Lecture : Lecture #3
Date   : 09/01/99
Scribe   : Woong Yoon
Subject : IMP / Operational Semantics

Course Overview: (See also Web Page)

- Operational Semantic (One and a half weeks)
- Lambda Calculus
- Denotational Semantics
- Naming / State / Control
- Axiomatic Semantics
- Prelim (Mid Term)
- Types
  - Type Reconstruction
  - Polymorphic Type
  - Subtyping
  - Recursive Type
  - Object Type
- Current Topics
- Final Exam

IMP (continues from the last lecture)

a : arithmetic expression
b : boolean expression
c : command expression

Evaluations of the above expressions in state σ
$\langle a,\sigma\rangle \to n$
$\langle b,\sigma\rangle \to$ true, false
$\langle c,\sigma\rangle \to \sigma'$ (new state)

Initial state of program
$\sigma 0(X)=0 \quad \forall X$

Axiom (or rule) for command evaluation

skip: nothing happens

$$\langle skip,\sigma\rangle \to \sigma$$

sequence: c0 is evaluated in state σ, and then c1 is evaluated in the result state σ'. Final result is a new state σ".

$$\frac{\langle c0,\sigma\rangle \rightarrow \sigma' \quad \langle c1,\sigma'\rangle \rightarrow \sigma''}{\langle c0;c1,\sigma\rangle \rightarrow \sigma''}$$

assignment: Location X is assigned with a, and if a has value n in state σ, then the result is a state in which X is substituted with n.

$$\frac{\langle a,\sigma\rangle \rightarrow n}{\langle X:=a, \sigma\rangle \rightarrow \sigma[n/X]}$$

if statement:

case 1: when b is true in state σ, c0 is evaluated in state σ; result is a new state σ'

$$\frac{\langle b, \sigma\rangle \rightarrow \text{true} \quad \langle c0, \sigma\rangle \rightarrow \sigma'}{\langle \text{if } b \text{ then } c0 \text{ else } c1, \sigma\rangle \rightarrow \sigma'}$$

case 2: when b is false in state σ, c1 is evaluated instead of c0; the result is a new state σ'

$$\frac{\langle b, \sigma\rangle \rightarrow \text{false} \quad \langle c1, \sigma\rangle \rightarrow \sigma'}{\langle \text{if } b \text{ then } c0 \text{ else } c1, \sigma\rangle \rightarrow \sigma'}$$

while statement:

case 1: when b is false in state σ, state does not change

$$\frac{\langle b, \sigma\rangle \rightarrow \text{false}}{\langle \text{while } b \text{ do } c, \sigma\rangle \rightarrow \sigma}$$

case 2: when b is true in state σ, c is evaluated in state σ, and the while statement is evaluated again in the result state σ"; the result is a new state σ'

$$\frac{\langle b, \sigma\rangle \rightarrow \text{true} \quad \langle c, \sigma\rangle \rightarrow \sigma'' \quad \langle \text{while } b \text{ do } c, \sigma''\rangle \rightarrow \sigma'}{\langle \text{while } b \text{ do } c, \sigma\rangle \rightarrow \sigma'}$$

Now let's prove that the following program never terminates

```
while 0=0 do skip;
```

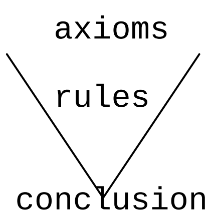
Since $\langle 0=0, \sigma\rangle$ is true, we will use the rule case2 of while statement.

$$\frac{\begin{array}{l}\langle 0=0, \sigma\rangle \rightarrow \text{true} \\ \langle \text{skip}, \sigma\rangle \rightarrow \sigma \quad \langle \text{while } 0=0 \text{ do skip}, \sigma\rangle \rightarrow \sigma'\end{array}}{\langle \text{while } 0=0 \text{ do skip}, \sigma\rangle \rightarrow \sigma'}$$

applying to the rule does not progress at all, and therefore it implies infinite loop.

The above semantic is called "structural operational semantics", and it is also known as "natural semantics" because it resembles proofs in natural deduction.

Proof tree will always starts out from axioms and above rules are applied to get a conclusion.



How do we show if two programs (expressions) are equivalent.

Two expressions are equivalent if they have same result in every state.

$a0 \sim a1$: arithmetic expressions are equivalent if
$\forall \sigma\ [\langle a0, \sigma\rangle \rightarrow n \Leftrightarrow \langle a1, \sigma\rangle \rightarrow n]$

$c0 \sim c1$: commands are equivalent if
$\forall \sigma\ [\langle c0, \sigma\rangle \rightarrow \sigma' \Leftrightarrow \langle c1, \sigma\rangle \rightarrow \sigma']$

We can express like this :

$\langle a, \sigma\rangle \rightarrow n1\ \&\ \langle a, \sigma\rangle \rightarrow n2 \Rightarrow n1 = n2$

$\langle c, \sigma\rangle \rightarrow \sigma'\ \&\ \langle c, \sigma\rangle \rightarrow \sigma'' \Rightarrow \sigma' = \sigma''$

Now, consider following two expressions:

x:=1;

while $\neg$X=1 do X:=1;

in order to prove those are equivalent, we have to show that

$$\forall \sigma \ \langle X{:=}1, \sigma\rangle \to \sigma' \iff \langle \text{while } \neg X{=}1 \text{ do } X{:=}1,\ \sigma) \to \sigma'$$

Proof tree of $\langle x{:=}1, \sigma\rangle \to \sigma'$

$$\frac{\begin{array}{l}\langle 1, \sigma\rangle \to 1\\ \sigma' = \sigma[1/X]\end{array}}{\langle X{:=}1, \sigma\rangle \to \sigma'}$$

Proof tree of $\langle \text{while } \neg x{=}1 \text{ do } X{:=}1,\ \sigma\rangle \to \sigma'$ has two cases.

When $\langle \neg X{=}1, \sigma\rangle \to \text{true}$:

$$\dfrac{\dfrac{\dfrac{\begin{array}{l}\sigma(X)\,!{=}\,1\\ \langle X, \sigma\rangle \to \sigma(X)\\ \langle 1, \sigma\rangle \to 1\end{array}}{\langle \neg X{=}1, \sigma\rangle \to \text{false}}}{\langle \neg X{=}1, \sigma\rangle \to \text{true}} \qquad \dfrac{\sigma'' = \sigma[1/X]}{\langle X{:=}1, \sigma\rangle \to \sigma''} \qquad \dfrac{\begin{array}{c}\sigma' = \sigma'' = \sigma[1/X]\\ \langle \neg X{=}1\ , \sigma[1/X]\rangle \to \text{false}\end{array}}{\langle \text{while } \ldots\ , \sigma''\rangle \to \sigma'}}{\langle \text{while } \neg x{=}1 \text{ do } X{:=}1,\ \sigma\rangle \to \sigma'}$$

Note that in the third premises $\langle \text{while } \ldots\ , \sigma''\rangle \to \sigma'$,
when $\langle \neg X{=}1, \sigma[1/X]\rangle \to \text{true}$, it reads to a contradiction.

$$\dfrac{\dfrac{1\ !{=}\ 1}{\langle X{=}1, \sigma[1/X]\rangle \to \text{false}}}{\langle \neg X{=}1, \sigma[1/X]\rangle \to \text{false}} \qquad \text{(contradiction)}$$

From above, therefore

$$(\sigma(X)\ !{=}\ 1\ ) \text{ and } (\sigma' = \sigma[1/X]) \qquad (1)$$

When $\langle \neg X=1, \sigma\rangle \rightarrow false$:

$$\dfrac{\dfrac{\dfrac{\sigma(X) = 1}{\langle X=1, \sigma\rangle \rightarrow true}}{\langle \neg X=1, \sigma\rangle \rightarrow false} \qquad\qquad \sigma' = \sigma}{\langle \text{while } \neg x=1 \text{ do } X:=1,\ \sigma\rangle \rightarrow \sigma'}$$

therefore

$$(\sigma(X) = 1) \text{ and } (\sigma' = \sigma) \qquad\qquad (2)$$

From (1) and (2):

$((\sigma(X) \mathrel{!}= 1) \text{ and } (\sigma' = \sigma[1/X])) \text{ or } ((\sigma(X) = 1) \text{ and } (\sigma' = \sigma))$

$\sigma(X) = 1 \Rightarrow \sigma = \sigma[1/X]$

$(\sigma(X) = 1) \text{ and } (\sigma' = \sigma[1/X])$

$\sigma' = \sigma[1/X]$

CONCLUSION:

Because proof tree of both expressions “X:=1;” and “while ¬X=1 do X:=1;” leads to premises $\sigma' = \sigma[1/X]$, those two expressions are equivalent //

We can build IMP in ML: (please recheck the syntax of ML below)

Type :

Arithmetic expression

```
A = datatype
      num of int
    | plus of A A
    | times of A A
    | sub of A A
    | LOC of string
```

Boolean expression

```
B = datatype
      not of B
    | and of B B
    | or of b b
    | eq of A A
```

Command expression

```
C = datatype
      Skip
    | Seq of C C
    | If of B C C
    | While of B C
    | Assgn of string A
```

Evaluation function:

```
fun eval_C(c:C, s=state) : state =
   case c of
      Skip => s
    | Seq(c0,c1) => eval_C(c1,eval_c(c0,s))
    | If(b,c0,c1) => if eval_B(b,s)
                     then eval_C(c0,s)
                     else eval_C(C1,s)
    | While(b,c) => if eval_B(b,s)
                     then eval_C(while(b,c),eval_C(c))
                     else s
    | Assgn(n,a) =>
```

.. To be continued in the next lecture