Software Eng. 2F03: Logic For Software Engineering

Dr. Mark Lawford
Dept. of Computing And Software,
Faculty of Engineering
McMaster University

Motivation

Why study logic?

- You want to learn some “cool” math.
- You want to make the world a better, safer place.
- You want to save Intel$^{\text{TM}}$ a lot of money.
- You want to make a lot money yourself!

Cool Mathematics?

Halting Problem - It is impossible to write a diagnostic program that will tell you if arbitrary programs terminate.

Gödel's Incompleteness Theorems state that any formal system that includes arithmetic is either:

Incomplete - there are some things that are true that can't be proved,

OR

Inconsistent - contains one or more contradiction that allow you to prove things that are false.

All mathematicians are humbled in the face of Gödel!

You want to make the world a better, safer place?

Safety Critical Systems

Failure results in:

- physical injury or loss of life
- unacceptable financial loss

Applications Areas:

- Medical equipment
- Aerospace
- Process control - e.g. Darlington Nuclear Generating Station Shutdown Systems (SDS)

NOTE: You have one chance to get it right!

Example Software System Failures:

- Medical equipment -

  THERAC-25 radiation therapy machine killed several patients

- Aerospace -

  Space shuttle - 1st flight delayed timing bug at initialization

  Ariane 5 launcher - 1st flight self-destructed after 45 seconds due to floating point overflow error in Inertial Guidance System

Other software related problems:

Process control - e.g. Nuclear Generating Station Shutdown Systems (SDS)

- Spurious trips cost $$$
- Difficult to make modifications & even more difficult to get regulatory approval for changes

Example: Reactor Shutdown System (SDS)

What is an SDS?

- watchdog system that monitors system parameters
- shuts down (trips) reactor if it observes “bad” behavior
- process control is performed a separate Digital Control computer (DCC) - not as critical

Consider one simple “trip”:

- monitors plant parameters (Primary Heat Transport Pressure & Reactor Power) using sensors & A/D conversion
- if parameters exceed set-points in particular way, shutdown (trip) the reactor

Safety/Performance Considerations:

- Check for short circuits/sensor failures
- Use dead-band to eliminate “chatter”
- Power dependent set points increase operating margin
- “Condition out” sensor in unreliable operating region
- Digital trip output uses “-ve logic” (fail-safe in power loss)

Additional Considerations:

- Use multiple sensors to improve reliability
- There are many sensor trips, parameter trips, channel trips, warning lights, input buttons, etc. that all have to be given the same fail-safe treatment (i.e. 100s of functions)!

Electrical student’s reaction:

“But I never had to worry about that stuff in Matlab?”

- Welcome to the real world.

Computer science student's reaction:

"Still way simpler than my 1st java text-editor applet."

- Did it ever crash?

How can such systems be handled properly?

Review, Review, Review . . .

Multiple independent reviewers do:

- Software Requirements Specification (SRS) review
- Software Design Description (SDD) review
- Code review

Then . . . Test, Test, Test:

Independent testers do one & only one of:

- Unit Testing (UT) - test each individual program separately
- Software Integration Testing (SWIT) - test components when they are combined
- Validation Testing - test system against original system requirements

Logic provides a precise, unambiguous method of specifying system details for reviewers and testers

That’s still not enough!

- I’ve discovered incorrect designs that have been reviewed by as many as 5 different people - there is just too much detail for a person to catch everything!
- Testing can’t cover all possible cases - e.g. 1st shuttle flight initialization CPU overload had 1 in 67 probability of occurring
- Minor changes result in another extensive (& expensive) round of testing

Logic provides a means of mechanizing verification details - Computer Aided Verification!

Computer Aided Verification

What is CAV? . . . Prove, prove, prove!

Use tools to mathematically "prove" a design implements a well defined specification. E.g.

- Automated theorem proving of functional equivalence (i.e. use PVS or IMPS to prove for all inputs x: Spec(x) = Design(x))
- Model-checking automatically verifies that a Design is a model of a Spec written as a logical formula

Why use CAV Tools?

- Independent check of system unaffected by verifier's expectations
- Domain coverage - Tools can often be used to check ALL input cases
- Tools let you automate verification and reverification
- Provide additional capabilities (e.g. generation of counter example for debugging, type checking, verifying whole classes of systems, etc.)

You want to save Intel™ \$125 M?

- The Pentium™ floating point bug could have been detected by CAV.
- CAV was used after the bug was detected to prove the proposed fix corrected the problem.
- PVS has been used to verify similar circuits

You want to make a lot money?

- Learn Mathematical Logic and Computer Aided Verification.
- Chip makers, Utilities & Aerospace Companies will pay you a lot to use these skills.