Consider:

- Well-formed follows for relation, typically

\[ R \subseteq X \times X \]

Just a relation over \( X \times X \), i.e.,

- Just a set of tuples

The set of elements in a tuple.

A "true" for tuples in set, false otherwise.

\[ H \subseteq \text{e \{x, y\}} \]
Inference Rules: how we define relations in PL

Top: hypotheses / Input

Bottom: conclusion / Output


H ≤ C → H \not\subseteq C

H \lor C \lor C, H \not\subseteq C and

H \lor C \lor Ct, C \not\subseteq C

\therefore \forall x \in H \subseteq C

\therefore \forall x \in H \not\subseteq C

Rules really "scheme" you instantiate consistency - if hypotheses hold, then conclusion holds - usually read bottom up.
\[
\begin{align*}
    c_2 &= 5 \\
    e_2 &= 5 \\
    c_1 &= 7 \\
    e_1 &= 3 + y \\
    h &= y + 1 - y \\
\end{align*}
\]

Example: In saturation ABD w/
Derivations

complete derivation has no hyps dangling

\[ \because y \vdash 4; 3 \triangleright 3 \quad \therefore y \vdash 4; \gamma \downarrow 4 \]

We say \( H; e \downarrow \gamma \)

if \( \exists \) derivation w/ \( H; e \downarrow \gamma \) @ root
How prove a sentence is empty?

"smallest" relation closed under inference rules

\[ R = \bigcup_{i \geq 0} R_i \]

What gives? This subset of all inference rules defines meaning for our inference rules?
Proof system: prove facts from other facts.

of grammar, unambiguos

attach each rule to each rule

"Syntax directed" no need for interpreter

each hy in deriu dispartys recursive call

Abstract interpreter, sort of very precise,
defined on don?
(induction on height of e)

Proofs by structural induction.

What would I do if I get time?

Deterministic: H; => C, N, H; else C, => C, = C;

A, e, E, C, H, e, C

Our Semantics is NICE (translation)
nothing wrong w/ that, problem?

H, s ↑ Hz

we could do something like:


Statements don't equal to constants
different

assign meaning to need help to use vars

similar statements have expresssions

return: statements
Instead: Small-step Semantics

Write relation as

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

... IF

H : s ! s' 

H ! e \downarrow c 

C \downarrow 0

H ! e \downarrow c

H ! x \leftarrow H , x \in e

Assign

H ! e \downarrow c

H ! s \leftarrow H 2 ! s 2

H ! s 1 ! s 2
The syntax is not directed. These are directed.

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

H; while e < H; if e (s; while e; s) srip

H; s; H2; s2 < ...
Notation

$H_1; s_1 \rightarrow^n H_2; s_2 \equiv$ takes $n$ steps

$H_1; s_1 \rightarrow^* H_2; s_2 \equiv$ takes 0 or more steps

"reflexive, transitive closure"

equivalence class? No!

no symmetry

could pick special variable "ans" for answer

Then "S produces C" if $\cdot \cdot \cdot \rightarrow^* H; \text{skip } \land \#(\text{ans}) =$

Does every $s$ produce a $c$?
Example program execution

Let's write some of the state sequence. You can justify each step.

\[ x = x \rightarrow 1 \]

With a full derivation, let's write some of the state sequence. You can justify each step.
Recap

Imp, proofs

Hi etc.,

w/ "big step" exps done

His \rightarrow H's

w/ "small step"

starts done

semantics

by interp

Precise interp

written in metalanguage

Program means

What means

Very abstract

25
can reason about pages by "runnings" the Proving Pages

Experts can't disagree
IMF has no errors & divergence
- can't distinguish errors & divergence
Bigs Step Innovations