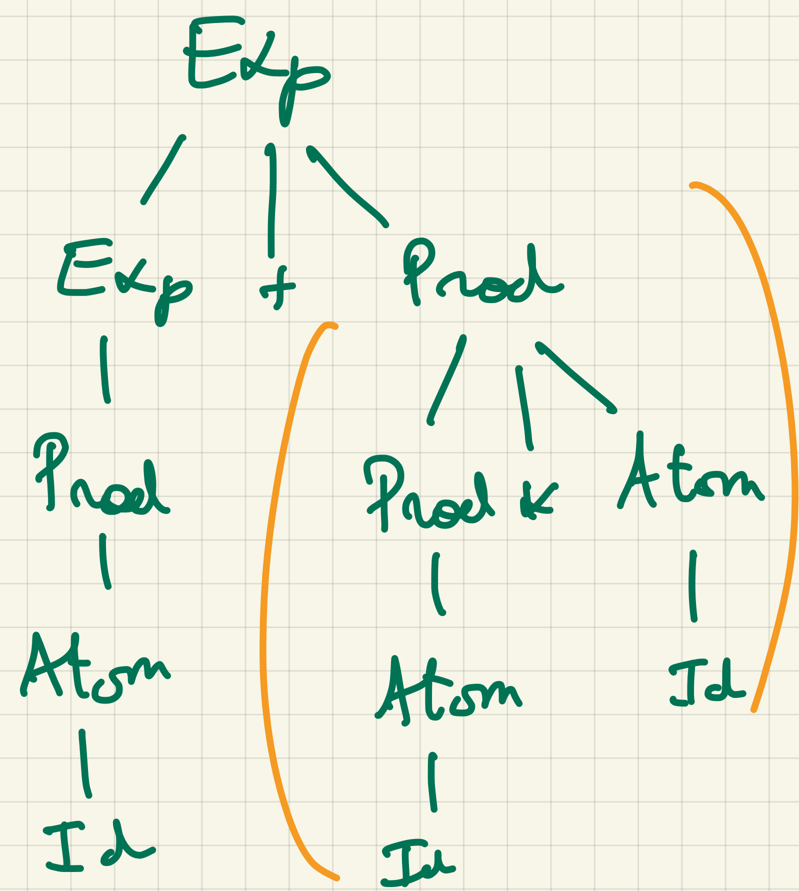
October, 24th

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

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

Id + (Id * Id)



Prod $\Rightarrow \cdots \Rightarrow w$, w contains no +

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

But... we still love left-recursion!!

First (Exp + Prod) = { Cst, ... }

Exp ⇒ Prod ⇒ Atom ⇒ Cst

First (Prod) = { Cst, ... }

Removing left-recursion

It will always be problematic for top-down parsers.

$$\boxed{\begin{aligned} S &\rightarrow Sa \\ &\rightarrow \varepsilon \end{aligned}}$$

$$\begin{array}{l} (1) \\ (2) \\ (3) \end{array} \boxed{\begin{aligned} S &\rightarrow Aa \\ A &\rightarrow Sb \\ A &\rightarrow \varepsilon \end{aligned}}$$

indirect

$\text{First}(Aa) = \{a\}$

$\text{First}(Sb) = \{a\}$

$\text{Follow}(A) = \{a\}$

	a	b
S	(1)	
A	(2) (3)	

$$V \to V\alpha_1$$
$$\to V\alpha_2$$
$$\to \beta_1$$
$$\to \beta_2$$

$$V \Rightarrow V\alpha_1 \Rightarrow V\alpha_1\alpha_1$$
$$\Rightarrow V\alpha_2\alpha_1\alpha_1$$
$$\Rightarrow \beta_2\alpha_2\alpha_1\alpha_1$$

$\Rightarrow$

$$V \to \beta_1 V'$$
$$\to \beta_2 V'$$
$$V' \to \alpha_1 V'$$
$$\to \alpha_2 V'$$
$$\to \varepsilon$$

$$V \Rightarrow \beta_2 V'$$
$$\Rightarrow \beta_2\alpha_2 V'$$
$$\Rightarrow \beta_2\alpha_2\alpha_1 V'$$
$$\Rightarrow \beta_2\alpha_2\alpha_1\alpha_1 V'$$
$$\Rightarrow \beta_2\alpha_2\alpha_1\alpha_1$$

Back to the grammar...

(1)	Exp	$\rightarrow$	Exp + Exp
(2)		$\rightarrow$	Exp − Exp
(3)		$\rightarrow$	Exp ∗ Exp
(4)		$\rightarrow$	Exp / Exp
(5)		$\rightarrow$	(Exp)
(6)		$\rightarrow$	−Exp
(7)		$\rightarrow$	Id
(8)		$\rightarrow$	Cst

high priority

$-5 * 2 = (-5) * 2$

priority

(1)	Exp	$\rightarrow$	Exp + Prod
(2)		$\rightarrow$	Exp − Prod
(3)		$\rightarrow$	Prod
(4)	Prod	$\rightarrow$	Prod ∗ Atom
(5)		$\rightarrow$	Prod / Atom
(6)		$\rightarrow$	Atom
(7)	Atom	$\rightarrow$	−Atom
(8)		$\rightarrow$	Cst
(9)		$\rightarrow$	Id
(10)		$\rightarrow$	(Exp)

Removing left recursion

(1)	Exp	$\rightarrow$	Exp + Prod
(2)		$\rightarrow$	Exp − Prod
(3)		$\rightarrow$	Prod
(4)	Prod	$\rightarrow$	Prod $*$ Atom
(5)		$\rightarrow$	Prod / Atom
(6)		$\rightarrow$	Atom
(7)	Atom	$\rightarrow$	−Atom
(8)		$\rightarrow$	Cst
(9)		$\rightarrow$	Id
(10)		$\rightarrow$	(Exp)

$\Longrightarrow$

(1)	Exp	$\rightarrow$	Prod Exp′
(2)	Exp′	$\rightarrow$	+Prod Exp′
(3)		$\rightarrow$	−Prod Exp′
(4)		$\rightarrow$	ε
(5)	Prod	$\rightarrow$	Atom Prod′
(6)	Prod′	$\rightarrow$	$*$Atom Prod′
(7)		$\rightarrow$	/Atom Prod′
(8)		$\rightarrow$	ε
(9)	Atom	$\rightarrow$	−Atom
(10)		$\rightarrow$	Cst
(11)		$\rightarrow$	Id
(12)		$\rightarrow$	(Exp)

Let's try to build the parser

(1)	S	$\rightarrow$	Exp\$
(2)	Exp	$\rightarrow$	Prod Exp′
(3)	Exp′	$\rightarrow$	+ Prod Exp′
(4)		$\rightarrow$	− Prod Exp′
(5)		$\rightarrow$	ε
(6)	Prod	$\rightarrow$	Atom Prod′
(7)	Prod′	$\rightarrow$	∗ Atom Prod′
(8)		$\rightarrow$	/ Atom Prod′
(9)		$\rightarrow$	ε
(10)	Atom	$\rightarrow$	− Atom
(11)		$\rightarrow$	Cst
(12)		$\rightarrow$	Id
(13)		$\rightarrow$	(Exp)

\$ = end of file
= terminal marker.

	+	−	∗	/	Cst	Id	(	)	\$
S		1			1	1	1		
Exp		2			2	2	2		
Exp′	3	4						5	5
Prod		6			6	6	6		
Prod′	9	9	7	8				9	9
Atom		10			11	12	13		

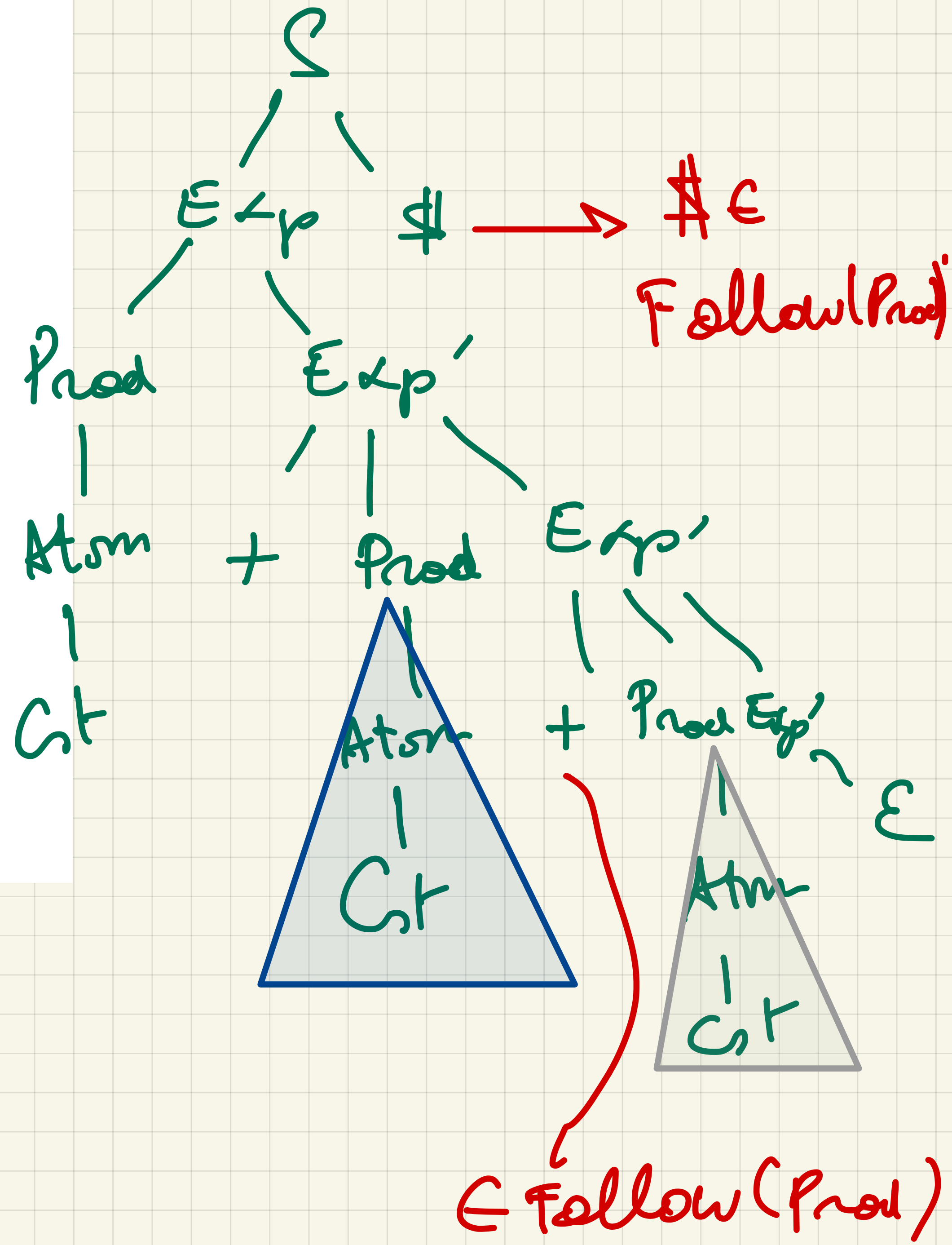
(1)	S	$\to$	Exp\$
(2)	Exp	$\to$	Prod Exp′
(3)	Exp′	$\to$	+ Prod Exp′
(4)		$\to$	− Prod Exp′
(5)		$\to$	ε
(6)	Prod	$\to$	Atom Prod′
(7)	Prod′	$\to$	∗ Atom Prod′
(8)		$\to$	/ Atom Prod′
(9)		$\to$	ε
(10)	Atom	$\to$	− Atom
(11)		$\to$	Cst
(12)		$\to$	Id
(13)		$\to$	(Exp)

First(S) = { −, Cst, Id, (}

Follow (Exp′)
= Follow (Exp)
= { \$,) }

Follow (Prod′) = Follow (Prod) =
{ +, −, \$,) }

(1)	S	$\rightarrow$	Exp$
(2)	Exp	$\rightarrow$	Prod Exp′
(3)	Exp′	$\rightarrow$	+ Prod Exp′
(4)		$\rightarrow$	− Prod Exp′
(5)		$\rightarrow$	ε
(6)	Prod	$\rightarrow$	Atom Prod′
(7)	Prod′	$\rightarrow$	$*$ Atom Prod′
(8)		$\rightarrow$	/ Atom Prod′
(9)		$\rightarrow$	ε
(10)	Atom	$\rightarrow$	− Atom
(11)		$\rightarrow$	Cst
(12)		$\rightarrow$	Id
(13)		$\rightarrow$	(Exp)



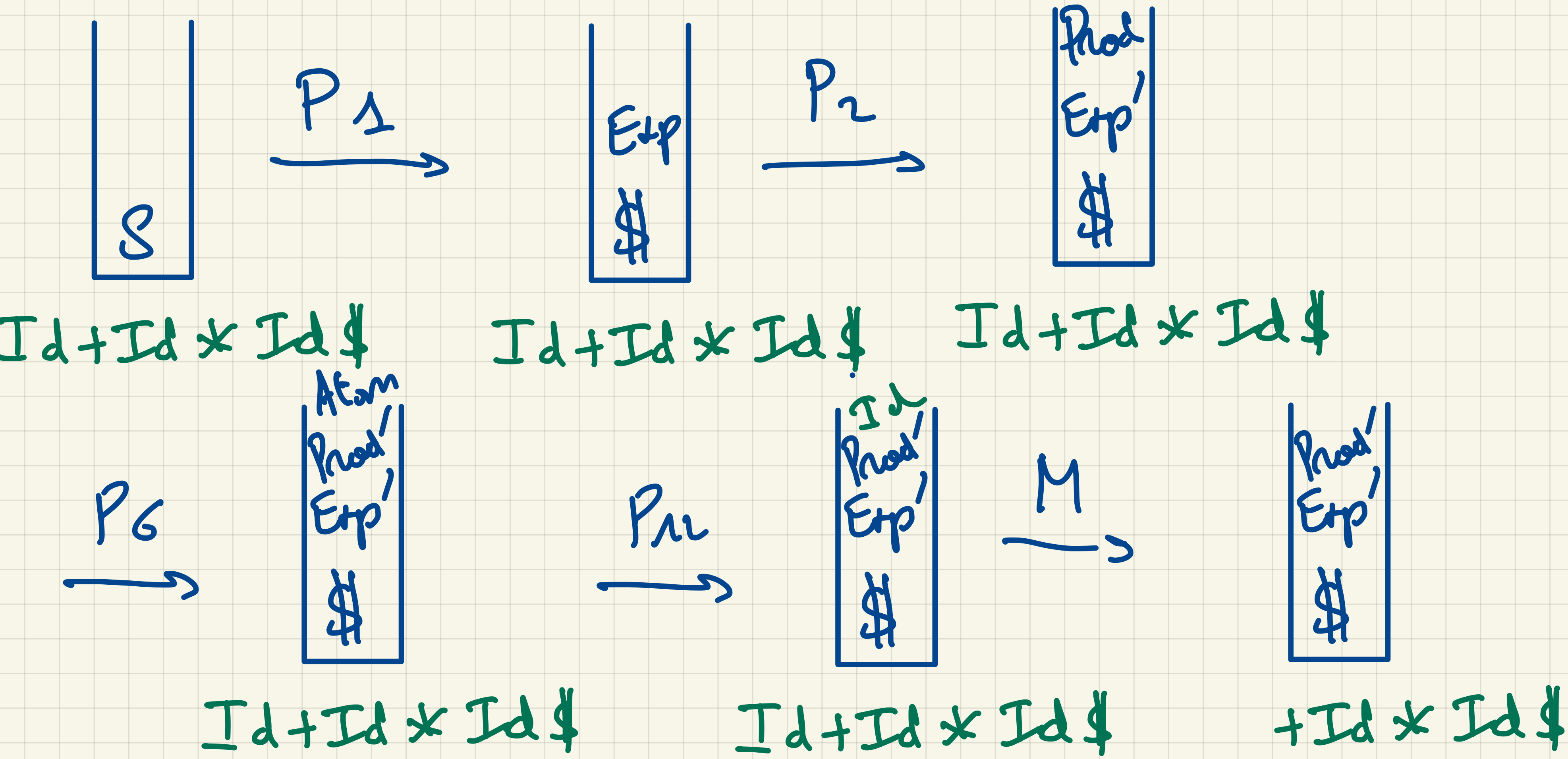
M	\$	+	−	$*$	/	Cst	Id	(	)
S			1			1	1	1	
Exp			2			2	2	2	
Exp′	5	3	4						5
Prod			6			6	6	6	
Prod′	9	9	9	7	8				9
Atom			10			11	12	13	
\$	A → accept								
+		M							
−			M						
$*$				M					
/					M				
Cst						M			
Id							M		
(								M	
)									M

Produce

Match

M	\$	+	−	$*$	/	Cst	Id	(	)
S			1			1	1	1	
Exp			2			2	2	2	
Exp′	5	3	4						5
Prod			6			6	6	6	
Prod′	9	9	9	7	8				9
Atom			10			11	12	13	

(1)	S	$\to$	Exp\$
(2)	Exp	$\to$	Prod Exp′
(3)	Exp′	$\to$	+Prod Exp′
(4)		$\to$	−Prod Exp′
(5)		$\to$	ε
(6)	Prod	$\to$	Atom Prod′
(7)	Prod′	$\to$	$*$Atom Prod′
(8)		$\to$	/Atom Prod′
(9)		$\to$	ε
(10)	Atom	$\to$	−Atom
(11)		$\to$	Cst
(12)		$\to$	Id
(13)		$\to$	(Exp)



LL(k) grammars

look-ahead.

left scanning: we scan from left to right

left parsing: we generate a leftmost derivation.

A grammar is LL(k) iff it can be parsed deterministically by a top-down parser using k symbols of look-ahead.

Simple observations

① $LL(k) \subseteq LL(k+1)$

the set of all $LL(k)$ grammars

② For all $k \geq 0$: a grammar that has left-recursion is not $LL(k)$

We want to characterize LL(k) grammars

We will characterize the situations that "confuse" the parser with k characters of look-ahead.

$$\text{Assume:} \quad S \Rightarrow^* \overset{\text{terminals.}}{\widehat{w}} A \gamma \qquad A \in V$$

and in this derivation, we need to do: $A \to \alpha_1$

$$S \Rightarrow^* w A \gamma \Rightarrow w \alpha_1 \gamma \Rightarrow^* \overset{\in T^*}{\widehat{w x_1}}$$

matched!

$$S \Rightarrow^* w A \gamma \Rightarrow w \alpha_1 \gamma \Rightarrow^* \overbrace{w x_1}^{\in T^*}$$

Let's assume there is another derivation in the grammar, where the right choice is $A \to \alpha_2$ $(\alpha_1 \neq \alpha_2)$

$$S \Rightarrow^* w A \gamma \Rightarrow w \alpha_2 \gamma \Rightarrow \overbrace{w x_2}^{\in T^*}$$

The look-ahead should allow me to take the right decision!!

Case ①: look-ahead $= \text{First}^k(x_1)$

Case ② $\quad$ — " — $= \text{First}^k(x_2)$

must be $\neq$

Definition 5.10 (LL(k) CFGs)**.** A CFG $\langle P, T, V, S\rangle$ is LL(k) iff for all pairs of derivations:

$$S \Rightarrow^* wA\gamma \Rightarrow w\alpha_1\gamma \Rightarrow^* wx_1$$
$$S \Rightarrow^* wA\gamma \Rightarrow w\alpha_2\gamma \Rightarrow^* wx_2$$

with $w, x_1, x_2 \in T^*$, $A \in V$ and $\gamma \in (V \cup T)^*$, *and* $\mathrm{First}^k(x_1) = \mathrm{First}^k(x_2)$, we have: $\alpha_1 = \alpha_2$.

Problem: semantic definition

Talks about the infinitely many $\neq$ derivations.

We would like a syntactic definition

that talks about the rules of the grammar!

Definition 5.12 (Strong LL(k) CFG)**.** A CFG $G = \langle V, T, P, S\rangle$ is *strong* LL(k) iff, for all pairs of rules $A \to \alpha_1$ and $A \to \alpha_2$ in P (with $\alpha_1 \neq \alpha_2$):

$$\mathrm{First}^k\Big(\alpha_1 \mathrm{Follow}^k(A)\Big) \cap \mathrm{First}^k\Big(\alpha_2 \mathrm{Follow}^k(A)\Big) = \emptyset$$

Theorem:

$$\text{Strong } LL(k) \subsetneq LL(k)$$

There are grammars that you can parse with k characters of look-ahead but are not strong $LL(k)$.

But

$$\boxed{\text{Strong } LL(1) = LL(1).}$$