7 Context Free Grammars

Friday, February 21, 2025 12:16 PM

Why? Because RegEx are not powerful enough.
for programming languages:

([ ] { }) ¿? BEGIN ... END

Another appoach is needed.

- GRAMMARS

DEF: $G = \langle N, T, S, P \rangle$

where: N: set of non-terminal symbols
T: set of terminal symbols Σ
$N \cap T = \emptyset$ N.T are disjoint
$S \in N$: the start symbol
P: a set of "production rules"
rules of the form
$A \to \alpha$ where $A \in N$
$\alpha \in (N \cup T)^*$

i.e. α is a string made of terminals and non-terminals.
note α could be the empty string

- In a production $A \to \alpha$ (A: head, α: body)
- a shorthand: $A \to \alpha_1$, $A \to \alpha_2$, $A \to \alpha_3$ } $A \to \alpha_1 \mid \alpha_2 \mid \alpha_3$

E.G.

$N = \{A, B, J\}$
$T = \{x, y, z\}$
$S = A$

$P = \{$
$A \to x\,B\,J\,z$
$A \to x\,y$
$B \to z\,J\,y$
$J \to zzz$
$J \to Rx$

$S = A$

$$\begin{cases} J \to zzz \\ J \to Bx \end{cases}$$

How does a grammar specifies a Language?

Intuition: the production rules are rules to rewrite strings

e.g

$JxAyzB \xrightarrow{B \to zJy} JxAyzzJy \xrightarrow{A \to xy} JxxyzzJy$

DEF:

Sentence:- a string of terminals and non-terminals

Application of a rule R into sentence β

$R: A \to \alpha$

replacing one occuance of A in β by α

Derivation: a sequence of sentences each derived from the previous one by the application of a rule.

The language specified by grammar G

$w \in L$ iff there exists a derivation from the start symbol S to w

($w \in T^*$ i.e. w is made only of terminals)

E.G.

$N = \{A, B, J\}$

$T = \{x, y, z\}$

$S = A$

$$P = \begin{cases} A \to xBJz \\ A \to xy \\ B \to zJy \\ J \to zzz \\ J \to Bx \end{cases}$$

$L = \{xy, xzzzyzzzz, \ldots\}$

$A \to xy$

$A \to xBJz \to xzJyJz \to xzJyzzzz \to xzzzzyzzzz$

E.G.

$N = \{S, A, B, C\}$

$T = \{(\ )\ [\ ]\ a\ b\}$

$$P \begin{cases} S \to A \\ S \to AaS \end{cases}$$

N = { S, A, B, C }
T = { ( ) [ ] a b }
S = S

S → AaS
A → b | B | C
B → (S)
C → [S]

every (is used is also a) introduced.

E.G.

b a (b) a [(b)]

S → AaS → baS → baAaS → baBaS → ba(S)aS → ba(A)aS →
ba(b)aS → ba(b)aA → ba(b)aC → ba(b)a[S] → ba(b)a[A] →
ba(b)[B] → ba(b)[(S)] → ba(b)[(A)] → ba(b)[(b)]

MORE ON DERIVATIONS

DEF: Leftmost Derivation: a derivation where the leftmost non-terminal is the one re-written

Rightmost Derivation: a derivation where the rightmost non-terminal is the one re-written.

E.G.

S → SAS | x | y | z
A → a | b

Prove: xaybz ∈ L

Leftmost

S
SAS
xAS
xaS
xaSAS
xayAS
xaybS
xaybz

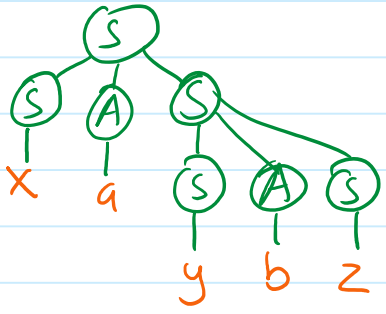
Rightmost.

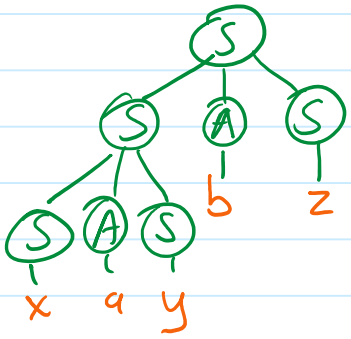
S
SAS
SAz
Sbz
SASbz
SAybz
Saybz
xaybz

DEF: Parse tree a tree representation of a derivation.

DEF: Parse tree a tree representation of a derivation.

- root: the start symbol
- leafs: terminal symbols
- intermideate nodes: non-terminal symbols
- A node's children are the symbols it is rewritten with.





- AMBIGUITY

A grammar is called ambiguous when there is a word w in the language that has
- more than one leftmost derivations

or
- more than one rightmost derivations

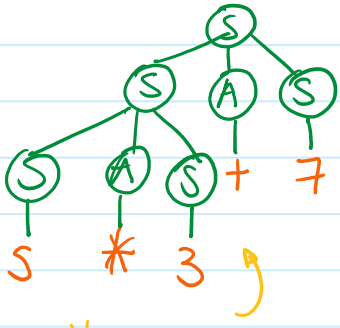
or, more than one distinct parse tree.

E.G.

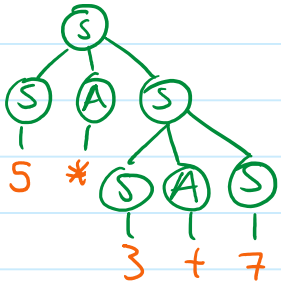
$S \to SAS \mid 5 \mid 3 \mid 7$

$A \to + \mid *$

$w = 5 * 3 + 7$



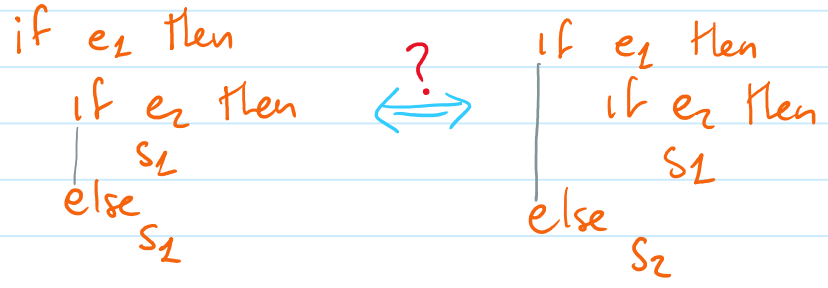
this one is prefered.

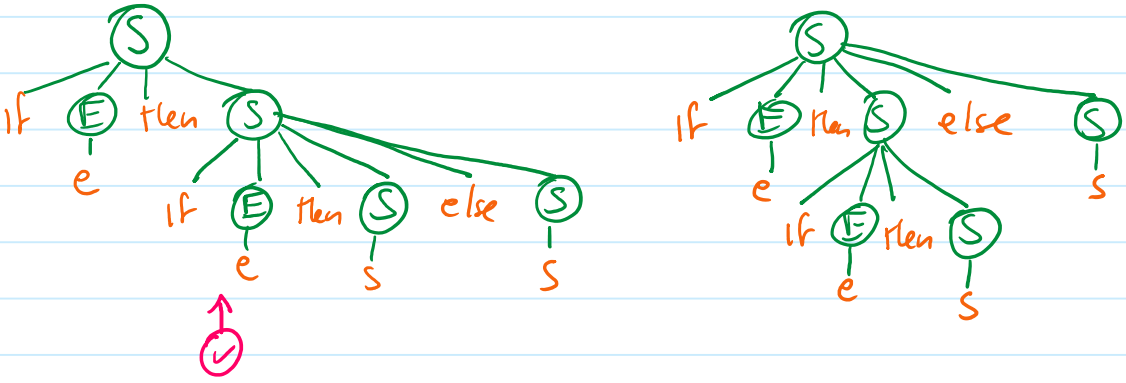


E.G. "The dangling 'else'" – Algol '60

S → I | s
I → if E then S | if E then S else S
E → e

w = if e_1 then if e_2 then s_1 else s_2





FIX: "an else statement is paired with the closest then"