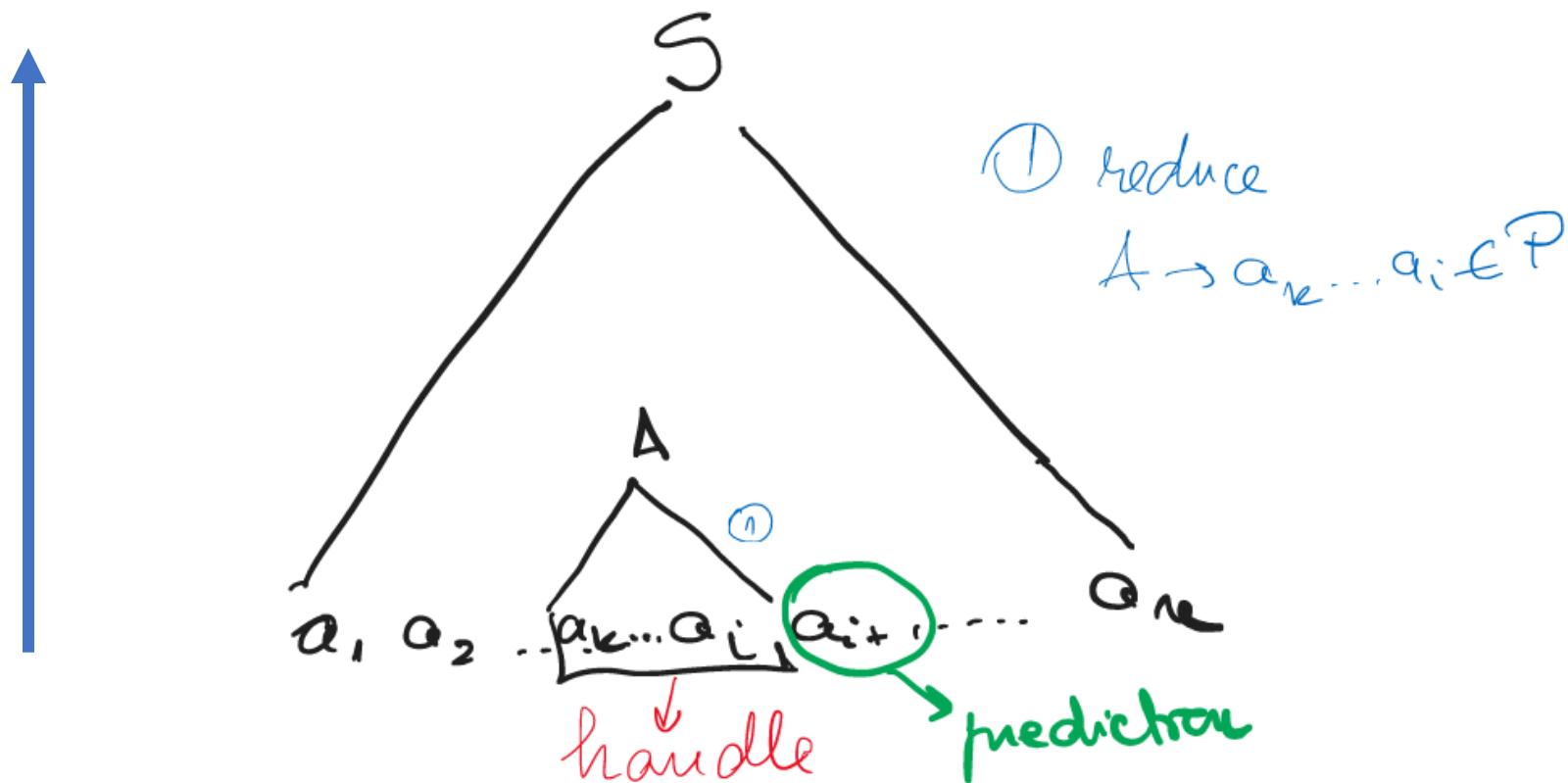


# Course 9

LR(k) Parsing (cont.)

LR(K)



LR( $k$ ) parsing:

LR(0), SLR, LR(1), LALR

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

Executed 1 time

# Algorithm *ColCan\_LR(0)*

---

**INPUT:**  $G'$ - gramatica îmbogățită

**OUTPUT:**  $C$  - colecția canonică de stări

$\mathcal{C} := \emptyset;$

$s_0 := closure(\{[S' \rightarrow .S]\})$  // state corresponding to prod. of  $S'$  = initial state

$\mathcal{C} := \mathcal{C} \cup \{s_0\}$ ; //initialize collection with  $s_0$

**repeat**

**for**  $\forall s \in \mathcal{C}$  **do**

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

**if**  $goto(s, X) \neq \emptyset$  and  $goto(s, X) \notin \mathcal{C}$  **then**

$\mathcal{C} = \mathcal{C} \cup goto(s, X)$  //add new state

**end if**

**end for**

**end for**

**until**  $\mathcal{C}$  nu se mai modifică

---

# Algorithm *Closure*

I = LR(0) item of the form [A-> $\alpha.\beta$ ]

---

**INPUT:** I-element de analiză; G' - gramatica îmbogățită

**OUTPUT:** C = closure(I);

$C := \{I\}$ ; //initialize Closure with the LR(0) item

**repeat**

**for**  $\forall[A \rightarrow \alpha.B\beta] \in C$  **do** //search productions with dot in front of nonterminal

**for**  $\forall B \rightarrow \gamma \in P$  **do** //search productions of that nonterminal

**if**  $[B \rightarrow .\gamma] \notin C$  **then**

$C = C \cup [B \rightarrow .\gamma]$  //adds item formed from production with dot in  
                //front of right hand side of the production

**end if**

**end for**

**end for**

**until** C nu se mai modifică

---

# Function *goto*

$\text{goto} : P(\mathcal{E}_0) \times (N \cup \Sigma) \rightarrow P(\mathcal{E}_0)$  //creates new states  
where  $\mathcal{E}_0$  = set of LR(0) items

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

goto( $s, X$ ): in state **s**, search LR(0) item that has dot in front of symbol **X**.  
Move the dot after symbol **X** and call closure for this new item.

# Remarks

1) Initial state of parser = state containing  $[S' \rightarrow .S]$

2) No shift from accept state:

*if s is accept state then  $\text{goto}(s, X) = \emptyset, \forall X \in N \cup \Sigma$ .*

3) *If in state s action is reduce then  $\text{goto}(s, X) = \emptyset, \forall X \in N \cup \Sigma$ .*

4) Argument G': Let  $G = (\{S\}, \{a, b, c\}, \{S \rightarrow aSbS, S \rightarrow c\}, S)$

states  $[S \rightarrow aSbS.]$  and  $[S \rightarrow c.]$  – accept / reduce ?

# SLR Parser

- SLR = Simple LR

Prediction = next symbols on  
input sequence

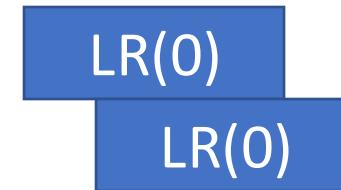
- Remark:
  - LR(0) – lots of conflicts – solved if considering prediction

=>

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

# SLR Parsing:

- 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  $\in \Sigma$

goto – a column for each symbol  $X \in N \cup \Sigma$

Optimize table structure:  
merge *action* and *goto*  
columns for  $\Sigma$

**Remark (LR(0) table):**

- if  $s$  is accept state then  $\text{goto}(s, X) = \emptyset, \forall X \in N \cup \Sigma$ .
- If in state  $s$  action is reduce then  $\text{goto}(s, X) = \emptyset, \forall X \in N \cup \Sigma$ .

# SLR table

And goto

	Action		GOTO			
	$a_1$	$\dots$	$a_n$	$B_1$	$\dots$	$B_m$
$s_0$						
$s_1$						
$\dots$						
$s_k$						

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

# Rules for SLR table

1. If  $[A \rightarrow \alpha.\beta] \in s_i$  and  $\text{goto}(s_i, a) = s_j$  then **action( $s_i, a$ )=shift  $s_j$**   
*// dot is not at the end*
2. if  $[A \rightarrow \beta.] \in s_i$  and  $A \neq S'$  then **action( $s_i, u$ )=reduce l**, where l – number of production  $A \rightarrow \beta$ ,  $\forall u \in \text{FOLLOW}(A)$   
*//dot is at the end, but not for  $S'$*
3. if  $[S' \rightarrow S.] \in s_i$  then **action( $s_i, \$$ )=acc**  
*// dot is at the end, prod. of  $S'$*
4. if  $\text{goto}(s_i, X) = s_j$  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 (more than one value in a cell)

# Parsing sequences

- INPUT:
  - Grammar  $G' = (N \cup \{S'\}, \Sigma, P \cup \{S' \rightarrow S\}, S')$
  - SLR table
  - Input sequence  $w = a_1 \dots a_n$
- OUTPUT:

*if* ( $w \in L(G)$ )      **then string of productions**  
*else* **error & location of error**

SLR = LR(0) configurations

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

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

where:

- $\alpha$  = working stack
- $\beta$  = input stack
- $\pi$  = output (result)

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

# Moves

head( $\beta$ ) = prediction

## 1. Shift

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

$$(\$s_0x_1 \dots x_m s_m, a_i \dots a_n \$, \pi) \vdash (\$s_0x_1 \dots x_m s_m a_i s_j, a_{i+1} \dots a_n \$, \pi)$$

## 2. Reduce

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

then

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

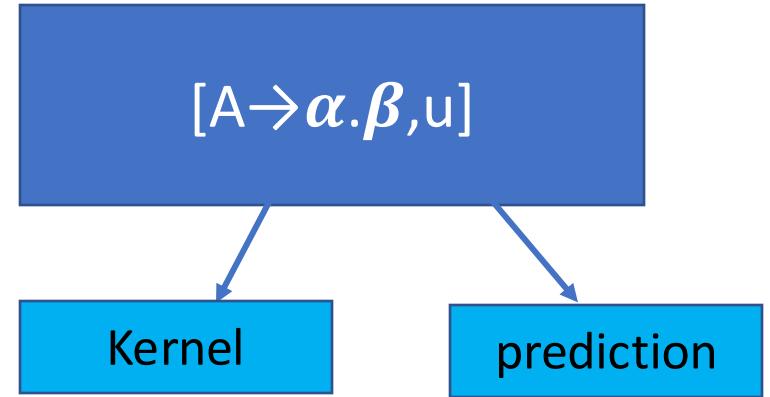
## 3. Accept

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 îmbogățită

**OUTPUT:** C - colecția canonică de stări

$\mathcal{C}_1 := \emptyset;$

$s_0 := closure(\{[S' \rightarrow .S, \$]\})$

$\mathcal{C}_1 := \mathcal{C}_1 \cup \{s_0\};$

**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

If kernels exists in a previous state

# Function *goto1*

$\text{goto1} : P(\mathcal{E}_1) \times (N \cup \Sigma) \rightarrow P(\mathcal{E}_1)$

where  $\mathcal{E}_1$  = set of LR(1) items

$\text{goto1}(s, X) = \text{Closure1}(\{[A \rightarrow \alpha \lambda. \beta, u] \mid [A \rightarrow \alpha . X \beta, u] \in s\})$

# Algorithm *Closure1*

- $[A \rightarrow \alpha.B\beta, u]$  valid for live prefix  $\gamma\alpha \Rightarrow$

$$S \xrightarrow{*_{dr}} \gamma Aw \Rightarrow_{dr} \gamma\alpha B\beta w$$

$$u = FIRST_k(w)$$

- $[B \rightarrow .\delta, smth] \in P \Rightarrow S \xrightarrow{*} \gamma Aw \Rightarrow_{dr} \gamma\alpha B\beta w \Rightarrow_{dr} \gamma\alpha\delta\beta w.$

$\Rightarrow$

# Algorithm *Closure1*

**INPUT:** I-element de analiză; G' - gramatica îmbogățită;  
 $FIRST(X), \forall X \in N \cup \Sigma;$

**OUTPUT:**  $C_1 = \text{closure}(I);$

$C_1 := \{I\};$

**repeat**

```
    for  $\forall[A \rightarrow \alpha.B\beta, a] \in C_1$  do
        for  $\forall B \rightarrow \gamma \in P$  do
            for  $\forall b \in FIRST(\beta a)$  do
                if  $[B \rightarrow .\gamma, b] \notin C_1$  then
                     $C_1 = C_1 \cup [B \rightarrow .\gamma, b]$ 
                end if
            end for
        end for
    end for
```

**until**  $C_1$  nu se mai modifică

# Construct LR(1) table

- Structure – SLR
- Rules:
  1. if  $[A \rightarrow \alpha.\beta, u] \in s_i$  and  $\text{goto}(s_i, a) = s_j$  then **action( $s_i, a$ )=shift  $s_j$**
  2. if  $[A \rightarrow \beta., u] \in s_i$  and  $A \neq S'$  then **action( $s_i, u$ )=reduce l**, where l – number of production  $A \rightarrow \beta$
  3. if  $[S' \rightarrow S., \$] \in s_i$  then **action( $s_i, \$$ )=acc**
  4. if  $\text{goto}(s_i, X) = s_j$  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  $G' = (N \cup \{S'\}, \Sigma, P \cup \{S' \rightarrow S\}, S')$
  - LR(1) table
  - Input sequence  $w = a_1 \dots a_n$
- OUTPUT:

*if* ( $w \in L(G)$ )      **then string of productions**  
*else* **error & location of error**

# LR(1) configurations

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

where:

- $\alpha$  = working stack
- $\beta$  = input stack
- $\pi$  = output (result)

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

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

# Moves

head( $\beta$ ) = prediction

## 1. Shift

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

$$(\$s_0x_1 \dots x_m s_m, a_i \dots a_n \$, \pi) \vdash (\$s_0x_1 \dots x_m s_m a_i s_j, a_{i+1} \dots a_n \$, \pi)$$

## 2. Reduce

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

then

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

## 3. Accept

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

## 4. Error - otherwise

# LALR Parser

- LALR = Look Ahead LR(1)
- why?

# LALR principle

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

$[A \rightarrow \alpha.\beta., u] \in s_i$  apply reduce (k) then  $\text{goto}(s_i, A) = s_m$   
 $[A \rightarrow \alpha.\beta., v] \in s_j$  apply reduce (k) then  $\text{goto}(s_j, A) = s_n$

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

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

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

# 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 corresponding to same kernel
  - Most used algorithm (most performant)

# Parsing - recap

	<b>Descendent</b>	<b>Ascendent</b>
Recursive	Descendent recursive parser	Ascendent recursive parser
Linear	LL(1)	LR(0), SLR, LR(1), LALR

# Parsing - recap

