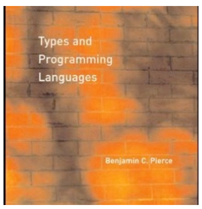
Natural and Contextual Semantics

Lecture 4
CS 565



Natural Semantics

The semantics given previously is known as “small-step”

- Evaluation relation shows how each individual step in the computation takes place
- Closely mirrors how an interpreter might evaluate a program
- Apply a multi-step evaluation relation $\rightarrow^*$ on top to talk about terms evaluating (in many steps) to values

An alternative style called “natural semantics” directly formulates the notion of “this term evaluates to this value”

- (Details omitted)

Evaluation Contexts

Both styles of semantics address two concerns:

- order of evaluation
  - explicit in small-step semantics
  - implicit in natural semantics
- meaning of terms

Can we separate out these two notions?

- Decompose a term into two parts:
  - the part of the term that is to be evaluated
  - the remaining portion of the term that should be examined after the subterm evaluates; call this part of the term a “context”

Contextual semantics

Small-step semantics where the atomic execution step is a rewrite of the program

- Evaluation terminates when program has been rewritten to a terminal program
- For IMP terminal command is “skip”

Need to define

- What constitutes an atomic reduction step
- How to select the next reduction step

Redex

A redex is a term that can be transformed in a single step

- A redex has no antecedents

```
r ::= x | x := int | int + int' | skip; c |
      if true then c1 else c2 |
      if false then c1 else c2 |
      true and b | false or b |
      ....
```

Evaluation Contexts

An evaluation context is a term with a "hole" in the place of a subterm

- Location of the hole points to the next subexpression that should be evaluated
- If `E` is a context then `E[r]` is the expression obtained by replacing redex `r` for the hole defined by context `E`
- Now, if `r`,σ → `t`,σ' then `E[r]`,σ → `E[t]`,σ'

Global reduction rule + Local reduction rules for individual `r`

Contexts

Can define evaluation context via a grammar:

```
E ::= [] | n + E | E + e | x := E |
       if E then c1 else c2 |
       E; c | ...
```

The grammar fixes the order of evaluation, allowing us to simplify the number and structure of the rules used in the semantics

Evaluation Contexts

A context has exactly one hole

Redexes that are substituted for a context are never values

A context uniquely identifies the next redex to be evaluated

Consider `e1+e2` and its decomposition as `E[r]`

- If `e1=n1` and `e2=n2` then `E=[]` and `r=n1+n2`
- If `e1=n1` and `e2≠n2` then `E=n1+E'` and `e2=E'[r]`
- If `e1≠n1` then `E=E'+e2` and `e1=E'[r]`

Last two cases are evaluated recursively

Evaluation Contexts

Consider `c = c1;c2`

- Suppose `c1 = skip`.
  Then, `c=E[skip;c2]` with `E=[]`
- Suppose `c1 ≠ skip`.
  Then, `c1=E[r]` and `c=E'[r]` with `E'=E;c2`

Consider `c = if b then c1 else c2`

- If `b=true` then `c=E[r]` where `r` is a redex in `c1` and `E` defines its context
- If `b=false` then `c=E[r]` where `r` is a redex in `c2` and `E` defines its context
- Otherwise, `b=E[r]`, so `c =E'[r]` where `E'= if E then c1 else c2`

Evaluation Contexts

Decomposition theorem:

- If `c≠skip` then there exists unique `E, r` such that `c=E[r]`

exists ⇒ progress

unique ⇒ determinism

Example

Consider the evaluation of:

`x:=1; x:=x+1 with σ = [x -> 0]`

State	Context	Redex
`x:=1;x:= x+1,σ`	`[];x:=x+1`	`x:=1`
`skip;x:=x+1,[x->1]`	`[]`	`skip; x:= x+1`
`x:= x+1,[x->1]`	`x := [] + 1`	`x`
`x:= 1+1,[x->1]`	`x: = []`	`1 + 1`
`x:= 2,[x->1]`	`[]`	`x := 2`
`skip, [x->2]`		

IMP

$E \quad ::= \qquad$ Contexts

$$
\begin{array}{rl}
E ::= & [\,] \\
\mid & E = a_2 \\
\mid & int = E \\
\mid & E < a_2 \\
\mid & int < E \\
\mid & E \ \mathbf{and}\ b_2 \\
\mid & bool\ \mathbf{and}\ E \\
\mid & E + a_2 \\
\mid & int + E \\
\mid & E * a_2 \\
\mid & int * E \\
\mid & E - a_2 \\
\mid & int - E \\
\mid & \mathbf{x} := E \\
\mid & E\ ;\ c_2 \\
\mid & \mathbf{if}\ E\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2
\end{array}
$$

IMP

$$\boxed{c\,,\,\sigma \Longrightarrow c'\,,\,\sigma'}$$

$$\frac{c = \mathbf{E}[\mathbf{r}] \qquad r\,,\,\sigma \longrightarrow r'\,,\,\sigma' \qquad c' = \mathbf{E}[\mathbf{r}']}{c\,,\,\sigma \Longrightarrow c'\,,\,\sigma'}\ \textsc{Ctxt}$$

$$\frac{}{\mathbf{skip}\,;\,c\,,\,\sigma \Longrightarrow c\,,\,\sigma}\ \textsc{Skip}$$

IMP

$$\boxed{r\,,\,\sigma \longrightarrow r'\,,\,\sigma'}$$

$$\frac{\sigma(\mathbf{x}) = \mathit{int}}{\mathbf{x}\,,\,\sigma \longrightarrow \mathit{int}\,,\,\sigma}\ \textsc{AexpVar}$$

$$\frac{\mathit{int}_1 + \mathit{int}_2 = \mathit{int}_3}{\mathit{int}_1 + \mathit{int}_2\,,\,\sigma \longrightarrow \mathit{int}_3\,,\,\sigma}\ \textsc{AexpPlus}$$

$$\frac{\mathit{int}_1 * \mathit{int}_2 = \mathit{int}_3}{\mathit{int}_1 * \mathit{int}_2\,,\,\sigma \longrightarrow \mathit{int}_3\,,\,\sigma}\ \textsc{AexpTimes}$$

$$\frac{\mathit{int}_1 - \mathit{int}_2 = \mathit{int}_3}{\mathit{int}_1 - \mathit{int}_2\,,\,\sigma \longrightarrow \mathit{int}_3\,,\,\sigma}\ \textsc{AexpSub}$$

$$\frac{}{\mathit{int}_1 = \mathit{int}_2\,,\,\sigma \longrightarrow \mathtt{true}\,,\,\sigma}\ \textsc{BexpEq}$$

$$\frac{\mathit{int}_1 \neq \mathit{int}_2}{\mathit{int}_1 = \mathit{int}_2\,,\,\sigma \longrightarrow \mathtt{false}\,,\,\sigma}\ \textsc{BexpNEq}$$

$$\frac{}{\mathbf{not}\,\mathtt{true}\,,\,\sigma \longrightarrow \mathtt{false}\,,\,\sigma}\ \textsc{BexpNotT}$$

$$\frac{}{\mathbf{not}\,\mathtt{false}\,,\,\sigma \longrightarrow \mathtt{true}\,,\,\sigma}\ \textsc{BexpNotF}$$

$$\frac{\mathit{bool}_1\ \mathbf{and}\ \mathit{bool}_2 = \mathit{bool}}{\mathit{bool}_1\ \mathbf{and}\ \mathit{bool}_2\,,\,\sigma \longrightarrow \mathit{bool}\,,\,\sigma}\ \textsc{BexpAnd}$$

$$\frac{\mathit{bool}_1\ \mathbf{or}\ \mathit{bool}_2 = \mathit{bool}}{\mathit{bool}_1\ \mathbf{or}\ \mathit{bool}_2\,,\,\sigma \longrightarrow \mathit{bool}\,,\,\sigma}\ \textsc{BexpOr}$$

$$\frac{\sigma' = \sigma[\mathbf{x} \mapsto \mathit{int}]}{\mathbf{x} := \mathit{int}\,,\,\sigma \longrightarrow \mathbf{skip}\,,\,\sigma'}\ \textsc{Assign}$$

$$\frac{}{\mathbf{if}\,\mathtt{true}\,\mathbf{then}\,c_1\,\mathbf{else}\,c_2\,,\,\sigma \longrightarrow c_1\,,\,\sigma}\ \textsc{IfT}$$

$$\frac{}{\mathbf{if}\,\mathtt{false}\,\mathbf{then}\,c_1\,\mathbf{else}\,c_2\,,\,\sigma \longrightarrow c_3\,,\,\sigma}\ \textsc{IfF}$$

$$\frac{}{\mathbf{while}\,b\,\mathbf{do}\,c_1\,,\,\sigma \longrightarrow \mathbf{if}\,b\,\mathbf{then}\,c_1\,;\,\mathbf{while}\,b\,\mathbf{do}\,c_1\,\mathbf{else}\,\mathbf{skip}\,,\,\sigma}$$

Contextual Semantics

Summary

- Think of a hole as representing a program counter
  - The rules for advancing holes is non-trivial
    - Must decompose entire command at every step
    - How would you implement this?
- Major advantage of contextual semantics is that allows a mix of global and local reduction rules
- Global rules indicate next redex to be evaluated
  - defined by contexts
- Local rules indicate how to perform the reduction
  - one for each redex