGCBEB*). We should chose between step 10 and 17. The latter case results (*ccc, SGCKEB*). Step 4 is the applied: (*ccc, SGCKLFEB*). Now, step 13 and 19 can be used. Step 19 yields (*ccc, SGCKLFOB*). Step 7 results (*ccc, SGCKLFOLCB*). Now step 10 is the only applicable: (*cc, SGCKLFOLC*). Step 15 must be applied twice: (*c, SGCKLFOL*), and (λ*; SGCKL*). The stack contains only shadow symbols, so it is empty, the word is accepted.

5 Conclusion

In this paper "derivation-trees" and left-most derivations for context-sensitive grammars are presented. A new type of automata is defined. The so-called shadow-pushdown automata are such extensions of the pushdown automata that recognize exactly the context-sensitive languages. The work of these automata simulates the left-most derivation which can be easily translated to a real (sentential) derivation. The shadow symbols are used in the stack representing the non-terminals from which the derivation is already continued, but they have shadow, i.e. they may be needed as contexts in rules type $AB \rightarrow AC$ later. The stacks of these automata are used in the following way: it is assumed that the shadow symbols are half-translucent. We can access the top non-shadow symbol through them and it is also possible to check the shadow-symbols above the top non-shadow symbol. Our formalism is based on derivations in generative grammars, rather than the work of Turing-machines and other equivalent devices ([2], [4], etc.). The presented class of automata is the next element of the sequence starting by non-deterministic finite automata and pushdown automata. The difference between the finite and pushdown automata is the infinite stack. The difference of the pushdown and the shadow-pushdown automata is the set of shadow symbols which have additional influence on the work. There are some open problems are arisen. For instance, complexity analysis of the work of the shadow-pushdown automata, the stack-languages of accepted runs and analysis of the deterministic version are subjects of future work.

Acknowledgements. The work is supported by grants OTKA T049409, F043090 and SegraVis HPRN-CT-2002-00275.

References:

[1] Autebert, J-M; Berstel, J and Boasoon, L: *Context-Free Languages and Pushdown Automata*, in [10]

[2] Cerny, A: Description of words by cellular automata. *Kuwait J. of Sci. & Engineering* 24 (1997), 199-215.

[3] Hopcroft, J E and Ullmann, J D: *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[4] Latteux, M and Simplot, D: Context-sensitive string languages and recognizable picture languages. *Information and Computation* 138 (1997), 160-169.

[5] Martin-Vide, C; Mitrana, V and Paun, Gh (eds.): Formal Languages and Applications, Studies in Fuzziness and Soft Computing 148, Springer, 2004.

[6] Mateescu, A: *On Context-Sensitive Grammars*, in [5].

[7] Morvan, Ch and Rispal, Ch: Families of automata characterizing context-sensitive languages, *Acta Informatica* 41 (2005), 293-314.

[8] Nagy, B: Derivations in Chomsky-type grammars in mirror of parallelism, Theoretical Computer Science - Information Society 2004, Ljubljana, pp. 181-184.

[9] Penttonen, M: One-sided and two-sided context in formal grammars. *Information and Control* 25 (1974), 371-392.

[10] Rozenberg, G and Salomaa, A (eds.): *Handbook of Formal Languages*, 3 volumes, Springer, 1997.