Bachelor Thesis

Markus B. Jensen

June, 2022

Abstract

Preface

Contents

1 Introduction

2 Background Material

2.1 Dependency Graphs

A dependency graph consists of vertices and so-called hyper edges connecting the vertices. A hyper edge points from a single vertex to zero or more vertices. Thus, a hyper edge can be expressed as a tuple (v, S) where v is the vertex and S the set of vertices that this exact hyper edge points to. This implies that it is possible that $S = \emptyset$ which is indeed the case. This is not the same as a vertex not having a hyper edge: A vertex can have no hyper edges; a vertex with a hyper edge pointing to the empty has at least one hyper edge. Note that a vertex v can have multiple hyper edges: (v, S_1), (v, S_2), $\ldots$ (v, S_n). A simple directed graph can be characterized as a dependency graph with only singleton hyper edges.

2.2 Railway Networks

The most fundamental part of railway network is the tracks.

3 Implementation

3.1 Programming Paradigm

F# is a functional-first programming language. This is great as functional programming can be very succinct. But functional programming is not always the fastest approach. For this project, the local algorithm from [Liu and Smolka, 1998] is implemented as a `Solver`-object with an iterative implementation of the algorithm.

[Liu and Smolka, 1998] uses arrays in the implementation of the local algorithm. Arrays are not very functional as they are mutable. To make the code functional, the arrays could easily be replaced by a combination of linked lists and maps. This will result in 1) the code being functional but also 2) the code being much slower and taking up more memory. When interested in speed of the algorithm, these trade-offs are severe.

The assignments `A` and `D` in the paper are implemented as arrays in their `Algorithm Local2`. A more apparent solution would be to implement them as the F# datatype `Map`. This makes sense as they more resemble the mathematical understanding of the assignments. But Maps in F# are implemented as binary trees making both look-up and insertion $O(\log n)$ operations (where n is the height of the tree). Insertions and look-up are used often.

The state space for this project can easily explode: Arrays are much more compact than Maps. Maps can be more flexible but not even this is needed: We know the entire state space before running the algorithm.

Thus, the only not to use arrays is that they are not functional. But given that F# allows for iterative programming, implementing this algorithm iteratively makes the most sense. This will make this project not purely functional, but that is a worthy trade-off.

3.2 Railway Networks

A railway network can be represented as a tuple (triple) of the following: A map representing the port connections, a map of the signals identifications and there placement, and a map of the trains' initial and final positions.

3.2.1 Network thoughts

When

3.3 Domain-Specific Language (DSL)

For easier creation of railway systems a DSL has been implemented. This DSL is inspired by the DSL described in [Kasting, 2016]. Contrary to the approach cited, the F# library FsLexYacc is used to write the lexer and parser for this DSL.

This has been a great exercise in writing a lexer/parser. One problem with FsLexYacc, though, is error handling: While it is very easy to spot wrong tokens and characters in the lexer, generating a useful error message for the parser is difficult. If this project is to expand into something bigge, this would definitely be an area to improve.

3.4 Unit Tests

Some key elements of the code has been tested using unit tests. For this, the `xUnit` library has been used.

4 Conclusion

A Appendix Example

Just replace me with the first appendix to be written.

Bibliography

[Kasting, 2016] Kasting, P. F. S. (2016). System synthesis using parity games, system syntese ved brug af parity games.

[Liu and Smolka, 1998] Liu, X. and Smolka, S. (1998). Simple linear-time algorithms for minimal fixed points. *Automata, Languages and Programming*, 1443:53–66.