Parameter Passing and Evaluation Strategy

Prof. Evan Chang
Meeting 22, CSCI 3155, Fall 2009

Announcements

• Come to office hours to see your exam
• Study them and come see us to make sure you understand all of the problems
• HW7 due tonight

Review Parameter Passing

One-Slide Summary

• Procedures/functions/subprograms allow code to be reused.
• On a call, formal parameters used by the callee are bound to actual parameters (or arguments) provided by the caller.
• In eager evaluation, actuals are evaluated before a call. In lazy evaluation, they are evaluated only when used.
• Pass-by-value, pass-by-reference, pass-by-value-result are parameter passing modes in an eager setting. Pass-by-name gives rise to lazy evaluation.

Formal Parameters vs. Actual Parameters

Parameter passing modes

• in — take as input
  pass a value from caller to callee
  “normal ones”
• out — return as output
  pass a value from callee to caller
  (like return value)
• in-out — do both
  pass a value caller → callee and
  pass a value back from callee → caller
Eager evaluation vs. Lazy evaluation

- Arguments are evaluated to values before a call is evaluated.

\[
\begin{align*}
\text{val } x &= 3, y = 4, z = 5 \rightarrow \text{add}(x-y, z) & \rightarrow & 1 + \frac{z}{x} \\
\text{fun } \text{add}(a, b) &= a + b \\
\text{fun } \text{mul}(a) &= \text{add}(a, a) \\
\text{val } a &= 3 \\
\text{add}(a, 2, 1) &\rightarrow \text{add}(24, 2, 1) \\
\end{align*}
\]

Subtyping

\[
\begin{align*}
\text{void } f(S a) \{ \ldots \} \\
\text{subtyping } S \Rightarrow \text{f(c)} \{ \ldots ; \text{return } -3 \} \\
T t; \\
f(t); \\
in T < S \\
out S \otimes T \\
in-out T < S \text{ and } S < T \\
(T = a) \\
\end{align*}
\]

Pass-by-value

- `value is copied in` by the callee.

\[
\begin{align*}
\text{fun } \text{add}(x, y) &= x + y \\
\text{add}(3, -4, 5) &\rightarrow \text{add}(3 + (-4), 5) \\
&\rightarrow [x+3, y+5] \times y \\
\text{val } a &= 24 \\
\text{add}(a + 2, 1) &\rightarrow \text{add}(24 + 2, 1) \\
\end{align*}
\]

Pass-by-result

- Without out parameters or globals, we can typically still get information from a callee, how?

\[
\begin{align*}
\text{void } f(\text{out } m + x) \{ \\
x &= 42, 3 \\
\text{int } y &= 0; \\
f(y) &\text{ return } f(\text{out } y) \\
\text{println } y \\
\end{align*}
\]

Pass-by-result

- We never had a need for out parameters in SML, why?

\[
\begin{align*}
\Rightarrow \text{out param } = \\
\text{more than I return value} \\
= f(-1, -1) \\
\Rightarrow f : \text{int} \rightarrow \text{int} \times \text{int}
\end{align*}
\]
Pass-by-result

```plaintext
void f(int x, int y) {
    y = 2; x = 1; i = 1;
    m = a[0];
    printf(a);
}
```

Pass-by-value-result

```plaintext
void f(int x) {
    x = x + 1;
}
```

Pass-by-reference

```plaintext
int a = 0;
```

Pass-by-reference

```plaintext
int y = 1; x = 2;
```

Exercises

- Write a MYSTERY program to distinguish between call-by-value and call-by-reference

```plaintext
VAR a : INTEGER;
PROCEDURE f( ) =
BEGIN

END;
BEGIN

PRINT a;
END
```

Exercises

- Write a MYSTERY program to distinguish between call-by-reference and call-by-value-result

```plaintext
VAR a : INTEGER;
PROCEDURE f( ) =
BEGIN

END;
BEGIN

PRINT a;
END
```
For Next Time

- Reading
- Forum comment
- Start HW8