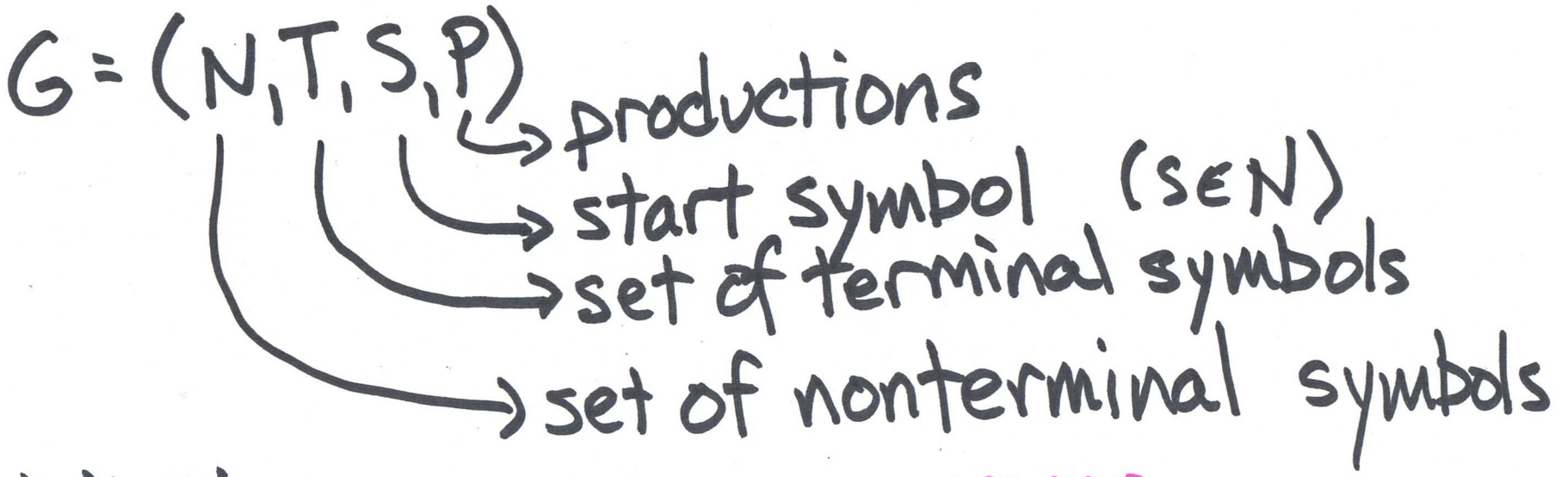
- context-free grammars
- derivations and parse trees
- parsing
  - left-factoring
  - predictive parsing example
- making a predictive grammar
  - removing left-recursion

→ chapter 3



productions:

$A \to \alpha$ $\quad A \in N$ $\quad \alpha$ is a string (reg. expr.) on $N \cup T \cup \{\varepsilon\}$

BNF: Backus-Naur Form

EBNF: regular expressions as α.

$E \to T + E$
$E \to T$
$T \to F * T$
$T \to F$
$F \to c$
$F \to i$
$F \to (E)$

$E \to T + E$
T

$T \to F * T$
F

$F \to c$
i
(E)

nonterminals: $\{E, T, F\}$

terminals: $\{+, *, c, i\}$ $(,)$

start symbol. E

$$E \to T+E \mid T$$
$$T \to F*T \mid F$$
$$F \to (E) \mid i \mid c$$

$A \to \alpha$ is a production for A

Derivation

$a_0 \ldots a_n$ strings of $(N \cup T \cup \{\varepsilon\})^*$ (sentential forms)

$a_0 = S$ start symbol

a_i goes to a_{i+1} by finding a production for a nonterminal of a_i and replacing that nonterminal with RHS of the production.

ends when a_i contains no nonterminals. a_i is a **sentence**

$E \Rightarrow T+E \Rightarrow T+T+E$

$\Rightarrow T+T+T \Rightarrow F+T+T \Rightarrow$

$i+T+T \Rightarrow i+F*T+T \Rightarrow$

$i+c*T+T \Rightarrow i+c*F+T \Rightarrow$

$i+c*i+T \Rightarrow i+c*i+F \Rightarrow$

$i+c*i+c$

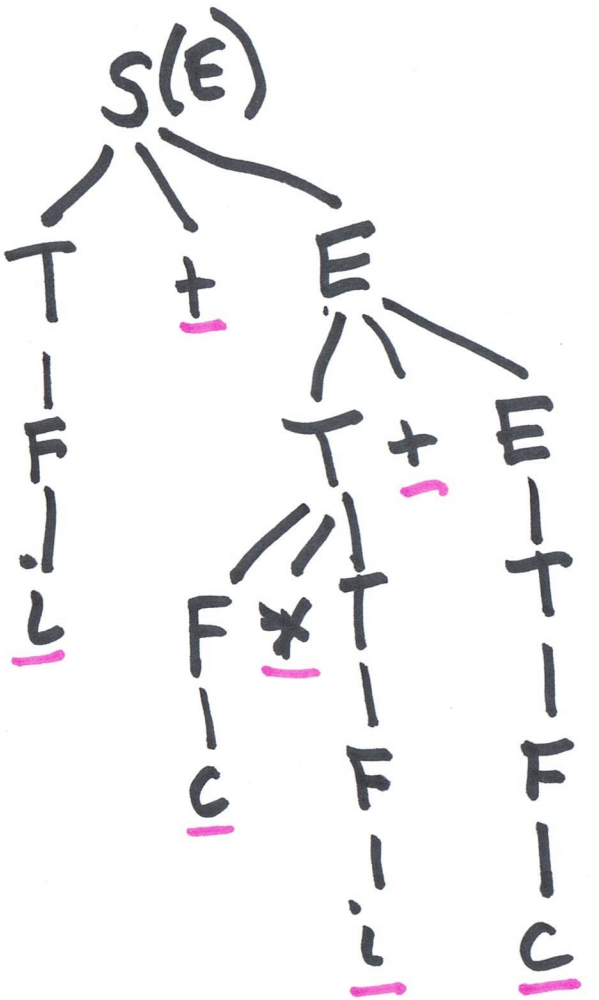
sentence

$E \overset{*}{\Rightarrow} i+c*i+c$

eg. $(f+3*g+4)$

Language of a grammar: set of all possible derived sentences.

A <u>parse tree</u> of a derivation:



i+c*i+c

parser: stream of tokens : terminals.

produces a derivation of the grammar that leads to the given set of tokens.

→ or parse tree

<stmt> → while (<condition>) <stmt>

for (<for control>) <stmt>

let <var> = <expr>

return <expr>

call ident (<exprList>)

⋮

Parsing is more difficult if two different productions for the same nonterminal start with the same token.

$E \to T+E \mid T$
$T \to F*T \mid F$
$F \to (E) \mid i \mid c$

→ $B \to A\bullet \mid A\alpha \mid A\beta \ldots$
$B \to AB'$
$B' \to \bullet \mid \alpha \mid \beta \ldots$

left-factoring

$B \to AB \mid Az \mid CC \mid Cx$
$B \to AA' \mid CA''$
$A' \to B \mid z$
$A'' \to C \mid x$

predictive: always tell the production by looking at next term.

$E \to TE'$
$E' \to +E \mid \varepsilon$
$T \to FT'$
$T' \to *T \mid \varepsilon$
$F \to (E) \mid i \mid c$

not left-factored

doStmt → do <stmt> while (<cond>) | doStmtVar

doStmtVar → do <stmt> until (<cond>)

doStmt → do <stmt> <doStmtEnd>

doStmtEnd → while(<cond>)
until(<cond>)