UNIVERSITY OF COPENHAGEN

Bachelor's Project

Christian Bendix Fjordstrøm

Hybrid Syntax For Composition of Regular Languages

Date: May 21, 2024

Advisor: Andrzej Filinski

Abstract

Table of content

1 Introduction

2 Analysis

2.1 Hybrid Syntax

- Should allow for arbitrarily nested compositional expressions consisting of regular expressions, regular grammars and automata
- Regular expressions
- Undecidable if a CFG describes a regular language, so it needs to be restricted
- Before conversion, it needs to be checked that regular grammars/automata can be converted

2.2 I/O

The user should also be able to define which output format they want, and the syntax of the output should be the same as the syntax for the input, such that it can be fed back in. The output should also exhibit a roundtrip property, meaning that feeding the output back in should result in the same language. Additionally, there should also be an option for a user to check if a string is recognised by the given regular language.

2.3 Composition

The standard regular expression operators such as concatenation, repetition(zero or more and one or more) and union as well as intersection and complement should be applicable to both regular expressions and automata. In addition, the syntax for how to express these combinations must be designed and implemented.

2.4 Conversion

To facilitate composition using intersection and complement as well as the ability to choose the output format, one be able to convert regular expressions to NFA's, convert NFA's to DFA's, minimise DFA's and convert automata back to regular expressions.

3 Design

3.1 Hybrid Syntax

- To have hybrid syntax of RE and automata one must have a way of expressing compositions
- To enable this, input split in two: grammar and extended RE where nonterminals which are defined in the grammar can be referenced in the RE

3.1.1 Extended Regular Expressions

- Operations
- Grammar of regex
- Simplify AST during parsing by replacing ?, +, .

3.1.2 Grammar

- Accodomate regular grammar, nfa, dfa
- Syntax: nt -> regex;
- Having a non-recursive production means that the non-terminal represents an accepting state

3.1.2.1 Restrictions

- Since it is undecidable if a cfg is regular, the grammar must be restricted Like in regular grammar, only allow recursion in tail position No recursion inside zero or more, intersection and complement

3.2 Checking Well-formedness of Grammar

- Two parts: checking that recursive nonterminals do not occur where they are not allowed and checking if the grammar is stratifiable
- Explanation of algorithm:
- Part 1: for every non-terminal, build set of the non-tail non-terminals and tail nonterminals that exist in the productions of that non-terminal
- if union then non-tail = union of non-tail and tail = union of tail
- if sequence then tail is simply the last element
for the above, if recursive non-tail NT then fail
- zero or more, intersection and complement are simpler: if any recursive non-terminals exist,

then the check fails, because of this, the distinction between tail and non-tail nonterminals is irrelevant

- Part 2:
- For the grammar to be stratifiable, it must be true for all non-terminals that the non-terminals that exist in its productions must either be the non-terminal itself or not depend on that non-terminal at all. This approach disallows mutual recursion, and it assumes that the grammar has already been checked such that recursive non-terminals only exist in the tail-position.
- The algorithm builds a series of layers where each layer is a set of NT's that do not contain each other, and where the productions of the NT's in a layer only depend on non-terminals defined in previous layers
- every iteration of the algorithm builds a new layer
- for every NT, find the NT's that exist in its productions
- if the NT contains no NT's then it cn immediately be added to the current layer
- else if forall NT's in its productions, if the NT's are either itself or exist in previous layers, then the NT is added to the current layer
- the productions of the NT's in the current layer are removed from the grammar, if the grammar is non-empty and the layer is empty, then the grammar is not stratifiable and the check for well-formedness fails

3.3 Conversion

3.3.1 Regular expression to NFA

3.3.1.1 Standard regular expressions

3.3.1.2 Intersection

3.3.1.3 Complement

3.3.1.4 Nonterminal occurences

- Two parts: NT's with recursive and non-recursive productions
- No recursion: since productions are regular expressions, the union of all the productions can be directly converted using the algorithm for conversion of standard regular expressions, intersection and complement as explained previously.
- Recursion: as explained earlier, recursive productions must only have recursion in the tail

position of the expression. This means that an expression containing recursion is always the same as removing the recursive NT and making the NFA that corresponds to that expression end at the same state as its starting state. The NFAs representing all the individual productions can then be combined to get the final result.

3.3.2 NFA to DFA

3.3.3 Minimisation of DFA’s

3.3.4 Automata to regular expression

3.4 I/O

4 Implementation

5 Testing

6 User Guide

7 Conclusion