Theory of Programming and Types —Introduction—

Wouter Swierstra, Johan Jeuring

Utrecht University

3 February, 2014

Who?

State Your...

- Name
- Place of origin
- Experience in programming and types (e.g. courses taken, projects)
- Something we can remember about you

Outline

1. Who?
2. Introduction to Concepts
3. The course

Introduction to Concepts

Principles of programming languages

Back to 1973:

Principles of programming languages

Back to 1973:
First POPL: the ACM SIGACT-SIGPLAN symposium on Principles of programming languages

Principles of programming languages

Back to 1973:
First POPL: the ACM SIGACT-SIGPLAN symposium on Principles of programming languages

- Deterministic parsing of ambiguous grammars
- Programming language semantics and closed applicative languages
- Types are not sets
- Recursively defined data types
- Advice on structuring compilers and proving them correct
- Reasoning about programs

Programming and types

Structuring data (types) and manipulating data (programming) are fundamental components of Computer Science.

- How can types help a programmer?
- Can types express advanced properties of programs?
- Can we use types to construct a program?
- What is the meaning of a program, and its type?
- What properties of types, programs, programming languages and types systems would we like to have?
- How can we prove such properties?

These are the kind of question we want to address in the course on theory of programming and types.

Themes

The course is roughly divided into three themes:

- Dependently-typed programming in Agda
- Generic programming
- Semantics, type-systems, reasoning about types

Dependently-typed programming in Agda

- A dependent type is a type that depends on a value
- Example: lists of a particular length (aka vectors)
- Dependent types are used to describe very precise properties of programs
- Several implementations, we will use Agda

Generic programming

- A generic program is a program that depends on (the structure of) a type
- Example: equality of values
- Generic programs implement common programming patters and reduce code duplication
- Several implementations

Semantics, type systems, and their properties

- There are several ways to describe the semantics of a program: denotational, operational, big-step, small-step etc.
- The semantics depends on many aspects, such as call by value parameter passing, call by name, lazy evaluation, ...
- How do we set up a type system for a programming language?
- Can we show that a type doesn't change when a value is evaluated?

The course

Intended learning objectives

1. Program in a dependently typed programming language such as Agda
2. Prove properties of programs and systems using a dependently typed programming language such as Agda
3. Design and write datatype generic programs, for example by using a universe construction in Agda, but also by using fixed-points
4. Design a type system for a programming language
5. Formulate and prove properties of such a type system
6. Read, present, and use results from papers from the important conferences in the field, such as the Symposium on Principles of Programming Languages (POPL), and International Conference of Functional Programming (ICFP). This course will prepare you to to contribute to research in this area.

Mode of working

- Three blocks of two weeks (dependently typed programming, generic programming, semantics)
- Each block: approximately 3 lectures, one reading session (mainly classic papers), concluded with an exercise set
- 4 weeks of reading sessions and project work
- presentation of projects

Assessment of exercise sets

Self- and peer-assessment.

- Provide practice with assessment
- Develop skills for judging the quality of solutions

Process

1. Submit solutions
2. Lecturer distributes "gold standard" and student solutions
3. Assess own solutions and those of fellow student
4. Submit assessment of fellow student with grade and comments
5. Lecturer distributes grades and comments
6. The result contributes 40% to the final grade

Assessment of reading sessions

- Pick a paper
- Briefly introduce a paper at a reading session, and lead the ensuing discussion, going through the paper
- Assessment based on:
  - introduction of the paper
  - analysis of the results
  - raising questions
  - leading the discussion
- The result contributes 20% to the final grade

Papers

- John Launchbury; A natural semantics for Lazy Evaluation
- Jeremy Gibbons, Graham Hutton, Thorsten Altenkirch; When is a function a fold unfold
- Erik Meijer, Maarten Fokkinga, Ross Patterson; Functional Programming with Bananas, Lenses, Envelopes and Barbed Wire
- Rod Burstall; Proving properties of programs by structural induction
- Jean-Louis Krivine, Call-by-name lambda-calculus machine
- Nils Anders Danielsson; Operational semantics using the partiality monad
- Robby Findler and Matthias Felleisen; Contracts for Higher-order functions
- Luca Cardelli and Peter Wegner; On understanding types, data abstraction, and polymorphism

Research Project

Goals

- Develop problem solving and evaluation skills
- Test critical thinking skills
- Provide practice with teamwork
- Provide practice with writing research papers

Research project process

- Pick a project from the ideas provided, or an interesting research question (approved by lecturer) to answer
- Present research approach, results, relationship to course material, analysis, etc.
- Submit (via email) to lecturer a report describing your project
- Grading based on research approach, critical analysis, presentation style, and quality of report
- The result contributes 40% to the final grade

Projects

- Generic contracts
- Semantics of effects
- Verified compilers
- Contract inference
- Generic function inference
- Optimising generics using Coq
- ...

Contact

- Course URL: `http://www.cs.uu.nl/education/vak.php?stijl=2&vak=INFOMTPT`
- Email: W.S.Swierstra@uu.nl or J.T.Jeuring@uu.nl
- Office: BBL 571/572