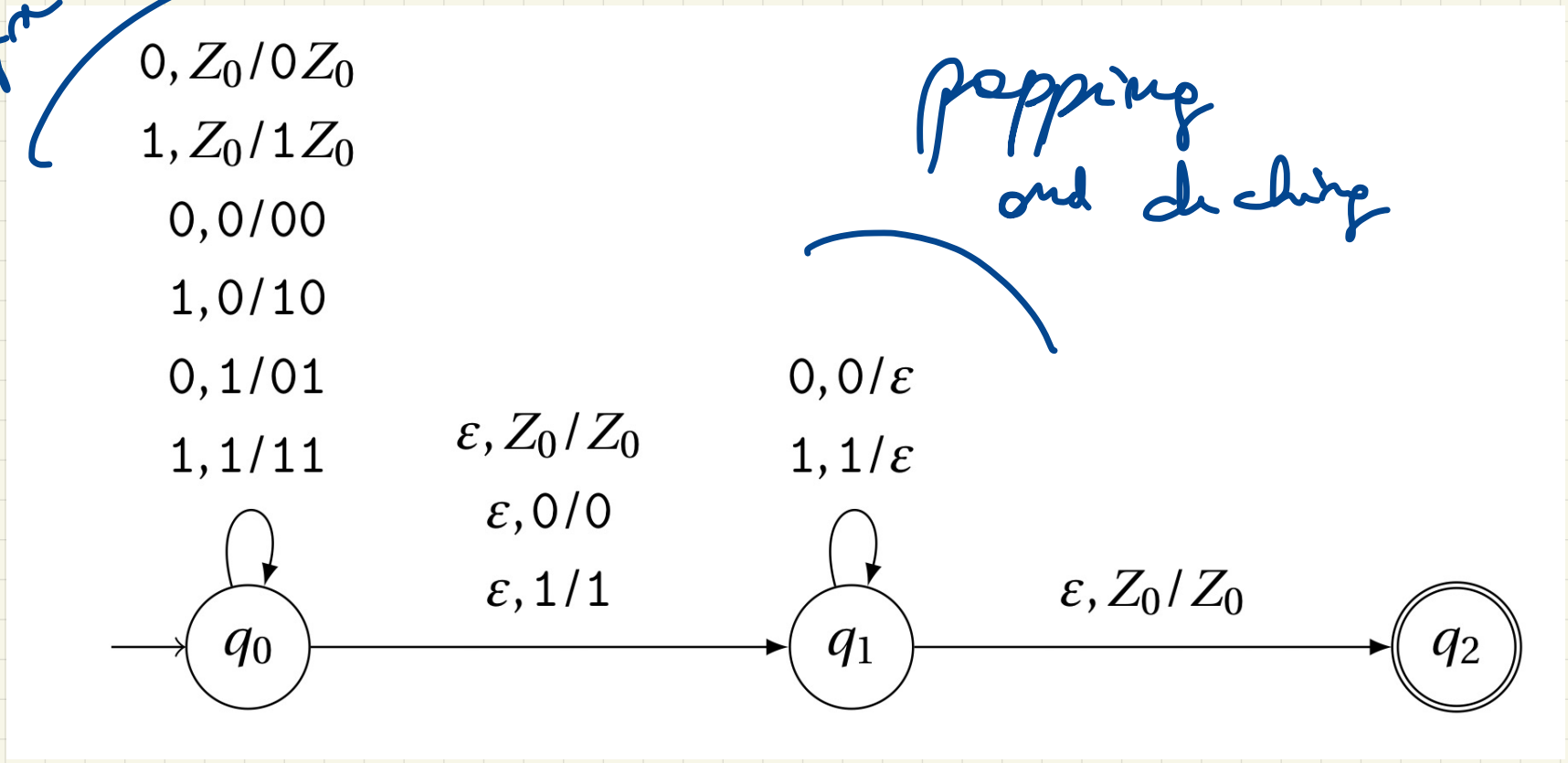
October, 17th

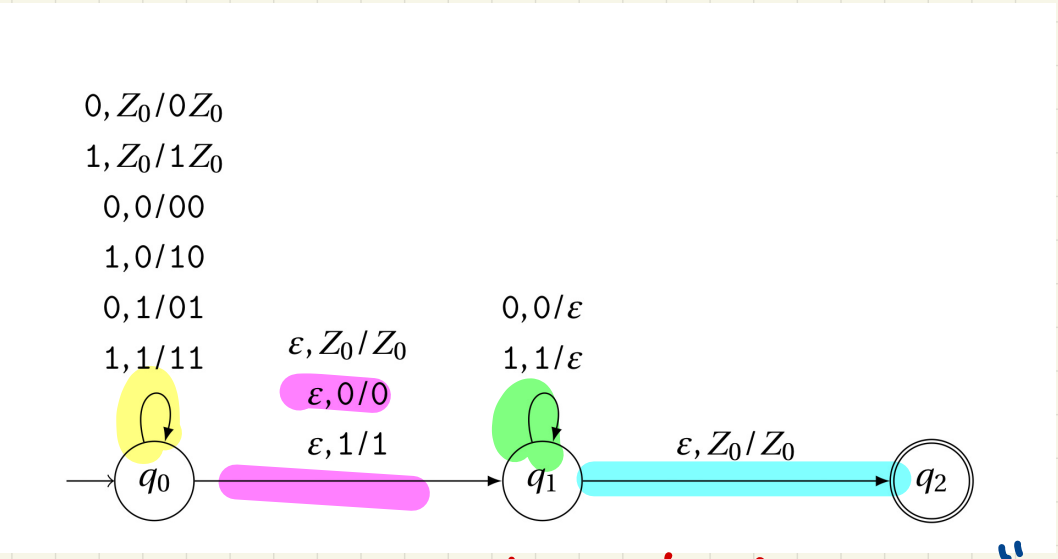
Deterministic PDAs

Let's look at another language:

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

pushing the prefix





0 1 1 0

input stack "push"

$\langle q_0, 0110, Z_0 \rangle \vdash \langle q_0, 110, 0Z_0 \rangle$

"push"

$\vdash \langle q_0, 10, 10Z_0 \rangle$

$\vdash \langle q_1, 10, 10Z_0 \rangle$

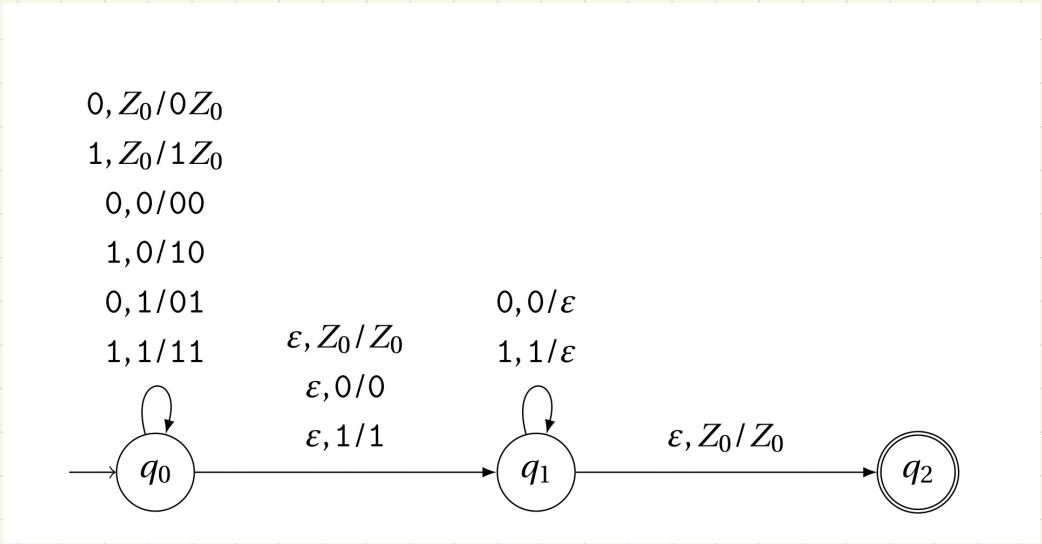
$\vdash \langle q_1, 0, 0Z_0 \rangle$

$\vdash$ accept

$\vdash \langle q_1, 110, 0Z_0 \rangle$

$\hookrightarrow \vdash$ X (no Z0 on top)

$\hookrightarrow \vdash$ X (no 1 on top)



$\langle q_0, 110, Z_0 \rangle$

$\vdash \langle q_1, 110, Z_0 \rangle$

$\vdash \langle q_2, 110, Z_0 \rangle$

not accepting !!

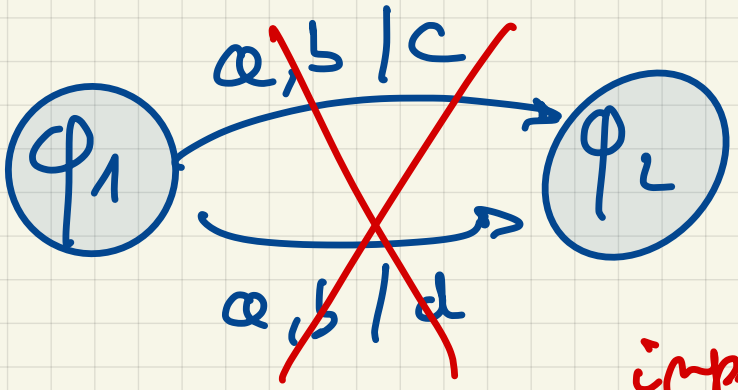
$\neq \varepsilon$

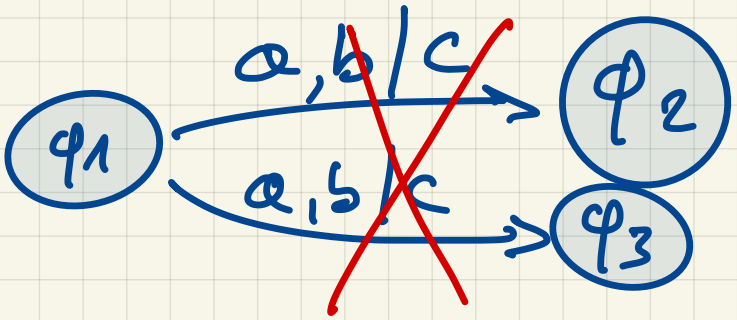
This language needs non-determinism to be accepted!

In general PDAs are more expressive than DPDAs

↓

deterministic





input

$\delta(q, a, \gamma) =$ 1 single element

state stack

Definition 4.17 (Deterministic Pushdown automaton)**.** A *Deterministic Pushdown automaton* (DPDA) for short is a PDA $\langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$ s.t.:

1. for all $q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$ and $\gamma \in \Gamma$: $\delta(q, a, \gamma)$ has at most one element; and

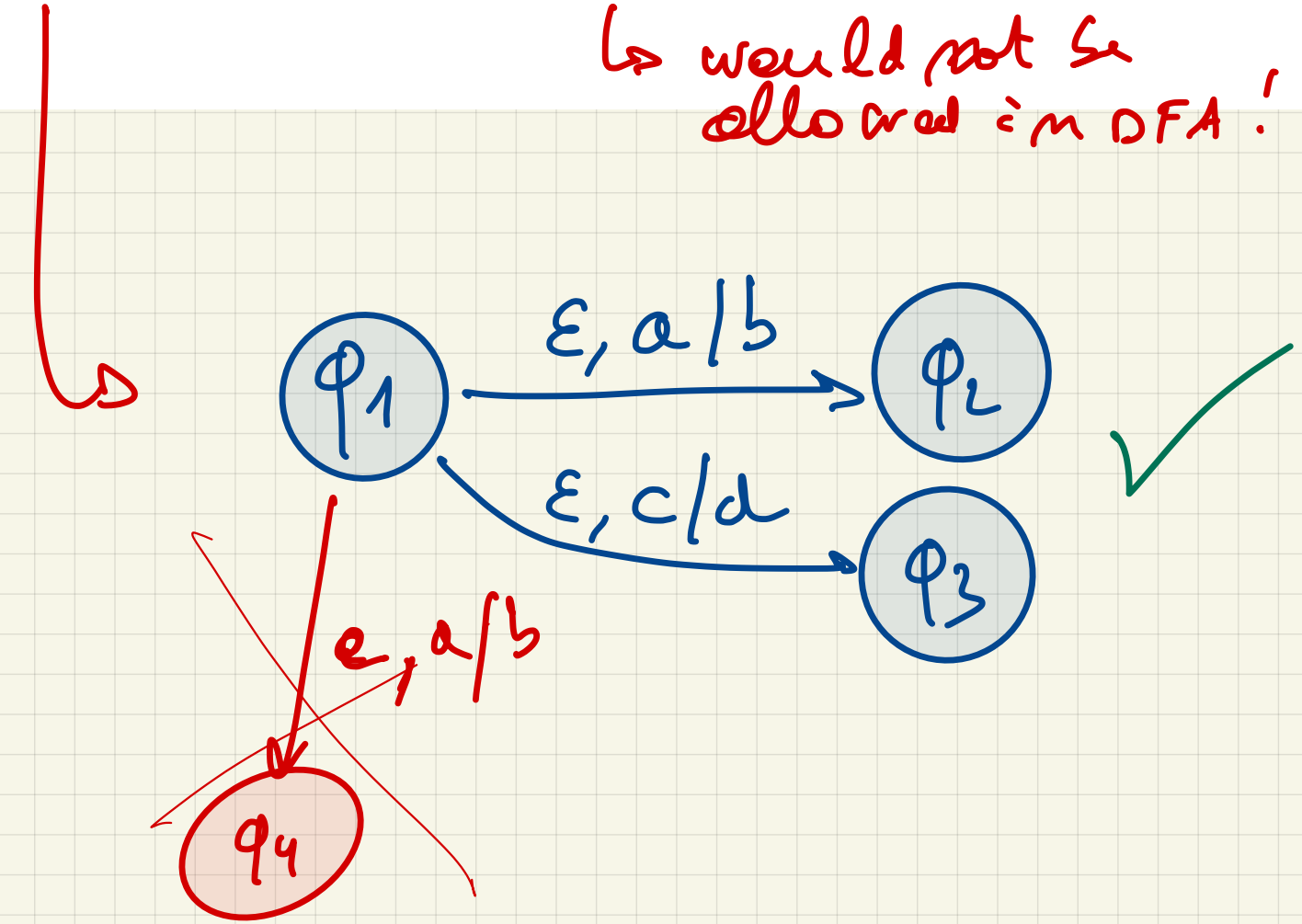
2. for all $q \in Q$ and $\gamma \in \Gamma$: if $\delta(q, \varepsilon, \gamma) \neq \emptyset$, then $\delta(q, a, \gamma) = \emptyset$ for all $a \in \Sigma$.

↳ would not be allowed in DFA!

Definition 4.17 (Deterministic Pushdown automaton)**.** A *Deterministic Pushdown automaton* (DPDA) for short is a PDA $\langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$ s.t.:

1. for all $q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$ and $\gamma \in \Gamma$: $\delta(q, a, \gamma)$ has at most one element; and

2. for all $q \in Q$ and $\gamma \in \Gamma$: if $\delta(q, \varepsilon, \gamma) \neq \emptyset$, then $\delta(q, a, \gamma) = \emptyset$ for all $a \in \Sigma$.



Equivalence between CFG and PDAs

CFG → PDA

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

Sentential form.

left most!

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

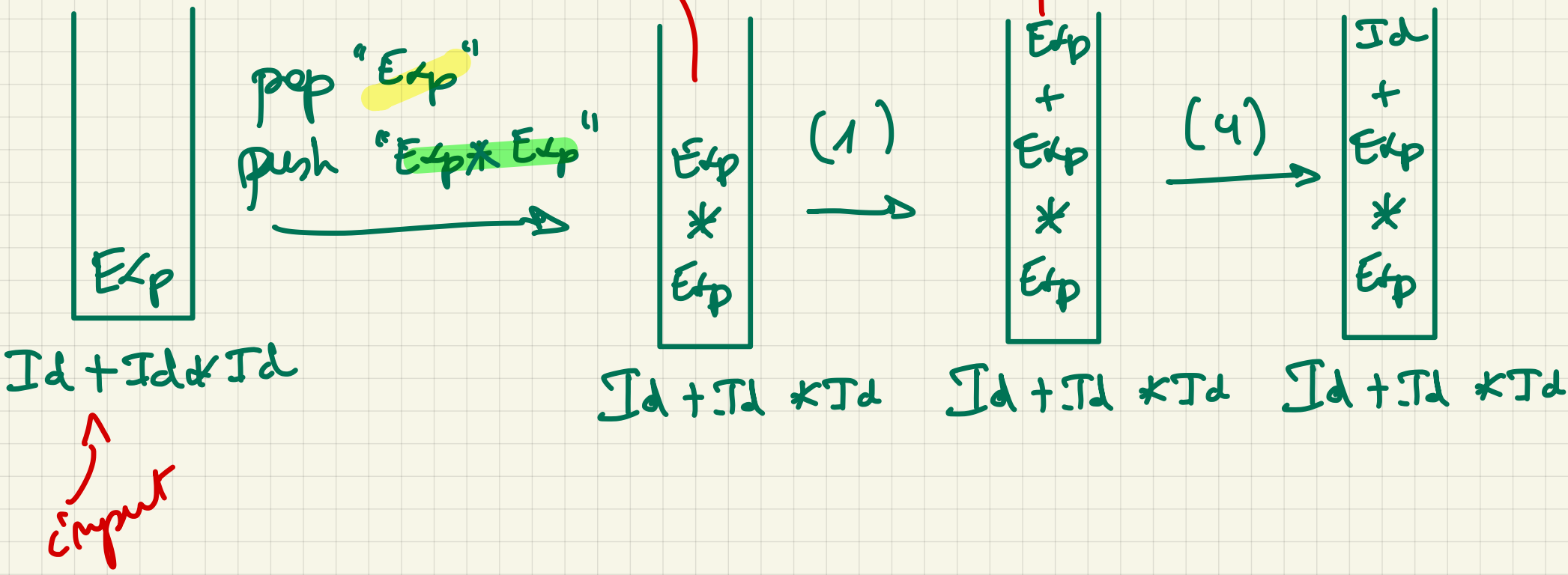
We will store the sentential forms on the Stack, with the leftmost part on top.

CFG → PDA

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

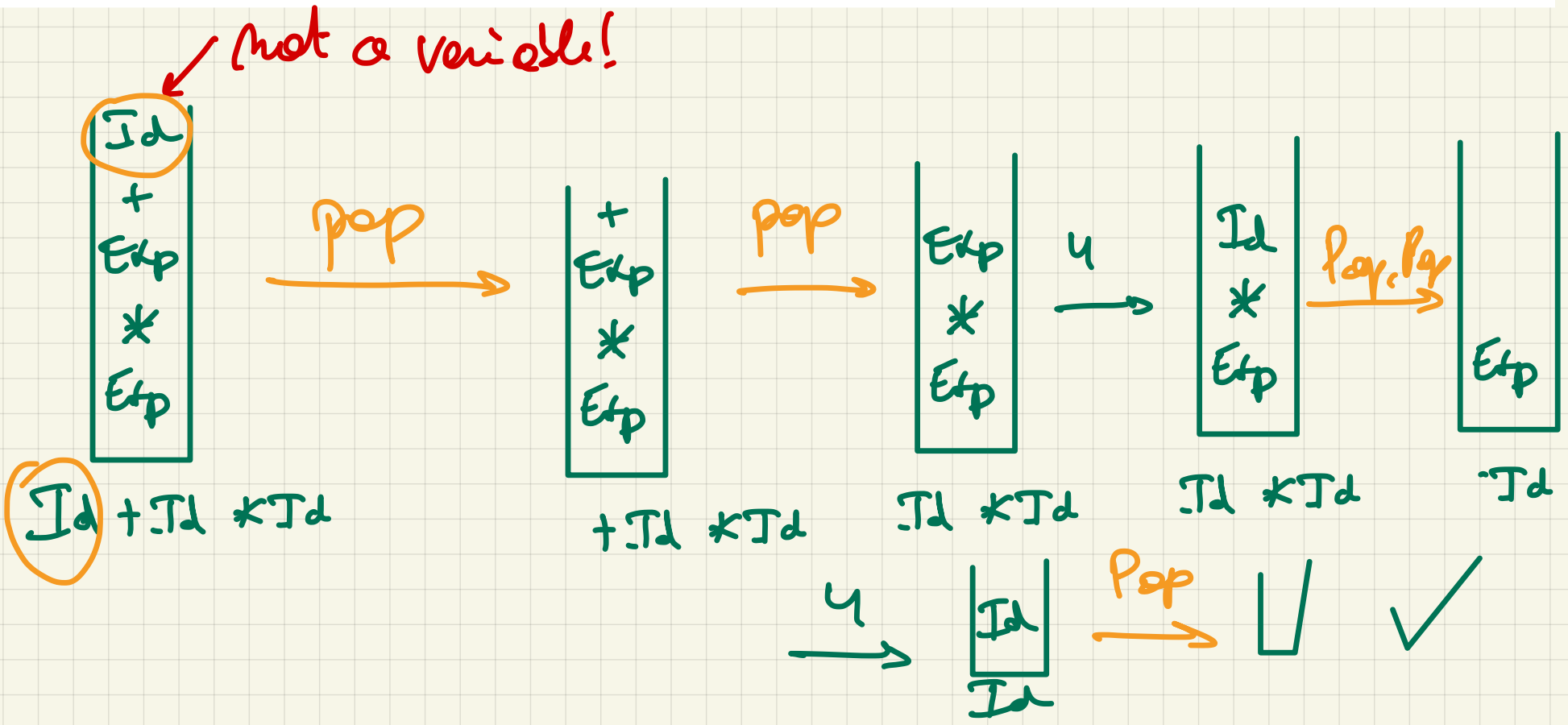
left most !

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



left most !

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

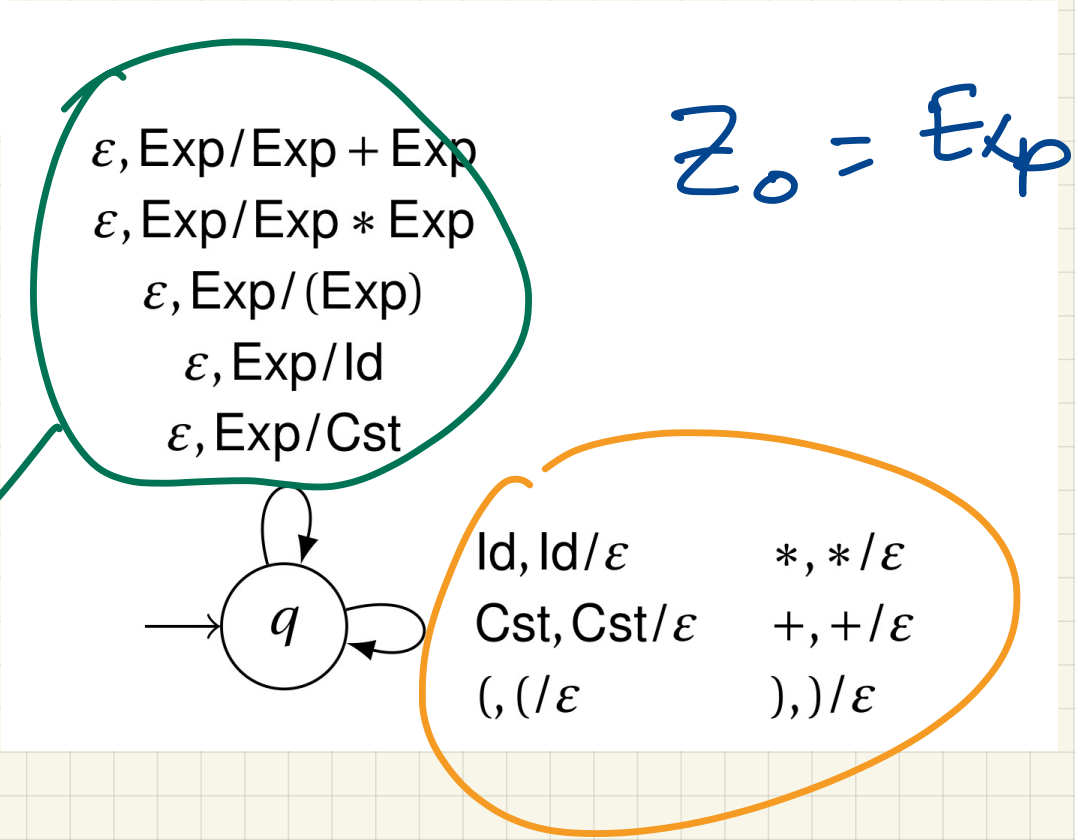


(1)	Exp	→	Exp + Exp
(2)	Exp	→	Exp * Exp
(3)		→	(Exp)
(4)		→	Id
(5)		→	Cst

G =

P =

empty stack!



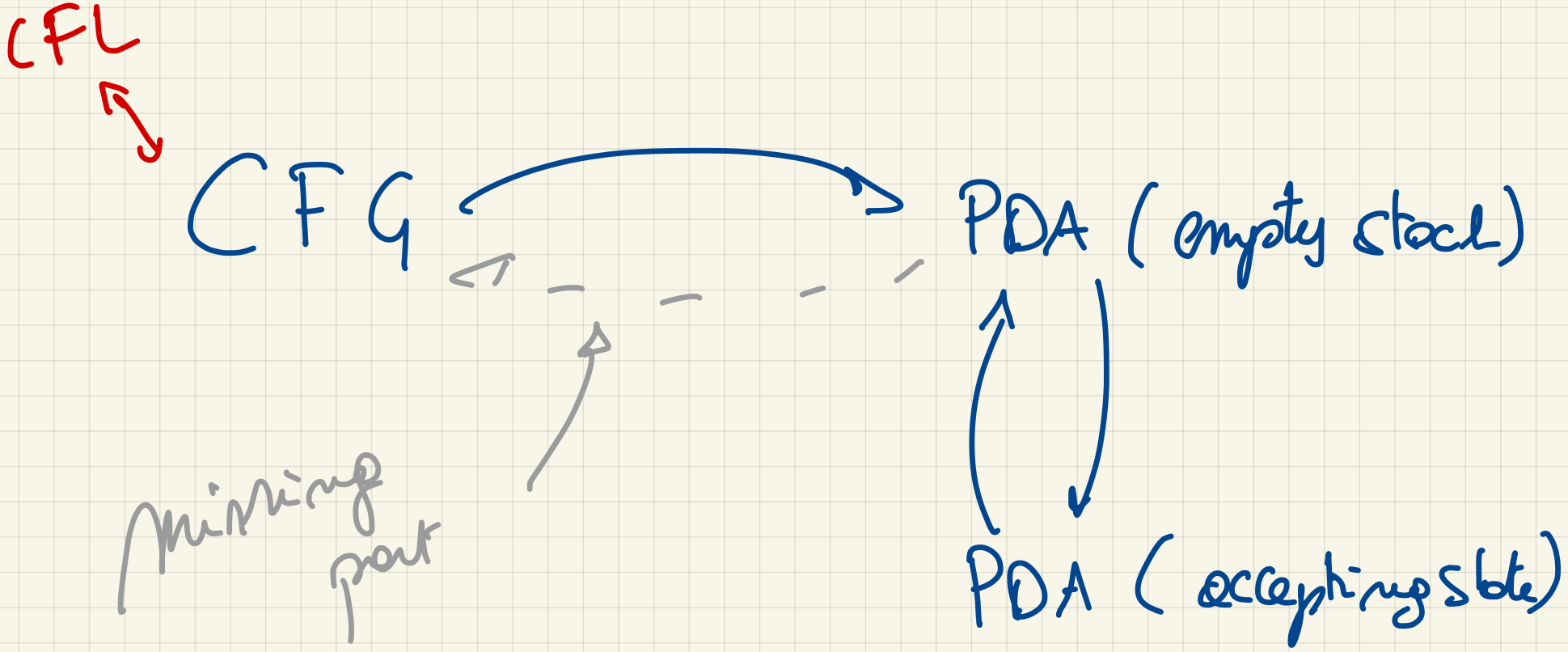
$Z_0 = Exp$

Produce

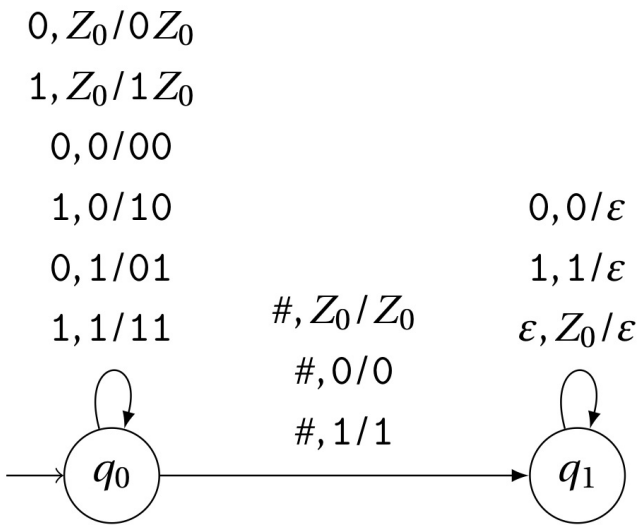
Match

$N(P) = L(G)$

Let's sum up



From PDA to CFG (empty stack)



→ accepts $L_{pal\#}$ by empty stack.

We will build a CFG whose variables are of the form

$[p\,\gamma\,q]$

p, q: states; γ: stack content (one letter)

We want:

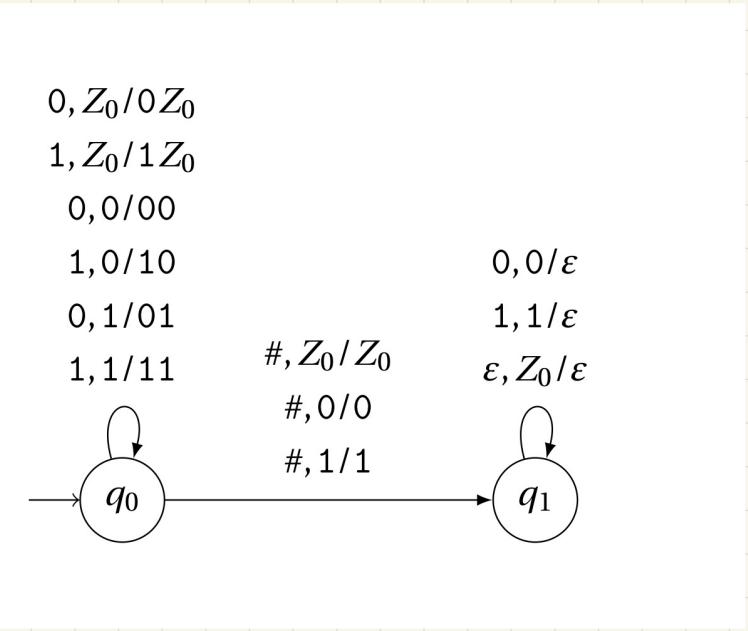
$$[p\,\gamma\,q] \overset{*}{\Rightarrow} w$$

iff w is accepted by the PDA $\mathcal{P}$:

- it starts in state p with γ on the stack
- it ends in state q with an empty stack.

$[p\,\gamma\,q] \Rightarrow \dots \Rightarrow w$ iff

$[p\,r\,q] \Rightarrow \dots \Rightarrow w$ iff

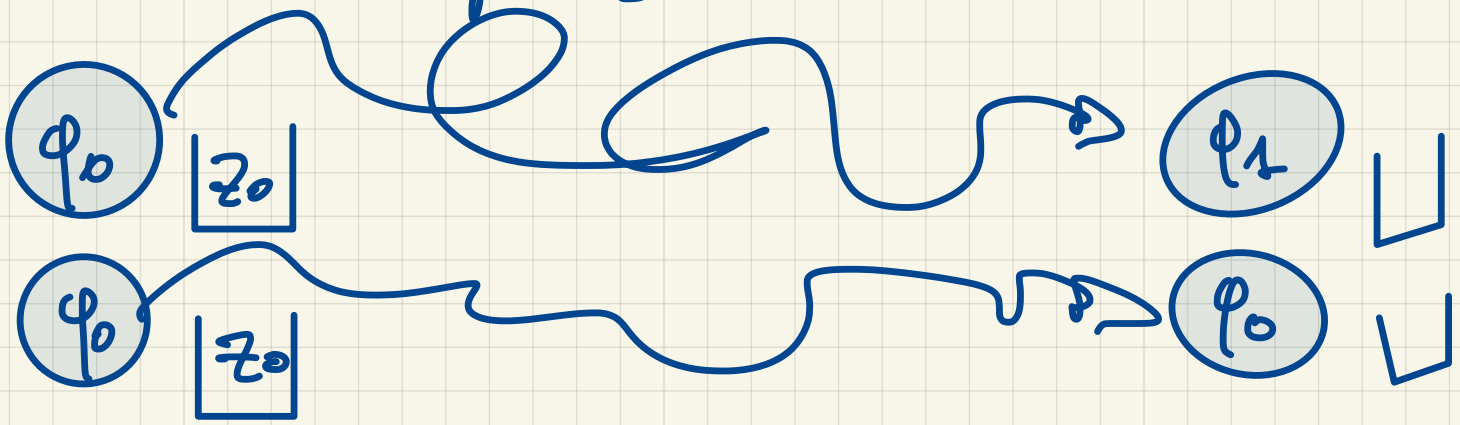


Let's try to build the grammar...

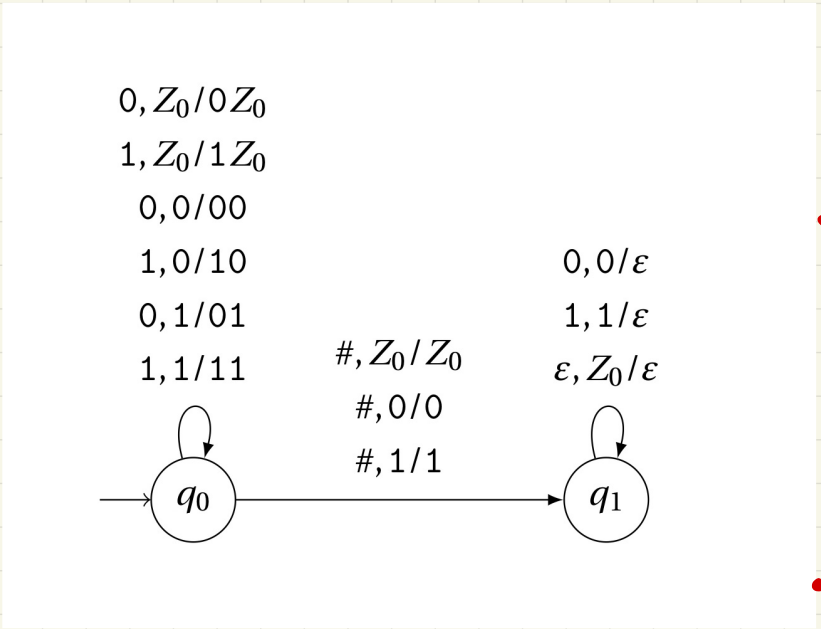
$S \rightarrow [q_0\, Z_0\, q_0]$

$\rightarrow [q_0\, Z_0\, q_1]$

What do accepting runs look like?



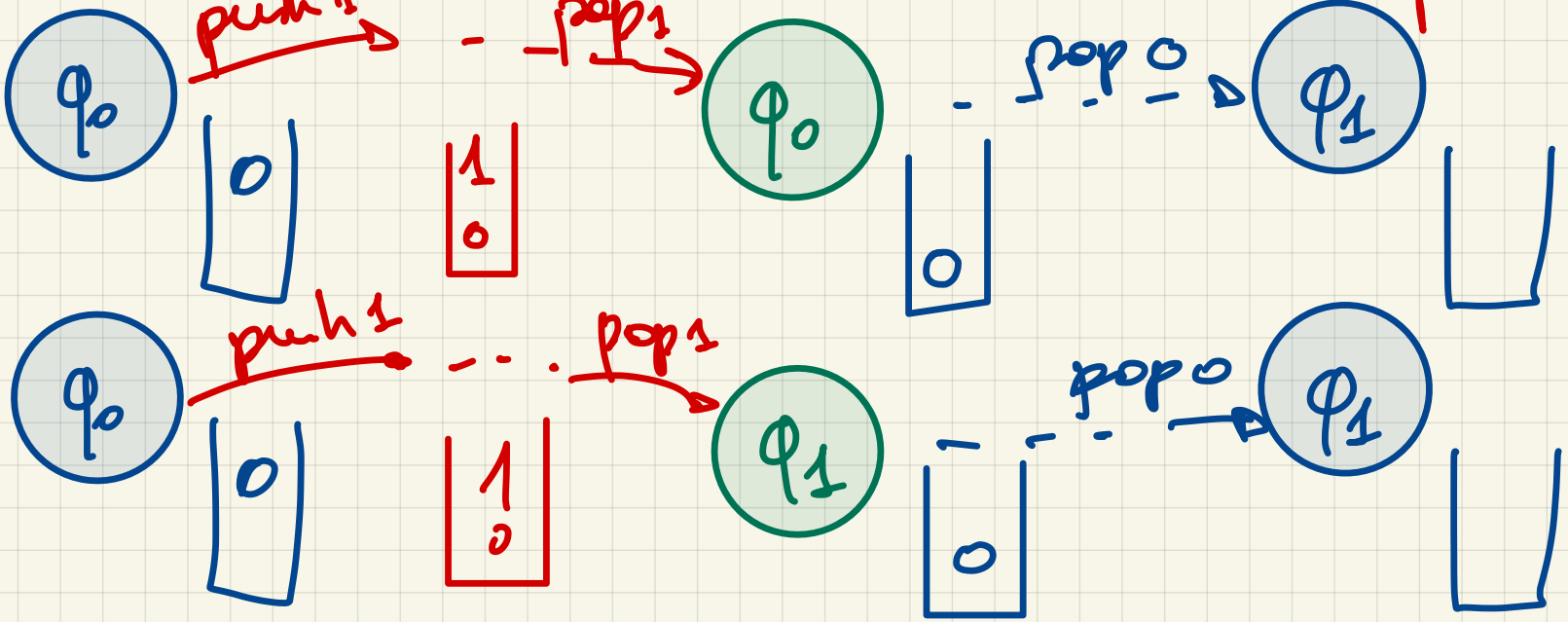
$[p\,r\,q] \Rightarrow \dots \Rightarrow w$ iff



$[q_0\,0\,q_1] \rightarrow 1\,[q_0\,1\,q_0]\,[q_0\,0\,q_1]$

$\rightarrow 1\,[q_0\,1\,q_1]\,[q_1\,0\,q_1]$

let's assume we read a 1
we will push it



Derivation on 01#10

$$S \overset{2}{\Rightarrow} [q_0 Z_0 q_1]$$

$$\overset{20}{\Rightarrow} 0[q_0 0 q_1][q_1 Z_0 q_1]$$

$$\overset{12}{\Rightarrow} 01[q_0 1 q_1][q_1 0 q_1][q_1 Z_0 q_1]$$

$$\overset{18}{\Rightarrow} 01\#[q_1 1 q_1][q_1 0 q_1][q_1 Z_0 q_1]$$

$$\overset{25}{\Rightarrow} 01\#1[q_1 0 q_1][q_1 Z_0 q_1]$$

$$\overset{24}{\Rightarrow} 01\#10[q_1 Z_0 q_1]$$

$$\overset{26}{\Rightarrow} 01\#10.$$

= stack content

= current state.

Closure properties of CFL

If L_1 and L_2 are CFL

$L_1 \cup L_2$	is a CFL
$L_1 . L_2$	is a CFL
L_1^*	is a CFL

let G_1 is a CFG for L_1

G_1

P_1

$S_1 \to A$

$A \to a$

G_2

P_2

$S_2 \to B$

$B \to Bb$

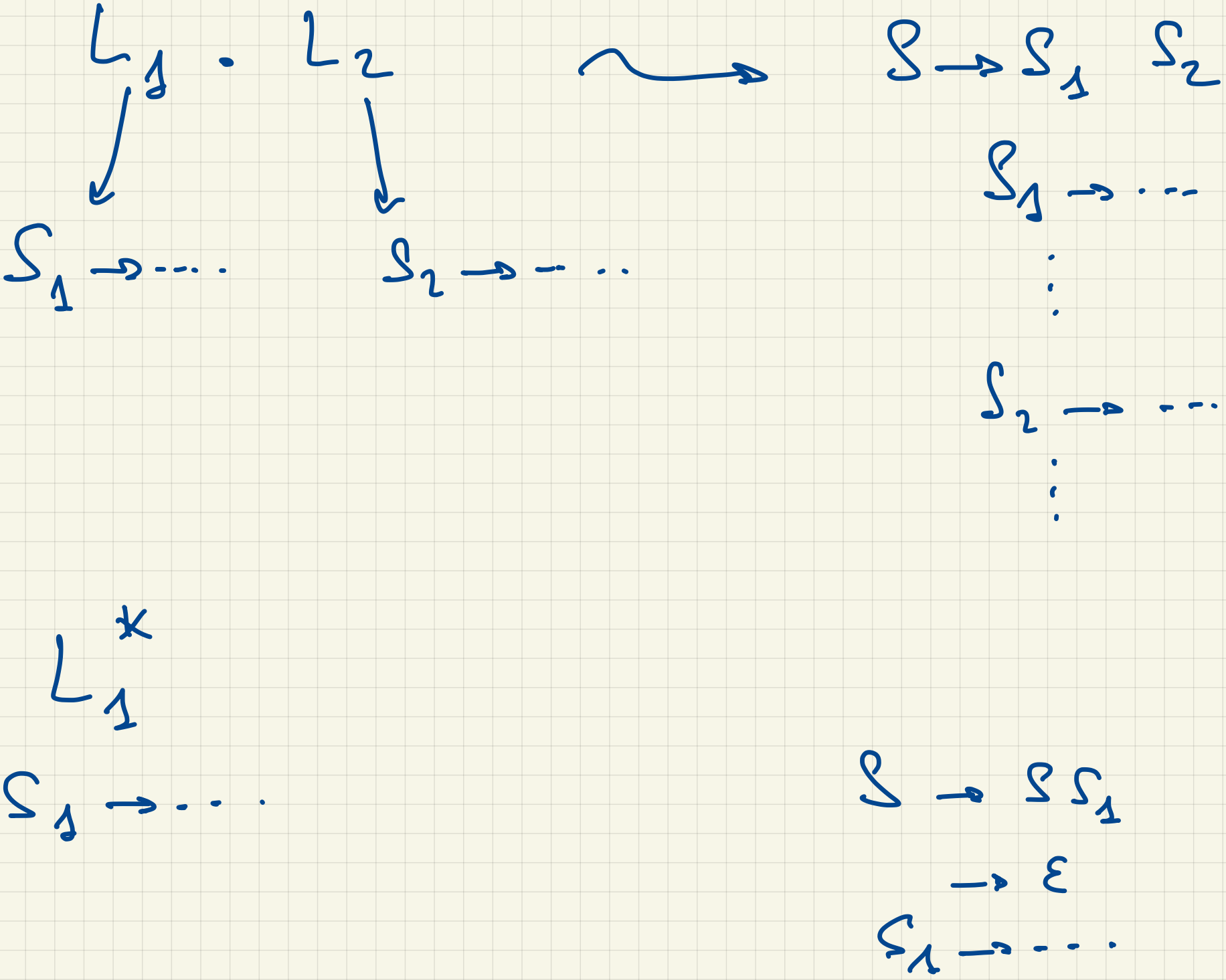
$\to a$

L_2

let G be the grammar with set of rules

$$P_1 \cup P_2 \cup \{ S \to S_1 \mid S \to S_2 \}$$

$$L(G) = L(G_1) \cup L(G_2)$$


$L_1 \cdot L_2$
$S_1 \to \cdots$
$S_2 \to \cdots$
$S \to S_1 \; S_2$
$S_1 \to \cdots$
$S_2 \to \cdots$
L_1^*
$S_1 \to \cdots$
$S \to S S_1$
$\to \varepsilon$
$S_1 \to \cdots$

What about ∩ and complement?

rem: $L_1 \cap L_2 = \overline{(\overline{L_1} \cup \overline{L_2})}$

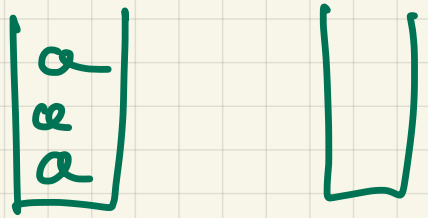
Theorem: $L_{abc} = \{ a^n b^n c^n \mid n \geq 0 \}$ is not a CFL!

Let's try: Idea n°1

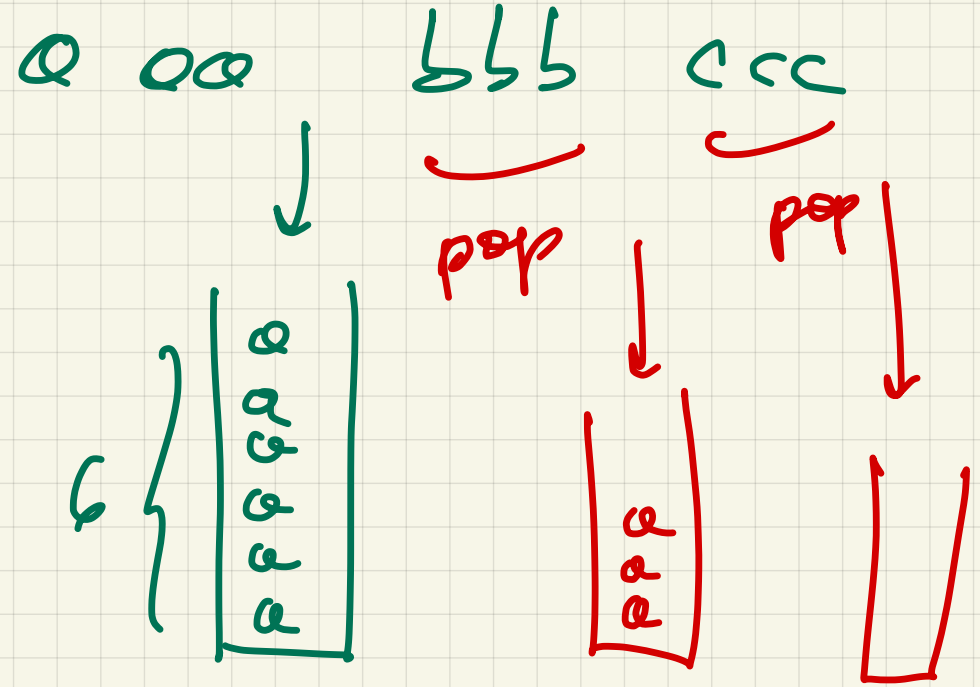
We push the a's, we pop with the b's

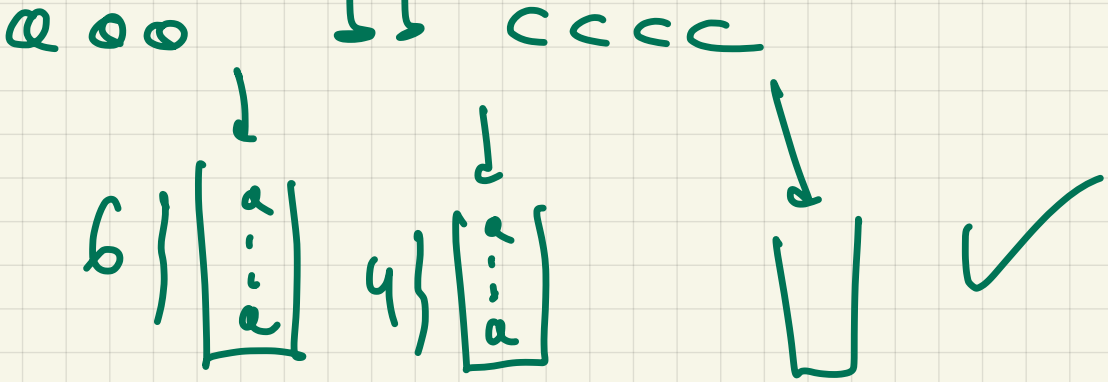
aaa bbb ccc

↓ ↓ → Correct check.

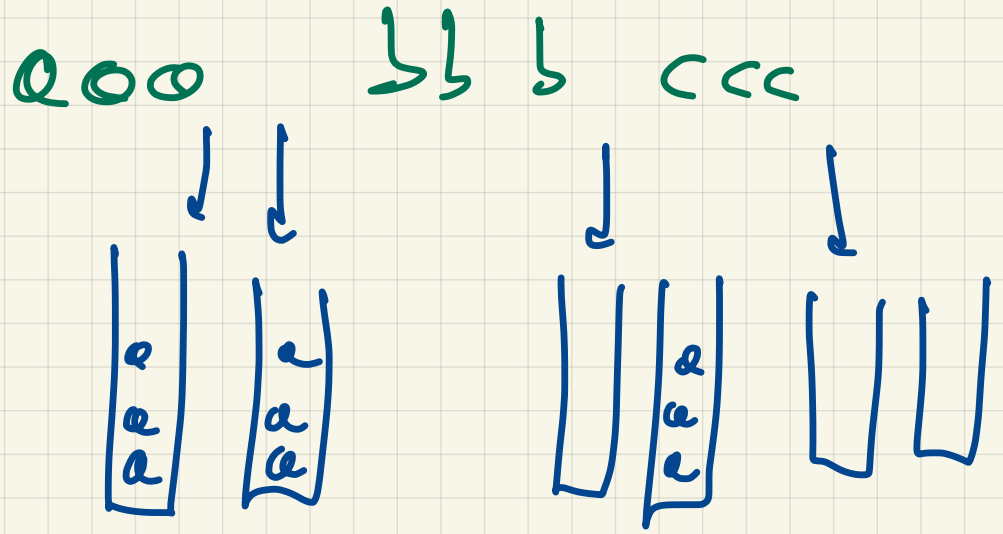


Idea now let's push two the number of a's





Idea #3: a second stack!



But this is equivalent to a Turing machine!

$$L_1 = \{ a^n b^n c^m \mid n \geq 0, m \geq 0 \}$$

aabbccccc ✓

In a CFL.

$$= \{ a^n b^n \mid n \geq 0 \} \cdot c^*$$

CFL · CFL

$$L_2 = \{ a^m b^n c^n \mid n \geq 0, m \geq 0 \}$$ in a CFL

But: $L_1 \cap L_2 = L_{abc} \notin$ CFL.

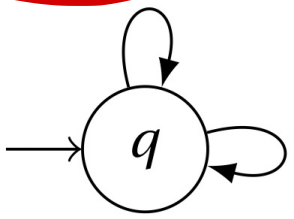
Top-down Parsers

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

Top-down parser.

ε, Exp/Exp + Exp
ε, Exp/Exp ∗ Exp
ε, Exp/ (Exp)
ε, Exp/Id
ε, Exp/Cst

Z_0 = Exp

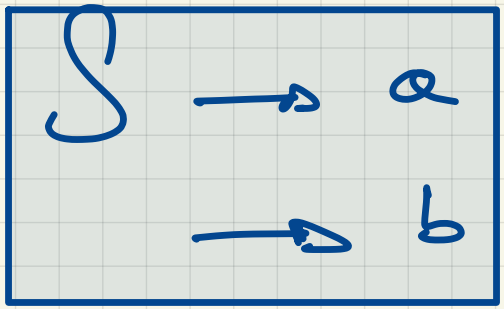


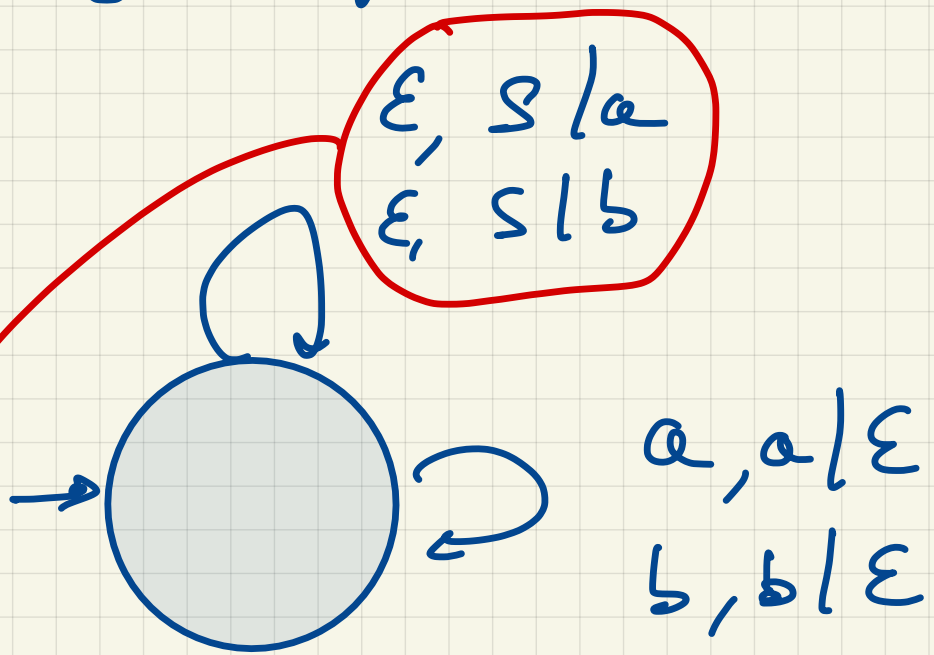
Id, Id/ε ∗, ∗/ε
Cst, Cst/ε +, +/ε
(, (/ε),)/ε

⚠ not deterministic !!

Because Exp can be derived in ≠ ways!

Let's try to lift non-determinism!



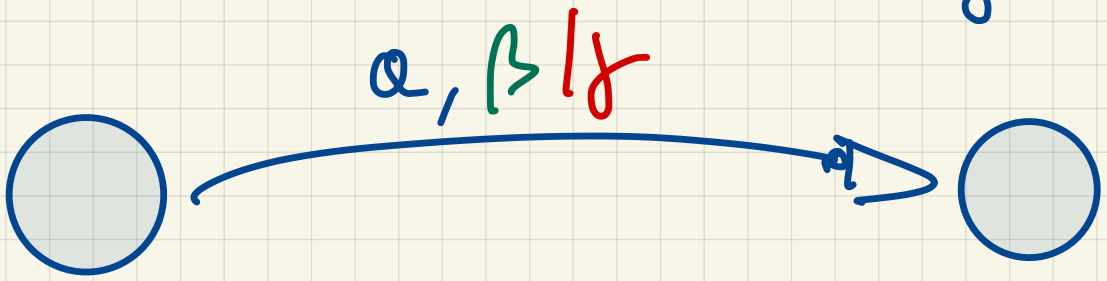


When we have S the PDA doesn't know which choice to take!

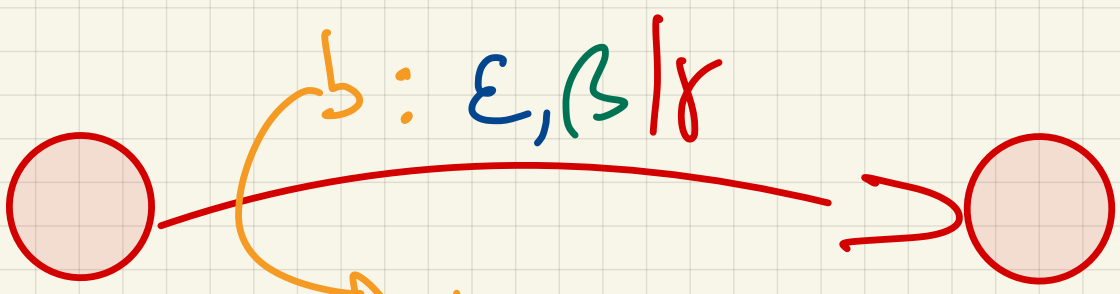
If the input word is a, it makes no sense to try S → b.

We will use PDAs with look-ahead

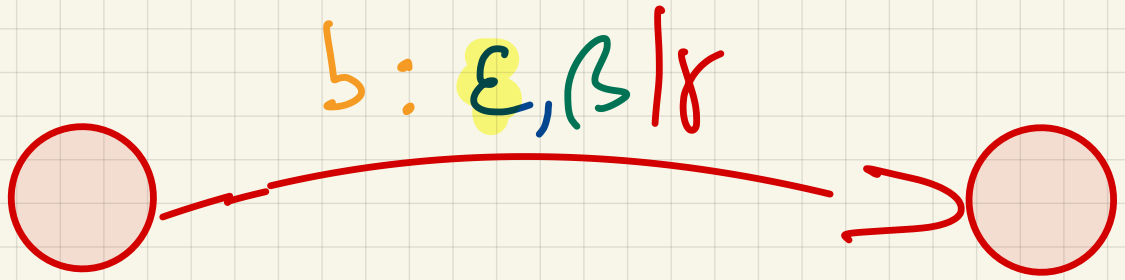
These PDAs can look at the next characters on the input to take their decision without reading the characters!



w/o look-ahead



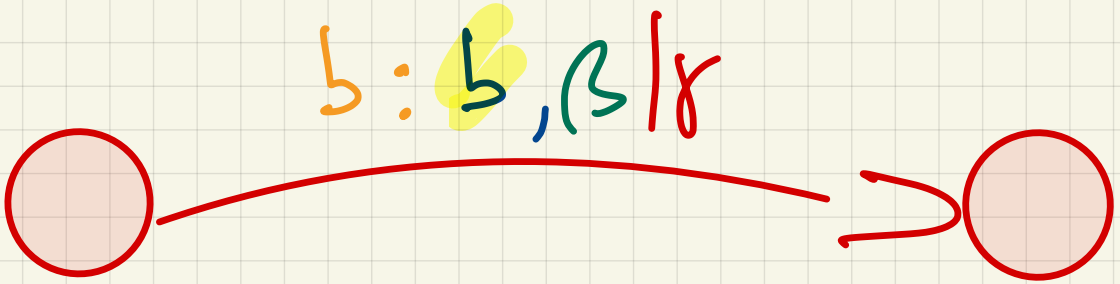
b is the next character on the input.



input: b a a c

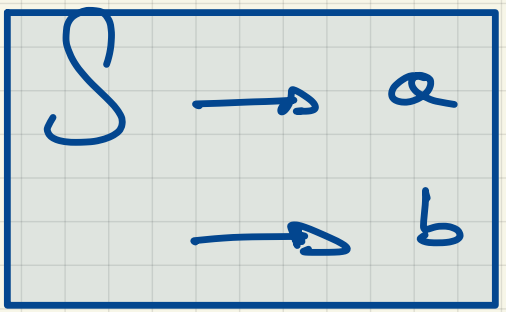
b a o c

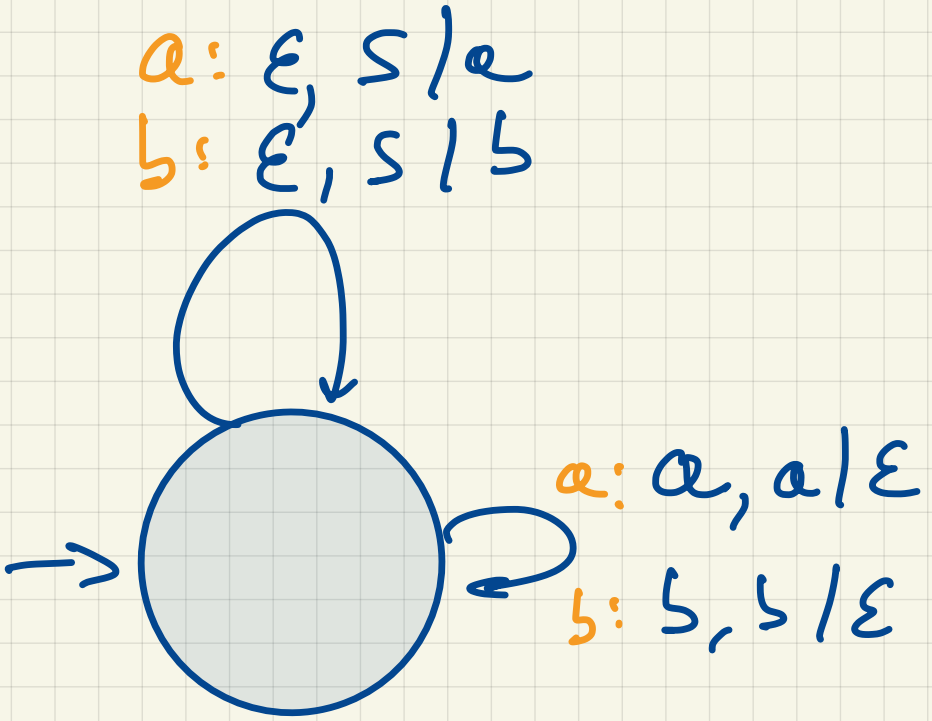
the input does not change!



input b o o c

a o c


S
a
b


a: ε, S / a
b: ε, S / b
a: a, a / ε
b: b, b / ε

Let's exploit this on an example

(1)	A	$\rightarrow$	aaa
(2)		$\rightarrow$	Bbb
(3)		$\rightarrow$	Cdd
(4)	B	$\rightarrow$	b
(5)	C	$\rightarrow$	c
(6)		$\rightarrow$	ε

I need help to left the non-determinism ? Which decision should I take when A is on the top of the stack:

	a	b	c	d	)
A	1	2	3	3	

look-ahead.

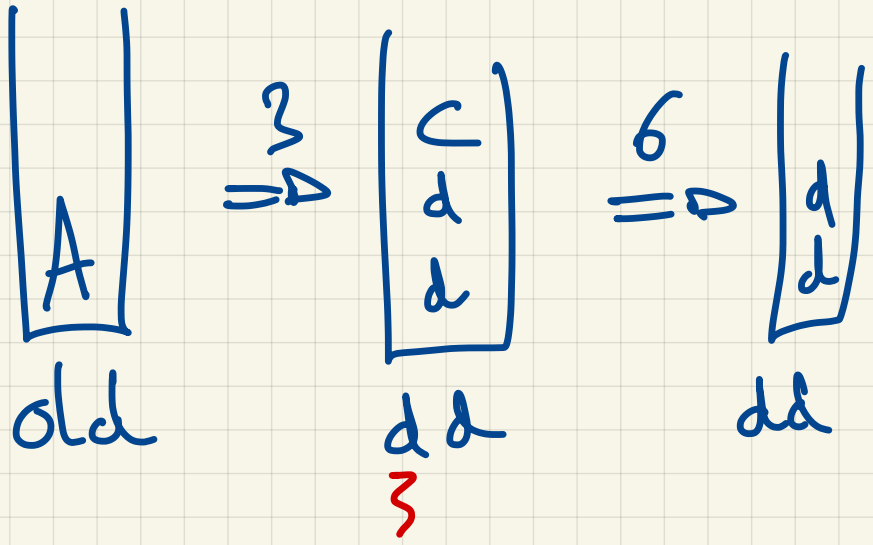
top of stack.

2 because B → b

A ⇒ Cdd
5 ↓ cdd
6 ↓ dd

(1)	A	$\rightarrow$	aaa
(2)		$\rightarrow$	Bbb
(3)		$\rightarrow$	Cdd
(4)	B	$\rightarrow$	b
(5)	C	$\rightarrow$	c
(6)		$\rightarrow$	ε

	a	b	c	d
A	1	2	3	3
B	x	4	x	x
C	x	x	5	6



✓ no conflict!

(1) $A \rightarrow$ aaa

(2) $\rightarrow B$bb

(3) $\rightarrow C$dd

(4) $B \rightarrow$ b

$\rightarrow$ aa

(5) $C \rightarrow$ c

(6) $\rightarrow \varepsilon$

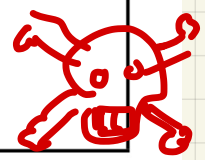
An example of Conflict

	a	b	c	d
A	1, 2			

$First^k(\alpha)$ = all prefixes of length k of words that can be generated from α.

↑ sentential form

(1)	A	→	aaa
(2)		→	Bbb
(3)		→	Cdd
(4)	B	→	b
(5)	C	→	c
(6)		→	ε



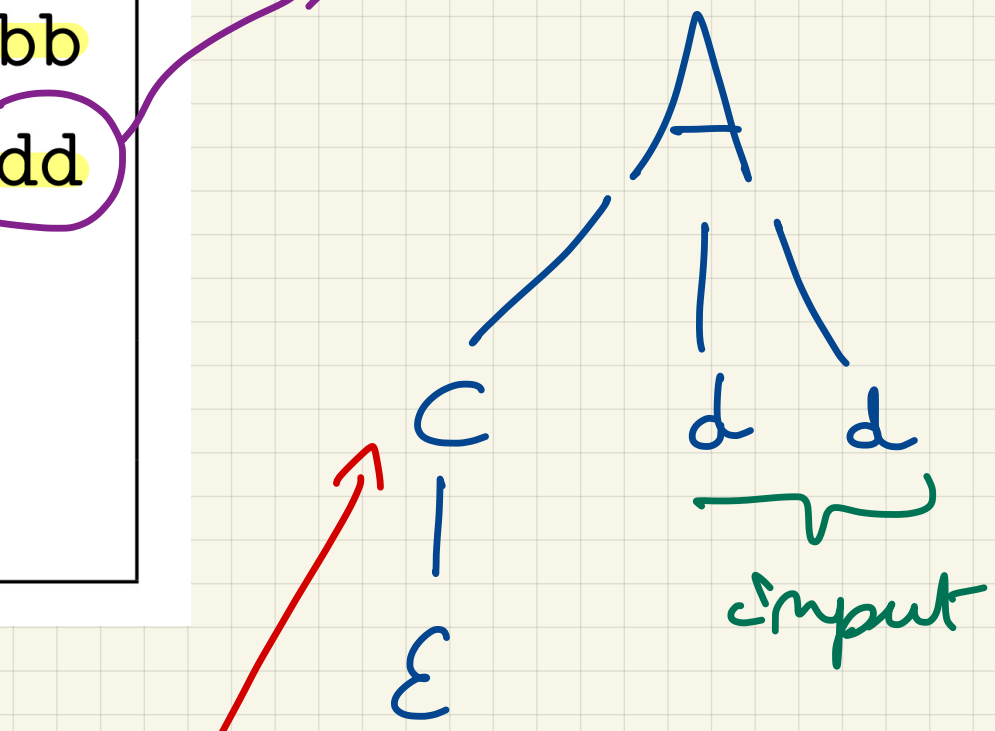
	a	b	c	d
A	1			3

$First^1(aaa) = \{a\}$

$First^1(Cdd) = \{c, d\}$

(1)	A	$\rightarrow$	aaa
(2)		$\rightarrow$	Bbb
(3)		$\rightarrow$	Cdd
(4)	B	$\rightarrow$	b
(5)	C	$\rightarrow$	c
(6)		$\rightarrow$	ε

$\text{Follow}^1(C) \supseteq \text{First}^1(dd)$



top of the stack.

$\text{Follow}^1(C)$ = set of all prefixes of length 1 of words that can follow what is generated by C

What about this grammar?

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

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

$First^1(Exp + Exp) = \{Id, Cst, (\}$

$First^1(Exp * Exp) = \{Id, Cst, (\}$

All grammars with left-recursion will generate conflicts.

First, let's fix the priority of the operators

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

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

Id + Id * Id