

Lecture 13: PCF & Natural Semantics

CSC 131

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:
 - $y = 4$
 - $f\ x = x + y$
 - $g\ (h) = \text{let } y = 5 \text{ in } (h\ 2) + y$
 - $g(f)$
- 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:

- $y = 4$
- $f\ x = x + y$
- $g\ (h) = \text{let } y = 5 \text{ in } (h\ 2) + y$
- $g(f)$

- f evaluates to $\langle \text{fn } x \Rightarrow x + y, [y \rightarrow 4] \rangle$

- $g(f)$ partially evaluates to $(h\ 2) + y$ in environment where $\text{env} = [y \rightarrow 5, h \rightarrow \langle \text{fn } x \Rightarrow x + y, [y \rightarrow 4] \rangle]$

PCF Syntax & Semantics with Environments

$\text{env} :: \text{string} \rightarrow \text{value}$

- (0) $(\text{id}, \text{env}) \Rightarrow \text{env}(\text{id})$
- (1) $(n, \text{env}) \Rightarrow n$ for n an integer.
- (2) $(\text{true}, \text{env}) \Rightarrow \text{true}, (\text{false}, \text{env}) \Rightarrow \text{false}$
- (3) $(\text{error}, \text{env}) \Rightarrow \text{error}$
- (4) $(\text{succ}, \text{env}) \Rightarrow \text{succ}$, similarly for other initial functions
- (5)
$$\frac{(b, \text{env}) \Rightarrow \text{true} \quad (e_1, \text{env}) \Rightarrow v}{(\text{if } b \text{ then } e_1 \text{ else } e_2, \text{env}) \Rightarrow v}$$

More PCF Semantics

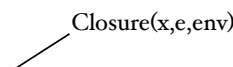
$$(6) \frac{(b, \text{env}) \Rightarrow \text{false} \quad (e_2, \text{env}) \Rightarrow v}{(\text{if } b \text{ then } e_1 \text{ else } e_2, \text{env}) \Rightarrow v}$$

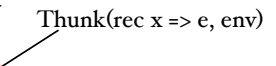
$$(7) \frac{(e_1, \text{env}) \Rightarrow \text{succ} \quad (e_2, \text{env}) \Rightarrow n}{((e_1\ e_2), \text{env}) \Rightarrow (n+1)}$$

(8) ...

(9) ...

Revised PCF Semantics

- (10) $((\text{fn } x \Rightarrow e), \text{env}) \Rightarrow \langle \text{fn } x \Rightarrow e, \text{env} \rangle$

- (11)
$$\frac{(e_1, \text{env}) \Rightarrow \langle \text{fn } x \Rightarrow e_3, \text{env}' \rangle \quad (e_2, \text{env}) \Rightarrow v_1}{(e_3, \text{env}'[v_1/x]) \Rightarrow v}$$
- (12)
$$\frac{(e, \text{env}[(\text{rec } x \Rightarrow e)/x]) \Rightarrow v}{((\text{rec } x \Rightarrow e), \text{env}) \Rightarrow v}$$



Imperative Languages

Adding State For Assignment

$$\frac{(e_1, ev, s) \Rightarrow (m, s') \quad (e_2, ev, s') \Rightarrow (n, s'')}{(e_1 + e_2, ev, s) \Rightarrow (m+n, s'')}$$

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

$$(fn x => M, ev, s) \Rightarrow (< fn x => M, ev >, s)$$

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

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.

Axiomatic Semantics

- No model of computation.
- Specification of meaning via pre- and post-conditions:
 - {P} stats {Q}
 - If P is true before executing stats and computation halts, then Q will be true at end.

Axiomatic Rules

- Assignment axiom:
 - $\{P \text{ [expression / id]}\} \text{id} := \text{expression} \{P\}$
 - Ex: $\{a+47 > 0\} x := a+47 \{x > 0\}$
 - $\{x > 1\} x := x - 1 \{x > 0\}$
- While rule:
 - If $\{P \ \& \ B\}$ stats $\{P\}$, then
 $\{P\}$ while B do stats $\{P \ \& \ \text{not } B\}$
 - P is *invariant* of loop.

Axiomatic Rules

- Composition rule:
 - If $\{P\} s_1 \{Q\}$, $\{R\} s_2 \{T\}$, and $Q \Rightarrow R$, then $\{P\} s_1; s_2 \{T\}$
- Conditional rule:
 - If $\{P \ \& \ B\} s_1 \{Q\}$, $\{P \ \& \ \text{not } B\} s_2 \{Q\}$,
 then $\{P\}$ if B then S1 else S2 $\{Q\}$
- Consequence rule:
 - If $P \Rightarrow Q$, $R \Rightarrow T$, and $\{Q\} s \{R\}$, then $\{P\} s \{T\}$

Correctness using Axioms & Rules



- Due to Bob Floyd & Tony Hoare
- Prove $\{\text{precondition}\} \text{Prog} \{\text{postcondition}\}$
- Usually work backwards from postcondition.

```
{Pre: exponent0 >= 0}
base <- base0
exponent <- exponent0
ans <- 1
while exponent > 0 do
{assert: ans * (base ** exponent) = base0 ** exponent0}
{
    & exponent >= 0}
    if odd(exponent) then
        ans<- ans*base
        exponent <- exponent - 1
    else
        base <- base * base
        exponent <- exponent div 2
    end if
end while
{Post: exponent = 0 & ans = base0 ** exponent0}
```

Steps in Proof

- Show
ans * (base ** exponent) = base₀ ** exponent₀
& exponent >= 0
is loop invariant
- Show postcondition follows from
(ans * (base ** exponent) = base₀ ** exponent₀
& exponent >= 0) & not(exponent > 0)
- Push invariant back to beginning of program.

Type Safety

- Is there any connection between type checking rules and semantics?
- If $E \vdash e: T$, what does that say about computation $(e, env) \Rightarrow v$?
- If E and env “correspond”, then expect $v: T$

Typed PCF

- $T ::= \text{Int} \mid \text{Bool} \mid T \rightarrow T$
- Provide identifiers w/type when introduced.
- $e ::= x \mid n \mid \text{true} \mid \text{false} \mid \text{succ} \mid \text{pred} \mid \text{iszero} \mid$
if e then e else e | (fn (x:T) => e) | (e e) |
rec (x:T) => e *Ignore recursion for now!*

Type-checking Rules

E is *type environment*: identifiers \rightarrow types

$E \vdash n: \text{Int}$, if n is an integer

$E \vdash \text{true}: \text{Bool}$, $E \vdash \text{false}: \text{Bool}$

$E \vdash \text{succ}: \text{Int} \rightarrow \text{Int}$, $E \vdash \text{pred}: \text{Int} \rightarrow \text{Int}$

$E \vdash \text{iszero}: \text{Int} \rightarrow \text{Bool}$

$E \vdash x: E(x)$