

Grammars

- are rules by which valid sentences can be constructed

nonterminal - variables that can be replaced/expanded to a sequence of variables and terminals

terminals - an actual word in the language they cannot be replaced.

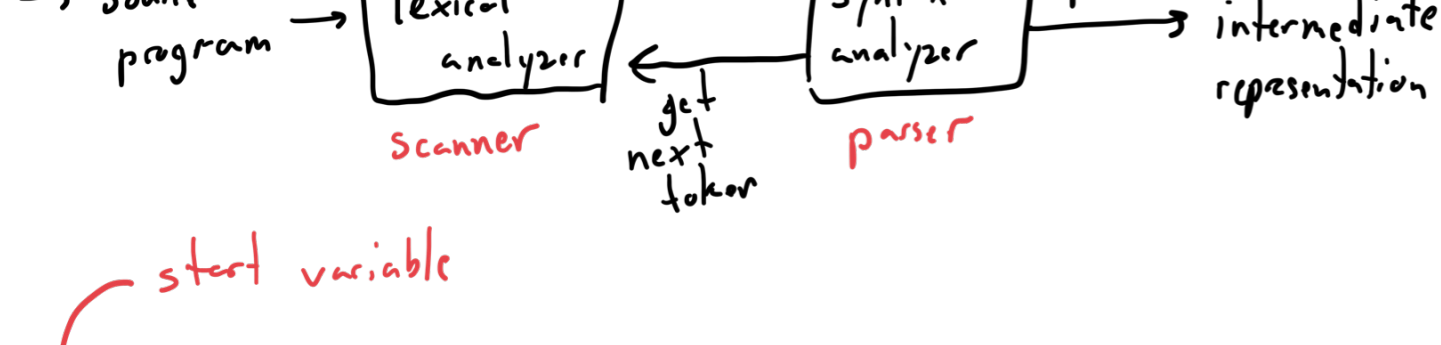
production (rule) - describes how to replace symbols, of the form $X \rightarrow Y_1 Y_2 Y_3 \dots$

derivation - sequence of applied rules to produce a string of terminals

- used in parsers

- yacc (yet another compiler compiler)
↳ bison

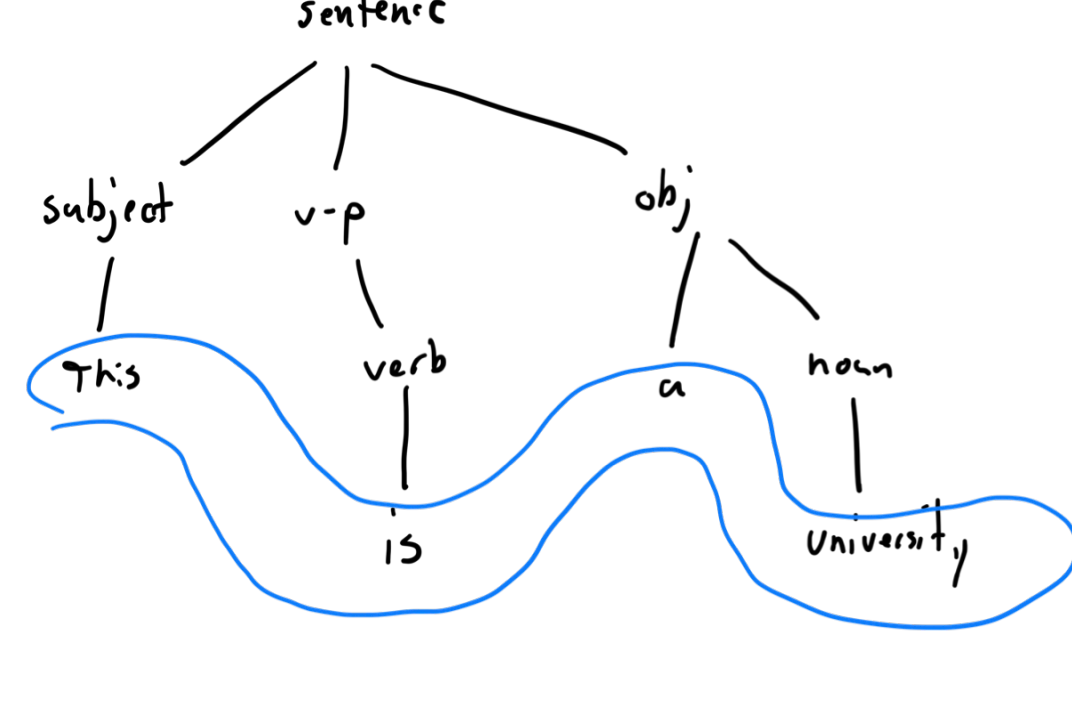
- lex/yacc $\Rightarrow$ Flex/Bison



productions

- sentence $\rightarrow$ <subject> <verb-phrase> <obj>
- subject $\rightarrow$ This | Computers | I
- verb-phrase $\rightarrow$ <adv-verb> <verb> | <verb>
- verb $\rightarrow$ is | run | am | tell
- obj $\rightarrow$ the <noun> | a <noun> | <noun>
- noun $\rightarrow$ university | world | lies

sentence $\rightarrow$ <subject> <verb-phrase> <obj>
 $\rightarrow$ This <verb-phrase> <obj>
 $\rightarrow$ This <verb> <obj>
 $\rightarrow$ This is <obj>
 $\rightarrow$ This is a <noun>
 $\rightarrow$ This is a university.

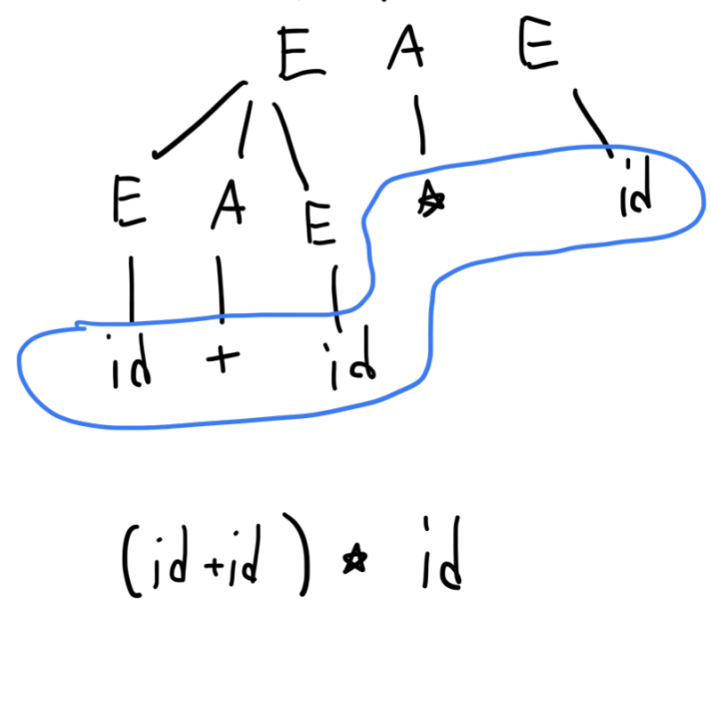
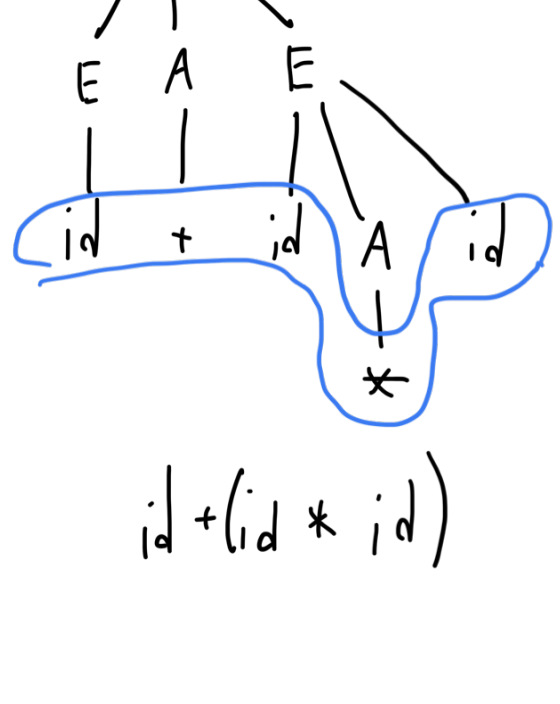


$E \rightarrow EAE \mid (E) \mid -E \mid id$

$A \rightarrow + \mid - \mid * \mid /$

(an ambiguous grammar)

$id + id * id$



- Bison uses certain rules to parse when given an ambiguous grammar, never generating more than 1 possible tree.

Thus...

Not all grammar languages can be recognized by a conventional parser.

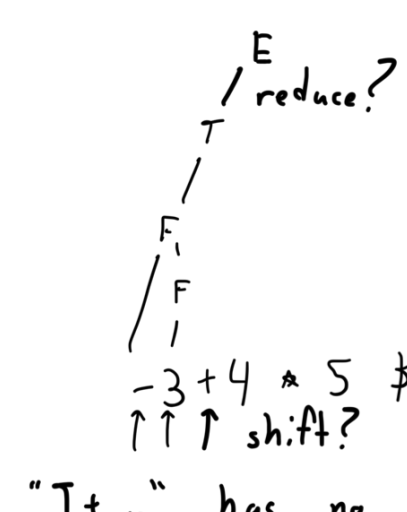
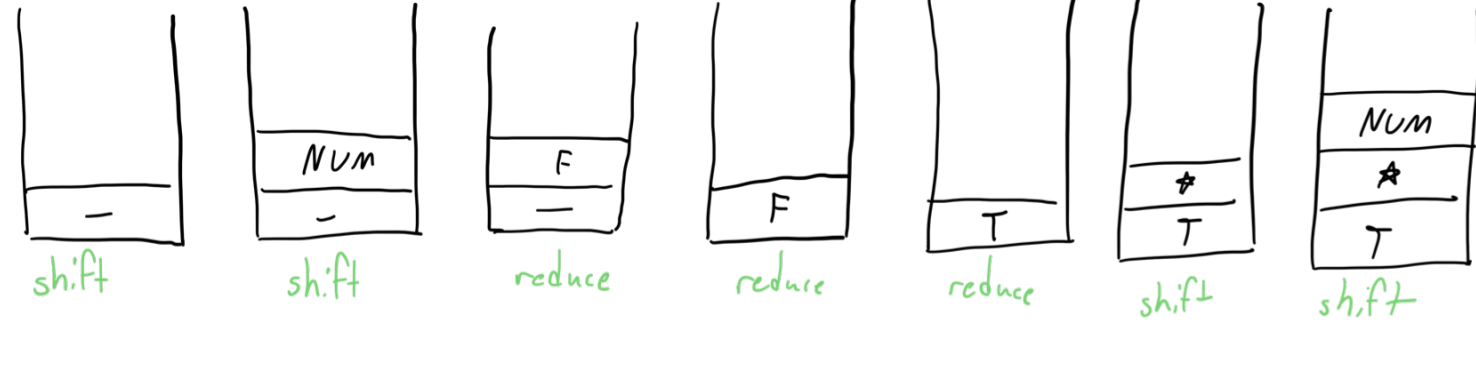
- Bison LALR (look ahead left-to-right, rightmost derivation parser.)

shift - scan right
reduce - reverse substitution

lowest precedence $\left\{ \begin{array}{l} S \rightarrow E \$ \\ E \rightarrow E + T \end{array} \right.$
medium $\left\{ \begin{array}{l} T \rightarrow T * F \\ T \rightarrow F \end{array} \right.$
highest precedence $\left\{ \begin{array}{l} F \rightarrow (E) \\ F \rightarrow -F \\ F \rightarrow NUM [0-9] \end{array} \right.$

Bison looks ahead at the next token to eliminate obviously invalid moves.

precedence is naturally defined by what is deeper in the grammar



"T+..." has no valid rule, so it knows shifting is a dead-end.
But, "E+..." exists, so reducing T to E is okay!

due to precedence what is lower in the grammar bounds more tightly.

$S \rightarrow E$
 $E \rightarrow E + F$
 $\rightarrow E * F$
 $\rightarrow F$
 $F \rightarrow (E)$
 $\rightarrow -E$
 $\rightarrow NUM$

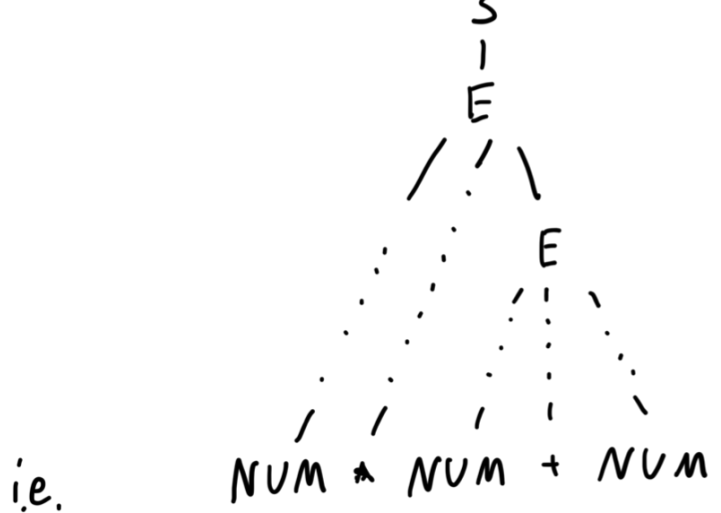
unlike above, both "E+..." & "E*..." are valid now!

Due to equal precedence, the + and * operators are bound via right-most order, consequently.

such as this...

$-3 + 4 * 5 \$ \rightarrow -3 + (4 * 5) \$$
 $-3 * 4 + 5 \$ \rightarrow -3 * (4 + 5) \$$

The parse trees will "bunch up" on the right-most side because Bison breaks ties by shifting and later reducing (when confronted with a shift/reduce conflict).



$S \rightarrow E$
 $E \rightarrow E + E$
 $\rightarrow E * E$
 $\rightarrow E$
 $\rightarrow (E)$
 $\rightarrow -E$
 $\rightarrow NUM$

This grammar is ambiguous (many parse trees exist for 1 input).
 Bison throws countless warnings at us. We need some precedence.

what if we add....

$S \rightarrow S$

then it's ambiguous!

