Interactive Formal Verification

Welcome

Dr. Dominic P. Mulligan

Programming, Logic, and Semantics Group,
University of Cambridge

Academic year 2017–2018

Administrivia

Course usually lectured by Prof. Lawrence Paulson

Sabattical leave this year

Administrivia

Course usually lectured by Prof. Lawrence Paulson

Sabattical leave this year

My office: FS16

- Until start of November
- Then at ARM, but will return to finish course

My e-mail: `dominic.p.mulligan@gmail.com`

Administrivia

Course usually lectured by Prof. Lawrence Paulson

Sabattical leave this year

My office: FS16

- Until start of November
- Then at ARM, but will return to finish course

My e-mail: `dominic.p.mulligan@gmail.com`

Course lab assistant: Dr. Victor Gomes

Victor's e-mail: `vb358@cam.ac.uk`

Administrivia

Course website:

`https://www.cl.cam.ac.uk/teaching/1718/L21/`

Administrivia

Course website:

`https://www.cl.cam.ac.uk/teaching/1718/L21/`

Course consists of 16 hours of contact time:

- 12 hours of lab-based lecturing,
- 4 hours of lab-based practicals

Administrivia

Course website:

`https://www.cl.cam.ac.uk/teaching/1718/L21/`

Course consists of 16 hours of contact time:

- 12 hours of lab-based lecturing,
- 4 hours of lab-based practicals

Assessed via two practical exercises:

- First (computer science) on parser combinators
- Second (maths) on metric spaces

IMPORTANT

All lecturing materials developed using Isabelle2016-1

Isabelle2017 about to be released imminently

Make sure you use Isabelle2016-1 for this course!

I recommend you install a local copy (ASAP) to follow along

Obtaining Isabelle

For your own machines: check course website

For lab machines see:

```
/auto/groups/acs-software/L21/Isabelle2016-1/
```

Contains `Isabelle2016-1_app.tar.gz` for installation in home directory

Also can start Isabelle2016-1 from your machine via:

```
/auto/groups/acs-software/L21/Isabelle2016-1/
Isabelle2016-1/Isabelle2016-1
```

Course text

Free! See:

`http://concrete-semantics.org/`

A stripped down version is distributed with Isabelle

Motivation

Developing software is hard

Most software (and hardware) has bugs

Bugs are costly, and potentially dangerous

Developing software is hard

Most software (and hardware) has bugs

Bugs are costly, and potentially dangerous

IDEA: treat program as a formal mathematical object

Prove relevant properties about model and obtain certified implementation thereafter

Developing software is hard

Most software (and hardware) has bugs

Bugs are costly, and potentially dangerous

IDEA: treat program as a formal mathematical object

Prove relevant properties about model and obtain certified implementation thereafter

Increases confidence in software/hardware implementation

Writing and checking proofs is hard

Proofs in mathematics and computer science may:

- Be tedious to check
- Contain subtle mistakes
- Be controversial (due to e.g. size, inability to review adequately)

Writing and checking proofs is hard

Proofs in mathematics and computer science may:

- Be tedious to check
- Contain subtle mistakes
- Be controversial (due to e.g. size, inability to review adequately)

IDEA: have a computer check that proof is valid

Writing and checking proofs is hard

Proofs in mathematics and computer science may:

- Be tedious to check
- Contain subtle mistakes
- Be controversial (due to e.g. size, inability to review adequately)

IDEA: have a computer check that proof is valid

Increases confidence in proof

Interactive theorem proving

Want to work in an expressive logic (which?)

Interactive theorem proving

Want to work in an expressive logic (which?)

The more expressive our logic the worse it behaves computationally

Proof search undecidable, intractable even in decidable fragments

Interactive theorem proving

Want to work in an expressive logic (which?)

The more expressive our logic the worse it behaves computationally

Proof search undecidable, intractable even in decidable fragments

IDEA: have the computer and a human work together

Human guides the proof search with computer:

- Checking that the human's reasoning is valid
- Helping when it can: (semi-)decision procedures, counterexample finders…

Isabelle, and Isabelle/HOL

Isabelle: a generic proof assistant

Isabelle initially written by Paulson starting mid 80s

Nipkow, Wenzel and others in Munich and elsewhere now a major development force

Written in Standard ML, follows LCF design philosophy

Isabelle: a generic proof assistant

Isabelle initially written by Paulson starting mid 80s

Nipkow, Wenzel and others in Munich and elsewhere now a major development force

Written in Standard ML, follows LCF design philosophy

Isabelle is a logical framework:

- Provides a relatively weak base (meta) logic
- More interesting (object) logics can be embedded in it
- Provides common reasoning tools, document preparation, and so on

Many instantiations

Many different object logic embeddings:

- ZF set theory
- First-order logic
- Martin-Löf type theory

Many instantiations

Many different object logic embeddings:

- ZF set theory
- First-order logic
- Martin-Löf type theory

In this course:

- (Mostly) ignore Isabelle's status as a logical framework
- Focus on one object logic: HOL
- Show off Isabelle/HOL as an interactive proof assistant for HOL

Gordon’s higher-order logic (HOL)

HOL = Church’s Simple Theory of Types + type polymorphism

Gordon’s higher-order logic (HOL)

HOL = Church’s Simple Theory of Types + type polymorphism

Suggested by Mike Gordon as a suitable logic for hardware verification

Implemented in HOL4, HOL Light, ProofPower HOL, HOL Zero

Gordon’s higher-order logic (HOL)

HOL = Church’s Simple Theory of Types + type polymorphism

Suggested by Mike Gordon as a suitable logic for hardware verification

Implemented in HOL4, HOL Light, ProofPower HOL, HOL Zero

…and of course Isabelle/HOL

HOL

HOL as a logic:

- Is polymorphically typed (as opposed to e.g. ACL2)
- Does not have type-dependency (as opposed to e.g. Coq or Agda)
- Is higher-order (as opposed to e.g. ACL2, or tools like Vampire)
- Strikes a good middle ground between expressivity and ability to interact with external tools (e.g. FOTPs, SMT solvers, etc.)

HOL

HOL as a logic:

- Is polymorphically typed (as opposed to e.g. ACL2)
- Does not have type-dependency (as opposed to e.g. Coq or Agda)
- Is higher-order (as opposed to e.g. ACL2, or tools like Vampire)
- Strikes a good middle ground between expressivity and ability to interact with external tools (e.g. FOTPs, SMT solvers, etc.)

As a functional programmer HOL will “feel” very familiar

No need to learn a radically different way of doing things

First taste of Isabelle/HOL

See associated theory...