Master-Thesis Planning Report

A Generator of an Incremental Divide-and-Conquer Lexers

Groupmember 1:
Jonas Hugo
MPALG
861017-5534

Groupmember 2:
Kristofer Hansson
MPALG
861208-4817

30 augusti 2013

1 Background

Editors normally have regular-expression based parsers, which are efficient and robust, but lack in precision: they are unable to recognize complex structures. Parsers used in compilers are precise, but typically not robust: they fail to recover after an error. They are also not efficient for editing purposes, because they have to parse files from the beginning, even if the user makes incremental changes to the input. More modern IDEs use compiler-strength parsers, but they give delayed feedback to the user. Building a parser with good characteristics is challenging: no system offers such a combination of properties.

2 Aim for the work

The goal of the project is to develop a generic tool that can translate any lexical specification into an incremental lexer for use in a syntax-aware editor. Most programming environments provide interactive feedback to the programmer. This feedback is often largely based on syntactic analysis of the source code being edited. Instead of using syntactical analysis this project will base the analysis on a lexer that can be run in real time while the user modifies the text.

The project will be divided into three main parts, a generic tool which generate an incremental lexer, and if time premits integrate the result with an incremental parser. The tool for generating the lexer will take precedence in the project. All the parts should fill the following requirements:

Robust Errors in the input should be handled as gracefully as possible. Ideally, errors in the input should only have local effects on the lexer.

Efficient Fast feedback is important. If feedback is slow, either the user will have to make frequent pauses, disrupting their work, or they risk reacting to outdated, incorrect information.

Precise The analysis performed by the parser should be as close as possible to that made by the compiler.

3 Limitations

The project will not focus on the building of underlying structres as DFA. It will neither focus on layout specific structure in a language. And for simplicity the lexer will only be tested for correcness on lexing the java language.

4 Method

The project will be carriedout in the follwing steps.

- We will analyse parser descriptions and produced lexing tables suitable for divide-and-conquer lexing

- We will write a lexer which takes an input (provided as a finger tree) and the above tables, and produces a sequences of tokens corresponding to the input.
- Because the lexer will use a divide-and-conquer strategy, it will be usable incrementally. That is, if the input is updated, only the results corresponding to the nodes of the tree containing the update will have to be recomputed.
- The results will be integrated in an existing lexer-generation tool such as Alex or BNFC

5 Time plan

2013-06-17 - 2013-10-01	Write report as project goes along
2013-06-17 - 2013-07-17	Research in lexing techniques
2013-07-17 - 2013-08-20	Develop the incremental lexer
2013-08-29	Attend first oral presentations
2013-08-20 - 2013-09-02	Prove correctness of developed lexer
2013-08-20 - 2013-09-09	Preformence testing the lexer
2013-08-20 - 2013-10-01	Focus on writing report
2013-10-01 <	attend secound oral presentation
2013-10-01 <	Opponent at another project
2013-10-01 <	Oral presentation