Type	Regular Expression	RLG	LLG
Single terminal	e	$S \rightarrow e$	$S \rightarrow e$
Union operation	$(e + f)$	$S \rightarrow e \mid f$	$S \rightarrow e \mid f$
Concatenation	ef	$S \rightarrow eA,\ A \rightarrow f$	$S \rightarrow Af,\ A \rightarrow e$
Star closure	e^*	$S \rightarrow eS \mid \epsilon$	$S \rightarrow Se \mid \epsilon$
Plus closure	e^+	$S \rightarrow eS \mid e$	$S \rightarrow Se \mid e$
Star closure on union	$(e + f)^*$	$S \rightarrow eS \mid fS \mid \epsilon$	$S \rightarrow Se \mid Sf \mid \epsilon$
Plus closure on union	$(e + f)^+$	$S \rightarrow eS \mid fS \mid e \mid f$	$S \rightarrow Se \mid Sf \mid e \mid f$
Star closure on concatenation	$(ef)^*$	$S \rightarrow eA \mid \epsilon;$ $A \rightarrow fS$	$S \rightarrow Af \mid \epsilon;$ $A \rightarrow Se$
Plus closure on concatenation	$(ef)^+$	$S \rightarrow eA;$ $A \rightarrow fS \mid f$	$S \rightarrow Af;$ $A \rightarrow Se \mid e$

- Regular grammar : RLG or LLG
  - RLG: aA
  - LLG: Aa

- If a nonterminal is on right part then it can't be also ε
  ex: $S_3 \to 0S_3$ ✗ $S_3 \to \varepsilon$

- If we have a regular grammar either RLG or LLG then after or before nonterminal on the right part there should be a terminal
  ex: $S_3 \to 0S_2 \Rightarrow S_2 \to 1$ ✓
  $S_3 \to S_2 \Rightarrow S_2 \to 1$ ✗

- $x = aX + Yb \Rightarrow X = a^*Yb$, Y can be 0
  $b^+ = bb^*$
  $a + ab = a(\varepsilon + b)$

- Recursive Descendant
- the main idea is to keep trying productions until you get the sequence
- configuration : (s, i, α, β)
  - state
  - position
  - working stack
  - input stack
  - → state : q = normal state
    b = back state — this one is used for b or mi
    f = final state
    e = error state

→ position: index used for how many terminals moved to working stack (+ i) or how many terminals moved to input stack (- i) — i <= len(sequence) + 1
→ working stack: starts empty ⇒ ends up with sequence
→ input stack: starts with starting symbol ⇒ ends up empty

- moves between configuration
  → expand: used for an production if input stack starts with nonterminal
  $(q, i, \alpha, A\beta) \vdash (q, i, \alpha A_1, \gamma_1 \beta)$ where $A \to \gamma_1 | \gamma_2 \ldots$

  → advance: used for moving terminals from input stack to working stack if input stack starts with terminal
  $(q, i, \alpha, a_i \beta) \vdash (q, i+1, \alpha a_i, \beta)$ where α rest of the working stack and β rest of input stack

ex: (1) $S \to aS$ and $w = ab$
(2) $S \to b$

$(q,1,\varepsilon,S) \overset{exp}{\vdash} (q,1,S_1,aS) \overset{adv}{\vdash} (q,2,S_1a,S) \overset{exp}{\vdash} (q,2,S_1aS_2,b) \overset{adv}{\vdash}$
$\vdash (q,3,S_1aS_2b,\varepsilon) \Rightarrow w = ab$ ✓
$i = len(w)+1 = 2+1$

→ momentary insuccess: used for staying when index > len(w) + 1 or input[0] ≠ sequence (index)
$(q, i, \alpha, a_i\beta) \vdash (b, i, \alpha, a_i\beta)$ where $w[i] = b_i \ (\neq a_i)$ or $i > len(w)+1$

→ back: used for going back when working[-1] is a terminal and input is not empty yet; state must be "b" which means back appears after m.i.

$(b, i, \alpha a, \beta) \vdash (b, i-1, \alpha, a\beta)$

→ another try: used for trying next production after m.i. and working[-1] is a nonterminal

$(b, i, \alpha A_j, \gamma_j \beta) \vdash (q, i, \alpha A_{j+1}, \gamma_{j+1}\beta)$, if $\exists A, \gamma_{j+1}$

$(b, i, \alpha, A\beta)$, otherwise with the exception

(e, i, α, β), if $i=1$, $A=S$, ERROR

this case is used when working[-1] has a last production and the only option is to do the production in reverse

ex: (1) $S \to aSbS$ and $w = aacbc$

(2) $S \to aS$

(3) $S \to c$

back at

$\vdash (b, 5, S_1aS_1aS_3cbS_3, cbS) \vdash (b, 5, S_1aS_1aS_3cb, ScbS)$

→ success: when index = len(w)+1 and input stack is empty and working stack has sequence

→ error: when index = 1 and A = S (starting symbol); this configuration occurs only when there is no production that can form the sequence and you end up with index = 1 and A = S

ex: (1) $S \to aSbS$ and $w = aabbc$

(2) $S \to aS$

(3) $S \to c$

The rec dec will go until $(b, 3, S_1 a S_1 a S_3, c \$ S \$ S)$ and then does back and at until $(q, 1, S_3, c)$ which happens to be error.

- LL(1)

→ FIRST

- if X is a terminal ⇒ FIRST(X) is $\{X\}$
- if X is a nonterminal with $X \to Y_1 Y_2 \dots Y_m$ ⇒ ⇒ FIRST(X) is $\{Y_1\}$ (without ε) but if all $Y_1 Y_2 \dots Y_m$ are ε ⇒ FIRST(X) is ε

ex1: $S \to AB$ ⇒ FIRST(S) = $\{A\}$ ⇒ FIRST(S) = $\{a\}$
$A \to a$ ⇒ FIRST(A) = $\{a\}$

- if X is a nonterminal with $X \to \varepsilon$ ⇒ FIRST(X) is ε

ex2: $B \to \varepsilon$ ⇒ FIRST(B) = $\{\varepsilon\}$

→ FOLLOW

- for starting symbol $L_0 = \{\varepsilon\}$
- if we have a production $A \to \alpha B C$ ⇒ FOLLOW(B) = = FIRST(C) (without ε) if $C \to \varepsilon$ ⇒ FOLLOW(C) is added to FOLLOW(B)

ex1: $S \to BA$
FIRST(A) = $\{+, \varepsilon\}$ ⇒ FOLLOW(B) = $\{+\}$ ⇒ FOLLOW(B) = $\{+, \varepsilon\}$
$A \to \varepsilon$ and FOLLOW(A) = $\{\varepsilon\}$

- if we have a production $A \to \alpha B$ or $A \to \alpha BA$ where $A \to \varepsilon$ ⇒ FOLLOW(B) = FOLLOW(A)

ex2: $B \to \Delta C$ ⇒ after C is nothing ⇒ FOLLOW(C) = FOLLOW(B)
$C \to * \Delta C$ ⇒ FOLLOW(Δ) = FIRST(C), without (ε)
$C \to \varepsilon$ ⇒ add FOLLOW(C) in FOLLOW(Δ)

- for nonterminals with $X \to \varepsilon$ ⇒ add FOLLOW L-1

→ parsing table
- if the terminal is in FIRST(X) ⇒ prod of X
  ex1: (1) $S \to BA$, FIRST(S) = {(, a} ⇒

	a
S	BA, 1

- if the terminal is in FOLLOW(X) and $X \to \varepsilon$ ⇒
  ⇒ {ε, id}, where id is coming from $X \to \varepsilon$
  ex2: (6) $C \to \varepsilon$, FOLLOW(C) = {+, ε,)} ⇒

	+	)
C	{ε, 6}	{ε, 6}

- if $X \to \varepsilon$ ⇒ for $ ⇒ {ε, id}

- for (terminal, terminal) ⇒ pop
  for ($, $) ⇒ accept

→ conflict

$A \to \alpha B$
$A \to \alpha \gamma$ ⟹ $A \to \alpha B$, $B \to B / \gamma$

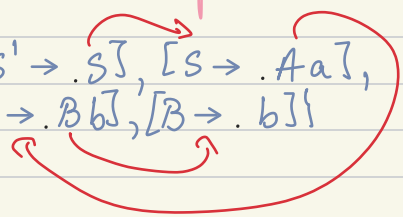
Both options are correct, but because first one has productions starting both with α the LL(1) does not know which one to use

- LR(0)

→ canonical collection

- when executing closure we have to write all productions of a nonterminal after dot

ex1: $s_0 = \text{closure}(\{[S' \to .S]\}) = \{[S' \to .S], [S \to .Aa], [A \to .Bb], [B \to .b]\}$



where $S' \to S$

(1) $S \to Aa$

(2) $A \to Bb$

(3) $B \to b$

- goto is performed for each terminal or non-terminal of a state until dot is at the final

→ parsing table

- shift: where dot is placed after terminal or non-terminal

ex1: for a at s_0:

$s_1 = \text{closure}(\{[S \to a.A]\})$

- accept: when the parser accepted the entire string

ex2: $s_1 = \text{closure}(\{[S' \to S.]\}) = \{[S' \to S.]\}$

- reduce: eliminate the production

ex3: $S' \to S$

(1) $S \to aA$

(2) $A \to bA$

(3) $A \to c$

→ for s_3 reduce prod 1

$\text{closure}(\{[S \to aA.]\}) = \{[S \to aA.]\}$

- SLR

→ parsing table

- if we have dot after a production => action will be reduce that production for FOLLOW(X) where X is left part

ex1: (1) $E \to T$

$I_2 = \{[E \to T.]\}$

$FOLLOW(E) = \{\$, +,)\}$

=> reduce 1 for +,) and $

- for state k if we have goto functions we make shift and fill goto columns

ex2: goto(s0, E), goto(s0, T), goto(s0, (), goto(s0, id), goto(s0, const) => action for (, id and const is shift state i, where i is state number of goto function. and for goto E and T shift i also

→ parse sequence

- if you don't have the terminal after the first one in any production then you must go to reduce where that first terminal have been reduced and easily start form the production for next terminals after that reduce one by one

- LR(1)

→ SLR but with FIRST instead of FOLLOW (lookahead)

- where is a closure with $[A \to a\,X\,,\,u]$ and u is a terminal => reduce is placed in table for each u including $ if $u = \$$

- the difference is when you make the canonical coll you have to take in consideration u as well, initial is $, after dot of a terminal place that value next

- u is being defined according to FIRST without ε; for a terminal X each production of that will have u equal with FIRST(Y) where Y is the terminal after X on the right part

ex: FIRST(A) = {a, b} => for each production of A, u is either a or b because S → AA, after A is A that has FIRST(A) = {a, b}