Software System Design and Implementation

Wrapping up

The University of New South Wales
School of Computer Science and Engineering
Sydney, Australia

Lessons learnt

Think about properties!

- First think about what your programs is supposed to do!
  - How it does what it does comes later.
- Examples:
  - `prop_revApp xs ys =`

```
reverse (xs ++ ys) == reverse ys ++ reverse xs
```

  - Binary search tree property

```
appendV :: Vec n a -> Vec m a -> Vec (n + m) a
```

  - How do different operators relate to each other (turtle graphics)

Tells us that `reverse` is its own inverse, without telling us how

Properties as types

Properties help design

- Properties apply to large system components as well as to individual functions
- Properties discourage premature focus on implementation detail and optimisations
- Properties can naturally lead to formal specification if that is desired
- Properties may be formal or informal
- Properties are resilient to refactoring
- Convey meaning precisely

Properties help reasoning

- Properties are essential to formal reasoning
  - Can be denoted using formal logic and/or type systems
  - Deductive systems
- Link between compiler (i.e., type checker) and programmer
- Properties also facilitate informal reasoning
- Properties are precise

Properties help testing

- Property-based testing using randomised test-case generation
  - Reduces the amount testing code
  - Provides better documentation than unit tests
  - Increased coverage by repeated testing
- Ties in with (formal) specifications
- Can be combined with formal verification

Type systems are powerful tools

- Types are embedded in the program and change with the program
- Types are always checked by the compiler
- Types can be as expressive and precise as we like
- Every type checker is a theorem prover!

Types link programs and proofs!

Assurance is a continuum

- The level of desired assurance depends on the application
- Good tools work at different levels of the spectrum
    - Properties support informal reasoning, testing, and formal verification
    - Types can be rather simple (as in C) or very precise (as in Haskell GADTs, dependent types etc)
- Usually we need to mix different levels in one application

Applying the techniques

The lessons apply to all programming languages

- Think about properties
  - Applies fully to imperative and object-oriented programming
  - Properties can take global state and side effects into account
  - Requires shifting your attention from the details to the big picture
- Types
  - C++, C#, Scala, Rust and Java have sophisticated type systems, too — use them
  - Think about types/contracts in dynamically typed languages

Boost your productivity with functional programming

- But there are many other functional languages widely used — a small sample:
  - Erlang — developed by Ericsson, but used by many other companies, too
  - Clojure — Lisp dialect hosted on the JVM, strong concurrency support
  - Scala — mixed-paradigm, FP & OO on the JVM (Twitter, LinkedIn, etc.)
  - Swift — mixed-paradigm, FP & OO (Apple's heir to Objective-C)
  - F# — Microsoft's FP language in Visual Studio 2010

Coursework

- COMP3161: Concepts of Programming Languages (S2, Liam)
- COMP4161: Advanced Topics in Software Verification (S2, June Andronick, Gerwin Klein)

COMP 3161 —
Concepts of Programming Languages

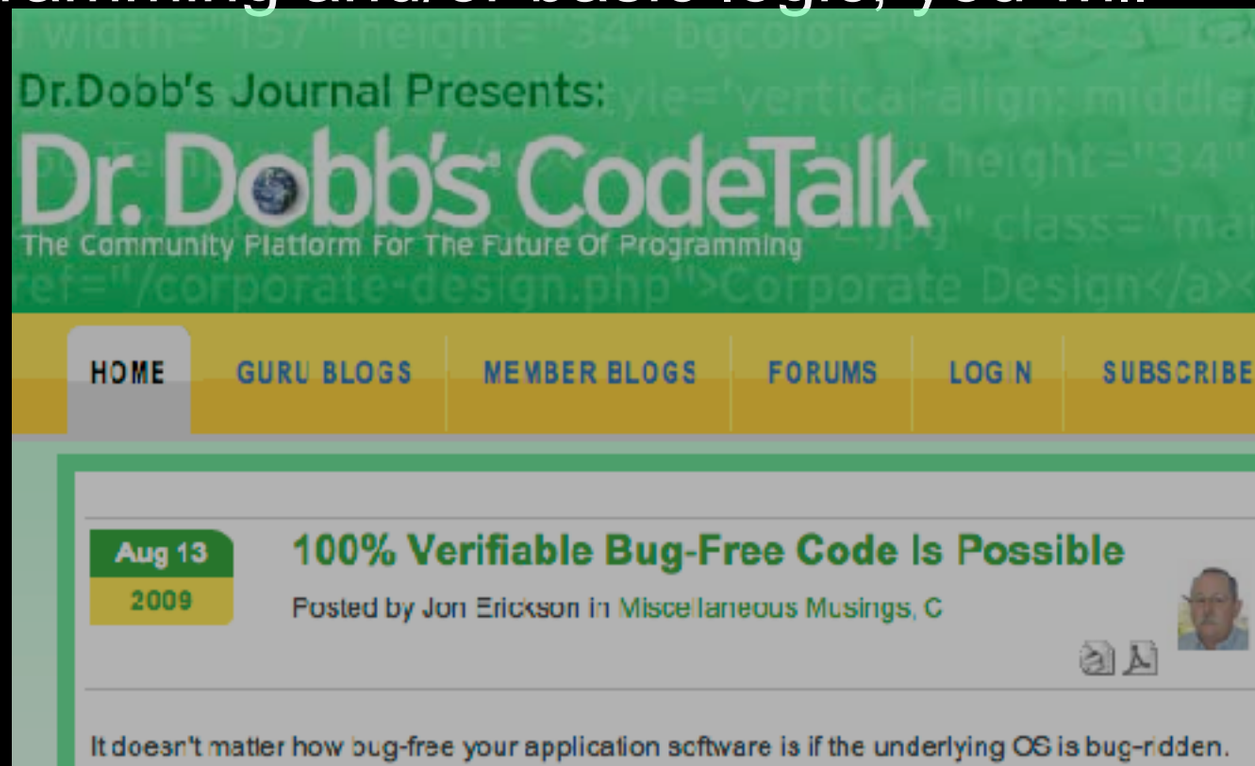
- What makes programming languages tick?
- Is there a system behind all those different language features?
- Static and dynamic semantics of common programming language features

COMP 4161

COMP4161 - Advanced Topics in Software Verification
Gerwin Klein, June Andronick

2018 - Session 2

- Advanced Topics in Software Verification
- learn how to use an interactive theorem prover
- automation, Higher-Order Logic, program verification
- done by the people who did the seL4 proof
- prereq: if you know functional programming and/or basic logic, you will be fine



COMP3151 — Foundations of Concurrency

- Design and implementation of
  - multi-threaded,
  - parallel, and
  - distributed programs.
- Talk, write, and reason about such programs — including formal reasoning.
- Appreciate the complexities involved in the above.
- Fundamentals for the next big round of kernel verification at NICTA!

Cogent

- With Data61, UNSW Systems and Formal Methods group
- Functional language for systems programming
- Code and proof co-generation

Structure

- 2 hour closed book exam (you can bring two A4 pages of handwritten notes, single sided)
- 4 questions
- Each question has several subquestions (either 4 or 5 subquestions)
    - The subquestions are not of equal value
- You will get two answer booklets (one for Q1 & Q2 and one for Q3 & Q4)

Format of questions and answers

- There are three types of subquestions:
  - textual questions (asking for an explanation),
  - coding questions (asking for Haskell code), and
  - a combination of the two previous types.

- Textual questions
  - They test your understanding of various concepts, or ask you to explain some code.
  - A few sentences are sufficient for each subquestion.
  - Overly verbose answers will lose marks.

- Coding questions
  - They require you to write Haskell code
  - A few lines of code suffice
  - Always include type signatures; add brief comments where helpful
  - Keep your answers clear — confusing or illegible code loses marks
  - Small syntactic mistakes will not lose marks; serious mistakes will

Tips

- The various subquestions are of strongly varying difficulty
- Some of the later questions are pretty easy
- Proceed as follows:

1. Carefully read through the entire exam
2. Mark easy questions that seem easy to you
3. Do the easy questions first!

How to prepare

- Have a look at the sample exam (ignore the assignment of marks)
- Make sure that you can solve all the exercises
- Make sure that you understand all the Haskell code that I posted together with the lecture slides

Supplementary exam

- I generally do not award a supplementary exam to students who sat the final
    - The supplementary exam is only for absentees
- Don’t sit the exam if you’re unwell
- To be considered for the supplementary exam, you must
    - have completed all other course components to a satisfactory standard,
    - have been absent from the final exam, and
    - have requested special consideration at NSQ within three working days.