\[ x \in \{ x_1, x_2, \ldots, y_1, y_2, \ldots, z_1, z_2, \ldots, 3 \} \\
C \in \mathbb{Z} = \{ \ldots, -3, -2, -1, 0, 1, 2, \ldots, 3 \} \\
\]

```
| while e s |
| if e s |
| e + e |
| e - e |
| x |
| e :: c |
```

```
S :: Skip
```

Program is statement defined as follows:

- In common "metalanguage"
- In abstract syntax

- Instead of ints, mutable vars, control flow
- Instead of functions, objects, records, threads, exceptions

Formal language (= set of constructs defined by precise
Children are more ASTs – nodes have labels for constructors ("which are 
abstract syntax") means this defines a set of these

Meta variables mean anything in that set (syntax)

" \Rightarrow \" means "or"

" \Rightarrow \Rightarrow \" means "can be a"
(pretend all int = Z)

if exp

| Mul of exp exp
| Add of exp exp
| Var of String

type exp = Const of int

| While of exp stmt
| If of exp stmt stmt
| Seq of stmt stmt
| Assign of String exp

In Ocaml:
\[ \text{if } \text{skip} \quad (y := \text{h2}; \quad x := y) \]

\[ \text{vs.} \]

\[ \text{if } \text{skip} \quad (y := \text{h2}; \quad x := y) \]

Underneath, every thing always unambiguous tree.

However, concise the syntactically convenient, so we'll use parentheses to disambiguate.

Potentially ambiguous:

Normally write code in concrete syntax, i.e. strings

\[ \text{AST} \quad \text{VS.} \quad \text{String} \]
Remove ambiguity. Will add parentheses when necessary to
may write as strings for convenience.
Moving forward, always ASTs, but
lots of good frameworks / tools
Fewer papers these days
Heavily studied in 70's & 80's

`parse: string stmt`
Our grammar denotes E.

\[
E = \emptyset \cup E_0 \cup \{ E_1 \cup E_2 | E_1, E_2 \in E \}^{>0}
\]

Precise meaning for our metanotation:

- Always "bottoms out" function: uses self-reference but
  - Similar to all ways - terminating recursion
  - As we use well-founded induction.
  - Self-reference? Not a problem as long

\[\text{Inductive Definition}\]

\[\text{our grammar is finite description of}\]
\[\text{a set of trees.}\]
Proof: Consider $e = 1 + (2 + 3)$. Need to show $e \in E$.

Simple proofs (proving obvious stuff)

What is $E_1 \cap E_2$?

E_1 \cap E_2 = E_{1-2} \cap E_{1-2} = E_{1-2}$

What about $S_i \cap S_i$? $S_i \subseteq S_i$

Another case: $x \in \varnothing$
Certain ways. Exploit in project. We know e can only be built in

\[ e = e^3 + e \in 1 \text{ or } e^2 \]

\[ e = e^2 + e^2 \in 1 \text{ or } e^2 \]

\[ e = e \wedge e^2 \]

\[ e = e \wedge e \]

Ind. Suppose holds for \( i \), show for \( i + 1 \).

Base: \( i = 0 \implies e = 0 \). Vacuous

Proof: Induction on \( i > 0 \).

Thm: A e e, e contains at least 1 constant.

\[ \text{Consider } e = 1 + (2 + 3) \text{ and define } E \]

\[ \text{Proof 2 (Better)}: \]
No less convenient.

You get IH on "smaller" terms, (Just what we were doing w/ i).

1. e1 + e2 (assuming holds on e1, e2)
2. e1 + e2 (assuming holds on e1, e2)

By "structural induction", (rules for forming an e) on e.
\( \text{sub proof's!} \)

nodes in AST w/ errors by replacing

\( f(x) \) must show directly

\( \text{it } \times \) takes no args.

\( x, a_1, a_2 \ldots \) an

then prep holds on

gargs \( (a, 1, a_2) \ldots \) an prep

hold on its evidence that it

for each \( c \) provide

\( \text{exp - ind } \)

also first time
Intuitions as programmers.  

We want to precisely capture our notion of how to interpret them.  

So far only said what programs are.
Encode our notion of evaluation (meaning) as a triple over \( H, e, c \) and C.

Use heap \( H \) as total func: \( \text{var} \rightarrow \text{val} \).

Need a notion of memory or "heaps".

Depends on the values of variables:

\[
\begin{align*}
X + x & \quad \text{iii}\\
1 + a & \quad ?
\end{align*}
\]

Informally: Given expr \( e \), what does it equal to?
We will only discuss {\lesssim} r{\lesssim} m semi-metrics w.r.t. a heap $H$. 

- all mean initialized to 0
- avoids "errors"
- case forces lookup to be total

$x \neq y \land y \subseteq C \land \forall x \in C \Rightarrow H(x) \neq 0 \Rightarrow H(x) \not\in H, \iff H = H$,

$H(x) = \begin{cases} H, & \text{if } C \not\subseteq x \Leftrightarrow H, \\ 0, & \text{if } C \subseteq x \end{cases}$

look-up:

$H \leftarrow \text{Heaps}$
- Will be false (not in our relation)

\[ x + y \geq 6 \]

- Will be true (in our relation)

\[ x > 3 \; \; \text{and} \; \; x + y \geq 3 \]

Consider:

- Well formed follows PL convention, type: C

- Just a relation over \( H \times E \times C \), i.e.

\[ \exists \text{true for tuples in set, false otherwise} \]

Just a set of tuples means \( e \) equals to a under A

The judgment...