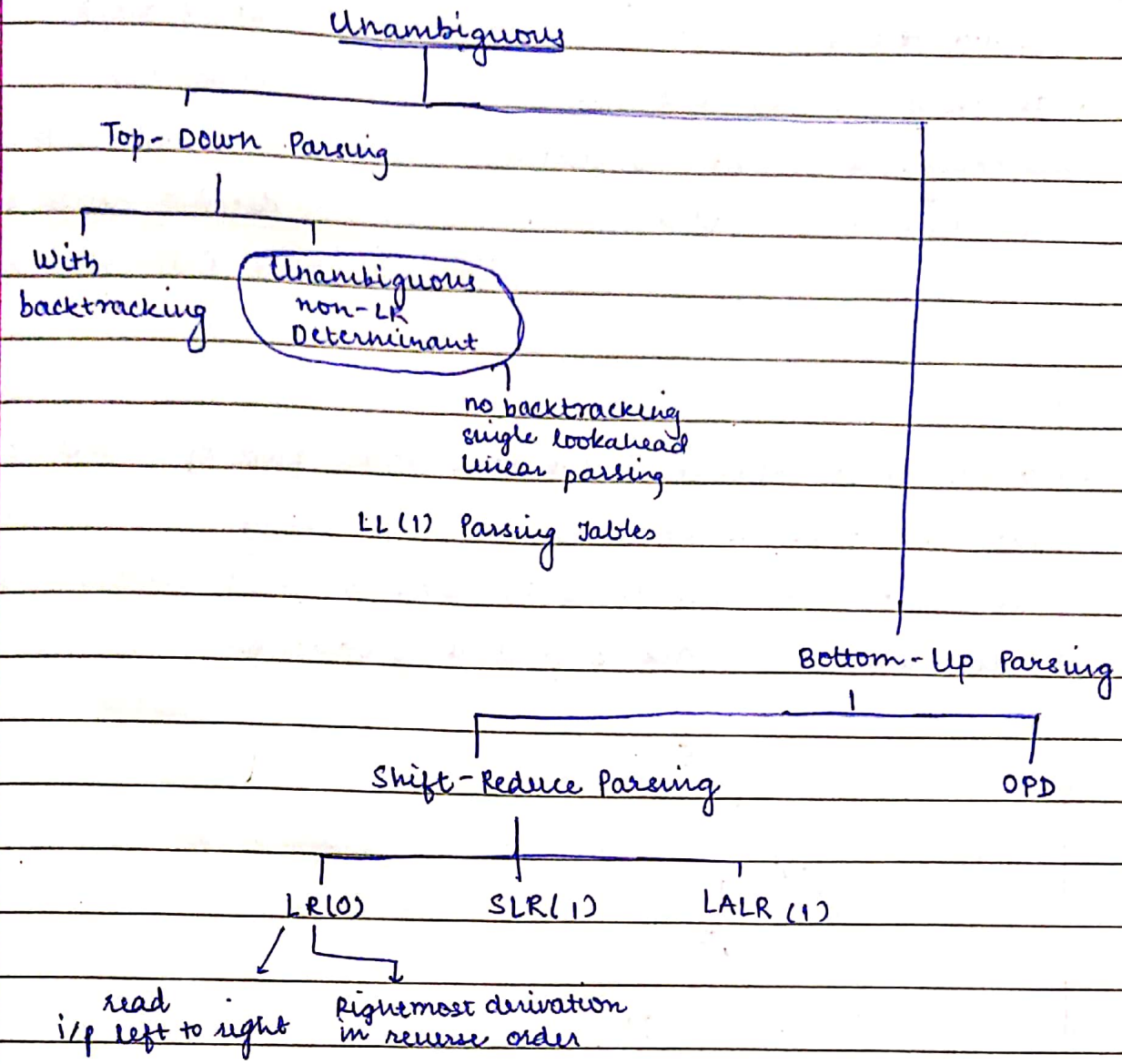
12/3/19

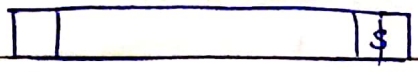


Analogy :-

In-place quicksort

AVL Trees.

→ Input Buffer :



- In LL(1), when we read the 1st symbol input, we match it in Parsing Table under S & find a prod"

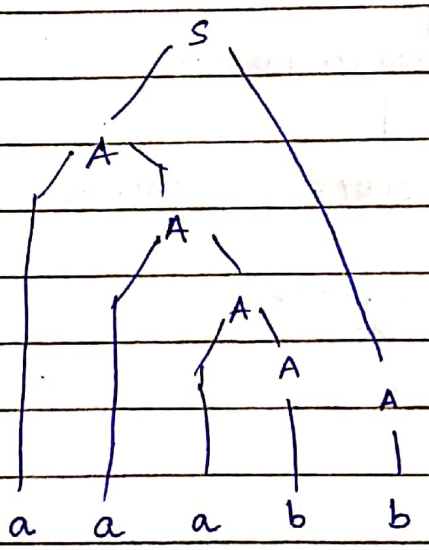
→ If a grammar can't be parsed using an algo then the parsing table would tell us.

→ In LR(0), we only need the unambiguous grammar, no further conversations are needed

• When we reach $ in i/p string & also in stack, that is accepted state. Otherwise, we give error.

→ If a symbol from i/p doesn't match terminals in P.T., give error.

• We use shift-reduce : we take some part of input string and put it in the stack for later processing. We don't have the time to work on that substring but later we come back & find a production



LR(0) Parsing

eg.

$S \longrightarrow AA$ — (P1)

$A \longrightarrow aA$ — (P2)

$A \longrightarrow b$ — (P3)

Augmented Grammar

(1) Add a new ~~at~~ artificial prod^n

$S' \longrightarrow S$ — (P4)

This way, S collapses to S'

② Closure & Goto (Transition) { Algorithmic Operations }

States of the given grammar – Canonical coll^n of states (There is no less & no more no. of productions)
$I_0, I_1, \ldots, I_n$

I_0 state : Compute it

(1) Insert $(S' \rightarrow \cdot S)$ into I_0

(2) For every NT in I_0 which has a dot "." before it, insert ALL productions of that NT with dot "." in the beginning of RHS. — Closure

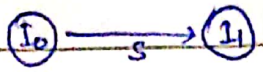
$I_0 \equiv \{ (S' \rightarrow \cdot S)$
$(S \rightarrow \cdot AA)$
$(A \rightarrow \cdot aA)$
$(A \rightarrow \cdot b) \}$

When we have a dot in RHS of a production before NT, then expand that NT.

If we got a dot befor a terminal, don't take that case.

→ So, firstly we have $S' \rightarrow \cdot S$ in I_0
⇒ we expand S & find its prod^n's

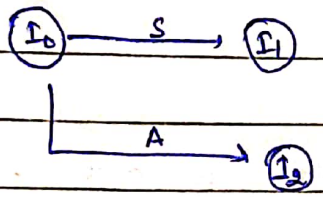
so, we got $S \rightarrow \cdot AA$ in I_0
and so on.



we move the dot over the string and find a prod^n and then we transition to I_1

$I_1 \equiv \{(S' \rightarrow S.)\}$

Move the dot in the RHS to the right of the symbol 'S' in this ~~symbol~~ case.



We also found another production from I_0 to jump a dot over a symbol from productions in I_0

We got transition from I_0 to I_2.

$I_2 \equiv \{ (S \rightarrow A \cdot A)$
$(A \rightarrow \cdot aA)$
$(A \rightarrow \cdot b) \}$

we have a NT 'A' after dot, so we'll expand it & add all its prod^{n}s in the state.

→ A transition = ~~dot jump~~ dot jumps over a symbol.

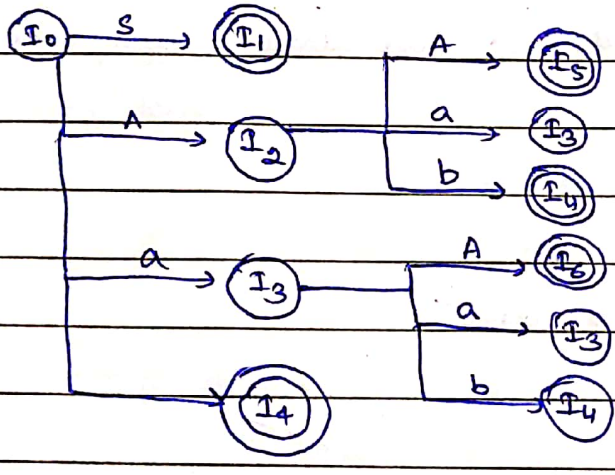
In I_0, we've 'a' symbol & a dot in front of it. So, we can further have transition as:

$I_3 \equiv \{ (A \to a \cdot A)$
$(A \to \cdot aA)$
$(A \to \cdot b) \}$

similarly, for b in I_0,

$I_4 \equiv \{(A \to b \cdot)\}$

don't have any symbol after dot, so it's a final state



Transition from I_2:

$I_5 : \{ (S \to AA \cdot) \}$
↳ no symbol after dot → final state

Transition from I_3:

$I_6 : \{ (A \to aA \cdot) \}$

PAGE NO.:

LR(0) Parsing Table

	Action Part (t)			Goto Part (NT)	
	a	b	$	A	S
I_0	S-I_3	S-I_4		I_2	I_1
I_1			Accept (R-P4)		
I_2	S-I_3	S-I_4		I_5	
I_3	S-I_3	S-I_4		I_6	
I_4	R-P3	R-P3	R-P3		
I_5	R-P1	R-P1	R-P1		
I_6	R-P2	R-P2	R-P2		

- Accept : S has produced the whole string until $ & it can collapse to S'
- There shouldn't be multiple entries in any cell else the grammar is not processable.
- If there is a blank under an interaction, then error of unexpected symbol.

14/03/19

LR(0) items of a given prodⁿ:

$X \rightarrow X_1 X_2 X_3$

are:

$X \rightarrow \cdot X_1 X_2 X_3$

$X \rightarrow X_1 \cdot X_2 X_3$

$X \rightarrow X_1 X_2 \cdot X_3$ and

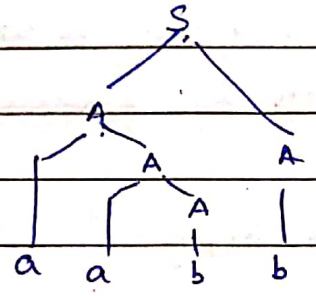
$X \rightarrow X_1 X_2 X_3 \cdot$

(1) In general, a dot '.' in the beginning indicates that the parser has NOT yet utilised (seen) this prodⁿ

- A prod^n is useful to recognize a string

(2) A dot '.' in the middle indicates that the parser has utilized (seen) a part of the i/p but the parser can't apply this prod^n.

(3) $X \rightarrow X_1 X_2 X_3$.
Indicates that the parser has utilized the prod^n & it's ready to apply this prod^n



The terminals which are read first are collapsed last.

- The terminal symbols are reduced to NT.
- I/p string is put in the stack & a part of it is reduced to NT (& popped out of the stack.

→ Actions : Shift & Reduce

→ Shift the terminal symbol in the stack.
Reduce when reached the final state.

→ There will be shifts for non-final states.

For final states, we apply reduce using a prod^n on the basis of what is in the stack in the that state.

eg. a b b $
↑

Look in the parsing table

a b b $
↑ ↑ ↑

I_0 & a : S - I_3

I_3 & b : S - I_4

I_4 & b : R - P3

P3 : A → b

↳① In RHS, we've 1 symbol, so ~~delete~~ delete twice this length, ie, 2 entries from this stack

② After delting entries from stack, apply the prod^n

• Don't shift

→ when we reduce, ptr remains at same symbol

a b b $
↑ ↑

A & I_3 : I_6

b & I_6 : R - P2

P2 : A → aA

↳ delete 4 entries from stack

I_0 & A : I_2

b & I_2 : S - I_4

I_4 & $: R - P3

A & I_2 : I_5

I_5 & $: R - P1

↳ (S → AA)

~~S & $~~ → S & I_0 : I_1

I_1 & $: Accept R - P4

A	
~~I_4~~	R - P3 (0)
~~b~~	R - P3 (0)
I_3	
a	
I_6	

S1	→ Accept
~~I_1~~	R - P4 (0)
~~S~~	R - P4 (0)
~~I_5~~	R - P1 (0)
~~A~~	R - P1 (0)
~~I_4~~	R - P3 (1)
~~b~~	R - P3 (1)
~~I_5~~	R - P1 (0)
~~A~~	R - P1 (0)
~~I_6~~	R - P2 (0)
~~A~~	R - P2 (0)
~~I_4~~	R - P3 (0)
~~b~~	R - P3 (0)
~~I_5~~	R - P2 (0)
~~a~~	R - P2 (0)
I_0	

Grammar (P1) $S \rightarrow AA$

(P2) $A \rightarrow aA$

(P3) $A \rightarrow b$

(P4) $S' \rightarrow S$

LR(0) Parsing

transition b/w I_0 & I_3 & I_4 in terms of a.

	Action a	Action b	Action $	Goto A	Goto S
I_0	SI_3	SI_4		I_2	I_1
I_1	RP_4	$R-P_4$	$R-P_4$		
I_2	$S-I_3$	$S-I_4$		I_5	
I_3	"	"		I_6	
I_4	$R-P_3$	$R-P_3$	$R-P_3$		
I_5	$R-P_1$	$R-P_1$	$R-P_1$		
I_6	$R-P_2$	"	"		

SI_3 : shift I_3

$R-P_4$: Reduce P_4

- Red^n appears in all NT column.

$I_4 : \{(A \rightarrow b.)\}$

I_4
b
I_3
a
I_3
a
I_0 → in start

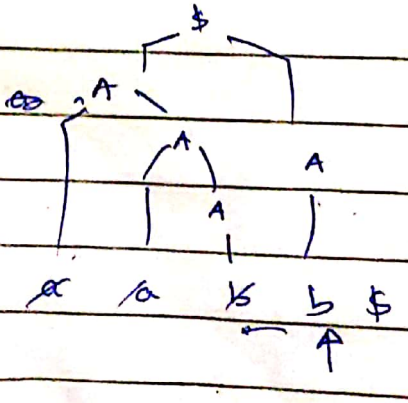
a a b b $

I_0 & a : shift a & I_3

I_3 & a : Shift I_3

I_3 & b : S. I_4

I_4 & b : R $-P_3$: $ 'T' before b gets reduced ('b')



Red^n : current ptr remains same. Red^n happens for 1 or more T already tnt in the stack

b as well as its state will be stored in stack
↳ if Prod^n is $(A \rightarrow b)$

PAGE NO. :

If $A \to aA$: 4 entries will be stored in stack.

Ex: (P1) $E \to E+T$

(P2) $E \to T$

(P3) $T \to id$

(P4) $E' \to E$

$I_0 \equiv \{ (E' \to \cdot E)$

$(E \to \cdot E+T)$

$(E \to \cdot T)$

$(T \to id) \}$

$I_0 \xrightarrow{E} \{ (E' \to E \cdot) \ (P_4) \ I_1 \}$

$(E \to E \cdot + T) \}$

$I_0 \xrightarrow{T} \{ (E \to T \cdot) \}_{P_2} \ I_2$

$I_0 \xrightarrow{id} \{ (T \to id \cdot) \}_{P_3} \ I_3$

$I_1 \xrightarrow{+} \{ (E \to E + \cdot T)$

$(T \to \cdot id) \} \ I_4$

$I_4 \xrightarrow{T} \{ (E \to E + T \cdot) \}_{P_1} \ I_5$

$I_4 \xrightarrow{id} I_3$

* for every final state, there is always a reduction.

only where that prodⁿ follows that terminal symbol.

(Modified to SLR 1)

	+	id	$	E	T
I_0		S-I_3		I_1	I_2
I_1	~~R-P4~~ S-I_4	~~R-P4~~	R-P_4 Accept		
I_2	R-P_2	R-P_2	R-P_2		
I_3	R-P_3	~~R-P3~~	R-P_3		
I_4		S-I_3			I_5
I_5	R-P_1	R-P_1	R-P_1		

final state

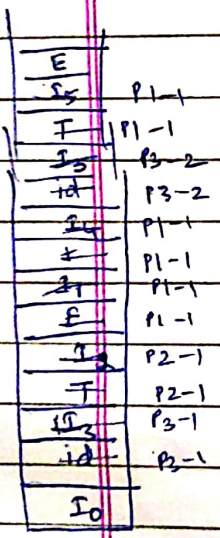
id + id

Input String

id	+	id

↕

← LR(0) Passer



id + id

↑ ↑ ↑

I_0 & id : shift id

I_3 & + : R-P_3 P_3 : T → id

I_2 & + : Reduce P_2

I_1 & + : S-I_4 / R-P_4 ×

id + id

(1) Follow of a prodⁿ is follow of its LHS (↳ NT)

(2) ~~Follow(E) = {$}~~

FOLLOW(E') = {$}

* None of intersection should've 2 entries. If 2 ⇒ grammar is unacceptable.

Take P_4 (because R-P_4 is there) ← when 2 entries

* This red" is only applied when next symbol is what follows this production.

~~SR~~

SLR(1) → Reduce only when $ symbol is next.

↓

won't have R-P_4 for I_1 & + or I_1 & id

what follows P_4 : FOLLOW(E')

↓

SLR(1)

looks at lookahead symbol → subset of FOLLOW of that prod. for which red^n is occurring ← before red^n

DATE : / /

PAGE NO. :

Unacceptable grammar in LR(0) → acceptable in SLR(1)

id + id $

NOW, I_4 & id : $S-I_3$

I_3 & $: R-P_3$:

P_3 : T → id : before applying $R-P_3$. check whether SLR(1) or not

follows (P_3) = follows (T)

= {+, $} → remove P_3 from id

Here, we've $ ✓ can apply P_3

I_5 & $: P_1

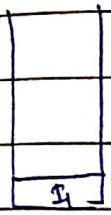
follows (P_1) = follows (E) = {+, $}

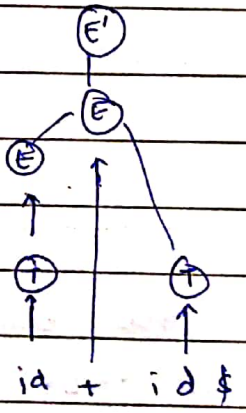
I_0 & E : I_1

↓

only remaining in stack

I_1 & $: accept ✓





Ex. +, -, ÷, *, (,), ↑

Ex. (P1) S → dA DO SLR(1) for this

(P2) S → aB

(P3) A → bA

(P4) A → ε

(P5) B → bB

(P6) B → ε

28-03-19

Top-Down

LL(1)

Requires non-LR Deterministic Unambiguous grammars

(Shift-Reduce Parsers)

Bottom up

* LR(0), SLR(1), CLR(1)

Requires Unambiguous grammar

LR(0), SLR(1)

Use LR(0) Items

LR(0) Items of any production $P_i \rightarrow x_1 x_2 \ldots x_n$ are

$P_i \rightarrow \cdot x_1 x_2 \ldots x_n$

$P_i \rightarrow x_1 \cdot x_2 \ldots x_n$

$P_i \rightarrow x_1 x_2 \ldots x_n \cdot$

In LR(0); Red^n is applied in ALL Terminal symbols

In SLR(1) : Red^n is applied in ONLY those terminal symbols which are in the FOLLOW of that R-P_i

CLR(1) & LALR(1) Parsing techniques use

LR(1) Items ≡ { LR(0) Item + { set of look-ahead Terminal symbols } }

↓ subset of FOLLOWS

LR(0)
↳ Rightmost deriv" in Reverse Order is applied
↓
Left to Right

DATE : / /
PAGE NO. :

Red" is applied in ONLY those terminal symbols which are "Look-Ahead" symbols of the R-P;

→ In SLR(1) & LR(0) : this is same as FOLLOWS

Canonical
CLR (1)

P1 : S → AA
P2 : A → aA
P3 : A → b
P4 : S' → S

$I_0 \equiv$ { (S' → .S) + {$} ; FIRST of this : $
(S → .AA) + {$} ; FIRST of this ≡ FIRST(A) = {a,b}
(A → .aA) + {a,b}
(A → .b) + {a,b}
}

follows

* Look-ahead of any NT in a (I_j) ≡ { FIRST of remaining after NT }

only leaf prod^n's where S occurs : FIRST of remaining.

A → .aA came from S → .AA + {$}
This A ; ↳ first of this: a/b

I'm interested in Lookahead of this A only.

$I_1 \equiv$

(I_0) —S→ {(S' → S.) + {$}} ; when I do transition, LOOK-ahead remains same ⟷ (I_1)

—A→ { (S → A.A) + {$} ; FIRST of this
(A → .aA) + {$}
(A → .b) + {$} } → (I_2)

—a→ { (A → a.A) + {a,b} ; remains same
(A → .aA) , {a,b}
(A → .b , {a,b} } → (I_3)

* SLR(1) : follow (A) = {$, a, b} : will ~~state~~ I have {$} also in I_6.

DATE : / /

PAGE NO. :

I_3 is in the context {a, b} is the next

I_2 - - — — — — {$} — — —

I_1 — — — — — — — —

I_0

b → { (A → b.) , {a, b} } → I_4 - final state

I_2 — A → { (S → AA.) , {$} } → I_5 ⇒ accepting the string ↓ $ should be remaining only (look-ahead)

a → { (A → a.A) , {$}
(A → .aA) , {$}
(A → .b) , {$} } → I_6

b → { (A → b.) , {$} } → I_7 (not same as I_4)

I_3 — A → { (A → aA.) , {a, b} } → I_8

a → I_3 { (A → a.A) , {a, b}
(A → .aA) , {a, b}
(A → .b) , {a, b} } → I_3

b → I_4 { (A → b.) , {a, b} }

I_6 — A → { (A → aA.) , {$} } → I_9

a → I_6 { (A → a.A) , {$}
(A → .aA) , {$}
(A → .b) , {$} }

b → I_7 { (A → b.) , {$} }

30/3/1

DATE : / /
PAGE NO. :

LR(1) Parsing Table

	a	b	$	S	A
I_0	S-I_3	S-I_4		I_1	I_2
I_1			R-P_4		
I_2	S-I_6	S-I_7			I_5
I_3	S-I_3	S-I_4			I_8
I_4	R-P_3	R-P_3	~~R-P_3~~		
I_5			R-P_1		
I_6	S-I_6	S-I_7			I_9
I_7			R-P_3		
I_8	R-P_2	R-P_2			
I_9			R-P_2		

→ I_4 : Final state wrt $A \to b : P_3$ so, R-P_3 Lookahead symbol: {a, b}

Q. In this eg, what states have SAME LR(0) Items BUT DIFFERENT LR(1) Items ?

(1) I_4 & I_7

(2) I_8 & I_9

(3) I_3 & I_6 { In $I_3 \to$ I look for a, b
In $I_6 \to$ " " $ }

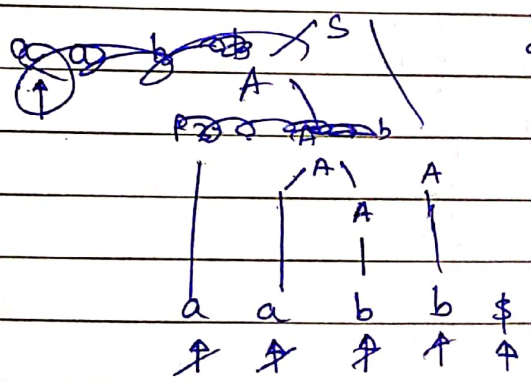
$P_3 : A \to b$

$P_2 : A \to aA$ (got 2, 4 entries cancelled in stack)

Ex. aabb

~~A~~	R-P1
~~I_7~~	R-P3 (1)
~~b~~	R-P3 (1)
~~I_2~~	R-P1
~~A~~	R-P1
~~I_8~~	R-P2(1)
~~A~~	R-P2(1)
~~I_8~~	R-P2
~~A~~	R-P2
~~I_4~~	R-P3
~~b~~	R-P3
~~I_3~~	R-P2
~~a~~	R-P2
~~I_3~~	R-P2(1)
~~a~~	R-P2(1)
I_0	

I_6 & b : R-P_2
I_3 & A : I_8
I_8 & b : R-P_2
I_0 & A : I_2
I_2 & b : S-I_7
I_7 & $: R-P_3



I_0, a : S-I_3
I_3 & a : S-I_3
I_3 & b : S-I_4
I_4 & b : RP3 → Reduced b to A
I_3 & A : I_8

PAGE NO.:

A & I_2 : I_5

I_5 & $: R-P1 → 4 entries removed

S & I_0 : I_1

I_1 & $: R-P4 : S' → S

Accept ← S'	
~~I_1~~	R-P4
S	R-P4
~~I_5~~	R-P1

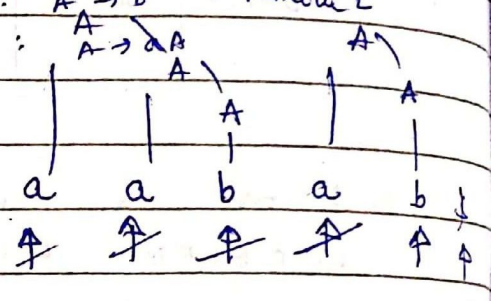
1) Red^n happens to previous symbol (s).

S → AA : check 2 A's are reduced to S. (Remove 4 stack entries)

Ex. aabab

P3 : A → b ↑ Remove

P2 : A → aA



S'	← Accept
~~I_1~~	R-P4-0
~~S~~	R-P4-0
~~I_5~~	R-P1-0
~~A~~	R-P1-0
~~I_3~~	R-P2-2
~~A~~	R-P2-2
~~I_7~~	R-P3-1
~~b~~	R-P3-1
~~I_6~~	R-P2-1
~~a~~	R-P2-1
~~I_2~~	R-P1-0
~~A~~	R-P1-0
~~I_8~~	R-P2-1
A	R-P2-1
~~I_6~~	R-P2-0
~~A~~	R-P2-0
~~I_4~~	R-P3-0
~~b~~	R-P3-0
~~I_3~~	R-P2-0
~~a~~	R-P2-0
~~I_3~~	R-P2-1
~~a~~	R-P2-1
I_0	

a & I_0 : S-I_3

I_3 & a : S-I_3

I_3 & b : S-I_4

I_4 & a : R-P3

~~$ A~~ & I_3 : I_8

I_8 & a : R-P2

I_3 & A : I_8

I_6 & a : R-P2

A & I_0 : I_2

I_2 & a : SI_6

I_6 & b : SI_7

~~I_7 & $: R-P3~~

A & I_6 : I_9

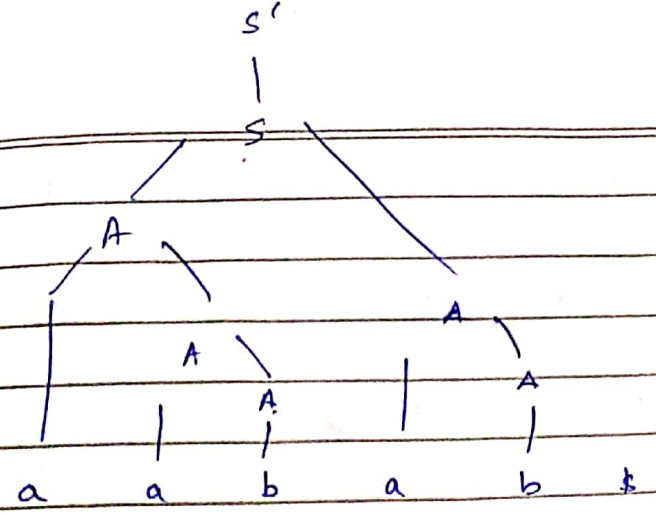
I_9 & $: R-P2

A & I_2 : I_5

I_5 & $: R-P1

S & I_0 : I_1

I_1 & $: R-P4



Ex-2.

P1 : S → ABd

P2 : A → Bb

P3 : A → c

P4 : B → d

P5 : B → eB

P6 : S' → S

I_0 = { (S' → · S) + {$}
(S → · ABd) + {$}
(A → · Bb) + {d, e}
(A → · c) + {d, e}
(B → · d) + {b}
(B → · eB) + {b} }

I_0 —S→ { (S' → S ·) + {$} } — I_1

I_0 —A→ { (S' → A · Bd) + {$}
(B → · d) + {d}
(B → · eB) + {d} } — I_2

I_0 —B→ { (A → B · b) + {d, e} } — I_3

I_0 —c→ { (A → c ·) + {d, e} } — I_4

I_0 —d→ { (A → d ·) + {b} } — I_5

I_0 —e→ { (B → e · B) + {b}
(B → · eB) + {b}, (B → · d) + {b} } — I_6

I_2 — B → { (S → AB·d) + {$} } — I_7

d → { (B → d·) + {d} } — I_8

e → { (B → e·B) + {d}
(B → ·d) + {d}
(B → ·eB) + {d} } — I_9

I_3 — b → { (A → Bb·) + {d, e} } — I_{10}

I_6 — B → { (B → eB·) + {b} } — I_{11}

d → { (B → d·) + {b} } — I_8

e → I_6 { (B → e·B) + {b}
(B → ·d) + {b}
(B → ·eB) + {b} }

I_7 — d → { (S → ABd·) + {$} } — I_{12}

I_9 — B → { (B → eB·) + {d} } — I_{13}

d → I_8 { (B → d·) + {d} }

e → I_9 { (B → e·B) + {d}
(B → ·d) + {d}
(B → ·eB) + {d} }