Brian Forsberg

Java Candidate Experience Report

The most difficult part of this project was coming up with a way to recursively parse the functions and function arguments. I had a lot of trouble thinking of how to identify the structure of an arbitrary expression. After several hours of brainstorming and research, I decided I had two options: come up with my own recursive descent parser and lexer, or use a framework such as ANTLR to do it for me. Either way, the process would have to look something like this:

If it wasn't for the 'let' expression, I could have evaluated every integer operation in the same way I would a prefix expression – with a stack. However, 'let' called for a more sophisticated approach.

I decided to use ANTLR, because writing my own parser and lexer from scratch seemed like a huge task and inefficient use of time. Given a grammar for a language, ANTLR can generate the corresponding parser and lexer for you. Figuring out how to define a language grammar was tricky, though. Explanations online were lacking, so I had to look at examples and try to draw conclusions as to how grammars work.

After a long time of struggling with ANTLR, I discovered ANTLRWorks. It is essentially an IDE for ANTLR grammars, which was exactly what I needed. I was able to get feedback on the syntax of my grammar. At this point I finally gained some traction on the project. Before long, I had a working project.

The bulk of my time was spent thinking about the problem and unsuccessfully trying to implement ideas for parsing. This project took me about 4 days to complete, and only a fraction of that time was spent actually writing code. Configuring maven and the dependencies took a significant amount of time also. I had never created my own maven build from scratch before, so I had to learn a few things. I had never setup a logging framework either, so that took the better part of a night to figure out. If there is one thing in this project that could be improved, it is the logging. My approach was adequate, but somewhat crude. There is still room to log more useful information.

How to run

1) Generate CalculatorLexer.java and CalculatorParser.java (created in target/generated-sources/antlr)
$: mvn generate-sources

2) Create .jar (stored in target/). I prefer this method because it circumvents issues with classpaths.
$: mvn package

3) Run jar with your expression as an argument
$ java -jar target/calculator-0.1.0.jar "let(a, 5, add(a, a))"

Appendix

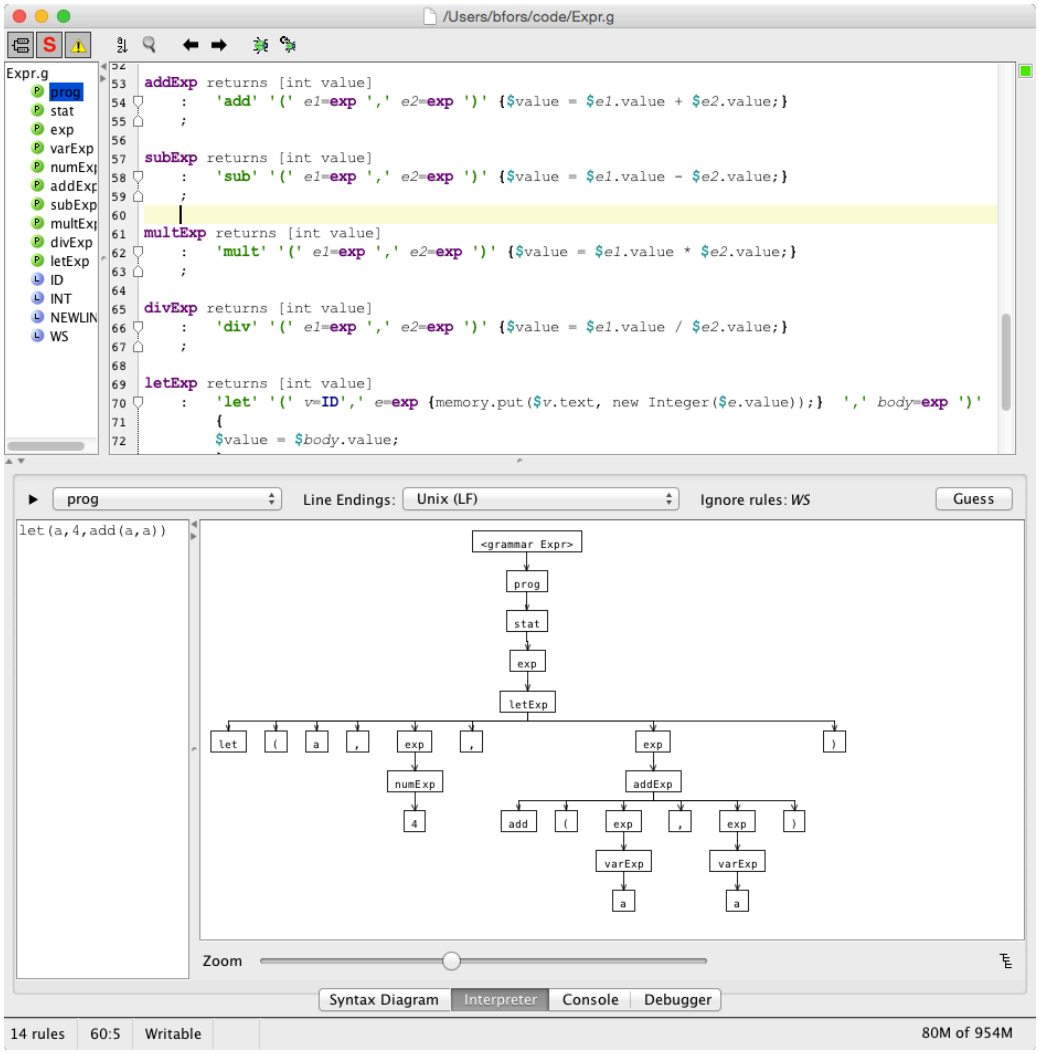

Figure 1: The structure of an expression displayed in ANTLRWorks.