Incremental and parallel parsing in Yi

Planning report

Tobias Olausson

gusolaut@student.gu.se

Note: references are omitted here; see the project proposal for details about those. Some material in this report has been reused from the proposal.

Preliminary title

The preliminary title for this thesis is *Incremental and parallel parsing in Yi*. An alternative, in case time doesn't permit implementation into Yi would simply be *Implementing incremental and parallel parsing*.

Background

Syntax highlighting is a key feature in modern day text editors. Programmers want to get information about how well their code conforms to the programming language grammar, but without having to compile or run their code. Instead of using the widely adopted regular expression approach to syntax highlighting, we could use a fast lexer and parser to give us syntax information in the form of an abstract syntax tree. Such a tree would provide more information about the given code, and makes for the ability to manipulate the code in new, cool ways.

Aim for the work

The aim for this thesis project is to write an incremental, parallelizable parser in Haskell, which can be connected to the lexer written by Hugo and Hansson. The ultimate goal would be to put these both to the test by integrating them into the Yi text editor.

The assignment

Recent research by Claessen and Bernardy has shown that parsing of programming language text can be done efficiently by using a divide-and-conquer algorithm. Another MSc thesis by Hugo and Hansson has given an incremental divide-and-conquer lexer for a context-free grammar. The lexer is written in an incremental manner, but the proof-of-concept implementation of the parser is not. The main task is to reimplement the parser in an incremental fashion, and to integrate the lexer and parser to work as one.

Delimitations

The project will be concerned with only the lexer generator and parsing, other parts of the Yi editor will be left untouched as much as possible. Cool applications using the syntax information generated by the lexer and parser will not be a priority, but rather a bonus if time allows for it.

Methodology

This project is mainly concerned with implementing theories already proven into a practical application. Obviously, before implementing anything the existing Yi code base must be carefully examined. When it comes to the actual code, much work will revolve around testing and debugging the code, in an iterative manner. The resulting parser should be easily checked against existing grammars both in the editor and in a more stand-alone environment.

Time plan

The project is planned for the spring semester of 2014. Implementation and integration of the components will be performed in several steps. Time plan will be revised during the work, together with the supervisor.

1. Study the background papers (first 1-2 weeks)
2. Implement the incremental parallel parser (should be done by end of february
    - Extend the matrix computation code
    - Write a parsing monoid that can be used together with BNFC generated code

    - Combine (by hand) the lexer and parser

3. Generalize (templatify) the lexer generator (should take no more than two weeks)

4. Integrate the parser (with the lexer) into Yi (until the middle of april)

5. Use syntactic information for cool stuff (if time permits it)

By the middle of April, most of the coding should be finished, and focus should rather be on writing the report. Some coding will be done, however, since one usually finds some mistakes when writing about what was implemented.