Pragmatism

Musa Al-hassy

McMaster University
`alhassm@mcmaster.ca`

January 6, 2017

True or false?

If a monotonic function attains its maximum value at the start of an interval, then it is necessarily constant on the interval.

Reminder: course objective

Correct-by-construction programming

English requirements $\rightarrow$ Math specifications $\rightarrow$ Executable Code
All along, the proof and the code are developed *hand in hand!*

We illustrate the practicality of this approach using many examples in a simple language, which can then be readily "tried out" in Frama-C.

End goal – what you'll learn

Some mathematical techniques for reasoning, proving as a means to construct programs, and the usage of some proving tools.

Computer Programming is a rigorous, mathematical discipline

A student who wished to learn computer programming asked me for advice on two courses to take. One was more mathematical than the other, and he did not like Mathematics. Since he was a pianist, I asked him which course one should take to become a composer, if one does not like music. – Eric Hehner

It's always been "We can't do it that way. It would be too slow." You know what's slow? Spending all day trying to figure out why it doesn't work. That's slow. That's the slowest thing I know.
–Paul Phillips

What Kind of Examples?

We'll limit ourselves to small, and occasionally complex, problems. This' more than adequate since a large program is composed of small trusted/correct, reusable parts. (Recursion!)

Programming in the small

Small, trusted, parts tend to be more general than large parts which tend to be more specific.

small → accessible and convenient → many users → any bugs or flaws are fixed → trustworthy!

(http://www.johndcook.com/blog/2008/11/11/errors-in-math-papers-not-a-big-deal/)

The Vasa Disaster

1628 Swedish warship that sank on her maiden voyage after only a meager 1300 meters into the harbour; 53 lives were lost in the disaster.

Why?

- Ever changing requirements by the king
- Lack of specifications by the ship builder
- Lack of explicit design, scientific calculation of the ship stability, by subcontractors
- Test outcome was not followed by a certain admiral

The Ariane 5 Disaster

1996 rocket exploeds 37 seconds into maiden voyage! No injury in the disaster.

Why?

- Run time error! Overflow and out of range!
- The faulty program worked fine for Ariane 4 and as such not tested for Ariane 5.

Lack of attention to the strict preconditions below, especially the last term in each, was the direct cause of the destruction of the Ariane 5 and its payload – a loss of approximately DM 1200 million.

What is a “Blue Print”?

It seems the failures needed some guiadance, a blue print if you will.

Blue print

a representation of the system we want to build

- lacks a basis: cannot be “test drive” a blueprint of a car
- permits reasoning about the future system during its design
    - defining and calculating its behaviour –what it does— and incorporating constraints —what is must not do; recall our sorting algorithm that ignored repetitions.

Unsurprisingly, blueprints are important —or so say the professionals!

The Easy Part

Programming

The *Science* of designing efficient algorithms that meet their specifications.

Judged by criteria

- Correctness: does it solve the ‘right’ problem?
- Performance: how fast does it run and hos much space does it need?

We’ll look at the second point later on.

Now What?

Guess and Check: I've written some code, so know let me test it and make patches as needed.
(The other way around is known as: Test Driven Development.)

Scary

What if a civil engineer proposed building a bridge by trial-and-error?

We should run test cases not to look for bugs, but to increase our confidence in a program we are quite sure is correct; finding an error should be the exception rather than the rule. – David Gries

Verification

Fine, I'll prove my code correct now.

Verification

Proving the correctness of an algorithm after it has been designed. That is, we are checking the candidacy of a guessed solution.

This is terrible idea!

- Make proofs an "after thought", a necessary evil.
- Sometimes we need to do this!

The role verification in this class?

Our intentions, again

Being Sensible

As mentioned last time, the `Pre` and `Post` conditions define a program.
It is silly to implement something whose definition we don't even know! –recall last discussion!

We design a program that in a given state `Pre` establishes another state `Post` by solving for `X` in {`Pre`} `X` {`Post`} —this' program derivation!

Correct-by-construction Programming

A program together with its specification is a theorem expressing that the program satisfies the specification.

Hence, all programs require proofs, as theorems normally do!

Why Test!? I've proofs!

What could go wrong with a proof?

Unless you're using a proof checker: a lot can go wrong!

- You're implementation and your properties are inconsistent!
- The statement you wish to prove is too strong or too weak!
- You have a bug in your proof!

What could go wrong with a proof?

Proofs are for an idealised machine! Testing can reveal limitations of a particular machine!

Testing vs Proof

Testing

executing the program on selected inputs and comparing the result with the expected result

Proof

an argument that the program works as expected for all inputs.

Setting x to be the max

Proposed code

$\mathbf{do}\ x \leq y \rightarrow x, y := y, x\ \mathbf{od}$

Testing

If $x, y = 2, 3$ initially then we expect $x, y = 3, 2$ after execution.

Proof – only specification provided; details for later!

$\{x = X \land y = Y\}\ \mathbf{do}\ x \leq y \rightarrow x, y := y, x\ \mathbf{od}\ \{x = X \uparrow Y\}$

Note that in this case, we are guaranteed that x will always be the maximum.

Specfications for Testing

Testing

- increases confidence,
- *may* show existence of bugs, but it
- does not show non-existence of bugs.

Requirements can be turned into very specific test cases and specifications give a range of possibilities to be tested against.

References

Jean-Raymond Abrial's slides for formal methods
http://wiki.event-b.org/index.php/Event-B_Language
The first few slides we have are essentially taken from his 'Chapter 1' slides.

Muntazir Fadhel's notes on Testing
personal communication

Why The Vasa Sank: 10 Lessons Learned
http://faculty.up.edu/lulay/failure/vasacasestudy.pdf
Google "The Vasa: A Disaster Story with Software Analogies" and "Why the Vasa Sank: 10 Problems and Some Antidotes for Software Projects."

The Ariane 5 explosion as seen by a software engineer
http://www.cas.mcmaster.ca/~baber/TechnicalReports/Ariane5/Ariane5.htm
Short read; only a few paragraphs long.