CS357 Software Verification

Module Overview

Lecturer:Rosemary Monahan

18 September 2017

Module Overview: Outline

- Introduction
- Background
- CS357 Administration Details

Module Overview: Outline

What is “Software Verification”?

Software Verification means

proving that a piece of software is correct

- i.e. that, when run, it does what it is supposed to do.

Consider the following method...

```
public static int m(int n)
{
    int x = 1;
    int i = n;
    while (i != 1)  {
        x = x*i;
        i--;
    }
    return x;
}
```

What does it do? Is it correct? How can you convince yourself? others?

What is “Software Verification”?

- Software Verification means proving that a piece of software is correct *with respect to its specification*.
- The specification is a statement of what the program is supposed to do.

Consider the following method...

```
public static int factorial(int n)
{
    int x = 1;
    int i = n;
    while (i != 1)  {
        x = x*i;
        i--;
    }
    return x;
}
```

What does the *specification* look like?

What is “Software Verification”?

- Software Verification means proving that a piece of software is correct with respect to its *formal* specification.
- The *formal* specification is a statement of what the program is supposed to do, written in some *formal* language.

What is “Software Verification”?

- Software Verification means proving that a piece of software is correct with respect to its *formal* specification.
- The *formal* specification is a statement of what the program is supposed to do, written in some *formal* language.
- Typically, this formal language is some form of logic (with a bit of maths).
- So a permanent theme of this module is *the role of logic in software engineering*.

Motivations for software verification.

When you might want to verify your code:

- Safety-critical software e.g. airplanes, hospitals, nuclear power plants, ...
- Programs with expensive failure consequences: e.g. space missions, stock exchange, ...
- Legal obligations: as as software engineer, how would you show in court that you'd done everything possible to ensure the correctness of your program?

Testing vs. verification?

Program testing can be used to show the presence of bugs, but never to show their absence!

- Edsger W. Dijkstra (1930-2002)

Module Overview: Outline

- Introduction
- Background
- CS357 Administration Details

Software verification in CS & SE

Software verification is:

- based on the study of *formal methods*: mathematical techniques for the specification, modelling, construction and verification of programs.
- a software engineering subject that draws heavily on *computer science*
- founded on logic and automata theory, and closely related to the *theory of computation* and the *formal semantics of programming languages*.

Software Verification: Beginnings

"How can one check a routine in the sense of making sure that it is right?"

Dr A Turing , 1949

Software Verification: Beginnings

Earliest example of software verification:

Checking a large routine by Alan Turing in 1949 outlined a verification strategy using a factorial program as an example.

See: *An early program proof by Alan Turing*, L. Morris and C. B. Jones, Annals of the History of Computing 6(2) pp. 129-143 (1984).

Software Verification: History

- 1947 Goldstine/v. Neumann: flow diagrams & assertions
- 1949 Turing: flow diagrams & assertions
- 1963 McCarthy: mathematical logic & computation
- 1966 Naur:general snapshots as comments relating variables
- 1967 Floyd: flow diagrams & assertions
- 1969 Hoare: derivation system for valid triples
- 1976 Dijkstra: functions & schemas for valid triples
- 1981 Gries: science of programming
- 2003 Hoare: verifying compiler as a grand challenge

Turings Flowchart in the style of Floyd (1967)

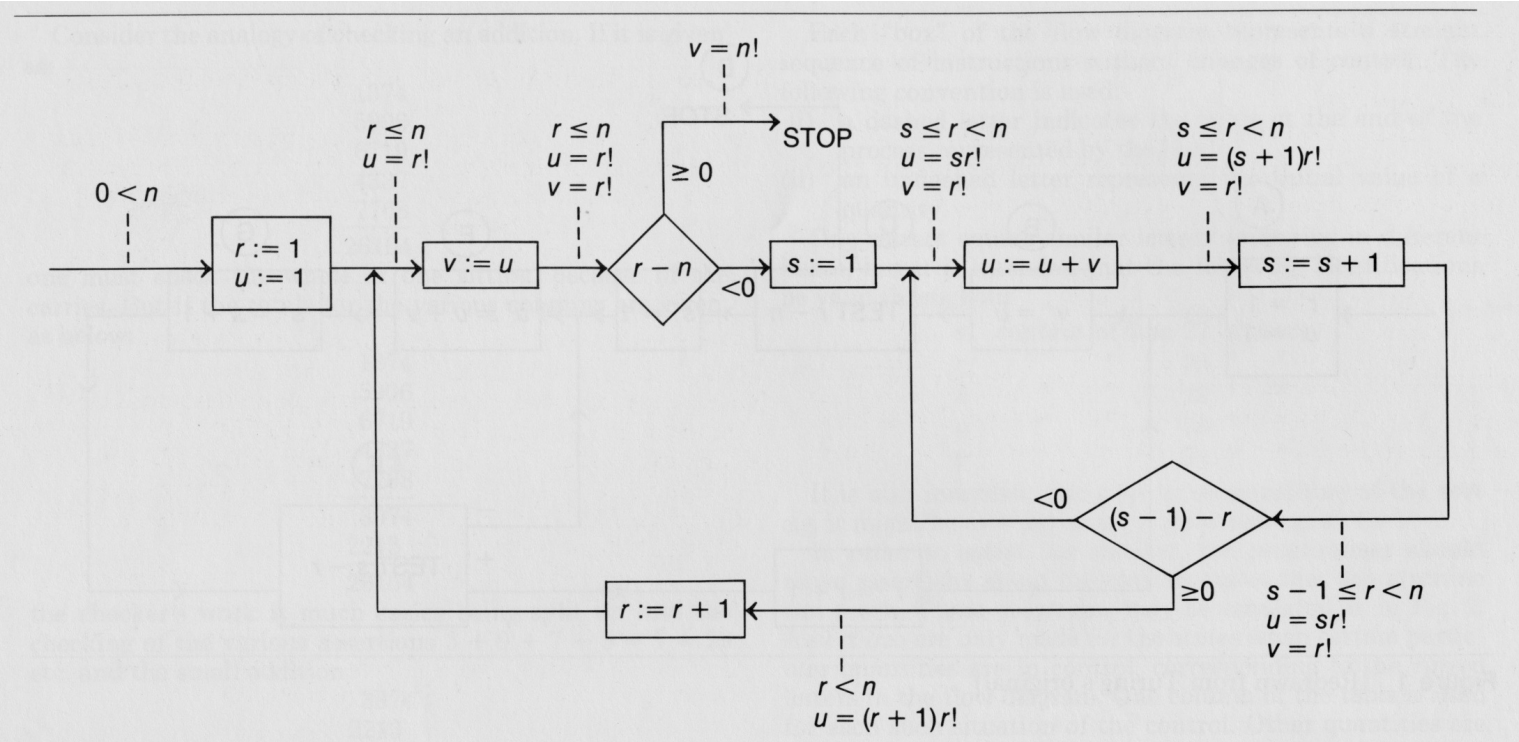
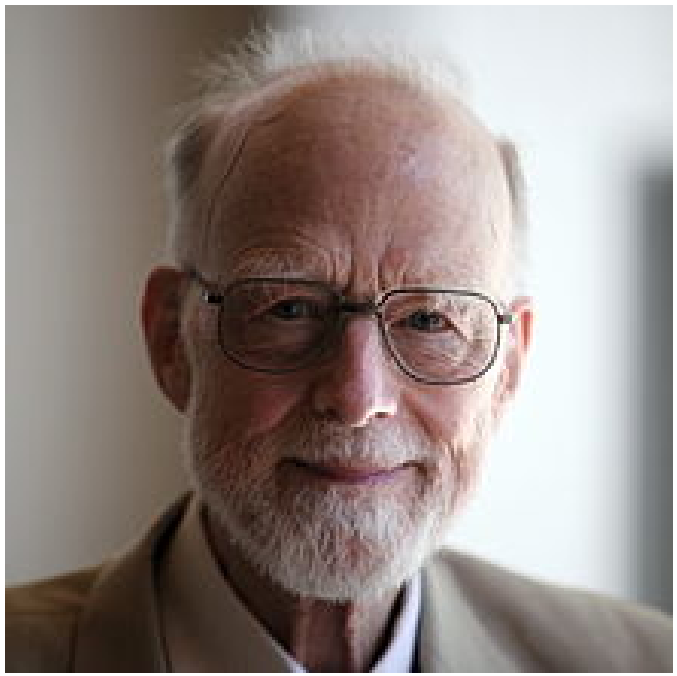


Figure A

Software Verification: History

- Influential paper: *An axiomatic basis for computer programming*, C.A.R. Hoare, Communications of the ACM, 12:576-580, 1969.
- "Popularised" by
  - *A Discipline of Programming*, E.W. Dijkstra, Prentice-Hall, 1976.
  - *The Science of Programming*, David Gries, Springer-Verlag, 1981.
- Main approaches: specification *languages* such as VDM, Z, OBJ - strong emphasis on formal proof.

The fully “formal” viewpoint



Programming is a mathematical activity ... [whose] successful practice requires the determined and meticulous application of traditional methods of mathematical understanding, calculation, and proof.

- C.A.R. Hoare

Software Verification: Move to Pragmatism

A strong reaction (1980s) against “purist” view lead to more pragmatic approaches:

- Bertrand Meyer pioneered *Design by Contract* in the Eiffel programming language
- An emphasis on temporal logic and *model checking*: SMV, SPIN
- More recently, *lightweight formal methods* advocates a blend of formality and testing

Software Verification

Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification weâĂŹre building tools that can do actual proof about the software and how it works in order to guarantee the reliability.
Bill Gates, April 18, 2002.

Software Verification: Recent Achievements

- The SLAM Project at Microsoft uses software verification techniques for checking windows device drivers
  `http://research.microsoft.com/slam/`
- Java PathFinder developed at the NASA Ames Research Center uses model-checking to find bugs in Java programs
  `http://javapathfinder.sourceforge.net/`
- Edmund Clarke (Carnegie Mellon), E. Allen Emerson (University of Texas at Austin) and Joseph Sifakis (University of Grenoble) win the 2007 A.M. Turing Award for their work on model checking

Software Verification: Today

Deductive Verification tools:

- Input: Program and Specification
- Generated: Verification Conditions
- Automated where possible: Proof
- Output: Error reports if the input program does not meet the input specififcation

Software Verification: Today

There are many different tools in the verification system landscape where a human user contributes in two ways:

- formalising an informally stated specification for a program
- providing (if necessary) guidance to a verification system to show formally the conformance of the program to the specification.

So what would Turings proof of factorial look like using one of these tools?

http://www.rise4fun.com/SpecSharp/i7HE

Software Verification: The future

- In 2005 Tony Hoare launched the Grand Challenge for computer science:

  > "the time is ripe to embark on an international Grand Challenge project to construct a program verifier that would use logical proof to give an automatic check of the correctness of programs submitted to it."

- Analogy with:
  - Hilbert's Programme for foundation of mathematics in the 1920s
  - the lunar challenge in the 1960s
  - the Human Genome project (1991-2004)

Module Overview: Outline

CS357 Prerequisites

- The main prerequisite is a good knowledge of programming
  (We'll assume Java and an ability to learn a little C#)

- The following would also be helpful:
  - Logic (e.g. Discrete Structures)
  - Automata theory (just a little)

CS357 Textbook

- Logic in Computer Science: modelling and reasoning about systems
  by Michael Huth and Mark Ryan,
  (Second edition),
  Cambridge University Press, 2004, ISBN: 052154310X.

We'll be mainly concentrating on chapters 1, 2, 3 and 4.

Moodle: Additional information will be available through the NUIM Moodle site.

Topics

1. Specification (e.g.design-by-contract, logic)
2. Verification (e.g. Hoare logic)
3. Tools for Verification (e.g. Spec#, OpenJML, Dafny, Why3, Event-B)
4. Theorem Proving (e.g. Coq)
5. Model checking (e.g. NuSMV, SPIN)

With thanks to Dr James Power for developing the course curriculum and generation of resources.

Some software we'll use in CS357

Spec# from Microsoft Research - uses Hoare logic and invariants to prove correctness for C# programs.

http://research.microsoft.com/SpecSharp

The Coq proof assistant from INRIA - uses higher-order constructive logic to allow you perform formal proofs

http://coq.inria.fr/

The NuSMV model checker originally from CMU - uses state machines and temporal logic to specify software behaviour.

http://nusmv.irst.itc.it/

CS357 Assessment

1. 30% Continuous Assessment via class exams.
   - Short class exams held weekly in the lecture or lab(weeks 3-12) focusing on specification and verification.
   - Best 7 results form CA result.
2. 70% Written Exam in January 2018.
   - Focus on specification and verification.
   - Note that the exam paper format will change from previous years - a sample paper will be provided.