Centrum Wiskunde & Informatica

REPORT*RAPPORT*

SEN

Software Engineering

Software ENgineering

A new automata for parsing semi-bracketed contextual grammars

L. Kuppusamy, M. Anand

REPORT SEN-E0802 DECEMBER 2008

Centrum Wiskunde & Informatica (CWI) is the national research institute for Mathematics and Computer Science. It is sponsored by the Netherlands Organisation for Scientific Research (NWO).
CWI is a founding member of ERCIM, the European Research Consortium for Informatics and Mathematics.

CWI's research has a theme-oriented structure and is grouped into four clusters. Listed below are the names of the clusters and in parentheses their acronyms.

Probability, Networks and Algorithms (PNA)

Software Engineering (SEN)

Modelling, Analysis and Simulation (MAS)

Information Systems (INS)

P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

ISSN 1386-369X

A new automata for parsing semi-bracketed contextual grammars

ABSTRACT
Bracketed and fully bracketed contextual grammars were introduced to bring the concept of tree structure to the strings by associating a pair of parentheses to the adjoined contexts in the derivation. But these grammars fail to generate the basic non-context free languages thus unable to provide a syntactical representation to natural languages. To overcome this problem, a new variant called semi-bracketed contextual grammar was introduced recently, where the selectors can also be non-minimally Dyck covered strings. The membership problem for the new variant is left unsolved. In this paper, we propose a parsing algorithm (for non-projected strings) of maximal semi-bracketed contextual grammars. In this process, we introduce a new automaton called k-queue Self Modifying Weighted Automata (k-quSMWA).

2000 Mathematics Subject Classification: 68Q45
1998 ACM Computing Classification System: F.4.2
Keywords and Phrases: Automata, contextual grammars, MCS formalism, derivation tree
Note: The first author,Äôs work was carried out during the tenure of an ERCIM ‚ÄúAlain Bensoussan‚Äù Fellowship Programme.

A NEW AUTOMATA FOR PARSING SEMI-BRACKETED CONTEXTUAL GRAMMARS

K. Lakshmanan[(1),]* and M. Anand [(2)]
(1) Centrum Wiskunde en Informatica, Amsterdam, The Netherlands. Email: L.Kuppusamy@cwi.nl
(2) School of Computing Sciences, VIT University, Vellore-632 014. Email: manand@vit.ac.in

Abstract: *Bracketed and fully bracketed contextual grammars were introduced in [7] to bring the concept of tree structure to the strings by associating a pair of parentheses to the adjoined contexts in the derivation. But these grammars fail to generate the basic non-context free languages thus unable to provide a syntactical representation to natural languages. To overcome this problem, a new variant called semi-bracketed contextual grammar was introduced in [4], where the selectors can also be non-minimally Dyck covered strings. The membership problem for the new variant is left unsolved. In this paper, we propose a parsing algorithm (for non-projected strings) of maximal semi-bracketed contextual grammars. In this process, we introduce a new automaton called k-queue Self Modifying Weighted Automata (k-quSMWA).*

Keywords: Contextual grammars, derivation tree, parsing, membership problem, MCS formalism.

1. INTRODUCTION

Contextual grammars were introduced by S. Marcus in 1969. They produce languages starting from a finite set of *axioms* and adjoining *contexts*, iteratively, according to a *selector* present in the current sentential form. If the contexts is adjoined at the ends is called *external* [5] and if the contexts is adjoined to the selector strings appearing as substrings of the string is called *internal* contextual grammars [10].

One of the important problems in the area of formal language theory and natural language processing is to obtain new classes of languages that provide an appropriate description for natural languages. In fact, the classes of languages searched for should have the so called 'mildly context sensitive' (MCS) properties which are considered to be an appropriate description for natural languages. The properties which describe MCS are as follows:

1. The class of languages contains all context-free languages.
2. The class of languages contains the following three basic non-context free languages.
   1. *multiple agreements* $L_1=\{a^nb^nc^n \mid n\geq 1\}$,
   2. *crossed dependencies* $L_2= \{a^nb^mc^nd^m \mid n\geq 1\}$,
   3. *marked duplication* $L_3=\{wcw \mid w\in\{a,b\}^*\}$
3. The class of languages should be *parsable in polynomial time.*
4. All languages in the class have the *bounded growth property*.

A class of languages possesses the MCS properties characterize the MCS family of languages and the corresponding class of grammars forms the MCS formalism. Generally, these MCS formalisms are considered to be a good model for the syntactical description of natural languages. For more details on MCS formalisms, we refer to [8]. Even though contextual grammars were introduced to give an appropriate model description for natural languages [5], the basic class, internal contextual languages itself is failed to contain the non-context-free constructions [1], [6]. Further, the membership problem for the above families of languages still remains open [3].

In context-free grammars the structure to the strings is preserved by means of derivation tree, where as for contextual grammars, no structure by means of derivation tree exists to the generated strings. In order to introduce the structure to the generated strings of contextual grammars, bracketed and fully bracketed grammars were introduced in [7]. The structure is preserved by introducing a pair of parenthesis to the contexts inserted at each derivation step but, when considering the suitability of these grammars to MCS formalisms, these grammars fail to generate the three basic non-context free languages [9], [4]. This has been overcome in [4] by relaxing the condition on the selectors in fully bracketed contextual grammars and this new class of grammars named semi-bracketed contextual grammars where the structure to the string is also maintained [4]. Besides, it was proved in [4] that this of class of languages contains the class of context-free languages and thus the properties 1 and 2 mentioned above are satisfied. By default all contextual languages satisfy the property 4.

In this paper, we make an attempt to solve the property 3 for the semi-bracketed contextual grammars by introducing new automata called k-queue Self Modifying Weighted Automata. By using backtracking concept at each derivation step a maximal selector is identified and the corresponding left and right contexts are removed from the input string. At latter stage of the parsing, if needed the necessary contexts are inserted in the input string and the parsing is continued until the corresponding axiom is identified.

* The author's work was carried out during the tenure of an ERCIM "Alain Bensoussan" Fellowship Programme.

2. PRELIMINARIES

In this section, we introduce the notion of formal languages and contextual grammars which are used in the paper. A finite non-empty set V is called an *alphabet*. We denote by V* the free monoid generated by V, by λ its identity or the *empty string*, and by V^+ the set V*- {λ}. The elements of V* are called *words*. For any word x ∈ V*, we denote |x| the *length of x*. For more details on formal language theory, we refer to [2], [11].

An *internal contextual grammar* is defined as $G = (V, A, (S_1, C_1),\ldots,(S_m, C_m)), m\geq 1$, where

- V is an alphabet
- A is a subset of V* is a finite set called the set of *axioms*,
- S_i is a subset of V*, $1\leq i\leq m$, are the finite set of *selectors*,
- C_i is a subset of V* x V*, C_i finite, $1\leq i\leq m$, are the finite set of *contexts*.

The *modular presentation* [10] of a contextual grammar is given as G = (V, A, P) where V, A are defined as above and P is the finite set of selector-context rules of the form $(S_1, C_1),\ldots,(S_m, C_m)$. The derivation in the *internal mode* (denoted by in) is defined as $x \Rightarrow_{in} y$ iff $x=x_1x_2x_3$, $y=x_1ux_2vx_3$, for $x_1,x_2,x_3 \in V^*$, $x_2 \in S_i$, $(u,v) \in C_i$ for some $1\leq i\leq m$. The *maximal mode* of the grammar is defined in the following way $x \Rightarrow_{M} y$ iff $x = x_1x_2x_3$, $y = x_1ux_2vx_3$, for $x_1x_2x_3 \in V^*$, x_2 belongs to S_i, $(u,v) \in C_i$ for some $1\leq i\leq m$ and there are no $x'_1, x'_2, x'_3 \in V^*$, such that $x = x'_1x'_2x'_3$, $x'_2 \in S_i$, and x'_2 contains x_2. That is, in this maximal mode, the chosen selector x_2 for the next derivation should not be contained (in substring sense) in a longer selector x'_2, where both $x_2, x'_2 \in Si$ for some *i*. The language generated by a contextual grammar G in internal and maximal mode is given as $L_\alpha(G) = \{x \in V^* \mid z \Rightarrow^*_\alpha x, z \in A\}$, where $\Rightarrow^*_\alpha$ is the reflexive transitive closure of the relation $\Rightarrow_\alpha$ and $\alpha \in \{in, M\}$.

Let us consider the brackets [,] and denote the set {[,]} by B. The *Dyck language* over B is denoted by D_B and it is the language generated by the context-free grammar G = ({S}, B, S, {S→SS, S→[S], S →λ}). Given the two disjoint sets V and B, we can define the projection mappings pr_V, pr_B, from (V ∪B)* to V*, B*, respectively as follows:

$$pr_\beta(a) = \begin{cases} a, \text{ for } a \in \beta \\ \lambda, \text{ for } a \notin \beta, \text{ where } \beta \in (V, B) \end{cases}$$

A string $x \in (V \cup B)^*$ is said to be a *Dyck covered string* if $x \Rightarrow^* \lambda$, by reduction rules of the form [w]→λ, for $w \in V^*$. For instance, x_1=[a[a]a[a]], x_2=[[a]], x_3=[[a[a]a]a] are Dyck covered strings. A *Dyck covered string* $x \in (V \cup B)^*$ is said to be *minimally Dyck covered string* if the following conditions are hold:

1. if $x = x_1]x_2[x_3$ with $x_1, x_3 \in (V \cup B)^*$ and $x_2 \in V^*$, then $x_2 = \lambda$
2. The reduction rule [ ]→ λ is not used when reducing x to λ.

Condition 1 refutes string x_1 above, condition 2 refutes string x_2, hence these strings are not minimally Dyck covered; the string x_3 is of this type. We denote the language of all minimally Dyck covered strings over the alphabet V by MDC(V). For every string $x \in$ MDC(V), a unique derivation tree can be associated (refer Appendix).

A *bracketed contextual grammar* is a tuple $G = (V,A,(S_1, C_1),\ldots,(S_m, C_m))$, $m\geq 1$, where V is an alphabet, A is a finite subset of MDC(V), called axioms, $S_i \subseteq V^*$, and C_i are finite subsets of $V^* \times V^* - \{(\lambda, \lambda)\}$ for all $1 \leq i \leq m$. The derivation relation (in internal contextual mode) is defined as follows: for x, $y \in (V \cup B)^*$, we write $x \Rightarrow_G y$, iff $x = x_1x_2x_3$, $y = x_1[ux_2v]x_3$, $x_1,x_3 \in (V \cup B)^*$, $x_2 \in$ MDC(V) and $pr_V(x_2) \in S_i$, $(u,v) \in C_i$, for some $1 \leq i \leq n$.

A *fully bracketed contextual grammar* (in short, FBIC grammar) is very similar to bracketed contextual grammar, except that the selectors are in MDC(V) instead of $S_i \subseteq V^*$, and no projection is applied to the chosen selector. It is proved in [7] that if $x \Rightarrow_G y$ is a derivation step in a bracketed or fully bracketed contextual grammar, then $y \in$ MDC(V) whenever $x \in$ MDC(V).

A *semi-bracketed contextual grammar* (in short, SBIC grammar) is a construct $G = (V,A,(S_1,C_1),\ldots,(S_m, C_m))$, $m\geq 1$, where V is a finite set of alphabet, A ∈ MDC(V) is a finite set of axiom, $S_i \subseteq [(V \cup B)^* \cup (V \cup B)^*]$ and C_i are finite subsets of $V^* \cup V^* - \{(\lambda, \lambda)\}$ for $1 \leq i \leq m$, with the condition that whenever C_i contains a context (u, λ), $u \in V^+$ for some *i*, then the corresponding selector is of the form $S_i \in [(V \cup B)^*$ and whenever C_i contains a context (λ, v), $v \in V^+$ for some *i*, then the corresponding selector is of the form $S_i \in (V \cup B)^*]$. Note that, when the context is not one-sided (one sided means either u = λ or v = λ in (u, v)), the corresponding selector may be of any type; may start or end with a bracket. The derivation relation is defined as follows. For x, $y \in (V \cup B)^*$, we write $x \Rightarrow_G y$ if and only if $x = x_1x_2x_3$, $y = x_1[ux_2v]x_3$, where $x_1, x_3 \in (V \cup B)^*$, $x_2 \in S_i$, (u, v) $\in C_i$, for some $1\leq i \leq m$.

When the maximal condition (i.e., choosing the selector of maximal length) is included with this semi-bracketed contextual grammar, the grammar is said to be maximal semi-bracketed contextual grammar and is denoted by MSBIC grammar. The string language generated by a bracketed or fully bracketed or semi-bracketed contextual grammar $G = (V, A,(S_1,C_1),\ldots,(S_m, C_m)),m\geq 1$, can be defined as two types of languages, one by collecting the strings without applying any projection (non-projected) and the other by collecting the strings after applying the projection. The former one is defined as $L_{NPro}(G) = \{w' \mid z \Rightarrow^*_G w'$, for some $z \in A\}$, and the later one is defined as $L(G) = \{pr_V(w) \mid z \Rightarrow^*_G w$, for some $z \in A\}$, where $\Rightarrow^*_G$ is the one discussed already.

3. MEMBERSHIP ALGORITHM

In this section, we introduce our proposed parsing algorithm for the non-projected strings of semi-bracketed contextual grammars. First we present the general flow of the algorithm in the following major V steps and discuss each one in detail.

Step I: The input is checked for MDC(V)

Step II: All possible Left contexts (Lc) are identified in the given input string

Step III: A selector of maximal length is identified for the corresponding Lc by using k-queue Self Modifying Weighted Automata (k-quSMWA).If k-queue Self Modifying Weighted Automata (k-quSMWA) is not able to identify a maximal selector for all Lc in a particular Selector context table (S_i, Lc_i) then insertion of context is done (either single or multiple) in the input string and step III is repeated

Step IV: The Lc and corresponding Rc are identified for maximal selector and removed from the input string

Step V: This step is used to identify whether w' $\in$ L(G)

Input: Input string (non-projected) w' $\in$ MDC(V) and a MSBIC (G) grammar

Output: w' $\in$ L(G) before applying projection, no if if w' $\notin$ L(G)

Method:

Step I: Scan the input from left to right and check whether the w' $\in$ MDC(V)

Step II: // This step is used to fill the necessary values in the selector-context and axiom table by using the given grammar G and by using the selector-context table all possible left contexts are identified in the input string and position of the identified contexts are appended in the corresponding selector-context table.

Step 2.1: Using the MSBIC grammar the following values are entered in the selector-context table (S_i, Lc_i where S_i denotes the selector and Lc_i denotes the left context) and axiom table.

(1)selector, (2) (Lc, length), (3) (Rc, length)

(1)axiom, (2)length

Step 2.2: By using the selector-context table (using the ([x, length) all possible positions of Lc are identified in the given input string and the following value are appended in selector- context table

(1) position of the context in the input string (the position will be n-tuple where n is the |Lc|)

Step 2.3: Repeat the step 2.2 for all Lc and for all selectors by using selector-context table.

Step III: // Using the selector-context table a maximal selector is identified by using k-quSMWA and the required values are entered in selector-position table. If k-quSMWA is not able to identify a maximal selector for all Lc in a particular (Si, Lc_i) table then single or multiple insertion of context is done and necessary values are filled in recently-inserted context table and |w'| is modified and proceed further.

Step 3.1: for all (S_i, Lc_i) tables do

Step 3.2: for (n-1) entries in (if the number of entries is more than one) each (S_i, Lc_i)table

Step 3.2.1: Start scanning the input from the (last index in n-tuple)+1 position (the position can be identified from S_i Lc_i table)

Step 3.2.2: if there exists a maximal selector for the Lc by using k-queue Self Modifying Weighted Automata *(k-quSMWA)* the following values are filled in selector-position table and go to step (IV)

(1) selector, (2) selector position (positions are identified by using index i from (k-quSMWA), (3) Length of the identified selector

else return to step 3.2

if k-quSMWA does not identifies the maximal selector for all the contexts Lc_i in (S_i, Lc_i) table then do the following

Step 3.2.2.1:// insertion of context in the input string is done from the end of table

insert the last removed Lc and Rc in the input string such that the remaining (n-1) entries (i.e from the end of the table) are not substring of the selector S_i and the following values are entered in recently-inserted context table for corresponding selector S_i selector with position and |w'| is modified as |w'| + total length of inserted context (|Lc| + |Rc|) which can be identified from removed context table.

Step 3.2.2.2:

if there is a presence of a substring then do

Step 1: Insert the necessary removed Lc and Rc in input string such that their positions does not overlap in the input string and the following values are entered in the recently-inserted context table for corresponding selector S_i

(1) selector with position, (2) Lc, (3) Rc

Step 2: |w'| is modified as |w'| + total length of inserted context (|Lc| + |Rc|) which can be identified from removed- context table.

Step 3: For the both steps check the (S_i, Lc_i) table and insert Lc with position if the Lc is not there.

Step 3.2.3:Continue from step 3.1 ((S_i, Lc_i) will be the recently modified table).

Step IV: Using the selector-position table the Rc is identified and (Lc, Rc) are removed from the input string and |w'| is modified. If the corresponding Rc is not identified by using the recently-inserted context table the necessary Lc and Rc are removed from the input string and |w'| is modified.

Step 4.1: Using selector-position table start scanning the input from the end of the identified selector to identify the corresponding Rc

Step 4.2: if Rc is found then do

Step 1: remove Lc and Rc from the input string and fill the following values in the removed-context table

(1) selector, (2)Lc, (3) Rc, (4) position of Lc and Rc, (5) total length of removed contexts (Lc + Rc)

Step 2: remove the Lc with position from the (S_i, Lc_i)table

Step 3: |w'| is modified as |w'| - total length of removed context and goto step (4.3)

else

Step 1: Remove Lc and Rc from the input string by using recently-inserted context table for the corresponding S_i and the |w'| is modified as |w'| - total length of inserted context

Step 2: Remove the Lc with position from the (S_i, Lc_i)table and go to step(3.2)

Step 4.3: Scan the input for the presence of a substring with a pattern of removed Lc and include it in the corresponding (S_i, Lc_i) table if it is not available and go to stepV.

Step V: //This step is used to check for the presence of the axiom

if (|w'| > length of all axioms in axiom table)
continue from step III
else if (|w'|=|a_i|)
case 1: // if the axiom table is having only one axiom
step 1: check w' with a_i if yes w' $\in$ L(G) & exit
else w' $\notin$ L(G) & exit
case 2: // if the axiom table is having more than one axiom
step 1: check w' with a_i if yes w' $\in$ L(G) & exit
else
continue from step (III) until |w'| identifies its corresponding axiom in the axiom table.

In order to identify the maximal selector we introduce a new automata in the following section. The transition graph to the corresponding automata will have the edges with weights (i,j,k) which are self modifying according to the input read.

4. DEFINITION AND BEHAVIOR OF k-queue Self Modifying Weighted Automata (k-quSMWA)

A k-quSMWA is defined as = (Q, Σ, δ, q_0, F, $\mathbb{N}'$)
where Q is non-empty finite set of states
Σ is finite set of symbols given by (Σ $\cup$ B)* where Σ is alphabets from MSBIC (G) $\cup$ { γ }
where γ consists of following symbols

Δ is used for a transition between non-looping automata to looping automata

π is used for a transition to the starting state of the looping automata

is used for a transition between looping automata to non-looping automata

$ is used for a transition between two non-looping automata (whenever there is a looping automata between non-looping automata)

$q_0 \in Q$ is the initial state
$F \subseteq Q$ is a set of final states.
$\mathbb{N}'$ is $\mathbb{N}_f \cup \{ \Phi \}$
where $\mathbb{N}_f$ is a finite set of positive integers {i, j, k}

i is an index denoting the position of the current input symbol scanned (initially it will be (last index in the n-tuple)+1 which can be identified from the selector-context table for the current left context)

j is an index denoting the length of the selector for the corresponding left context (initially it will be zero)

k is an index defining k-level queue whenever regular selector is used (i.e for the following operations kleene or positive closure initially it will be zero)

Φ is null value

where δ is defined as follows

In general δ will be

$\delta: Q \times \Sigma \times \mathbb{N}' \times \mathbb{N}' \times \mathbb{N}' \rightarrow Q \times \mathbb{N}' \times \mathbb{N}' \times \mathbb{N}'$

where i,j,k values will be modified based on Σ and for some of the moves all the $\mathbb{N}_f$ may not be used for such cases Φ will be used.

1) $\delta(q_i, \Sigma\text{-}\{\gamma\}, i, j, \Phi) = (q_{i+1}, i, j+1, \Phi)$
2) $\delta(q_i, \Delta, i, j, \Phi) = (q_j, i, j, \Phi)$ where q_j is the start state of the looping automata
3) $\delta(q_j, \Sigma, i, j, k) = (q_{j+1}, i, j+1, k+1)$
4) $\delta(q_p, \pi, \Phi, \Phi, \Phi) = (q_j, \Phi, \Phi, \Phi)$ where q_j is the start state of the looping automata
5) $\delta(q_p, \#, i, j, k) = (q_{p+1}, i, j, \Phi)$ (applied only when q_p is a non-final state)
6) $\delta(q_i, \$, i, j, \Phi) = (q_k, \Phi, \Phi, \Phi)$ where the automata should have the following move (q_k, next input symbol, i,j+1,Φ)

Various data structures introduced:

k-level Queue(k-Qu):
A k-level queue is defined based on the current k-value in the looping automata which is used to identify the maximal length in the loop. The entries in the queue will be (current input symbol scanned, position)

k-Length Substring Pumping Stack (k-LSPS):
A k-LSPS is based on the current k-value in the looping stack which is used to store the removed contexts after the identification of a loop in the selector. The entries in the stack will be (k-length substring, positions)

Recently Removed Context Stack (RRCS):
After identifying that the popped contexts k-level does not matches the string in the looping automata the removed contexts should be stored in RRCS with their respective positions. When the move $\delta(q_p, \pi, \Phi, \Phi, \Phi) = (q_j, \Phi, \Phi, \Phi)$ is applied based on the current value of k in the looping automata k-level queue is defined and the following operations should be done

1) scan k-symbols from the input from i_{th} position
2) push the k-symbols on the queue

3) push the k-length substring on k-LSPS
4) pop the k-symbols from the k-queue and perform the move (3)
5) When the popped context does not matches the string in the automata store the recently removed context in RRCS.

if q_p is a final state then return maximal selector with positions and length.

else

(1) apply the move (5)

(2) case 1:

pop the top most string from k-LSPS apply move (1) k-times and pop the top most string from RRCS and apply move (1) k-times

case 2:

pop the top most string from RRCS and apply move (1) k-times and perform move (1)

(3) return maximal selector with positions and length

Special Case :

$\delta(q_s, \Delta, i, j, \Phi) = (q_j, \Phi, \Phi, \Phi)$ where q_s is the final state and q_j is the start state of the looping automata then

step 1) current=j

step2)$\delta(q_j, \Sigma, i, j, k) = (q_{j+1}, i, j+1, k+1)$

By applying (step 2) k-times if q_p is reached where q_p is a final state

do 2.1) a k-level queue is defined based on the current k-value and $Current_1=j$

2.2) scan k-symbols from the input from the i^{th} position and push k-symbols on queue

2.3) pop the k-symbols and continue from step (1) where j=current1 (Σ is popped contexts from the queue)

else

return the corresponding maximal selector and its position and its length (i.e., current)

By constructing the above automaton (refer appendix for examples) the selector of maximal length for the Lc is identified and the selector position is entered in the selector-position table.

Refer Appendix for examples of k-quSMWA for various selectors.

5. CONCLUSION

In this paper, we have proposed a parsing algorithm for maximal semi-bracketed contextual grammars for the non-projected strings by introducing a new type of automata called k-queue Self Modifying Weighted Automata (k-quSMWA). We have used the following back tracking concept to solve the membership problem: at each derivation step a maximal selector is identified and the corresponding left and right contexts are removed from the input string. If some contexts pair is removed from the input string in wrong positions, they are identified at a little later stage and corrected by replacing the removed contexts. Since the correction happens only minimum number of times, we believe that the algorithm runs in deterministic polynomial time. The automaton is constructed to identify the maximal selector while back tracking. In connection with the proposed work, the parsing of maximal semi-bracketed contextual grammars for projected strings in polynomial time is left as an open problem. If this is solved, a new class of MCS formalism with structured strings is obtained in the domain of contextual grammars. Also, it will be interesting to analyze the acceptance power of the introduced automata in relevance with existing types of automata.

REFERENCES

[1] A. Ehrenfeucht, L. Ilie, Gh. Paun, G. Rozenberg and A. Salomaa, *"On the generative classes of contextual grammars. In Mathematical Linguistics and Related Topics",* The publ. House of the Romanian Academy: Bucharest, 105-118, 1995.

[2] J.E. Hopcroft and J.D. Ullman, "*Introduction to automata theory, languages and computation",* Narosa Publishing House, 1979.

[3] L. Ilie, *"On computational complexity of contextual languages",* Theo. Comp. Science. 183/1, 33-44, 1997.

[4] K. Lakshmanan, *"Semi-Bracketed contextual grammars"*, Proceedings of the second International Workshop on Non-Classical Formal Languages in Linguistics (ForLing) 2008, Tarragona, Spain, 41-55, Sept. 2008.

[5] S. Marcus, *"Contextual grammars"*, Rev. Roum. Pures. Appl. 14, 1525-1534, 1969.

[6] S. Marcus, C. Martin-Vide and Gh. Paun, *"On internal contextual grammars with maximal use of selectors",* Proceedings of 8th Conference on Automata and Formal Languages. Salgotarjan, 1996.

[7] C. Martin-Vide and Gh. Paun, *"Structured contextual grammars"*, 1, 33-55, 1998.

[8] S. Marcus, C. Martin-vide and Gh. Paun, *"Contextual grammars as generative models of natural languages",* Computational Linguistics, 24(2), 245-274, 1998.

[9] Gh. Paun, *"Marcus Contextual grammars",* Kluwer Academic Publishers: Dorrecht, The Netherland, 1997.

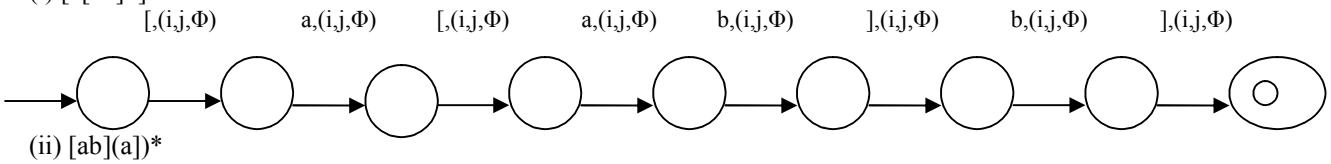
[10] Gh. Paun and X.M. Nguyen, *"On the inner contextual grammars",* Rev. Roum. Pures. Appl. 25, 641-651, 1980.

[11] A. Salomaa, *"Formal languages",* Academic Press: New York, 1973.

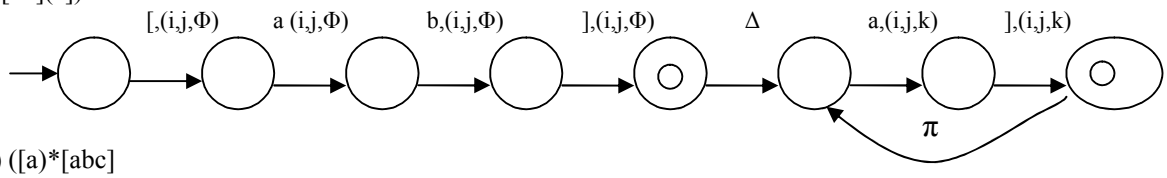
Appendix

Examples of k-quSMWA for different selectors

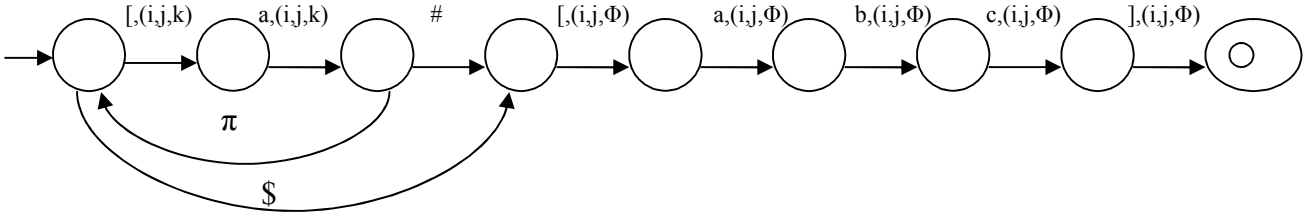
(i) [a[ab]b]



(ii) [ab](a])*



(iii) ([a)*[abc]



Derivation tree for strings∈MDC(V)

For every string $x \in$ MDC(V), we can associate a tree $\tau(x)$ with the labeled edges in the following way

- draw a dot representing the root of the tree; the tree will be represented with the root up and all the leaves down
- scan x from the left to right and grow $\tau(x)$ according to the following two rules
    1. for each *maximal* substring $[w$ of x, with $w \in V^*$ (since w is maximal, after w we find either [or]), we draw a new edge, starting at the current point of the partially constructed $\tau(x)$, marked with w on its left side, and placed to the right hand of the currently constructed tree;
    2. For each maximal $w]$, $w \in V^*$, not scanned yet (hence, either we find) before w, or $w=\lambda$ and to the left of] we have a substring $[z$ for some $z \in V^*$ already scanned], we climb the current edge, writing w on its right side.

A derivation tree $\tau(x)$ for the word x=[a][a[a[a[ab]b]c]ab][a]