Types & Semantics - Lecture 4

Wouter Swierstra

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Exercises

Please submit your exercises through the submit website:

`www.cs.uu.nl/docs/submit`

You have until midnight tomorrow.

I'll follow up with instructions for peer assessment next week.

Recap

1. Introduction to Agda
2. Simply typed functional programming
3. Basic proofs
4. Using dependent types

When does a programming language exist?

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Semantics

A programming language is defined by its:

- syntax define which strings are program terms
- (static) semantics describes whether a given term makes sense (is type correct, well-scoped, etc.)
- (dynamic) semantics describes the result a program will produce when executed

Terms and values

The *terms* of a programming language describe the collection of all possible programs - think of basically describing the abstract syntax tree.

The subset of terms that cannot be evaluated any further are referred to as *values*.

A dynamic semantics somehow relates these terms and values.

Today's language

Today we'll start by looking at a trivial programming language:

```
data Term where
  True : Term
  False : Term
  Case : Term -> Term -> Term -> Term
```

What are the values of this language?

Semantics in Agda

A semantics is a relation over terms.

We can define these relations as dependent types in Agda. Just as we defined less than or equals as a relation on natural numbers.

We can then use Agda to test run our semantics or reason about our definitions.

Demo

Recap

- Denotational semantics define an interpreter
- Small step semantics define a relation between terms;
- Big step semantics define a relation between terms and values.

These definitions each have their pros and cons.

It is possible to prove different semantics equivalent.

Small step semantics

A *normal form* is a term that cannot be evaluated further.

Ideally, normal forms should correspond to values and be unique.

These semantics give us a very fine-grained view of how evaluation is performed.

Big step semantics

The big step semantics are defined as a relation between terms and values.

These definition `hide' intermediate evaluation states -- a single induction step may correspond to many small steps.

This sometimes makes reasoning about end results easier, but hides low-level details.

Denotational semantics

This corresponds to giving a total function defining an interpreter for our language.

This is easy for trivial languages -- but much harder for possibly non-terminating languages such as the untyped lambda calculus.

These definitions compute -- which makes them very pleasant to work with in a type theory such as Agda.

But may be harder to write than their big-step counterparts.