COMPGC04 – Systems Infrastructure – Compilers Coursework

Miguel Marin Vermelho
MSc Computer Science
SN:13038070
miguel.vermelho.13@ucl.ac.uk

Report and description of the system developed:

The aim of this coursework was to create a parser for JSON (JavaScript Object Notation). This was achieved by using a lexer and parser generator, JFlex and CUP respectively. Essentially the task was to create a specification that generates a program. Note that all files mentioned in the text are attached to this report and that the system was tested using the command prompt on Windows.

JSON is a text format for data interchange and it is built around two main structures: 1) collection of name/pair values i.e. an object and 2) an ordered list of values i.e. an array. In JSON values can be a String, number, object, array, Boolean or null. The first step was an analysis of the JSON syntax in order to determine which tokens were valid in the language and which elements were terminal or non-terminal. This information was the used to write JFlex and CUP code.

JFlex generates lexical analyzers to which we input a set of regular expressions. The program generated, also known as lexer, can match the regular expressions to a given input running actions which are specified by the regular expressions themselves. JFlex lexers are fast because "they are based on deterministic finite automata"[1]. JFlex is designed to work with CUP, a LALR (look-ahead, left to right (i.e. right-most derivation)) parser generator. CUP programs contain the grammar and are capable of parsing input that satisfies those grammars. In this case our derivation is built backwards. This corresponds to the machine-independent end (*front end*) of a compiler.

The regular expressions of this system are derived from the JSON specification found at: http://json.org/ (ECMA404 standard)[2]. These regular expressions can be found in the commented code file: Scanner.jflex. The scanner recognizes the regular expressions and passes input tokens as terminals to the CUP parser. If a given input does not conform with any regular

expression, the system will mark it as an illegal character and print it to the console giving the line and column where the character is located. The CUP parser receives valid JSON tokens from the scanner as terminals. The parser checks the input and tests whether it is valid JSON grammar. The grammar rules of this system are specified in the commented code file: Parser.cup and also based on the JSON specification.

The system is "built" using the *ant jar* command. This generates a jar file containing the parser. This file, located at jar\Compiler.jar can be run with the command: *java -jar jar\Compiler.jar*. This is assuming that we are working inside the minimal folder provided in the coursework brief. The system can the output whether parsing was successful or not. It outputs invalid characters which are rejected by the lexer and gives a *syntax error* if the tokens are valid but the grammar is not. The lexer and parser were tested on multiple input files, all of which were successfully parsed by the system. These tests progressively tested different and increasingly complex JSON structures in order to individually verify the regular expressions. Tests are attached in the tests folder. All the files were tested from the command line terminal. The parsing is working from the bottom up and outputs the structures that were parsed in the respective order. For instance, running the Zero_value.test file gives the following output:

```
C:\Users\migue\Documents\CS Files (16-17)\minimal>java -jar jar\Compiler.jar Zero_value.test
Pair parsed!
Item pair parsed!
Object parsed!
```

One of the limitations of this program is for example the fact that there is no specification of operator precedence and hence no way to deal with ambiguous mathematical expressions.

To conclude, the proposed system was successfully develop and is functional for the full JSON specification. The system is a combination of a lexer generated with JFlex and a parser generated with CUP. Regular expressions for JSON tokens as well as they grammar rules are specified in the lexer and in the parser, respectively. The system is designed to output structures that were successfully parsed as well as indicating whether an input is invalid; this can be either at the level of the lexer or the parser. Thus, this projects shows that it is easy to construct a parser using available software tools for a language with a high level of complexity.

References:

1. JFlex - JFlex The Fast Scanner Generator for Java. Available at: http://www.jflex.de/. (Accessed: 18th November 2016)
2. ECMA-404.pdf (Attached).