TABLE-DRIVEN PARSING OF SCATTERED CONTEXT GRAMMAR

Ota Jirák
Doctoral Degree Programme (3), FIT BUT
E-mail: ijirak@fit.vutbr.cz

Supervised by: Dušan Kolář
E-mail: kolar@fit.vutbr.cz

ABSTRACT

The existing methods of the scattered context grammar parsing expand nonterminals deep in the pushdown. This expansion is implemented using either a linked list, or some kind of an auxiliary pushdown. This paper presents the parsing algorithm of an LL(1) scattered context grammar based on the table-driven principle commonly known for the context-free top-down parsing. It illustrates the function of this algorithm on a short example, and it discusses the future work. This approach works with the pushdown top only. It is assumed that this algorithm will be faster than other techniques.

1 INTRODUCTION

Till these days, several implementations of compilers for scattered context grammars (SCG, see [2]) have been proposed: (1) bottom-up (see [8]), (2) top-down (see [4, 5, 6]). These top-down techniques simulate the expansion of several nonterminals in one step usually with some type of a modification of a deep pushdown content. It is a time expensive operation due to the implementation (linked list, auxiliary pushdown).

This article presents an algorithm working with the top of the pushdown using principles of a lazy function evaluation. First, we adduce basic definitions. Then, we introduce the parsing algorithm for an LL scattered context grammar. Then, we illustrate the algorithm with a small example. Finally, we discuss the future work and open problems.

2 PRELIMINARIES AND DEFINITIONS

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

Definition 1. Let $G = (V,T,P,S)$ be an SCG, $r\colon (A_1,A_2,\ldots,A_n) \to (x_1,x_2,\ldots,x_n) \in P, n \in \mathbb{N}$. We define *indexing of rules*.

$$\begin{aligned} r[k] &= (A_k) \to (x_k), k \in \mathbb{N}, 1 \leq k \leq n, \\ r[k:] &= (A_k,\ldots,A_n) \to (x_k,\ldots,x_n), k \in \mathbb{N}, 1 \leq k \leq n, \\ r[k] = r[k:] &= \emptyset, k \in \mathbb{N}, k > n. \end{aligned}$$

Definition 2. Let $G = (V, T, P, S)$ be an SCG. P_1 is a multiset, such as $P_1 = \{p[1] : p \in P\}$. Then, G is LL SCG if $G_1 = (V, T, P_1, S)$ is LL context-free grammar.

Definition 3. *A predictive parsing LL-table for SCG* is a two-dimensional data structure that is indexed by nonterminal A and terminal a. *LL-table*$[A, a]$ contains an SCG rule, that must be used when A is a nonterminal on the top of the pushdown and a is a terminal under the reading head in the input string.

Definition 4. We recognize the *generation* of a symbol and the generation of a rule. The generation of a symbol means the rank of a derivation step where the symbol appears. The generation of a rule means the rank of a derivation step where the rule is used.

Definition 5. Let x be an input string $x = x_1 x_2 \dots x_n$, g is a generation, $x_1, \dots, x_n \in V$, and *reversal* be a function that reverses string x and add the generation g to each symbol.

$$reversal(x, g) = \langle x_n, g \rangle \dots \langle x_2, g \rangle \langle x_1, g \rangle$$

Definition 6. A lookup structure *Delay-List* represents a collection of keys and values. We use a pair $\langle generation, nonterminal \rangle$ as a key and the unprocessed components of the rule as a value. A *Delay-List*$[N, g]$ determines the delayed parts of the SCG rule with the lowest generation greater then g and it has nonterminal N on the left-hand side of the first component.

3 ALGORITHM

The parsing algorithm is based on the table-driven predictive parsing of a context-free grammar (see [1]). In our algorithm, we use a pushdown as usually and add a generation number to each symbol. It means, we have a pair $\langle symbol, generation \rangle$ for each pushdown item.

We initialize the generation counter and the pushdown at the beginning of the pushdown (Alg. 1.1–1.2). Depending on the type of the pushdown top symbol, we divide the algorithm into the three parts: (1) handling \$ (Alg. 1.6–1.12), (2) handling terminals (Alg. 1.13–1.20), (3) handling nonterminals (Alg. 1.21–1.36). The symbol \$ denotes the bottom of the pushdown or the end of the input string.

The first part deals with \$. Let a be the current token, $\langle X, g \rangle$ be the top of the pushdown. If a and X are equal to \$, the processing ends.

The second part handles terminal symbols. The same symbol on the pushdown and the current token leads to execution *pop* operation that removes the top of the pushdown and read a new token from the input string.

Third, the most complex part processes nonterminals. This part provides the selection of rules: try look-up (1) *Delay-List*, else (2) LL-table. This order is left without a proof. Naturally, this follows from the function representation of a lazy evaluation of a derivation step (see [3]). We try to find a delayed rule in *Delay-List*. When a rule is found, we replace the top of the pushdown according to the first component. The remaining components of SCG rule are returned into *Delay-List* (replace the record).

If we find no rule, we try to find a rule in the LL-table for nonterminal symbol X and terminal symbol a. If the LL-table does not contain a rule, it means that the input string does not belong to the language described by the input grammar.

When the while-cycle quits with empty *Delay-List*, the input string is accepted; otherwise the input string does not belong to the language described by the grammar.

Algorithm 1: Table-Driven Parsing of SCG

Input : LL-table for $G = (N, T, P, S)$; $x \in T^*$
Output: Left parse of x if $x \in L(G)$; otherwise, *error*

```
1.1  generation := 0;
1.2  initialize the pushdown by ⟨$, 0⟩ ⟨S, 0⟩
1.3  while pushdown is not empty do
1.4      let ⟨X, g⟩ = the pushdown top and a = the current token
1.5      switch X do
1.6          case X = $:
1.7              if a = $ and Delay-List is empty then
1.8                  success;
1.9              else
1.10                 error;
1.11             end
1.12         end
1.13         case X ∈ T:
1.14             if X = a then
1.15                 pop(⟨X, g⟩) ;
1.16                 read next a from the input string;
1.17             else
1.18                 error;
1.19             end
1.20         end
1.21         case X ∈ N:
1.22             if Delay-List[X, g] is not empty then
1.23                 denote Delay-List[X, g] as ⟨r : (X, X2, ..., Xn) → (x, x2, ..., xn), g'⟩;
1.24                 replace ⟨X, g⟩ with reversal(x, g') on the pushdown;
1.25                 replace Delay-List[X, g] by ⟨g', r[2 :]⟩;
1.26             else
1.27                 if r: (X, X2, ..., Xn) → (x, x2, ..., xn) ∈ LL-table[X, a] then
1.28                     generation := generation + 1;
1.29                     replace ⟨X, g⟩ with reversal(x, generation) on the pushdown;
1.30                     write r to the output;
1.31                     add ⟨generation, r[2: ]⟩ into Delay-List;
1.32                 else
1.33                     error;
1.34                 end
1.35             end
1.36         end
1.37     end
1.38 end
```

4 EXAMPLE

Algorithm 1 is demonstrated on Fig. 1. We get 1223 on the output. That represents the rank of rules used in the leftmost derivation.

LL Table

	a	b	c	$
S	1			
A	2	3		
B				
C				

SCG $G = (N,T,P,S)$, N=$\{S,A,B,C\}$, T=$\{a,b,c\}$,
$P = \{$

$$\begin{aligned} 1 &: (S) \quad \rightarrow (ABC), \\ 2 &: (A,B,C) \rightarrow (aA,bB,cC), \\ 3 &: (A,B,C) \rightarrow (\varepsilon,\varepsilon,\varepsilon)\} \end{aligned}$$

Input string: *aabbcc*$.

Pushdown	**Input**	**Action**	**Derivation**	**Delay-List**
$\langle \$,0\rangle\langle \underline{S},0\rangle$	$\underline{a}$abbcc$	1	$\underline{S} \Rightarrow \underline{A}BC$	
$\langle \$,0\rangle\langle C,1\rangle\langle B,1\rangle\langle \underline{A},1\rangle$	$\underline{a}$abbcc$	2	$\Rightarrow a\underline{A}BC$	$\langle 2,(B,C) \rightarrow (bB,cC)\rangle$
$\langle \$,0\rangle\langle C,1\rangle\langle B,1\rangle\langle A,2\rangle\langle \underline{a},2\rangle$	$\underline{a}$abbcc$	pop		$\langle 2,(B,C) \rightarrow (bB,cC)\rangle$
$\langle \$,0\rangle\langle C,1\rangle\langle B,1\rangle\langle \underline{A},2\rangle$	$\underline{a}$bbcc$	2	$\Rightarrow aa\underline{A}BC$	$\langle 2,(B,C) \rightarrow (bB,cC)\rangle$ $\langle 3,(B,C) \rightarrow (bB,cC)\rangle$
$\langle \$,0\rangle\langle C,1\rangle\langle B,1\rangle\langle A,3\rangle\langle \underline{a},3\rangle$	$\underline{a}$bbcc$	pop		$\langle 2,(B,C) \rightarrow (bB,cC)\rangle$ $\langle 3,(B,C) \rightarrow (bB,cC)\rangle$
$\langle \$,0\rangle\langle C,1\rangle\langle B,1\rangle\langle \underline{A},3\rangle$	$\underline{b}$bcc$	3	$\Rightarrow aa\underline{B}C$	$\langle 2,(B,C) \rightarrow (bB,cC)\rangle$ $\langle 3,(B,C) \rightarrow (bB,cC)\rangle$ $\langle 4,(B,C) \rightarrow (\varepsilon,\varepsilon)\rangle$
$\langle \$,0\rangle\langle C,1\rangle\langle \underline{B},1\rangle$	$\underline{b}$bcc$	d2	$\Rightarrow aab\underline{B}C$	$\langle 2,(C) \rightarrow (cC)\rangle$ $\langle 3,(B,C) \rightarrow (bB,cC)\rangle$ $\langle 4,(B,C) \rightarrow (\varepsilon,\varepsilon)\rangle$
$\langle \$,0\rangle\langle C,1\rangle\langle B,2\rangle\langle \underline{b},2\rangle$	$\underline{b}$bcc$	pop		$\langle 2,(C) \rightarrow (cC)\rangle$ $\langle 3,(B,C) \rightarrow (bB,cC)\rangle$ $\langle 4,(B,C) \rightarrow (\varepsilon,\varepsilon)\rangle$
$\langle \$,0\rangle\langle C,1\rangle\langle \underline{B},2\rangle$	$\underline{b}$cc$	d3	$\Rightarrow aabb\underline{B}C$	$\langle 2,(C) \rightarrow (cC)\rangle$ $\langle 3,(C) \rightarrow (cC)\rangle$ $\langle 4,(B,C) \rightarrow (\varepsilon,\varepsilon)\rangle$
$\langle \$,0\rangle\langle C,1\rangle\langle B,3\rangle\langle \underline{b},3\rangle$	$\underline{c}$c$	pop		$\langle 2,(C) \rightarrow (cC)\rangle$ $\langle 3,(C) \rightarrow (cC)\rangle$ $\langle 4,(B,C) \rightarrow (\varepsilon,\varepsilon)\rangle$
$\langle \$,0\rangle\langle C,1\rangle\langle \underline{B},3\rangle$	$\underline{c}$c$	d4	$\Rightarrow aabb\underline{C}$	$\langle 2,(C) \rightarrow (cC)\rangle$ $\langle 3,(C) \rightarrow (cC)\rangle$ $\langle 4,(C) \rightarrow (\varepsilon)\rangle$
$\langle \$,0\rangle\langle \underline{C},1\rangle$	$\underline{c}$c$	d2	$\Rightarrow aabbc\underline{C}$	$\langle 3,(C) \rightarrow (cC)\rangle$ $\langle 4,(C) \rightarrow (\varepsilon)\rangle$
$\langle \$,0\rangle\langle C,2\rangle\langle \underline{c},2\rangle$	$\underline{c}$c$	pop		$\langle 3,(C) \rightarrow (cC)\rangle$ $\langle 4,(C) \rightarrow (\varepsilon)\rangle$
$\langle \$,0\rangle\langle \underline{C},2\rangle$	$\underline{c}$$	d3		$\langle 4,(C) \rightarrow (\varepsilon)\rangle$
$\langle \$,0\rangle\langle C,3\rangle\langle \underline{c},3\rangle$	$\underline{c}$$	pop		$\langle 4,(C) \rightarrow (\varepsilon)\rangle$
$\langle \$,0\rangle\langle \underline{C},3\rangle$	$\underline{\$}$	d4	$\Rightarrow aabbcc$	
$\langle \underline{\$},0\rangle$	$\underline{\$}$	pop	success	

Figure 1: The Execution of the Algorithm 1 - d*n* denotes delayed rule number *n*

5 CONCLUSION AND FUTURE WORK

The presented algorithm exploits the principle of lazy evaluation to analyze a sentence defined by LL SCG. This approach avoids an expansion in the middle of the pushdown by working solely with the top of the pushdown.

The future work will focus on the generative power of LL SCG and an efficient implementation of *Delay-List* lookup structure.

ACKNOWLEDGEMENT

This work was partially supported by the BUT FIT grant FIT-10-S-2, the research plan MSM 0021630528, and the Czech Ministry of Education, Youth and Sports grant MŠMT 2C06008 "Virtual Laboratory of Microprocessor Technology Application" (visit http://www.vlam.cz).

REFERENCES

[1] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, & Tools*. Addison-Wesley Publishing Company, USA, 2nd edition, 2007.

[2] Sheila A. Greibach and John E. Hopcroft. Scattered context grammars. *J. Comput. Syst. Sci.*, 3(3):233–247, 1969.

[3] Ota Jirák and Dušan Kolář. Derivation in scattered context grammar via lazy function evaluation. In *Dagstuhl Post-proceedings of the Annual Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS'09)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2009.

[4] Dušan Kolář. *Pushdown Automata: Another Extensions and Transformations*. Brno, CZ, FIT BUT, 2005.

[5] Dušan Kolář. Scattered context grammars parsers. In *Proceedings of the 14th International Congress of Cybernetics and Systems of WOCS*, pages 491–500. Wroclaw University of Technology, 2008.

[6] Dušan Kolář and Alexander Meduna. Regulated pushdown automata. *Acta Cybernetica*, 2000(4):653–664, 2000.

[7] Alexander Meduna. *Automata and Languages: Theory and Applications*. Springer Verlag, 2005.

[8] Fred Popowich. Chart parsing of scattered context grammars. *Applied Mathematics Letters*, 7(1):35 – 40, 1994.