1L-4

The construction technique of a unific abstract syntax tree for two or more programming languages

Junichi Kobayashi[1], Kazunori Sakamoto[2], Hironori Washizaki[1], and Yoshiaki Fukazawa[1]

[1]Waseda University
[2]National Institute of Informatics

1 Introduction

In recent years, software is developed by various programming languages, such as C, Java, and Scala.

Some software is composed of multiple languages. For example, server side software is developed by javascript, and client side software is developed by Java in Web application.

There are many software engineering tools supporting each kind of languages, however these tools behave similarly. Such similarity is caused by the abstract syntax trees generated for each languages are different.

In addition, it takes large cost to develop new tool.

Moreover, there are many similar tools today, however the results of using these tools are different. This difference is caused that there is not existed an guideline of developing these tools.

In this paper, we propose the tool to solve these problems.

2 Background

Software engineering tools perform various operations to the *Abstract Syntax Tree*(AST). ASTs are obtained by conducting *lexical analysis* and *syntax analysis* in order to a program.

In this section, we explain these words and phrases about the technology used as our backgrounds.

2.1 Lexical Analysis

Lexical analysis is an operation which divides program for *tokens*. Token is an atomic term which has meanings such as word, numeral value, and symbol.

For example, we use expression $123*45+(67-8)/9$. Analysing with lexical analysis for this expression, we obtain the 11 tokens(123,$*$, 45, +, (, 67, $-$, 8,), /, 9).

2.2 Syntax Analysis

Syntax analysis is an operation which makes tree structure using tokens. This tree made by syntax analysis is called a syntax tree or concrete syntax tree.

For example, we use tokens obtained by lexical analysis in 2.1, the result of syntax analysis is shown in fig.1.

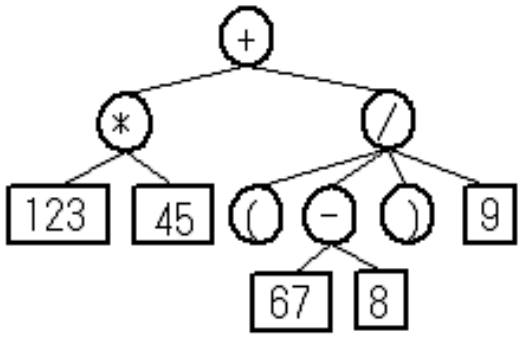

Figure 1: Example of syntax tree. In this figure, in order to deepen an understanding here, the token which serves as a sign containing a square and a operator in the token used as a numerical value is enclosed and expressed with a circle.

2.3 Abstract Syntax Tree (AST)

Abstract Syntax Tree (AST) is an tree structure. This is obtained by removing elements which is unnecessary for interpreting a program.

For example, the two elements "(" and ")" in Fig. 2.1 is used only determination of computation sequence. Therefore, these are removed in AST.

2.4 Backus-Naur Form(BNF)

Backus-Naur form(BNF) is a notation of the rule which usually uses lexical analysis and syntax analysis. BNF is one of the notations which define the grammar of a program, and it can be judged by performing description as shown in Fig.2 whether a certain program is grammatically right.

```
<文> ::= … | <条件分岐文> | …
<条件分岐文> ::= if ( <式> ) <処理>;
              | if ( <式> ) <処理>; else <処理>;
```

Figure 2: BNF example of if statement

3 Problems

There are two problems to develop a multiple language tool.

1. When we develop a software engineering tool, it takes large cost to create solving algorithms.
2. Although we have some algorithm, it takes large cost to correspond multiple language the tool.
3. Today, there are many tools behave similarly. However, result of applying these tools are different each other[1].

4 Proposal Technique

In this paper, we use following three techniques to solve the problems described in Section 3.

1. Create the extended language of BNF. This language allows to write some information for making a tool.
2. We propose the tool which automatically picks out only some elements required for the target tool from the AST of the program using the information described the language of 1.
3. We decrease the cost of developing multiple languages tools to half-automate description of process extracted by tool of 2.

4.1 Extraction BNF (ExBNF)

We define a new language named Extraction BNF (ExBNF). ExBNF is an extended language of BNF which can describe for element extraction.

In this language, the sign for element extraction can be attached during description of BNF. For example, if you want to count at the top of "if-statement" in Fig.2, you can write an asterisk($*$) in that description. This function enables a tool developer to take out only the necessary element from a program.

4.2 Auto-Extraction of Elements

We develop a tool which generates automatically files for ANTLR and element extraction data from ExBNF file.

Generated ANTLR file is used lexical analysis and syntax analysis by ANTLR 4. On the other hand, element extraction data file is used in next step.

This tool is composed of Xtext and Xtend, so this tool can operate on Eclipse. Xtext is a framework of description of DSL(Domain-Specific Language), and Xtend is a programming language extended for java and combined with Xtext.

4.3 Auto-Description of Processing

Below tool also generates some simple program for extracted elements. simple program contains a part of using element extraction data file. Therefore, it is no description for measuring some simple metrics.

In addition, Tool developers can use these programs and develop new tools combining these programs, so it is decreased to use this tool the cost of developing tools some complex metrics.

5 Conclusion and Future Work

In this paper, we proposed a new tool which analyse multiple language program and measure some metrics. Now, the program which counts each element to a expression was able to be created.

In the future, we consider that the following items.

1. apply this tool to each programming language and measurement some simple metrics.
2. use this tool measurement to complicated metrics for a language.
3. develop some software engineering tool such as mutation test.
4. apply some programming language which is not able to explain for BNF.

References

[1] Rdiger Lincke, Jonas Lundberg and Welf Lowe, "Comparing Software Metrics Tools," ISSTA '08 Proceedings of the 2008 international symposium on Software testing and analysis pp. 131-142