# Introduction to Semantics

February 13, 2001

CS 131: Programming Languages

# Semantics

- To understand a programming language, not enough to know its syntax.
- The *semantics* of a language specifies the meaning of a program
  - What program phrases mean when put together.
  - What answer does each program produce?
  - How should execution proceed?
- The large majority of the work in defining a language is specifying the semantics.

# Purposes of a Language Definition

- For the programmer
  - Understanding the language
  - Reasoning about programs
- For the language implementor
  - Understanding what correct implementations must/may do
  - Deciding whether program transformations are correct
  - Facilitate multiple (compatible) implementations
- For the language designer
  - Recording design decisions
  - Understanding interaction between language features
  - Reasoning about the language

# Formal Definitions?

- Why a formal semantics?
  - Informal definitions invariably contain ambiguities or errors.
  - Facilitates reasoning about the language
  - Facilitates reasoning about programs in the language
  - Facilitates reasoning about program transformations
  - May permit automatic generation of implementations
- Truth in advertising: very hard to give a formal description of a full, real language
  - But can handle quite large subsets
  - Active research topic

## Two Approaches to Formal Semantics

- Denotational semantics
  - The meaning of every program phrase is a mathematical object (a number, a function, a pair, a sequence, etc.)
  - *Compositionality*: meaning of an expression is a function of the meanings of its sub-expressions.
    - E.g., the meaning of the loop **`while`** *b* **`do`** *c* is calculated from the meanings of the guard expression *b* and of the loop body *c*
  - Two expressions with the same meaning are interchangeable.

## Two Approaches to Formal Semantics

- Operational semantics
  - Explains execution of complete programs
    - Formal specification of an interpreter
  - We can choose the level of abstraction
    - Which (if any) low-level machine details we want to describe
      - Data representations
      - Memory management
    - Which concepts considered primitive

## "Big-Step" Operational Semantics

- Also known as "natural semantics".
- Definition of an "evaluates to" relation.

$$\boxed{exp \Downarrow value}$$

- Example: `(3+4)+2` ⇓ `9`

## Recall: Abstract Syntax

```
exp ::= num                          integers
     |  bool                         booleans
     |  var                          variables
     |  (var) => exp                 functions
     |  exp exp                      applications
     |  exp + exp                    additions
     |  exp == exp                   equality-test
     |  exp ? exp : exp              conditionals
     |  <exp,exp>                    pairs
     |  fst exp                      1st projection
     |  snd exp                      2nd projection
     |  let var be exp in exp        local definitions
```

## Recall: Values

| | |
|---|---|
| *value* ::= *num* | *integers* |
| \| *bool* | *booleans* |
| \| (*var*) => *exp* | *functions* |
| \| <*value*,*value*> | *pairs* |

$$exp \Downarrow value$$

## Defining the Semantics

- An operational semantics is usually specified as a deductive system of inference rules.
- Recall the form of these rules:

$$\frac{\texttt{...premises...}}{\texttt{...conclusion...}}$$

## Arithmetic

- An axiom scheme:

$$\frac{}{value \Downarrow value}$$

- A rule of inference with two premise:

$$\frac{exp_1 \Downarrow m \qquad exp_2 \Downarrow n}{exp_1 + exp_2 \Downarrow m \oplus n}$$

## Exercise

- Give the complete proof tree showing that

$$(3+4)+2 \Downarrow 9$$

## Rules for Equality and Conditional

$$\frac{exp_1 \Downarrow value_1 \qquad exp_2 \Downarrow value_2}{exp_1 \;\texttt{==}\; exp_2 \Downarrow value_1 \equiv value_2}$$

$$\frac{exp_1 \Downarrow \textbf{true} \qquad exp_2 \Downarrow value}{exp_1 \;?\; exp_2 \;:\; exp_3 \Downarrow value}$$

$$\frac{exp_1 \Downarrow \textbf{false} \qquad exp_3 \Downarrow value}{exp_1 \;?\; exp_2 \;:\; exp_3 \Downarrow value}$$

## Exercise

- Show the proof tree for evaluation of

`if (3=4) then (5+6) else (7+8)` $\Downarrow$

## Exercise

- Complete the following three inference rules:

$$\frac{\qquad\qquad\qquad}{\langle exp_1, exp_2 \rangle \Downarrow}$$

$$\frac{\qquad\qquad\qquad}{\textbf{fst}\; exp \Downarrow}$$

$$\frac{\qquad\qquad\qquad}{\texttt{snd}\; exp \Downarrow}$$

## Local Definitions

- Local definitions can be described using substitution as seen in last class:

$$\frac{exp_1 \Downarrow value_1 \qquad exp_2[var \leftarrow value_1] \Downarrow value_2}{\textbf{let}\; var \;\textbf{be}\; exp_1 \;\textbf{in}\; exp_2 \Downarrow value_2}$$

- Example instance of this rule:

$$\frac{\texttt{1+1} \Downarrow \texttt{2} \qquad \texttt{2+2} \Downarrow \texttt{4}}{\texttt{let x be 1+1 in x+x} \Downarrow \texttt{4}}$$

## Function Application

$$\frac{exp_1 \Downarrow ((var)\ \texttt{=>}\ exp_3) \qquad exp_2 \Downarrow value_2 \qquad exp_3[var \leftarrow value_2] \Downarrow value_3}{exp_1\ exp_2 \Downarrow value_3}$$

3 premises!

$$\cfrac{\cfrac{}{((x)\texttt{=>}x+1) \Downarrow ((x)\texttt{=>}x+1)} \qquad \cfrac{\cfrac{}{2\Downarrow 2}\quad \cfrac{}{3\Downarrow 3}}{2+3 \Downarrow 5} \qquad \cfrac{\cfrac{}{5\Downarrow 5}\quad \cfrac{}{1\Downarrow 1}}{5+1 \Downarrow 6}}{((x)\texttt{=>}x+1)\ (2+3) \Downarrow 6}$$

## Reading Off an Interpreter

```
exception Error

fun eval expr =
   (case expr of
       Num n => Num n
     | Bool b => Bool b
     | FnVal(x,e) => FnVal(x,e)


     | Var _ => raise Error
```

$$\frac{}{v \Downarrow v}$$

## Addition

```
| Plus(e1,e2) =>
   let
     val v1 = eval e1
     val v2 = eval e2
   in
     (case (v1,v2) of
        (Num m, Num n) => Num (m+n)
      | _              => raise Error)
   end
```

$$\frac{exp_1 \Downarrow m \qquad exp_2 \Downarrow n}{exp_1 + exp_2 \Downarrow m \oplus n}$$

## Equality Test

$$\frac{exp_1 \Downarrow value_1 \qquad exp_2 \Downarrow value_2}{exp_1 == exp_2 \Downarrow value_1 \equiv value_2}$$

```
| Equal(e1,e2) =>
   let
     val v1 = eval e1
     val v2 = eval e2
   in
     (case (v1,v2) of
        (Num m1, Num m2) => Bool(m1=m2)
      | _                => raise Error)
   end
```

# Conditional

$$\frac{exp_1 \Downarrow \textbf{true} \qquad exp_2 \Downarrow value}{exp_1 \ ?\ exp_2 : exp_3 \Downarrow value}$$

$$\frac{exp_1 \Downarrow \textbf{false} \qquad exp_3 \Downarrow value}{exp_1 \ ?\ exp_2 : exp_3 \Downarrow value}$$

```
| Cond(e1,e2,e3) =>
   (case eval e1 of
      Bool true  => eval e2
    | Bool false => eval e3
    | _          => raise Error)
```

# Application

$$\frac{exp_1 \Downarrow ((var) \Rightarrow exp_3) \quad exp_2 \Downarrow value_2 \quad exp_3[var \leftarrow value_2] \Downarrow value_3}{exp_1\ exp_2 \Downarrow value_3}$$

```
| Apply(e1,e2) =>
    let
       val v1 = eval e1
       val v2 = eval e2
    in
      (case v1 of
         (FnVal(x,e3)) => eval(subst(e3,x,v2))
       | _             => raise Error)
    end
```

# Local Definitions

$$\frac{exp_1 \Downarrow value_1 \qquad exp_2[var \leftarrow value_1] \Downarrow value_2}{\textbf{let}\ var\ \textbf{be}\ exp_1\ \textbf{in}\ exp_2 \Downarrow value_2}$$

```
| Let(x,exp1,exp2) =>
   let
     val v1 = eval exp1
     val v2 = eval(subst(exp2,x,v1))
   in
     v2
   end
```

# Pairs and Projections

```
| Pair(e1,e2) => Pair(eval e1, eval e2)

| Fst M => (case (eval M) of
               Pair(v1,_) => v1
            | _ => raise Error)

| Snd M => (case (eval M) of
               Pair(_,v2) => v2
             | _ => raise Error))
```