5. otherwise Error

6. the initial state [S' → .S]

| | Action | S | A | B (GOTO) | a | c |
|---|---|---|---|---|---|---|
| $D_0$ | shift | $D_1$ | $D_2$ | | $D_3$ | |
| $D_1$ | acc | | | | | |
| $D_2$ | shift | | | $D_4$ | | $D_5$ |
| $D_3$ | shift | $D_6$ | $D_8$ | $D_7$ | $D_3$ | $D_5$ |
| $D_4$ | 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

$(\alpha, \beta, \pi)$

$\alpha$ ↓ working set; $\beta$ ↑ input stack; $\pi$ ↑ output stack (prod)

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

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

Moves

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

2) Reduce $(\$ D_0 \dots X_i D_i X_{i+1} D_{i+1} \dots X_{i+m} D_{i+m}, w\$, \pi) \vdash$ (SLR)

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$(D_i, A) = D_j$

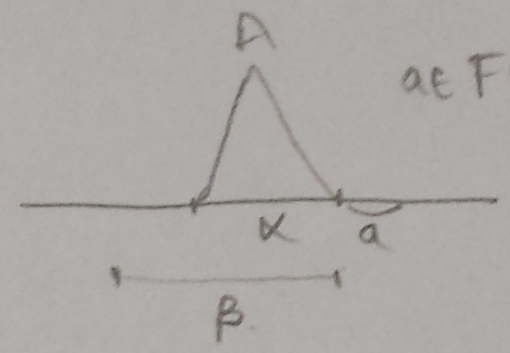
3) Accept $(\$ D_i, \$, \pi) \vdash$ acc

if action$(D_i)$ = acc

4) Otherwise: ERROR

Conflicts:
- shift-reduce
- reduce-reduce ⇒ {[A→x.], [B→δ.]}

SLR (Simple. LR(1))
- use. LR(0) can. collect.
- when performing reduce look at prediction of length 1.
- construct can. collection. of states. → LR(0) ✓
- construct SLR table.
- dif config + moves




2) SLR

| states | action Σ ∪ {$} | goto N ∪ {S} |
|---|---|---|
| | shift + goto | |
| | reduce (m, goto) | |
| | accept (m, goto) | |

Rules for SLR table
1) if $p_i \ni [A \to x.\beta]$, $\beta \neq \varepsilon$ and $goto(p_i, u) = p_j$ ⇒ action$(p_i, u)$ = shift $p_j$
2) if $[A \to x\beta.] \in p_i$, $A \neq S'$, $b \in FOLLOW(A)$, $(m)\ A \to x\beta \in P$ ⇒ action$(p_i, b)$ = reduce (m)
3) if $[S' \to S.] \in p_i$ then action$(p_i, \$)$ = accept
4) goto$(p_i, A) = p_j$ if goto$(p_i, A) = p_j$
5) $[S' \to .S]$ - initial state c) Otherwise error.

SLR config - LR(0) config.
1) if action$(p_i, a)$ = shift.
2) if action$(p_{i+m}, b)$ = reduce m.

A gr is SLR if SLR table does not contain conflicts.

## LR(1) Parser

$[A \to x.\beta, u]$
Kernel; prod. |u| = 1

1) Create. can collect of states.
2) Constr. LR(1) table
3) Def config + moves

1) Alg can. Col LR(1).
input G'
output $\mathcal{C}_1 = \{p_0, \dots p_i\}$.
$p_0$ = closure($\{[S' \to .S, \$]\}$)
$\mathcal{C}_1 = \{p_0\}$.
Repeat.
  for $\forall p \in \mathcal{C}_1$ do.
    for $\forall X \in N \cup \Sigma$ do.
      $T = goto1(p, X)$
      if $T \neq \emptyset$ and $T \notin \mathcal{C}_1$ then $\mathcal{C}_1$.append(T)
until $\mathcal{C}_1$ is unchanged

Alg goto: $\mathcal{P}(\mathcal{E}_1) \times (N \cup \Sigma) \to \mathcal{P}(\mathcal{E}_1)$

goto(D, X) = closure($\{[A \to \alpha X . \beta, u] \mid [A \to \alpha . X\beta, u] \in D\}$)

Alg closure
input G', FIRST, I
output $C_1$

$C_1 = \{I\}$
Repeat
  for $\forall [A \to \alpha . B\beta, u] \in C_1$ do
    for $\forall B \to \delta \in P$ do
      for $\forall v \in$ FIRST$(\beta u)$ do
        $C_1$.append($[B \to .\delta, v]$)
until $C_1$ is not changed.

$[A \to \alpha . B\beta, u]$ - valid for $\gamma\alpha$, $u \in$ FIRST$(w)$.

$S \overset{*}{\underset{R}{\Rightarrow}} \gamma A w \underset{R}{\Rightarrow} \gamma\alpha B\beta w \overset{*}{\underset{R}{\Rightarrow}} \gamma\alpha B w' w \Rightarrow$

$B \to \delta$

$\Rightarrow \gamma\alpha\delta w' w$.

$[B \to .\delta, v]$ $v \in$ FIRST$(\beta u)$

$v \in$ FIRST$(w'w)$.

FIRST$(w'w)$ = F$(w')$ ⊕ F$(w)$ =

= FIRST$(\beta)$ ⊕ $u$ = F$(\beta u)$.

Example. $S' \to S$.
1) $S \to AA$
   $A \to aA$
   $A \to b$

$D_0$ = closure($\{[S' \to .S, \$]\}$) = $\{[S' \to .S, \$], [S \to .AA, \$],$
$[A \to .aA, a], [A \to .aA, b], [A \to .b, a], [A \to .b, b]\}$ First(A) = {a, b}