COMP3141

Software System Design and Implementation

Introduction

Zoltan A. Kocsis
University of New South Wales
Term 2 2022

Meet the staff

I am Zoltan A. Kocsis (Research Asst., CSE): I'm a researcher at the Trustworthy Systems group at UNSW. I work on the applications of formal mathematical methods to the development of safe and secure software.

Dr John Shepherd (Senior Lecturer, CSE): the Course Convener for COMP3141.

Raphael Douglas Giles (CSE): will deliver practical lectures (Thursdays).

James Davidson (Casual Academic, Math): course admin, course forums.

Contacting Us

`http://www.cse.unsw.edu.au/~cs3141`

Forum

There is a course forum available on the course website. Questions about course content should typically be asked there. You can ask us private questions to avoid spoiling solutions to other students.

Administrative questions should be sent to the course email

`cs3141@cse.unsw.edu.au`

Student Support

For help with anything else, there is always

Student Support - I Need Help With...

Screenshot This Slide

Uni and Life in Australia Stress, Financial, Visas, Accommodation & More	**Student Support**	student.unsw.edu.au/**advisors**
Reporting Sexual Assault/Harassment	**Equity Diversity and Inclusion (EDI)**	edi.unsw.edu.au/**sexual-misconduct**
Educational Adjustments To Manage my Studies and Disability / Health Condition	**Equitable Learning Services (ELS)**	student.unsw.edu.au/**els**
Academic and Study Skills	**Academic Skills**	student.unsw.edu.au/**skills**
Special Consideration Because Life Impacts our Studies and Exams	**Special Consideration**	student.unsw.edu.au/**special-consideration**

My Feelings and Mental Health Managing Low Mood, Unusual Feelings & Depression				
	Mental Health Connect	student.unsw.edu.au/**counselling** Telehealth	**In Australia Call Afterhours** **UNSW Mental Health Support Line**	1300 787 026 5pm-9am
	Mind HUB	student.unsw.edu.au/**mind-hub** Online Self-Help Resources	**Outside Australia Afterhours** **24-hour Medibank Hotline**	+61 (2) 8905 0307

What is this course?

Our software should be
correct, safe and secure.

Our software should be
developed cheaply and quickly.

Safety-uncritical Applications

;

Video games: Some bugs are acceptable, to save developer effort.

Safety-critical Applications

Think of the worst group assignment you ever had! Imagine you...

- are logging into your online banking...
- are investing in a new hedge fund...
- are travelling in a self-driving car...
- are travelling on a plane...
- are getting treatment from a radiation therapy machine...
- are about to launch nuclear missiles...

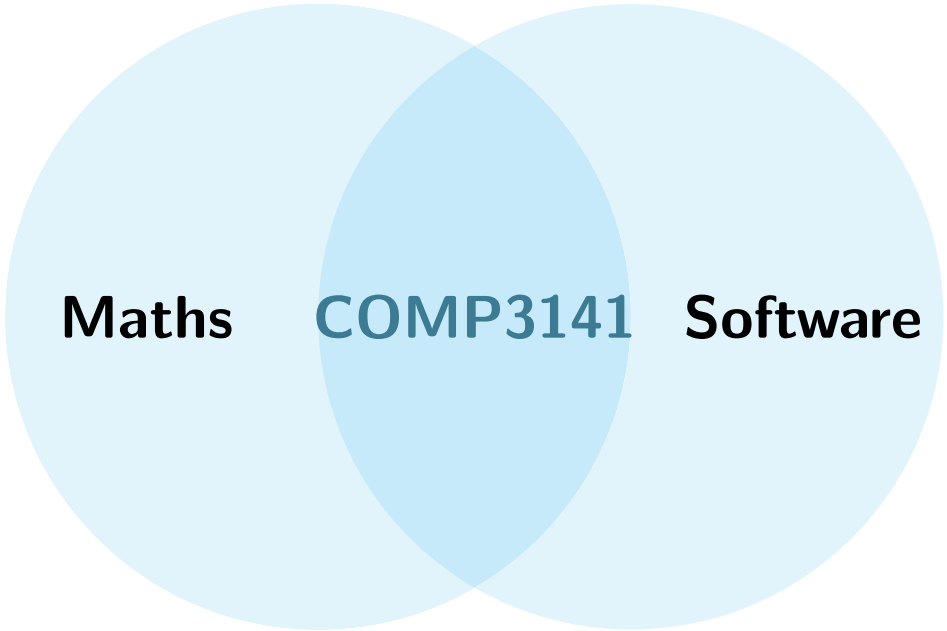
...using software written by your groupmates from that group.

Safety-critical Applications

Airline Blames Bad Software in San Francisco Crash

The New York Times

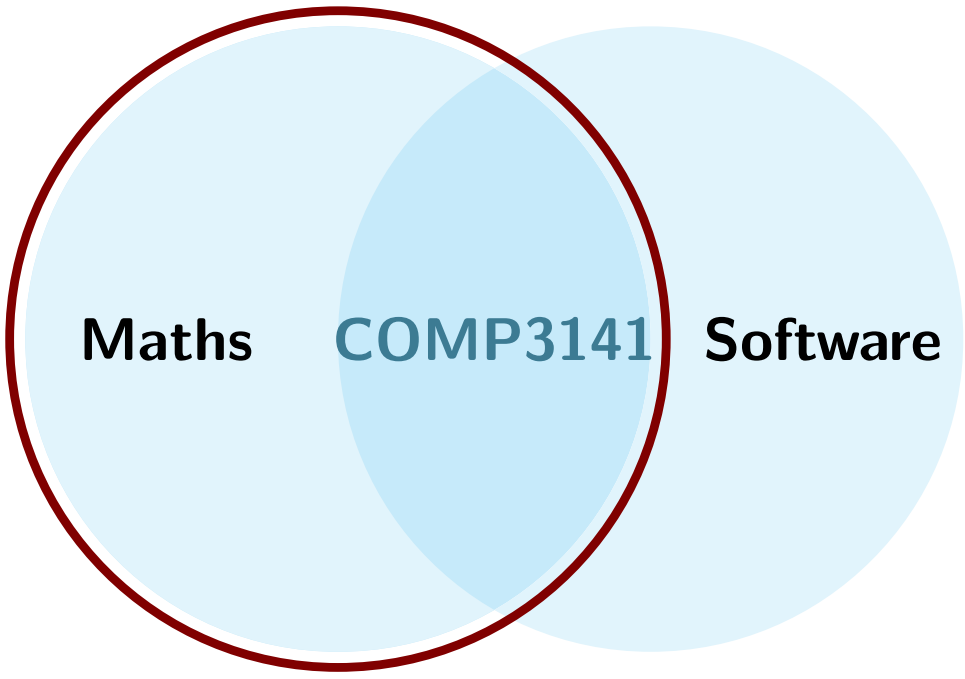
What is this course?



What is this course?

Maths?

- Logic
- Sets
- Proofs
- Induction
- Algebra (a bit)
- but no Calculus

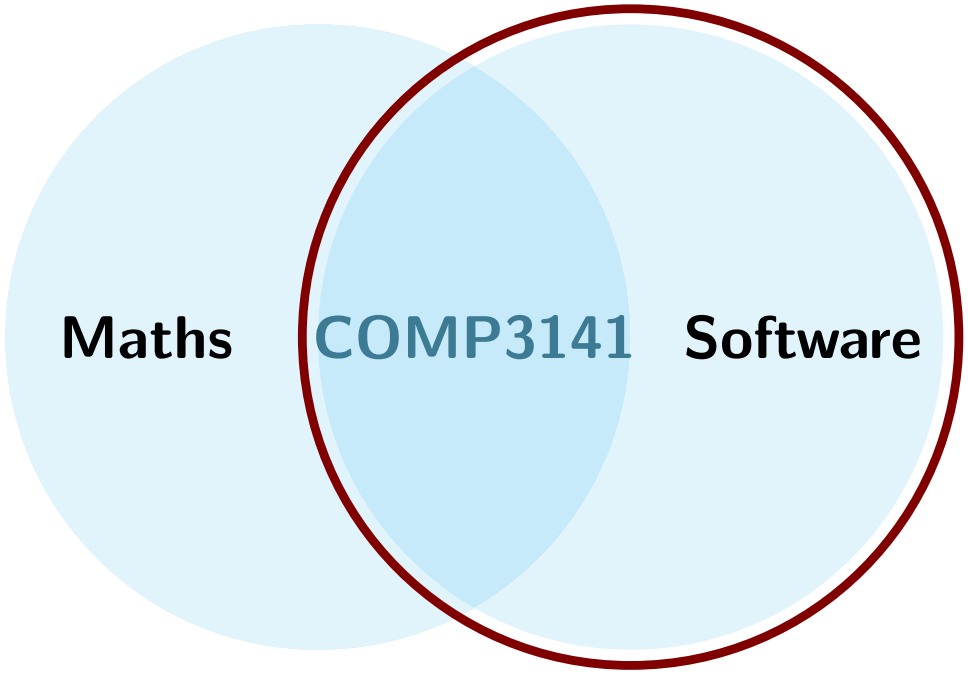


MATH1081 is neither necessary nor sufficient for COMP3141.

What is this course?

Software?

- Programming
- Reasoning
- Design
- Testing
- Types
- Haskell



N.B: Haskell knowledge is not a prerequisite for COMP3141.

What this course is not?

What this course is not?

- **not a Haskell course**
- **not** a formal verification course (for that, see COMP4161),
- **not** an OOP software design course,
- **not** a programming languages course (see COMP3161).
- Certainly **not** a cakewalk; but hopefully **not** a soul-crushing nightmare either.

Assessment

Warning

For many of you, this course will present a lot of new topics. Even if you are a seasoned programmer, you may have to learn as if you were starting from scratch.

- Class Marks (out of 100)
  - **Two** programming assignments, each worth 20 marks.
  - Weekly online quizzes, worth 20 marks.
  - Weekly programming exercises, worth 40 marks.
- Final Exam Marks (out of 100, hurdle: 40)

$$result = \frac{class + exam}{2}$$

Lectures

- Lectures and Practical Lectures: online, delivered using Zoom.
- **Lecture (Wed 1pm-3pm):** I introduce new material.
- **Practical (Thu 9am-11am):** Raphael or I reinforce Wednesday's material with questions and examples.
- **Quiz**: due on Thu (one week after the lectures they examine), but do them early!

Books

- **We won't set a textbook** (a long COMP3141 tradition).
- **Resources**: see the course outline for various books and online resources that are useful for learning Haskell.
- We can also provide more specialised text recommendations for specific topics.

Why Haskell?

- This course uses Haskell, because it is the most widely used language with good support for *mathematically structured programming*.
- You will learn a substantial amount of Haskell (we will provide some guidance). But the course is about learning techniques for mathematically structured programming.

About Haskell

- Haskell is old! It's turning 32 this year.
- Throughout the years: **Haskell 98**, **Haskell 2010**, **GHC2021**.

Warning

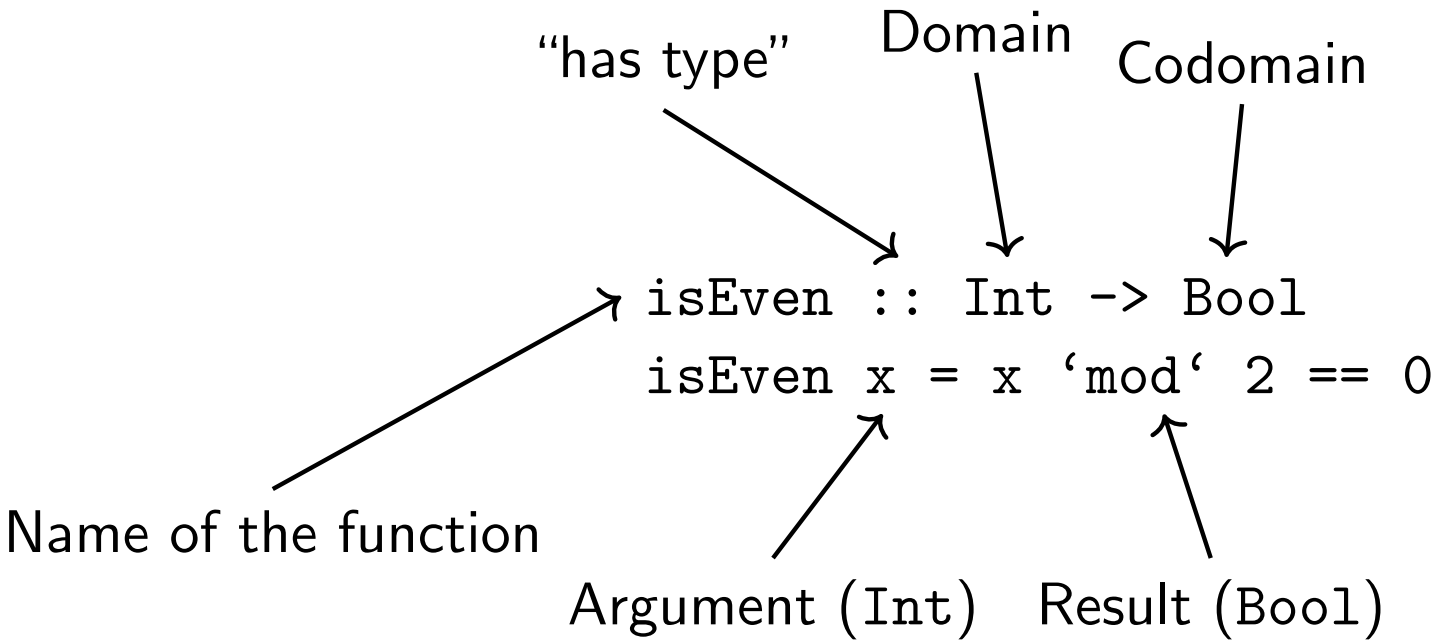
This means that some (possibly even most) tutorials, resources, answers you find on the Internet will be outdated!

Demo 1: Haskell Workflow

- Now we'll give you a Haskell Crash Course.
- This is to get you coding (solving problems) quickly.
- If you prefer "deep" understanding, don't worry: next week.

Demo: GHCi, Modules

Demo 2: Declaring Functions



In mathematics, we would apply a function f to an argument x by writing $f(x)$. In Haskell we write `f x`, omitting the parentheses.

Demo: basic functions

Demo 3: Currying

- Haskell functions have one input domain and one output codomain. But some functions take multiple inputs.
- In mathematics, we treat $\log_{10}(x)$ and $\log_2(x)$ and $\ln(x)$ as separate functions.
- In Haskell, we have a single function `logBase` that, given a number n, produces a function for $\log_n(x)$.

```
log10 :: Double -> Double
log10 = logBase 10

log2 :: Double -> Double
log2 = logBase 2

ln :: Double -> Double
ln = logBase 2.71828
```

What's the type of `logBase`?

Demo 3: Currying

```
logBase :: Double -> (Double -> Double)
```

(parentheses are optional above, we could write:)

```
logBase :: Double -> Double -> Double
```

Function application associates to the **left** in Haskell, so:

```
logBase 2 64  ≡  (logBase 2) 64
```

Demo: currying, multiple arguments

Demo 4: Tuples

We now know how to handle multiple inputs to a function? But what if we want to have multiple outputs?
Haskell provides data types called tuples to handle multiple outputs:

```
neighbors :: Int -> (Int, Int)
neighbors x = (x - 1, x + 1)
```

Now, `(neighbors 1)` evaluates to `(0,2)`.

Demo: tuples

Demo 5: Higher Order Functions

In addition to returning functions, functions can take other functions as arguments:

```
applyTwice :: (t -> t) -> t -> t
applyTwice f x = f (f x)

square :: Int -> Int
square x = x * x

fourthPower :: Int -> Int
fourthPower = applyTwice square
```

Demo: higher-order functions, equational reasoning

Demo 6: Lists

Haskell makes extensive use of lists, constructed using square brackets. Each list element must be of the same type.

```
[True, False, True]    ::   [Bool]
[3, 2, 5+1]            ::   [Int]
[sin, cos]             ::   [Double -> Double]
[ (3,'a'),(4,'b') ]    ::   [(Int, Char)]
```

Demo 6: Lists

A useful function is `map`, which, given a function, applies it to each element of a list:

```
map not [True, False, True] = [False, True, False]
map square [3, -2, 4]       = [9, 4, 16]
map (\x -> x + 1) [1, 5]    = [2, 6]
```

The last example here uses a *lambda expression* to define a one-use function without giving it a name.

What's the type of map?

```
map :: (a -> b) -> [a] -> [b]
```

Demo 6: Lists

The type `String` in Haskell is just a list of characters:

```
type String = [Char]
```

This is a *type synonym*, like a `typedef` in C.

Thus:

```
"hi!" == ['h', 'i', '!']
```

Demo: lists

Word Frequencies

Let's solve a problem to get some practice implementing stuff:

Example (Task 1)

Given a number n and a string s containing English words, generate a report that lists the n most common words in the given string s.

I'll even give you an algorithm:

1. Break the input string into words.
2. Convert the words to lowercase.
3. Sort the words.
4. Group adjacent occurrences (runs) of the same word.
5. Sort runs words by length.
6. Take the longest n runs of the sorted list.
7. Generate a report.

Demo: word frequencies

Homework

1. Get a Haskell development environment up and running. Instructions are on the course website.
2. The quiz will become available on the website after tomorrow's lecture. You'll use Hoogle, the Haskell documentation, and GHCi to answer the questions. (**assessed!**).