Theoretical and Applied Informatics
ISSN 1896–5334
Vol.23 (2011), no. 2
pp. 97–106
DOI: 10.2478/v10179-011-0007-7

Scattered Context Grammars Generating Sentences Followed by Derivation Trees

ALEXANDER MEDUNA, STANISLAV ŽIDEK

Department of Information Systems
Faculty of Information Technology
Brno University of Technology
Božetěchova 2,
612 66 Brno

Received 17 May 2011, Revised 12 July 2011, Accepted 22 July 2011

Abstract: Propagating scattered context grammars are used to generate sentences of languages defined by scatterd context grammars followed by the strings corresponding to the derivation trees. It is proved that for every language defined by a scattered context grammar, there exists a propagating scattered context grammar whose language consists of original language sentences followed by strings representing their derivation trees.

Keywords: parallel grammars, scattered context grammars, derivation trees

1. Introduction

Parsing performed in parallel represents an area of intensive research concerning compilers today (see [2], [4] and [3]). As compilers are usually based on suitable formal models, such as grammars, the investigation of various parallel grammars is of great importance. Scattered context grammars are one of the intuitive yet very powerful types of parallel grammars, so their use related to parsing definitely deserves our attention.

In this paper, we use the propagating scattered context grammars, which contain no erasing productions, to generate sentences of languages defined by scattered context grammars followed by the string representation of the corresponding derivation tree. This approach extends the idea of generating sentences followed by their parses (see [8], [11], [7])—instead of parses, we generate the derivation trees, which in our opinion more exhaustively describes the derivation process. We demonstrate that for every scattered context grammar G, there exists a propagating scattered context grammar that

generates the sentences of the language $L(G)$ followed by the string representation of the derivation tree corresponding to the derivation of the sentence in grammar G. This characterization of recursively enumerable languages is of some interest, because the family of languages generated by *propagating* scattered context grammars is included in the family of context sensitive languages, which is properly included in the family of languages generated by scattered context grammars.

In Section 2., we introduce the preliminaries used throughout the rest of the paper. Also, we define the key notion of our article. In Section 3., we present our results, i.e. the algorithm taking a scattered context grammar and constructing the propagating scattered context grammar, which generates input grammar's sentences followed by the string representation of derivation tree. Furthermore, the proof of the algorithm's correctness is given and various implications are discussed, too. In Section 4., we make some final notes and suggestions regarding the future investigation.

2. Preliminaries and the definition

We assume a reader is familiar with the formal language theory (see [9]).

For an alphabet V, V^* denotes the free monoid generated by V under the operation of concatenation, with the unit element ε. Set $V^+ = V^* - \{\varepsilon\}$. For $w \in V^*$, $|w|$ denotes the length of w and $\mathrm{alph}(w)$ denotes the set of symbols appearing in w. For $U \subseteq V$, $|w|_U$ denotes the number of occurrences of symbols from U in w.

A scattered context grammar (SCG, see [6]) is a quadruple, $G = (V, T, P, S)$, where V is a total alphabet, $T \subset V$ is a finite set of terminal symbols (terminals; symbols from $V - T$ are called nonterminal symbols or nonterminals), $S \in V - T$ is the starting symbol and P is a finite set of productions of the form $p : (A_1, \ldots, A_n) \to (x_1, \ldots, x_n)$, where $A_i \in V - T$, $x_i \in V^*$ for all $i : 1 \leq i \leq n$ and p is a unique production label. For $p : (A_1, \ldots, A_n) \to (x_1, \ldots, x_n) \in P$, $\mathrm{lhs}(p)$ and $\mathrm{rhs}(p)$ denote $A_1 A_2 \ldots A_n$ and $x_1 x_2 \ldots x_n$, respectively. A propagating SCG is a SCG $G = (V, T, P, S)$ in which every $(A_1, \ldots, A_n) \to (x_1, \ldots, x_n) \in P$ satisfies $x_i \in V^+$ for all $i : 1 \leq i \leq n$.

Let $G = (V, T, P, S)$ be a (propagating) SCG, $y = u_1 A_1 u_2 \ldots u_n \ A_n u_{n+1}$, $z = u_1 x_1 u_2 \ldots u_n \ x_n u_{n+1}$, $y, z \in V^*$, $p = (A_1, \ldots, A_n) \to (x_1, \ldots, x_n) \in P$. Then y directly derives z in the SCG G according to the production p, $y \Rightarrow_G z \ [p]$ (or simply $y \Rightarrow_G z$). Let $\Rightarrow_G^+$ and $\Rightarrow_G^*$ denote the transitive and the reflexive-transitive closure of $\Rightarrow_G$, respectively. To express that G makes the derivation from u to v by using the sequence of productions $p_1, p_2, \ldots, p_n \in P$, we write $u \Rightarrow_G^* v \ [p_1 p_2 \ldots p_n]$ (or $u \Rightarrow_G^+ v \ [p_1 p_2 \ldots p_n]$ to emphasize that the sequence is non-empty). The language generated by G is denoted by $L(G)$ and defined as $L(G) = \{w : w \in T, S \Rightarrow_G^* w\}$. We often abbreviate $\Rightarrow_G$ to $\Rightarrow$ when it is clear which grammar we refer to.

We also assume that the reader is familiar with graph theory. By a *tree*, we will

automatically mean a *labeled ordered tree*. Let Υ be a tree, Θ be a set of nodes of Υ, $\theta \in \Theta$, n be a nonnegative integer. Then, $\mathrm{root}(\Upsilon)$ denotes the root node of the tree, $\mathrm{child}(\theta)$ denotes an n-tuple of node's child nodes (zero-tuple for leaf nodes), and $\mathrm{lab}(\theta)$ denotes a label of the node θ. Sometimes, we will generalize the notion of lab to n-tuples, $\mathrm{lab}((\theta_1, \ldots, \theta_n)) = (\mathrm{lab}(\theta_1), \ldots, \mathrm{lab}(\theta_n))$.

Let $G = (V, T, P, S)$ be a SCG and $p = (A_1, \ldots, A_i, \ldots, A_n) \rightarrow (x_1, \ldots, x_i, \ldots, x_n) \in P$, $x_i = a_1 \ldots a_m$, $m \geq 0$, $a_j \in V$ for all $j : 0 \leq j \leq m$. The *production tree* of the i-th component of the production p, denoted by $\mathrm{pt}(p, i)$, is an elementary tree Υ such that $\mathrm{lab}(\mathrm{root}(\Upsilon)) = A_i$ and

$$\mathrm{lab}(\mathrm{child}(\mathrm{root}(\Upsilon))) = \begin{cases} \lambda & \text{if } m = 0 \\ (a_1, \ldots, a_m) & \text{otherwise} \end{cases}$$

Consider a SCG $G = (V, T, P, S)$ and a derivation $S \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_m$ in G. The *derivation tree* corresponding to this derivation, denoted by $S = w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_m \; [\![\Upsilon]\!]$, is a labeled tree Υ constructed as follows:

1. Create a root node, $\mathrm{lab}(\mathrm{root}(\Upsilon)) = S$.
2. Set $j = 0$.
3. Repeat until $j = m$:
   (a) Let $w_j = u_1 A_1 \ldots A_n u_{n+1}$, $w_{j+1} = u_1 x_1 \ldots x_n u_{n+1}$, $A_i \in V - T$, $x_i \in V^*$ for all $i : 1 \leq i \leq n$, $p : (A_1, \ldots, A_n) \rightarrow (x_1, \ldots, x_n) \in P$, $\theta_1, \ldots \theta_n$ be a leaf nodes of Υ in this order (considering preorder tree traversal), $\mathrm{lab}(\theta_i) = A_i$. Add child nodes to θ_1 through θ_n so that it holds that subtree rooted at θ_j is a $\mathrm{pt}(p, i)$.
   (b) Increment j.

A left-bracketed representation of a derivation tree Υ, denoted by $\mathrm{lbr}(\Upsilon)$, is defined recursively as follows:

1. If $\mathrm{child}(\mathrm{root}(\Upsilon)) = ()$, then $\mathrm{lbr}(\Upsilon) = \mathrm{lab}(\mathrm{root}(\Upsilon))$.
2. If $\mathrm{child}(\mathrm{root}(\Upsilon)) = (\theta_1, \ldots, \theta_n)$, then

$$\mathrm{lbr}(T) = \mathrm{lab}(\mathrm{root}(\Upsilon))\langle \mathrm{lbr}(\Upsilon_1) \ldots \mathrm{lbr}(\Upsilon_n) \rangle,$$

where Υ_i is a subtree rooted at θ_i for all i: $1 \leq i \leq n$ (see 1).

Definition 2.1. *Let $G = (V, T, P, S)$ be a SCG. Then $G^{\blacktriangle}$ denotes a set of SCGs generating sentences of G followed by their derivation trees in the left-bracketted representation, that is*

$$G^{\blacktriangle} = \left\{ G_{\triangle} : L(G_{\triangle}) = \{ w \; \mathrm{lbr}(\Upsilon) : S \Rightarrow_G^* w \; [\![\Upsilon]\!], \; w \in T^* \} \right\}$$

3. Results

This section demonstrates that for every SCG G, there exists a *propagating* SCG $G_\triangle \in G^{\blacktriangle}$.

Algorithm 1. *Construction of* $G_\triangle \in G^{\blacktriangle}$
Input: *a SCG* $G = (V, T, P, S)$
Output: *a* propagating *SCG* $G_\triangle = (V_\triangle, T_\triangle, P_\triangle, S'') \in G^{\blacktriangle}$
Method: *Let* $T_\triangle = V \cup \{\langle, \rangle, \lambda\}$, $\Phi_\triangle = \{A_\triangle : A \in (V - T)\}$, $\Phi'_T = \{a' : a \in T \cup \{\langle, \rangle, \lambda\}\}$, $\Phi'_N = \{A' : A \in V - T\}$, $\Phi' = \Phi'_T \cup \Phi'_N$, $\Phi = \{\$, @, \#\}$, $N_\triangle = \Phi_\triangle \cup \Phi' \cup \Phi$, *and* $V_\triangle = T_\triangle \cup N_\triangle$. *Without loss of generality, assume that* $\Phi_\triangle$, Φ', Φ, $\{\langle, \rangle, \lambda\}$ *and* V *are pairwise disjoint.*

Let h *be a coding* $h : V^* \to \big((\Phi'_T - \{\langle', \rangle', \lambda'\}) \cup \Phi_\triangle\big)^*$ *such that* $h(a) = a'$ *for every* $a \in T$ *and* $h(A) = A_\triangle$ *for every* $A \in (V - T)$.
Let g *be a function* $g : V^* \to \big((\Phi'_T - \{\langle', \rangle'\}) \cup \Phi_\triangle\big)^*$ *such that:*

$$g(x) = \begin{cases} \lambda' & \text{if } x = \varepsilon \\ h(x) & \text{otherwise} \end{cases}$$

Construct $P_\triangle$ *as follows:*

1. *For each* $p : (S) \to (x) \in P$, *add* $1_x : S'' \to \$g(x)$ *to* Ξ_1. *Add contents of* Ξ_1 *to* $P_\triangle$.
2. *For each* $p : (A_1, \ldots, A_n) \to (x_1, \ldots, x_n) \in P$, *add*

$$2_p : (\$, A_{1_\triangle}, \ldots, A_{n_\triangle}) \to (\$, A'_1\langle' g(x_1)\rangle', \ldots, A'_n\langle' g(x_n)\rangle')$$

   to Ξ_2. *Add contents of* Ξ_2 *to* $P_\triangle$.
3. *Add* $3 : (\$) \to (@\#)$ *to* $P_\triangle$.
4. (a) *For each* $A \in (V - T) \cup \{\langle, \rangle, \lambda\}$, *add* $4_{A'} : (@, \#, A') \to (@, A, \#)$ *to* Ξ_4. *Add contents of* Ξ_4 *to* $P_\triangle$.
   (b) *For each* $a \in T$, *add* $4_a : (@, \#, a') \to (a@, a, \#)$ *to* $P_\triangle$.
5. *Add* $5 : (@, \#) \to (S\langle, \rangle)$ *to* $P_\triangle$.

Lemma 3.1. *Algorithm 1 is correct, i.e.* $G_\triangle \in G^{\blacktriangle}$.

Proof

Basic Idea $G_\triangle$ makes the derivation of string $w\ \mathrm{lbr}(\Upsilon)$, such that $S \Rightarrow_G^* w\ [\![\Upsilon]\!]$, by productions introduced in 1 through 5 in this order. It starts by applying a production from Ξ_1. Then, it simulates the construction of the derivation tree of grammar G in the left-bracketted representation. More precisely, it simulates the construction of the production tree of every

$$p : (A_1, \ldots, A_n) \rightarrow (x_1, \ldots, x_n)$$

by using

$$2_p : (\$, A_{1_\triangle}, \ldots, A_{n_\triangle}) \rightarrow (\$, A'_1\langle g(x_1)\rangle, \ldots A'_n\langle g(x_n)\rangle).$$

Construction of the derivation tree ends by applying the production introduced in (3), after which the sentential form of $G_\triangle$ has the form

$$@\#a'_1a'_2\ldots a'_n,$$

where $S\langle a_1a_2\ldots a_n\rangle = \mathrm{lbr}(\Upsilon)$. By using productions from (4), it goes through the derivation tree and transforms every $a' \in \Phi'$ to $a \in T_\triangle$ while copying every $a \in T$ in front of non-terminal @ (using productions introduced in (4b)). Before the final step, the sentential form has the form

$$w\ @a_1a_2\ldots a_n\#$$

Finally, $G_\triangle$ uses production 5 to turn nonterminals @ and # into the root of derivation tree and missing top level brackets.

To be more compact, whole derivation can be expressed as follows:

$$\begin{array}{lllllll} S'' & \Rightarrow_{G_\triangle} & & \$ & & g(x) & [1_x] \\ & \Rightarrow_{G_\triangle}^* & & \$ & & y & [\rho] \\ & \Rightarrow_{G_\triangle} & & @ & \# & y & [3] \\ & \Rightarrow_{G_\triangle}^* & w & @z & \# & & [\sigma] \\ & \Rightarrow_{G_\triangle} & w & S\langle z\rangle & & & [5]\,, \end{array}$$

where $x \in V^*$, $1_x \in \Xi_1$, $\rho \in \Xi_2^*$, $y = a'_1a'_2\ldots a'_m$, $z = a_1a_2\ldots a_m$, $m > 0$, $a_1, a_2, \ldots, a_m \in T_\triangle$, $\sigma \in \Xi_4^*$, $w \in T^*$, $S \Rightarrow_G^* w\ [\![\Upsilon]\!]$, $S\langle z\rangle = \mathrm{lbr}(\Upsilon)$.

Formal Proof We estabilish Lemma 3.1 by Claims 3.2 through Claim 3.4 stated below.

Claim 3.2. *$G_\triangle$ generates every $w_\triangle \in L(G_\triangle)$ in the following way:*

$$\begin{array}{lll} S'' & \Rightarrow_{G_\triangle} & \$g(x) \quad [1_x] \\ & \Rightarrow^*_{G_\triangle} & w_2 \quad [\rho] \\ & \Rightarrow_{G_\triangle} & w_3 \quad [3] \\ & \Rightarrow^*_{G_\triangle} & w_4 \quad [\sigma] \\ & \Rightarrow_{G_\triangle} & w_\triangle \quad [5]\,, \end{array}$$

where $w_2, w_3, w_4, w_\triangle \in V_\triangle$, $x \in V^*$, $1_x \in \Xi_1$, $\rho \in \Xi_2^*$, $\sigma \in \Xi_4^*$.

Proof. First, let us make these two observations:

- Since the only productions with S'' on their left-hand sides are the productions introduced in (1), the derivation must surely start with a derivation step made by one of these productions. Furthermore, $S'' \notin \mathrm{rhs}(p_\triangle)$, for any $p_\triangle \in P_\triangle$, so these productions cannot be used during the rest of the derivation. The derivation ends by applying the production 5, because it is the only production without nonterminals on its right-hand side. Thus, $S \Rightarrow^+_{G_\triangle} w_\triangle$ can be expressed as

$$\begin{array}{lll} S'' & \Rightarrow_{G_\triangle} & \$g(x) \quad [1_x] \\ & \Rightarrow^*_{G_\triangle} & w_4 \\ & \Rightarrow_{G_\triangle} & w_\triangle \quad [5]\,. \end{array}$$

- For each $1_x \in \Xi_1$, $2_p \in \Xi_2$, $4_a \in \Xi_4$, the constructed productions satisfy

$$\begin{array}{rll} 1 = & |\mathrm{rhs}(1_x)|_{\{\$\}} & = |\mathrm{lhs}(2_p)|_{\{\$\}} \\ = & |\mathrm{rhs}(2_p)|_{\{\$\}} & = |\mathrm{lhs}(3)|_{\{\$\}} \\ = & |\mathrm{rhs}(3)|_{\{@\}} & = |\mathrm{lhs}(4_a)|_{\{@\}} \\ = & |\mathrm{rhs}(4_a)|_{\{@\}} & = |\mathrm{lhs}(5)|_{\{@\}} \end{array}$$

  and

$$\begin{array}{rll} 0 = & |\mathrm{rhs}(1_x)|_{\{@\}} & = |\mathrm{lhs}(2_p)|_{\{@\}} \\ = & |\mathrm{rhs}(2_p)|_{\{@\}} & = |\mathrm{lhs}(3)|_{\{@\}} \\ = & |\mathrm{rhs}(3)|_{\{\$\}} & = |\mathrm{lhs}(4_a)|_{\{\$\}} \\ = & |\mathrm{rhs}(4_a)|_{\{\$\}} & = |\mathrm{lhs}(5)|_{\{\$\}} \end{array}$$

Based on these observations, notice that $G_\triangle$ generates every sentence in the way described in Claim 3.2. □

Claim 3.3. *Consider the derivation from Claim 3.2. In its beginning,*

$$\begin{array}{lll} S'' & \Rightarrow_{G_\triangle} & \$g(x) \quad [1_x] \\ & \Rightarrow^*_{G_\triangle} & w_2 \quad [\rho] \\ & \Rightarrow_{G_\triangle} & w_3 \quad [3]\,, \end{array}$$

every sentential form s in $\$g(x) \Rightarrow^*_{G_\triangle} w_2$ satisfies

$$s \in \{\$\}\{\Phi' \cup \Phi_\triangle\}^+,$$
$$w_2 \in \{@\}\{\Phi'\}^+.$$

Proof. By the definition of g, $g(x) \in \{\Phi' \cup \Phi_\triangle\}^+$. Since productions from Ξ_2 rewrite symbols from $\Phi_\triangle$, every sentential form s in $\$g(x) \Rightarrow^*_{G_\triangle} w_2$ satisfies

$$s \in \{\$\}(\Phi' \cup \Phi_\triangle)^+.$$

Only productions $2_p \in \Xi_2$ satisfy

$$\mathrm{alph}(\mathrm{lhs}(2_p)) \cap \Xi_\triangle \neq \emptyset.$$

Therefore, to generate $w_\triangle \in T_\triangle$, productions labeled with 2_p have to be applied until

$$s \in \{\$\}\Phi'^+.$$

Furthermore, observe that every production from Ξ_2 simulates the construction of production tree in left-bracketted representation; more precisely, every production $2_p \in \Xi_2$ has the form,

$$p : (\$, A_{1_\triangle}, \ldots, A_{n_\triangle}) \to (\$, A'_1\langle' y_1\rangle', \ldots, A'_n\langle' y_n\rangle'),$$

and satisfies that for every $i : 1 \leq i \leq n$,

$$y_i \in \Phi_\triangle \cup \Phi'_T.$$

Finally, the production 3 is used, so

$$w_3 \in \{@\}\Phi'^+,$$

and the claim holds. □

Claim 3.4. *In*

$$\begin{array}{lll} w_3 & \Rightarrow^*_{G_\triangle} & w_4 \quad [\sigma] \\ & \Rightarrow_{G_\triangle} & w_\triangle \quad [5]\,, \end{array}$$

of the derivation from Claim 3.2, every sentential form s in $w_3 \Rightarrow^+_{G_\triangle} w_4$ can be expressed as

$$s \in T^*\{@\}T^*_\triangle\{\#\}\Phi'^*,$$

and

$$w_4 \in T^*\{@\}T^*_\triangle\{\#\}.$$

In greater detail,

$$\begin{array}{lrll} w_3 & = & @ & \#a'_1a'_2a'_3\dots a'_m & \\ & \Rightarrow_{G_\triangle} & f'(a_1)@ & a_1\#a'_2a'_3\dots a'_m & [4_{a_1}] \\ & \Rightarrow_{G_\triangle} & f'(a_1a_2)@ & a_1a_2\#a'_3\dots a'_m & [4_{a_2}] \\ & \Rightarrow^{m-2}_{G_\triangle} & f'(a_1a_2\dots a_m)@ & a_1a_2a_3\dots a_m\# & [\sigma'] \\ & \Rightarrow_{G_\triangle} & f'(a_1a_2\dots a_m)S & \langle a_1a_2a_3\dots a_m\rangle & [5], \end{array}$$

where $a'_i \in \Phi'$, $a_i \in T_\triangle$, $\sigma' \in \Xi_4^*$ *and* f' *is a function* $T_\triangle \to T$, *such that*

$$f'(a) = \begin{cases} a & \text{if } a \in T \\ \varepsilon & \text{otherwise} \end{cases}$$

Proof. In every derivation step of $w_3 \Rightarrow^*_{G_\triangle} w_4$, the first symbol $a' \in \Phi'$ following nonterminal $\#$ is replaced by $a \in T_\triangle$ and positions of a and $\#$ are swapped. Additionally, if $a \in T$, then $@$ is also changed to $a@$, therefore constructing a sentence of G by preorder traversal of derivation tree. More precisely, every sentential form s in $w_3 \Rightarrow^*_{G_\triangle} w_4$ has the form

$$\begin{array}{l} s \in T^*\{@\}T^*_\triangle\{\#\}\Phi'^*, \\ w_4 \in T^*\{@\}T^*_\triangle\{\#\}. \end{array}$$

Note that if $G_\triangle$ uses a production $p \in \Xi_4$ to replace any other nonterminal than the first one following the $\#$, there is no way to replace the skipped nonterminal by some terminal symbol, so the derivation cannot lead to any sentence. Therefore, all nonterminals from Ξ' are processed in the order they were generated, otherwise $\mathrm{alph}(w_4) \cap \Xi' \neq \emptyset$ and the derivation cannot be successful.

Left-bracketted representation produces the pre-order tree traversal. Since we "copy" only leaf nodes, which contain marked terminal symbols, to the front of the sentential form and ignore other symbols, we construct precisely the sentence of grammar G in front of the string representation of its derivation tree.

Finally, after every occurence of $a' \in \Phi'$ is removed, $G_\triangle$ applies the production 5 to change the remaining nonterminals ($@$ and $\#$) to the missing root of derivation tree and to the top-level brackets. □

From Claims 3.2 through 3.4, it follows that grammar $G_\triangle$ produced by Algorithm 1 generates sentences of input grammar followed by their derivation trees in left-bracketed representation. □

By Algorithm 1, we can convert any scattered context grammar G to *propagating* scattered context grammar $G_\triangle \in G^{\blacktriangle}$, that is, to the grammar generating setences of G followed by the left-bracketted representation of their derivation trees.

Theorem 3.5. *For every SCG* $G = (V, T, P, S)$, *there exists a* propagating *SCG* $G_\triangle$ *such that* $G_\triangle \in G^{\blacktriangle}$.

Proof. Since Algorithm 1 is correct, it follows that we can construct a propagating SCG $G_{\triangle} \in G^{\blacktriangle}$ for any SCG G (see Lemma 3.1). □

4. Concluding Remarks

Let us conclude with some final notes.

First, note that similar results can be estabilished for propagating scattered context languages generating sentences *preceded* by their derivation trees.

Second, the algorithm could be altered to generate the derivation tree with different terminal symbols, which are not present in the original language, e.g., instead of $aS\langle a\rangle$, we could generate $aS\langle a'\rangle$. This approach would allow us to characterize the original grammar's language by using operation of right quotient with respect to language $(\{\langle, \rangle, \varepsilon\} \cup (V - T) \cup \{a' : a \in T\})^*$.

Finally, there remains a question whether the presented transformation of scattered context grammars is possible in terms of other (parallel) rewriting mechanism, possibly producing the rewriting mechanism that is known to define the language family properly contained in the family of languages generated by the original mechanism.

Acknowledgment

This work was supported by the Research Plan No. MSM, 0021630528 – Security-Oriented Research in Information Technology.

References

1. A. V. Aho, J. D. Ullman: *The Theory of Parsing*, Translation, and Compiling. Prentice Hall (1972).
2. M. J. Wolfe: *High performance compilers for parallel computing*. Addison-Wesley (1996).
3. G. R. Gao, L. Pollock, J. Cavazos, X. Li (Eds.): *Languages and Compilers for Parallel Computing*. Springer (2009).
4. J. N. Amaral (Ed.): *Languages and Compilers for Parallel Computing*. Springer (2008).
5. S. Pande, D. P. Agrawal (Eds.): *Compiler Optimizations for Scalable Parallel Systems: Languages, Compilation Techniques, and Run Time Systems (Lecture Notes in Computer Science*). Springer (2001).
6. S. Greibach, J. Hopcroft: *Scattered context grammars*. Journal of Computer and System Sciences **3** (1969) 233–247.
7. A. Meduna, J. Techet: *Scattered Context Grammars and Their Applications*. WIT Press (2009) 137–155.

8. A. Meduna, J. Techet: *Generation of sentences with their parses: the case of propagating scattered context grammars*. Acta Cybernetica **17** (2005) 11–20.

9. A. Meduna: *Automata and Languages: Theory and Applications*. Springer-Verlag (2000).

10. A. V. Aho, R. Sethi, J. D. Ullman: *Compilers: Principles, Techniques and Tools*. Addison-Wesley (1988).

11. A. Meduna, J. Techet: *Canonical scattered context generators of sentences with their parses*. Theoretical Computer Science **389** (2007) 73-âĂŞ81.