

# Formal Type Theory

## Lecture 1

Lauri Alanko

University of Helsinki

November 3, 2009

# Practicalities

- Lectures from Nov 3rd to Dec 10th, Tue, Thu 12-14, B222
- Recitations on Thursdays after the lecture, B221
  - ▶ Hands-on Coq sessions, discussions and advice
  - ▶ Exact time still open
  - ▶ This Thursday: 14.15–17:30
- Course exam on Mon Dec 14th, 16–19, A111
- Textbook: Benjamin C. Pierce: *Types and Programming Languages* (MIT Press, 2002)
- Discussion forum?

# Homework

- *Really* important
  - ▶ Seriously!
- Returned electronically
  - ▶ Checked semi-automatically
  - ▶ Detailed instructions forthcoming
- Exercises published on Wednesdays
- Due by next Wednesday
  - ▶ Late returns penalized
  - ▶ Accepted until Friday

# Grading

- Homework: 20 p
- Exam: 40 p
- Homework points still applied to the repeat exam

# Prerequisites

- Elementary logic
  - ▶ Propositional connectives
  - ▶ Natural deduction
- Typed functional programming
  - ▶ E.g. Haskell, ML, Scala...
  - ▶ Polymorphism

# Learning objectives

After the course, you should be able to

- read papers on language design
- use a highly advanced dependently typed language
- formalize simple typed functional languages
  - ▶ maybe non-functional ones, too...
- prove their basic properties rigorously
  - ▶ and apply these skills elsewhere, too

# Motivation

Case studies:

- Java classloader bug [Saraswat 1997]
- Generic Java inference bug [Jeffrey 2001]
- Paper proofs get complex [Saha, Trifonov, Shao 2000]

# Problems with paper proofs

- Programming languages are designed to reflect our intuitions
- Thus they are directly intuitively correct
- Rigorous proofs of correctness are not interesting
  - ▶ They just affirm our intuition was correct all along
- Hence few people bother checking the proofs
  - ▶ They get huge for modern convoluted calculi
- But intuition and paper proofs can fail!

# Why mechanize?

- In programming languages, proofs of correctness are desirable but boring
- Proofs of incorrectness are undesirable but interesting!
- There's always a possibility of errors in proofs
  - ▶ ...which can have serious practical consequences
  - ▶ (Unlike some mathematical proofs)
- Mechanically verified proofs are much more reliable
  - ▶ But nothing is ever truly certain...

# Why types?

- Relatively simple but potentially powerful form of analysis
- “Well-typed programs don’t go wrong”
- Ubiquitous
  - ▶ Even lay programmers understand them
  - ▶ Well, *some* of them...
- Type annotations are part of the language
  - ▶ Best way to get the programmer to help ensure correctness

- Proof assistant developed at INRIA
  - ▶ Implemented in Objective Caml (also from INRIA)
- Don't laugh at the name!
  - ▶ Calculus of Constructions
  - ▶ (By Thierry Coquand et al.)
- Based on a functional language with a powerful type system
- ...and the Curry-Howard isomorphism:
  - ▶ Types are propositions
  - ▶ Programs are proofs
  - ▶ Programming techniques are proof techniques!

# A word of warning

- Rigorous formal proofs in Coq are more difficult than paper proofs
  - ▶ No handwaving!
- Interactive proving is *very* addictive
  - ▶ It's an adventure game!
- Be prepared for sleepless nights!
  - ▶ I know I will...