Systemtheory 2  
Formal Systems 2  
#342201  

SS 2006  
Johannes Kepler Universität  
Linz, Österreich  

Univ. Prof. Dr. Armin Biere  
Institute for Formal Models and Verification  

http://fmv.jku.at/fs2

Introduction

Motivation

- more and more complex systems  
  Moore’s Law $\Rightarrow$ soon we will have $10^{30}$ transistors / processor  
  multi-million LOC / OS  
  $\Rightarrow$ exploding testing costs (in general not linear in system size)

- increased dependability  
  everything important depends on computers:  
  stir by wire, banking, stock market, workflow, . . .  
  $\Rightarrow$ quality concerns

- increased functionality  
  security, mobility, new business processes, . . .

Test and Verification

Test

standard definition: dynamic execution / simulation of a system  

integration in development process necessary  

extreme position: testing should actually “drive” the development process

Verification

standard definition: static checking, symbolic execution  

hardware design: verification is the process of testing

$\Rightarrow$ our view: Test = Verification

Implications

- not unusual to have more than 50% of resources allocated to testing  
- testing and verification are (becoming) the bottleneck of development  
- quality dilemma (drop quality for more features)  
- more efficient methods for test and verification needed  
  $\Rightarrow$ formal verification is the most promising approach  
- experts in new testing and verification methods are lacking  
- long term: more formal development process not just formal verification
Formal Methods

- **formal = mathematical**
- mathematical models $\Rightarrow$ precise semantics
- emphasizes static / symbolic reasoning about programs (so standard definition of verification falls into this category)
- rather narrow view in digital design: equivalence and model checking
- not esoteric: compilation in a broad sense is a formal method (high-level description is translated into low-level description)
- our view: use tools for reasoning (i.e. programs are formal entities)

Formal Methods Classification

- Synchronous
- Theorem Proving
- Model Checking
- Z
- SAT
- Specification
- Formal
- SDL
- Languages
- Compiler
- Formal
- Synthesis
- Equivalence
- Checking
- B−Method
- ASM
- Formal
- Verification

Formal Specification

- abstracts from unnecessary implementation details
- high-level mathematical model of the system
- very useful for high-level design
- catches ambiguous or inconsistent specifications
- formal specification per se: no tools for refinement / checking
- good example: ASM

Formal Synthesis

- Initial Formal Spec
- 1st Refinement
- 2nd Refinement
- 3rd Refinement
- 4th Refinement (last formal step)
- C Program
- Compiler
Formal Synthesis

- integrates verification in the development process
- usually pure top-down design and **incremental refinement steps**
- splits large verification tasks (divide et impera) …
- … but forces dramatic change in development process
- it works but is costly
- each refinement step uses formal verification methods
  \[\Rightarrow\] more powerful verification algorithms allow more automation
- good example: B-Method

Formal Verification

- assumptions: specification and system are given
- **formal verification checks formally that system fulfills specification**
- least change in development process
- full blown verification is really difficult: “post mortem verification”
- simplifications: focus on simple partial specifications
  (type safety, functional equivalence of two systems, …)
- methods (implemented in tools):
  simple algorithms for deducing properties directly
  complex algorithms for hard or even undecidable problems

Layered System Design

<table>
<thead>
<tr>
<th>HW</th>
<th>SW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Requirements</td>
</tr>
<tr>
<td>RTL</td>
<td>High-Level Design</td>
</tr>
<tr>
<td>Gate</td>
<td>Low-Level Design</td>
</tr>
<tr>
<td>Transistor</td>
<td>Implementation</td>
</tr>
</tbody>
</table>

- 1. no implementation without Synthesis
- 2. Verification is added value (Quality)
- 3. both processes are incremental
- 4. both processes can be formal

Overview

- boolean methods:
  SAT, BDDs, ATPG, Combinational Equivalence Checking
- finite state methods:
  Bisimulation and Equivalence Checking of Automata, Model Checking
- term based methods:
  Term Rewriting, Resolution, Tableaux, Theorem Proving
- Abstraction (eg SLAM uses BDDs, Model Checking, Theorem Proving)
Focus

Intro

Revision: 1.16

13

• how does it work?
  (algorithms and data structures)

• necessary background for use of formal verification
  (and formal methods in general)

• capacity and restrictions

• first step to become an expert in a fast expanding area