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 Learning goals

This is where the learning goals go.

Chapter 2

Related work

Related work should be about describing the history of formal verification and formal languages in both software and hardware.

In 1969, Tony Hoare proposed the Hoare logic[?] in his paper *An axiomtic basis for computer programming.* He proposed that a 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. Hoare was much inspired by Robert W. Floyd, who had published his paper *Assigning meaning to programs*[?] a few years before. The paper described a method for proving the partial correctness of a program by using flowcharts and Hoare's logic provided a good way of formulating Floyd's method.

A lot of verification systems used today is based on Hoare logic. Hoare was much inpired by Robert W. Floyd, who a few years before had published Several different software verification tools exists today, and they all have different advantages.

SPIN is a verification system that uses process interactions to prove correctness for a system. The system is described in the formal language `PROMELA`(PROcess MEta LAnguage)and the correctness properties are spcified in Linear Temporal Logic or LTL . Spin was developed at Bell Labs, starting in 1980. Since 1991 it has been freely available and today it is used by thousands of people worldwide. The job is to find other types of models and compare them, as well as the types of languages. so different tools and different languages..

Add reference to SPIN

add reference to promela

add reference to Linear Temporal Logic

Occam-pi is a programming language that came from occam and was merged with pi-calculus.

The two most popular methods for automatic formal verification are language containment and model checking. The current version of VIS emphasizes model checking, but it also offers to the user a limited form of language containment (language emptiness)." from https://embedded.eecs.berkeley.edu/research/vis/doc/VisUser/vis_user/node4.html

Chapter 3

Theory

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

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

7.2 Appendix