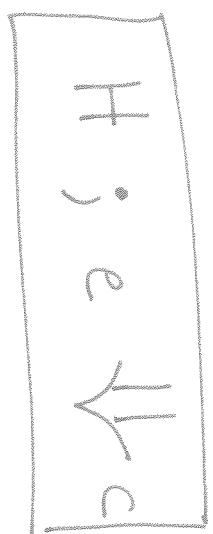


The Judgement



means "e evals to c under H"
just a set of tuples
"true" for tuples in set, false otherwise

- just a relation over $H \times E \times C$, i.e. \subseteq
- weird syntax follows PL convention, typical

Consider :

- , $x \mapsto 3 ; x + y \downarrow 3$
 - will be true (in our relation)
- , $x \mapsto 3 ; x + y \downarrow 6$
 - will be false (not in our relation)

Inference Rules : how we defn relations in PL

$$\frac{}{H; c \downarrow c} \text{const}$$

$$\frac{}{H; x \downarrow H(x)} \text{var}$$

instantiate
muvars
class

$$\frac{H; e_1 \downarrow c_1 \quad H; e_2 \downarrow c_2}{H; e_1 + e_2 \downarrow c_1 + c_2} \text{Add}$$

$$\frac{}{} \text{MUL}$$

Top : hypotheses / input

Bottom : conclusion / output

how to construct

- usually read bottom-up

- if hyps hold, then concl holds

- rules really "schema" you instantiate consistently

Example Instantiation

$$\begin{array}{l} \cdot, Y \mapsto 4; 3+Y \downarrow 7 \\ \hline \cdot, Y \mapsto 4; (3+Y)+5 \downarrow 12 \\ \cdot, Y \mapsto 4; 5 \downarrow 5 \end{array}$$

Instantiates ADD w/

$$H = \cdot, Y \mapsto 4$$

$$e_1 = 3 + Y$$

$$c_1 = 7$$

$$e_2 = 5$$

$$c_2 = 5$$

Derivations

complete derivation has no hyps dangling



...

We say $H; e \downarrow c$

if \exists derivation $w/ H; e \downarrow c @ \text{root}$

Precise Meaning for our Inference Rules

What relations are we defining? ^{we said all} this subset ^{start} what gives?

$$R_0 = \emptyset$$

(i>0) $R_i = R_{i-1} \cup \{ (t, e, c) \text{ s.t. } t, e, c \text{ derivable using an inference rule instantiated w/ hyps all in } R_{i-1} \}$

$\Rightarrow R_i =$ all triples @ bottom of ~~the~~ height j derivations for $j \leq i$ ($0 < ?$)

$$R = \bigcup_{i \geq 0} R_i \approx R_\infty$$

"smallest" relation closed under inference rules

Q: How prove 2 semantics equiv?

What have we done?

defined a sort of very precise,
abstract, interpreter

- each hyp in deriv dispatches recursive call

- "syntax directed" \rightarrow no need for interp
to search

\rightsquigarrow attach eval rule to each rule
of grammar, unambiguous

- proof system: prove facts from
other facts

Our Semantics is Nise (turner)

^{"Total"}
~~Profs~~: $\forall h, e, \exists c, H; e \downarrow c$

Deterministic: $H; e \downarrow c, \wedge h; e \downarrow c_2 \Rightarrow c_1 = c_2$

What would div do? gettime?

Profs by structural induction.

(\approx induction on height of e)

Onward: statements

similar

statements have exprs → use vars
→ need heap's to assign meaning

different

- statements don't eval to constants
- what do they do? A: produce new heap's!

Q: Always? A: No! non-termination!!

we could do something like:

$H_1; S \Downarrow H_2$

- but it'd be partial $H_1; S \rightarrow H_2$
- nothing wrong w/ that, problem?

Instead: Small-step semantics

- so single step of eval
- "iterate" to "run" the prog

write relation as

$$H_1; s_1 \rightarrow H_2; s_2$$

$H_1; s_1 \rightarrow H_2; s_2$

 $H; e \Downarrow c$

Assign

 $H; x := e \rightarrow H, x; e; \text{skip}$

 $H; s_1; s_2 \rightarrow H'; s_1'; s_2'$

seq

 $H; \text{skip}; s \rightarrow H; s$

seq

 $H; e \Downarrow c \quad c > 0$

 $H; \text{if } e \text{ then } s_1 \text{ else } s_2 \rightarrow H; s_1$

IFT

... IFF

What about while? (do s & loop if e > 0)

are these directed? syntax

$H; \text{while } e \rightarrow H;$ if $e (s; \text{while } e \text{ } s)$ step while

$[H; s \rightarrow H'; s']$,

so we defined but what does "s" mean / do?

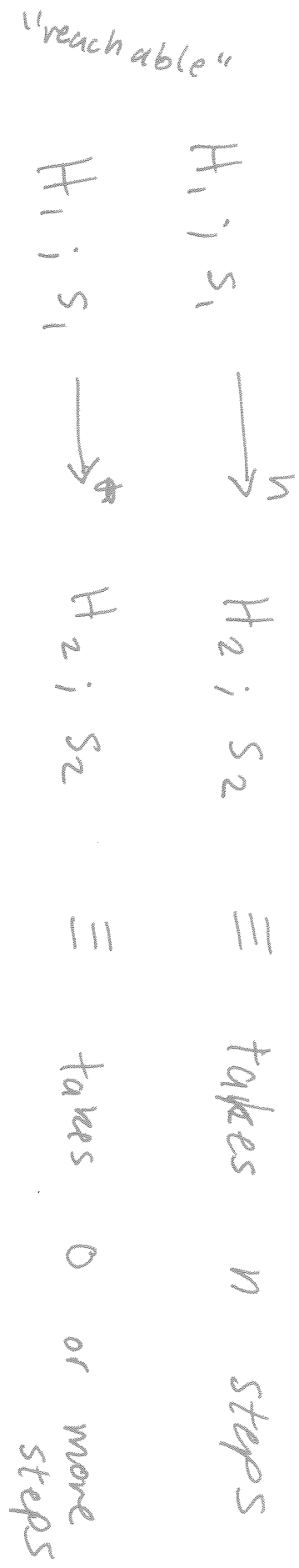
- It transforms a heap in one step
- which heap to start in?

We can iterate:

$H_1; s_1 \rightarrow H_2; s_2 \rightarrow \dots$

each step justified by complete derivation

Proof



"reflexive, transitive closure"

equivalence class? NO!

no symmetry

could pick special variable "ans" for answer

then "s produces c" if $\bullet; s \xrightarrow{k} H; \text{skip } \Lambda \text{ (trans)}$

Does every s produce a c?

Example program execution

```
x := 3; (y := 1; while x (y := y * x; x := x-1))
```

Let's write some of the state sequence. You can justify each step with a full derivation. Let $s = (y := y * x; x := x-1)$.

```
•; x := 3; y := 1; while x s
```

```
→ •, x ↦ 3; skip; y := 1; while x s
```

```
→ •, x ↦ 3; y := 1; while x s
```

```
→2 •, x ↦ 3, y ↦ 1; while x s
```

```
→ •, x ↦ 3, y ↦ 1; if x (s; while x s) skip
```

```
→ •, x ↦ 3, y ↦ 1; y := y * x; x := x - 1; while x s
```

Recap

IMP, Proofs

$H; e \Downarrow c$,
expers done
w/ "big step"

$H; s \rightarrow H'; s'$
stmts done
w/ "small step"

Semantics by interp:
- prog means what ^{very} abstract
- interp says it means.
- precise interp written in Meta language

Big Step limitations

- can't distinguish errors & divergence
- IMP has no errors
- EXPRs can't diverge

Proving Progs

can reason about progs by "running" them