Course 9

SLR Parser

- SLR = Simple LR

- Remark:

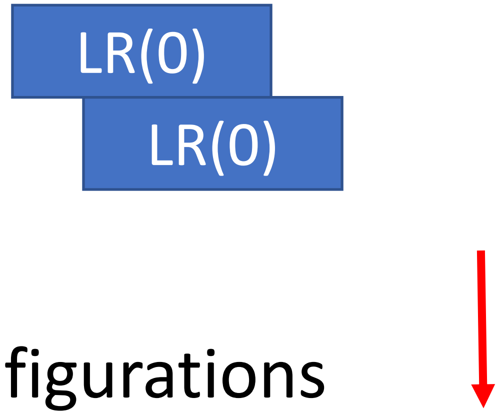
  LR(0) – lots of conflicts – solved if considering prediction

=>

1. LR(0) canonical collection of states– prediction of length 0
2. Table and parsing sequence – prediction of length 1

LR(k) Parsing:
LR(0), SLR, LR(1), LALR

- define item
- Construct set of states
- Construct table
- Parse sequence based on moves between configurations



Construct SLR table

Remarks:

1. Prediction = next symbol from input sequence => FOLLOW

   - see LL(1)

2. Structure – LR(k):
   - Lines - states
   - action + goto

action – a column for each prediction ∈ **Σ**

goto – a column for each symbol X ∈N∪**Σ**

Remark (LR(0) table):

- *if* s is accept state *then* goto(s, X) = ∅, ∀X ∈ N ∪ Σ.
- *If* in state ***s*** action is reduce *then* goto(s, X) = ∅, ∀X ∈ N ∪ Σ.

SLR table

	Action		**GOTO**	
	a_1 …	a_n	B_1 …	B_m
s_0				
s_1				
…				
s_k				

$a_1,\dots,a_n \in \Sigma$
$B_1,\dots,B_m \in N$
$s_0,\dots,s_k$ - states

Rules for SLR table

1. If $[A \to \alpha.\beta] \in s_i$ and goto(s_i,a) = s_j then **action(s_i,a)=shift s_j**
2. if $[A \to \beta.] \in s_i$ and $A \neq S'$ then **action(s_i,u)=reduce l**, where l – number of production $A \to \beta$, $\forall u \in$ FOLLOW(A)
3. if $[S' \to S.] \in s_i$ then **action(s_i,\$)=acc**
4. if goto(s_i, X) = s_j then **goto(s_i, X) = s_j** , $\forall X \in N$
5. otherwise **error**

Remarks

1. Similarity with LR(0)

2. A grammar is SLR if the SLR table does not contain conflicts

Parsing sequences

- INPUT:
  - Grammar G' = (NU{S'}, $\boldsymbol{\Sigma}$, P U {S'->S},S')
  - SLR table
  - Input sequence w =$a_1 \ldots a_n$
- OUTPUT:

  if (w ∈L(G)) *then* **string of productions**

  else **error & location of error**

SLR = LR(0) configurations

$$(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\pi})$$

where:

- α = working stack
- β = input stack
- π = output (result)

Initial configuration:
($\$s_0$,w\$,ε)

Final configuration:
($\$s_{acc}$, \$, π)

Moves

head(β) = prediction

1. **Shift**

if action(s_m,a_i)= shift s_j **then**

$$(\$s_0x_1 \ldots x_ms_m,a_i \ldots a_n\$, \pi) \vdash (\$s_0x_1 \ldots x_ms_ma_is_j,a_{i+1} \ldots a_n\$, \pi)$$

2. **Reduce**

if action(s_m,a_i) = reduce t AND (t) $A \rightarrow x_{m-p+1} \ldots x_m$ AND goto(s_{m-p},A) = s_j

then

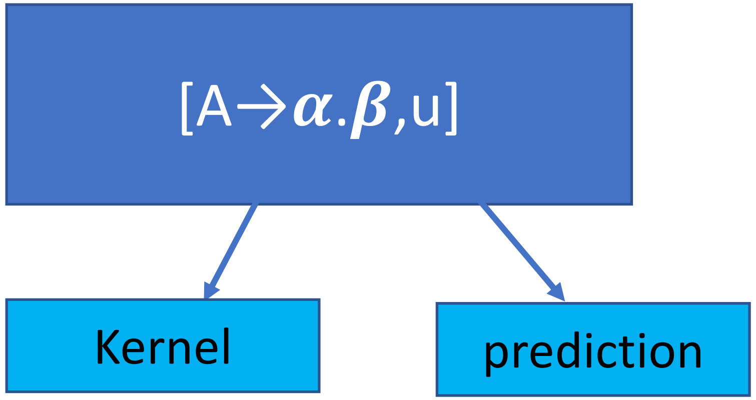
$$(\$s_0 \ldots x_ms_m,a_i \ldots a_n\$, \pi) \vdash (\$s_0 \ldots x_{m-p}s_{m-p}As_j,a_i \ldots a_n\$,t\,\pi)$$

3. **Accept**

if action(s_m,\$) = accept **then** **(**$\$s_m,\$, \pi$)=acc

4. **Error** - otherwise

LR(1) Parser



1. Define item
2. Construct set of states
3. Construct table
4. Parse sequence based on moves between configurations

Construct LR(1) set of states

- Alg *ColCan_LR1*
- Function *goto1*
- Alg *Closure1*

Algorithm *ColCan_LR(1)*

INPUT: G'- gramatica îmbogăţită
OUTPUT: C - colecţia canonică de stări

$\mathcal{C}_1 := \emptyset;$
$s_0 := closure(\{[S' \to .S, \$]\})$
$\mathcal{C}_1 := \mathcal{C}_1 \cup \{s_0\};$
repeat
    for $\forall s \in \mathcal{C}_1$ **do**
        for $\forall X \in N \cup \Sigma$ **do**
            T:=goto(s, X);
            if $T \neq \emptyset$ and $T \notin \mathcal{C}_1$ **then**
                $\mathcal{C}_1 = \mathcal{C}_1 \cup T$
            end if
        end for
    end for
until $\mathcal{C}_1$ nu se mai modifică

Function *goto1*

goto1 : $P(\mathcal{E}_0) \times (N \cup \Sigma) \rightarrow P(\mathcal{E}_0)$

where $\mathcal{E}_0$ = set ofLR(0) items

goto1(s, X) = closure($\{[A \rightarrow \alpha X.\beta, u] \mid [A \rightarrow \alpha.X\beta, u] \in s\}$)

Algorithm *Closure*

- [A → α.Bβ,u] valid for live prefix γα =>

$$S \overset{*}{\Rightarrow}_{dr} \gamma A w \Rightarrow_{dr} \gamma \alpha B \beta w$$

$$u = FIRST_k(w)$$

- [B → .δ, *smth*]∈P => $S \overset{*}{\Rightarrow} \gamma A w \Rightarrow_{dr} \gamma \alpha B \beta w \Rightarrow_{dr} \gamma \alpha \delta \beta w.$

=> [B → .δ,b] valid for live prefix γα,

∀b ∊ FIRST(βu)

Algorithm *Closure1*

INPUT: I-element de analiză; G'- gramatica îmbogăţită;
$FIRST(X), \forall X \in N \cup \Sigma$;
OUTPUT: C_1 = closure(I);
$C_1 := \{I\}$;
repeat
  for $\forall [A \to \alpha.B\beta, a] \in C_1$ **do**
    for $\forall B \to \gamma \in P$ **do**
      for $\forall b \in FIRST(\beta a)$ **do**
        if $[B \to .\gamma, b] \notin C_1$ **then**
          $C_1 = C_1 \cup [B \to .\gamma, b]$
        end if
      end for
    end for
  end for
until C_1 nu se mai modifică

Construct LR(1) table

- Structure – SLR
- Rules:

1. if $[A \to \alpha.\beta,u] \in s_i$ and goto(s_i,a) = s_j then **action(s_i,a)=shift s_j**
2. if $[A \to \beta.,u] \in s_i$ and $A \neq S'$ then **action(s_i,u)=reduce l**, where l – number of production $A \to \beta$
3. if $[S' \to S.,\$] \in s_i$ then **action(s_i,\$)=acc**
4. if goto(s_i, X) = s_j then **goto(s_i, X) = s_j** , $\forall X \in N$
5. otherwise = **error**

Remarks

1. A grammar is LR(1) if the LR(1) table does not contain conflicts

2. Number of states – significantly increase

4. Define configurations and moves

- INPUT:
  - Grammar G' = (NU{S'}, $\boldsymbol{\Sigma}$, P U {S'->S},S')
  - LR(1) table
  - Input sequence w =$a_1 \ldots a_n$
- OUTPUT:

  if (w ∈L(G)) *then* **string of productions**

  else **error & location of error**

LR(1) configurations

$$(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\pi})$$

where:

- α = working stack
- β = input stack
- π = output (result)

Initial configuration:
($\$s_0$,w$,ε)

Final configuration:
($\$s_{acc}$, $, π)

Moves

head(β) = prediction

1. **Shift**

if action(s_m,a_i)= shift s_j **then**

$$(\$s_0 x_1 \ldots x_m s_m, a_i \ldots a_n\$, \pi) \vdash (\$s_0 x_1 \ldots x_m s_m a_i s_j, a_{i+1} \ldots a_n\$, \pi)$$

2. **Reduce**

if action(s_m,a_i) = reduce t AND (t) $A \rightarrow x_{m-p+1} \ldots x_m$ AND goto(s_{m-p},A) = s_j

then

$$(\$s_0 \ldots x_m s_m, a_i \ldots a_n\$, \pi) \vdash (\$s_0 \ldots x_{m-p} s_{m-p} A s_j, a_i \ldots a_n\$, t\,\pi)$$

3. **Accept**

if action(s_m,\$) = accept **then (**$\$s_m$,\$, π)=acc

4. **Error** - otherwise

LALR Parser

- LALR = Look Ahead LR(1)

- why?

LALR principle

- Merge states with the same kernel, conserving all predictions, if **no conflict** is created

$[A \rightarrow \alpha.\beta,u] \in s_i$

$\Rightarrow [A \rightarrow \alpha.\beta,u/v] \in s_{i,j}$

$[A \rightarrow \alpha.\beta,v] \in s_j$

LALR Parsing

- Same as LR(1)
- Number of LALR states = number of SLR / LR(0) states

- How? - LR(1) states

LR(k) Parsers

- LR(0):
  - Items ignore prediction
  - Reduce can be applied only in singular states (contain one item)
  - Lot of conflicts
- SLR:
  - Use same items as LR(0)
  - When reduce consider prediction
  - Eliminate several LR(0) conflicts (not all)
- LR(1):
  - Performant algorithm for set of states
  - Generate few conflicts
  - Generate lot of states
- LALR:
  - Merge LR(1) states ccorresponding to same kernel
  - Most used algorithm (most performant)

Parsing - recap

	Descendent	**Ascendent**
Recursive	Descendent recursive parser	Ascendent recursive parser
Linear	LL(1)	LR(0), SLR, LR(1), LALR

Parsing - recap