example:

① $G \to E$

② $E \to T\ E'$

③ $E' \to +\ T\ E'$
$\quad\quad -\ T\ E'$
$\quad\quad \varepsilon$

⑥ $T \to F\ T'$

⑦ $T' \to *\ F\ T'$

⑧ $\quad\quad /\ F\ T'$

⑨ $\quad\quad \varepsilon$

⑩ $F \to (\ E\ )$

⑪ $\quad\quad id$

⑫ $\quad\quad num$

First:	G	E	E'	T	T'	F
	(, num id	(, num id	ε, + −	(, num id	* / ε	(num id

	G	E	E'	T	T'	F
follow:						
init	EOF	ϕ	ϕ	ϕ	ϕ	ϕ
1st.	EOF	EOF,)	EOF	+, − EOF	+, − EOF	+, −, EOF * /
2nd.	EOF	EOF,)	EOF)	+, −, EOF)	+, −, EOF)	+, − EOF *, /,)
3rd.	EOF	EOF,)	EOF)	+, −, EOF)	+, − EOF)	+, − EOF * /)

03/24/2025

(2 + 3)

2 + 3 EOF

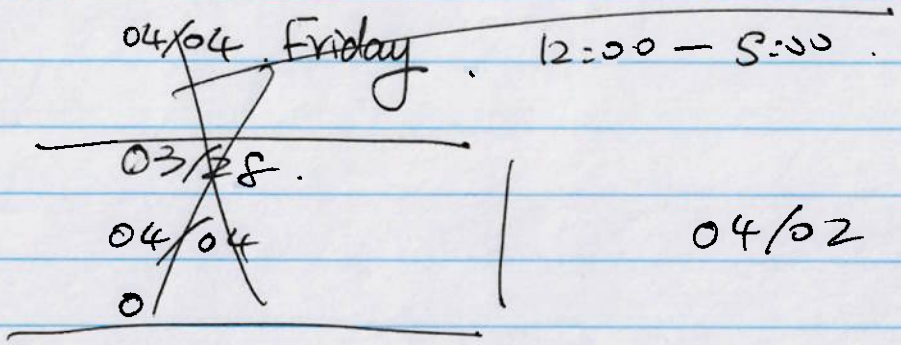
2 + 3

2 * 3

2

2 2

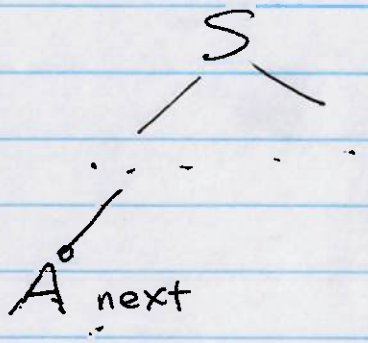
Exam:



with first. follow. we can differentiate

$A \to \varepsilon$

syntax error.



next $\in$ first(A) $\vee$ pick a rule $A \to -$

next $\notin$ first(A) $\varepsilon \notin$ first(A) error

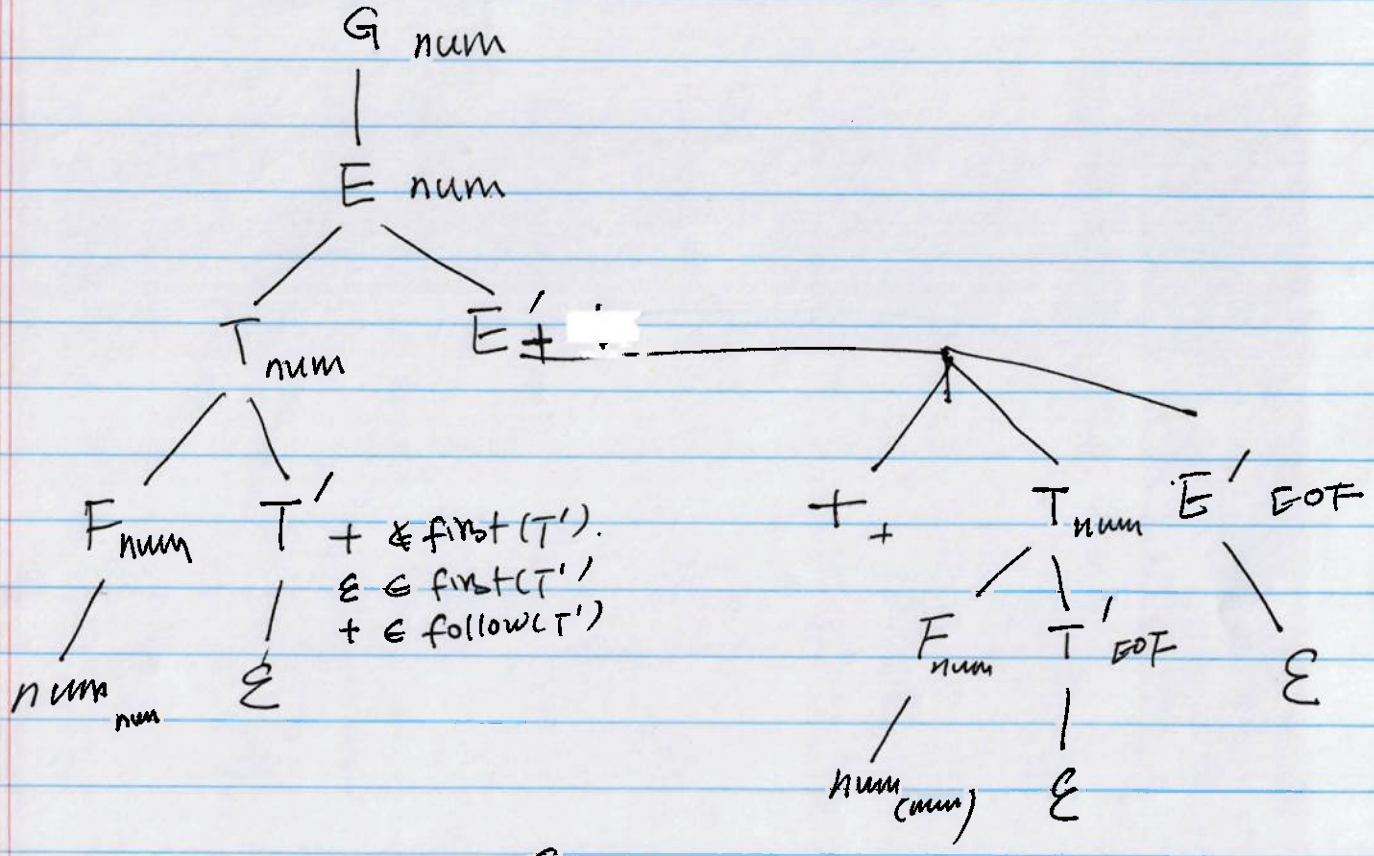
next $\notin$ first(A) $\varepsilon \in$ first(A), next $\in$ follow(A)

$A \to \varepsilon$

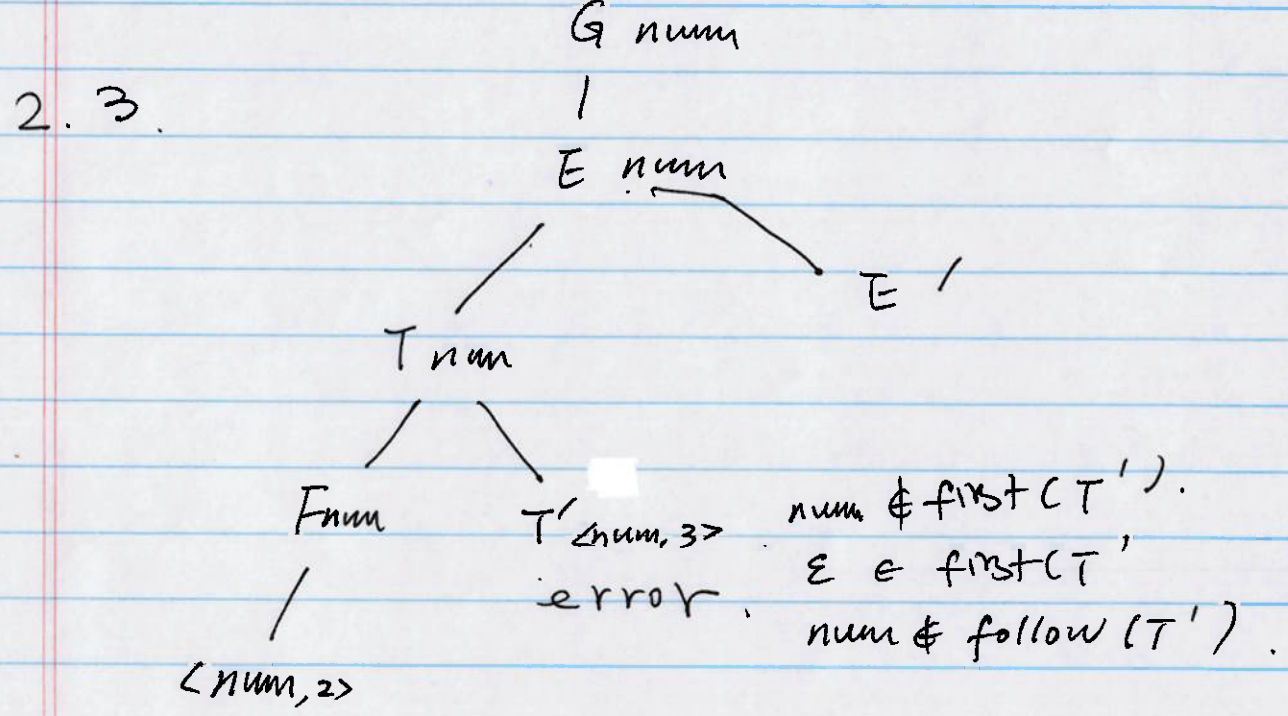
next $\notin$ first $\varepsilon \in$ first next $\notin$ follow

error

2 + 3 eof . num + num eof .



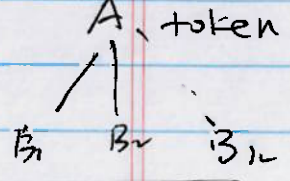
2.3.



2.3 start of production rule.

pick the right production.

$$start(A \to B) = \begin{cases} first(B) & \varepsilon \notin first(B) \\ first(B) - \{\varepsilon\} \cup follow(A) & \varepsilon \in first(B) \end{cases}$$



ex1. ① $S \to AB$ start(①, $S \to AB$) = {a}.

② $A \to aA$ start($A \to aA$) = {a}.

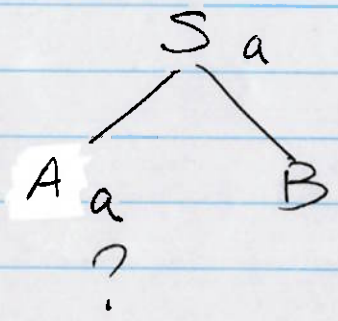
③ $A \to a$ start($A \to a$) = {a}

④ $B \to bB$ start($B \to bB$) = {b}

⑤ $B \to b$ start($B \to b$) = {b}

	S	A	B
first.	a	a	b
follow.	EOF	b	EOF

ab EOF :



start

ex2. ① $S \rightarrow AB$ | {a, b, eof} | first(AB) − {ε} ∪ follow(AB).
= [first(A) − {ε} ∪ first(B)] − {ε} ∪ follow(B)

② $A \rightarrow aA$ | {a}

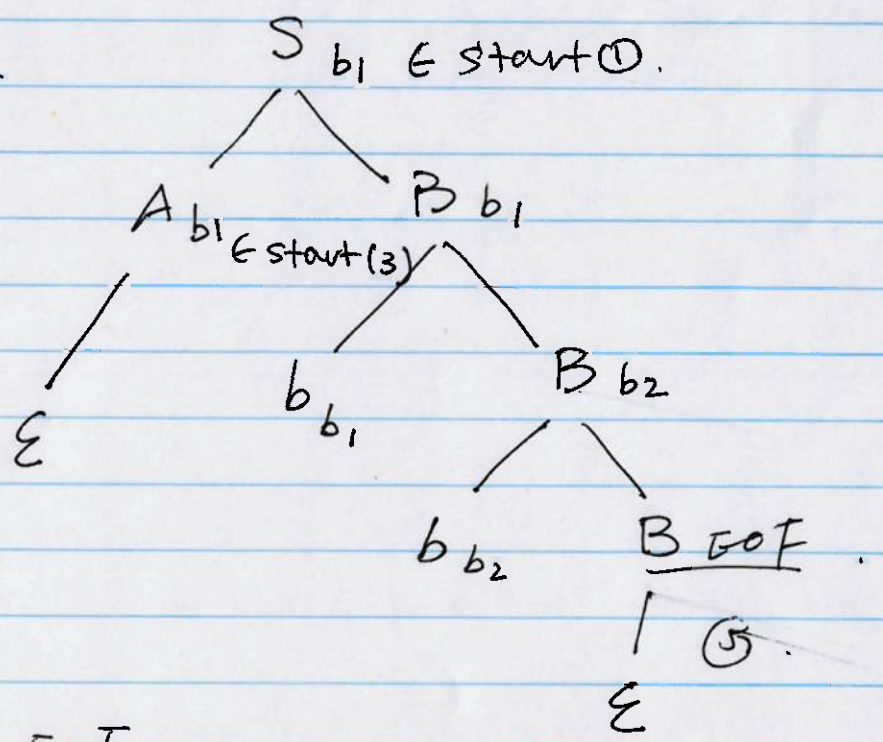
③ ε | ~~first(ε)~~ {ε} ∪ follow(A) = {b, EOF}.

④ $B \rightarrow bB$ | {b}

⑤ ε | first(ε) − {ε} ∪ follow(B) = {EOF}.

	S	A	B
first	{a, b, ε}	{a, ε}	{b, ε}.
follow.	{EOF}.	{b. EOF}.	{EOF}

$b_1 b_2$ EOF.



$a_1 a_2 b_1 b_2 b_3$ EOF.

Ex.

0	$G \to E$		$= first(E) = \{(, name, num\}$ (start)
1	$E \to TE'$		$= first(TE') = first(T) = \{(, name, num\}$
2	$E' \to +TE'$		$= first(+TE') = first(+) = \{+\}$
3		$-TE'$	$= first(-TE') = first(-) = \{-\}$
4		ε	$= first(\varepsilon) - \{\varepsilon\} \cup follow(E')$ $= \{EOF,)\}$
5	$T \to FT'$		$= \{(, num, name\}$
6	$T' \to *FT'$		$\{*\}$
7		$/FT'$	$\{/\}$
8		ε	$\{+, -, EOF,)\}$
9	$F \to (E)$		$\{(\}$
10		num	$\{num\}$
11		name	$\{name\}$

	G	E	E'	T	T'	F
first	(, num, name	(, num, name	+, -, ε	(, num, name	*, /, ε	(, num, name
follow	EOF	EOF,)	EOF,)	+, -, EOF,)	+, -, EOF,)	+, -, *, /, EOF,)

name * num + num EOF

2.4. characteristics of backtrack free grammar — LL(1)

A CFG is backtrack free if for any NT A with multiple rhs

$$A \to \beta_1$$
$$A \to \beta_2$$
$$\vdots$$
$$A \to \beta_n$$

for any $i \neq j$, $start(A \to \beta_i) \cap start(A \to \beta_j) = \phi$

① not all CFGs are backtrack free

ex1 is not backtrack free.

$S \to AB$

$A \to aA \mid a$

$B \to bB \mid b$

a
a
a
b
b

② rewrite a grammar to make it backtrack free using left factoring

$S \to AB$

$A \to aA \mid a$ $\Longrightarrow$ $A \to aA'$, $A' \to A \mid \varepsilon$

$B \to bB \mid b$ $\Longrightarrow$ $B \to bB'$, $B' \to B \mid \varepsilon$

$S \to AB$: a

$A \to aA'$: a

$A' \to A$: a

ε : b

$B \to bB'$: b

$B' \to B$: b

ε : EOF.

3. top-down parser for LL(1) grammer.

LL(1): left to right scan, leftmost derivation, 1 look ahead one word

3.2 table driver parser.

one row for each NT.

one col for each T and EOF.

cell(A, a) = { error / rule # }

start.

ex. ① $S \rightarrow AB$ — a

② $A \rightarrow aA'$ — a

③ $A' \rightarrow A$ — a, b

④ ϵ — a, b

⑤ $B \rightarrow bB'$ — b

⑥ $B' \rightarrow B$ — b

⑦ ϵ — eof.

NT \ T	a	b	eof
S	①	×	×
A	②	×	×
A'	③	④	×
B	×	⑤	×
B'	×	⑥	⑦

skeleton parsor

```
word = next word( )
push eof into stack.
push S
while true.
    focus = top of the stack.
    if focus = eof and word = eof
        success.
        break.
    else if. focus ∈ T or focus = eof.
        if focus = word
            = pop stack.
            word = next word
```

```
        else.
            error
            break.
    else. // NT.
        if table [ focus, word ] is A → B1 B2 -- Bk
            pop stack
            for i = k to 1
                if Bi ≠ ε
                    push Bi
        else
            error.
    // end while.
```

3.1. recursive descent parsers.

A parser is a set of mutually recursive procedures, one procedure for each NT A. assuming "word" stores next token from input.

```
A ( )                                   sub
    try to do derivation from A // builds parse tree root at A
    if success
            read next word
            return true.
    else
            return false.
```

```
main( ).    // recognize. start symbol S.
    word = next word( )
    if (S( ))
        if word = eof
            return true
        else
            return false
    else
        return false.
```

ex1. $S \rightarrow AB$

$A \rightarrow aA'$

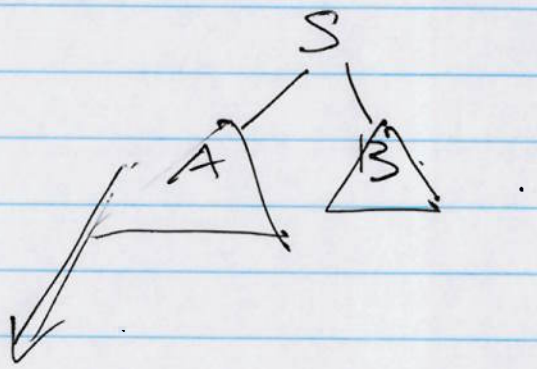
$A' \rightarrow A$

ε

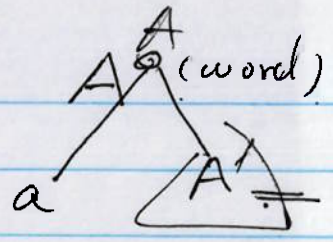
$B \rightarrow bB'$

$B' \rightarrow B$

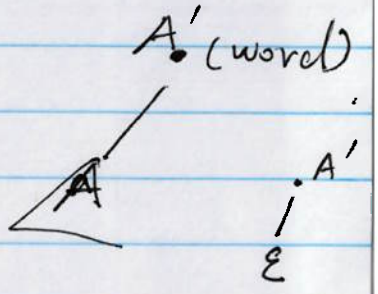
ε .




```
* S( )
    if (A( ))
        returnB( )
    else
        return false.
```




```
* A.( )          // A → a A'
    if word = 'a'
        word = next word( )
        return A'( )
    else return false .
```




```
* A'( )          //  A' → A      a
                      | ε       b
    if word = 'a'
        return A( )
    else word = 'b'
        return true
    else
        return false .

* B( ).          //  B → b B'
    ..

* B'( )          // B' → B  b
                      | ε  eof .
    if word = 'b'
        return B( )
    else if word = eof
        return true
    else return false
```

start

ex.2. $G \rightarrow E$

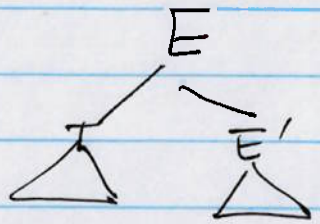
$E \rightarrow TE'$

$E' \rightarrow +TE'$ +

$-TE'$ −

ε), EOF

main( ) . .



* E( )

if T( )

return E'( )

else return false .

* E'( )

if word = "+" or "−"

word = next word( ).

If (T( ))

return E'( ).

else return false

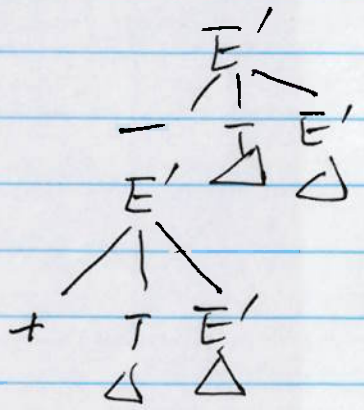
else if word = ')' or EOF

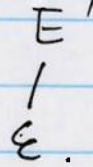
return true

else

return false .

4 cases .





quiz6. on. 03/31.

first. follow. start. parser.

exam on 04/02. 12:30 — 1:45.

1. DFA. /NFA.

2. min. DFA.

3. left recursion

4. ambiguity.

5. first. follow. start. parser.