Procedure Calls

March 25, 2002

CS 131: Programming Languages

Example 1

- Consider the following SML definition:

```
fun optimizedMult (0,m) = 0
  | optimizedMult (1,m) = m
  | optimizedMult (2,m) = m+m
  | optimizedMult (n,m) = n*m
```

- How many additions happen in:

```
optimizedMult(0,1+2) ?

optimizedMult(2,1+2) ?
```

Example 2

- What is the result of the following code?

```
fun f() = f()

fun zero _ = 0

val x = zero(f())
```

Evaluating Calls

- Recall from our simple functional language:

$$\frac{exp_1 \Downarrow ((var) \Rightarrow \text{exp}_3) \quad exp_2 \Downarrow value_2 \quad exp_3[var \leftarrow value_2] \Downarrow value_3}{\text{exp}_1\ \text{exp}_2 \Downarrow value_3}$$

Alternate Design

- Why bother evaluating an argument that might not be needed?

$$\frac{exp_1 \Downarrow ((var) \Rightarrow \text{exp}_3) \quad exp_3[var \leftarrow exp_2] \Downarrow value_3}{\text{exp}_1\ \text{exp}_2 \Downarrow value_3}$$

3 Alternatives for Parameters

(more to come)

- **Call-by-Value**
- **Call-by-Name**
- **Call-by-Need** (**Lazy**)

Example 1

- Consider the following SML definition:

```
fun optimizedMult (0,m) = 0
  | optimizedMult (1,m) = m
  | optimizedMult (2,m) = m+m
  | optimizedMult (n,m) = n*m
```

- How many additions happen in:

```
optimizedMult(0,1+2) ?

optimizedMult(2,1+2) ?
```

ALGOL 60 was Call-by-Name

- Evaluate function calls by (capture-avoiding!) substituting in arguments: "copy rule"

```
real procedure double(n);
  integer n;
begin
  integer i;
  i := 7;
  id := n+n;
end id;
...
i := 2;
m := double(3);
n := double(A[i+i]);
```

Jensen's Device

```
real procedure sum(expr,i,low,high);
  real expr;
  integer i, low, high;
begin
  real rtn;
  rtn := 0;
  for i := low step 1 until high do
     rtn := rtn + expr;
  sum := rtn;
end sum

...
y := sum(x*x,x,1,10)
w := sum(sum(B[j,k],j,1,20),k,1,20)
```

Evaluation in ALGOL 60

- The following code looks ok.

```
procedure swap(a,b);
  real a, b;
begin
  real temp;
  temp := a;
  a := b;
  b := temp;
end swap;
...
swap(n,m);
swap(A[1],A[2]);
```

Evaluation in ALGOL 60

- But the code doesn't work for all inputs
  - One *cannot* write swap to work for all inputs!
- What's the problem?

- Common (but not universal) conclusion:
  - Having call-by-name in languages with assignment and other side-effects is too confusing.
  - Also carries implementation overhead.

Evaluation in Haskell

- The Haskell language is very like SML, but delays evaluating arguments.
  - No side-effects
  - Slightly different concrete syntax: compare with Example 2.

```
loop() = loop()

f _ = 3

x = f (loop())
```

Call by name
or
call by need?

Lazy Languages

- Haskell is a *lazy* functional language: no expression is evaluated until its value is really needed
  - Some very bright people *define* functional to imply pure and call-by-name/need
  - Has some nice advantages:
    - Definitions of infinite data structures

```
nats = 0 : map (\x -> x+1) nats
```

    - Programs really do behave mathematically

```
f(exp) + f(exp)  ≡  2*f(exp)
0 * f(exp)       ≡  0
```

Other Alternatives (for L-value arguments)

- **Call-by-reference**

- Call-by-value/result

Call-By-Reference

- Used by FORTRAN, optional in Pascal and C++
  - *Implicitly* passing pointers to the arguments
  - Formal parameters are aliases of the actual parameters.

```
procedure p(a : integer, var b : integer);
begin
   writeln (a, b);  { prints variables }
   a := 34;
   b := 34;
end;

n := 17; m := 18;  { Initial values }
p(n,m);
p(n,m);
```