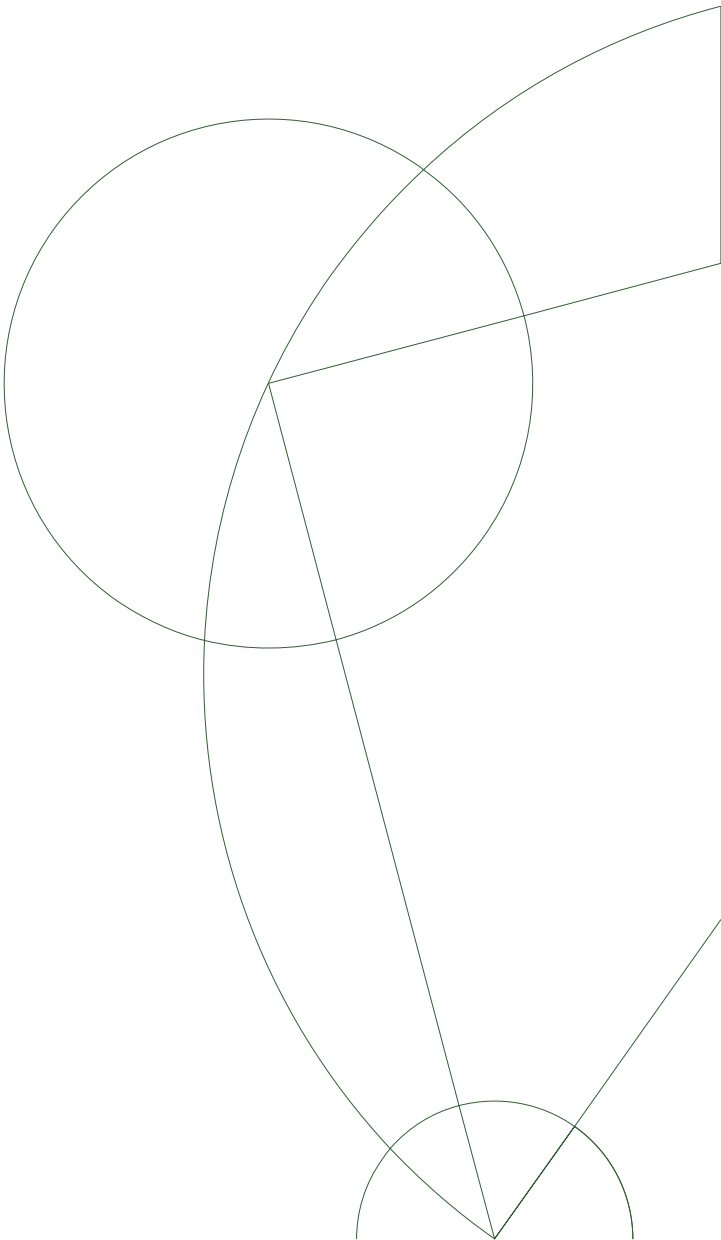
FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN

Master's Thesis

Alberte Thegler - alberte@thegler.dk

Towards formal verification of FDR4

Department of Computer Science



Professor Brian Vinter

August 2018

Abstract

Bla bla bla bla

Contents

Todo list

Chapter 1

Introduction

When we create programs, we wish to verify that it is also correct. There are several ways to do this, one commenly used is `testing` which require that the programmer creates several different scenarios and its expected output, or that the programmer programs a test-generator to create the scenarios and expected output. This, however, is not adequate for (word for important systens). Therefore it is of high interest to create a verification of the system or program.
Talk about how verification was first created and how it became to be used for concurrent systems. Then write about how it works and then write about the different systems and formal languages that is used for it.

In this thesis we look at model checking, that is, verifying that a specific property will always hold for a piece of code.

Formal verification is the process of checking whether a program satisfies specific properties. Different methods have evolved, all having different advantages and disadvantages. FDR is sometimes referred to as a model checker however is it actually a refinement checker.

Matematicians tend to reject proofs by exhaustive checking of all cases as being less satisfying than deductive proofs, and with good reason. First, they are not applicable for proving theorems about integers and real numbers, which are infinite domains so that the number of interpretations is infinite and they cannot be exhaustively checked. Second, they offer no insight into why a theorem is true. But computer scientists have more practical concerns. If they can check all computations of a program and show that they all satisfy a correctness property, we will be willing to forego elegance and be more than satisfied that our program has been proven correct. (from "A primer on model checking af Ben-Ari [1]

1.1 Learning goals

This is where the learning goals go.

Chapter 2

Related work

In this chapter we will discuss previous work that has lead formal verification to what it is today. We will also discuss different tools and languages that are used today and the differences between them.

In 1967 Robert W. Floyd was published with his paper *Assigning meaning to programs*[2]. In his paper he provide a basis for formal definitions of the meaning of programs which can be used for proving correctness, equvalence and termination. He uses flowcharts to argue that when a command is reached all previous commands will have been true as well.

C.A.R Hoare was inspired by Floyd and in 1969 his paper *An axiomtic basis for computer programming*[3] was published. He builds on Floyds ideas and proposed that program could be viewed as a partial correctness relation between a precondition and a postcondition predicate. This means that if the state the program starts in satisfies the precondition and it terminates, then the final state satisfies the postcondition. In 1972 his paper *Towards a Theory of Parallel Programming* was published and in 1978 his paper *Communicating Sequential Processes* was published. CSP was born and have been widely used and have also been expanded since Hoare initially described it in 1978.

download file and make citation

Citation to Communicating Sequential Processes: The first 25 years

The porgramming language Occam, which was first released in 1983, is a concurrent programming language that builds on the Communicating Sequential Processes process algebra. Occam developed over the years and the Kent Retargetable occam Compiler (KRoC) team at Kent University created the Occam-π variant of the occam programming language. It is a version that extends the idas of CSP in the original occam language but adding mobility features from the pi-calculus. On the KRoC webpage they describe the reason to include functionality from pi-calculus; *"Specifically, we want to allow networks of processes to evolve, to change their topologies, to cope with growth and decay without losing semantic or structural integrity. We want to address the mobility of processes, channels and data and understand the relationships between these ideas. We want to retain the ability to reason about such systems, preserving the concept of refinement."*[1]

[1] https://www.cs.kent.ac.uk/projects/ofa/kroc/ access date: 3/4/18

Chapter 3

Theory

This is where the theory go. fx. SME and the correlation between that and CSP.

3.1 Hoare's logic

Chapter 4

Method

This is the method section that describes what I did, how and why.

Chapter 5

Results and tests (Experiment?)

Does it work? why, why not

Chapter 6

Discussion

Chapter 7

Conclusion

7.1 Future work

Bibliography

[1] M. M. Ben-ari. A Primer on Model Checking. 1(1):40–47, 2010.

[2] R. W. Floyd. Assigning Meanings to Programs. pages 19–32, 1967.

[3] C. A. R. Hoare. An axiomatic basis for computer programming, 1969.

7.2 Appendix