Week 7 Thursday Lecture

Will Christensen, Ben Verney

November 2019

1 Memory Management

1.1 Parameter Passing

In the previous section, we went over the details of function calls, particularly the differentiation between formal and actual parameters. With this foundation, we can now talk in depth about parameter passing techniques. That said, there are four types to consider:

1. Call by value - Evaluates arguments
2. Call by reference - Evaluates arguments
3. Call by value result - Evaluates arguments
4. Call by name - No evaluation, just Beta reduction!

2 Examples

2.1 Consider the C function swap

```
void swap(int x, int y) {
    int z;
    z = x;
    x = y;
    y = z;
}
int a = 2;
int b = 3;
swap(a, b);
```

Note that C and C++ are languages that call by value, but in C++ you can simulate call by reference as well. In ML, L-value stores the 'a ref, which is the location and the R-value stores type 'a, which is the actual value.

2.2 Call by value

Create temp variables (t and w) to hold the values of our actual parameters.

```
swap(x, y)
    let t = ref x
    let w = ref y
    z : int ref
    z := !t
    t = !w
    w = !z

a = 2
b = 3
swap(!a, !b)
```

In call by value, we change the values of the temporary variables, not the values of a or b, so the function essentially does nothing. So a = 2 and b = 3

2.3 Call by reference

```
swap(x : int ref, y : int ref)
    z = int ref
    z = !x
    x = !y
    y = !z

a = 2
b = 3
swap(!a, !b)
```

In call by reference, we are changing the values that a and b point to, so: a = 3 and b = 2

2.4 Potential Exam Question

```
int f(int x) {
    return f(1);
}

int zero(int x) {
    return 0;
}

void main() {
    print(zero(f(1));
}
```

This will infinite loop in call by value, reference, and value result because they evaluate arguments.

Call by name will return 0.

2.5 Another example to consider

```
int i = 10
void foo(int x, int y) {
    i = y;
}

main() {
    int A[20];
    i = 2;
    A[i] = 99;
    foo(i, A[i]);
}
```

In this function call, the global i would become 99 after we call foo in main, but the value of A[2] will not change. What if we were to call foo(@i, @A[i])? You could write (@ = address):

```
i = ref 10
foo(x: int ref, y: int ref) = i = !y
A: (int list) ref
i := 2;
A[i] := 99;
foo(i, A[i])
```

3 Call By Value Result

Let's go back to swap, there are two ideas to consider when discussing this type of parameter passing. The copy-in-phase and the copy-out-phase. Four steps utilize these two concepts:

1. Compute r-value of actual parameters
2. Bind r-values to formal parameters
3. Compute l-values of i and A[i] (next example)
4. Save l-values

In general, the copy-in-phase involves the first three steps, and the copy-out-phase involves the last. Note that in the copy-out-phase, the value of the form of the parameters are copied back to the l-values. Here's an example:

```
swap(i, A[i]) =
    x = i     //copy in phase
    y = A[i]  //copy in phase
    px = &i   //copy in phase
    py = &A[i]

    z = x
    x = y
    y = z

    *px = x     //copy out phase
    *py = y     //copy out phase
```

4 Comparing the types of parameter passing

Consider this example:

```
var x: integer
x := 0
P(value-result y: integer)
    y := 1
    x := 0

P(x) // Call P with parameter x
```

P(x)	CBV	CBR	CBVR
x = ?	0	0	1
y = ?	null	0	1