Lecture 11: PCF & Natural Semantics

CSC 131

Kim Bruce

Scope

- Range of instructions where identifier is known
- Static: Scope associated with static text of program.
- Dynamic: Scope associated with execution path of program.

Hole in Scope (Static)

```
program ...
    var M: integer;
    ....
    procedure A ...
        var M: array [1..10] of real;
        begin
            ...
        end;
begin
    ...
end.
```

Static vs Dynamic Scope

```
program ...
    var A : integer;

    procedure Y(B: integer);
        begin
            ...;
            B := A + B;
            ...
        end; {Y}
```

which

```
procedure Z(...);
    var A: integer;
    begin
        ...;
        Y(...);
        ...
    end; {Z}
begin {main}
    ...;
    Z(...);
    ...
end.
```

Symbol Table: Built compile-time or run-time?

Semantics

- Operational
- Axiomatic
- Denotational (*Mitchell text*)

Natural (*Operational*) Semantics

- Arithmetic expressions example on web page
  - ArithSemantics.hs
- How to interpret identifiers?
- Environment: Association list of id's & values.
- Semantics defined recursively on abstract syntax trees.

PCF

- Programming language for Computable Functions
- Includes recursive definitions
- Call-by-value (eager) semantics
- Function application as substitution
- Rewriting semantics

Semantics in English

- Semantics of `succ e`
  - Evaluate expression `e` to value `v`
    - return v+1
- Semantics of `if b then e1 else e2`
  - Evaluate `b`
    - if b evaluates to true return value of e1
      otherwise return value of e2

PCF Syntax & Semantics

```
e ::= x | n | true | false | succ | pred | iszero |
      if e then e else e | (fn x => e) | (e e) |
      rec x => e | let x = e1 in e2 end
```

(1) n ⇒ n *for n an integer.*

(2) true ⇒ true, false ⇒ false

(3) error ⇒ error

(4) succ ⇒ succ, *and similarly for the other initial functions*

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

More PCF Semantics

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

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

    e1 ⇒ pred    e2 ⇒ 0          e1 ⇒pred    e2 ⇒ (n+1)
(8) ----------------------        ------------------------
        (e1 e2) ⇒ 0                   (e1 e2) ⇒ n

    e1 ⇒ iszero    e2 ⇒ 0        e1 ⇒iszero    e2 ⇒ (n+1)
(9) -----------------------       -------------------------
      (e1 e2) ⇒ true                 (e1 e2) ⇒ false
```

More PCF Semantics

```
(10)        (fn x => e) ⇒ (fn x => e)

     e1 ⇒ (fn x => e3)      e2 ⇒ v1    e3[x:=v1] ⇒ v
(11) --------------------------------------------------
                   (e1 e2) ⇒ v
```

Call by value!

```
                 e[x:=rec x => e] ⇒ v
(12)             ---------------------
                   (rec x => e) ⇒ v
```

Like Y combinator!

Recursion

```
f n  = if (n == 0) then 1 else n*(f(n-1)
```

is written in PCF (assuming have already defined `mult`) as

```
rec f => fn n => if (isZero n) then 1
                               else mult n (f (pred n))
```

which is equivalent to

```
Y(λf. λn. cond (isZero n) 1 (mult n (f (pred n))))
```

Computed via unwinding.

Substitution-based Interpreter

```
data Term = AST_ID String | AST_NUM Int | AST_BOOL Bool
          | AST_SUCC | AST_PRED | AST_ISZERO
          | AST_IF (Term, Term, Term) | AST_ERROR String
          | AST_FUN (String, Term) | AST_APP (Term, Term)
          | AST_REC (String, Term)
```

- Key is to get right definition of substitution that matches static scope
- Interpreter code matches semantic rules
  - PCFSubstInterpreter.hs

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
PCFEnvInterpreter.hs