# Incremental and parallel parsing in Yi

Planning report

Tobias Olausson

gusolaut@student.gu.se

## Preliminary title

The preliminary title for this thesis is *Incremental and parallel parsing in Yi*. An alternative, in case time does not 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 lexer and parser that is both incremental and fast 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[2]. The ultimate goal would be to put these both to the test by integrating them into the Yi text editor.

## Assignment

Recent research by Bernardy and Claessen[1] has shown that parsing of programming language text adhering to a context-free grammar can be done

efficiently by using a divide-and-conquer algorithm, improved from Valiant's [3]. Another MSc thesis by Hugo and Hansson has given an incremental divide-and-conquer lexer for a context-free grammar. The lexer works 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 together.

# 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. The time plan will be continuosly checked and possibly revised during the work, together with the supervisor.

## Background data

The first 1-2 weeks will be spent studying background papers and getting familiar with the lexer written by Hugo and Hansson.

## Implement incremental parser

The incremental parser should be implemented and tested by March 9th. The parser will utilize finger trees as its primary data structure, just as the

lexer does. There are two major steps in implementing the parser.

First, the existing proof-of-concept implementation of parsing using matrix multiplication has to be extended in order to be usable from an incremental parsing approach where merging two sub-trees should be possible. This means implementing some new functionality into BNFC[5].

Second, a parsing monoid that can be used in the finger tree as well as with the matrix multiplication code in BNFC has to be written. The monoid should transform lexemes into the abstract syntax tree that the parser will output as a result, and will use the matrix multiplication to do this.

## Generalize the lexer generator

The currently working code from the lexer generator by Hugo and Hansson is specific to the Java programming language and uses a state machine generated by Alex internally. This can be generalized, since the state machine and data types for tokens are the only language-specific parts of their code. It should not be too much work to turn their results into something that BNFC can generate for any given language. Of course, a reasonable code generation from BNFC will also include the aforementioned parser. An estimated time frame for this is two weeks.

## Integration with Yi

Once the lexer and parser works, and can be generated given any context-free grammar, this should be integrated into the Yi text editor[4], where incremental parsing could be used to check syntax correctness without having to call the compiler for the language. If time permits, this integration can be extended with functions that use the syntactic information to do cool stuff, such as AST transformations etc.

## Report

By the middle of April, most of the coding should be finished, and focus should rather be on writing the report. However, we expect some debugging and beautifying of the code to take place during the writing phase, as documenting a piece of work usually reveals minor shortcomings.

# References


[1] Efficient Divide-and-Conquer Parsing of Practical Context-Free Languages
Jean-Philippe Bernardy, Koen Claessen
ICFP 2013

[2] A generator of incremental divide-and-conquer lexers
Jonas Hugo, Christoffer Hansson
MSc Thesis, Chalmers University of Technology, draft as of February 2014

[3] General Context-Free Recognition in Less than Cubic Time
Leslie G. Valiant
Journal of Computer and System Sciences 10, 1975

[4] Yi: an editor in Haskell for Haskell.
Jean-Philippe Bernardy
In Proc. of the first ACM SIGPLAN symposium on Haskell, pages 61–62. ACM, 2008.

[5] The BNF Converter
http://bnfc.digitalgrammars.com/