


CS 5219 Introduction

Abhik Roychoudhury
School of Computing
National University of Singapore


CS5219 2007-08 by Abhik 1



What is it about ?

- Techniques to help reliable software development.
- Checking program **behavior**
 - Typically checking whether desired invariants hold at program control points.
- What is the programming language ?
 - Conventional languages like C/Java
 - Deeper issues remain ...


CS5219 2007-08 by Abhik 2



What kind of programs ?

- Conventional sequential programs
 - C like programs
- **Multi-threaded** software for distributed sys.
 - E.g. Multi-threaded Java
 - Many behaviors due to thread interleaving
- **Reactive** software
 - In continuous interaction with environment
 - e.g. control software in embedded sys.


CS5219 2007-08 by Abhik 3



Conventional development

- **Collect software requirements**
 - Programmers often do not collect complete sets of requirements.
- **Write code**
 - Good programming disciplines exist e.g. modular development
- **Debug**
 - Code walkthrough, Peer review, Testing
 - Again informal and/or incomplete.

CS5219 2007-08 by Abhik 4




So are we ...

... going to look at program debugging ?

- YES
 - All our validation techniques can be used for software debugging
- NO
 - We will not only look at conventional software engineering activities like testing.

CS5219 2007-08 by Abhik 5



Why bother ?

- Testing etc. is incomplete.
 - Checking program behavior for a specific execution
 - No guarantees about program behavior
 - **safety critical systems**
 - Brake controller software of your car
 - Substantial effort spent anyway in generating "good" test cases, ensuring "good" coverage.

CS5219 2007-08 by Abhik 6

Spectrum of Techniques

- **Static checking techniques**
 - Model Checking
 - Deductive proof techniques (e.g. Induction)
- Dynamic checking techniques
 - Monitoring, Invariant Detection
- Conventional debugging
 - Testing, Slicing (how to link with validation techniques)
 - Fault Localization

CS5219 2007-08 by Abhik

7

Static Checking

- Analyze program source code to establish invariants at control locations
 - Automated techniques
 - Deductive techniques
- Deductive techniques similar to constructing a proof of correctness by hand.
 - Involves guessing and proving **loop invariants** for loops in the program
 - Proof Assistants available to help mechanization.

CS5219 2007-08 by Abhik

8

Differences via Example

- For (i = 1; i < 10; i++) {}
- How to prove $i > 0$ always ?
 - Model checking
 - Generate a transition system whose states are
 - (Control Loc, Value of i)
 - Traverse the transition sys. to verify that $i > 0$ in all reachable states of the transition system.

CS5219 2007-08 by Abhik

9

Differences via Example

- For (i = 1; i < 10; i++) {}
- How to prove $i > 0$ always ?
 - Theorem Proving
 - Prove by induction on the iterations of the loop.
 - Static Analysis
 - Infer possible values of i at each control location (irrespective of how they are reached).
 - Check that all possible values are > 0

CS5219 2007-08 by Abhik

10

Automated Static Checking

- Difficulties in automation
 - Reasoning about infinite domains and structures in the memory store of the program
 - Reasoning about aliases in the memory store
 - Array indices
 - Pointers
- How to surmount these problems ?
 - **Abstract the memory store** (to a finite structure ?)

CS5219 2007-08 by Abhik

11

Model Checking

- Abstraction is designed for a specific program.
- Used for checking complex temporal properties (safety, liveness, response properties).
- User may have to dabble in constructing abstract model, in general.
 - Canonical abstractions (data abs.) available.
- Search based exact procedure at a certain level of abstraction
 - Provides detailed counter-example evidence.

CS5219 2007-08 by Abhik

12

Model Checking

- Inputs:
 - finite state transition system (implementation)
 - Temporal logic formula (specification language)
- Output:
 - True if the specification holds
 - A **counterexample behavior** if it does not
- Technique:
 - Implementation FSM is a finite graph.
 - Unfold and search this finite graph to check all behaviors.

CSS219 2007-08 by Abhik

13

Use of Model Checking

- Generate finite-state transition system like models from C/Java code
- Employ search on this model to verify invariants or other properties.
- If counter-example obtained by MC
 - Need to locate the bug from counterexample

CSS219 2007-08 by Abhik

14

An Example

- $x = 0; x = x + 1; x = x + 1;$
- if $(x > 2)\{ \text{error} \}$
- Is the error reachable ?
- Problem: domain of x is not finite

CSS219 2007-08 by Abhik

15

Step 1: Label the locations

- L0: $x = 0;$
- L1: $x = x + 1;$
- L2: $x = x + 1;$
- L3: if $x > 2$
- L4: error

CSS219 2007-08 by Abhik

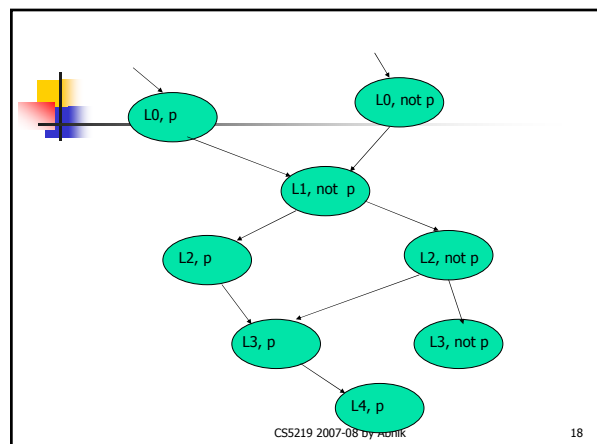
16

Step 2: Abstract x

- The finite state transition system generated for the abstraction $\{x > 2\}$ is constructed. Use shorthand $p \equiv x > 2$. This finite state transition system shows the reachability of location L4.
- Do this now
 - How did we get $x > 2$??

CSS219 2007-08 by Abhik

17



CSS219 2007-08 by Abhik

18

Step 3: Construct TS & check

- We find 1 or more counter-examples
- Use them to refine abstraction
 - Heuristics!
- Example:
 - $(L0, p), (L1, \neg p), (L2, p), (L3, p), (L4, p)$
 - Only remembering $p = (x > 2)$
 - Need to keep track of more information?

CS5219 2007-08 by Abhik

19

Dynamic Checking

- Monitoring amounts to run-time checks **during** program execution.
 - Testing checks program traces during program development, not at run-time.
- Other run-time techniques try to infer bugs by detecting a deviation from "normal" behavior as a potential bug.
 - Needs to be confirmed by user.
 - **Constructing program model based on observable traces.**

CS5219 2007-08 by Abhik

20

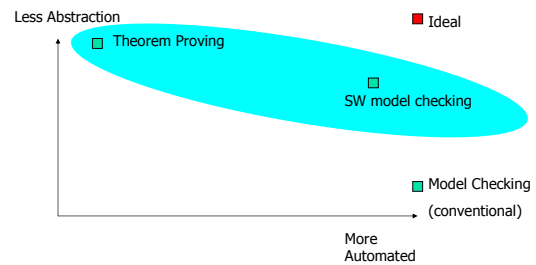
Debugging via Slicing

- Slicing
 - Input: A var. V at control location L
 - Output: Part of the program code which affects the value of V at location L
- Can be static or dynamic
 - Static: Part of code which affects V at L **for some exec**
 - Dynamic: **for a particular exec**
- Give explanations of problematic executions (which are detected by validation techniques)

CS5219 2007-08 by Abhik

21

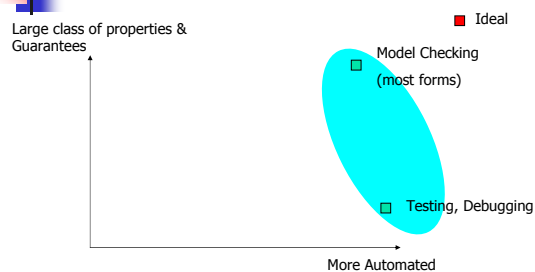
Checking Techniques



CS5219 2007-08 by Abhik

22

Checking Techniques



CS5219 2007-08 by Abhik

23

End Goal of the course

- Familiarity with host of debugging/verification techniques **beyond testing**.
- **Techniques help locate hard-to-detect bugs.**
- Focus is on bug hunting (pragmatic) rather than proving systems correct (quest of a theoretician).

CS5219 2007-08 by Abhik

24

Do I have the background ?

- The following is what we will need
 - UG course in Programming Languages
 - Understanding of algorithms (we will use search algorithms from time to time).
 - Familiarity with languages like C, Java etc
 - Interest in programming and developing reliable code
- The last point is the most important !

Assessment

- Midterm : 25 %
- Project : 25%
- Final Exam : 50 %

- Exams will be open book.

Sample Overview Readings

- *Software Analysis and Model Checking*, Gerard Holzmann, 2002.
- *Verification of Embedded Software: Problems and Perspectives*, Patrick and Radhia Cousot, 2001.
- *Automatically validating temporal safety properties of interfaces*, Thomas Ball and Sriram K. Rajamani, 2001
- *Trends in Software Verification*, Gerard Holzmann, 2003.

IVLE

- Lesson Plan
 - Updated every week
 - Weekly lectures and readings available here
- Discussion Forum
 - Post messages for query, discussion.
- Workbin
 - Submissions (e.g. Midterm reports)
 - Other handouts also made available here.

Dates, times

- Lecture: Friday 6:30 – 8:30 PM
 - COM1 #02-12
- Consultation
 - Drop by, or send e-mail.
 - My office is COM1 #03-20
- Midterm
 - Week 7 in class
- Any administrative questions ?

Course Outline (First Half)

- Introduction (Lecture 1)
- Systematic Software Debugging (Lecture 2)
 - > Tools: JSlice slicing tool
- Protocol/Software modeling (Lecture 3)
 - > Promela language in SPIN tool
- Property Specifications for checking (Lec 4)
- Model checking algorithms (Lec 5)
- Model checking tool (Lec 6)
 - > SPIN tool for model checking

Course Outline (Second Half)

Midterm (Lec 7)

Software Abstractions (Lec 8, 9)

-> Primarily abstracting data values to question/ans.

Deductive Verification – Hoare Logic (Lec. 10)

Deduction verification tools (Lec 11)

-> Sample usage of PVS tool

Software Testing strategies (Lec 12)

Project Presentations (Lec 13)

Discussion on Projects (1)

- Can be a substantial case study
 - Choose a protocol or software
 - Verify it using SPIN model checker
 - Covered in class early in the course
 - Feel free to use other tools also, if you are more familiar with them already.
 - Write a report sharing your experience and the verification results.

Discussion on Projects (2)

- ... Or a survey
 - Choose a cutting edge issue in software validation
 - Please drop by for a discussion.
 - Survey of existing literature.
 - Discussion of possible future work.

Project schedule

- Midterm Report
 - Due in 8th week (1 week after midterm)
- Project Presentation
 - On 13th week in class
- Final Report
 - After last lecture.

Project Guidelines

- Individual or group of 2 ??
- I will provide an initial list of possible case studies and survey areas.
- We will discuss the project progress at regular intervals.

■ **THANK YOU.**