Lecture 22: Writing Interpreters 2/Undecidability

CSCI 101
Spring, 2019

Kim Bruce

PCF Semantics w/Environments

- Substitution slow & space consuming
- Can't handle terms w/free variables
- Environment allows to evaluate once.
- Meaning now separate set of values -- not just rewriting
- Meaning of function is closure, which carries around its environment of definition.

The Problem

- Program:
  - y = 4
  - f x = x + y
  - g (h) = let y = 5 in (h 2) + y
  - g(f)
- When evaluate (h 2), the needed y is out of scope!

Values of Answers

- Key difference w/ new interpreter
  - Update environment, not rewrite term!
  - Not destructive!
- Mutually recursive type definitions:

```
data Value = NUM Int | BOOL Bool | SUCC | PRED |
        ISZERO | CLOSURE (String, Term, Env) |
        THUNK (Term, Env) | ERROR (String, Value)
type Env = [(String, Value)]
```

Solving the Problem

- Program:
  - y = 4
  - f x = x + y
  - g (h) = let y = 5 in (h 2) + y
  - g(f)
- f *evaluates to* <fn x => x+y, [y->4]>
- g(f) *partially evaluates to* (h 2) + y *in environment where* env = [y->5, h-> <fn x => x+y, [y->4]>]

PCF Syntax & Semantics with Environments

```
env:: string -> value

(0) (id, env) ⇒ env(id)

(1) (n, env) ⇒ n   for n an integer.

(2) (true, env) ⇒ true, (false, e) ⇒ false

(3) (error, env) ⇒ error

(4) (succ, env) ⇒ succ, similarly for other initial functions

    (b, env) ⇒ true     (e1, env) ⇒ v
(5) ----------------------------------
      (if b then e1 else e2, env) ⇒ v
```

More PCF Semantics

```
    (b, env) ⇒ false   (e2, env) ⇒ v
(6) ------------------------------
    (if b then e1 else e2, env) ⇒ v

    (e1, env) ⇒ succ    (e2, env) ⇒ n
(7) -------------------------------
         ((e1 e2), env) ⇒ (n+1)

(8) ...

(9) ...
```

Revised PCF Semantics

```
                                          Closure(x,e,env)
(10)        ((fn x => e), env) ⇒ <fn x => e, env>

        (e1,env) ⇒ <fn x => e3, env'>   (e2, env) ⇒ v1
                  (e3, env'[v1/x]) ⇒ v
(11)    -----------------------------------------
                    ((e1 e2), env) ⇒ v   Thunk(rec x => e, env)

             (e, env[(rec x => e)/x]) ⇒ v
(12)         -----------------------------
               ((rec x => e), env) ⇒ v
```

See code on-line in
<u>PCFEnvInterpreter.hs</u>

Imperative Languages

Adding State For Assignment

$$\frac{(e1, ev, s) \Rightarrow (m, s') \quad (e2, ev, s') \Rightarrow (n, s'')}{(e1 + e2, ev, s) \Rightarrow (m+n, s'')}$$

$$\frac{(M, ev, s) \Rightarrow (v, s')}{(X := M, ev, s) \Rightarrow (v, s'[v / ev(X)])}$$

$$(\text{fn } x \Rightarrow M, ev, s) \Rightarrow (<\text{fn } x \Rightarrow M, ev>, s)$$

$$\frac{(f, ev, s) \Rightarrow (<\text{fn } x \Rightarrow M, ev'>, s'), \quad (N, ev, s') \Rightarrow (v, s''), \quad (M, ev'[v/X], s'') \Rightarrow (v', s''')}{(f(N), ev, s) \Rightarrow (v', s''')}$$

Summary of Operational Semantics

- Meaning of program is sequence of states go through during execution
- Useful for compiler writers, complexity analysis
- Ideal is abstract machine that is simple enough that it is impossible to misunderstand operation.
- Should be easy to map to any computer.

If have time, come back later to talk about axiomatic semantics