

# CHALMERS



## A Generator of Divide-and-Conquer Lexers

A Tool to Generate an Incremental Lexer from a  
Lexical Specification

*Master of Science Thesis [in the Programme MPALG]*

JONAS HUGO

KRISTOFER HANSSON

CHALMERS UNIVERSITY OF TECHNOLOGY  
Department of Computer Science and Engineering  
Göteborg, Sweden, June 2013

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet. The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

An not to long headline describing the content of the report  
A Subtitle that can be Very Much Longer if Necessary  
JONAS HUGO,  
KRISTOFER HANSSON,

© JONAS HUGO, June 2013.  
© KRISTOFER HANSSON, June 2013.

Examiner: NAME A. FAMILYNAME

Chalmers University of Technology  
University of Gothenburg  
Department of Computer Science and Engineering  
SE-412 96 Göteborg  
Sweden  
Telephone + 46 (0)31-772 1000

Cover:  
an explanatory caption for the (possible) cover picture  
with page reference to detailed information in this essay.

Department of Computer Science and Engineering  
Göteborg, Sweden June 2013



## **Abstract**

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.



## Acknowledgements

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

The Authors, Location 11/9/11



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Known techniques</b>	<b>2</b>
2.1	Lexer . . . . .	2
2.1.1	Regular Expressions . . . . .	2
2.1.2	Finite State Machine . . . . .	3
2.2	FingerTree . . . . .	3
2.3	BNF . . . . .	3
2.4	Yi . . . . .	3
2.5	Haskell . . . . .	3
2.6	Monoid (data type) . . . . .	3
	<b>Bibliography</b>	<b>4</b>



# 1

## Introduction

\*Usage of BNFC \*With help of regexp build a finit state machine that will lex a code string. \*Give finite states with corresponding Monoid data type. \*Flag for errors from the Lexer, give meningfull info to the user, and stop the worklow after lexer, until new updated text. \*If no errors, handel layout \*Parse the Monoid data type tree, AKA integrate the result with an existing parser. \*Smile and be happy!

Known techniques: \*Regular Expressions \*FingerTree \*Finite State Machine \*BNF \*Yi \*Haskell \*Monoid (data type)

# 2

## Known Techniques

#Some Text describing what the report will illuminate in this chapter.

### 2.1 Lexer

A Lexer, lexical analyzer, is a program which jobb is to convert a string of a formal language into a sequence of tokens. #Hitta REF. This can be done by using regular expressions, regular sets and finite automata. Which are centerel consepts in formal language theory. [1]

#### 2.1.1 Languages

**Formal Languages**

**Regular Languages**

Like any formal language, a regular language is a set of strings. In other words a sequence of symbols, from a finite set of symbols. Only some formal languages are regular; in fact, regular languages are exactly those that can be defined by regulat expressions. [? ]

#### 2.1.2 Regular Expressions

Regular expressions are used to describe a patterns in a string. In a regular language, a programming language, this is usefull. Since these languages are build on very strict rules on how strings must follow a pattern. #Ref på detta!!

**Definition 2.1.1** (Regular Expressions [1]). 1. The following characters are meta characters  $\{ '|', '(', ')', '*', '\}$ .

2. A none meta character  $a$  is a regular expression that matches the string  $a$ .

3. If  $r_1$  and  $r_2$  are regular expressions then  $(r_1|r_2)$  is a regular expression that matches any string that matches  $r_1$  or  $r_2$ .
4. If  $r_1$  and  $r_2$  are regular expressions.  $(r_1)(r_2)$  is a regular expression of the form that matches the string  $xy$  iff  $x$  matches  $r_1$  and  $y$  matches  $r_2$ .
5. If  $r$  is a regular expression the  $r^*$  is a regular expression that matches any string of the form  $x_1x_2\ldots x_n, n \geq 0$ . Where  $r$  matches  $x_i$  for  $1 \leq i \leq n$ , in particular  $(r)^*$  matches the empty string,  $\varepsilon$ .
6. If  $r$  is a regular expression, then  $(r)$  is a regular expression that matches the same string as  $r$ .

Many parantheses can be reduced by adopting the convention that the Kleene closure operator  $*$  has the highest precedence, then concat and then or operator  $|$ . The two binary operators, concat and  $|$  are left left-associative. [1]

### 2.1.3 Finite State Machine

### 2.1.4 Known Solutions

## 2.2 FingerTree

## 2.3 BNF

## 2.4 Yi

## 2.5 Haskell

## 2.6 Monoid (data type)

# Bibliography

- [1] A. V. Aho, Handbook of theoretical computer science (vol. a), MIT Press, Cambridge, MA, USA, 1990, Ch. Algorithms for finding patterns in strings, pp. 255–300.  
URL <http://dl.acm.org/citation.cfm?id=114872.114877>