Foundations: program definition

- Syntax
  - *"... the rules and principles that govern the sentence structure of any individual language"* -- Wikipedia
  - Standard: base a programming language syntax on a *context free grammar*
- Semantics
  - *"the meaning or relationship of meanings of a sign or set of signs"* -- Merriam-Webster Dictionary
  - Standards for defining semantics vary

Context free grammar

- *Nonterminal symbols* for key syntactic categories
- A *start symbol* (a nonterminal)
- *Terminal symbols*, which appear in "sentences"
- *Productions*, rules showing how symbols compose to form sentences

Context free grammar

- *Nonterminal symbols* for key syntactic categories
- A *start symbol* (a nonterminal)
- *Terminal symbols*, which appear in "sentences"
- *Productions*, rules showing how symbols compose to form sentences

Example nonterminals:

P - programs ← start
S - statements
e - expressions

Example terminals:

0 3.14 15 'a' "hi" ... *literals*
if then else end... *keywords*
x y time len ... *identifiers*
< => + * / ... *operators*

Our examples will use:

num - for numeric literals
id - for identifiers
op - for operators

Context free grammar

- *Nonterminal symbols* for key syntactic categories
- A *start symbol* (a nonterminal)
- *Terminal symbols*, which appear in "sentences"
- *Productions*, rules showing how symbols compose to form sentences

Example productions

$e ::=$ num
| id
| e op e
| (e)

$S ::=$ read id
| write e
| id := e
| S ; S
| if e then S else S
| while e do S
| { S }

$P ::=$ prog S end

Derivation

A sequence of replacement steps, starting from the start symbol and producing a string of nonterminals

P ⟶ prog S end
⟶ prog S ; S end
⟶ prog read x; S end
⟶ prog read x; if e then S else S end
⟶ prog read x; if e > e then S else S end
⟶ prog read x; if x > e then S else S end
⟶ prog read x; if x > 0 then S else S end
⟶ . . .
⟶ prog read x; if x > 0 then write 0 else write 1 end

Derivation

A sequence of replacement steps, starting from the start symbol and producing a strin

P ::= prog S end

P ⟶ prog S end
⟶ prog S ; S end
⟶ prog read x; S end
⟶ prog read x; if e then S else S end
⟶ prog read x; if e > e then S else S end
⟶ prog read x; if x > e then S else S end
⟶ prog read x; if x > 0 then S else S end
⟶ . . .
⟶ prog read x; if x > 0 then write 0 else write 1 end

Derivation

A sequence of replacement steps, starting from the start symbol and producing a string of nonterminals

S ::= S ; S

P ⟶ prog S end
⟶ prog S ; S end
⟶ prog read x; S end
⟶ prog read x; if e then S else S end
⟶ prog read x; if e > e then S else S end
⟶ prog read x; if x > e then S else S end
⟶ prog read x; if x > 0 then S else S end
⟶ . . .
⟶ prog read x; if x > 0 then write 0 else write 1 end

Derivation

A sequence of replacement steps, starting from the start symbol and producing a string of nonterminals

S ::= read id

P ⟶ prog S end
⟶ prog S ; S end
⟶ prog read x; S end
⟶ prog read x; if e then S else S end
⟶ prog read x; if e > e then S else S end
⟶ prog read x; if x > e then S else S end
⟶ prog read x; if x > 0 then S else S end
⟶ . . .
⟶ prog read x; if x > 0 then write 0 else write 1 end

Derivation

A sequence of replacement steps, starting from the start symbol and producing a string of nonterminals

$P \longrightarrow$ prog S end
$\longrightarrow$ prog S ; S end
$\longrightarrow$ prog read x; S end
$\longrightarrow$ prog read x; if e then S else S end
$\longrightarrow$ prog read x; if $e >$ e then S else S end
$\longrightarrow$ prog read x; if x $>$ e then S else S end
$\longrightarrow$ prog read x; if x $>$ 0 then S else S end
$\longrightarrow$ . . .
$\longrightarrow$ prog read x; if x $>$ 0 then write 0 else write 1 end

$S ::=$ if e then S else S

Parsing

Parse tree:
- Root represents start symbol
- Nodes represent
  - Non-leaf ~ nonterminal
  - Leaf ~ terminal
- Parent-children represent application of a production
  - Parent ~ nonterm on LHS
  - Children ~ RHS of prod

Represents derivation(s)
- first litmus test for syntactic well-formedness
- shows the structure of the program