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:

```
exp_1 ↓ ((var) => exp_3)
exp_2 ↓ value_2
exp_3[var←value_2] ↓ value_3
exp_1 exp_2 ↓ value_3
```
Alternate Design

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

\[
\begin{align*}
\text{exp}_2 \downarrow \{(\text{var}) \Rightarrow \text{exp}_3\} \\
\text{exp}_2[\text{var} \leftarrow \text{exp}_3] \downarrow \text{value}_3 \\
\text{exp}_1 \text{exp}_2 \downarrow \text{value}_3
\end{align*}
\]

3 Alternatives for Parameters

(more to come)

• Call-by-Value

• Call-by-Name

• Call-by-Need (Lazy)

Example 1

• Consider the following SML definition:

\[
\text{fun optimizedMult} (0,m) = 0 \\
\text{optimizedMult} (1,m) = m \\
\text{optimizedMult} (2,m) = m+m \\
\text{optimizedMult} (n,m) = n*m
\]

• How many additions happen in:

\[
\text{optimizedMult} (0,1+2) \\
\text{optimizedMult} (2,1+2)
\]

ALGOL 60 was Call-by-Name

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

\[
\begin{align*}
\text{real procedure double(n)}; \\
\text{integer n}; \\
\begin{align*}
\text{begin} \\
\text{integer i}; \\
i := 7; \\
id := n+n; \\
\text{end id}; \\
\ldots \\
i := 2; \\
m := \text{double}(3); \\
n := \text{double}(A[i+1]);
\end{align*}
\end{align*}
\]
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
    - Programs really do behave mathematically

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

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);
```