David Rudolf
MSc Computer Science - COMPGC04

JSON Lexer and Parser

Description of lexer

The lexer was written in JFlex. The lexical rules closely follow the JSON specification that where found online. First, the macros are defined. These are expressions that link a name to a regular expression. They are used to make the lexical rules easier to read and less cumbersome. From the JSON specification we know that numbers in JSON are not explicitly split between integers and floats whilst coding in JSON but need to be recognised as such whilst compiling the code because floats will use a decimal point but integers will not. The '*number*' definition encompasses any type of numerical values that could be encountered in JSON. The '*negative_zero_fraction*' is used to define negative zero decimal numbers that start with a zero as they are not encompassed by the regular expression '*{integer}{frac}*'.

Strings in JSON are defined as any character enclosed in double quotation marks except for the list of special characters (\", \b, \n, etc.).

Following the macros, the lexical rules are defined, they will be linked to the terminal expressions in the parser. The lexical rules essentially list, with the help of regular expressions, every possible character that could be in the input stream and assigns them to tokens.

Description of parser

The parser was generated using CUP. It consists of three main parts that are of interest: terminal symbols, non-terminal symbols and the grammar rules. Defining the terminal symbols consists simply in listing the symbols defined in the lexer, adding a Java object to the symbol if it will be used in conjunction with java code later on in the parser. The non-terminals are the expressions used within the parser that define groups of terminal symbols according to the grammar rules. The non-terminals '*print_value*' and *'print_string*' are used in conjunction with the print functions within the parser. In order to accommodate all the different Java object types that fall under 'value' in the parser, '*print_value*' is associated with the Java Object class. Furthermore, since *'print_string*' is only used to print strings type objects, it is associated with the Java String class.

The grammar in the parser takes its basic form from the JSON specification. Here the two main data structures have to be defined: arrays and objects. Arrays are lists of values whereas objects are lists of name/value pairs where names have to be of type strings and values can be anything from string, number, Boolean, null, object or array.

Error handling was added within the parser. For both JSON objects and arrays, if an error is encountered, the parsing will stop until the end of the object/array which is defined by a right curly bracket or square bracket.