Journal of Robotics, Networking and Artificial Life
Vol. 6(2); September (2019), pp. 123–127
DOI: https://doi.org/10.2991/jrnal.k.190828.011; ISSN 2405-9021; eISSN 2352-6386
https://www.atlantis-press.com/journals/jrnal

Research Article

Implementation of Tamias to Check Production Rules for Parsing Expression Grammar

Tetsuro Katayama[1,*], Toshihiro Miyaji[1], Yoshihiro Kita[2], Hisaaki Yamaba[1], Kentaro Aburada[1], Naonobu Okazaki[1]

[1]*Department of Computer Science and Systems Engineering, Faculty of Engineering, University of Miyazaki, 1-1 Gakuen-kibanadai-nishi, Miyazaki 889-2192, Japan*
[2]*School of Computer Science, Tokyo University of Technology, 1404-1 Katakuramachi, Hachioji City, Tokyo 192-0982, Japan*

ARTICLE INFO

Article History
Received 15 October 2018
Accepted 22 November 2018

Keywords
Syntax analysis
parser
parsing expression grammar
packrat parsing

ABSTRACT

Parsing Expression Grammar (PEG) proposed by Ford has the higher expressive ability than traditional Backus–Naur form, but it also has problems such as prefix capture. "Prefix capture" is a problem of hiding the language to be accepted according to the order of choice. To support checking syntax files including such mistakes, this paper proposes Tamias: a production rules checker to support checking the PEG syntax files. Tamias has PEG interpreter which can check production rules of PEG. It can verify the behavior of production rules and measure the reach rate of choices.

1. INTRODUCTION

Backus–Naur Form (BNF) can express type-2 grammar (CFG: Context-Free Grammar) in the Chomsky hierarchy. Although BNF is traditionally used in the syntax definition of programming languages, it is possible to describe ambiguous grammar (e.g. dangling-else problem). The ambiguous grammar is not allowed programming languages because the compiler would interpret multiple languages. Unfortunately, it has been proved that there is no algorithm to judge that the CFG contains ambiguous grammar [1,2].

On the other hand, Parsing Expression Grammar (PEG) introduced by Ford [3] is not ambiguous grammar because it has ordered choice property. However, the ordered choice causes "prefix capture" [4]. "Prefix capture" is a problem of hiding the language to be accepted according to the order of choice. Checking syntax files that contain such mistakes usually confirms the behavior of the parser generated by the parser generator. However, in confirming the behavior of the parser, it is possible to check only the top level non-terminal symbols, and it is necessary to rebuild the parser for each change in the syntax files. To support checking the syntax files, this paper proposes Tamias to check the production rules in the PEG syntax files.

2. PARSING EXPRESSION GRAMMAR AND ITS PARSING TECHNIQUE

Parsing expression grammar is a formal grammar, introduced by Ford [3]. PEG is deterministic. Therefore, it is suitable for the syntax definition of the programming languages, and it was found that not only the CFG but also a part of the context-sensitive grammar can be expressed.

2.1. Definition of PEG

A PEG G is defined by the four-tuples:

$$G = (V_N, \Sigma, R, e_S)$$

where

1. V_N is a finite set of non-terminal symbols.
2. Σ is a finite set of terminals ($\Sigma \cap V_N = \phi$).
3. R is a finite set of production rules.
4. e_s is a parsing expression called the start expression.

The production rule is described in the form of "$A = e$" or "$A \leftarrow e$" (where $A \in V_N$, e is a parsing expression). A parsing expression is an instruction for an input string given by users and is described by a non-terminal or a terminal symbol. A parsing expression can use operators. When the input string s is applied to the parsing expression e, e matches the substring of s and indicates "success" or "failure". s has a pointer p representing the reading position. If e indicates success, e consumes the prefix of s (i.e. advances p by the length of the substring). Also, if e indicates failure, let s applies the next parsing expression.

2.2. Packrat Parsing Technique

Packrat parsing is one of the parsing techniques that can parse PEG [5]. It is a top-down parsing technique proposed by Ford

*Corresponding author. Email: kat@cs.miyazaki-u.ac.jp

in 2002. Ford solved the problem that the parsing time of PEG including a backtracking increases exponentially with linear time by memoization of the parsing result.

2.3. Prefix Capture

"Prefix capture" is a problem of hiding the language to be accepted according to the order of choice. For example, the grammar "A = ('+'/'++')[*a*–*z*]" does not accept the language "++*i*". The reason is that parsing "[*a*–*z*]" fails after parsing "+" of the language "++*i*". To solve this problem, the order '+' and '++' is reversed. Thus, it is difficult to understand prefix capture immediately.

3. TAMIAS

Tamias is a production rules checker to support checking the syntax files for PEG. Tamias has three areas: a text editor area, a list of non-terminal symbols, and a production rules check area. Tamias parses the production rules entered in the text editor area and converts them into an Abstract Syntax Tree (AST) as shown in Figure 1. Tamias has PEG interpreter which can check all choices and any non-terminal symbols in production rules according to PEG. Also, Tamias can parse without building a parser. A checking method using a PEG interpreter can be selected from the production rules check area. There are two methods of checking the production rules, "production rules verification" and "reach rate measurement". Both methods require one testable symbol and one input string. A testable symbol refers to a non-terminal symbol from which a terminal symbol can be derived. Tamias recursively searches and extracts testable symbols from the grammar described in the text editor area.

3.1. Production Rules Verification

Production rules verification can confirm whether the input string is accepted by the production rule. In one verification, production rules verification requires an expected output in addition to one testable symbol and one input string. The users can select whether the specified input string is accepted or rejected. Tamias displays the comparison result between the expected output and the output of PEG interpreter. If the expected output and the actual output are equal, "Passed" is displayed. Otherwise, "Failed" is displayed.

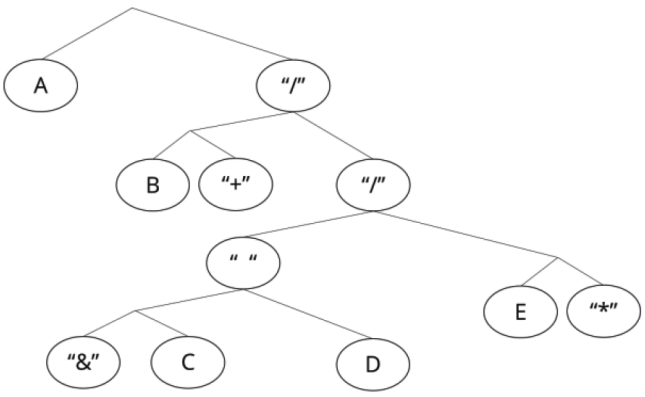

Figure 1 | AST for production rule "A = B + / & C D / E*".

3.2. Reach Rate Measurement

Reach rate measurement calculates the rate of success of parsing in the ordered choice of the selected production rule. The reach rate is obtained by assigning two or more check results in one production rule to the following calculating formula (1):

$$\text{Reach rate}\,[\%] = \frac{\text{Parsing success choices}}{\text{All choices}} \times 100 \qquad (1)$$

3.3. PEG Interpreter

Parsing expression grammar interpreter is an interpreter that executes parsing for PEG described in text editor in Tamias. PEG interpreter requires a testable symbol and an input string. PEG interpreter shows the output of the parsing expression after parsing. By selecting testable symbols, PEG interpreter can check not only top level non-terminal symbols.

4. IMPLEMENTATION OF PEG INTERPRETER

4.1. Data used in the PEG Interpreter

- Production rules:

  Production rules are rules obtained from non-terminal symbols that can be derived from testable symbols. One production rule is expressed as a pair of a non-terminal symbol and a parsing expression. The parsing expression is further, as shown in Figure 1, divided into ordered choice and sequence for easy execution with a PEG interpreter.

- Input string:

  Input string has a string and a pointer that expresses a reading position.

- Lookup table:

  Lookup table stores the reading position of the input string. The row of the lookup table corresponds to the parsing processing, and the column corresponds to the reading position of the input string.

4.2. PEG Interpreter Algorithm

We have developed a PEG interpreter based on Packrat parsing technique described in Subsection 2.2 to parse the grammar written in the text editor area. As a result, production rules can be quickly verified.

The algorithm of the PEG interpreter is shown in Figure 2. It recursively is executed until parsing of the terminal symbol.

5. EXPERIMENT AND DISCUSSION

We confirm that PEG interpreter can parse correctly by using grammar representing addition and multiplication and grammar including prefix capture.

Algorithm 1 PEG interpreter

Input: nt: testable symbol ∈ non-terminal symbol, s: input string
Output: success or failure

```
1: function PARSE(nt, s)
2:     if nt is stored in the lookup table then
3:         s consumes the stored value
4:         return success
5:     end if
6:     ex ← parsing expression corresponding to nt
7:     for choice in ordered choices in ex do
8:         if SEQUENCE(choice, s) is success then
9:             Memoize the reading position of s
10:            return success
11:        end if
12:        Memoize failed reading position of s
13:        return failure
14:    end for
15: end function
16: function SEQUENCE(choice, s)
17:    Get the current reading position of s
18:    for parsing expression in choice do
19:        if parsing by operator is failure then
20:            s backtrack to the previous position
21:            return failure
22:        end if
23:    end for
24:    return success
25: end function
```

Figure 2 | PEG interpreter algorithm.

5.1. Production Rules Verification

We confirm the production rules verification method by using grammar representing the addition and multiplication below:

Sum = Product '+' Product/Product
Product = Value '*' Value/Value
Value = [0–9]

We write the grammar in the text editor area of Tamias and check the two languages "1 + 2 * 3" and "*2" in the production rules check area. Next, we selected "accept" as the expected output in both languages. The experimental results are shown in Figure 3.

As shown in Figure 3, the check result of the language "1 + 2 * 3" according to the grammar was "Passed". On the other hand, the check result of the language "*2" not conforming to the grammar was "Failed". Therefore, the production rules verification can check the grammar correctly.

5.2. Reach Rate Measurement

We confirm the reach rate measurement method by using grammar "A = ('+'/'++')[a–z]" including prefix capture. However, since Tamias does not currently support input of parsing subexpression, we write the following grammar into the text editor area of Tamias.

A = Op Id
Op = '+'/'++'
Id = [a–z]

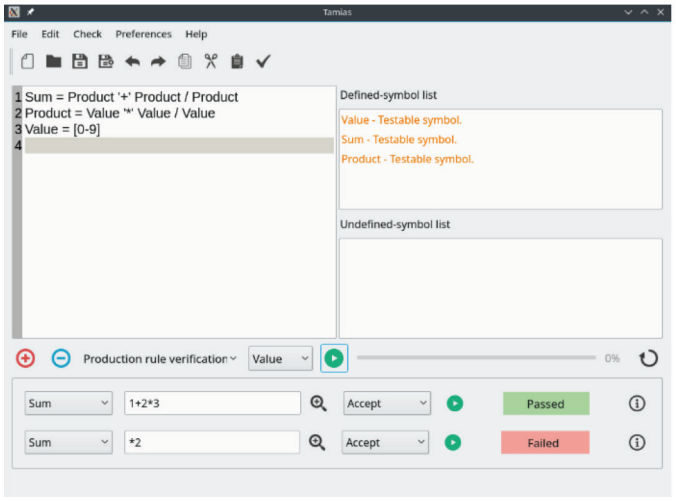

Figure 3 | Result of executing production rules verification.

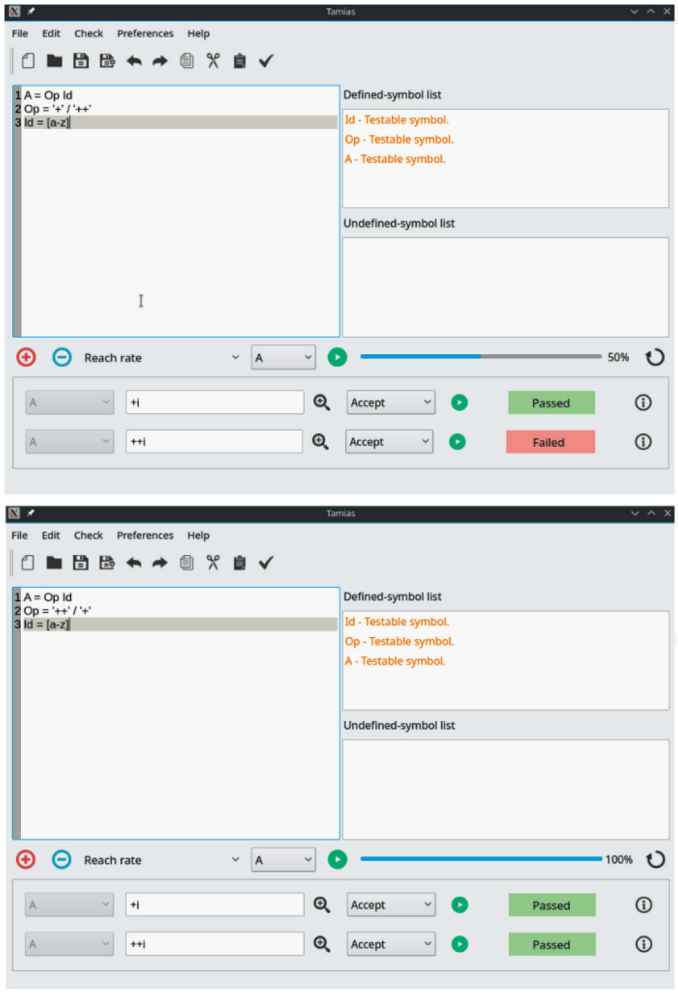

Figure 4 | Two results of executing reach rate measurement.

Next, we measure the reach rate of "A" using two languages "+i" and "++i". Finally, we reverse the choices of "Op" and measure the reach rate of "A" using the same languages. The experimental results are shown in Figure 4.

As shown in the upper part of Figure 4, the reach rate is not 100% in the grammar including prefix capture. In contrast, as shown in the lower part of Figure 4, the reach rate is 100% in the grammar excluding prefix capture. Therefore, from the formula of reach rate [formula (1)]:

$$\text{Parsing success choices} = \text{All choices}$$

it can be proved that prefix capture has not occurred for all input strings.

6. RELATED RESEARCH

Honda et al. [6] proposed a PEG debugger to support development of PEG. The PEG debugger has functions such as setting breakpoints, executing steps, displaying stack trace, and so on. The PEG debugger can check the behavior for one input and the occurrence of prefix capture. However, the PEG debugger cannot prove that prefix capture has not occurred for any input string.

Mori et al. [7] proposed a parser generator that can composite parsers and implemented its prototype. In the traditional method, when part of the syntax in PEG was changed, it was necessary to rebuild the corresponding parser. The proposed method can generate a parser that can deal with the problem of the traditional method. Tamias solves the problem of rebuilding the parser by implementing a PEG interpreter.

7. CONCLUSION

In this paper, we have proposed Tamias: a production rules checker to support checking the syntax files for PEG. Tamias has PEG interpreter which can check all choices and any non-terminal symbols in production rules according to PEG. Tamias can check the production rules and can measure reach rate of choices by using PEG interpreter. In the production rules verification, Tamias compares the expected output with the output of the PEG interpreter and displays the result. In the reach rate measurement, Tamias can check that the grammar does not contain prefix capture.

The experiment with a grammar representing addition and multiplication showed that Tamias can compare the expected output by users with actual output by PEG interpreter without building a parser. The experiment with a grammar including prefix capture showed that Tamias can prove that prefix capture has not occurred for any input strings by measuring reach rate.

Therefore, Tamias can support checking PEG syntax files. A future work is to improve Tamias to support parsing subexpression described by parentheses.

CONFLICTS OF INTEREST

The authors declare they have no conflicts of interest.

REFERENCES

[1] D.G. Cantor, On the ambiguity problem of Backus systems, J. Assoc. Comput. Mach. 9 (1962), 477–479.
[2] R.W. Floyd, On ambiguity in phrase structure languages, Commun. ACM 5 (1962), 526–534.
[3] B. Ford, Parsing expression grammars: a recognition-based syntactic foundation, 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages, ACM, Venice, Italy, 39 (2004), 111–122.
[4] R.R. Redziejowski, Parsing expression grammar as a primitive recursive-descent parser with backtracking, Fundam. Inform. 79 (2007), 513–524.
[5] B. Ford, Packrat parsing: simple, powerful, lazy, linear time, functional pearl, Seventh ACM SIGPLAN International Conference on Functional Programming, ACM, Pittsburgh, PA, USA, 37 (2002), 36–47.
[6] S. Honda, K. Kuramitsu, Implementing and evaluating a debugger for supporting development of PEGs. Japan Soc. Softw. Sci. Technol. 32 (2015), 1–6 (in Japanese).
[7] K. Mori, K. Wakita, Composition of parsers for incremental Parsing Expression Grammars. Japan Soc. Softw. Sci. Technol. 30 (2013), 641–650 (in Japanese).

Authors Introduction

Tetsuro Katayama

He received the PhD degree in Engineering from Kyushu University, Fukuoka, Japan in 1996. From 1996 to 2000 he has been a Research Associate at the Graduate School of Information Science, Nara Institute of Science and Technology, Japan. Since 2000, he has been an Associate Professor at Faculty of Engineering, Miyazaki University, Japan. He is currently a Professor with the Faculty of Engineering, University of Miyazaki, Japan. His research interests include software testing and quality. He is a member of the IPSJ, IEICE, and JSSST.

Toshihiro Miyaji

He received the Bachelor's degree in Engineering (Computer Science and Systems Engineering) from University of Miyazaki, Japan in 2018. He is currently a Master's student in Graduate School of Engineering at the University of Miyazaki, Japan. His research interests include parsing and testing for formal languages.

Yoshihir Kita

He received a PhD degree in Systems Engineering from University of Miyazaki, Japan, in 2011. He is currently an Assistant Professor with the Computer Science, Tokyo University of Technology, Japan. His research interests software testing and biometric authentication.

Kentaro Aburada

He received the B.S., M.S. and PhD degrees in Computer Science and System Engineering from the University of Miyazaki, Japan, in 2003, 2005 and 2009, respectively. He is currently an Associate Professor with the Faculty of Engineering, University of Miyazaki, Japan. His research interests include computer network and security. He is a member of IPSJ and IEICE.

Hisaaki Yamaba

He received the B.S. and M.S. degrees in Chemical Engineering from Tokyo Institute of Technology, Japan, in 1988 and 1990, respectively, and the Ph.D. degree in Systems Engineering from University of Miyazaki, Japan, in 2011. He is currently an Assistant Professor with the Faculty of Engineering, University of Miyazaki, Japan. His research interests include network security and user authentication. He is a member of SICE and SCEJ.

Naonobu Okazaki

He received his B.S., M.S., and PhD degrees in Electrical and Communication Engineering from Tohoku University, Japan, in 1986, 1988 and 1992, respectively. He joined the Information Technology Research and Development Center, Mitsubishi Electric Corporation in 1991. He is currently a Professor with the Faculty of Engineering, University of Miyazaki since 2002. His research interests include mobile network and network security. He is a member of IPSJ, IEICE and IEEE.