F1

Overview

Lennart Andersson

Revision 2009-03-17

2009

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 IntVar {
  int value;
  void assign(int value) {
    this.value = value;
  }
}

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

IntVar z;
p(z,z);
```


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

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

Sats. *This theorem is false.* ■

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?

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

Execution models

- SECD-machine for lambda calculus
- Execute Prolog programs using unification

This week

- Overview (today)
- Introduction, Concrete and abstract representation, Semantic function (tomorrow)
- Haskell (Thursday)
- Seminar 1 (Friday)

Concrete grammar for arithmetic expressions - Canonical

$$
\begin{array}{lll}
\mathit{expr} & \rightarrow & \mathit{number} \\
\mathit{expr} & \rightarrow & \mathit{expr} + \mathit{expr} \\
\mathit{expr} & \rightarrow & \mathit{expr} * \mathit{expr} \\
\mathit{number} & \rightarrow & \mathit{digit} \\
\mathit{number} & \rightarrow & \mathit{digit}\ \mathit{number} \\
\mathit{digit} & \rightarrow & 0 \\
\ldots & \rightarrow & \\
\mathit{digit} & \rightarrow & 9
\end{array}
$$

Concrete grammar for arithmetic expressions - EBNF

$$
\begin{array}{lll}
\mathit{expr} & \rightarrow & \mathit{number} \mid \mathit{expr} + \mathit{expr} \mid \mathit{expr} * \mathit{expr} \\
\mathit{number} & \rightarrow & \mathit{digit}\ \mathit{digit}^{\star} \\
\mathit{digit} & \rightarrow & 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{array}
$$

Unambiguous grammar for arithmetic expressions

$$
\begin{array}{lll}
\mathit{expr} & \rightarrow & \mathit{term}\ (+\ \mathit{term})^{\star} \\
\mathit{term} & \rightarrow & \mathit{factor}\ (*\ \mathit{factor})^{\star} \\
\mathit{factor} & \rightarrow & \mathit{digit}\ \mathit{digit}^{\star} \\
\mathit{digit} & \rightarrow & 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{array}
$$

Abstract representation in Java

```
interface Expr {
  int value();
}

class Add implements Expr {
  Expr expr1, expr2;
  Add(Expr expr1, Expr expr2) {
    this.expr1 = expr1;
    this.expr2 = expr2;
  }
  int value() {
    return expr1.value() + expr2.value();
  }
}
```

Java representation – 2

```
class Mul implements Expr {
  Expr expr1, expr2;
  int value() {
    return expr1.value() * expr2.value();
  }
}

class Int implements Expr {
  int value;
  int value() {
    return value;
  }
}
```


Languages

- *language*: a set of strings
- *string*: a finite sequence of symbols from an alphabet
- *alphabet*: a finite set of symbols
- *symbol*: anything

Language operations

- $w \in L$
- $L_1 \cup L_2$
- $L_1 \cap L_2$
- $L_1 \cdot L_2$ or $L_1 L_2$
- L^n where $n \geq 0$
- L^*

Concrete grammar for regular expressions

$$
\begin{aligned}
\mathit{expr} &\rightarrow \emptyset \\
\mathit{expr} &\rightarrow \epsilon \\
\mathit{expr} &\rightarrow \sigma \quad , \text{where } \sigma \in \Sigma \\
\mathit{expr} &\rightarrow \mathit{expr} \cdot \mathit{expr} \\
\mathit{expr} &\rightarrow \mathit{expr} \mid \mathit{expr} \\
\mathit{expr} &\rightarrow \mathit{expr}^* \\
\mathit{expr} &\rightarrow (\mathit{expr})
\end{aligned}
$$

Σ is an alphabet. Notice that $\cdot$ and $|$ and * are symbols. The $\cdot$ is usually omitted.

Semantics of regular expressions

$$
\begin{aligned}
\mathcal{L}[\![\emptyset]\!] &= \{\ \} \\
\mathcal{L}[\![\epsilon]\!] &= \{\, \epsilon \,\} \\
\mathcal{L}[\![\sigma]\!] &= \{\, \sigma \,\}, \qquad \sigma \in \Sigma \\
\mathcal{L}[\![\alpha\, \beta]\!] &= \mathcal{L}[\![\alpha]\!]\mathcal{L}[\![\beta]\!] \\
\mathcal{L}[\![\alpha \mid \beta]\!] &= \mathcal{L}[\![\alpha]\!] \cup \mathcal{L}[\![\beta]\!] \\
\mathcal{L}[\![\alpha^*]\!] &= (\mathcal{L}[\![\alpha]\!])^*
\end{aligned}
$$