Lecture 21: Writing Interpreters

CSCI 101
Spring, 2019

Kim Bruce

With More Work ...

- Show anything computable by TM is computable by lambda calculus ...
  - or by RAM, or WHILE language, or ...
- ... and vice-versa!

Writing Interpreters

Natural (*Operational*) Semantics

- Arithmetic expressions example on web page
  - Start w/parse tree: 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

Computations on Machines

- Look at the following languages:
  - E_{DFA} ={<M> | M is a DFA and L(M)=∅}
    EQ_{DFA} = {⟨M,N⟩ | M and N are DFAs and L(M) = L(N)}
    A_{DFA} = {⟨M,w⟩ | M is a DFA and w ∈L(M)}
- Showed before (informally) that these are decidable.
- Can do the same for PDA's.
  - First & third decidable, second is not!

What about TM's

- We'll see corresponding sets not decidable
  - and perhaps even not semi-decidable.
- Two more sets:
  - H_{TM} = {⟨M,w⟩ | M is a TM which halts on input w}
    $TOTAL_{TM}$ = {<M> | M halts on all inputs}
  - See later that neither is decidable.

Decision Problems with TM's

- Look at following sets:
  - A_{TM} = {⟨M,w⟩ | M is a TM and w ∈ L(M)}
  - H_{TM} = {⟨M,w⟩ | M is a TM which halts on input w}
  - $TOTAL_{TM}$ = {M | M halts on all inputs}
  - E_{TM} ={M | M is a TM and L(M)=∅}
- Halting easy for most, but:
  - times3(x: positive integer) =
    while x ≠ 1 do:
    If x is even then x = x/2.
    else x = 3x + 1

Universe of discourse

- Easy to determine if have encoding of a TM, so we'll ignore it when take complements, etc., so our universe of discourse will only consider those with valid TM encodings.

Semi-decidable

- A_{TM} and H_{TM} both semi-decidable using UTM.
- Show H_{TM} not decidable.
  - Let E be candidate TM to decide H_{TM}. Show can't be right.
  - From E, create TM D s.t. if input w, create <w,w> and simulate E on it (*i.e, it treats input as if of form <M,w>*)
    - If E rejects then make D accept and if E accepts, D loops forever
  - Now run D on <D> *Diagonal Argument*
    - If <D,D> ∈ H_{TM} then D halts on D, so E rejected <D,D> & <D,D> ∉ L(E)
    - If <D,D> ∉ H_{TM} then D not halt on D, so E accepted, <D,D> ∈ L(E)
    - Either way, L(E) ≠ H_{TM} with <D,D> in one but not other.

Decidable and Semi-decidable

- *Earlier:* If L and its complement are both semi-decidable then it is decidable.
  - Corollary: Complement of H_{TM} is not semi-decidable
- Note, if H_{TM} were decidable then every SD language would be decidable.
- Lots of other languages not decidable:
  - L_0 = {<M,w> | M on w eventually writes a 0}
  - ...