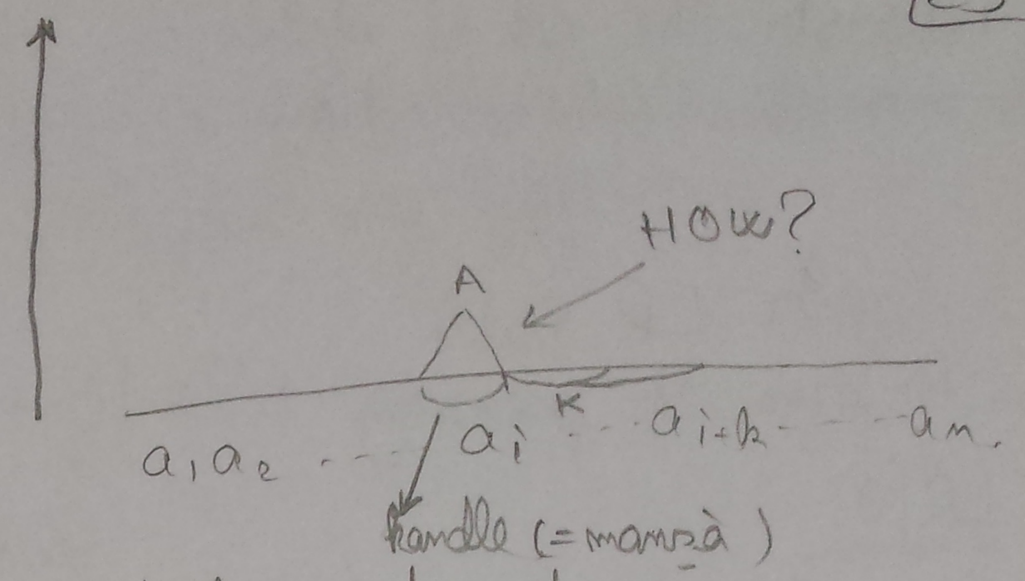
LR(K) Parser

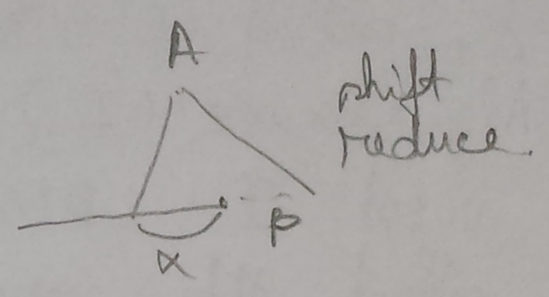
L - left - seq is read from left.
R - right - rightmost derivation.
$K \in \mathbb{N}$ - length of the prediction.



Def 1. In a cfg G any prefix of $\alpha\beta$ is called live prefix if

$S \overset{*}{\underset{r}{\Rightarrow}} \alpha A u \underset{r}{\Rightarrow} \alpha\beta\gamma u$

Def 2 Am. LR(k) item is denoted $[\underbrace{A \to \alpha . \beta}_{\text{kernel}}, \underbrace{u}_{\text{prediction}}]$.



$A \to \alpha\beta \in \mathcal{P}$.
$u \in \Sigma^k$
LR(K) item valid for live prefix $\gamma\alpha$.
$S \overset{*}{\underset{r}{\Rightarrow}} \gamma A w \underset{r}{\Rightarrow} \gamma\alpha\beta w$, $u \in FIRST_k(w)$.

$[A \to \alpha\beta., u]$.
LR(K) principle: the action is uniquely determined:
- state.
- the current symbol.
- prediction.

$[A \to \alpha . B\beta, u]$ valid for live prefix $\gamma\alpha$ if.
$S \overset{*}{\underset{r}{\Rightarrow}} \gamma A w \underset{r}{\Rightarrow} \gamma\alpha B\beta w \overset{*}{\underset{r}{\Rightarrow}} \gamma\alpha\delta w'$ $\quad B \to \delta$

$\Rightarrow [B \to .\delta, u]$ also valid for live prefix $\gamma\alpha$

LR state is the set of all LR items corresponding to the same live prefix

All. LR(K) parsers
$G = (N, \Sigma, P, S) \to$ enhanced gr. $G' = (N \cup \{S'\}, \Sigma, P \cup \{S' \to S\}, S')$

LR(0)
SLR → Simple LR(1).
LR(1) ✓
↳ LALR look ahead LR(1) optimized and used.

1. Constr. the set of states
   → canonical collection. (P
   = Figure
   goto
2 Constr LR table
3 Perform the parsing. (Config + moves def)

LR(0)

LR(0) item $[A \to \alpha . \beta]$

→ how to construct a state?

- Closure : $\mathcal{P}(\mathcal{E}_0) \to \mathcal{P}(\mathcal{E}_0)$

  $\mathcal{E}_0$ = set of LR(0) items
- Alg Closure (I)

  input : I : LR(0) items

  output : state : corresp to I.

  C = {I};

  Repeat

  for $\forall [A \to \alpha . B\beta] \in C$ do

  for $\forall B \to \delta \in P$ do

  if $[B \to . \delta] \notin C$ then

  C.append($[B \to \delta]$)

  ← until (C does not change)

  closure = C
- goto : $P(\mathcal{E}_0) \times (N \cup \Sigma) \to \mathcal{P}(\mathcal{E}_0)$

  goto(P, X) = closure ({$[A \to \alpha X . \beta] | [A \to \alpha . X \beta] \in P$}).

Alg ConCol 0 (0 from LR(0)

input : G'

output : $\mathcal{C} = \{P_0, P_1, \dots\}$ (all states)

P_0 = closure ({$[S' \to . S]$})

$\mathcal{C} = \{P_0\}$

Repeat :

for $\forall P \in \mathcal{C}$ do

for $\forall X \in N \cup \Sigma$ do

T = goto (P, X)

if $T \neq \emptyset$ and $T \in \mathcal{C}$ then : $\mathcal{C}$.append (T).

until (C is not changed)

$G = (\{S, A, B\}, \{a, c\}, P, S)$

$P: S \to AB$

$A \to aA \mid aB$

$B \to c$

$D_0 = \text{closure}(\{[S' \to .S]\}) = \{[S' \to .S], [S \to .AB], [A \to .aA], [A \to .aB]\}$

$D_1 = \text{goto}(D_0, S) = \text{closure}(\{[S' \to S.]\}) = \{[S' \to S.]\}$

$D_2 = \text{goto}(D_0, A) = \text{closure}(\{[S' \to A.B]\}) = \{[S \to A.B], [B \to .c]\}$

$\text{goto}(D_0, B) = \emptyset$.

$D_3 = \text{goto}(D_0, a) = \text{closure}(\{[A \to a.A], [A \to a.B]\}) = \{[A \to a.A], [A \to a.B], [A \to .aA], [A \to .aB], [B \to .c]\}$

$\text{goto}(D_0, C) = \emptyset$

$\text{goto}(D_1, X) = \emptyset \quad \forall X \in N \cup \Sigma$.

$D_4 = \text{goto}(D_2, B) = \text{closure}(\{[S \to AB.]\}) = \{[S \to AB.]\}$

$D_5 = \text{goto}(D_2, C) = \text{closure}(\{[B \to c.]\}) = \{[B \to c.]\}$

$D_6 = \text{goto}(D_3, A) = \{[A \to aA.]\}$

$D_7 = \text{goto}(D_3, B) = \{[A \to aB.]\}$.

$\text{goto}(AB, a) = \text{closure}(\{[A \to a.A], [A \to a.B]\}) = D_3$.

$\text{goto}(D_3, c) = \{[B \to c.]\} = D_5$.

$\text{goto}(D_4, X) = \emptyset$.

$\text{goto}(D_5, X) = \emptyset$.

$\text{goto}(D_6, X) = \emptyset$.

$\text{goto}(D_7, X) = \emptyset$.

$\mathcal{C} = \{D_0, D_1, \ldots, D_7\}$.

2. Construct LR(0) table

	Action	GOTO
		$N \cup \Sigma$
D_0		
D_1		
⋮		
D_m		

Rules:

1). if $[A \to \alpha.\beta] \in D_i$, $\beta \neq \varepsilon$ then
action (D_i) = shift.

2). if $[A \to \alpha\beta.] \in D_i$, $A \neq S'$ then
action (D_i) = reduce (m).
(m) $A \to \alpha\beta \in P$

3) if $[S' \to S.] \in D_i$ then
action (D_i) = acc

4) if $\text{goto}(D_i, X) = D_j$ then
$\text{goto}(D_i, X) = D_j$.

5. otherwise Error

6. the initial state [S' → .. S]

	Action	GOTO S	A	B	a	c
D_0	shift	D_1	D_2		D_3	
D_1	acc					
D_2	shift			D_4		D_5
D_3	shift		D_6	D_7	D_3	D_5
D_4	reduce (1)					
D_5	reduce (4)					
D_6	reduce (2)					
D_7	reduce (3)					

Remarks:

1) $S' \to S$ eliminates conflict between accept and reduce.

2) when action == acc OR action = reduce → goto void.

3) A grammar is LR(0) if the LR(0) table does not contain conflicts.

4) conflicts: shift - reduce $D_i \ni [A \to \alpha . \beta]$, $\beta \neq \emptyset$
$\ni [B \to \delta .]$

reduce - reduce $D_i \ni [A \to \alpha .]$ and $[B \to \beta .]$

8.12.2015 LR(0) Parser.

3) config + moves

(α, β, π)

α: working set, β: input stack, π: output stack (prod)

initial config: $(\$ D_0, w\$, \varepsilon)$

final config: $(\$ D_{acc}, \$, \pi)$

Moves

1) Shift: $(\$ D_0 \dots X_i D_i, aw\$, \pi) \vdash$ if action(D_i) = shift
$\vdash (\$ D_0 \dots X_i D_i a D_j, w\$, \pi)$ goto$(D_i, a) = D_j$

2) Reduce $(\$ D_0 \dots X_i D_i \underbrace{X_{i+1} D_{i+1} \dots X_{i+m} D_{i+m}}, w\$, \pi) \vdash$
if action(D_{i+m}) = reduce n
(n) $A \to X_{i+1} \dots X_{i+m}$.
$\vdash (\$ D_0 \dots X_i D_i A D_j, w, n\pi)$
goto$(D_i, A) = D_j$

3) Accept $(\$ D_i, \$, \pi) \vdash$ acc
if action(D_i) = acc

4) otherwise: ERROR