October, 10th

(1)	Exp	→	Exp + Exp
(2)		→	Exp * Exp
(3)		→	(Exp)
(4)		→	Id
(5)		→	Cst

Syntax of programs

Production rules P

V = set of variables = { Exp }

T = set of terminals = alphabet = { +, *, (,), Id, Cst }

S = start symbol = Exp

$G =$

(1)	Exp	$\rightarrow$	Exp + Exp
(2)		$\rightarrow$	Exp * Exp
(3)		$\rightarrow$	(Exp)
(4)		$\rightarrow$	Id
(5)		$\rightarrow$	Cst

Semantics of grammars.

Derivation

$Exp \overset{2}{\Rightarrow} Exp * Exp \overset{1}{\Rightarrow} Exp * Exp + Exp$

$Exp \overset{4}{\Rightarrow} Exp * Id + Exp \overset{5}{\Rightarrow} Exp * Id + Cst$

$\overset{4}{\Rightarrow} Id * Id + Cst$

Exp : start symbol

$w \in T^*$, si $w \in L(G)$ ← $Id * Id + Cst$: "w"

Let's look for a grammar for $L_()$

(1) $S \to ()$

(2) $\to (S)$

(3) $\to SS$

Context free

$w = (()()) \in L_()$

$S \overset{2}{\Rightarrow} (S) \overset{3}{\Rightarrow} (SS)$

$\overset{1}{\Rightarrow} (()S)$

$\overset{1}{\Rightarrow} (()())$

(1)	S	$\to$	ε
(2)	S	$\to$	S'
(3)	S'	$\to$	$ABCS'$
(4)	S'	$\to$	ABC
(5)	AB	$\to$	BA
(6)	BA	$\to$	AB
(7)	AC	$\to$	CA
(8)	CA	$\to$	AC
(9)	BC	$\to$	CB
(10)	CB	$\to$	BC
(11)	A	$\to$	a
(12)	B	$\to$	b
(13)	C	$\to$	c

$V = \{S, S', A, B, C\}$

$T = \{a, b, c\}$

accepts all words on $\{a, b, c\}$

s.t. $\#a = \#b = \#c$.

$aacbbc$ ✓

$aaabcc$ ✗

(1)	S	$\to$	ε
(2)		$\to$	S'
(3)	S'	$\to$	$ABCS'$
(4)		$\to$	ABC
(5)	AB	$\to$	BA
(6)	BA	$\to$	AB
(7)	AC	$\to$	CA
(8)	CA	$\to$	AC
(9)	BC	$\to$	CB
(10)	CB	$\to$	BC
(11)	A	$\to$	a
(12)	B	$\to$	b
(13)	C	$\to$	c

Context-sensitive not context-free

aabbc ?

Let's look at a derivation...

$S \overset{2}{\Rightarrow} S' \overset{3}{\Rightarrow} ABCS' \overset{4}{\Rightarrow} ABCABC$

$\overset{8}{\Rightarrow} ABACBC$

$\overset{6}{\Rightarrow} AABCBC$

$\overset{9}{\Rightarrow} AACBBC$

$\overset{11}{\Rightarrow} aACBBC$

$\vdots$

$\overset{13}{\Rightarrow} aabbc$

Chomsky Hierarchy

Class 0: Unrestricted grammars
= all grammars.

Class 1: Context-sensitive grammars
for all rules $\alpha \to \beta$
1. either $\alpha = S$ and $\beta = \varepsilon$
2. or: $|\alpha| \leq |\beta|$
and S does not appear in β.

Class 2: Context-free grammars

for all rules $\alpha \to \beta$:

$$\alpha \in V$$

The left-hand side is always a variable.

Class 3: Regular grammars:

→ left-regular grammars:

- for all rules $\alpha \to \beta$: $\alpha \in V$ and $(\beta \in T^* \text{ or } \beta \in V.T^*)$

→ right-regular grammars

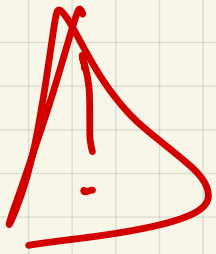
for all rules $\alpha \to \beta$: $\alpha \in V$ and $(\beta \in T^* \text{ or } \beta \in T^*.V)$

Example of class 3 grammars.

$$S \to Sa \quad \text{(left-regular)}$$
$$\to \varepsilon$$

accepts a^* which is regular.

$$S \to aS \quad \text{(right-regular)}$$
$$\to \varepsilon$$



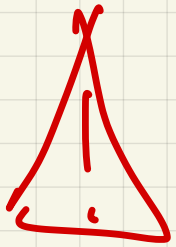
$S \to aS$ ← right-regular

$\to Sa$ ← left-regular

$\to \varepsilon$

This is not a regular grammar!

We cannot mix left-regular and right-regular rules.



You can have context-free grammars that are not Context-sensitive.

Example: $S \to Sa$ ← 1

$\to \varepsilon$

CFree ✓

Regular ✓

not CSensitive

CFL

<u>Definition</u> A context-free language is a language s.t. there exists a context-free grammar that accepts it.

CSL

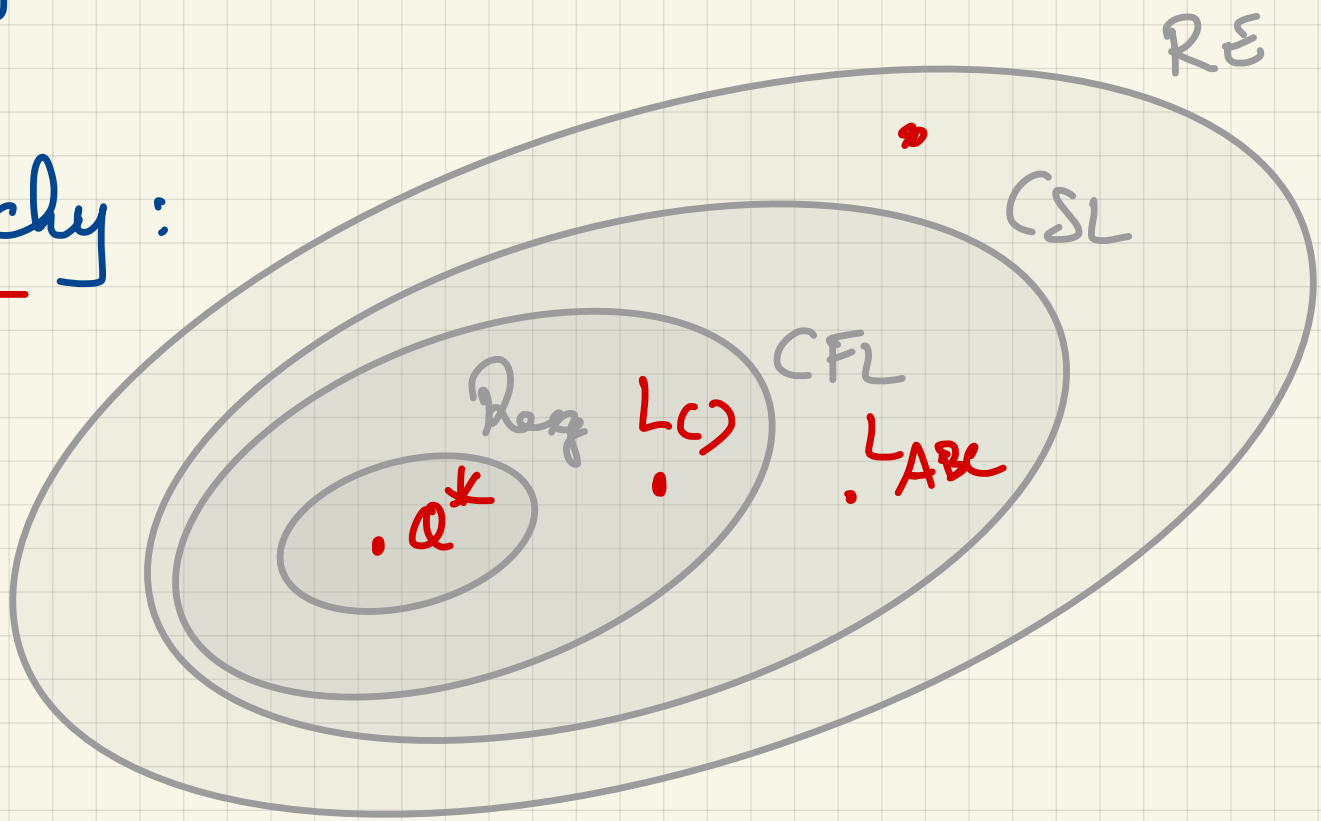
<u>Definition</u> A context-sensitive language
____

a context-sensitive grammar that accepts it.

Definition : A recursively Enumerable language is a language that is accepted by a grammar.

RE

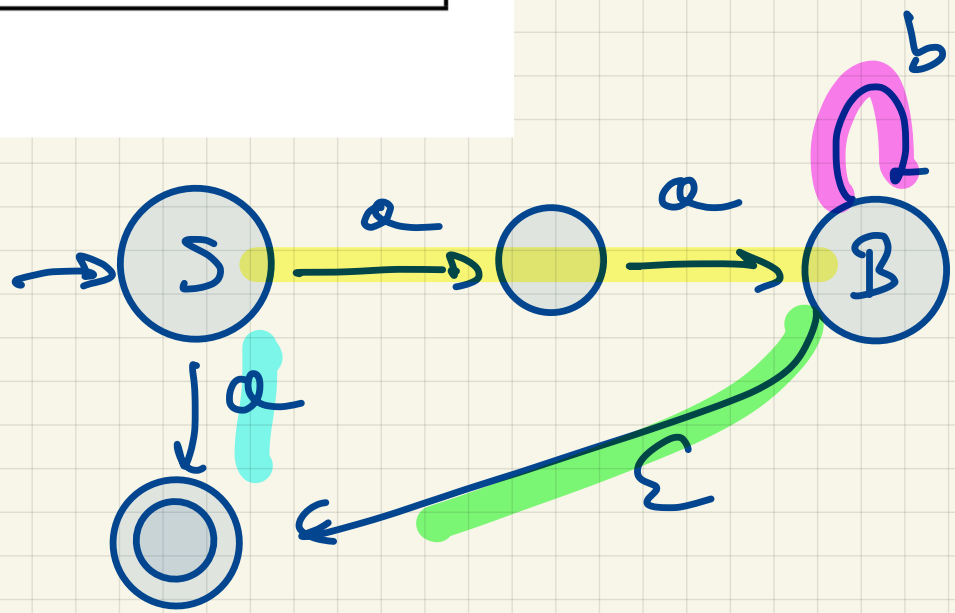
Chomsky Hierarchy :

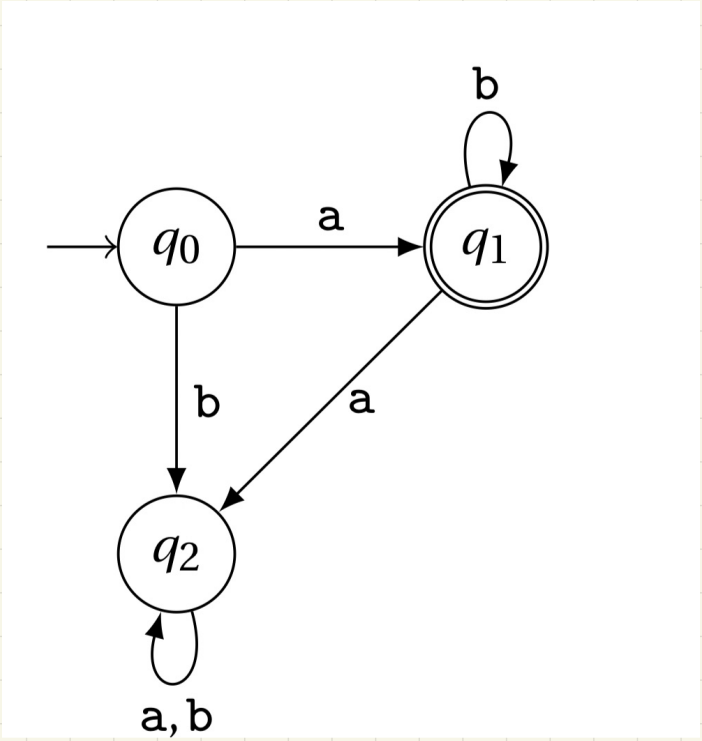


Let's show that regular grammars correspond to regular languages.

(1)	S	$\rightarrow$	$\mathtt{a}\mathtt{a}B$
(2)		$\rightarrow$	$\mathtt{a}$
(3)	B	$\rightarrow$	$\mathtt{b}B$
(4)		$\rightarrow$	ε

Reg grammar
$\longrightarrow$ finite automaton





Reg langs.
(finite automaton)
↓
Reg grammar

$S_0 \to a\, S_1$
$\to b\, S_2$

$S_1 \to a\, S_2$
$\to b\, S_1$
$S_1 \to \varepsilon$

$S_2 \to a\, S_2$
$\to b\, S_2$

We have shown that regular languages correspond to regular grammars.

this implies

$$Reg \subseteq CFL$$

Because: ① By definition, all reg. grammars are context-free

② So any language accepted by a Reg. grammar is a CFL

Last difficulty: $CFL \subseteq CSL$

not all CFG are CSG.

But we can convert all CFG into an equivalent CSG.

Context-free grammars & languages

$$L_{pal\#} = \{ w \cdot \# \cdot w^R \mid w \in \{0,1\}^* \}$$

w^R = mirror image of w

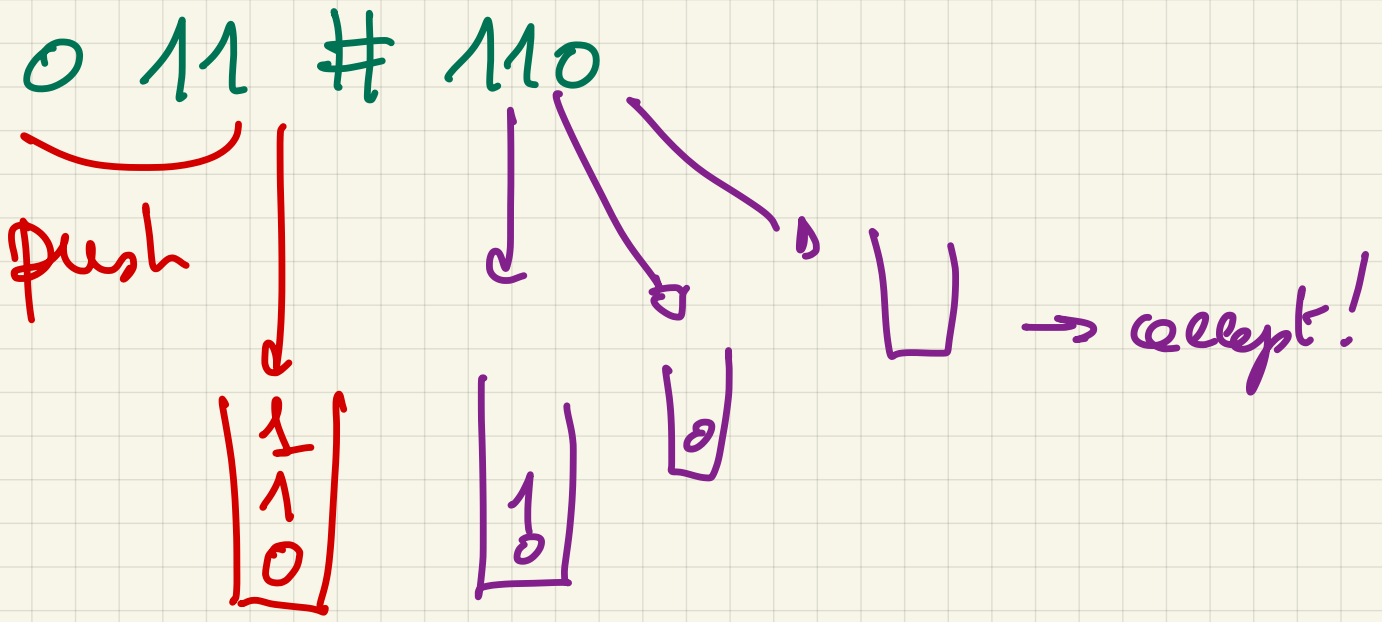
$w = 1011$ $\quad$ $w^R = 1101$

$1011 \# 1101 \in L_{pal\#}$

$101 \# 01 \notin L_{pal\#}$

How to recognize this language!

Idea: use a stack to store the prefix of the word and later check that the suffix matches.



$S \to 0S0$

$\to 1S1$

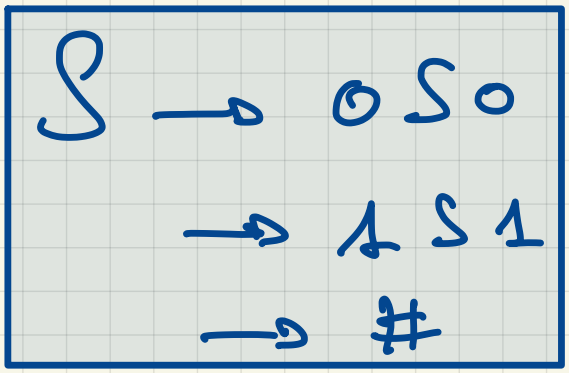
$\to \#$

$\to$ accepts $L_{pal\#}$

$S \Rightarrow 0S0 \Rightarrow 01S10 \Rightarrow 011S110$

$\Rightarrow 011\#110$

We could interpret this as a recursive function that accepts the words.




```
1  def S():
2      n = read_next_character()
3
4      if n == '#':
5          return True
6
7      if n == '0':
8          r = S()
9          if (!r): return False
10         n = read_next_character()
11         if (n == '0'): return True
12         else: return False
13
14     if n == '1':
15         r = S()
16         if (!r): return False
17         n = read_next_character()
18         if (n == '1'): return True
19         else: return False
```

(1)	Exp	→	Exp + Exp
(2)		→	Exp ∗ Exp
(3)		→	(Exp)
(4)		→	Id
(5)		→	Cst

Derivations

$$\underline{\text{Exp}} \overset{2}{\Rightarrow} \underline{\text{Exp}} * \text{Exp} \overset{1}{\Rightarrow} \text{Exp} + \underline{\text{Exp}} * \text{Exp} \overset{4}{\Rightarrow} \text{Exp} + \text{Id} * \underline{\text{Exp}} \overset{4}{\Rightarrow} \underline{\text{Exp}} + \text{Id} * \text{Id} \overset{4}{\Rightarrow} \text{Id} + \text{Id} * \text{Id}$$

Right

Left most derivation: when there are several variable, we derive the left most first

Right

$$\underline{\text{Exp}} \overset{2}{\Rightarrow} \text{Exp} * \underline{\text{Exp}} \overset{4}{\Rightarrow} \underline{\text{Exp}} * \text{Id} \overset{1}{\Rightarrow} \text{Exp} + \underline{\text{Exp}} * \text{Id} \overset{4}{\Rightarrow} \underline{\text{Exp}} + \text{Id} * \text{Id} \overset{4}{\Rightarrow} \text{Id} + \text{Id} * \text{Id}$$

right

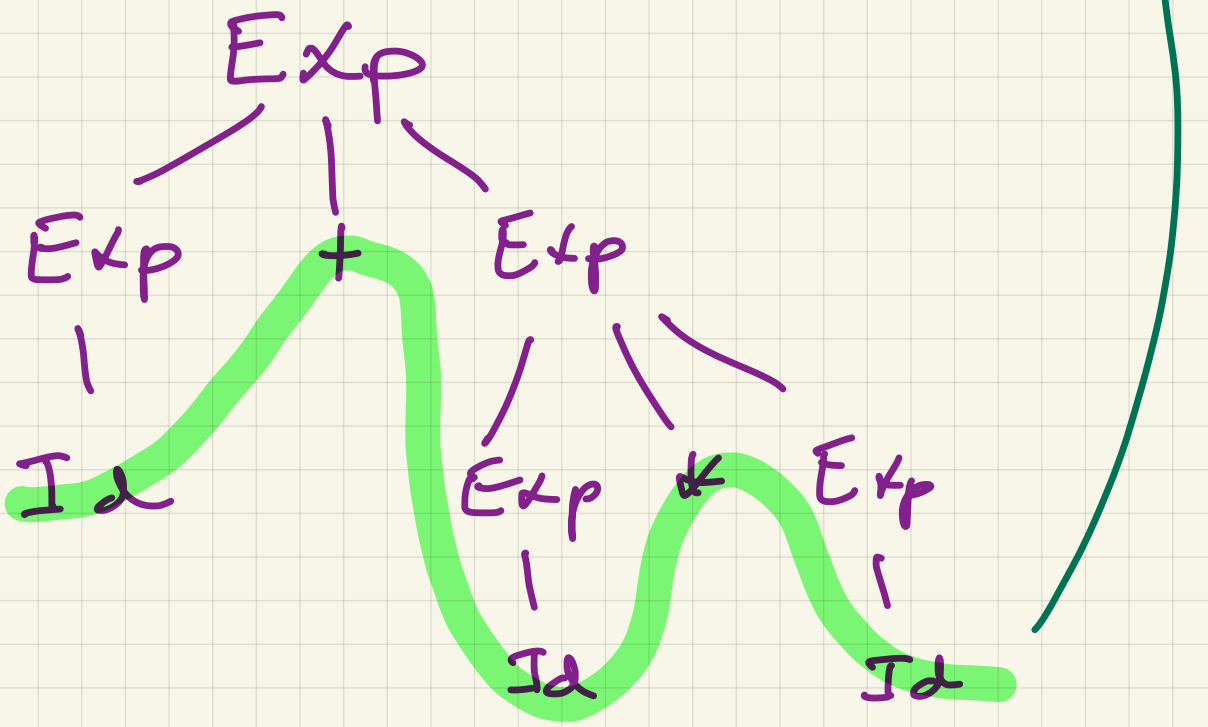
$$\underline{\text{Exp}} \overset{2}{\Rightarrow} \underline{\text{Exp}} * \text{Exp} \overset{1}{\Rightarrow} \underline{\text{Exp}} + \text{Exp} * \text{Exp} \overset{4}{\Rightarrow} \text{Id} + \underline{\text{Exp}} * \text{Exp} \overset{4}{\Rightarrow} \text{Id} + \text{Id} * \underline{\text{Exp}} \overset{4}{\Rightarrow} \text{Id} + \text{Id} * \text{Id}$$

left

Derivation tree

(1)	Exp	→	Exp + Exp
(2)		→	Exp * Exp
(3)		→	(Exp)
(4)		→	Id
(5)		→	Cst

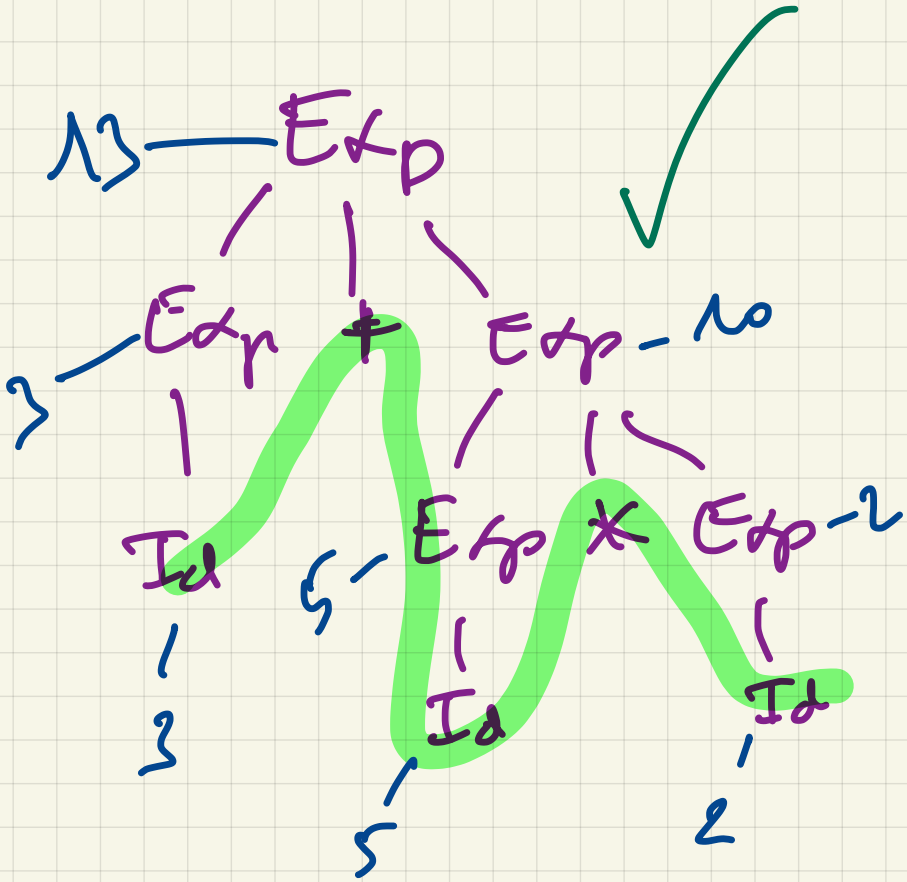
Id + Id * Id.

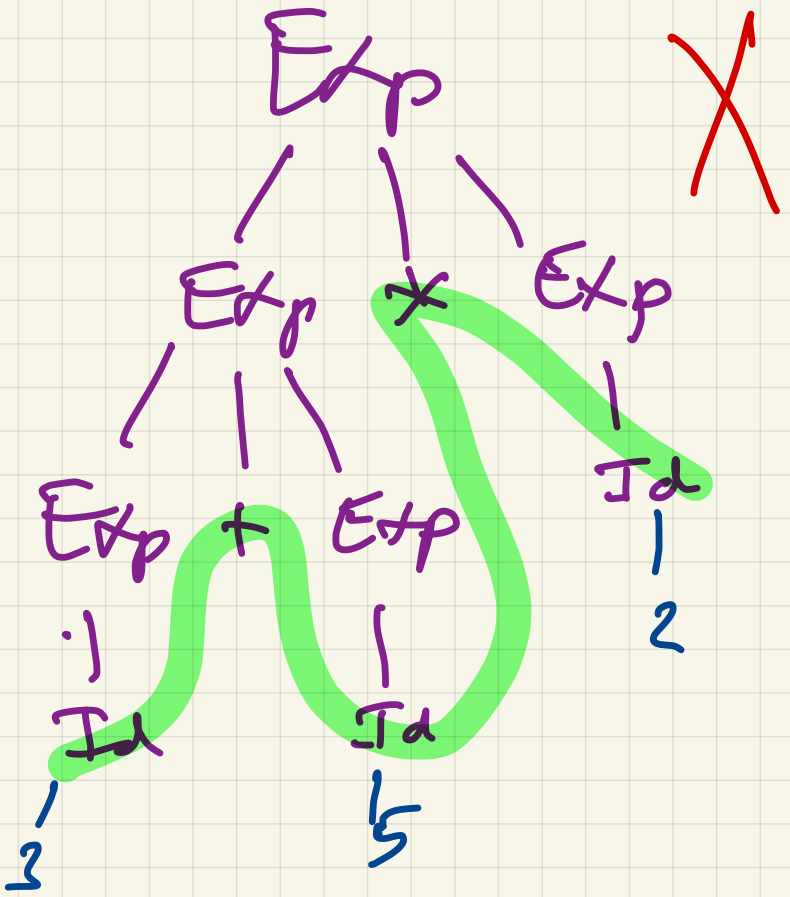


Interests of the Derivation Tree

We can find the priority of the operators

Id + Id * Id





⚠ Some grammars are ambiguous.
They admit ≠ derivation trees
on the same word.

(1)	Exp	→	Exp + Exp
(2)		→	Exp ∗ Exp
(3)		→	(Exp)
(4)		→	Id
(5)		→	Cst

→ ambiguous.

Figure 4.4: The grammar G_{Exp} to ge

Membership problem

Given a CFG G, a word w

$w \stackrel{?}{\in} L(G)$

We will assume that the CFG is in Chomsky Normal Form (CNF)

All the rules are of the form

$A \to BC$ $\quad A, B, C \in V$

$A \to a$ $\quad a \in T$

$S \to \varepsilon$

(1)	S	$\rightarrow$	XB
(2)		$\rightarrow$	$X'B$
(3)	X	$\rightarrow$	XA
(4)		$\rightarrow$	a
(5)	X'	$\rightarrow$	AY
(6)	Y	$\rightarrow$	AA
(7)	A	$\rightarrow$	a
(8)	B	$\rightarrow$	b

accepts $a^+b = aa^*b$

$w = aaab$

length = 4

$w = w_1 w_2 \dots w_n$

Cell (i,j) of the table corresponds to the substring $w_i \dots w_j$

It contains all the variables that can generate $w_i \dots w_j$

1	2	3	4	
A, X	X, Y	X, X'	S	1
|||	A, X	X, Y	S	2
|||	|||	A, X	S	3
|||	|||	|||	B	4

Pushdown automata

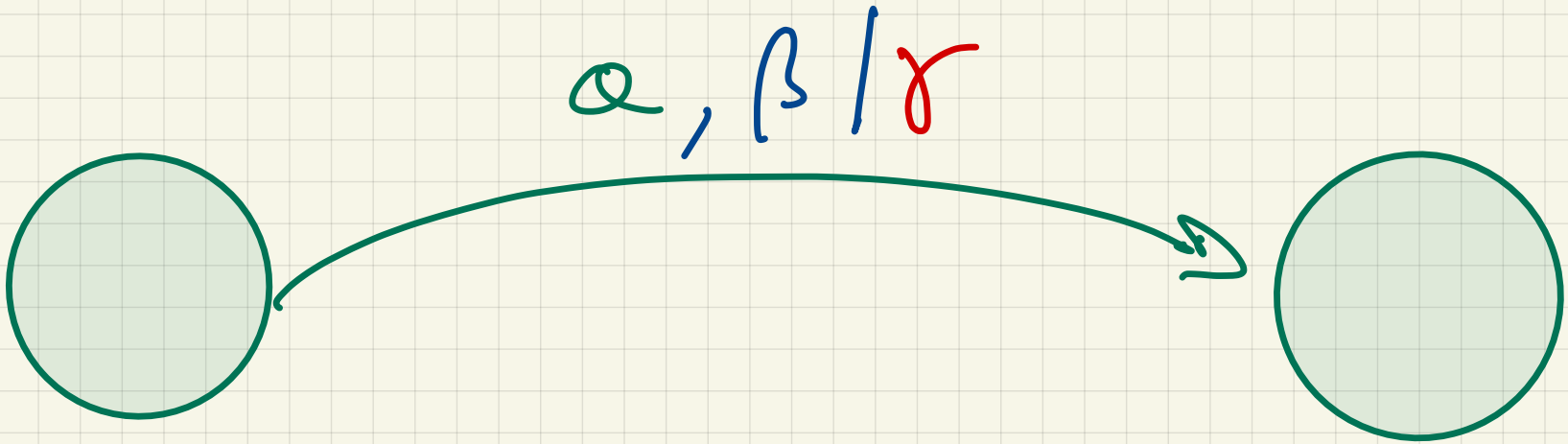
PDA

stack.

Idea: finite automata + stack

each transition can read and write from the stack.

Transitions in PDA



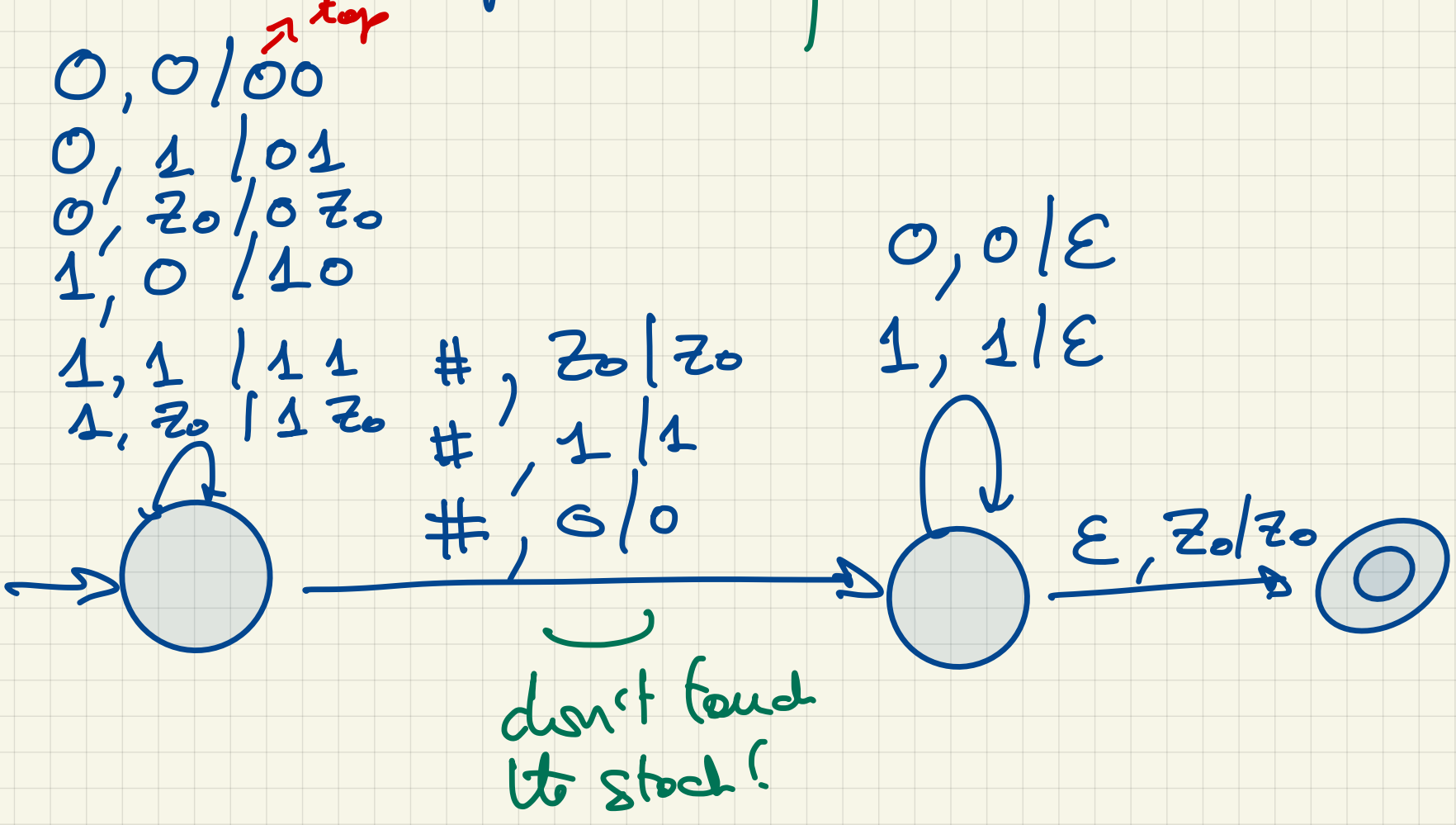
α = letter that is read on the input

β = letter on the top of the stack

γ = replacement of β after the transition.

We assume that there is a special value on the top of the stack initially Z_0

a PDA for $L_{pal\#}$?



Execution of a PDA

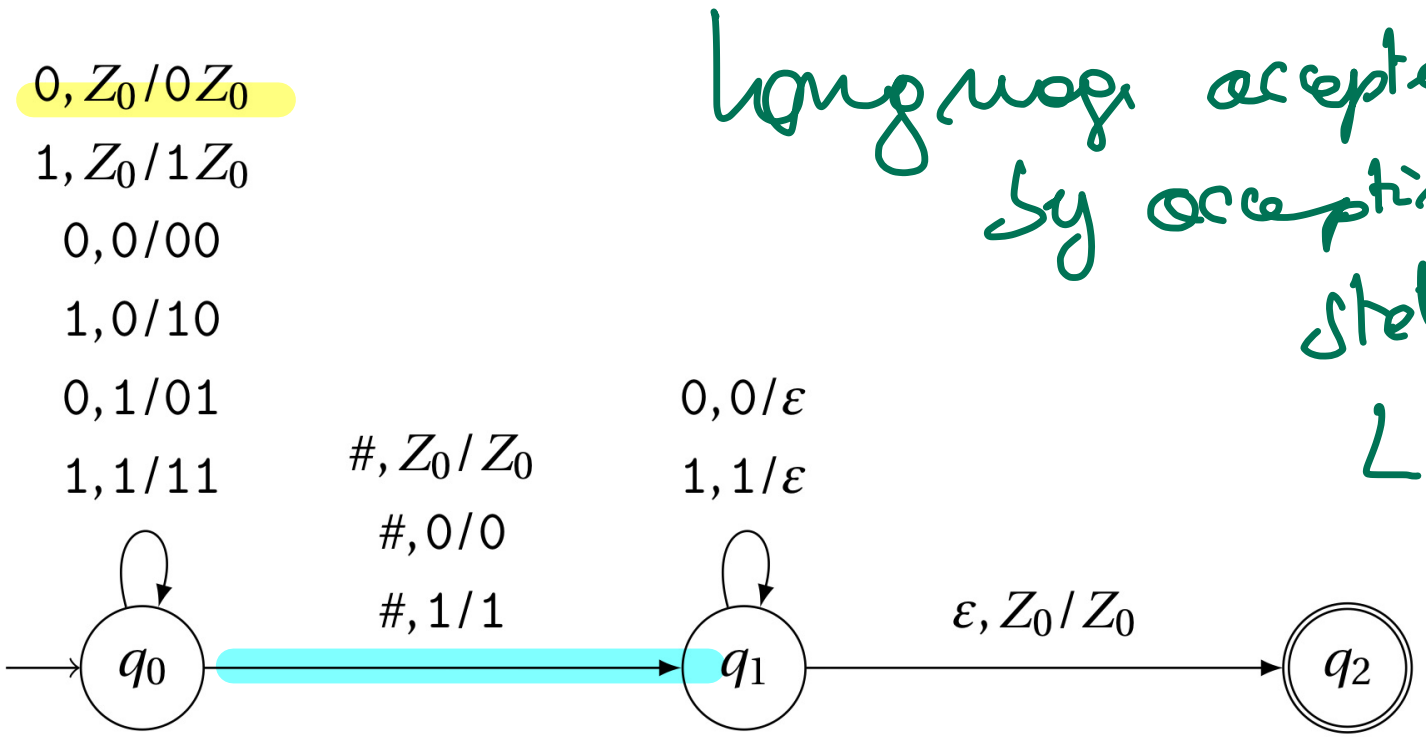
= Sequence of configurations

configuration = (q, w, γ)

- q: current state
- w: remaining input
- γ: stack content

Language accepted by accepting state: $L(P)$

$P =$



$(q_0, 01\#10, Z_0) \vdash (q_0, 1\#10, 0Z_0) \vdash (q_0, \#10, 10Z_0) \vdash (q_1, 10, 10Z_0) \vdash (q_1, 0, 0Z_0) \vdash (q_1, \varepsilon, Z_0) \vdash (q_2, \varepsilon, Z_0).$

we can write that:

word to read

top

$\in F$

initial state

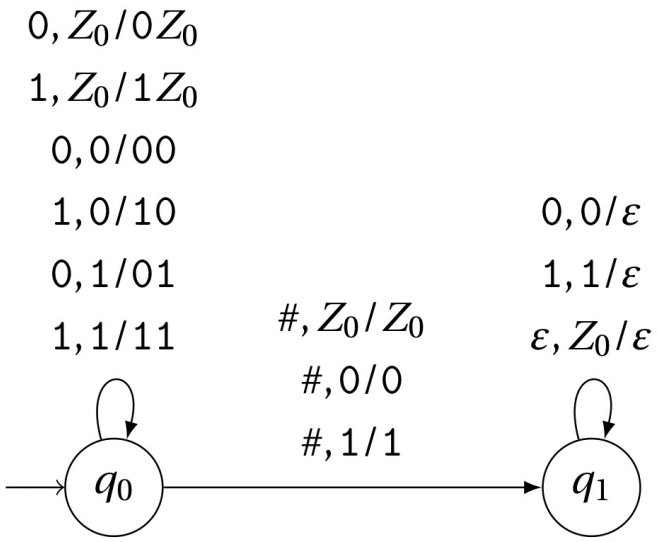
"empty"

empty!

alternative definition of the accepted language: $N(P)$

= we accept when the stack is empty (ϵ).

$P =$



$N(P) = L_{pal\#}$

$L(P) = \emptyset$