CS611 Lecture 4: Interpretation, More Language Features 09/03/99
Hubie Chen

1 ML interpreter for IMP

Here is a ML interpreter for the IMP language.

```
type S = string -> int

datatype A =
      Num of int
    | Plus of A*A
    | Minus of A*A
    | Times of A*A

datatype B =
    | Leq of A*A
    | Equals of A*A
    | And of B*B
    | Or of B*B
    | Not of B

datatype C =
      Skip
    | Assign of string*A
    | Semi of C*C
    | If of B*C*C
    | While of B*C

fun eval_A( a : A, s : S ) : int =
    case a of
          Num => a
        | Var => Lookup( s, a )
        | Plus( a1, a2 ) => eval_A( a1 ) + eval_A( a2 )

fun eval_C( c : C, s : S ) : S =
    case c of
          Skip => s
        | Assign( v, a ) => Update( s, v, eval_A( a ))
        | Semi =>  . . .
        | If =>  . . .
        | While =>  . . .

fun lookup( s : S, v : string ) = s( v )

fun update( s : S, v : string, n : int ) =
    fn v2 : string =  if v1 = v2 then n else s( v2 ) end
```

2 More Language Features

2.1 do-until loops

Now, let's try adding a **do-until** loop to IMP. We could do this by introducing the following two rules:

$$\frac{\langle c, \sigma\rangle \to \sigma' \qquad \langle b, \sigma'\rangle \to \text{true}}{\langle \text{ do } c \text{ until } b, \sigma\rangle \to \sigma'}$$

$$\frac{\langle c, \sigma\rangle \to \sigma'' \qquad \langle b, \sigma''\rangle \to \text{false} \qquad \langle \text{ do } c \text{ until } b, \sigma''\rangle \to \sigma'}{\langle \text{ do } c \text{ until } b, \sigma\rangle \to \sigma'}$$

Another way to add **do-until** would be to introduce the following single rule:

$$\frac{\langle c \text{ ; while } \neg b \text{ do } c, \sigma\rangle \to \sigma'}{\langle \text{ do } c \text{ until } b, \sigma\rangle \to \sigma'}$$

2.2 Division and an Exception Mechanism

What about division, that other arithmetic operation from second grade which we dread but so? Alas, let us try to augment IMP so as to support it as well. Try this on for size:

$$\frac{\langle a_1, \sigma\rangle \to n_1 \qquad \langle a_2, \sigma\rangle \to n_2 \qquad n = \lfloor n_1/n_2 \rfloor}{\langle a_1/a_2, \sigma\rangle \to n}$$

Oops! This is clearly unsatisfactory - what if $n_2 = 0$?

This seeming problem can be resolved with an exception mechanism. For instance, we might include the following rule:

$$\frac{\langle a_1, \sigma\rangle \to n_1 \qquad \langle a_2, \sigma\rangle \to 0}{\langle a_1/a_2, \sigma\rangle \to \text{fail}}$$

Unfortunately, we would have to propagate "fail" through every rule. As an example:

$$\frac{\langle a_1, \sigma\rangle \to \text{fail} \ \langle a_2, \sigma\rangle \to n_2}{\langle a_1 + a_2, \sigma\rangle \to \text{fail}}$$

This is evidence of a problem with this particular style of semantics (called "large-step semantics").

2.3 Arrays

We now direct our attention toward the problem of adding arrays to IMP. Currently, we have

$\sigma : \text{Loc} \rightarrow \mathcal{Z}$

Potentially, we could change the range of our state functions:

$\sigma : \text{Loc} \rightarrow (\mathcal{Z} + (\mathcal{Z} \rightarrow \mathcal{Z}))$

But with this definition, we risk dynamic type errors.

So instead, we use two state functions:

$\sigma_Z : \text{Loc} \rightarrow \mathcal{Z}$
$\sigma_A : \text{Loc} \rightarrow (\mathcal{Z} \rightarrow \mathcal{Z})$

. . . and change IMP as follows:

$a := \ldots | A[a_1]$
$c := \ldots | A[a_1] := a_2$

$\langle a, \sigma_Z, \sigma_A \rangle \rightarrow n$
$\langle c, \sigma_Z, \sigma_A \rangle \rightarrow (\sigma'_Z, \sigma'_A)$

$$\frac{\langle a_1, \sigma_Z, \sigma_A \rangle \rightarrow n_1 \qquad \sigma_A(A)(n_1) = n}{\langle A[a_1], \sigma_Z, \sigma_A \rangle \rightarrow n}$$

$$\frac{\langle a_1, \sigma_Z, \sigma_A \rangle \rightarrow n_1 \qquad \langle a_2, \sigma_Z, \sigma_A \rangle \rightarrow n_2 \qquad \sigma'_A = \sigma_A[\sigma_A(A)[n_2/n_1]/A]}{\langle A[a_1] := a_2, \sigma_Z, \sigma_A \rangle \rightarrow n}$$