Naming and State

A4M36TPJ, 2014/2015

Naming Features

- Nameable values
- Parameter-passing mechanisms
- Scoping
- Name control
- Multiple namespaces
- Name capture
- Side effects

Parameter Passing

- call-by-name - a formal parameter names the computation designated by an unevaluated argument expression. Normal-order reduction strategy. (Haskell)
- call-by-value - a formal parameter names the value of an evaluated argument expression. Strict argument evaluation strategy. (C, Java, Pascal)

Call-by-name vs. Call-by-value

<table>
<tr><th>CBN</th><th>CBV</th></tr>
<tr><td>(app (lam x (prim * x x))
     (prim + 2 3))
$\overset{}{\underset{CBN}{\Longrightarrow}}_{[\beta]}$ (prim * (prim + 2 3)
              (prim + 2 3))
$\underset{CBN}{\Longrightarrow}_{[+]}$ (prim * 5 (prim + 2 3))
$\underset{CBN}{\Longrightarrow}_{[+]}$ (prim * 5 5)
$\underset{CBN}{\Longrightarrow}_{[*]}$ 25</td><td>(app (lam x (prim * x x))
     (prim + 2 3))
$\underset{CBV}{\Longrightarrow}_{[+]}$ (app (lam x (prim * x x))
         5)
$\underset{CBV}{\Longrightarrow}_{[\beta\text{-value}]}$ (prim * 5 5)
$\underset{CBV}{\Longrightarrow}_{[*]}$ 25</td></tr>
<tr><td>(app (lam x 2) (prim / 1 0))
$\underset{CBN}{\Longrightarrow}_{[\beta]}$ 2</td><td>(app (lam x 2) (prim / 1 0))
{This stuck expression models an error}</td></tr>
<tr><td>(app (lam x 3)
     (app (lam a (app a a))
          (lam a (app a a))))
$\underset{CBN}{\Longrightarrow}_{[\beta]}$ 3</td><td>(app (lam x 3)
     (app (lam a (app a a))
          (lam a (app a a))))
$\underset{CBV}{\Longrightarrow}_{[\beta\text{-value}]}$
(app (lam x 3)
     (app (lam a (app a a))
          (lam a (app a a))))
$\underset{CBV}{\Longrightarrow}_{[\beta\text{-value}]}$ ... {Infinite loop}</td></tr>
</table>

Call-by-denotation (CBD)

- Call-by-name determines the meaning of an operand expression relative to the environment available at the **point of call.**

- Call-by-denotation instead determines the meaning of an operand expression relative to the environment **where the formal parameter is referenced**.

CBD Example

```
(app (lam y
         (app (lam x y)
               3))
      x)
```

- Error in Call-by-name or Call-by-value (because x is unbound).
- In CBD, the unevaluated outer x is effectively substituted for y.

CBD Example

```
(app (lam x x) 3)
```

- CBD allows name capture.
- The evaluation of the outer x yields not what we would normally think of as a value but an environment accessor that is eventually applied to an environment that has a binding for the inner x.

Static Scope

```
function f(int a) {

    function g(int b) {
        return a + b;
    }

    return a + g(3);
}
```

- In a statically scoped language, every variable reference refers to the variable introduced by the **nearest lexically enclosing variable declaration** of that identifier in the abstract syntax tree of the program.

Dynamic Scope

- A free variable in a procedure (or macro) body gets its meaning from the environment at the point **where the procedure is called** rather than the environment at the point where the procedure is created.

- In these languages, it is not possible to determine a **unique declaration corresponding to a given free variable reference**; the effective declaration depends on where the procedure is called.

Dynamic Scope

```
(let ((a 1))
  (let ((f (abs (x) (@+ x a))))
    (let ((a 20))
      (f 300))))
```

- In static scope a in f refers to 1, where the f was defined. The result is 301.
- In dynamic scope a in f refers to 20, where the f was called. The result is 320.

Multiple Namespaces

```
class X {
    int x;

    X(int x) {
        this.x = x;
    }

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

State

- Purely functional languages and math are **stateless**.
- We can model state in functional languages as **an iteration** over states.
- **An iteration** is a computation that characterizes the state of a system in terms of the values of a set of variables known as its **state variables**.
- The value of each **state variable** in **an iteration** at time **t** is a **function of the values** of the state variables at time **t – 1**.

State

$$max : N^* \to N$$
$$max(\langle a_1, \ldots, a_n \rangle) = loop(\langle a_1, \ldots, a_n \rangle, 1, 0)$$
$$loop : N^* \times N \times N \to N$$

$$loop(\langle a_1, \ldots, a_n \rangle, c, m) = m \quad \text{if } c > n$$
$$loop(\langle a_1, \ldots, a_n \rangle, c, m) = loop(\langle a_1, \ldots, a_n \rangle, c+1, m) \quad \text{if } c \leq n \wedge a_c \leq m$$
$$loop(\langle a_1, \ldots, a_n \rangle, c, m) = loop(\langle a_1, \ldots, a_n \rangle, c+1, a_c) \quad \text{otherwise}$$

Monadic Style

- Monadic style separates state handling code.
- The name “monadic style” is derived from an algebraic structure, **the monad**, that captures the essence of manipulating information that is single-threaded through a computation.

Monadic Style Example

$$State = N^* \times N \times N$$
$$Action = State \to State$$
$$Condition = State \to Boolean$$

$$updateMax : Action$$
$$updateMax(\langle a_1, \ldots, a_n \rangle, c, m) = (\langle a_1, \ldots, a_n \rangle, c, a_c)$$

$$updateNeeded : Condition$$
$$updateNeeded(\langle a_1, \ldots, a_n \rangle, c, m) = true \quad \text{if } a_c > m$$
$$updateNeeded(\langle a_1, \ldots, a_n \rangle, c, m) = false \quad \text{otherwise}$$

$$increaseIndex : Action$$
$$increaseIndex(\langle a_1, \ldots, a_n \rangle, c, m) = (\langle a_1, \ldots, a_n \rangle, c + 1, m)$$

Monadic Style Example

$$notFinished : Condition$$
$$notFinished(\langle a_1, \ldots, a_n \rangle, c, m) = false \quad \text{if } c > n$$
$$finished(\langle a_1, \ldots, a_n \rangle, c, m) = true \quad \text{otherwise}$$

$$ifStatement : Condition \times Action \rightarrow Action$$
$$ifStatement(cond, body) = \lambda s.body(s) \quad \text{if } cond(s) = true$$
$$ifStatement(cond, body) = \lambda s.s \quad \text{otherwise}$$

$$forLoop : Condition \times Action \times Action \rightarrow Action$$
$$forLoop(cond, iter, body) = \lambda s.ifStatement(cond,$$
$$forLoop(cond, iter, body))(iter(body(s)))$$

Monadic Style Example

$$max : N^* \to N$$

$$max(\langle a_1, \ldots, a_n \rangle) = \pi_3(forLoop(notFinished, increaseIndex,$$
$$ifStatement(updateNeeded, updateMax))$$
$$(\langle a_1, \ldots, a_n \rangle, 1, 0))$$