Web-based Learning Support System for Recursive Decent Parsing using Haste

Kenta Ohashi and Koji Kagawa
Kagawa University
2217-20 Hayashi-cho, Takamatsu, Kagawa 761-0396, JAPAN
kagawa@eng.kagawa-u.ac.jp

ABSTRACT

The Recursive descent parsing is covered in the latter half of the “Compiler" class of our department. It is difficult for beginners to understand this topic in a short period because there are multiple processes for them to learn. This paper proposes a Web-based learning support system for recursive descent parsing using Haste. Haste is a compiler that converts Haskell to JavaScript. Haskell is a functional programming language which is good at symbolic processing. The implemented system is Web-based and therefore it is easy for teachers to customize the user interface. It returns the calculation result of each step of recursive descent parsing to the BNF given by learners. The number of accesses and input data were collected and analyzed by Google Analytics.

KEYWORDS

Recursive Descent Parsing, Web, Haskell, Haste

1 INTRODUCTION

In recursive descent parsing, there is a lot of processes to be learned such as elimination of left recursions, calculation of the First and the Follow sets, creation of a parsing table and finally creation of a parser program. Therefore, it takes a considerable time to learn for the first time. However, in the compiler class, the time for the exercises is often not sufficient enough. In addition, it may be necessary to install software packages on the client computer and configure them such as setting up the path. These things are troublesome for the first learner, and there is a fear that this will decrease motivation for learning. On the other hand, it would be easy to install if the learner can use a Web-based system. It is also easy to update from the teacher side, and every user can use the latest version without explicit action for updating. Therefore, in this research, we will develop a Web-based learning support system for recursive descent parsing.

There are some existing systems for supporting learning parsing such as JFLAP [1], VCOCO [2] and ParseIT [3]. JFLAP is a graphical tool used when learning basic concepts of automata and formal languages. However, in general, learners cannot intuitively use it without reading the instruction manual. Furthermore, because the JFLAP itself and the Web page of its instruction manual are in English only, it is difficult for Japanese learners to understand the contents. In order to solve these problems, our research group has once developed a Web-based learning support system for automata where we extracted functions from JFLAP and implemented the user interface with Java Applets. However, Java Applets are no longer available as a Web browser plug-in. We were planning to use HTML and JavaScript for the user interface and Java Servlets for the interface with JFLAP. However, JFLAP is difficult to modify in such a way because such separation of user interface is not intended in its design. Therefore, we will develop a new system independent of JFLAP.

The system requires a lot of symbolic processing. Symbolic processing programs can be easier to write using functional programming languages such as Haskell than JavaScript. However, programs created by Haskell cannot be executed directly on Web browsers. Therefore, we use Haste [4, 5], which compiles Haskell programs into JavaScript codes. On the other hand, the implementation language of VCOCO [2] and ParseIT are not

clearly stated in their papers. To use these systems, learners have to use a stand-alone GUI application or a console application. It seems also possible to run them as server-side Web applications.

The structure of this paper is organized as follows. First, we will explain the Haste briefly in Section 2. Then, Section 3 will explain the internal structure of the system. Section 4 summarizes the result and Section 5 will discuss future directions.

2 HASTE

Haste is one of altJS languages and is a compiler that converts Haskell code into JavaScript code. AltJS is a generic term referring to languages that represent alternative to JavaScript such as CoffeeScript and TypeScript. AltJS often generates JavaScript code by compiling. Among altJS for Haskell, Haste can use the Haskell code without modification and has an advantage that compilation is fast enough. This makes it possible for us to use programs written in Haskell for Web systems.

3 OVERVIEW OF THE SYSTEM

Figure 1 shows the structure of the system. In this system, we use Haskell for implementing the computational part of recursive descent parsing.

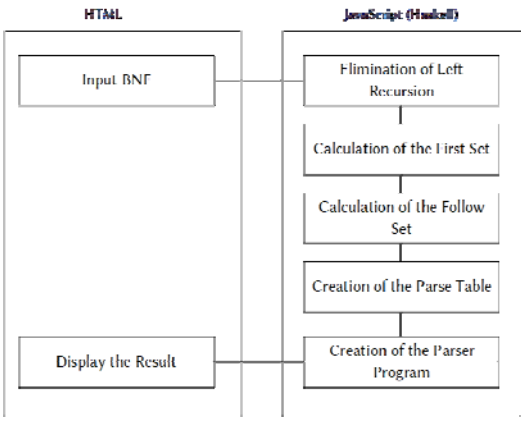

Figure 1. System Structure

Table 1 shows the types used in the computation part, where `Symbol` is a data type with two types of symbols for non-terminals and terminals. `F_rhs` is a type that has First and Follow symbols, rule numbers, and production rules in records. Table 2 shows functions that are used from the Web pages. After implementing these functions, we implemented I/O for HTML. The type of BNF received from HTML is `[[String]]`. We cannot call the function `elim` directly. Therefore we have implemented a function to convert inputs to the `BNF` type. In addition, if the return type of each function is not a `String`, it cannot be displayed in HTML directly. Therefore, we implemented functions to convert types `BNF`, `FIRST`, `FOLLOW`, and `TABLE` to the `String` type.

Table 1. Type Declarations

Name	Definition
RHS	`[Symbol]`
BNF	`[(String,[RHS])]`
FIRST	`[(String,[F_rhs])]`
FOLLOW	`[(String,[F_rhs])]`
TABLE	`[(String,[(String,[Symbol])])]`

Table 2. Core Functions

Name	Type	Remark
elem	`BNF -> BNF`	
first	`BNF -> FIRST`	
follow	`BNF -> FOLLOW`	
table	`BNF -> TABLE`	
prog	`TABLE -> String`	

The input window of the system is shown in Figure 2. The addition and deletion of the input form is implemented by jQuery.

ISBN: 978-1-941968-52-9 ©2018 SDIWC

Figure 2. Startup Screen

We can add and delete a nonterminal symbol to the lower side with the button on the upper side of the form and we can add and delete a production rule with the button on the left side of the form. By clicking the button written as samples learners can enter one of the samples that teachers prepared in advance. Figure 3 shows the screen after clicking the sample button. The contents of the sample BNF are contained in the array, and therefore they can be changed simply by changing the contents of the array. The tutorial button is implemented using intro.js (`http://introjs.com/`). Clicking it will play the tutorial. By clicking the delete-all button, it is possible to return to the initial state of Figure 2. By typing in the form at the bottom left learners can use multi-character terminal symbols. When input is finished and the create button on the upper left is clicked, calculation is performed and the result is displayed.

Figure 3. Input Tab

The tab with the parser table displays the result of the removal of left recursion, the First and the Follow set, and the parser table as shown in Figure 4. On the tab labeled Program, the parser program in the C language is displayed.

Figure 4. Table Tab

In addition, each time the step button on the right side of the create button is clicked once, calculations are carried out so as to advance the flowchart shown in Figure 5 one by one and the result of the step is displayed.

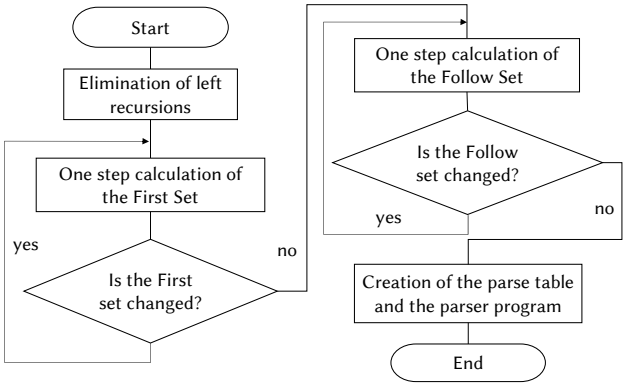

Figure 5. Flowchart of the step button

When learners move the mouse cursor onto the display part of the First and Follow set, the system shows by which rule it is added to the set using balloon.js (`https://kazzkiq.github.io/balloon.css/`).

ISBN: 978-1-941968-52-9 ©2018 SDIWC

The First set is calculated as follows.

1. If X is a terminal symbol, let $First(X) = X$.

2. If there is a production rule $A \rightarrow \varepsilon$, add ε to $First(A)$.

3. If there is a production rule $A \rightarrow B_1 B_2 ... B_n$ and for all k that $k < n$, and B_1, B_2, ..., and B_k can be ε, then add $First(B_{k+1})$ other than ε to $First(A)$. Furthermore, if all B_1, B_2, ..., and B_n contain ε, add ε to $First(A)$.

And the Follow set is calculated as follows.

1. For the start symbol S, add \$ to $Follow(S)$. This means reading the symbol \$ which indicates the end of the input after finishing the analysis of S.

2. If there is a production rule of $A \rightarrow \alpha B \beta$, add all the symbols of $First(\beta)$ except ε to $Follow(B)$.

3. If there is a production rule of $A \rightarrow \alpha B \beta$, and if $First(\beta)$ contains ε, or if there is a rule $A \rightarrow \alpha B$, add all the symbols contained in $Follow(A)$ to $Follow(B)$.

4 EVALUATION

In February 2017 before we introduce the step button and the terminal symbol of multiple characters, we asked three undergraduate the impression of the system. As a result, the following opinions came up.

· Since there are tutorials and several samples, it is easy to understand the system.

· It is pleasant to create programs by just entering production rules.

Based on these opinions, the learning support system seems to be used without much burden of understanding the usage of the system.

As limitations, we had the following opinions.

· It is difficult to use for beginners because it is a system based on some understanding of parsing.

· I want to input a symbol name that has multiple characters

· I want to select an object to be deleted instead of the rightmost one.

· I do not know where I made a mistake so I want to see the steps of calculation.

These opinions lead to the introduction of the step button and the terminal symbol of multiple characters.

On July, 2017, we introduced the system to 41 students of lecture “Compiler” of our department and encouraged them to use it. In addition, we conducted a questionnaire survey to confirm the purpose of use, whether it was useful, why they did not use, and so on in multiple choice forms, and to demand bug reports and function requests in free description forms. As a result, 23 users answered the questionnaire. Among them, 14 users answered that they only used the system to check the answer of the report problems. We also used Google Analytics to obtain usage history of each button. As a result, there were fewer uses of the step button than the “display-all” button.

Based on this result, we fixed and added some features to the system. The second survey was conducted for students of “Compiler” from July 19, 2018. We collected data on usage until the end of the examination of the compiler class using Google Analytics. As a result, the unique event that executed steps until the end was 77 cases except for the empty input. A unique event is an event that counts only the first time in one session. There were 49 kinds of input data other than 5 types of the prepared samples. 30 of them were ones that were slightly changed from past examination problems and report problems, and 19 were other data. In addition, the first sample was used more often than the other samples. The system was used only at a lecture and an exercise day on one week before the examination. Upon entering the examination week, the usage increased little by little; nine people each used the system at the

day before the examination and on the day of examination.

5 SUMMARY

We have developed a Web-based learning support system for recursive descent parsing. The system can show the steps of eliminating left recursions, computing the First and the Follow sets, creation of a parsing table, and creation of a parser program, and then can display the results on an HTML page. By inputting BNF in the form, or using prepared samples, learners can use it without knowing the detailed usage in advance. In addition, I/O is described in jQuery, and it is easy for teachers to customize the user interface component. By recording the usage using Google Analytics, it is shown that various inputs were executed in experiments, but most of the data input by learners were such ones that are slightly changed from report problems or past examination problems.

6 FUTURE WORK

Addition of answer field: We would like to add an answer field so that learners can enter their answers. By pointing out the wrong part of the answer, it can help learning by making learners conscious of what are not properly understood.

Display of errors: It is necessary to display errors more kindly for incorrect input.

Other parser topics: We would also like to implement learning support systems on other topics covered in the compiler class such as regular expression, finite automata, and bottom-up parsers.

Though it is an advantage to be able to execute without logging in the current system, teachers cannot know how much it is used by a certain user. Therefore, in the future we would like to build a system that can associate the learning record with users. This system must be able to prove what a certain learner achieved while maintaining anonymity. In addition, we believe that it is necessary to increase the number of samples and to create input randomly in order for learners to encourage testing various kinds of inputs.

ACKNOWLEDGEMENTS

This work is partially supported by JSPS KAKENHI Grant Number 15K01075.

REFERENCES

[1] S. Rodger and T. Finley, “JFLAP - An Interactive Formal Languages and Automata Package,” Jones and Bartlett, 2006.

[2] R. D. Resler and D. M. Deaver, “VCOCO: a visualisation tool for teaching compilers,” ACM SIGCSE Bulletin, 30(3), pp.199-202, 1998

[3] A. Karkare and A. Nimisha, “ParseIT: A Question-Answer based Tool to Learn Parsing Techniques,” Proceedings of the 10th Annual ACM India Compute Conference on ZZZ. ACM, pp. 115-120, 2017.

[4] A. Ekblad and K. Claessen, “A seamless, client-centric programming model for type safe web applications,” In Proceedings of the 2014 ACM SIGPLAN symposium on Haskell (Haskell '14). ACM, pp.79-89. DOI=http://dx.doi.org/10.1145/2633357.2633367, 2014

[5] A. Ekblad, “Foreign exchange at low, low rates a lightweight FFI for web-targeting Haskell dialects,” In Proceedings of the 27th Symposium on the Implementation and Application of Functional Programming Languages (IFL '15). ACM, Article 2, 13 pages. DOI: http://dx.doi.org/10.1145/2897336.2897338, 2015.