Software Engineering using Formal Methods

Lecture Notes by Thomas Schulz
Last Update: February 29, 2012 - 16:25

TECHNISCHE UNIVERSITÄT DARMSTADT

Lecture held in WS 2011/12 by:
Prof. Dr. Reiner Hähnle
Dr. Richard Bubel

Contents

Disclaimer

This document is a summary of the lecture “Software Engineering using Formal Methods”. It was created for personal study and exam preaparation.

The author do not warrant or assume any legal responsibility for the accuracy, completeness, or usefulness of any information described in this document.

Motivation

Defects in Software can cause (financially) *severe* and *omnipresent* failures. Unfortunately, best practices known from other engineering disciplines are not adaptable to developing software (see Table 1).

Table 1: Hardware vs. Software

Best Practices for Hardware	**Why not for Software?**
Redundancy	Does not help against bugs!
Separation of Subsystems	Usually not (completely) possible!
Precise Calculation	Software is too complex!
Follow patterns	No mature methods in SE!
Robust Design	Local Errors often affect the whole system!

One possible approach is to test a software product, but this shows only the *presence* of errors, not their *absence*. Besides, testing is always incomplete, expensive and time consuming.

This motivates the topic of the lecture. Formal methods provide tools to verify correctness and completeness. The idea for both parts of this course is to provide a specification of a system, provide a specification of the requirements and (semi-)automatically check whether the specification meets the requirements.
The first part discusses an approach for concurrent processes while the second part adresses object-oriented programs.

1 Modeling & Model Checking with PROMELA & SPIN

1.1 PROMELA Introduction

- put variable declarations at start
- non-initialized arithmetic variables are set to 0
- the values $\mathbb{B} = \{\texttt{true}, \texttt{false}\}$ are syntactic sugar for the bit values 1 and 0
- there is at most one `mtype` *(message type)* per program
- first statement after "`::`" is considered as the *guard*
- if more than one guard is true, then one is randomly chosen
- use "->" after command that starts with "`::`", not "`;`"
- *blocking* occurs if no guard is true
- feel free to declare constants with "`#define C val`"
- there are two possibilities to express a for-loop
  1. "`for (i : 1 .. 6)`" iterates over `i` from 1 to 6
  2. "`for (i in a)`" iterates over all indices `i` of array `a`

1.2 Verifying with SPIN

1.3 Modeling Concurrency

1.4 Introduction to PROMELA/SPIN

1.5 Modeling Distribution

1.6 Propositional Logic & Temporal Logic (1)

1.7 Temporal Logic (2)

1.8 Channels & Linear Temporal Logic

1.9 Temporal Model Checking with SPIN

2 Modeling & Verification with JML & KeY

2.1 First-Order Logic (Syntax and Semantics)

2.2 First-Order Logic – Calculus

2.3 JML (1)

2.4 JML (2)

2.5 Dynamic Logic 1

2.6 Dynamic Logic Calculus

2.7 Proof-Obligations

2.8 Loop Invariants