Proving with Computer Assistance
Lecture 1.1

Herman Geuvers

Administration

- Teacher: Herman Geuvers (Thursday only)
- Mail to `herman@cs.ru.nl`
- Web page:
  `http://www.cs.ru.nl/H.Geuvers/onderwijs/provingwithCA/`
  or look at the link at my homepage.
- Weekly overview: see the webpage
- Lectures will be recorded; recordings will appear on Canvas.
- For practical work we will use the Proof Assistant Coq, which you can install yourself. See the webpage.
- We will be working on Coq .v files that will be provided via the webpage.

Examination

- Written exam + Coq Assignment
- Final grade = (Written Exam + Coq Assignment)/2
  with the condition that your Written Exam mark should be 5 or higher.
- If your Written Exam mark is below 5, this is your Final grade.
- You don't receive a mark (so I will write "NV") if you haven't completed all parts of the course.
- Written exam: Monday April 7, Time: 9:00–12:00.
  It is an *open book exam*, so you can bring any paper material you want
- Deadline Coq Assignment: Sunday April 13.
- In the resit period you can "redo" the written exam and/or the assignment. Marks from the first period will be retained.

Content

- Logic, Natural Deduction (known)
- Lambda calculus (known?)
- Type Theory
- Working with the Proof Assistant Coq

First (this) lecture:

- Some history of the field
- The general picture of proof assistants

First half of the previous century

- Untyped lambda calculus (Church, Curry, Turing)
  - What is (machine) computation? What is computability?
  - Untyped lambda calculus as a model for computation, equivalent to Turing Machines
  - Undecidable problems
  - Untyped lambda calculus is not a good basis to do logic
- Type Theory (Russell and Whitehead)
  - Ramified Theory of Types
  - A formal system for mathematics that avoids paradoxes
- Typed Lambda Calculus (Church)
  - Simple Theory of Types
  - Define higher order logic
  - Use lambda calculus to be clear about variable binding renaming bound variables, substitution, comprehension

Later Developments

Curry-Howard, De Bruijn, Girard, Martin-Löf.

- Interpreting formulas-as types and proofs-as-terms
- Dependent Types
- Polymorphic and Higher Order Types
- Inductive Types

Type theory as

- a language for describing proofs (deductions) as terms
- a basis for proof checking ($\rightarrow$ proof assistants)
- a formalism to define the provable total functions in arithmetic
- a foundation for (constructive) mathematics

And apart from that: typed lambda calculus as a basis for functional programming (Turner)

Type Theory now

- Theoretical basis of proof assistants
  - formulas are types
  - proofs (deductions) are terms
  - proof checking = type checking
  - proving = interactively constructing a term of a given type
- Theoretical basis for (functional) programming languages
  - types are specifications
  - terms are programs (with anotations)
  - compiler checks types $\rightarrow$ guarantees (partial) correctness
- Foundation of Mathematics: Constructive Mathematics; Homotopy Type Theory.

The general picture of Proof Assistants

What are Proof Assistants for?

- Precise mathematical modelling (defining)
- Verification of properties of systems (proving)

Computer supports in these activities:

- Checking correctness of definitions
- Take care of the bookkeeping
- Do some computation
- Do some proving for us

What have PAs ever done for us?

Does the Proof Assistant do all the proving for us?
No . . .
It is undecidable in general whether a formula is true or not.

Automated Theorem Provers	Proof Assistants
Specific domains	Generally applicable
Massage your problem	Modelling is direct
False or True (or Don't Know)	Interactive, user guided
No proofs	Complete, checkable proofs

Use of PAs

Who is using Proof Assistants and what for?
Computer Scientists for

- Modelling and specifying systems
- Proving the correctness of models / software /systems

Mathematicians for

- Building up theories
- Verifying proofs

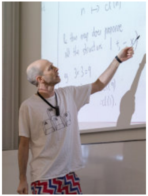
Mathematicians are not (yet) big users of Proof Assistants

- Mechanically verifying a proof takes too much time. (Too much idiosyncracy, not enough automation.)
- We don't need computers to verify proofs! We are much better at it!

Mathematical users of Proof Assistants

Gradually, more mathematicians are getting interested, young mathematicians are less afraid of computers.

- Store formalized mathematics on a computer and make large repositories of formal mathematics actively available.
- Various mathematicians observe that the proofs in their field are becoming too long, complex, abstract that one can only trust them if they are machine verified.
- Kevin Buzzard: Mathlib
  a user maintained library for the
  Lean theorem prover



Computer Science users of Proof Assistants

Compcert (Leroy et al.)

- verifying an optimizing compiler from C to x86/ARM/PowerPC code
- implemented using Coq's functional language
- verified using using Coq's proof language

Xavier Leroy

why?

- your high level program may be correct, maybe you've proved it correct ...
- ... but what if it is compiled to wrong code?
- compilers do a lot of optimizations: switch instructions, remove dead code, re-arrange loops, ...
- for critical software the possibility of miscompilation is an issue

C-compilers are generally not correct

Csmith project *Finding and Understanding Bugs in C Compilers*, X. Yang, Y. Chen, E. Eide, J. Regehr, University of Utah.

> *... we have found and reported more than 325 bugs in mainstream C compilers including GCC, LLVM, and commercial tools.*
> *Every compiler that we have tested, including several that are routinely used to compile safety-critical embedded systems, has been crashed and also shown to silently miscompile valid inputs.*
>
> *As of early 2011, the under-development version of CompCert is the only compiler we have tested for which Csmith cannot find wrong-code errors. This is not for lack of trying: we have devoted about six CPU-years to the task.*

Some history of Proof Assistants

- Church 1940: λ-calculus, simple type theory, higher order logic
- Curry Howard (De Bruijn): Formulas-as-Types
  Interpret formulas as types,
  Encode proofs as terms
  Proof-checking = Type-checking
- Automath (De Bruijn 1970s): first implementation of these ideas
- LCF (Milner), ML
- Coq, Hol, Isabelle, Lean, Agda, Mizar, PVS, ACL2, ...

These lectures

- Untyped lambda calculus next hour
  See the notes by Barendregt & Barendsen.
- Working with the proof assistant Coq
- Type Theory as a basis for proof assistant
  (NB Type Theory is also very much used in (Functional) Programming Languages. In the lectures I'll devote attention to this: type inference algorithm, ...)