Juan Bancamper
Formal Languages and Computability
31 January 2016

Implementing a DFA for Java For Loop

When creating a DFA, you must be able to determine all possibilities of input and output of each state. This can get very hectic as arrows are used to show the different options a user has to take in order to get to an accepting state. For Java's For Loop, a programmer has many options to declare a for loop that can do a certain number of statements, or do nothing at all. This declaration can also be included with whitespace, causing more states to be determined by a DFA. Using Java we will walk through parsing a Java for loop.

When receiving input for code, a for loop must always start with 'for'. This way we can look for this word followed by a grouping in between parenthesis. In Java, parenthesis cannot be parsed by a regular expression, so we must use a grouping to determine the initialization, termination, and incrementation. We can escape the parenthesis character, but this would complicate the grouping and the escaped parentheses would be considered the end of the line, when in actuality, it is not. So we group the loop head within the parentheses and we look for at least two semi-colons. This way it follows the basic declaration of a for loop. Within the spaces between the semi-colons, we can search for the implementation of an integer, or any primitive number type, followed by a correct boolean expression (equality, method, literal), and an increment/decrement assignment. The line should end with a curly brace. The last place can end with a new line, or just continue with a Java statement.

After the declaration of the loop head, we can go directly to the end of the loop by reading an end curly brace, or the program may choose to repeat a certain number of statements. As we stated before, a DFA for every single statement in Java would be ridiculously huge. Within the for loop, the line should be matched with any correct statement that can be made in Java. The example we are using just prints out a count, and this way we can search for the literal 'System.out.print', adding a possible 'ln' for new line terminator, followed by a grouping of available characters, numbers, symbols, etc… This is a short description of how a compiler works and how it uses pattern matching to be able to scan, parse, and generate code for a written for loop.