CSCI 234

Design of Internet Protocols: Introduction

George Blankenship

Outline

- Motivation for the class
- Verification Techniques
- Verification Approach

Motivation

- Protocols are fundamental to every aspect of computer science
  - Protocols define the actions across a system bus
  - Protocols define the actions between processes within a system
  - Protocols define the action between programs within and between systems
- Protocols are fundamental to every aspect of life

Motivation
(why study protocols)

- Communication software is becoming more and more complex
  - More and more systems are multiprocessor, distributed, and real-time
- Size of produced software has increased dramatically
- Software is executing in heterogeneous environments
- Standards in IT and telecommunications have increased tremendously
  - GSM Specifications 1306
  - 3G Specifications 2290
- Communication software is not unique to telecomm industry
  - Internet, data communication, mobile communication
- Networks convergence, interconnection & interoperability

Introduction George Blankenship 4

Motivation
(business perspective)

- Verification & validation of software is challenging
- Pressure of speed to market
- Pressure to improve quality
- Informal techniques (i.e. walk-through) for protocol testing are inadequate
- Generation/execution of test cases is time-consuming and error-prone
- Desire to make the testing process cheaper and to reuse test cases

Conclusion: *formal design & testing is necessary*.*

Introduction George Blankenship 5

Basic Rules of Protocol Design
(or any other problem)

1. Make sure that the problem is well-defined
2. Define the service to be performed at each level of abstraction (what before how)
3. Design external functionality before internal functionality (what before how)
4. Apply the KISS principle
5. Keep orthogonal elements independent
6. Keep extraneous information out design
7. Build high-level prototype and verify requirements
8. Implement the design (and evaluate implementation)
9. Verify implementation against requirements
10. Don't skip Rules 1 to 7

Holtzmann section 2.8

Introduction George Blankenship 6

Protocol

- **S**et of rules that govern the interaction of concurrent processes in distributed systems
  - Task broken into subtasks
  - Implemented separately in layers in stack
  - Functions needed in both systems
  - Peer layers communicate
- Examples
  - TCP/IP, HTTP, FTP, SMTP, UDP….
  - Telephone conversation /file transfer
    - Source must activate communications.
    - Path or inform network of destination
    - Source must check destination is prepared to receive
    - File transfer application on source must check destination file management system will accept and store file for his user
    - May need file format translation

Introduction George Blankenship 7

Case Study
(from the real world)

- *Supervisor*:
  - We are about to spend $1M for a communications software package. I want to make sure the software will be flawless.
  - OK?
- *You*:
  - Hmmm. Errr, ..., Aham, Huh? Doh!!

Introduction George Blankenship 8

Software Verification
(from the real world)

- What technique do I use?
- Which tool?
- What are the limitations?
- How expensive will it be to do this?
- What expertise do I need?
- Any special training?
- Any size limitations?
- Exhaustiveness?
- Reliability?
- Expressiveness?
- Support?
- ?

Introduction George Blankenship 9

Verification Techniques

- Models
- Simulations
- Formal Methods

Modeling

- Mathematical representation of a process (i.e. activity cycle diagrams, flow charts etc.)
  - Contain key elements of system under study.
  - Complex process decomposition
- Mathematical models are "toy" systems
  - A model represents a part of a reality
  - A model is not reality

Simulations

- Conceptual understanding reduced to algorithmic or math logic
  - Algorithm or math logic implemented in a program
  - Simulation represents a part (or a total) reality
  - Simulation is a close approximation of reality
- Stochastic (nondeterministic) models contain probabilistic element (uncertainty) in a system or process.
  - Next state of environment not fully determined by previous state of environment
- Deterministic models behave predictably.
  - Particular input, will always produce same correct output, and the underlying machine will always pass through same sequence of states.

Formal Methods

- Techniques for analyzing systems, based on some math
  - Some of the work done informally, due to complexity.
- Deductive verification (…more later):
  - Using logical formalisms, prove that the software satisfies its specification. Eg.: Floyd - Hoare Logic, Type checking
- Model checking
  - Using software to automatically check that the software satisfies its specification.
- Testing
  - Checking executions of the software according to coverage schemes.

Introduction George Blankenship 13

Concerns

- Formal methods can only be used by mathematicians.
  - *They are based on math, but anybody can just use them.*
- Verification process is itself prone to errors, why bother?
  - *We attempt to reduce the errors, not eliminate them.*
- Formal Methods slow down projects.
  - *Maybe they speed them up, if errors are found earlier*
- Automatic verification can always find errors.
- Deductive verification can show that the software is completely safe.
- Testing is the only industrial practical method.
- What are the problems with simulations ?

Introduction George Blankenship 14

Verification Approach

- Learn several methods (deductive verification, model checking, testing, process algebra).
- Learn advantages and limitations, in order to choose the right methods and tools.
- Learn how to combine existing methods.

Introduction George Blankenship 15

Key Components of Verification

- The process:
  - Selecting the tools, modeling, verification, locating errors.
- Use of tools:
  - Hands on. DOVE, Isabelle or HOL, SPIN
- Visual notation:
  - State charts, MSCs, UML.

Case Study

(Where do we start?)

- *Supervisor*:
  - Can you verify this for me?
- *You*:
  - OK, first I have to ...

Verification Process

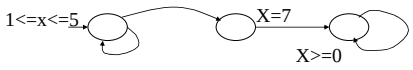
- Check the kind of software to analyze.
- Choose methods and tools.
- Express system properties.
- Model the software.
- Apply methods.
- Obtain verification results.
- Analyze results.
- Identify errors.
- Suggest correction.

Software Type and Approach

Software Type:	Aspect Specified:
Sequential.	Protocols.
Concurrent.	Abstract algorithms.
Distributed.	
Reactive.	Finite state.

Specification: Informal, textual, visual

- Informal definition:
  - The value of x will be between 1 and 5,
  - until some point where it will become 7.
  - In any case it will never be negative.
- Textual specification (formal):
  - **(1<=x<=5 U x=7) /\ [] x>=0**
- Visual specification (formal):



Verification methods

- Finite state machines
  - Apply model checking
- Deductive verification
  - Apply theorem proving
- Program too big, too complicated
  - Apply testing techniques.
- Apply a combination of the above!

Modeling - Different Approaches

- Use the program text.
- Translate to a programming language embedded in some proof system.
- Translate to some notation (transition system).
- Translate to finite automata.
- Use visual notation.
- Special case: black box system.