MEMORY CONSUMPTION OF CLASSICAL AND LAZY SCATTERED CONTEXT GRAMMAR PARSER

Ota Jirák
Doctoral Degree Programme (5), FIT BUT
E-mail: xjirak03@stud.fit.vutbr.cz

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

Abstract: In this paper, the space complexity of two scattered context grammar (SCG) parser implementations are studied. The classical and lazy SCG parsers are analyzed. The dependency of the memory consumption on an input data size is provided. A space complexity dependency on a grammar is demonstrated using several examples.

Keywords: SCG, parser, lazy, memory consumption, space complexity, comparison.

1 INTRODUCTION

In [1], both deep and lazy approaches of *scattered context grammar* (SCG) parser are compared from the time complexity point of view. In this paper, the space complexity point of view is discussed. A similar dependency of memory consumption on the size of the input data is presented.

At first, basic definitions are given. The description of a measurement procedure follows. The next section demonstrates a computation of a memory complexity dependency on the size of an input data. Finally, the generalization of the space complexity functions is proposed.

2 DEFINITIONS

It is expected that the reader is familiar with the formal languages [2]. In this section, definitions of *scattered context grammar* (SCG, [3]), *deep expanded pushdown automaton* (DEPDA, [4, 5]), and *lazy pushdown automaton* (LPDA, [5]) are provided.

Definition 1 (Scattered Context Grammar - SCG [3]) *SCG is a quadruple $G = (V,T,P,S)$, where V is a total alphabet, $T \subset V$ is the set of terminals, P is a finite set of rules of the form $(A_1,A_2,\ldots,A_n) \rightarrow (x_1,x_2,\ldots,x_n)$, where $n \geq 1$, $A_i \in V \setminus T$, and $x_i \in V^*$, for all $1 \leq i \leq n$, $S \in V \setminus T$ is the start symbol of G. An scg rule is an n-tuple of context-free rules.*

If $u = u_1A_1 \ldots u_nA_nu_{n+1}$, $v = v_1x_1 \ldots v_nx_nv_{n+1}$, and $p = (A_1,\ldots,A_n) \rightarrow (x_1,\ldots,x_n) \in P$, where $u_i, v_i \in V^*$, for all $1 \leq i \leq n+1$, then G makes a derivation step from u to v according to p, symbolically written as $u \Rightarrow v$. In addition, if A_i does not occure in u_i for all $1 \leq i \leq n$, then the derivation step is leftmost.

Definition 2 (Generation) *Let $G = (V,T,P,S)$, be an SCG. Let $u_1,\ldots,u_n \in V^*, p_1,\ldots,p_n \in P$, $S \Rightarrow_1 u_1 \Rightarrow_2 \cdots \Rightarrow_i u_i \Rightarrow \cdots \Rightarrow_n u_n$ be a sequence of leftmost derivations. Relation $\Rightarrow_i$ represents the i-th derivation step. The* generation *of the symbol is the number of derivation step when the symbol was appeared. The* generation *of the rule is the number of derivation step when the rule was used. The initial content of the pushdown has the generation zero.*

Definition 3 (Deep Expanded PDA - DEPDA [4, 5]) *Let $M = (Q, \Sigma, \Gamma, \delta, s, S, F)$, be a* pushdown automaton *(PDA, [2]) used for parsing of* context free grammar *(CFG, [2]) with an LL(1) table. The DEPDA is the PDA that processes the scg rules and expands deep in the pushdown. The LL-table is modified so as to be able to choose scg rule.*

The operation expansion is modified as follows. The LL-table chooses the scg rule based on the topmost nonterminal of the pushdown and the terminal under the reading head. All the left-hand side nonterminals from the context-free rules from the scg rule are selected in the pushdown and automatically rewriten. The leftmost derivation is used.

Definition 4 (Lazy PDA - LPDA [5]) *Let $M = (Q, \Sigma, \Gamma, \delta, s, S, F)$ be the PDA used for parsing of CFG with the LL-table. LPDA is the automaton M with a few modifications. A symbol in the pushdown is a pair $\langle s, gen \rangle$ where the gen is a generation of the symbol $s, s \in \Gamma$. The LL-table is modified so as to be able to choose scg rule. A structure called delay-bag is added. This structure keeps information about the partially processed rules. All the information of one postponed scg rule is stored in a structure called item. The item is of the form <the scg rule, the index to the first unprocessed part, the generation of the rule>.*

The operation expansion is modified as follows. First, the delay-bag is searched to determines the postpone rule. The oldest rule applicable on the topmost symbol of the pushdown is chosen from the delay-bag. If the delay-bag does not return scg rule the LL-table is searched. The first unprocessed context-free rule from the returned scg rule is processed. If there remain some unprocessed rules, the scg rule is postponed. A delay-bag item is inserted, or updated so it refers to the first unprocessed context-free rule from the scg rule.

3 MEASUREMENTS

The memory profiler *valgrind* [6] is used to obtain the memory consumption of a running algorithm. Valgrind plugin called *massif* is used for heap profiling. Each allocation and deallocation is logged. A log record contains the size of the allocated memory and/or the detailed stack trace. The measurement was conducted on Intell® Core™2 Duo CPU E8400 3.00 GHz with 32–bit Debian Linux 2.6.32-5-686, GCC 4.4.5.

The stack traces are used in the process of the identification of particular allocations in the log file. For more detailed information in the stack traces, analyzed program must be compiled with debugging information. After the identification process, less detailed logs can be used. The next measured data was used for confirmation of identified dependencies.

The measured values contain the size of the input data and the size of the memory representation of the grammar. These values are the overhead of encapsulating program. Therefore, they must be subtracted to get the maximum amount of the memory used only by the algorithm.

4 EXAMPLES

In this section, examples and derived space complexity functions are presented. Two algorithms are described: Deep (D) and Lazy (L), respectively. The first character in the function name indicates the algorithm. The second character indicates the implementation of the pushdown: Array (A) and Linked List (L), respectively.

Example 1. We measured for SCG $G_1 = (N, T, P_1, S), N = \{A, B, C, S\}, T = \{a, b, c\}, P_1 = \{1 : (S) \rightarrow (ABC), 2 : (A, B, C) \rightarrow (aA, bB, cC), 3 : (A, B, C) \rightarrow (\varepsilon, \varepsilon, \varepsilon)\}$.

The input data are in the form $a^n b^n c^n$. The size of the valid input, $|x|$, is $|a^n b^n c^n| = 3n$. Therefore, $n = \frac{x}{3}$. For DEPDA, the length of pushdown is $|aAb^n Bc^n C\$| = 2n + 5 = \frac{2x}{3} + 5$. \$ is the bottom

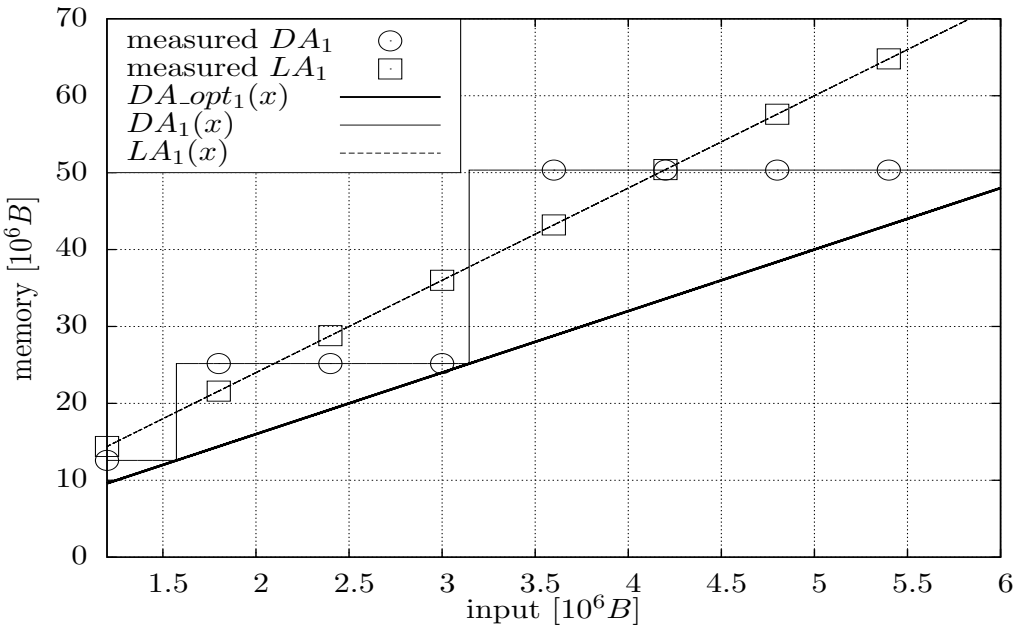

Figure 1: G_1 memory consumption

symbol (stopper) in the pushdown.

The array implementation has an initial size (2, in this case) and doubles its size, when more space is needed. Therefore, the size is proportional to $2^{\lceil \log_2(\frac{2x}{3}+5)\rceil}$.

For LPDA, the length of pushdown is given by the size of $|aABC\$| = 5$. The number of delayed rules ($S \Rightarrow ABC \Rightarrow^n a^nABC \Rightarrow a^nBC$) is $n+1 = \frac{x}{3}+1$.

Using static analysis, functions describing maximum memory consumption (without grammar and input) for G_1 were derived from measured data:

$$DA_1(x) = 12 \cdot 2^{\lceil log_2(\frac{2x}{3}+5)\rceil} + 28 \quad (1a)$$

$$DL_1(x) = 8x+64 \quad (1b)$$

$$LA_1(x) = 12x+296 \quad (1c)$$

$$LL_1(x) = 12x+268 \quad (1d)$$

Based on the knowledge of the grammar and based on the knowledge of the input data size, the size of the array can be optimized to:

$$DA\_opt_1(x) = 8x+88 \quad (1e)$$

$$LA\_opt_1(x) = 12x+220 \quad (1f)$$

In Figure 1, there are dependencies of measured data, functions DA_1, DA_opt_1, LA_1 on the input data size. Function DL_1 is similar to DA_opt_1, therefore only DA_opt_1 is presented. Functions DL_1, LL_1, and LA_opt_1 are not presented in Figure 1, because they overlap with the other presented functions.

The memory consumption for G_1 is better for the DEPDA. The size of the bag item memory representation (the structure for storing data in delay-bag) is bigger than the size of the pushdown item memory representation.

Because of saved pushdown space, LPDA memory consumption is lower in the case of long rules. This case is illustrated in Example 2. (see equations (2) and Figure 2).

Example 2. SCG $G_2 = (N,T,P,S), N = \{A,B,C,S\}, T = \{a,b,c\}, P = \{1:(S) \to (ABC), 2:(A,B,C) \to (a^{50}A, b^{50}B, c^{50}C), 3:(A,B,C) \to (\varepsilon,\varepsilon,\varepsilon)\}$.

For DEPDA, the length of pushdown is given by the size of $|a^{50}Ab^{50n}Bc^{50n}C\$|$. For LPDA, the length of the pushdown is given byt the size of $|a^{50}ABC\$|$. The number of postponed rules is $n+1$, where $n = \frac{x}{150}$. Using static analysis, functions describing maximum memory consumption (without grammar and input) for G_2 were derived from measured data:

$$DA_2(x) = 12 \cdot 2^{\lceil log_2(\frac{2x}{3}+54)\rceil} + 28 \quad (2a)$$

$$DL_2(x) = 8x + 652 \quad (2b)$$

$$LA_2(x) = 0.24x + 1192 \quad (2c)$$

$$LL_2(x) = 0.24x + 900 \quad (2d)$$

$$DA\_opt_2(x) = 8x + 676 \quad (2e)$$

$$LA\_opt_2(x) = 0.24x + 1032 \quad (2f)$$

In Figure 2, there are dependencies for grammar G_2. In this case, the lazy approach has significantly smaller memory complexity. Functions DL_2, LL_2, LA_opt_2 are not presented in Figure 2, because they overlap with the other presented functions.

The change of the second rule (from $(A,B,C) \to (aA,bB,cC)$ to $(A,B,C) \to (aA,Bb,Cc)$) in G_1 causes changes in the memory complexity of algorithms. The space complexity of the deep approach remains unchanged. The space complexity of the lazy approach is increased. This is caused by the change of the number of pushdown items ($|Bb^nC\$|$).

5 SPACE COMPLEXITY

The functions describing the maximum bytes in the memory used by the parsing algorithm can be generalized to:

$$\begin{aligned} DA(x) &= |ITEM_{DA}| \times ITEMS + |PD_{DA}| + |AUX| \\ &= 12 \times ITEMS + 28 + |AUX| \quad (3a) \\ DL(x) &= |ITEM_{DL}| \times ITEMS + |PD_{DL}| + |AUX| \\ &= 12 \times ITEMS + 4 + |AUX| \quad (3b) \\ LA(x) &= |ITEM_{LA}| \times LITEMS + delayed\_rules \times |bag\_item| + |PD_{LA}| + |AUX| \\ &= 16 \times LITEMS + 36 \times delayed\_rules + 132 + |AUX| \quad (3c) \\ LL(x) &= |ITEM_{LL}| \times LITEMS + delayed\_rules \times |bag\_item| + |PD_{LL}| + |AUX| \\ &= 16 \times LITEMS + 36 \times delayed\_rules + 104 + |AUX| \quad (3d) \end{aligned}$$

where $|AUX|$ means the size of auxiliary memory containing the representation of the input data and the grammar. $|ITEM_x|$ and $|PD_x|$ means the size of memory representation of the x pushdown symbol and the size of the pushdown x, respectively, where $x \in \{DA, DL, LA, LL\}$. The $|bag\_item|$ means the size of the delay-bag item. $ITEMS$, and $LITEMS$ represent the maximum length of the deep pushdown, and the lazy pushdown, respectively. The *delayed_rules* is the maximum number of delayed rules stored in the delay-bag. As we can see in (3a) and (3b), or (3c) and (3d), there are very small differences in the memory consumption between pushdown implementations. The array item and the list item of pushdown have the same size. The only difference is the length of the pushdown.

6 CONCLUSION

In this paper, the space complexity of two scattered context grammar parsers were provided. Both implementations have linear memory complexity. The linear complexity enables us processing of a

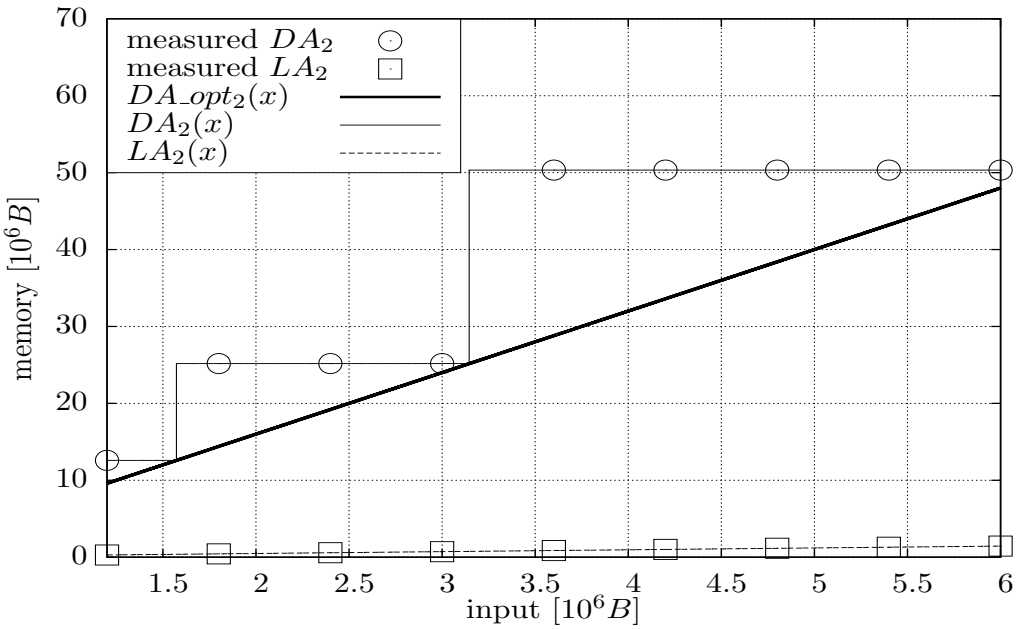

Figure 2: G_2 memory consumption

big amount of input data. A small difference between the optimized array implementation and the list implementation of the pushdown was presented. The dependency of memory consumption on the SCG grammar was demonstrated.

ACKNOWLEDGEMENT

This work was supported by the research programme MSM 0021630528 and the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070).

REFERENCES

[1] Jirák, O., Kolář, D.: Comparison of Classical and Lazy Approach in SCG Compiler, In: NUMERICAL ANALYSIS AND APPLIED MATHEMATICS ICNAAM 2011: International Conference on Numerical Analysis and Applied Mathematics, Halkidiki, GR, AIP, 2011, p. 873-876, ISSN 1551-7616.

[2] Meduna, A.: Automata and Languages: Theory and Applications. Springer-Verlag, London, 2000.

[3] Greibach, S., Hopcroft, J.: Scattered context grammars. J. Comput. Syst. Sci. 3, 233-247(1969).

[4] 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.

[5] Jirák, O., Kolář, D.:Derivation in Scattered Context Grammar via Lazy Function Evaluation, In: Annual Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS'09), Wadern, DE, DROPS, 2009, p. 10, ISBN 978-3-939897-15-6.

[6] Valgrind project, <`http://www.valgrind.org`> [cited 2012-03-05].