F1

Overview

Lennart Andersson

Revision 2011-03-14

2011

What is the value of x?

```
void p(int& x, int& y) {
  x = 1;
  y = 2;
}

int main() {
  int z;
  p(z, z);
}
```


What is the value?

```
class Variable {
  int value;
  void assign(int value) {
    this.value = value;
  }
}

void p(Variable x, Variable y) {
  x.assign(1);
  y.assign(2);
}

Variable z;
p(z,z);
```


Why take this course?

- Learn to read and write scientific articles with a theoretical flavor.

Who should not take this course?

- If you hate mathematics then you should keep away.

Why formal semantics

- Exact definitions
- Programming as a science rather than an art
- Proving program properties
- New perspectives on programs
- Discriminates good program constructs from bad
- Supports construction of correct programs

Different semantics

- Operational
- Denotational
- Axiomatic
- Predicate transformer
- Attribute grammars
- Algebraic semantics

Structural operational semantics

$$< z = x; x = y; y = z,\ [x \mapsto 5, y \mapsto 7, z \mapsto 0] > \Rightarrow$$

$$< x = y; y = z,\ [x \mapsto 5, y \mapsto 7, z \mapsto 5] > \Rightarrow$$

$$< y = z,\ [x \mapsto 7, y \mapsto 7, z \mapsto 5] > \Rightarrow$$

$$[x \mapsto 7, y \mapsto 5, z \mapsto 5]$$

Natural operational semantics

$$\sigma_0 = [x \mapsto 5, y \mapsto 7, z \mapsto 0]$$
$$\sigma_1 = [x \mapsto 5, y \mapsto 7, z \mapsto 5]$$
$$\sigma_2 = [x \mapsto 7, y \mapsto 7, z \mapsto 5]$$
$$\sigma_3 = [x \mapsto 7, y \mapsto 5, z \mapsto 5]$$

$$\dfrac{\dfrac{< z = x,\ \sigma_0 > \rightarrow\ \sigma_1 \quad < x = y,\ \sigma_1 > \rightarrow\ \sigma_2}{< z = x; x = y,\ \sigma_0 > \rightarrow\ \sigma_2} \qquad < y = z,\ \sigma_2 > \rightarrow\ \sigma_3}{< z = x; x = y; y = z,\ \sigma_0 > \rightarrow\ \sigma_3}$$

Denotational semantics

A *state* is a function (map) from names to values.

$$\sigma_0 = [x \mapsto 5, y \mapsto 7, z \mapsto 0]$$

$$State = Name \rightarrow \mathbb{Z}$$

Denotational semantics

The *denotational semantics* of a statement is a function from states to states.

$$\mathcal{S}_{ds}[\![x = e]\!] \in State \rightarrow State$$

$$\mathcal{S}_{ds}[\![z = x]\!](\sigma_0) = \sigma_1$$
$$\sigma_0 = [x \mapsto 5, y \mapsto 7, z \mapsto 0]$$
$$\sigma_1 = [x \mapsto 5, y \mapsto 7, z \mapsto 5]$$

Denotational semantics

$$\mathcal{S}_{ds}[\![z = x; x = y]\!](\sigma_0) =$$
$$\mathcal{S}_{ds}[\![x = y]\!](\mathcal{S}_{ds}[\![z = x]\!](\sigma_0)) =$$
$$(\mathcal{S}_{ds}[\![x = y]\!] \circ \mathcal{S}_{ds}[\![z = x]\!])(\sigma_0)$$

$$\mathcal{S}_{ds}[\![z = x; x = y]\!] = \mathcal{S}_{ds}[\![x = y]\!] \circ \mathcal{S}_{ds}[\![z = x]\!]$$

Weakest precondition semantics

Weakest precondition semantics is a variant of axiomatic semantics.

$$\mathcal{S}_{\mathrm{wp}}[\![z = x; x = y; y = z]\!](\mathit{postcondition}) = \mathit{weakest\ precondition}$$

$$\mathcal{S}_{\mathrm{wp}}[\![z = x; x = y; y = z]\!](x = m \wedge y = n) =$$

$$(x = n \wedge y = m)$$

$$\mathcal{S}_{\mathrm{wp}}[\![z = x; x = y; y = z]\!] \in \mathit{Predicate} \rightarrow \mathit{Predicate}$$

Lambda calculus

- Minimal functional language
- Simple syntax
- Simple semantics
- $(\lambda x.x\,x)(\lambda x.x) = (\lambda x.x)(\lambda x.x) = \lambda x.x$
- Untyped

Recursive definitions and domain theory

Theorem
This theorem is false.

Liar paradox

All Cretans lie.

(Stated by a Cretan.)

A recursive definition

Let X be a set of numbers.

$$X = \{0\} \cup (X + 1), \text{ where}$$
$$X + 1 = \{n + 1 \mid n \in X\}$$

- Is there a solution?
- Are there several solutions?
- Which is the intended solution?

Other recursive definitions

Let A and B be a sets.

$$A = \{A\}$$
$$B = B \cup \{B\}$$

- Are there solutions?
- Are there several solutions?
- Which are the intended solutions?

Domain theory

Domain theory provides means to investigate recursive definitions. It is required as a basis for denotational semantics.

A *domain* is partially ordered set with a possibility to define limits of sequences.

The partial order, often denoted by $\sqsubseteq$, may be interpreted as "is less defined than" or "has less information than".

Type inference

- Haskell has an advanced polymorphic type system
- `map (\x -> x+1) [1, 2, 3] = [2, 3, 4]`
- `map :: (a -> b) -> [a] -> [b]`
- Infer types using unification.

Prolog

- Prolog is a programming language based on predicate logic.
- A program consists of facts (axioms) and inference rules.
- The execution model uses unification.

Execution models

- Imperative languages (text book).
- SECD-machine for lambda calculus.
- Execute Prolog programs using unification.

Other topics

- The Peano axioms; $m + n = n + m$.
- Natural deduction; how do we think?
- Attribute grammars; another way to specify semantics of programming languages.
- Dependent types; types may be used to specify the meaning of a program to any desired degree of precision.

This week

- Overview (today)
- Concrete and abstract representation, Grammars, Regular expressions (tomorrow)
- Haskell (Wednesday)
- Seminar 1 (Friday)

Seminars

How?