Static Program Checking

Introduction

Jun.-prof. Dr. Mana Taghdiri

Thursday – April 24, 2014
Administrative notes

- Lecturer
  - Mana (mana.taghdiri@kit.edu, Geb. 50.34, Room 229)
  - Office hours by appointment

- Class material
  - Recent research papers
  - Will practice with the tools whenever possible (bring your laptops)
  - Exchange of ideas (the more interactive, the better)

- Exam
  - Part of the ‘formal methods’ module
  - Oral exam
Contents Overview

- Class focuses on systematic bug-finding techniques
  - Emphasis on cost, practicality, and automation
  - Push-button techniques
  - In contrast to verification approaches
    - E.g. theorem proving

- Announced topics
  - Finding bugs in OO programs statically
    - As opposed to testing
  - Inferring what programs do
    - Summaries
      - Static techniques
      - Invariants
        - Static and dynamic techniques
    - Iterative analysis via feedback loops
Approach

- Flexible about the topics
  - Will adjust based on your feedback

- If interested in such topics
  - Diploma/masters thesis
  - student work
  - Discussions

- Check out the website regularly
  - [http://asa.iti.uka.de/](http://asa.iti.uka.de/)
  - For the list of references, schedule, and slides
Traditional testing is not cost-effective

- Zero-tolerance for bugs in safety-critical software
  - Air-traffic controllers, medical equipments, automotive industry, etc.
- Pressure to reduce time-to-market

- Testing is easy
  - Few first tests reveal many quick bugs
  - Tests are usually run automatically and repeatedly
- Testing is incomplete
  - Requires domain experts to pinpoint troubling scenarios
- Testing is costly
  - Consumes half the total cost of software development
  - Microsoft hires one tester for every developer
Automatic test-case generation

- **Exhaustive** generation
  - Test cases generated for a method based on its pre-condition
  - All non-isomorphic test cases up to a certain size
  - Runs the code on generated tests and compares against the post-condition
  - Either declarative (based on Alloy) or imperative algorithm

- **Random** generation
  - But “feedback-directed”
  - Randomly selects which method to call next and its arguments from available objects
  - Executes generated tests and uses the feedback to generate better tests
  - Execution results determine whether the input is redundant, illegal, contract-violating, or useful for generating more inputs

- Automated test generation is a solution, but not our topic!
Cost vs. confidence

cost

confidence

testing
Cost vs. confidence

- Testing
- Theorem proving
Cost vs. confidence

- theorem proving
- bounded model checking
- testing
Cost vs. confidence

- theorem proving
- bounded verification
- bounded model checking
- testing
Cost vs. confidence

confidence

bounded verification
bounded model checking

testing

theorem proving

cost
Static software checking

- Checks a functionality of the code *(property)*
  - Provided by the user
  - Says what the code is supposed to do

- Provides certainty for program correctness *(confidence)*
  - What kind of properties does it check?
  - How complete is the analysis?

- Requires efforts from users *(cost)*
  - Code preparations before the analysis?
  - User interaction during the analysis?
  - Understanding the reported bug?
  - False alarms?
  - Analysis time?
Inferring what programs do (Examples)

- **Summarization**
  - Static
  - Syntactic specifications in Alloy
  - Infers post-conditions based on pre-state values
  - Good for OO code
  - Based on symbolic execution and abstract interpretation

- **Invariant** detection (Daikon)
  - Dynamic
  - A machine learning technique
  - Properties that hold at a certain point in the program
  - Unsound, but likely
  - Runs on a suite of test cases and learns invariants

- Why is invariant detection/summarization important?
Iterative analysis via feedback loops

Program behavior → Property ?
Iterative analysis via feedback loops

Abstract behavior

Program behavior

Property ?
Iterative analysis via feedback loops

Abstract behavior

Program behavior

Property

YES! ✔
Iterative analysis via feedback loops

Abstract behavior

Program behavior

Property

YES! ✔

NO …
Iterative analysis via feedback loops
Iterative analysis via feedback loops

Abstract behavior

Program behavior

Property ?
Iterative analysis via feedback loops
Iterative analysis via feedback loops

Abstract behavior

Program behavior

Property

YES! ✅

NO …
Iterative analysis via feedback loops

Abstract behavior

Program behavior

Property

YES!

NO …
Iterative analysis via feedback loops
Iterative analysis via feedback loops

Abstract behavior

Program behavior

Property

YES! ✓
Iterative analysis via feedback loops

Counterexample-guided Abstraction Refinement (CEGAR)
Alloy

- Invented by Daniel Jackson at MIT in 2000
  - http://alloy.mit.edu/community/

- A modeling language

- **Declarative**
  - As opposed to *imperative*
  - Describes the logic of a computation without describing its control flow
  - Example
    -Sorting

- Common declarative languages
  - Regular expressions
  - Logic programming (Prolog)
  - Functional programming (ML)
Other modeling languages

- **JML, OCL**
- **Larch**
  - Developed in 1980s
  - Good for concurrent programs and algebraic datatypes
  - Based on theorem proving
  - Not fully automatic, but good for its time
- **Z**
  - Based on the simple notions of set theory
  - But even less analyzable than Larch
- **SMV language**
  - Model checker
  - Checked a billion states in seconds with no aid from user – explicit
  - Formal methods became fashionable overnight
  - Widely used for hardware
  - Language not suitable for structure-rich software
Alloy

Motivation
- Brings the SMV-like automation to a Z-like language

For writing succinct and precise descriptions of
- Software systems (design level)
  - Pick the right design, implementation follows naturally
  - Check properties before committing to code
  - Build a model incrementally, simulate and check as you go along
- Program behavior (implementation level)
  - Check properties before delivering the software

Applications
- File system analysis
- Network protocols
- Course scheduler
Alloy

- Efficient for describing structures
  - Network topology
  - Program data structures
- Can be analyzed automatically
  - Research tool, but very well supported
  - Useful library functions, sample models
- Analysis technique
  - Nothing like model checkers of that time
  - Translates constraints to boolean formulas and uses SAT solver
  - Exploits off-the-shelf solvers
  - Now model checkers translate to SAT too
- Both as
  - Environment for checking correctness by manual modeling
  - Engine for checking correctness by automatic modeling