Types & Semantics - Lecture 1

Wouter Swierstra

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Types and Semantics

This is a new course, drawing from the previous courses on *Automatic Program Analysis* and *Theory of Types and Programming*.

It will be taught by two lecturers:

- Jurriaan Hage
- Wouter Swierstra

If you have completed either of these courses previously, this may not be the right course for you.

Course schedule

Very roughly, the course can be divided into the following parts:

1. Intro to Agda (WS)
2. Semantics (WS)
3. Type and effect systems (JH)
4. Abstract interpretation (JH)
5. Papers and presentations (JH & WS)

Learning outcomes - Semantics

After completing this course, you will be able to:

1. Use a dependently typed programming language such as Agda to define the operational and denotational semantics of a simple programming language.
2. Formally define a type system for such programming languages.
3. Formulate and prove properties of programming language's type system and semantics, such as type soundness, progress, and preservation.

Learning outcomes - Types

1. Understand the notion of polymorphism and how algorithms such as algorithm W can infer the polymorphic types.
2. Understand the mathematical concepts underlying program analysis (such as lattices and fixed-points), and how these concepts may be used in the definition of type and effect systems for control-flow and data-flow analysis of higher-order languages.
3. Understand how such type and effect systems can be defined and implemented.

Learning outcomes - Generally

1. Understand the relationship between static analysis and semantics through the framework of Abstract Interpretation.
2. 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.

Background

Ideally, everyone here will already have encountered Agda in *Advanced Functional Programming*.

But in practice, it seems many of you have not taken that course (yet).

How much time should I spend introducing Agda?

How many people have already AFP/APA?

Completing the course

Your mark will consist of:

1. Hand-in exercises
2. Presenting a research paper
3. A small research project
4. Contributions during class

More details

More information on the schedule, rooms, exercises, etc. will show up in due course on the website:

http://wouter-swierstra.github.io/types-semantics/

Any organizational questions so far?

Who are you?

Introduction to Agda

We will, very roughly, follow the tutorial:

- James Chapman and Ulf Norell, Dependently typed programming in Agda

But the website has several useful links:

- Emacs cheatsheets
- List of Agda keyboard shortcuts
- Guide to writing unicode characters in Agda
- …

What is Agda?

Agda is a dependently typed programming language & proof assistant.

I'll try to give a basic introduction in a series of live coding sessions.

But be sure to follow along by studying the associated reading material yourself.

Installing Agda

The very brief version...

0. Make sure you have Emacs installed.
1. cabal update
2. cabal install Agda
3. agda-mode setup

If you have any trouble, check the instructions on Hackage. If all else fails, let me know and I can try to help you out.

Demo

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Agda

- A functional language, a bit similar to Haskell
- Implicit arguments
- Flexible syntax: mixfix operators and unicode
- But we can only define *total* functions