Handout E: Pass-By-Need

Christoph Reichenbach

November 15, 2019

The textbook covers several parameter passing modes:

- **Pass-by-Value**
- **Pass-by-Return**
- **Pass-by-Value-Return**
- **Pass-by-Reference**
- **Pass-by-Name**

These are also (alternatively) known as *Call-by-Value* etc.

1 Pass-by-Name

Recall that **Pass-by-Name** is a parameter passing mechanism that passes, as parameter, a *closure*, i.e., a piece of code that evaluates to the passed result. For example, consider the following piece of code in a Python-like language:

```
a = ["red", "green", "blue"]  # a0
x = 1

def P(y):
  a = y               # Point B
  x = 0
  return a + y        # Point C

P(a[x])               # Point A
```

If we use Pass-by-Name with static scoping, then that means that the procedure call at Point A in this program binds the formal parameter y to the expression a[x]. Pass-by-Name defers evaluating a[x] until we need it, so we only really evaluate a[x] at Point B. Since we are using static scoping, the variable a will bind to the list at a0. We will thus load the value "green".

However, when we access y again, at Point C, x has been updated to 0. Thus, at Point C, $y \Downarrow$ "red", meaning that P returns "greenred".

Compared to Pass-by-Value, Pass-by-Name has the advantage that if we don't need the parameter, we will not waste time evaluating it (analogously to short-circuit evaluation). It has the disadvantage that when we *do* need the parameter, evaluating it is slower than for Pass-by-Value, and when we need the parameter more than once, we have to keep re-evaluating the actual parameter.

However, Pass-by-Name adds expressivity to the language: as we saw in the example above, we can use parameters that are passed by name to pass a computation, rather than a value, and have that computation adapt to changes in the variables that the computation depends on.

This form of 'passing a computation' as a parameter is widely available in modern languages, but usually in the form of passing functions as parameters, i.e., hidden behind an additional layer of syntax and type checking.

2 Pass-by-Need

A sixth parameter passing mechanism that is used in the Haskell language is **Pass-by-Need**. This mechanism is similar to Pass-by-Name, except that it evaluates the actual parameter *at most once*. In our previous example, this means that after evaluating `y` to `"green"`, the language would store the value `"green"` as the result of `y`, and re-use it at Point C, ignoring the update to variable `x`.

Pass-by-Need is thus more efficient than Pass-by-Name, but also less flexible in the presence of updates to variables. However, Haskell does not have a notion of updatable variables (in the usual sense), so this limitation is not relevant for Haskell.