Derivation in Scattered Context Grammar via Lazy Function Evaluation

Ota Jirák[1] and Dušan Kolář[2]

[1] FIT BUT, Brno, Czech Republic,
`ijirak@fit.vutbr.cz`,
WWW home page: `http://www.fit.vutbr.cz/~ijirak/`
[2] FIT BUT, Brno, Czech Republic,
`kolar@fit.vutbr.cz`,
WWW home page: `http://www.fit.vutbr.cz/~kolar/`

Abstract. This paper discusses scattered context grammars (SCG) and considers the application of scattered context grammar production rules. We use function that represents single derivation step over the given sentential form. Moreover, we define this function in such a way, so that it represents the delayed execution of scattered context grammar production rules using the same principles as a lazy evaluation in functional programming. Finally, we prove equivalence of the usual and the delayed execution of SCG production rules.

1 Introduction

Family of languages that is described by scattered context grammars is very important due to their generative power. This paper discusses usage of functions over sentential forms to simulate derivation steps. Function representing delayed execution of scattered context grammar rules is introduced. Next, we discuss lazy evaluation of this recursively defined function.

The main goal of this article is to prove that this function is equivalent to commonly known derivation step.

The proof is divided into several parts:

- we use example to demonstrate that sentential form completely processed and the same sentential form partially processed are equivalent when processed by the delayed execution function,
- we demonstrate that introduced function can handle any SCG rules on any sentential form,
- we demonstrate that application of nested calling of delayed function is equivalent with nested calling of regular derivation function,
- we demonstrate that application of nested delayed derivation, lazy evaluated, is equivalent to nested regular derivation function.

Mathematical and Engineering Methods in Computer Science (MEMICS), Znojmo, Czech Republic, 2009.
Petr Hliněný, Vashek Matyáš, Tomáš Vojnar (Eds.)
Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/DROPS.MEMICS.2009.2351

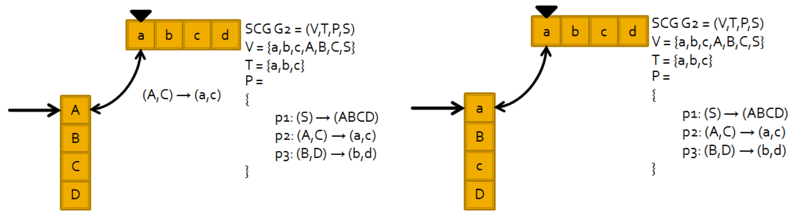
2 Motivation

We have several principles for implementation of compilers for SCG: deep pushdown [8] (with a certain limitation), and regulated pushdown automata [4–6] (RPDA).

The first approach uses nonterminal expansion not only on the pushdown top, but even deeper. Implementation of this pushdown is inefficient (linked list).

The RPDA usually uses auxiliary pushdown to restore the main pushdown. This data shuffling from one pushdown to another is also inefficient.

We are interested in deterministic compilers. Thus, we have to use leftmost derivation principles and LL SCGs to make the parser work deterministically [5, 6]. We need to work only with the pushdown top.



(a) Before Applying the Rule (b) After Applying the Rule

Fig. 1. Normal Derivation Using RPDA

Productions of SCG are defined as an n-tupple of CFG productions (see Section 3 - Preliminaries and Definitions). This is origin of basic idea. We would like to use parsing principles from CFG parsers to parse context-free parts of SCG productions in the right time.

We rely on principles of deterministic context-free parsers. We use leftmost derivation. We have to use some kind of LL/LR grammars to choose productions in a deterministic way.

We use one CFG production of particular SCG production. The others are delayed and used in the right time. The unprocessed part of sentential form is marked to be processed later with this delayed part of SCG production.

We can see difference between regular derivation and delayed derivation on Figures 1 and 2. Applying one production in regular way means rewrite several nonterminals in one step. One regular derivation step is shown on Figure 1(a) and 1(b). Nonterminals A and C are rewritten in one derivation step.

Example of delayed execution is shown on Figure 2. Production p2 should be used in Figure 2(a). We apply $A \to a$ and we delay $C \to c$. Then, pop is applied on symbol a on the top of pushdown and a under the reading head.

Production p3 should be used in Figure 2(b). We apply $B \to b$ and we delay $D \to d$. Then, pop is applied on symbol b on the top of pushdown and b under the reading head.

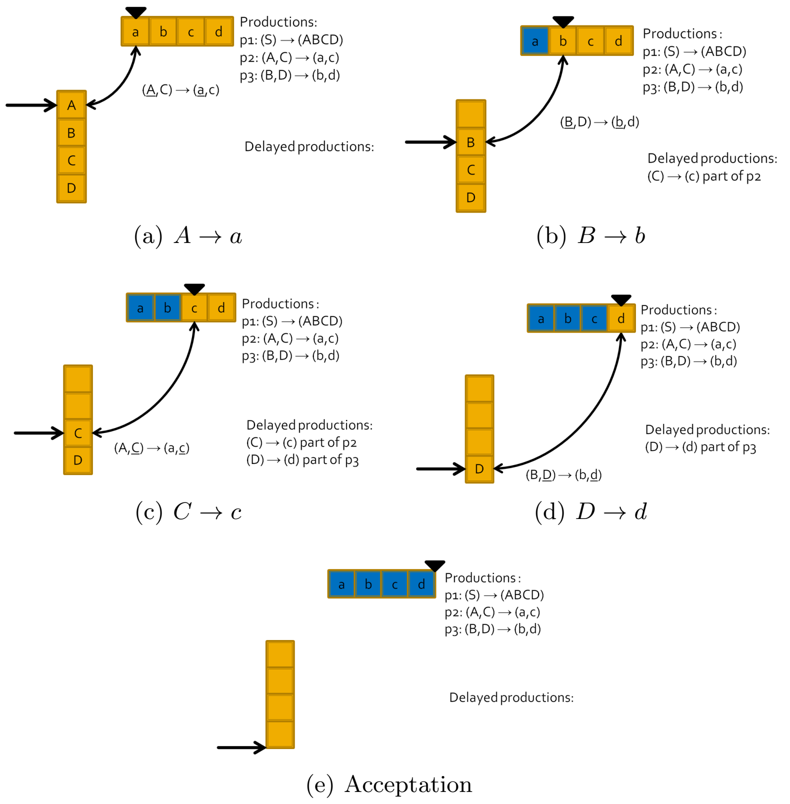


Fig. 2. Delayed Derivation

In Figure 2(c), there is no other option than to use delayed production $C \to c$. If there are more delayed productions that could be used then we use the oldest one. So we applied delayed production $C \to c$ and then removed it from delayed production list. Then, pop is applied on symbol c on the top of pushdown and c under the reading head.

In Figure 2(d), there we used delayed production $D \to d$ and then removed it from the delayed production list. Then, pop is applied on symbol d on the top of pushdown and d under the reading head.

In Figure 2(e), there we can see accepted sentential form by given scattered context grammar. Sentence is accepted with empty pushdown and zero delayed productions.

3 Preliminaries and Definitions

It is expected that a reader is familiar with formal language theory [7].

Let V^* be a free monoid over alphabet V, $w \in V^*$, w is called string of symbols from V, $|w|$ denotes the length of w. Let ε be an empty string, $|\varepsilon| = 0$.

A context-free grammar (CFG, see [7]) is a quadruple $G = (V, T, P, S)$, where V is a finite set of symbols, $T \subset V$ is a terminal alphabet, $S \in V \backslash T$ is the starting nonterminal, and P is a finite set of rules of the form $A \rightarrow w$, where $A \in V \backslash T$ and $w \in V^*$.

Now, we introduce definition of sentential form indexing. Besides common concepts from the formal languages, we define for a string, X, X[n] and X[n:] to denote one symbol from string and a substring of the string.

Definition 1. *Let $X = a_1 a_2 \dots a_n, a_i \in V, i \in \{1, \dots, n\}, n \in \mathbb{N}$.*

$$\begin{aligned} X[k] &= a_k, k \in \mathbb{N}, 1 \le k \le n, \\ X[k:] &= a_k \dots a_n, k \in \mathbb{N}, 1 \le k \le n, \\ X[k] = X[k:] &= \varepsilon, k \in \mathbb{N}, k > n. \end{aligned}$$

A scattered context grammar (see [1]) is a quadruple $G = (V, T, P, S)$, where V is a finite set of symbols, $T \subset V$ is a terminal alphabet, $S \in V \backslash T$ is the starting nonterminal, and P is a finite set of production rules of the form $(A_1, A_2, \dots, A_n) \rightarrow (w_1, w_2, \dots, w_n)$, for some $n \ge 1$, where $A_i \in V \backslash T$ and $w_i \in V^*$. Let $u = x_1 A_1 x_2 A_2 \dots x_n A_n x_{n+1}$, $v = x_1 w_1 x_2 w_2 \dots x_n w_n x_{n+1}$, $x_i \in V^*, A_i \in V \backslash T, 1 \le i \le n$, for some $n \ge 1$. $u \Rightarrow v, x_1 A_1 x_2 A_2 \dots x_n A_n x_{n+1} \Rightarrow x_1 w_1 x_2 w_2 \dots x_n w_n x_{n+1}$ is a derivation. If $x_1 \in T^*, x_i \in (V \backslash \{A_i\})^*$, it is a leftmost derivation. Let $u_0, \dots, u_n \in V^*, p_1, \dots, p_n \in P$, $u_0 \Rightarrow_1 u_1 \Rightarrow_2 \cdots \Rightarrow_i u_i \Rightarrow \cdots \Rightarrow_n u_n$ is a sequence of leftmost derivations. Number of the derivation step, i, is a position in the sequence of derivations. $\Rightarrow_i$ represents usage of p_i — production of the i-th derivation step.

Definition 2. *Let $X = x_1 A_1 x_2 A_2 \dots x_n A_n x_{n+1}$ be a sentential form, $A_i \in V \backslash T, x_i \in (V \backslash \{A_i\})^*, x_{n+1} \in V^*$, for some $n \ge 1$, j is a number of derivation step and $p_j : (A_1, A_2, \dots, A_n) \rightarrow (w_1, w_2, \dots, w_n) \in P$ is an SCG rule used in the j-th derivation step. Function $h_j(X)$ stands for leftmost application of the SCG rule used in the j-th derivation step that is:*

$$\begin{aligned} h_j(X) &= h_j(x_1 A_1 x_2 A_2 \dots x_n A_n x_{n+1}) \\ &= x_1 w_1 x_2 w_2 \dots x_n w_n x_{n+1} \end{aligned} \qquad (1)$$

Note 1. We say, this is a regular derivation step.

Definition 3. *Let $X = x_1 A_1 x_2 A_2 \dots x_n A_n x_{n+1}$ be a sentential form, $A_i \in V \backslash T, x_i \in (V \backslash \{A_i\})^*, x_{n+1} \in V^*$, for some $n \ge 1$, j is a number of derivation step and $p_j : (A_1, A_2, \dots, A_n) \rightarrow (w_1, w_2, \dots, w_n) \in P$ is an SCG rule used in the j-th derivation step, $m \in \{1, \dots, n+1\}$.*

$$g_j(m, X) = \begin{cases} X[1] g_j(m, X[2:]) & \text{for } m \le n, X[1] \ne A_m, \\ w_m g_j(m+1, X[2:]) & \text{for } m \le n, X[1] = A_m, \\ \varepsilon & \text{for } m > n, |X| = 0, \\ X[1] g_j(m, X[2:]) & \text{for } m > n, |X| > 1, \\ X[1] & \text{for } m > n, |X| = 1. \end{cases} \qquad (2)$$

Note 2. We say, this is a delayed derivation step.

Let x be some sentential form. $x^{'}$ is sentential form processed by function g or h — added apostroph to x.

Let n_b^x is an index into SCG rule, which is used in the b-th derivation step. Symbol x is a counter of symbols processed by function, which is used in b-th derivation step.

Example 1. Now, we demonstrate that sentential form completely processed and the same sentential form partially processed are equivalent when processed by the delayed execution function.
$g(n_1^0, x_1x_2 \dots x_ix_{i+1} \dots x_k) = x_1^{'} \dots x_i^{'} g(n_1^i, x_{i+1} \dots x_k)$, $i \in \{1, \dots, k\}$, $x_i \in V$ for some $k \in \mathbb{N}$

$$g(n_1^0, x_1x_2 \dots x_k) = \Big|_{\text{based on definition 3}} x_1^{'} g(n_1^1, x_2 \dots x_k) \tag{3}$$

$$= \Big|_{\text{based on definition 3}} x_1^{'} x_2^{'} g(n_1^2, x_3 \dots x_k) \tag{4}$$

$$\dots$$

$$= \Big|_{\text{based on definition 3}} x_1^{'} x_2{'} \dots x_{k-1}^{'} g(n_1^{k-1}, x_k) \tag{5}$$

$$= \Big|_{\text{based on definition 3}} x_1^{'} x_2^{'} \dots x_k^{'} \tag{6}$$

$\Rightarrow g(n_1^0, x_1x_2 \dots x_ix_{i+1} \dots x_k) = x_1^{'} \dots x_i^{'} g(n_1^i, x_{i+1} \dots x_k)$ for some $i \in \{1, \dots, k\}$

Lazy evaluation [9] based on call-by-need strategy is a scheduling policy that does not evaluate an expression (or invoke a procedure) until the results of the evaluation are needed. Lazy evaluation may avoid some unnecessary work. It may allows a computation to terminate in some situations that otherwise would not.

Lazy evaluation is often used in functional and logic programming, e.g. Haskell[2].

Lazy evaluation of delayed derivation is application of delayed derivation in a lazy way. It means that the leftmost symbols of sentential form are processed by several derivation steps while the rest of the sentential form is still unchanged. In other words, we make recursive step (see definition 2) only on the leftmost outermost symbol being unprocessed by particular function for delayed derivation.

4 Basic Idea

Each scattered context grammar derivation step can be described as an application of some function over given sentential form. In our case, this function represents leftmost application of an SCG rule.

We go through the sentential form and test each symbol. If the tested symbol is a particular nonterminal from the appropriate (context-free) part of the SCG rule we replace it with the right-hand side of the part of the SCG rule. And so on until the whole string is processed.

5 Results

At first, we show that delayed function can process sentential forms of any length and any number of context-free parts of an SCG rule.

Lemma 1. *$g(1, uA_1\alpha_1 \dots \alpha_k A_{k+1}\alpha_{k+1}) = h(uA_1\alpha_1 \dots \alpha_k A_{k+1}\alpha_{k+1}), i \in \{1, \dots, k\}$ for some $k \in \mathbb{N}, u \in T^*, x_i \in (V \backslash \{A_{i+1}\})^*, (A_1, \dots, A_{k+1}) \rightarrow (\beta_1, \dots, \beta_{k+1}) \in P$*

Proof. k — number of parts of SCG rule.
Basis. $g(1, uA_1\alpha_1) = h(uA_1\alpha_1), k = 1$

$$\begin{aligned} g(1, uA_1\alpha_1) &=\Big|_{\text{based on definition 3}} ug(1, A_1\alpha_1) && (7)\\ &=\Big|_{\text{based on definition 3}} u\beta_1 g(2, \alpha_1) && (8)\\ &=\Big|_{\text{based on definition 3}} u\beta_1\alpha_1 && (9)\\ &=\Big|_{\text{based on definition 2}} h(uA_1\alpha_1) && (10)\end{aligned}$$

$\Rightarrow g(1, uA_1\alpha_1) = h(uA_1\alpha_1)$ □

$g(1, uA_1\alpha_1A_2\alpha_2) = h(uA_1\alpha_1A_2\alpha_2)$, k=2

$$\begin{aligned} g(1, uA_1\alpha_1A_2\alpha_2) &=\Big|_{\text{based on definition 3}} ug(1, A_1\alpha_1A_2\alpha_2) && (11)\\ &=\Big|_{\text{based on definition 3}} u\beta_1 g(2, \alpha_1A_2\alpha_2) && (12)\\ &=\Big|_{\text{based on definition 3}} u\beta_1\alpha_1 g(2, A_2\alpha_2) && (13)\\ &=\Big|_{\text{based on definition 3}} u\beta_1\alpha_1\beta_2 g(3, \alpha_2) && (14)\\ &=\Big|_{\text{based on definition 3}} u\beta_1\alpha_1\beta_2\alpha_2 && (15)\\ &=\Big|_{\text{based on definition 2}} h(uA_1\alpha_1A_2\alpha_2) && (16)\end{aligned}$$

$\Rightarrow g(1, uA_1\alpha_1A_2\alpha_2) = h(uA_1\alpha_1A_2\alpha_2)$ □

Induction Hypothesis. We suppose that the statement holds for all $k, 1 \leq k \leq n$, for some $n \geq 1$.

Induction Step. $g(1, uA_1\alpha_1 \dots A_{k+1}\alpha_{k+1}) = h(uA_1\alpha_1 \dots A_{k+1}\alpha_{k+1}), n = k+1$

$$g(1, uA_1\alpha_1 \dots A_{k+1}\alpha_{k+1}) \tag{17}$$

$$=\Big|_{\gamma=\alpha_k A_{k+1}\alpha_{k+1}} g(1, uA_1\alpha_1 \dots A_{k-1}\alpha_{k-1}A_k\gamma) \tag{18}$$

$$=\Big|_{\text{based on induction hypothesis and equation 14}} u\beta_1\alpha_1 \dots \beta_{k-1}\alpha_{k-1}\beta_k g(k+1, \gamma) \tag{19}$$

$$=\Big|_{\gamma=\alpha_k A_{k+1}\alpha_{k+1}} u\beta_1\alpha_1 \dots \beta_{k-1}\alpha_{k-1}\beta_k g(k+1, \alpha_k A_{k+1}\alpha_{k+1}) \tag{20}$$

$$=\Big|_{\text{based on definition 3}} u\beta_1\alpha_1 \dots \beta_{k-1}\alpha_{k-1}\beta_k\alpha_k g(k+1, A_{k+1}\alpha_{k+1}) \tag{21}$$

$$=\Big|_{\text{based on definition 3}} u\beta_1\alpha_1 \dots \beta_{k-1}\alpha_{k-1}\beta_k\alpha_k\beta_{k+1} g(k+2, \alpha_{k+1}) \tag{22}$$

$$=\Big|_{\text{based on definition 3}} u\beta_1\alpha_1 \dots \beta_{k-1}\alpha_{k-1}\beta_k\alpha_k\beta_{k+1}\alpha_{k+1} \tag{23}$$

$$=\Big|_{\text{based on induction hypothesis and definition 2}} h(uA_1\alpha_1 \dots \alpha_{k-1}A_k\alpha_k A_{k+1}\alpha_{k+1}) \tag{24}$$

Therefore, $g(1, uA_1\alpha_1 \dots A_{k+1}\alpha_{k+1}) = h(uA_1\alpha_1 \dots A_{k+1}\alpha_{k+1})$, so the lemma holds.

Next, we show that we can use any number of rules.

Lemma 2. *$g_k(1, \dots g_2(1, g_1(1, \alpha))) = h_k(\dots h_2(h_1(\alpha))), \alpha \in V^*$ for some $k \in \mathbb{N}$*

Proof. k — number of nested functions.
Basis. $g_1(1, \alpha) = h_1(\alpha), k = 1$. Proof in Lemma 1.
$g_2(1, g_1(1, \alpha)) = h_2(h_1(\alpha)), k = 2$

$$g_2(1, g_1(1, \alpha)) =\Big|_{\text{apply lemma1 on } g_1} g_2(1, h_1(\alpha)) \tag{25}$$

$$=\Big|_{\text{apply lemma1 on } g_2} h_2(h_1(\alpha)) \tag{26}$$

$\Rightarrow g_2(1, g_1(1, \alpha)) = h_2(h_1(\alpha))$ □
Induction Hypothesis. We suppose that the statement holds for all $k, 1 \leq k \leq n$ for some $n \geq 1$.
Induction Step. $g_{k+1}(1, g_k(1, \dots g_1(1, \alpha))) = h_{k+1}(h_k(\dots h_1(\alpha)))$

$$g_{k+1}(1, g_k(1, \dots g_1(1, \alpha))) =\Big|_{\text{based on induction and } \beta = h_k(h_{k-1} \dots h_1(\alpha))} g_{k+1}(1, \beta) \tag{27}$$

$$=\Big|_{\text{apply lemma1 on } h_{k+1}} h_{k+1}(\beta) \tag{28}$$

$$=\Big|_{\text{based on induction and } \beta = h_k(h_{k-1} \dots h_1(\alpha))} h_{k+1}(h_k(\dots g_1(\alpha)) \tag{29}$$

Therefore, $g_{k+1}(1, g_k(1, \dots g_1(1, \alpha))) = h_{k+1}(h_k(\dots h_1(\alpha_1)))$, so the lemma holds.

The following Lemma 3 shows that delayed derivation steps (lazy evaluated) return the same values as regular derivation steps.

Lemma 3. *Lazy evaluation of* $g_{k+1}(1, \dots g_1(1, \omega_1 \dots \omega_m)) = h_{k+1}(\dots h_1(\omega_1 \dots \omega_m))$, $\omega_i \in V, i \in \{1, \dots, m\}$ *for some* $m \geq 1$ *and* $k \geq 1$.

Proof. Basis. For $k = 1$ holds from definition 3. For $k = 2$, lazy evaluation of $g_2(n_2^0, g_1(n_1^0, \omega_1 \dots \omega_m)) = h_2(h_1(\omega_1 \dots \omega_m)), n_1^0 = n_2^0 = 1$

$$g_2(n_2^0, g_1(n_1^0, \omega_1 \dots \omega_m)) \tag{30}$$

$$=\Big|_{\text{apply one step from definition3 on } g_1} g_2(n_2^0, \omega_1^{'} g_1(n_1^1, \omega_2 \dots \omega_m)) \tag{31}$$

$$=\Big|_{\text{apply one step from definition3 on } g_2} \omega_1^{''} g_2(n_2^1, g_1(n_1^1, \omega_2 \dots \omega_m)) \tag{32}$$

$\dots$

$$=\Big| \omega_1^{''} \dots \omega_{m-1}^{''} g_2(n_2^{m-1}, g_1(n_1^{m-1}, \omega_m)) \tag{33}$$

$$=\Big|_{\text{apply one step from definition3 on } g_1} \omega_1^{''} \dots \omega_{m-1}^{''} g_2(n_2^{m-1}, \omega_m^{'}) \tag{34}$$

$$=\Big|_{\text{apply one step from definition3 on } g_2} \omega_1^{''} \dots \omega_m^{''} \tag{35}$$

$$=\Big|_{\text{based on definition 2}} h_2(h_1(\omega_1 \dots \omega_m)) \tag{36}$$

$\Rightarrow$ lazy evaluation of $g_2(n_2^0, g_1(n_1^0, \omega_1 \dots \omega_k)) = h_2(h_1(\omega_1 \dots \omega_k))$ □

Induction Hypothesis. Suppose that the statement holds for all $k, 1 \leq k \leq n$, for some $n \in \mathbb{N}$.

Induction Step. $g_{k+1}(n_{k+1}^0, \dots g_1(n_1^0, \omega_1 \dots \omega_m)) = h_{k+1}(\dots h_1(\omega_1 \dots \omega_m)), \omega_i \in V, i \in \{1, \dots, m\}$ for some $m \in \mathbb{N}$ and $n_j^0 = 1, j \in \{1, \dots, k+1\}$. We can say without loss of generality that it returns one processed symbol in each step. To simplify the proof, we write $g_k(n_k^0, \omega_1 \dots \omega_m)$ instead of $g_k(n_k^0, \dots g_1(n_1^0, \omega_1 \dots \omega_m))$.

$$g_{k+1}(n_{k+1}^0, g_k(n_k^0, \omega_1 \omega_2 \dots \omega_m)) \tag{37}$$

$$=\Big|_{\text{apply one step from definition3 on } g_k} g_{k+1}(n_{k+1}^0, \omega_1^{'} g_k(n_k^1, \omega_2 \dots \omega_m)) \tag{38}$$

$$=\Big|_{\text{apply one step from definition3 on } g_{k+1}} \omega_1^{''} g_{k+1}(n_{k+1}^1, g_k(n_k^1, \omega_2 \dots \omega_m)) \tag{39}$$

$\dots$

$$=\omega_1^{''} \dots \omega_{m-1}^{''} g_{k+1}(n_{k+1}^{m-1}, g_j(n_j^{m-1}, \omega_m)) \tag{40}$$

$$=\Big|_{\text{apply one step from definition3 on } g_k} \omega_1^{''} \dots \omega_{m-1}^{''} g_{k+1}(n_{k+1}^{m-1}, \omega_m) \tag{41}$$

$$=\Big|_{\text{apply one step from definition3 on } g_{k+1}} \omega_1^{''} \dots \omega_m^{''} \tag{42}$$

$$=\Big|_{\text{based on definition 2}} h_{k+1}(h_k(\omega_1 \dots \omega_m)) \tag{43}$$

Therefore, lazy evaluation of $g_{k+1}(n_{k+1}^0, \dots g_1(n_1^0, \omega_1 \omega_2 \dots \omega_m)) = h_{k+1}(\dots h_1(\omega_1 \dots \omega_m))$, so the lemma holds.

The following theorem and its proof, which represents the main result of this paper, demonstrates that delayed execution of SCG rules is equivalent with SCG derivation.

Theorem 1. *Lazy evaluation of* $g_m(1, g_{m-1}(1, \dots g_1(1, \omega_1 \dots \omega_j))) = h_m(h_{m-1}(\dots h_1(\omega_1 \dots \omega_j))) = w \equiv \omega_1 \dots \omega_j \Rightarrow^m w, \omega_i \in V, i \in \{1, \dots, j\}$ *for some* $j, m \in \mathbb{N}$.

Proof. $\alpha \in V^*$

1. Using function g or h is equivalent for sentential forms of any length and for any SCG rules. It has been proved in Lemma 1.
2. Using any number of functions, $g_n(\dots g_1(\alpha))$, is equivalent to $h_n(\dots h_1(\alpha))$, for any n. It has been proved in Lemma 2.
3. Lazy evaluated g returns the same result as h. It has been proved in Lemma 3.
4. $h_m(\dots h_1(\omega_1 \dots \omega_j)) = w \equiv \omega_1 \dots \omega_j \Rightarrow^m w$ holds by definition 2.

From 1, 2, 3, and 4 follows:
$g_m(1, g_{m-1}(1, \dots g_1(1, \omega_1 \dots \omega_j))) = h_m(h_{m-1}(\dots h_1(\omega_1 \dots \omega_j))) = w \equiv \omega_1 \dots \omega_j \Rightarrow^m w$. □

6 Conclusion

In this paper, we have shown usage of functions instead of derivation steps. Lazy evaluation of delayed execution of scattered context grammar rules has been presented.

The main result of this article is equivalence of lazy evaluated delayed executed function and the function representing regular leftmost derivation over a string. This approach allows us to work only with the pushdown top during compilation time.

7 Open Questions and Future Work

Scattered context grammar was introduced by Greibach and Hopcroft in 1969 (see [1]). Since these days, several implementation methods of compilers for scattered context grammars has been discovered [4–6, 8].

Next research will lead to study compilers that use delayed execution of SCG rules and to compare with compilers using regulated pushdown automata.

Intuitively, it should be faster, because we expand only topmost symbol on the stack. Basic principle of using delayed executed SCG rules in compilers is in [3].

Nevertheless, exploitation of lazy evaluation in implementation of an SCG parser traditional way [6] may be an option. That is why; we want to compare both approaches.

The work has been supported by Research Plan No. MSM 0021630528 - Security-Oriented Research in Information Technology and by the Czech Ministry of Education, Youth and Sports grant MŠMT 2C06008 "Virtual Laboratory of Microprocessor Technology Application".

References

1. Greibach, S., Hopcroft, J.: Scattered context grammars. J. Comput. Syst. Sci. 3, 233-247(1969)
2. Haskell, http://www.haskell.org/haskellwiki/Haskell/Lazy_evaluation, cited Sep. 2009
3. Jirák, O.: Delayed Execution of Scattered Context Grammar Rules, In: Proceedings of the 15th Conference and Competition STUDENT EEICT 2009 Volume 4, Brno, CZ, FIT VUT, 2009, p. 405-409, ISBN 978-80-214-3870-5
4. Kolář, D., Meduna, A.: Regulated Pushdown Automata, In: Acta Cybernetica, Vol. 2000, No. 4, US, p. 653–664, ISSN 0324-721X
5. Kolář, D.:Pushdown Automata: Another Extensions and Transformations, Brno, CZ, FIT BUT, 2005, p. 76
6. Kolář, D.: Scattered Context Grammar Parsers, In: Proceedings of the 14th International Congress of Cybernetics and Systems of WOSC, Wroclaw, PL,PWR WROC, 2008, p. 491–500, ISBN 978-83-7493-400-8
7. Meduna, A.: Automata and Languages: Theory and Applications. Springer-Verlag, London, 2000
8. Meduna, A.: Deep Pushdown Automata, In: Acta Informatica, Vol. 2006, No. 98, DE, p. 114–124, ISSN 0001-5903
9. University of Florida, http://www.cise.ufl.edu/research/ParallelPatterns/glossary.htm, cit. Sep 07 2009