Program Analysis and Code Verification

http://d3s.mff.cuni.cz

Pavel Parízek

Department of Distributed and Dependable Systems

FACULTY OF MATHEMATICS AND PHYSICS
Charles University
Language

- Lectures: English
- Labs: English
- Homework: Czech/English
- Final exam: Czech/English
- Questions: Czech/English
Software bugs and errors

- Race condition
- Deadlock
- Null pointer dereference
- Array index out of bounds
  ...
- Firefox crashes
- Blue screen of death
  ...
- Train accident
Why bugs matter?

- Mission- and safety-critical systems
  - Industry: robots, assembly lines
  - Transportation: cars, trains, airplanes

- Embedded systems
  - Mobile phones, tablets, household appliances, consumer electronics
Detecting bugs

- Software testing is not enough
  - Pros: scalable, precise, well-established (industry)
  - Cons: very expensive (people, money), selected executions, bugs depend on thread interleaving

- Program verification
  - Pros: coverage, multi-threaded programs
  - Cons: precision, scalability, performance
Tools

  - exhaustive state space traversal of Java

  - systematic testing of multi-threaded programs in C#

  - systematic testing of multi-threaded programs in C#

- **KLEE** ([http://klee.github.io/](http://klee.github.io/))
  - symbolic execution for low-level C programs (e.g., Linux binutils)

- **CBMC** ([http://www.cprover.org/cbmc/](http://www.cprover.org/cbmc/))
  - bounded model checking for system programs in C and C++

  - behavior specification language for C# + abstract interpretation

  - programming language with built-in support for verification (based on Spec#)

  - static analysis frameworks/libraries for Java and C/C++

- **Infer** ([http://fbinfer.com/](http://fbinfer.com/))
  - static analysis and bug-finding tool for Java, C/C++ and Objective-C
Goals of the course

- Show algorithms and tools for program analysis, verification, and bug detection
- Practical experience with selected tools
Why you should attend

- Basic knowledge of the main program analysis and verification techniques
  - Key aspects: scalability, coverage, automation, ...

- Current state of the art
  - How good or bad the tools are
Program

- Model checking of programs
- Detecting concurrency errors
- Symbolic execution
- Dynamic analysis
- Deductive methods (SAT solvers, SMT solvers)
- Bounded model checking
- Predicate abstraction and CEGAR
- Selected applications of deductive methods in software verification
  - Verification of program code against contracts
- Static analysis and its usage in program verification
- Abstract interpretation
- Combination of verification techniques
- Program termination
- Program synthesis and repair
Theoretical limitations
Know your enemy !!
Know your enemy !!

Kurt Gödel
(1906-1978)

Alan Turing
(1912-1954)
Know your enemy!!

Completeness theorem
\[ T \models f \implies T \vdash f \]

Incompleteness theorem
For "interesting" theories \( T \)
\[ \exists f : ( T \not \vdash f ) \land ( T \not \vdash \lnot f ) \]

"Halting problem is undecidable"
What do they really say?

**Completeness theorem (CT)**

\[ T \vDash f \Rightarrow T \vdash f \]

**Incompleteness theorem (IT)**

For “interesting” theories \( T \)

\[ \exists f : ( T \not\vDash f ) \land ( T \not\vdash \neg f ) \]

**Claim:**

The completeness and incompleteness theorems contradict.

1) Let’s take \( f \) from IT

2) Any \( f \) either holds or not:

\[ ( T \vDash f ) \lor ( T \vDash \neg f ) \]

3) From CT follows:

\[ ( T \vdash f ) \lor ( T \vdash \neg f ) \]

4) Contradiction
What do they really say?

Completeness theorem (CT)
\[ T \models f \implies T \vdash f \]

Incompleteness theorem (IT)
For “interesting” theories T
\[ \exists f: ( T \notmodels f ) \land ( T \notmodels \neg f ) \]

1) Let’s take f from IT
2) Any f either holds or not:
\[ ( T \models f ) \lor ( T \notmodels \neg f ) \]
3) From CT follows:
\[ ( T \notmodels f ) \lor ( T \notmodels \neg f ) \]
4) Contradiction
What do they really say?

Completeness theorem
\[ T \models f \implies T \vdash f \]

Incompleteness theorem
For “interesting” theories \( T \)
\[ \exists f: ( T \not\models f ) \land ( T \not\models \neg f ) \]

Claim:
The completeness and incompleteness theorems contradict.
“Halting problem is undecidable”

Claim:
Given a program \( A \) and input data \( D \), you can never decide whether \( A(D) \) terminates or not.
What do they really say?

“Halting problem is **undecidable**”

**Claim:**

Given a program $A$ and input data $D$, you can never decide whether $A(D)$ terminates or not.

Sometimes you can. Consider:

```c
void main() {
    printf(“Going to halt right away!\n”);
}
```
"Halting problem is **undecidable**"

**Claim:**

You can **never** construct a general algorithm that would **for any** program $A$ and **any** input data $D$ **always** answer YES if $A(D)$ terminates.
What do they really say?

“Halting problem is **undecidable**”

**Claim:**

You can never construct a general algorithm that would for any program $A$ and any input data $D$ always answer YES if $A(D)$ terminates.

Yes, you can (but it may not terminate). Consider:

```c
void main(program A, data D) {
    ... simulate A(D) ...
    printf("YES");
}
```
What do they really say?

“Halting problem is **undecidable**”

Claim:
There is no general algorithm that would always terminate and solve the halting problem for all programs and all inputs.
Consequences

- Program verification (analysis) is **undecidable**
  - Example: assertion checking for multi-threaded programs with procedures

- But, in practice, ...
  - Many interesting properties can be successfully verified for many interesting programs
Consequences

• It may take very long
  - Out of reach of current hardware and user patience
  - More than the expected age of the known universe
  - Definitely past the hard deadline of your project

• But there is still hope
  - Full verification is not always necessary
  - Search for errors (detect some bugs)
Grading

- Five homeworks
  - Each will be awarded with 0-20 points
  - No. 5: presenting research publication

- Final exam (voluntary)
  - Awarded with 0-25 points
  - Basic principles (algorithms, theory)
  - Comparing different techniques

**Result**
- 85-125 → excellent
- 72-84 → very good
- 60-71 → good
Homework assignments

- Deadlines are strict
  - We will deduct 10% of your points total for every calendar day your assignment is late
- You have to do homework no. 5 (presentation) and two other to get “zápočet”

- Topics
  - Java Pathfinder
    - Implement custom modules and verify given program
  - Contracts languages (Dafny, ...)
    - Write specification for given program and then verify it
  - Static analysis
  - Finding real bugs
  - Presentation of research publication
    - Group homework (2-3 people)
Be active during lectures and labs!!

- Participate
- Answer questions
- Think deeply
Contact

- Web: http://d3s.mff.cuni.cz/teaching/nswi132
- Email: parizek@d3s.mff.cuni.cz
- Room 309
We are hiring

- Master thesis
- PhD studies

- Theory + Implementation

- Program verification, analysis, synthesis
- Debugging, tool support for developers
- Programming languages, concurrency
- Java, C/C++, C#, PHP, JavaScript, Scala