

Course 9

LR(k) Parsing (cont.)

LR(k) parsing:

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

- Define item
- Construct set of states
- Construct table

Executed 1 time

-
- Parse sequence based on moves between configurations

2. Construct set of states

- What a state contains – Algorithm *closure_LR(0)*
- How to move from a state to another – Function *goto_LR(0)*
- Construct set of states – Algorithm *ColCan_LR(0)*

Canonical collection

SLR Parser

Prediction = next symbols on
input sequence

- SLR = Simple LR

- Remark:

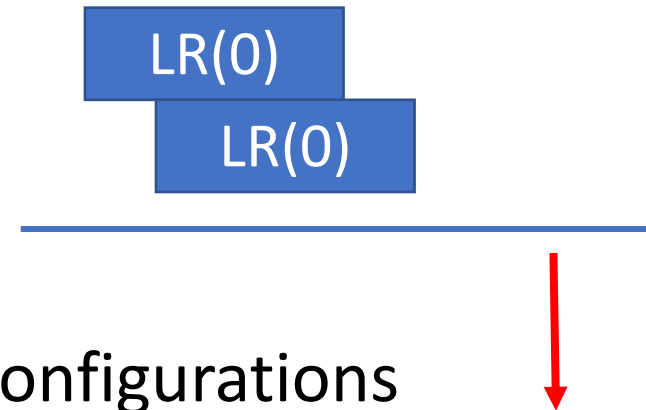
LR(0) – lots of conflicts – solved if considering prediction

=>

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

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 \Rightarrow 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 Σ

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

(α, β, π)

where:

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

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

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

Moves

$\text{head}(\beta) = \text{prediction}$

1. Shift

if $\text{action}(s_m, a_i) = \text{shift } s_j$ then

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

2. Reduce

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

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

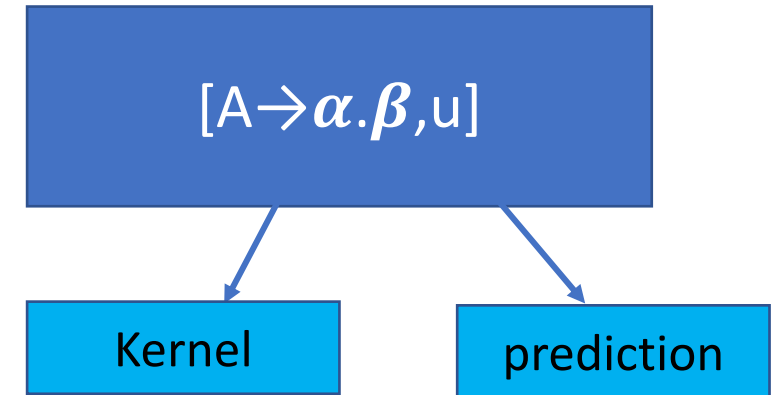
3. Accept

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

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 *goto_LR1*
- Alg *Closure_LR1*

Algorithm *ColCan_LR1*

INPUT: G' – enhanced grammar

OUTPUT: \mathcal{C}_1 – canonical collection of states

$\mathcal{C}_1 = \emptyset$

$s_0 = \text{Closure_LR1}(\{[S' \rightarrow .S, \$]\})$

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

Repeat

for $\forall s \in \mathcal{C}_1$ **do**

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

$T = \text{goto_LR1}(s, X)$

if $T \neq \emptyset$ **and** $T \notin \mathcal{C}_1$ **then**

$\mathcal{C}_1 = \mathcal{C}_1 \cup T$

endif

endfor

endfor

Until \mathcal{C}_1 *unchanged*

Function *goto_LR1*

$$Goto_LR1 : P(\mathcal{E}_1) \times (N \cup \Sigma) \rightarrow P(\mathcal{E}_1)$$

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

$$Goto_LR1(s, X) = Closure_LR1(\{[A \rightarrow \alpha X.\beta, u] \mid [A \rightarrow \alpha.X\beta, u] \in s\})$$

Algorithm *Closure_LR1*

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

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

$$u = FIRST_k(w)$$

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

\Rightarrow

Algorithm *Closure_LR1*

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 $[B \rightarrow \cdot\gamma, b] \notin C_1$ **then**

$C_1 = C_1 \cup [B \rightarrow \cdot\gamma, b]$

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:

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

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

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

Moves

$\text{head}(\beta) = \text{prediction}$

1. Shift

if $\text{action}(s_m, a_i) = \text{shift } s_j$ then

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

2. Reduce

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

4. Error - otherwise

LALR Parser

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

LALR principle

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

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

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

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

- 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

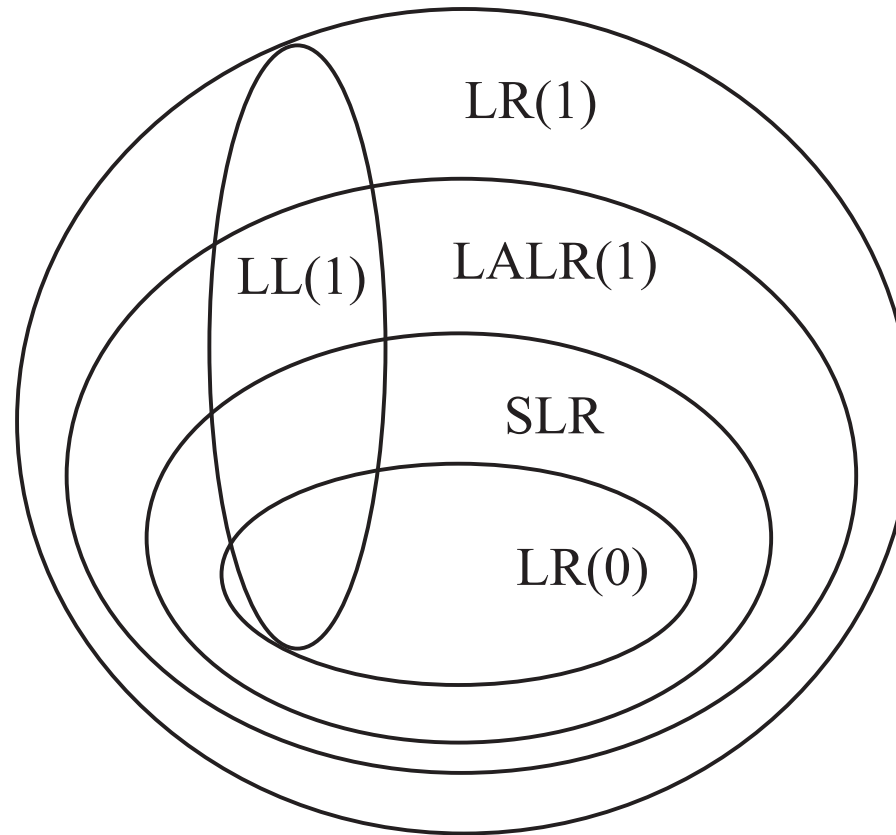
LR(k) Parsers

- 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

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

Parsing - recap



Menti 2749 942

Structure of compiler

